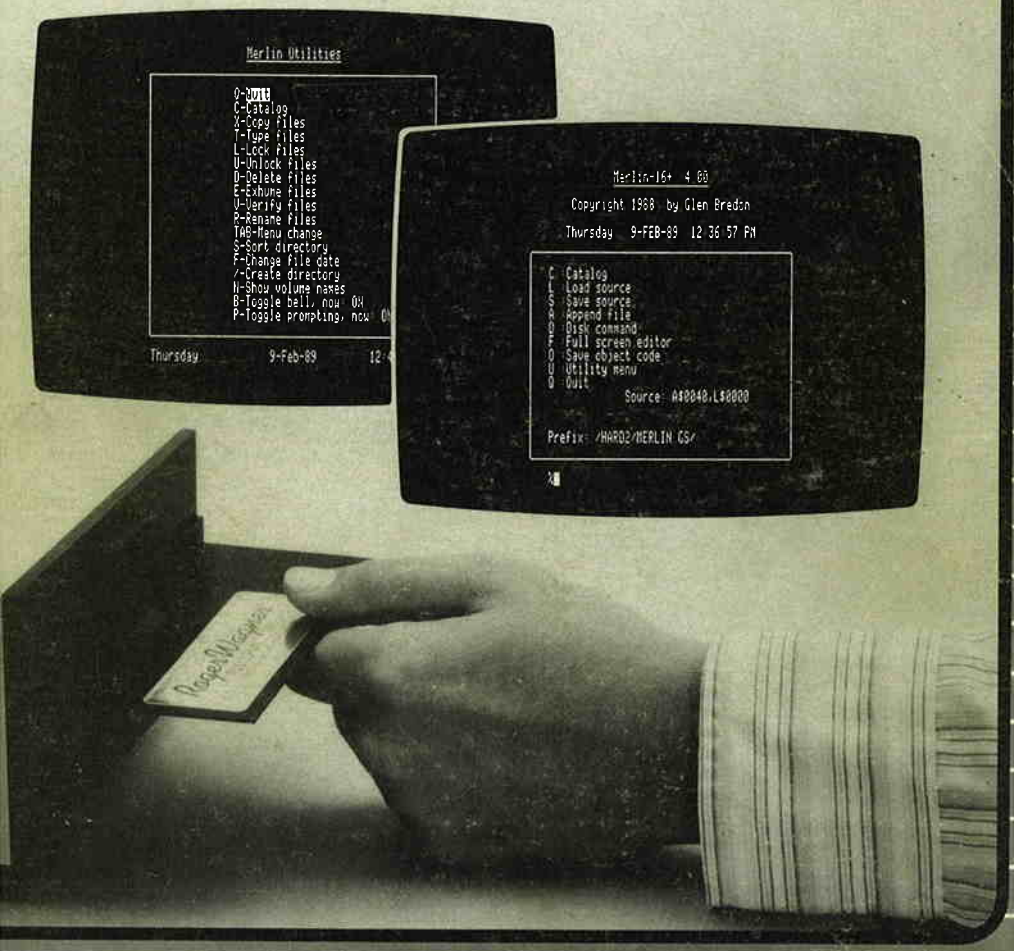
# Merlin 8/16™

## The Complete Macro Assembler System For the Apple® IIgs, 128K IIe and IIc




Roger Wagner™ PUBLISHING, INC.

# Merlin 8/16

## Complete Macro Assembler System For the Apple IIgs and 128K IIe/IIc

by Glen Bredon

Manual by Roger Wagner and Tom Burns

Produced by:

**Roger Wagner Publishing, Inc.**

1050 Pioneer Way, Suite P
El Cajon, California 92020

Customer Service & Technical Support:
619/442-0522

ISBN 0-927796-28-7
1M188

## Customer Licensing Agreement

The Roger Wagner Publishing, Inc. software product that you have just received from Roger Wagner Publishing, Inc., or one of its authorized dealers, is provided to you subject to the Terms and Conditions of the Software Customer Licensing Agreement. Should you decide that you cannot accept these Terms and Conditions, then you must return your product with all documentation and this License marked "REFUSED" within the 30 day examination period following the receipt of the product.

1. License. Roger Wagner Publishing, Inc. hereby grants you upon your receipt of this product, a nonexclusive license to use the enclosed Roger Wagner Publishing, Inc. product subject to the terms and restrictions set forth in this License Agreement.

2. Copyright. This software product, and its documentation, is copyrighted by Roger Wagner Publishing, Inc. You may not copy or otherwise reproduce the product or any part of it except as expressly permitted in this License.

3. Restrictions on Use and Transfer. The original and any backup copies of this product are intended for your personal use in connection with a single computer. You may not distribute copies of, or any part of, this product without the express written permission of Roger Wagner Publishing, Inc.

## Limitations of Warranties and Liability

Roger Wagner Publishing, Inc. and the program author shall have no liability or responsibility to purchaser or any other person or entity with respect to liability, loss or damage caused or alleged to be caused directly or indirectly by this software, including, but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of this software. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

ProDOS and DOS 3.3 are copyrighted programs of Apple Computer, Inc. licensed to Roger Wagner Publishing, Inc. to distribute for use only in combination with Merlin 8/16.

APPLE COMPUTER, INC. MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING THE ENCLOSED COMPUTER SOFTWARE PACKAGE, ITS MERCHANTABILITY OR ITS FITNESS FOR ANY PARTICULAR PURPOSE. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME STATES. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY PROVIDES YOU WITH SPECIFIC LEGAL RIGHTS. THERE MAY BE OTHER RIGHTS THAT YOU MAY HAVE WHICH VARY FROM STATE TO STATE.

**Copyrights**

The Merlin 8/16 documentation and software are copyrighted © 1987 by Roger Wagner Publishing, Inc. This documentation and/or software, or any part thereof, may not be reproduced in any form or by any means, electronic or mechanical, including photocopying, recording, storage in an information retrieval system, or otherwise, without the prior written permission of the publisher.

We have tried our best to give you a quality product at a fair price, and made the software copyable for your personal convenience. Please recommend our product to your friends, but respect our wishes to not make copies for others. Thanks!

Apple, ProDOS, and DOS 3.3 are trademarks or Apple Computer, Inc.

Merlin 8/16 is a trademark of Roger Wagner Publishing, Inc.

**About the Manual**

This manual was formatted using MacAuthor from Icon Technology, Ltd, 9 Jarrom Street, Leicester LE2 7DH, England, and Apple's LaserWriter printer.

**OUR GUARANTEE**

This product carries the unconditional guarantee of satisfaction or your money back. Any product may be returned to place of purchase for complete refund or replacement within thirty (30) days of purchase if accompanied by the sales receipt or other proof of purchase.

## ABOUT THE AUTHOR

Glen Bredon is a professor at Rutgers University in New Jersey where he has taught mathematics for over twenty years. He purchased his first computer in 1979 and began exploring its internal operations because "I wanted to know more than my students." The result of this study was Merlin, the first in a series of best selling assemblers (Merlin, Merlin Pro, Merlin 64, and Merlin 128) for the Apple and Commodore personal computers. Glen has also written other utilities including Prosel, the popular ProDOS program selector. A native Californian and concerned environmentalist, Glen spends his summers away from mathematics and computing, preferring the solitude of the Sierra Nevada mountains where he has helped establish wilderness reserves.

# MERLIN 8/16

## Table of Contents


**INTRODUCTION**

System Requirements . . . . . 1
Suggested Reading . . . . . 2
Making Back-up Copies of Merlin 8/16 . . . . . 3
Personalizing Merlin 8/16 . . . . . 4

**GETTING STARTED WITH MERLIN 8/16**

Entering a Source Listing . . . . . 5
Editing a Source Listing . . . . . 8
Assembling a Source Listing . . . . . 9
Saving Programs . . . . . 12
Running Programs . . . . . 14
The Rest of This Manual . . . . . 16

**THE MAIN MENU**

The Main Menu . . . . . 17

**THE MERLIN 8 EDITOR**

Merlin 8 Full Screen Editor Commands . . . . . 23
Merlin 8 Control Key Commands . . . . . 24
Merlin 8 Open Apple Key Commands . . . . . 27
Merlin 8 Editor Command Summary . . . . . 33
Merlin 8 Command Mode . . . . . 36
General Guidelines for the Command Mode . . . . . 36
About the Merlin 8 Editor Documentation . . . . . 36
Command Mode Commands . . . . . 37
Assembling a Merlin 8 File . . . . . 47

**THE MERLIN 16 EDITOR**

Merlin 16 Full Screen Editor Commands . . . . . 48
Merlin 16 Control Key Commands . . . . . 48
Merlin 16 Open Apple Key Commands . . . . . 52
Merlin 16 Command Box . . . . . 61
General Guidelines for the Command Box Commands . . . . . 61
About the Command Box Documentation . . . . . 61
Command Box Commands . . . . . 63
Merlin 16 Editor Command Summary . . . . . 70
Assembling a Merlin 16 File . . . . . 73

## THE ASSEMBLER


The Assembler . . . . . 74
About the Assembler Documentation . . . . . 75
Preliminary Definitions . . . . . 76
Assembler Syntax Conventions . . . . . 77
Source Code Format . . . . . 77
Local Labels . . . . . 78
Variables . . . . . 79
Number Format . . . . . 80
Expressions Allowed by the Assembler . . . . . 81
Immediate Data Syntax . . . . . 83
Addressing Modes . . . . . 83
Special Forced Non-Zero Page Addressing . . . . . 85
Sweet 16 Opcodes . . . . . 85
65C02 and 65802 Opcodes . . . . . 86
Assembler Pseudo Opcode Descriptions . . . . . 87
Formatting Pseudo Ops . . . . . 103
String Data Pseudo Ops . . . . . 107
Data and Storage Allocation Pseudo Ops . . . . . 110
Using Data Tables in Programs . . . . . 113
Conditional Pseudo Ops . . . . . 114
Using Conditional Assembly . . . . . 116
Miscellaneous Pseudo Ops . . . . . 119


## MACROS


Why Macros . . . . . 127
Macro Pseudo Ops . . . . . 127
How a Macro Works . . . . . 128
More About Defining a Macro . . . . . 133
Nested Macros . . . . . 134
Macro Libraries and the USE Pseudo-Op . . . . . 136


## THE LINKERS


The Merlin 8/16 Linkers . . . . . 137
Why a Linker . . . . . 137
About the Linker Documentation . . . . . 138
Pseudo Opcodes for Use with Relocatable Code Files . . . . . 140
The Merlin 8 Linker . . . . . 144
Linker Name Files . . . . . 145
The Linking Process . . . . . 146
Using Merlin 8 Linked Files . . . . . 147
The Merlin 16 Linkers . . . . . 150
The Absolute Linker . . . . . 151
The Linker Command File . . . . . 154
Using Merlin 16 Linked Files . . . . . 155

Multiple Output Files . . . . . 157
The GS Linker . . . . . 158
Quick Link . . . . . 159
Multiple LNK Input Files . . . . . 160
Using the GS Linker . . . . . 161
Large File GS Linker . . . . . 163

TECHNICAL INFORMATION
Technical Information . . . . . 164
General Information (DOS 3.3 only) . . . . . 164
General Information (ProDOS and DOS 3.3) . . . . . 165
Symbol Table . . . . . 165
Ultraterm Information . . . . . 165
Memory Allocation with Merlin 8/16 . . . . . 166
Configuring Merlin 8 ProDOS . . . . . 166
Configuring Merlin 8 DOS 3.3 . . . . . 166
Data Descriptions for Merlin 8 Configurations . . . . . 166
64K Merlin and Merlin 8/16 Source Files . . . . . 167
Merlin 8/16 ProDOS Notes . . . . . 168
Transferring Source Files from DOS 3.3 to ProDOS Merlin 8/16 . . . . . 169
Merlin 8/16 and Speed Up Cards . . . . . 169
Merlin 8 and Corvus Hard Disks . . . . . 170
Configuring Merlin 16 . . . . . 170
The Merlin 16 Parms File . . . . . 170
Merlin 8/16 Memory Maps . . . . . 176

ERROR MESSAGES
Error Messages . . . . . 179

SOURCEROR
Using Sourceror on Merlin 16 . . . . . 184
Using Sourceror on Merlin 8 . . . . . 184
65C02 Opcodes on Older IIe/IIc Computers . . . . . 186
Disassembly Commands . . . . . 186
Merlin 16 Sourceror Notes . . . . . 188
Sourceror XL . . . . . 188
Final Processing . . . . . 190
Modifying the Finished Source . . . . . 190
The Memory Full Message . . . . . 191
Changing Sourceror's Label Tables . . . . . 191

SOURCEROR.FP
Applesoft Source Listing . . . . . 192
Sourceror.FP . . . . . 192
Printing the Applesoft Source Listing . . . . . 193

Applesoft Source Cross Reference Listing . . . . . 195

**UTILITY PROGRAMS**

Auto Edit . . . . . 197
Classic Desk Accessories . . . . . 197
Calendar - Notepad - Rational Calculator . . . . . 197
Clock . . . . . 197
Conv Lnk Rel . . . . . 198
Converter: APW source to Merlin . . . . . 198
Using Macgen . . . . . 200
Cross Reference Programs . . . . . 205
Xref/XrefA . . . . . 205
Using Xref . . . . . 206
XrefA Notes . . . . . 208
Formatter . . . . . 208
Keyboard Macro Files . . . . . 209
Edmac - Keymac . . . . . 209
Keyboard Macro Equivalent Chart . . . . . 210
Make Dump . . . . . 211
Printfiler . . . . . 211
Printfiler Applications . . . . . 212
Using Printfiler . . . . . 212
Changing Printfiler Options . . . . . 213
Txted . . . . . 214
Type.Changer . . . . . 214
Additional Merlin 8/16 Resource Files . . . . . 215

**INDEX**

Index . . . . . 216

# MERLIN 8/16

Merlin 8/16 is an extremely powerful, comprehensive Macro Assembler system for the Apple IIgs or 128K IIe/IIc. It consists of four main modules and numerous auxiliary and utility programs which comprise one of the most complete assembler systems available for *any* personal computer. Merlin 8/16's four main modules are:

- FILE MANAGEMENT system, for disk I/O, file management, ProDOS interpreter, etc.

- EDITOR system, for writing and editing programs with word-processor-like power.

- ASSEMBLER system, with Macros, Macro libraries, conditional assembly, linked files, etc.

- LINKER system, for generating relocatable code modules, library routines, run-time packages, etc.

However, Merlin 8/16 is more than just the sum of these four parts. Here are some of the other features offered by Merlin 8/16:

- Merlin 8/16 comes with *three* assemblers: Merlin 16 (ProDOS), Merlin 8 (ProDOS), and Merlin 8 (DOS 3.3). On the Apple IIgs, or Apple IIe or IIc computers with the 65802 or 65816 chip, Merlin 16 assembles programs written for the 6502, 65C02, 65802 and 65816 microprocessors. On the standard 128K IIe or IIc, Merlin 8 assembles programs written for the 6502, 65C02 and 65802 microprocessors.

- Merlin 8/16 is compatible with existing Merlin and Merlin Pro source files.

- Merlin 8/16 recognizes over 50 Pseudo Opcodes for extreme programming flexibility.

- Merlin 8/16 has over 70 commands for ultimate editing and assembling power.

- Merlin 8/16 produces a fully commented, disassembled source listing of Applesoft BASIC.

- Merlin 8/16 comes with Sourceror, a powerful symbolic disassembler to generate Merlin 8/16 source code from binary programs.

- Merlin 8/16 comes with many sample programs, libraries and other aids to get you going with assembly language fast.

- Merlin 8/16 is copyable and hard disk compatible.

# INTRODUCTION

Merlin 8/16 is the most comprehensive macro assembler system for the Apple IIgs or 128K IIe/IIc offering virtually every feature and function that a programmer needs, thus making it unlikely that you'll outgrow it. At the same time, Merlin 8/16's easy built-in editor and fast assemblies make it a pleasure to use whether you're writing a few lines of code or 30,000!

## SYSTEM REQUIREMENTS

To run Merlin 8/16, you'll need at least one disk drive, and one of the following:

* Apple IIgs or
* Apple IIc or
* 128K Apple IIe or
* Laser 128 or Laser 128 EX

Merlin 8 will run on all the computer systems listed above; Merlin 16 requires a 65802 or 65816 microprocessor such as in the Apple IIgs, or a modified Apple IIe, IIc or compatible.

Merlin 8/16 supports a wide variety of 80 column display devices including the Apple, the Videx standard and Ultraterm cards, the Checkmate Multi-View, the Applied Engineering Viewmaster 80, and many others.

Merlin 8/16 works with all printers, producing formatted listings with page breaks and titles.

If you're familiar with assembly language programming already, you will find it's easy to adapt to Merlin 8/16. It follows the programming standards of the 65xx family of microprocessors, and its assembler-directed commands, or pseudo-ops, are a super-set of just about every other assembler. That is, assembler directives like HEX, ASC, DS, etc. that you've used in other assemblers are used in Merlin 8/16, and better still, you'll find a new complement of functions to make programming easier. These include assembling directly to or from disk files, multiple data formats for numbers and strings, a complete set of assembler utilities such as cross-referencing and a source code generator (Sourceror), macro capabilities and more.

If you're new to assembly language programming, Merlin 8/16 is the easiest assembler there is. However, the Merlin 8/16 manual does not make any attempt to teach the techniques of assembly language programming itself. Those techniques are covered in various tutorial books available from a number of publishers, including Roger Wagner Publishing. Because everyone has different goals and objectives, you should seek out those books which best match your current needs and experience.

## SUGGESTED READING

Some of the books we recommend include:

APPLE IIGS MACHINE LANGUAGE FOR BEGINNERS - Roger Wagner, COMPUTE! Books, Greensboro, NC 27408. Approximately 600 pages of new material, this IIgs assembly language tutorial covers a myriad of subjects such as writing your first routines, calling machine language routines from Applesoft, ProDOS 8 and 16, and concludes with a IIgs drawing program using Quickdraw, scrollable windows, and pull-down menus. The encyclopedic appendix includes examples of all machine language instructions.

APPLE IIGS TECHNICAL REFERENCE - Michael Fischer, Osbourne-McGraw-Hill. Filled with relevant technical material, this book is an outstanding resource for the advanced programmer interested software development on the IIgs.

APPLE IIGS TOOLBOX REFERENCE - Apple Computer, Inc. This 2-volume set is a highly technical reference for using the IIgs toolbox. Does not include programming examples or tutorial information.

APPLE PRODOS - by Gary Little. Brady Communications Co., Bowie, MD 20715. A good tutorial book on how to write assembly language programs that use ProDOS.

ASSEMBLY LINES: THE BOOK - by Roger Wagner. Roger Wagner Publishing, Inc. El Cajon, CA 92020. A tutorial on assembly language programming designed specifically for the novice. It gives a description of all 6502 instructions, disk access, reading and writing DOS 3.3 files, sound generation, basic math, keyboard and screen techniques and more.

ASSEMBLY COOKBOOK FOR THE APPLE II/IIE - by Don Lancaster. Howard Sams & Co., Indianapolis, IN 46268. This interesting book will give you real insights into Don Lancaster's view of programming theory and practice. A good addition to any library.

BENEATH APPLE DOS - by Don Worth and Peter Lechner. Brady Communications Co., Bowie, MD 20715.

BENEATH APPLE PRODOS - by Don Worth and Peter Lechner. Brady Communications Co., Bowie, MD 20715. These two books are the classic reference books for learning about the inner workings of DOS 3.3 and ProDOS. A must if you intend to do any programming in this area. More of a reference than a tutorial.

ENHANCING YOUR APPLE II (Volume 1) - by Don Lancaster. Howard Sams & Co., Indianapolis, IN 46268.

ENHANCING YOUR APPLE II AND IIE (Volume 2) - by Don Lancaster. Howard Sams & Co., Indianapolis, IN 46268. These two books are a continuation of Don Lancaster's unique instruction in the art of assembly language programming. More tips here on different kinds of short programs.

INSIDE THE APPLE IIC - by Gary Little. Brady Communications Co., Bowie, MD 20715. An overview of the entire Apple IIc system, from the assembly language programmer's viewpoint.

INSIDE THE APPLE IIGS - Gary Bond, Sybex. Contains a lot of important information but by-passes some of the rules for producing code compatible with future system upgrades.

NOW THAT YOU KNOW APPLE ASSEMBLY LANGUAGE, WHAT CAN YOU DO WITH IT? - by Jules Guilder, Redlig Systems, Inc., 2068 79th Street, Brooklyn, NY 11214. A good applications tutorial with lots of good subroutines for input and output, printer drivers and more. An excellent follow-up book to Assembly Lines: The Book.

PROGRAMMING THE 65816 - David Eyes and Ron Lichty, Prentice Hall Press, New York, NY 10023. In-depth coverage of the differences between the 65816, 6502, 65C02, and 65802 chips and how to best utilize them from a programming standpoint. Also includes a programming tutorial, code samples and a reference section.

THE APPLE IIGS TOOLBOX REVEALED - Danny Goodman, Bantam Computer Books. Details the philosophy of the toolbox and methods of accessing it, but lacks programming examples.

THE ELEMENTARY APPLE IIGS - William B. Sanders, COMPUTE! Books, Greensboro, NC 27408. An introductory book to the IIgs with information on Applesoft, Hi-Res and Super Hi-Res graphics, and sound on the IIgs.

65816/65802 ASSEMBLY LANGUAGE PROGRAMMING - by Michael Fischer. Osborne McGraw-Hill , Berkely, CA 94710. A thorough treatment of the 6502, 65C02, 65802 and 65816 microprocessors. The size of this book (nearly 700 pages) gives you an idea of why this manual for Merlin 8/16 does not attempt to cover programming on the 6500 family microprocessors itself.

## MAKING BACK-UP COPIES OF MERLIN 8/16

The Merlin 8/16 diskettes are unprotected and copies may be made using any copy utility. It is highly recommended that you use *only* the BACK-UP copy of Merlin 8/16 in your daily work, and keep the original in a safe place.

You can copy the Merlin 8/16 diskettes using:

1) The COPYA utility program from Apple's System Master Diskette for the DOS 3.3 version of Merlin 8/16.

2) The Copy Volume or Duplicate a Diskette function from the ProDOS System Disk for the ProDOS versions of Merlin 8/16, or System Utilities program on the Apple IIGS System Master diskette.

## PERSONALIZING MERLIN 8/16

Certain aspects of Merlin 8/16, such as printer line width, page length, default tab positions for the fields in a source listing, turning off the bell sound, etc., can be customized by changing the file PARMS.S on the Merlin 8/16 disk, and re-assembling the PARMS file. If you would like to change any of these defaults of Merlin 8/16, see the discussion of the PARMS file in the Technical Information section of this manual for details on making the changes. For now, though, we recommend you wait until you're more familiar with Merlin 8/16.

# GETTING STARTED WITH MERLIN 8/16

The purpose of this section is not to provide instruction in assembly language programming. Rather, it will show you the entry and running of a short assembly language program to give you an idea of how Merlin 8/16 works.

Many of the Merlin 8/16 commands and functions are very similar in operation. This section does not attempt to present demonstrations of each and every command option. The objective is to present examples of the more common operations, sufficient to get you started writing your own programs using Merlin 8/16. You should not expect to immediately use all of the various commands that Merlin 8/16 supports in your first program. The best approach is to use the Merlin 8/16 manual in an encyclopedia-like fashion, reading just those sections that provide some utility to a current programming task. We suggest that you lightly skim through the manual once, to become aware of generally what the software has to offer, and then return later to specific sections as needed.

## ENTERING A SOURCE LISTING

Now, let's try your first program with Merlin 8/16. Just follow these steps:

1. Start any of the Merlin 8/16 disks (ProDOS or DOS 3.3). A title screen appears, after which the screen changes to the Main Menu. The Main Menu is used for loading and saving files, disk operations, and of course, entering the Merlin 8/16 Editor and Assembler itself.

2. The percent (%) prompt appears at the bottom of the Main Menu. Press F if you are using Merlin 16. If you are using Merlin 8, press E to go the Editor, then A and Return to enter the Add Mode.

3. Since we are entering an entirely new program, a 1 appears at the top right corner of the screen. This number indicates the line the cursor is on in the listing. The 1 and all subsequent line numbers which appear serve roughly the same purpose as line numbers in BASIC except that in assembly source code, line numbers are not referenced for jumps to subroutines or in GOTO-like statements.

4. On line 1, type an asterisk (*). Entering an asterisk as the first character in any line is similar to a REM statement in BASIC - it tells the assembler this is a remark line and anything after the asterisk is to be ignored. Type the title DEMO PROGRAM 1 after the asterisk and press the Return key.

```
* DEMO PROGRAM 1
```

5. After Return, the cursor moves to the beginning of line 2.

6. An asterisk is not needed to just create a blank line. To create a blank line, with no text following it, press Return again.

7. The cursor once again drops down one line, and a 3 appears at the top right corner of the screen.

8. Press the space bar once and the cursor moves to the next field. Type ORG, press space again, type $8000 and then press Return.

```
* DEMO PROGRAM 1

        ORG   $8000
```

The above step instructs the assembler to create the following program so that it can run at memory location $8000. Merlin 8/16 almost always assembles your program in the same place in memory, but the ORG (for Origin) is used to tell Merlin 8/16 where you want the program to eventually be run. This is so that JMPs, JSRs and other location dependent code within your program is properly written with the final location in mind.

You'll notice that when you press the space bar, Merlin 8/16 automatically moves the cursor to the next field on the line. You'll recall that in assembly language programming, the position of text on each line determines what kind of information it is. Labels for routines and entry points are in the first position. On line 3 you skipped this field by pressing the space bar first, before entering any text. The second position is for the command itself. The command can either be a command such as LDA, RTS, etc., or it can be a directive to Merlin 8/16 itself, to be used during the assembly to write a file to disk, create a label, call up a macro, or any of Merlin 8/16's many assembler commands.

9. With the cursor at the beginning of line 4, type BELL and space to the next field, type EQU and space again, then type $FBDD and press Return.

```
BELL    EQU   $FBDD
```

This defines the label BELL to be equal to hex FBDD. This use of a label is known as an equate or constant. Wherever BELL appears in an expression, it will be replaced with $FBDD. Why don't we just use $FBDD? For one thing, BELL is easier to remember than $FBDD. Also, if a later assembly required changing the location of BELL, all that needs changing is the EQU statement, rather than all the other $FBDD's throughout the listing.

10. At the beginning of line 5, type START and press space, type JSR and space again, type BELL and another space, then type a semicolon (;) followed by RING THE BELL and then press Return.

```
START   JSR   BELL          ; RING THE BELL
```

Following the opcode is the operand, in this case BELL. The operand is the target information of the opcode. Where to JSR to, what value to load, etc.

Semicolons are like asterisks, used to mark a comment. Semicolons, however, are used to mark the start of a comment at the end of a line that contains other commands.

11. On line 6 type DONE and space, then type RTS and press Return.

12. The program has been completely entered. If you wanted to exit the Add Mode, you would press Open-Apple-Q ("Quit"). Since you are not done, *do not exit yet.*

13. The screen should now appear like this:

```
* DEMO PROGRAM 1

       ORG   $8000
BELL   EQU   $FBDD
START  JSR   BELL          ; RING THE BELL
DONE   RTS
```

Note that throughout the entry of this program, each bit of text has been moved to a specific field. Here is a summary of the fields as used so far:

```
Label  Opcode  Operand       ; Comment
START  JSR     BELL          ; RING THE BELL
```

Field One is reserved for labels. START is an example of a label.

Field Two is reserved for opcodes, such as the Merlin 8/16 pseudo-opcodes or directives such as ORG and EQU, and the JSR and RTS opcodes.

Field Three is for operands, such as $8000, $FBDD and, in this case, BELL.

Field Four contains comments which are preceded by a semi-colon (;).

It should be apparent from this exercise that it is not necessary to input extra spaces in the source file for formatting purposes, even if these spaces seem to exist in a listing you may be using.

In summary, on each line:

1) Do not space for a label. Space once after a label or if there is no label, once at the beginning for the opcode.

2) Space once after the opcode for the operand. Space once after the operand for the comment. If there is no operand, type a space and a semicolon for a comment if desired.

## EDITING A SOURCE LISTING

Assuming no errors have been made in the text entered so far, you could now assemble the program entered with Merlin 8/16. Before doing that, however, let's look at the editing abilities of Merlin 8/16.

Editing is the process of making alterations to text that you've already entered, and this ability is one of Merlin 8/16's strong points. In a sense, an assembler is just a word processor for the text that makes up a program. In that light, then, you can judge an assembler in part by how good its editing features are.

Merlin 8/16 has a powerful Full Screen Editor. Powerful in the range of operations possible and, after a little practice, remarkably easy to use. The following text describes how to use the Editor.

You can use the arrow keys to move the cursor anywhere in the listing. There are two types of cursors, the insert and the overstrike. How and when to switch between these two cursor will be explained later. For now, we'll use the insert cursor.

Inserting and deleting lines in the source code are both simple operations. The following example will insert three new lines between the existing lines 5 and 6.

1. Use the arrow key to move to the beginning of line 4 (BELL etc.). Press Return to insert a blank line.

2. Press Return.

3. Press Return again.

4. At line 7, press space once, then type TYA and press Return.

5. The listing should appear as follows:

```
* DEMO PROGRAM 1

       ORG   $8000
BELL   EQU   $FBDD


       TYA
START  JSR   BELL      ; RING  THE BELL
DONE   RTS
```

The three new lines (5, 6, and 7) have been inserted, and the subsequent original source lines (now lines 8 and 9) have been moved down.

Deleting lines is equally easy.

1. In Merlin 16, press a key to re-enter the Full Screen Editor.

   In Merlin 8, press A and Return to enter the Add Mode (assuming you are at the command mode).

2. Move the cursor to the beginning of line 8 (START etc.) and press Open-Apple-Delete.

   You've just deleted the TYA line, and the subsequent lines have been renumbered. Open-Apple-Delete always deletes the line *above* the cursor. (Open-Apple-D will delete the line the cursor is on.)

3. Press Open-Apple-Delete twice more.

Lines 5 and 6 from the previous example have been deleted, and the subsequent lines renumbered. The listing appears the same as when you first entered it into to the Editor.

While adding, editing, or deleting an existing line, you have many options within the line, all of which are accessed by using Control characters. To demonstrate using our BELL routine:

1. Move the cursor to the beginning of line 6. The cursor should be over the D in DONE.

2. Type Control-D. The character under the cursor disappears, and the text of the label moves to the left. Type Control-D again, and yet a third and fourth time. DONE has been deleted, and the cursor is positioned at the beginning of the label field.

3. Type DONE in again and note that the text is inserted in the label field but the opcode field does not move. The listing appears the same as when you first entered it into the Editor.

The other Control Character commands function similarly. All of the editing commands are discussed in the Merlin 16 and Merlin 8 Editor sections of this manual.

## ASSEMBLING A SOURCE LISTING

The next step in using Merlin 8/16 is to assemble the source code into object code.

1. If you are using Merlin 16, press Open-Apple-A to assemble and skip to step 3.

   If you are using Merlin 8, press Open-Apple-Q to enter the Command Mode. After the colon (:) prompt, type ASM and press Return.

**2a. The Merlin 16 screen appears as follows:**

```
Assembling.
                      1     * DEMO PROGRAM 1
                      2
                      3               ORG  $8000
           =FBDD      4     BELL      EQU  $FBDD
008000 20 DD FB       5     START     JSR  BELL     ; RING THE BELL
008003 60             6     DONE      RTS

End Merlin-16 assembly, 4 bytes, errors: 0 , symbol table: $1800-$181D

Symbol table - alphabetical order:

   BELL    =$FBDD   ?  DONE    =$8003      ?  START    =$8000

Symbol table - numerical order:

? START    =$8000   ?  DONE    =$8003         BELL     =$FBDD

Press a key.
```

**2b. The Merlin 8 screen appears as follows:**

```
Assembling
                      1     * DEMO PROGRAM 1
                      2
                      3               ORG  $8000
                      4     BELL      EQU  $FBDD
8000 20 DD FB         5     START     JSR  BELL     ; RING THE BELL
8003 60               6     DONE      RTS

--End assembly, 4 bytes, Errors: 0

Symbol table - alphabetical order:

   BELL    =$FBDD   ?  DONE    =$8003      ?  START    =$8000

Symbol table - numerical order:

? START    =$8000   ?  DONE    =$8003         BELL     =$FBDD
:
```

If instead of completing the above listing, the system beeps and displays an error message, note the line number referenced in the message, and press Return until the "End assembly..." message appears. Then refer back to the section where the program was first entered and compare the listing with the one shown in this manual. Look especially for elements in incorrect fields. Using the editing functions you've learned, change any lines in your listing which do not look like those in the listing, then assemble again.

If all went well, to the right of the column of line numbers down the middle of the screen is the now familiar, formatted *source code.*

To the left of the line numbers is a series of numeric and alphabetic characters. This is the *object code*, which are the opcodes and operands assembled to their machine language hexadecimal equivalents.

Merlin 16 example:

```
008000 20 DD FB       5    START     JSR  BELL    ;ring the bell
```

Merlin 8 example:

```
  8000 20 DD FB       5    START     JSR  BELL    ;ring the bell
```

Left to right, starting on line 5, the first group of characters is the routine's starting address in memory. See the definition of ORG in the section on the Assembler. On the left of line 5, the number 20 appears after the colon. This is the one-byte hexadecimal code for the opcode JSR.

**NOTE:** The label START is not assembled into object code; neither are comments, remarks, or pseudo-ops such as ORG. Such elements are for the convenience and utility of the programmer only and the use of the assembler program.

Each pair of hexadecimal digits is one byte. The next two bytes on line 5 bear a curious resemblance to the last group of characters on line 4; have a look. In line 4 of the source code we told the assembler that the label BELL was EQUated to address $FBDD. In line 5, when the assembler encountered BELL as the operand, it substituted the specified address. The sequence of the high- and low-order bytes was reversed, turning $FBDD into DD FB. This is a 65xx microprocessor convention.

The rest of the information presented should explain itself. The total errors encountered in the source code was zero. If you count the bytes following the addresses, you'll see there were four bytes of object code generated.

## SAVING PROGRAMS

There is an important step before running a program you have assembled. You should always save the source code first in case your program crashes when you run it and causes your computer to hang. If you save the program first, you'll be able to load it again and continue editing it. To save the source code, you will have to return to the Main Menu, and use the SAVE SOURCE command. Then you would use the SAVE OBJECT CODE. Note that SAVE OBJECT CODE can only be used if there has been a successful assembly.

Here are the steps to follow for **Merlin 16**:

1. After a successful assembly, Merlin 16 returns automatically to the Main Menu. (You can also return to the Main Menu in Merlin 16 by pressing Open-Apple-Q) If the Merlin 16 system disk is still in the drive, remove it and insert an initialized data disk.

   Press D for DISK COMMAND, then type PREFIX followed by the pathname for your data disk. For example, if the volume name of your data disk was MYDISK, the complete line would look like this:

   Disk command:PREFIX/MYDISK

   When you press Return, Merlin 16 will change the current prefix in the Main Menu box.

2. After the per cent (%) prompt, press S to SAVE SOURCE. The system is now waiting for a filename. Type DEMO1 and press Return. After the program has been saved, the prompt returns.

3. Press C and Return to catalog the data diskette. The source code has been saved as DEMO1.S and is a text file. The .S suffix is automatically appended by Merlin 16 to the filename by the SAVE SOURCE command. This is a file-labeling convention which indicates the subject file is source code.

4. After the per cent (%) prompt, press Return to go to the Main Menu and press O for SAVE OBJECT CODE. To avoid confusion, the object file should be saved under the same name as was earlier specified for the source file. Press Y to accept DEMO1 as the object name. The object code file is saved as a BIN file.

NOTE: There is no danger of overwriting the source file because no suffix is appended to the object code file name. In our example, the object file will be saved as DEMO1.

5. Press C to catalog again and note the files titled DEMO1.S and DEMO1.

Here are the steps to follow for **Merlin 8**:

1. From the Command Mode prompt (:), press Q to Quit and Return. The system will quit the Editor and go to Main Menu. If the Merlin 8 system disk is still in the drive, remove it and insert an initialized data disk. If you are using the DOS 3.3 version of Merlin 8, skip to step 2.

   On the ProDOS version of Merlin 8, press D for DISK COMMAND, then type PREFIX followed by the pathname for the data disk. For example, if the volume name of your data disk was MYDISK, the complete line would look like this:

   Disk command:PREFIX/MYDISK

   When you press Return, Merlin 8 will change the current prefix on the screen.

2. After the per cent (%) prompt, press S to SAVE SOURCE. The system is now waiting for a filename. Type DEMO1 and press Return. After the program has been saved, the prompt returns.

3. Press C to catalog the diskette. The source code has been saved as DEMO1.S and is a text file on the ProDOS version, or a binary file on the DOS 3.3 version of Merlin 8. The .S suffix is automatically appended by Merlin 8 to the filename by the SAVE SOURCE command. This is a file-labeling convention which indicates the subject file is source code.

4. Press Return to go to the Main Menu and type O for SAVE OBJECT CODE. To avoid confusion, the object file should be saved under the same name as was earlier specified for the source file. Press Y to accept DEMO1 as the object name.

NOTE: There is no danger of overwriting the source file because no suffix is appended to the object code file name. In our example, the object file will be saved as DEMO1.

5. Press C to catalog again and note the files titled DEMO1.S and DEMO1.

When looking at the Main Menu, you'll also notice that the address and length of your source code text is displayed. If you have done a successful assembly, the address and length of the assembled object code is also displayed. If the object code information is not displayed, Merlin 8/16 will not let you save an object file to disk. The object code save is disabled whenever an error has occurred during an assembly, or you have made a change to the source code and not yet re-assembled it, or the source code is either too big to fit or not allowed to reside in the memory range you have specified for the assembly.

See the sections on ORG, OBJ and Memory allocation if this latter problem occurs.

# RUNNING PROGRAMS

There are two methods of running programs after you have completed a successful assembly. *Method A is the recommended procedure.* It requires a little more time but it is much safer.

### Method A: Running a program from BASIC

1. Save the source code.
2. Save the object code.
3. Press Q to Quit.
4. Load BASIC if necessary.
5. BLOAD the object file from the Applesoft BASIC prompt (]).

   If you are using ProDOS, be sure to include the complete pathname. In our example, this would be:

   BLOAD/MYDISK/DEMO1 (Return)

   If you are using DOS 3.3, you would type:

   BLOAD DEMO1 (Return)

6. To run the program, type CALL followed by the appropriate load address. The DEMO1 program had an ORG of $8000 which is 32768 in decimal. Thus, from the Applesoft prompt you would type:

   CALL 32768 (Return)

   If you are running DEMO1, you will hear the beep indicating DEMO1 actually works.

### Method B: Running a program from the Monitor

1. If you are using Merlin 16, return to the Editor if necessary by pressing F.

   If you are using Merlin 8, return to the Editor if necessary by pressing E.

2. From the Merlin 16 Editor, press Open-Apple-O to open the Command Box.

   From the Merlin 8 Editor, press Open-Apple-Q to enter the Command Mode (:).

3. Type GET $8000 and press Return where $8000 is the address used in the ORG statement. The GET command tells Merlin 8/16 to take the program you've just assembled and transfer it to Main Memory at the specified location.

4. From the Merlin 16 Editor, press Open-Apple-O to open the Command Box, and type MON and press Return and the Monitor prompt (*) appears.

   In Merlin 8, type MON and press Return and the Monitor prompt (*) appears.

5. Type the ORG ad ress followed by a G and press Return to run the program. In our example, you would type 8000G and press Return. A beep is heard. The demonstration program DEMO1 was responsible for it. It works!

NOTE: With Method B, do not forget to use the GET command to move the code to Main Memory. If you assemble your program and then go directly to the Monitor *you will not see your program.* It only gets there after you move it to Main Memory or run it. Also, if your program is loaded in one location but runs in another, you must run the program before you can use the Monitor to examine the code at the final location.

6. You can return to the Main Menu from the Monitor by pressing Control-Y and Return.

**APPLE IIGS USERS:** Method B *cannot* be used to test programs that call Applesoft or Monitor routines in the address range $D000-$FFFF (such as DEMO PROGRAM #1 that calls the BELL routine at $FBDD). This is because on the GS, these routines are actually located in bank $FF of memory, while the routine you're testing, at that point, is in bank $00. When your routine calls the Applesoft or Monitor routine, it will not jump to the proper bank, and the results are unpredictable. You can use Method A to test this type of program, or use whatever instructions may be provided for testing the program, as will likely be the case with listings from magazines or other sources.

## THE REST OF THIS MANUAL...

The preceding section was a simple look at how to enter, assemble, save, and run a Merlin 8/16 program.

The remainder of this manual is an encyclopedic reference of the various commands that are available within Merlin 8/16 to make writing an assembly language program easier. Remember that the commands and directives available within Merlin 8/16 are merely the building blocks from which you can create your own programs. It is up to you to decide when and where they are to be used.

The manual describes the following aspects of Merlin 8/16:

1) **The Main Menu:** This level of Merlin 8/16 is used for loading and saving files, disk operations, and entering the Editor/Assembler.

2) **The Merlin 16 Editor and The Merlin 8 Editor:** These sections describes the functions available for creating and editing a source listing for an assembly language program.

3) **The Assembler:** This section covers assembler directives within Merlin 8/16. Remember that these are not editing or direct user commands, but rather, text commands included within a source listing to tell the assembler to do something special while your program is being assembled. This might include using a Macro definition, writing a file to disk, or other functions.

4) **Supplemental Sections:** There are a number of additional sections in this manual that describe the use of Macros, the Relocating Linker, Error Messages, Sourceror and Utility Programs, and many other aspects of Merlin 8/16's operation. These can be consulted as necessary.

# THE MAIN MENU

The Merlin 8/16 Main Menu is used for file maintenance operations such as loading or saving code or cataloging the disk. The following sections summarize each command available in this mode.

## C (Catalog - ProDOS)

When you press C, you will be asked for the pathname of the directory you wish to catalog. At the Prefix: prompt, enter a pathname or press Return. The catalog of the current directory will be shown. The Main Menu prompt (%) is displayed after the catalog is shown. You can then issue any Main Menu command such as L for Load Source. This permits you to give a Main Menu command while the catalog is still on the screen. In addition, if *any* key is typed during the catalog printing, the ProDOS catalog will pause until any other key is pressed.

After using the C command to show the catalog, you can press =, =1, or =2 where the number corresponds to the desired drive number. Merlin 8/16 will set the prefix to the volume found in the current or specified drive and then catalog that volume.

If you press Open-Apple during the catalog, Merlin 8/16 lists only the directory files present in the specified pathname.

If you press Closed-Apple during the catalog, Merlin 8/16 lists only the Text (usually source) files present.

If you press *both* Open- and Closed-Apple keys simultaneously during a catalog, Merlin 8/16 lists only the BIN (usually object) files present. Note that these keys must be pressed and held throughout the entire catalog listing process.

If you enter a 1 as the first character of a pathname, or just 1 and Return, then the catalog will be sent to the printer in slot 1.

If Merlin 8/16 cannot find a disk volume specified in the current prefix for a catalog, it will ask for the correct volume to be inserted. This can be aborted by pressing Control-C.

### C (Catalog- DOS 3.3 Merlin 8 only)

When you press C, the catalog of the current diskette will be shown. The Command: prompt appears to let you enter a DOS command if desired. This facility is provided primarily for locking, unlocking and deleting files. Unlike the Load Source, Save Source, and Append File commands, you must type the .S suffix when referencing a source file from this prompt. Do not use it to load or save files. If you do not want to give a disk command, just press Return. You can use Control-X to cancel a partially typed command. If you press Control-C and Return after the Command: prompt, you will be returned to the Main Menu prompt (%). You can then issue any Main Menu command such as L for Load Source. This permits you to give a Main Menu command while the catalog is still on the screen. In addition, if Control-C is pressed at the catalog pause point, printing of the remainder of the catalog is aborted.

### L (Load Source)

This is used to load a source file from disk. This is a text file for ProDOS, or binary file for Merlin 8 DOS 3.3 version. You will be prompted for the name of the file. You do not have to append .S since Merlin 8/16 does this automatically. To cancel the Load Source command, just press Return and the command will be cancelled without affecting any file that may be in memory.

After a Load Source or Append Source command, you are automatically placed in the Editor. The source will automatically be loaded to the correct address.

NOTE: Subsequent Load Source or Save Source commands will display the last filename used, followed by the ? prompt. If you press Y, the current file name will be used for the command. If you press the space bar, the cursor will be placed on the first character of the filename, and you may type in the desired name. You can cancel the command by pressing Return without typing a file name. In Merlin 16, pressing the TAB key will move the cursor to the end of the default filename. Merlin 16 also lets you add a slash (/) character to the end of the name to tell Merlin not to add the .S suffix for a Load or Save operation. This is provided to save and load files to and from other editors that do not use the .S suffix.

### S (Save Source)

Use this to save a source file to disk . This will save a text for ProDOS and a binary file for the DOS 3.3 version. As with the load command, you do not include the .S suffix.

NOTE: The address and length of the current source file are shown on the Main Menu, and are for information only. You do not need use these for saving; Merlin 8/16 does this automatically. As in the Load Source command above, the last loaded or saved filename will be displayed. You can press Y to save the same filename, or the space bar for a new file name. You can cancel the command by pressing Return.

### A (Append File)

This loads in a specified source file and places it at the end of the file currently in memory. It operates in the same way as the Load Source command, and does not affect the default file name. It does not save the resultant combined (appended) file; you are free to do that if you wish.

### D (Drive Change - DOS 3.3 Merlin 8 only)

When you press D, the drive used for saving and loading will toggle from one to two or two to one. The currently selected drive is shown on the menu. When Merlin 8 is first started, the selected drive will be the one used at startupt. To change the slot number, press C for to display the current disk's catalog. Then give the disk command CATALOG,Sn, where n is the slot number. This action will catalog the newly specified drive.

See the C command for the method of changing drive specification with the ProDOS versions.

### D (Disk Command - ProDOS)

This allows you to issue disk-related commands. The following commands are available:

| | | |
|---|---|---|
| BLOAD | pathname [,A$....] | (only hex addresses allowed) |
| BRUN | pathname [,A$....] | (only hex addresses allowed) |
| - | pathname [,A$.... | (only hex addresses allowed) |
| BSAVE | pathname,A$adrs,L$len | |
| DELETE | pathname | |
| LOCK | pathname | |
| ONLINE | | (shows the volumes currently on line and their names) |
| PFX | pathname | (shorthand for Prefix) |
| POP | | ("pops" Prefix level) |
| PREFIX | pathname | (sets the prefix to pathname) |
| RENAME | old pathname,new pathname | |
| SLOT | slot number | (set new slot # used by PFX= command) |
| UNLOCK | pathname | |

A disk command returns to the disk command mode. You can then issue another disk command or just press Return to go back to the Main Menu.

#### Bload, Brun and -

Bload, Brun and "-" accept both BIN and SYS files. In Merlin 8, the difference between Brun and "-" is in the state of the softswitches when control is passed to the program. Brun leaves Merlin 8 up; that is, auxiliary zero page and language card RAM are selected. The "-" command switches in the

main zero page and the $D000-$FFFF ROMs. In Merlin 16, Brun and "-" act the same and both switch in the main zero page and the $D000-$FFFF ROMs. Using an RTS from such a program will return to Merlin 8/16. Most of the utility programs supplied with Merlin 8/16 such as Sourceror, XREF, etc. can be run by either method. You cannot use Brun to run programs such as the ProDOS FILER. In addition, such programs do not return to Merlin 8/16 and the /RAM/ volume is left disconnected by this procedure.

### Online

This command scans all attached disk devices, including RAM disks and hard disks, and reports each volume name. This is handy for identifying the volume name of all available ProDOS disk devices.

### Pfx, Prefix

If you are not sure of the volume or subdirectory pathname, you can type PFX= or PFX=1 to specify Slot 6, Drive 1, or PFX=2 for Slot 6, Drive 2.

When Prefix or Pfx is entered without a pathname, this command sets the prefix to the volume part of the current prefix. For example, if the current prefix is /MERLIN/LIB and you type Pfx and press Return at the disk command prompt, the prefix will revert to /MERLIN.

### Pop

Typing POP as a disk command "pops" the directory level in the current prefix by one level. For example, if the current prefix was "/MERLIN.16/SAMPLES/GRAPHICS", typing POP would change the prefix to "/MERLIN.16/SAMPLES".

### Slot

Slot is used to set the slot which *subsequent* PFX= commands will use to set the prefix from. That is to say, it does not itself set the prefix to a given slot, but rather identifies which constant slot future PFX= commands will use. For example, the following steps would set a prefix to that of the disk in a specified slot:

1) Press D for Disk Command, then type Online. This will help remind you know which slots, drives, and pathnames are currently active. (This is an optional step).

2) While still at the Disk Command prompt, type Slot n where 'n' is the desired slot and press Return.

3) At the Disk Command prompt type PFX=n where 'n' is the desired drive number and press Return.

4) Finally, press Return alone at the prompt. Note that the Prefix now shows the pathname of the specified slot and drive. **Simply put, typing the Disk Commands Slot 5 then PFX= will set the prefix to slot 5.**

**E (Enter Editor/Assembler - Merlin 8 only)**
**F (Enter Full Screen Editor/Assembler - Merlin 16 only)**

This command places you in the Editor/Assembler mode. It automatically sets the default tabs for the editor to those appropriate for source files.

NOTE: If you wish to use the editor to edit an ordinary (non-source) text file, you can type TABS and press Return to zero all tabs, once you are in the Editor.

**O (Save Object Code)**

This command is valid only after the successful assembly of a source file. In this case you will see the address and length of the object code on the menu. As with the source address, this is given for information only.

NOTE: The object address shown is the program's ORG (or $8000, bank 0, by default) and *not that of the actual current location of the assembled code*, which is ordinarily $8000 in auxiliary memory. When using this command, you are asked for a name for the object file. Unlike the Save Source command, no suffix will be appended to this filename.

Thus you can safely use the same name as that of the source file since the .S will not be appended to the filename. When this object code is saved to the disk its address will be the correct one, i.e., the one shown on the menu. Then when you Bload or Brun it, the program will load at that address, which can be anything ($300, $8000, etc).

**R (Read Text File - DOS 3.3 Merlin 8 only)**

This reads text files into Merlin 8. They are always appended to the current buffer. To clear the buffer and start fresh, type NEW in the editor. If no file is in memory, the name given will become the default filename. The Append command is similar, but does not change the default filename.

When the read is complete, you are placed in the Editor. If the file contains lines longer than 255 characters, these will be divided into two or more lines by the Read command. The file will be read only until it reaches HIMEM, and will produce an Out Of Memory error if it goes beyond. Only the data read to that point will remain.

The Read Text File and Write Text File commands will automatically add a T. prefix to the beginning of the filename you specify *unless* you precede the filename with a space or any other character in the ASCII range of $20 to $40 (%...0...9...?). This character will be ignored and not used by DOS in the actual filename. For example:

Read File:TEST will read a file called T.TEST

but,

Read File:?TEST will read a file called TEST

If you are trying to read a file called TEST and you get a File Not Found error, you can also use the Merlin 8 Catalog/Disk Command to rename the file to T.TEST.

The Read Text File and Write Text File commands are used to load or create PUT files, or to access files from other assemblers or text editors.

### W (Write Text File - DOS 3.3 Merlin 8 only)

This writes a Merlin 8 source file into a text file instead of a binary file. The speed of the Read Text File and Write Text File commands is approximately that of a standard DOS Bload or Bsave. The Write Text File routine does a Verify after the write.

### @ (Set Date - ProDOS)

This allows you to set the current date for ProDOS. Note that this option does not set the date on a clock card. If you have a clock, the date stamping is automatic (provided you have a Thunderclock or have installed the proper clock driver). The Set Date provision is intended for people who do not have a clock. In that case, you may use this to set the current date and this date will then be used for date stamping. You may also just use this to check on the current date. Press Return alone to exit the Set Date routine.

### Q (Quit - DOS 3.3 Merlin 8 only)

This exits to BASIC. You may re-enter Merlin 8 by typing ASSEM and pressing Return. This re-entry will not destroy the source file currently in memory. This exit can be used to give disk commands, test machine language programs, run BASIC programs, etc.

### Q (Quit - ProDOS)

This exits the Merlin Interpreter. If you launched Merlin 8/16, it will quit to the program that started up Merlin 8/16. If you cold started Merlin 8/16, you must specify the prefix for the next volume name and then the pathname of the next SYS file. In most cases this will be the BASIC.SYSTEM interpreter.

# THE MERLIN 8 EDITOR

There are two modes in the Merlin 8 Editor: the Full Screen Editor, which includes all editing commands, and the Command Mode from which assemblies and other functions are done.

From the Merlin 8 Main Menu, you can press E to enter the Command Mode of the Editor. The Merlin 8 Command Mode is indicated by the colon (:) prompt. No actual editing is done at this level. Rather, you can either type a command which will start up the Full Screen Editor, or you may use a number of specific Command Mode operations, which are of a general utility nature, such as to print a listing, assemble a file, convert number types, etc.

When you type an editing command such as A to Add etc. from the Command Mode, the colon prompt will disappear and the screen display will change to the Full Screen Editor.

## MERLIN 8 FULL SCREEN EDITOR COMMANDS

From the Merlin 8 Main Menu, press E to enter the Command Mode, then A to Add to source listing.

The current line number is shown at the upper right corner of the screen. To the left of the line number is a vertical bar which is the End-of-Line Marker. It indicates the position at which an assembly listing will overflow the printer line. You can put characters beyond this mark, but they should be for information only, and will not be printed within a printer listing.

The fields are tabbed, and the arrow keys can be used to move the cursor to the next tab position.

The Edit Mode commands are divided into two types: Control key commands which are line oriented, and Open-Apple key commands which are global, i.e. oriented to the entire listing. The control key commands edit text and move the cursor on just the line the cursor is presently on. Use the Open-Apple key commands to make changes to groups of lines, or to move about in the listing. All editing commands work whether the Full Screen Editor was started up using the Add, Insert or Edit comands. When you are through editing, press Open-Apple-Q. The line is accepted as it appears on the screen, no matter where the cursor is when you exit the Edit Mode.

To get the most out of the Merlin 8 Full Screen Editor, you should keep in mind that a full screen editor is like a word processor. That is, any character you type is immediately entered into whatever line the cursor is on.

With the Merlin 8 Full Screen Editor, if you can see it on the screen, you can edit it, and moving to a line is a simple matter of using the arrow keys or other special commands to move to the part of the listing you want to edit. Just remember, when you are using the Full Screen Editor, think of yourself as using a word processor where you can freely scroll to whatever part of the page you want to edit, and the final document is just your source listing.

# MERLIN 8 CONTROL KEY COMMANDS
## (Line oriented)

### Control-B (Beginning of line)

Moves the cursor to the beginning of the line.

### Control-D (Delete character)

Deletes the character under the cursor. Also see Delete.

### Control-F (Find)

Finds the next occurrence on the current line of the character typed after the Control-F. The cursor changes to an inverse F to indicate the Find Mode. To move the cursor to the next occurrence on the line, press the desired character key again. Typing any other character will exit the Find Mode and enter the text typed at that position.

### Control-I or TAB (Toggle insert cursor)

Toggles the cursor mode between the insert cursor designated by an inverse I, and overstrike cursor which appears as an inverse block. The insert mode of the cursor should not be confused with entering the Full Screen Editor using the Add and Insert commands. The cursor can be in the insert mode regardless of whether lines are being added or inserted. The insert mode of the cursor refers only to whether individual characters are being inserted or typed over.

The character insert mode defaults to ON upon entry. When you change it with Control-I, it remains that way until changed again. Thus, moving from one line to another has no effect on this status.

### Control-L (Lower case convert)

The Merlin 8/16 PARMS file can be configured so that unless the cursor is in a comment or an ASCII string, lower case characters will be converted to UPPER CASE characters. To override this conversion, or to reinstate it, just use the Control-L command. This conversion is also in effect when you use the Open-Apple-F, Open-Apple-W, or Open-Apple-L find commands to specify the text to find. Even if enabled in the PARMS file, this conversion is defeated when the tabs are zeroed.

### Control-N (End of line)

Moves the cursor to the end of the line.

### Control-O (Other characters)

This is used as a special prefix key from the Command Mode. For example, if you wanted to type a Control-I or an Escape as part of a PRTR initialization string, you would press Control-O, followed by the control character you desire. The control character will appear in inverse. For multiple control characters, Control-O must be typed before each character is entered.

### Control-R (Restore)

This command restores the original line. For example, if you have used Control-Y to delete all characters to the end of the line, you can press Control-R to undo the effects of the Control-Y command.

### Control-S (Status box)

This command displays a status box showing the number free and used bytes.

### Control-T (Set marker on current line - Vertical "Tab")

This command can be used to set a marker at the current line for recall by the Open-Apple-T command.

### Control-W (Find word)

This command moves the cursor to the beginning of each word in the line (alphanumeric).

### Control-X (Exit global exchange, etc.)

This command can be used to cancel any global exchange, text selection, or string search while it is in progress.

### Control-Y (Delete to end of line)

Deletes all characters from the cursor to the end of the line.

### Arrow keys (Cursor movement)

The arrow keys move the cursor in the specified direction.

### Delete (Delete character)

Deletes the character to the left of the cursor. Also see Control-D.

### Escape (Move to beginning of next line)

This command moves the cursor to the beginning of the next line. This is similar to Return except that Escape does not insert a blank line.

### Return (Insert blank line)

Pressing Return anywhere in the line causes the cursor to move to the beginning of the next line and insert a blank line.

### TAB (Toggle insert cursor)

Toggles the cursor mode between the insert cursor, which is shown as an inverse I, and the overstrike cursor, which is shown as an inverse block.

Moving from one line to another has no effect on the status of the cursor; it only changes when toggled with TAB. Also see Control-I.

# MERLIN 8 OPEN APPLE KEY COMMANDS
## (Entire listing oriented)

In addition to the line-oriented commands (control key commands), the Merlin 8 Full Screen Editor uses Open-Apple (Ó) key commands to move within the listing, and to edit entire lines of text. These commands are as follows:

### ÓB (Beginning of source)

This command moves the cursor to the beginning of the source listing.

### ÓD (Delete current line)

This command deletes the current line and places it in a special 'undo' buffer which is independent of the clipboard.

The ÓR command replaces the current line with the contents of the 'undo' buffer. Therefore, to move a single line to another location, you could place the cursor on the line to be moved, and then type ÓD to delete the line. Then move the cursor to another line, press Return, ÓI or ÓTab to create an empty line, and press ÓR to replace that line with the deleted line. Also see ÓDelete.

### ÓE (Global Exchange, also called 'Find & Replace)

Sometimes called 'Find & Replace,' this command will let you search for a group of words, and replace them with another. The ÓE command opens a dialog box that asks for the text to change, and the new text to replace it. If you press Return alone for either of these, the command is canceled.

If you enter the text in both fields and press Return, the file is then searched for the change text. Unlike the FIND command, it looks only for full words. That is, the text found must be bounded by non-alphanumeric characters or it will be ignored.

If text is found with this method, the screen is reprinted with the replacement made and the cursor is placed on the first character of the replacement. Now you must press a key to continue. Pressing Return (or most any other control character) will defeat the change and the command will look for the next occurrence of the text to change. Pressing the space bar or any other character, except A, will accept the change and the routine will continue.

You can back out of the global exchange while the cursor is on an entry by pressing Control-X. You can also press the A key, which will cause *all* occurrences to be changed.

You can tell when the routine is finished by the fact that during the exchange sequence, the line number at the top right is missing. The line number will return when there are no more matches for the change text, or when you press Control-X.

### ÓF (Find text)

The ÓF command opens a window which asks for the text to find. It then finds the first occurrence of the text in the entire text file. The text can be anywhere on a line. After the first find, you can find the next occurrence by typing another ÓF. You can edit the line, and then type ÓF to go to the next occurrence.

During the search function, one or more plus (+) signs will be shown next to the line number at the top right of the screen. This is only an indicator that the search function is active. The number of plus signs shown is arbitrary, and has no relation to the total number of occurrences.

In Merlin 16, if the ÓF command is used after text has been selected, only the selected text will be searched for the text to be found. When the search has been completed, the text is no longer selected. Thus, you can use the ÓY, ÓC or ÓX commands to search just a portion of your listing.

The ÓB command and the Control-S status command both cancel the Find mode, as does failure to find the text below the current line.

The ÓW command is identical to ÓF except that it finds only whole words bounded by non-alphanumeric characters. If you type either ÓW or ÓF to find the next occurrence, this mode will change accordingly.

In all cases the line containing the text is moved to the center of the screen, unless it is within the first 10 lines of the start of the source.

### ÓI (Insert line)

Pressing ÓI will insert a blank line at the cursor. Also see ÓTAB.

### ÓL (Locate label, marker or line number)

This command will locate the first occurrence of a label or any text in the label column. Only the characters typed are compared with the labels. Thus the search string LOOP would jump to the label LOOP2 if LOOP did not occur first. To find a specific label when there may be another similar label, end the input with a single space.

If a number is entered after this command, the cursor will move to the beginning of the line number specified. This is particularly handy when editing a source file from a printed listing.

The intended use for this command is to move rapidly to a particular place in the source. You can use create your own 'markers' to enhance the capability of this command. Thus, if a line has *7 in it, you can specify *7 as the text to find for this command and it will locate it.

In all cases the line containing the text is moved to the center of the screen, unless it is within the first 10 lines of the start of the source.

### ÓN (End of source)

This command moves the cursor to the end of the source listing.

### ÓQ (Quit full screen editor)

This command returns to the Command Mode.

### ÓR (Replace)

This command exchanges the current line with the contents of the 'undo' buffer. Therefore, pressing ÓR a second time will cancel the effect of the first press.

Using ÓR when the cursor is on blank line will piace the contents of the 'undo' buffer on the line and place the empty line in the 'undo' buffer.

The ÓR command can be used to move a single line. Place the cursor at the beginning of the line to be moved and press ÓR. Move the cursor to the desired location, press Return to insert a blank line, and press ÓR again.

ÓR can be used by itself to easily interchange two lines. Just place the cursor on the first line, press ÓR, move the cursor to the second line and press ÓR again. Then move the cursor back to where the first line was and press ÓR for the third, and final time. Also see ÓD.

### ÓT (Return to line marker (vertical "Tab")

This command returns to the line marked by the last Control-T command.

### ÓV (Paste)

Pastes the contents of the clipboard at the line containing the cursor. Only full lines are moved. Using this command does not change the contents of the clipboard, so this command can be used to replicate a range of lines.

If the ÓV paste command is issued when a range of text has been selected, the text on the clipboard will be inserted before the last line of selected text.

### ÓW (Find word)

The ÓW command is identical to ÓF except that it finds only whole words bounded by non-alphanumeric characters. If you type either ÓW or ÓF to find the next occurrence, this mode will change accordingly.

If the ÓW command is used after text has been selected, only the selected text will be searched for the word to be found. When the search has been completed, the text is no longer selected. The search can be cancelled with Control-X.

### ÓX (Cut to Clipboard)

ÓX starts the select mode to cut text. The first time ÓX is pressed, the current line is selected and is shown in inverse. Use the down arrow or Escape keys to extend the selection if desired, or press any other key to cancel the selection. Additional selected lines are shown in inverse. Use the up arrow key to adjust the range selected if you go too far. The select mode will be canceled if you move the cursor above the first selected line or *past the top of the current screen.*

The second time ÓX is pressed the selected text is cut from the listing and is placed on the clipboard. Merlin 8 has no "copy" command, but if you immediately follow the ÓX command that cuts a portion of text with ÓV (for paste), the desired text will be on the clipboard, and the original listing will be restored.

If you are unfamiliar with the idea of a "clipboard", this is just an analogy to how you might put a piece of paper clipped from a magazine, letter, etc. on a clipboard, to hold it temporarily while you were getting ready to put it in its final location. In Merlin 8, the clipboard refers to a memory buffer that holds the text you have cut while you decide where you want the final text placed. Using the clipboard, you can cut tet from one source file, load another, and then paste the text into a second file. The clipboard is cleared when a file is assembled.

### ÓY (Select all text)

This command selects all text to be cut from the current line to the end of the listing. ÓX will cut the text, while pressing any other key will cancel the selection. This technique can be used to move the entire listing to the clipboard.

### ÓZ (Center screen)

This command repositions the screen so that the line the cursor is on becomes the center line on the screen.

### ÓDelete (Delete)

This command deletes the line *above* the cursor and places it in a special 'undo' buffer which is independent of the clipboard.

The ÓR command replaces the current line with the contents of the 'undo' buffer. Therefore, you could use ÓDelete to delete a line, move the cursor to another line, press Return, ÓI or ÓTab to insert a line, and press ÓR to replace that line with the deleted line.

### ÓTAB (Insert line)

Pressing ÓTAB will insert a blank line at the cursor. Also see ÓI

### ÓDown arrow (Move half-screen down)

Moves the cursor down 10 lines; that line then becomes the center line on the screen. This command has the effect of moving the current line to the top of the screen and then moving the cursor to what was the bottom line on the screen.

### ÓUp arrow (move half-screen up)

Moves the cursor up 10 lines; that line then becomes the center line on the screen. This command has the effect of moving the current line to the bottom of the screen and then moving the cursor to what was the 1st line on the screen.

### ⍁8 (Asterisk)

Produces a line of 32 asterisks. Overstrikes existing line, if any. Undo with Control-R.

### ⍁9 (Box)

Produces an asterisk, 30 spaces, and then another asterisk. This and the ⍁8 command can be used to produce a large box for titles and other information. Overstrikes existing line, if any. Undo with Control-R.

### ⍁- (Hyphen)

Produces a line of 1 asterisk and 31 hyphens. Overstrikes existing line, if any. Undo with Control-R.

### ⍁= (Equal sign)

Produces a line of 1 asterisk and 31 equal signs. Overstrikes existing line, if any. Undo with Control-R.

# MERLIN 8 EDITOR COMMAND SUMMARY

## CONTROL KEY COMMANDS (line oriented)

The Control Key commands consist of cursor moves and line oriented commands.

Control-B ..... Moves cursor to beginning of line
Control-C ..... Cancel assembly and return to Command Mode
Control-D ..... Deletes character under the cursor
Control-F ..... Finds next occurrence of next character typed
Control-I ..... Toggles insert and overstrike cursor
Control-L ..... Toggles lower case conversion
Control-N ..... Moves cursor to end of line
Control-O ..... Prefix key for typing optional characters
Control-R ..... Restores original line
Control-S ..... Displays status box
Control-T ..... Set marker on current line for recall by ⍁T (vertical "Tab")
Control-W ..... Finds next occurrence of word in line
Control-X ..... Exits global exchange, etc. while in progress
Control-Y ..... Delete characters to end of line
Arrows ..... Moves the cursor in the specified direction
Delete ..... Deletes character to left of cursor
Escape ..... Moves cursor to beginning of next line
Return ..... Moves cursor down and inserts blank line
TAB ..... Toggles insert and overstrike cursor

## OPEN-APPLE KEY COMMANDS (entire listing oriented)

The Open-Apple Key commands are global commands, which means they are generally oriented to the whole listing as opposed to just the current line (or a single character).

| | |
|---|---|
| ÓB | Moves to beginning. Cursor on center line |
| ÓD | Deletes line and places it in 'undo' buffer |
| ÓE | Global Exchange (Search & Replace) |
| ÓF | Finds next occurrence of text entered |
| ÓI | Inserts blank line at cursor |
| ÓL | Finds first occurrence of label or line |
| ÓN | Moves cursor to end of listing |
| ÓQ | Returns editor to Command Mode |
| ÓR | Swaps current line with 'undo' buffer |
| ÓT | Goes to line of last Control-T (go to vertical "Tab") |
| ÓV | Pastes contents of clipboard on current line |
| ÓW | Finds next occurrence of whole word |
| ÓX | Start text selection/Cut selected text to clipboard |
| ÓY | Selects text from current line to end of file |
| ÓZ | Current line becomes center line on screen |
| ÓDelete | Deletes line above cursor; puts in 'undo' buffer |
| ÓTAB | Inserts a blank line at cursor |
| ÓDown | Moves cursor down 10 lines |
| ÓUp | Moves cursor up 10 lines |
| Ó8 | Produces a line of 32 asterisks |
| Ó9 | Produces 1 asterisk, 30 spaces, and 1 asterisk |
| Ó- | Produces a line of 1 asterisk followed by 31 hyphens |
| Ó= | Produces a line of 1 asterisk followed by 31 equal signs |

## MERLIN 8 GENERAL REMARKS

When you move the cursor between lines, its horizontal position will jump around. This is because it is based on the actual position in the line and not on the screen position. If the tabs are zeroed you will not notice this, except for the fact that the cursor is never beyond the last character in the line.

The maximum line length is 80 characters. Lines longer than that will be truncated IF they are edited.

You must return to the Command Mode (ÓEscape) in order to use the ASM command to assemble, MON to use the Merlin 8 Monitor, or to Quit and go to the Main Menu, etc. An assembly will delete the contents of the clipboard.

### OOPS

Virtually any Editor action can be undone. You should remember that the proper undo command is of the same 'type' as the command you want to undo. Thus, any Control key command is undone by Control-R. This includes the Ó8, Ó9, Ó-, and Ó= commands which are considered line oriented commands for this purpose.

The line deletion commands ÓD and ÓDelete are undone by creating an empty line with ÓTab followed by ÓR. If you forget to create the empty line, type another ÓR and then insert the empty line to receive the undo buffer contents.

The ÓR command undoes itself.

A Cut (ÓX) is undone by a Paste (ÓV) without moving the cursor off its line.

If you are entering a line of text in response to a prompt, such as a filename, PRTR initialization, or dialog box, you can press Control-C or Control-X to cancel the line.

### ED.16

The full screen editor in Merlin 8 can be replaced with a version written specifically for the 65802 or 65816 chip. This editor is named ED.16 on both DOS 3.3 and ProDOS Merlin 8 disks. To enable ED.16 on the DOS 3.3 disk, change line 80 of the HELLO program to BRUN ED.16 instead of ED as is now done. On the ProDOS disk, you will have to rename ED to ED.OLD, and rename ED.16 to ED. Although ED.16 is provided on the Merlin 8 disks, it is unlikely you will ever need it. We recommend that if you do have the 65802 or 65816 that you use Merlin 16 instead of Merlin 8.

### EDMAC and KEYMAC

These are macro utilities that will automatically type entire phrases or lines in your source file with a single keystroke. EDMAC is for use with the full screen editor, KEYMAC is for use with the line editor (presuming that ED has not been loaded). See the Utilities section of this manual for a description of the actual keyboard definitions in these files. EDMAC and KEYMAC can be activated within Merlin 8 by typing BRUN EDMAC or BRUN KEYMAC from the Main Menu, as appropriate.

### Editor Technical Information

The ED and ED.16 files have been arranged so that certain parameters can be changed with a little effort. At relative byte 3 in the file there is an address that points to the main part of the program, hereinafter referred to as START, that is past a relocating header. At START+10 is a table of command characters used without the Ó key. This table ends in a zero. Following this is the table of the key commands used with the Ó, again ending in zero. If you are inclined, you can change these definitions to change the command keys of the editor.

# MERLIN 8 COMMAND MODE

## GENERAL GUIDELINES FOR THE COMMAND MODE

The Command Mode is used for assemblies, printing the source listing, and other general-purpose functions. The command mode is entered by pressing E at the Main Menu, or by pressing ÓQ while in the Full Screen Editor.

For most of the Merlin 8 Command Mode operations, only the first letter of the command is required, the rest being optional. This manual will show the required command characters in UPPER case and the optional ones in lower case.

## ABOUT THE MERLIN 8 COMMAND MODE DOCUMENTATION

For each of the commands available in the Merlin 8 Command Mode, the documentation consists of three basic parts:

1) the name and syntax of the command
2) examples of the use of each available syntax
3) a description of the function of each command

When the syntax for each command is given:

PARENTHESES () indicate a required value
ANGLE BRACKETS <> indicate an optional value or character
SQUARE BRACKETS [] are used to enclose comments about the commmand

### Line Numbers in Command Mode

With some commands, you must specify a line number, a range of line numbers, or a range list. A line number is just a number. A range is a pair of line numbers separated by a comma. A range list consists of several ranges separated by a slash (/).

Line Number examples:

| | | |
|---|---|---|
| 10 | LINE # | [ a single line number ] |
| 10,30 | RANGE | [ the range of lines 10 to 30 ] |
| 10,30/50,60 | RANGE LIST | [ ranges 10 to 30 AND 50 to 60] |

If a line number in a range exceeds the number of the last line in the source, the editor automatically adjusts the specified line to the last line number. For example, if you wanted to Delete all the lines past 100 in a source listing, entering D100,9999 would do it.

### Delimited Strings (or d-strings)

Several commands allow specification of a string. The string must be delimited by a non-numeric character other than the slash or comma. Such a string is called a delimited or d-string. The usual delimiter is single or double quote marks (' or ").

Delimited string examples:

'this is a delimited string'
"this is a delimited string"
@this is another d-string@

Note that the slash (/) cannot be used as a delimiter since it is the character that delimits range lists in the Editor.

### Wild Card Characters in Delimited Strings

For all of the commands that use delimited strings or d-strings, the '^' character acts as a wild card character. For example, the d-string 'Jon^s' is equivalent to both 'Jones' and 'Jonas' d-strings.

### Upper and Lower Case Control

The shift and caps lock keys work as you would expect. While editing or entering a line of text, you can also use the Control-L command, described earlier in this section.

## COMMAND MODE COMMANDS

Following are the commands recognized by Merlin 8 in the Command Mode of the Editor. The Command Mode is indicated by the colon prompt (:).

### A (Add)

A [ only option for this command ]

The Add command places you in the Full Screen Editor at the end of the existing source listing, if any. Adding lines is much like entering additional BASIC lines with auto line numbering. To exit from ADD mode, press ♂Q.

You may enter an empty line by pressing space then Return. This is useful for visually blocking off different parts of a listing.

### E (Edit)

Edit (line number)

| | |
|---|---|
| E 10 | [ edits line number 10 ] |

This enters the full screen editor, with the cursor on the specified line.

### C (Change)

Change (d-string d-string)
Change (line number) <d-string d-string>
Change (range) <d-string d-string>
Change (range list) <d-string d-string>

| | |
|---|---|
| C "hello"goodbye | [ finds "hello" and if told to do so will change it to "goodbye"] |
| C 50 "hello"bye | [ changes in line 50 only ] |
| C 50,100 "Hello"BYE | [ changes lines 50 through 100 ] |
| C 50,60/65,66 "AND"OR | [ changes in lines 50 through 60 and lines 65 and 66] |

This changes occurrences of the first string to the second string. The strings must have the same delimiters. For example, to change occurrences of *speling* to *spelling* throughout the range 20,100, you would type C 20,100 "speling"spelling. If no range is specified the entire source file is used.

Before the change operation begins, you are asked whether you want to change *all* or *some*. If you select *some* by pressing the S key, the editor stops whenever the first string is found and displays the line as it would appear with the change.

If you then press the Y key, the change will be made. If you press Return, the change will not be made. Typing any control character such as Escape, Return or any others will result in the change not being made. Any other key, such as Y or even N, will accept the change. Control-C or the slash (/) key will abort the change process.

### COPY

COPY (line number) TO (line number)
COPY (range) TO (line number)

| | |
|---|---|
| COPY 10 TO 20 | [ copies line 10 to just befor line 20 ] |
| COPY 10,20 TO 30 | [ copies lines 10 through 20 to just before line 30 ] |

This copies the line number or range to just *above* the specified number. It does not delete anything.

### CW (Change word)

Change (d-string d-string)
Change (line numbers) <d-string d-string>
Change (range) <d-string d-string>
Change (range list) <d-string d-string>

| | |
|---|---|
| CW "PTR"PRT | [ change all "PTR"s to "PRT"s ] |
| CW 20 "PTR"PRT | [ as above but only in line 20 ] |
| CW 20,30 "PTR"PRT | [ do the same as the above but for lines 20 through 30 ] |
| CW 1,9/20,30 "PTR"PRT | [ same as above but include lines 1 through 9 in the range ] |

This works similar to the CHANGE command with the added features as described under Find Word (FW).

### D (Delete)

Delete (line number)
Delete (range)
Delete (range list)

| | |
|---|---|
| D 10 | [ deletes line number 10 ] |
| D 10,32 | [ deletes lines 10 through 32 ] |
| D 20,30/10,12 | [ deletes ranges of lines 10 through 12 and 20 through 30 ] |

This deletes the specified lines. Unlike BASIC, the line numbers are fictitious; they change with any insertion or deletion.

**NOTE:** When deleting several blocks of lines at the same time, you *must* specify the higher range first for the correct lines to be deleted.

### F (Find)

Find (d-string)
Find (line number) <d-string>
Find (range) <d-string>
Find (range list) <d-string>

| | |
|---|---|
| F "A String" | [ finds lines with "A String" ] |
| F 10 "STRING" | [ finds "STRING" if in line 10] |
| F 10,20 "HI" | [ finds lines in range of 10 through 20 that contain "HI"] |
| F 10,20/50,99 "HI" | [ finds lines that contain "HI" in range of 10 through 20 and 50 through 99 ] |

### FIX

```
FIX                    [ only option for this command ]
```

This undoes the effect of the TEXT command. It also does a number of technical housekeeping chores. It is recommended that FIX be used on all source files from external sources that are being converted to Merlin 8 source files, after which the file should be saved.

NOTE: The TEXT and FIX commands are somewhat slow. Several minutes may be needed for their execution on large files. FIX will truncate any lines longer than 255 characters.

### FW (Find Word)

```
FW (d-string)
FW (line number) <d-string>
FW (range) <d-string>
FW (range list) <d-string>
  FW "LABEL"           [ find all lines with "LABEL" ]
  FW 20 "LABEL"        [ try to find "LABEL" in 20 ]
  FW 20,30 "PTR"       [ find all lines between 20 and 30 that contain "PTR" ]
  FW 20,30/50,99 "PTR" [ find all lines between 20 and 30 and between 50 and 99 that
                         contain the word "PTR" ]
```

This is an alternative to the FIND command. It will find the specified word only if it is surrounded, in source, by non-alphanumeric characters.

Therefore, FW "CAT" will find:

```
CAT
CAT-1
(CAT,X)
```

but will not find CATALOG or SCAT.

## GET

GET ( obj adrs )

| | |
|---|---|
| GET | [ put object code in Main Memory at the address specified in the source's ORG ] |
| GET $4000 | [ put object code at location $4000 in Main Memory ] |

This command is used to move the object code, after an assembly, from its location in Auxiliary Memory to its ORG location in Main Memory. The address must be above the existing source file, if any, and it will not be allowed to overwrite DOS. You can do a NEW if you want to load it lower in memory than allowed, but you must remember to save the source first. You cannot use GET to put object code at memory locations lower than $901, but you can go to the Monitor afterwards and use it to move the object to any desired location. However, any such move using the Monitor may destroy your source or other data necessary to Merlin 8's operation. Caution should be used.

The GET command does not check if a valid object code has been assembled.

This command is supplied for convenience only. The recommended method for testing a program is to save the source code first, save the object code, and then run the program from BASIC or with the G command from the Monitor.

## HEX-DEC CONVERSION

128 = $0080
$80 = 128

If you type a positive or negative decimal number in the Command Mode, the hex equivalent is returned. If you type a hex number using the $ prefix, the decimal equivalent is returned. All commands accept hex numbers.

## L (List)

List
List (line number)
List (range)
List (range list)

| | |
|---|---|
| L | [ list entire file ] |
| L 20 | [ list line 20 only ] |
| L 20,30 | [ list 20 through 30 ] |
| L 20,30/40,42 | [ list 20 through 30 and then list lines 40 through 42 ] |

Lists the source file with line numbers. Control characters in source are shown in inverse, unless the listing is being sent to a printer or other non-standard output device.

The listing can be aborted by Control-C or with the slash (/) key. You may stop the listing by pressing space and then advance a line at a time by pressing space again. By holding down space, the auto-repeat feature of the Apple will result in a slower listing. Any other key will resume the normal speed. This space bar technique also works during assembly and the symbol table printout. Any other key will restart it. The space bar pause also works during assembly and the symbol table printout.

### LEN (Length)

LEN [ only option for this command]

This gives the length in bytes of the source file, and the number of bytes free.

### MON (Monitor)

MON [ only option with this command ]

This exits to the Monitor. You may return to the Merlin 8 Main Menu by pressing Control-C, Control-B, or Control-Y. These commands re-establish important zero page pointers from a safe area inside Merlin 8. Thus Control-Y will give a correct entry even if you have damaged the zero page pointers while in the Monitor. DOS is not connected while using this entry to the Monitor.

You may return to the editor directly by typing 0G and pressing Return but unlike the above commands, this uses the zero page pointers stored at $0A-$0F. Therefore, you must be sure that these pointers have not been altered. For normal usage, any of the three Control commands should be used to return to Merlin 8.

When you exit to the Monitor with the MON command, the RAM-based $D000-$FFFF memory is enabled, and therefore, Merlin 8 and it's symbol table if any. If you want to examine the ROM memory that would normally correspond to Applesoft and the F8 Monitor, you should quit Merlin 8 with the Main Menu Quit command, and enter the Monitor with Call -151. Under ProDOS, this procedure necessitates loading BASIC.SYSTEM which removes Merlin 8 from memory.

### MOVE

MOVE (line number) TO (line number)
MOVE (range) TO (line number)

| | |
|---|---|
| MOVE 10 TO 20 | [ Move line 10 to just before 20] |
| MOVE 10,20 TO 30 | [ Move lines 10 through 20 to just before line 30 ] |

This is the same as COPY but after copying, automatically deletes the original range.

NEW

NEW [ only option for this command ]

Deletes the present source file in memory.

### P (Print without format)

Print
Print (line number)
Print (range)
Print (range list)

| | |
|---|---|
| P | [ print entire file ] |
| P50 | [ print line 50 only ] |
| P50,100 | [ print lines 50 through 100 ] |
| P1,10/20,30 | [ print 1 through 10 and then print lines 20 through 30 ] |

This is the same as LIST except that line numbers are not added. See PRTR for formatted printouts.

### PRTR (Formatted printout)

PRTR (command)

| | |
|---|---|
| PRTR 1 | [ activate printer in slot 1 with no printer init string ] |
| PRTR 1 ""Page Title" | [ printer in slot 1, no printer init string, "Page Title" is the page header ] |
| PRTR 1 "<Control-I>80N" | [ as above, add Control-I80N to initialize the printer ] |
| PRTR 3 | [ send formatted listing to screen ] |

This command is for sending a listing to a printer with page headers and provision for page boundary skips. No header is printed on the first page of the printout. See the section on Configuration for details on setting up default parameters, also TTL in the section on The Assembler.

The entire syntax of this command is:

PRTR # "(string)"<page header>"

If the page header is omitted, the header will consist of page numbers only.

*The initialization string may not be omitted if a page header is to be used.* If no special string is required by the printer, use a null string of two quotes only (""), as in the example showing "Page Title" in which case a carriage return will be used.

No output is sent to the printer until a LIST, PRINT, or ASM command is issued. See Control-O for information on inserting Control characters in the printer init string. The PRTR command only affects the next output command, and is canceled at the end of the listing.

### Q (Quit)

| | |
|---|---|
| Q | [ only option for this command ] |

Exits to Main Menu.

### TABS

TABS <number><, number><,...> <'tab character'>

| | |
|---|---|
| TABS | [ clear all tabs ] |
| TABS 10,20 | [ set tabs to 10 & 20 ] |
| TABS 10,20 " " | [ as above, space is tab character ] |

This sets the tabs for the editor, and has no effect on the assembler listing. Up to nine tabs are possible. The default tab character is a space, but any may be specified. The assembler regards the space as the only acceptable tab character for the separation of labels, opcodes, and operands. If you don't specify the tab character, then the last one used remains. Entering TABS and a Return will set all tabs to zero.

### TEXT

| | |
|---|---|
| TEXT | [ only option for this command ] |

This converts all spaces in a source file to inverse spaces. The purpose of this is for use on word processing type text files so that it is not necessary to remember to zero the tabs before printing such a file. This conversion has no effect on anything except the Editor's tabulation. The command FIX undoes the effect of the TEXT command.

### TROF (Truncate Off)

| | |
|---|---|
| TROF | [ only option for this command ] |

When used as an Immediate command, returns to the default condition of the truncation flag which also happens automatically upon entry to the editor from the Main Menu or from the Assembler. All source lines when listed or printed will appear normal.

### TRON (Truncate On)

```
TRON                    [ only option for this command ]
```

When used as an Command Mode command, sets a flag which, during LIST or PRINT, will suppress printing of comments that follow a semicolon. It makes reading of some source files easier.

### USER

```
USER
USER 1                  [ example for use with XREF ]
USER 0: FILENAME        [ example for use with PRINTFILER ]
```

This does a JSR to the routine at $3F5. This is the location of the Applesoft ampersand vector which normally points to a RTS. USER is designed to connect the various utilities supplied with Merlin 8 and for user defined printer drivers. You must be careful that your printer driver does not use zero page addresses, with the exception of the I/O pointers and $60-$6F, because this is likely to interefere with Merlin 8's heavy usage of zero page. Several supplied utilities operate through the USER command.

### VAL

```
VAL "expression"
    VAL "PTR            [ return value of label "PTR" ]
    VAL "LABEL"         [ gives the address (or value) of LABEL for the last assembly
                          done or "unknown label" if not found. ]
    VAL "$1000/2"       [ returns $0800 ]
    VAL "%1000"         [ returns $0008 ]
```

This will return the value of the expression as the assembler would compute it. All forms of label and literal expressions valid for the assembler are valid for this command. Note that labels will have the value given them in the most recent assembly.

### VID (VIDeo)

```
VID (slot number)
    VID 3               [ turns on 80 column display ]
```

This command selects an 80 column display device. To turn off the display, use Escape Ctrl-Q for the Apple IIe, IIc or IIgs video; use Escape-0 for the Videx UltraTerm. VID 3 is required to re-activate the display if you have set the PARMS file to switch to 40 columns for the PRTR command.

**W (Where)**

Where (line number)

| | |
|---|---|
| W 50 | [ where is line 50 in memory ] |
| W 0 | [ where is end of source file ] |

This prints in hex the location in memory of the start of the specified line. *Where 0* or *W0* will give the location of the end of source.

**. [period]**

| | |
|---|---|
| . | [ only option for this command ] |

Lists starting from the beginning of the last specified range. For example, if you type *L10,100*, lines 10 to 100 will be listed. If then you use the period (.) command, the listing will start again at 10 and continue until stopped. The end of the range is not remembered.

**/ (slash)**

/ <line number>

| | |
|---|---|
| / | [ start to list at last line listed ] |
| /50 | [ start listing at line 50 ] |

This command continues the listing from the last line number listed, or, when a line number is specified, from that line. This listing continues to the end of the file or until it is stopped as in LIST.

# ASSEMBLING A MERLIN 8 FILE

Once you have entered and edited your source listing, you will want to assemble it. ASM does that. Remember to exit the Full Screen Editor by pressing ÓEscape. When you see the colon (:) prompt, type ASM and press Return.

### ASM (Assemble)

ASM [ only option for this command ]

This passes control to the assembler, which attempts to assemble the source file.

If you wish to have a formatted printed listing of an assembly, just use the PRTR command immediately before typing in the ASM command.

### Control-C (Cancel assembly)

Terminates assembly and returns to the Command Mode.

### Control-D (Toggle Display Status)

Control-D [ only option for this command ]

During the second pass of assembly, pressing Control-D will toggle the list flag, so that the listing will either stop or resume. The next LST opcode in the source overrides this, but another Control-D can be used again.

# THE MERLIN 16 EDITOR

From the Merlin 16 Main Menu, you can press F to enter the Full Screen Editor, or if you have loaded a source file, you will enter the Full Screen Editor automatically.

At the upper right hand corner of the screen, the number of the line on which the cursor is located is shown. Somewhat to the left of this you may see a vertical bar. This bar is the End-of-Line Marker and it indicates the position at which an assembly listing will overflow the printer line. You can put characters beyond this mark, but they should be for information only, and will not be printed within a printer listing.

The fields are tabbed, and the arrow keys can be used to move the cursor to the next tab position.

The commands are divided into two types: Control key commands which are line oriented, and Open-Apple key commands which are global and thus oriented to the entire listing. The Control key commands edit text and move the cursor on just the line the cursor is presently on. Use the Open-Apple key commands to make changes to groups of lines, or to move about in the listing. When you are through editing, the line is accepted as it appears on the screen, no matter where the cursor is. The file is assembled by pressing ⌘A; ⌘Q will return you to the Main Menu. Other commands are available in the Command Box (⌘O).

To get the most out of the Merlin 16 Full Screen Editor, you should keep in mind that a full screen editor is like a word processor. That is, any character you type is immediately entered into whatever line the cursor is on.

With the Merlin 16 Full Screen Editor, if you can see it on the screen, you can edit it, and moving to a line is a simple matter of using the arrow keys or other special commands to move to the part of the listing you want to edit. Just remember, when you are using the Full Screen Editor, think of yourself as using a word processor where you can freely scroll to whatever part of the page you want to edit, and the final 'document' is just your source listing.

## MERLIN 16 CONTROL KEY COMMANDS (Line oriented)

### Control-B (Beginning of line)

Moves the cursor to the beginning of the line.

### Control-C (Cancel assembly)

Cancels assembly and returns to the Editor with cursor on line where assembly was interrupted.

### Control-D (Delete character)

Deletes the character *under* the cursor. Also see Delete.

### Control-F (Find)

Finds the next occurrence on the current line of the character typed after the Control-F. The cursor changes to an inverse F to indicate the Find Mode. To move the cursor to the next occurrence on the line, press the desired character key again. Typing any other character will exit the Find Mode and enter the text typed at that position.

### Control-I or TAB (Toggle insert cursor)

Toggles the cursor mode between the insert cursor (inverse I) and overstrike cursor (inverse block). The cursor can be in the insert mode regardless of whether lines are being added or inserted. The insert mode of the cursor refers only to whether individual characters are being inserted (inverse I) or typed over (inverse block).

The character insert mode defaults to ON upon entry. When you change it with Control-I, it remains that way until changed again. Thus, moving from one line to another has no effect on this status.

### Control-L (Lower case convert)

The Merlin 16 PARMS file can be configured so that, unless the cursor is in a comment or an ASCII string, lower case characters will be converted to UPPER CASE characters. To override this conversion, or to reinstate it, just use the Control-L command. This conversion is also in effect when you use the Open-Apple-F, Open-Apple-W, or Open-Apple-L find commands to specify the text to find. Even if configured in the PARMS file, this conversion is defeated when the tabs are zeroed.

### Control-N (End of line)

Moves the cursor to the end of the line.

### Control-O (Other characters)

This is used as a special prefix key. For example, if you wanted to type a Control-I or an Escape as part of an ASC string, you would press Control-O, followed by the control character you desire. For multiple control characters, Control-O must be typed before each character is entered.

### Control-R (Restore)

This command restores the original line. For example, if you have used Control-Y to delete all characters to the end of the line, you can press Control-R to undo the effects of the Control-Y command.

### Control-S (Status box)

This command displays a status box showing the number free and used bytes in the current source listing workspace.

### Control-T (Set marker on current line - Vertical "Tab")

This command can be used to set a marker at the current line for recall by the Open-Apple-T command.

### Control-W (Find word)

This command moves the cursor to the beginning of each word in the line (alphanumeric).

### Control-X (Cancel global Exchange, etc.)

This command can be used to cancel any global exchange, text selection or string search while it is in progress.

### Control-Y (Delete to end of line)

Deletes all characters from the cursor to the end of the line.

### Arrow keys (Cursor movement)

The arrow keys move the cursor in the specified direction.

### Delete (Delete character)

Deletes the character to the *left* of the cursor. Also see Control-D.

### Escape (Move to beginning of next line)

This command moves the cursor to the beginning of the next line. This is similar to Return except that Escape does not insert a blank line.

### Return (Insert blank line)

Pressing Return anywhere in the line causes the cursor to move to the beginning of the next line and insert a blank line.

### TAB (Toggle insert cursor)

Toggles the cursor mode between the insert cursor (inverse I) and overstrike cursor (inverse block).

Moving from one line to another has no effect on the status of the cursor; it only changes when toggled with TAB. Also see Control-I.

# MERLIN 16 OPEN APPLE KEY COMMANDS
## (Entire listing oriented)

In addition to the line-oriented commands (control key commands), the Merlin 16 Full Screen Editor uses Open-Apple (Ó) key commands to move within the listing, and to edit entire lines of text. These commands are as follows:

### ÓA (Assemble)

This command passes control to the assembler which attempts to assemble the source file.

### ÓB (Beginning of source)

This command moves to the beginning of the source listing, cursor on line 1.

### ÓC (Copy)

ÓC starts the select mode to copy text. The first time ÓC is pressed, the current line is selected and is shown in inverse. Use the down arrow or Escape keys to extend the selection if desired, or press any other key to cancel the selection. Additional selected lines are shown in inverse. Use the up arrow key to adjust the range selected if you go too far. The select mode will be canceled if you move the cursor above the first selected line or *past the top of the current screen.*

The second time ÓC is pressed the selected text is copied and placed on the clipboard.

If you are unfamiliar with the idea of a clipboard, this is just an analogy to how you might put piece of paper clipped from a magazine, letter, etc. on a clipboard, to hold it temporarily while you were getting ready to put it in its final location. The clipboard just refers to a memory buffer that holds the text you have selected while you decide where you want the final text placed. Using the clipboard, you can cut tet from one source file, load another, and then paste the text into a second file. The clipboard is cleared when a file is assembled.

If text has been selected, you can press the Right arrow key and the selection will be extended to the middle of the next page. This method can be used to quickly select a large section of the listing. Do not hold the Right arrow down continuously or the keyboard buffer will fill and the selection will continue after you release the Right arrow key.

### ÓD (Delete current line)

This command deletes the current line and places it in a special 'undo' buffer which is independent of the clipboard.

The ÓR command replaces the current line with the contents of the 'undo' buffer. Therefore, to move a single line to another location, you could place the cursor on the line to be moved, and then type ÓD to delete the line. Then move the cursor to another line, press Return, ÓI or ÓTab to create an empty line, and press ÓR to replace that line with the deleted line. Also see ÓDelete.

### ÓE (Global Exchange, also called 'Find & Replace')

Sometimes called 'Find & Replace,' this command will let you search for a group of words, and replace them with another. The ÓE command opens a dialog box that asks for the text to change, and the new text to replace it. If you press Return alone for either of these, the command is cancelled.

If you enter the text in both fields and press Return, the file is then searched for the change text. Unlike the FIND command, it looks only for full words. That is, the text found must be bounded by non-alphanumeric characters or it will be ignored. However, if you press and hold the Open-Apple key down when you press the Return key, the exchange will operate on *all* strings found, not just on full words.

If text is found with this method, the screen is reprinted with the replacement made and the cursor is placed on the first character of the replacement. Now you must press a key to continue. Pressing Return, or most any other control character, will defeat the change and the command will look for the next occurrence of the text to change. Pressing the space bar or any other character, except A, will accept the change and the routine will continue.

You can back out of the global exchange while the cursor is on an entry by pressing Control-X. You can also press the A key, which will cause *all* occurrences to be changed.

You can tell when the routine is finished by the fact that during the exchange sequence, the line number at the top right is missing. The line number will return when there are no more matches for the change text, or when you press Control-X.

Normally, the Exchange function will find and replace all occurrences of a word throughout the source listing. This can be modified in several ways.

1. To search only the listing, and ignore words in the comments field, use the TRON command before starting the search and replace. See *Command Box Commands* for details.

2. To search just a portion of a listing, first select just the portion you wish to search as though you were going to copy it, that is, use ÓC and the arrow keys. Then instead of copying the text with another ÓC, use ÓE, ÓF, etc. to search just the highlighted text.

3. Normally, the Exchange function searches for complete words, that is, characters bounded by spaces or other non-alphanumeric characters. Suppose, however, that you wished to replace LDA #LABEL with LDA LABEL. Because this is a search string of two words, a variation is required. To search for a literal character string that may be part of a larger word, hold down the closed Apple or Option key as you press the Return key after you've input the replacement string. Thus, for our example, you could enter:

Change: LDA #L

To: LDA L

### ÓF (Find text)

The ÓF command opens a window which asks for the find text. It then finds the first occurrence of the text in the entire text file. The text can be anywhere on a line. After the first find, you can find the next occurrence by typing another ÓF. You can edit the line and then type ÓF to go to the next occurrence.

If there are more occurrences to be found, one or more plus (+) signs will be shown next to the line number at the top right of the screen. This starts from the line below the current line, and only indicates the number of lines remaining with occurrences, and not the total number of occurrences.

If the ÓF command is used after text has been selected, only the selected text will be searched for the text to be found. When the search has been completed, the text is no longer selected. Thus, you can use the ÓY, ÓC or ÓX commands to search just a portion of your listing. You can also use the TRON command before starting the search to ignore comments during the search.

Control-X, and the ÓB command and Control-S status command both cancel the Find mode, as does failure to find the text below the current line.

The ÓW command is identical to ÓF except that it finds only whole words bounded by non-alphanumeric characters. If you type either ÓW or ÓF to find the next occurrence, this mode will change accordingly.

In all cases the line containing the text is moved to the center of the screen, unless it is within the first 10 lines of the start of the source.

### ÓH (Half screen)

Pressing ÓH toggles the half or split screen mode. In this mode, the bottom ten lines are frozen in a window. A bar is shown above these lines to separate the frozen text from the scroll window. Pressing ÓH will cancel the half screen mode and refresh the screen.

### ÓI (Insert line)

Pressing ÓI will insert a blank line at the cursor. Also see ÓTAB.

### ÓL (Locate label, marker or line number)

This command will locate the first occurrence of a label or any text in the label column. Only the characters typed are compared with the labels. Thus, the search string *LOOP* would jump to the label *LOOP2* if *LOOP* did not occur first. To find a specific label when there may be other similar labels, end the input with a single space.

If a number is entered after this command, the cursor will move to the beginning of the line number specified. This is particularly handy when editing a source file from a printed listing.

The intended use for this command is to move rapidly to a particular place in the source. You can use create your own 'markers' to enhance the capability of this command. Thus, if a line has **7* in it, you can specify **7* as the text to find for this command and it will locate it.

In all cases the line containing the text is moved to the center of the screen, unless it is within the first 10 lines of the start of the source.

### ÓN (End of source)

This command moves the cursor to the end of the source listing.

### ÓO (Open Command Box)

This command opens the Command Box, from which various non-editing commands are issued. See the section on *Command Box Commands* for details. Pressing Return alone will cancel this command.

### ÓQ (Quit to Main menu)

This command quits the Editor and returns to the Main Menu.

### ⌘R (Replace)

This command exchanges the current line with the contents of the 'undo' buffer. Therefore, pressing ⌘R a second time will cancel the effect of the first press.

Using ⌘R when the cursor is on a blank line will place the contents of the 'undo' buffer on the line and place the empty line in the 'undo' buffer.

The ⌘R command can be used to move a single line. Place the cursor at the beginning of the line to be moved and press ⌘R. Move the cursor to the desired location, press Return to insert a blank line, and press ⌘R again.

⌘R can be used by itself to easily interchange two lines. Just place the cursor on the first line, press ⌘R, move the cursor to the second line and press ⌘R again. Then move the cursor back to where the first line was and press ⌘R for the third, and final time. Also see ⌘D.

### ⌘T (Jump to marker line - Vertical "Tab")

This command returns to the line 'remembered' by the last Control-T command.

### ⌘V (Paste)

Pastes the contents of the clipboard at the line containing the cursor. Only full lines are moved. Using this command does not change the contents of the clipboard, so this command can be used to replicate a range of lines.

If the ⌘V paste command is issued when a range of text has been selected, the text on the clipboard will be inserted before the last line of selected text.

### ⌘W (Find word)

The ⌘W command is identical to ⌘F except that it finds only whole words bounded by non-alphanumeric characters. If you type either ⌘W or ⌘F to find the next occurrence, this mode will change accordingly. Either search mode can be cancelled by pressing Control-X.

If the ⌘W command is used after text has been selected, only the selected text will be searched for the word to be found. When the search has been completed, the text is no longer selected.

### ÓX (Cut)

This command is similar to Copy (ÓC), but selected text is removed from the screen after being copied to the clipboard. The first time ÓX is pressed, the current line is selected and is shown in inverse. Use the down arrow or Escape keys to extend the selection if desired, or press any other key to cancel the selection. Additional selected lines are shown in inverse. Use the up arrow key to adjust the range selected if you go too far. The select mode will be canceled if you move the cursor above the first selected line or past the top of the current screen.

The second time ÓX is pressed the selected text is cut and is placed on the clipboard.

One use for this command is to use ÓB to move to the beginning of the listing, then ÓY to select all of the text to the end of the listing, and then ÓX to cut the entire listing and place it on the clipboard. Pressing any other key will cancel the select mode.

If text has been selected, you can press the Right arrow key and the selection will be extended to the middle of the next page. This method can be used to quickly select a large section of the listing. Do not hold the Right arrow down continuously or the keyboard buffer will fill and the selection will continue after you release the Right arrow key.

### ÓY (Select text)

This command selects all text to be cut or copied from the current line to the end of the listing. ÓX or ÓC will cut or copy the text, while pressing any other key will cancel the selection.

This technique can be used to move the entire listing to the clipboard.

### ÓZ (Center screen)

This command repositions the screen so that the line the cursor is on becomes the centered line on the screen.

### ÓDelete (Delete)

This command deletes the line above the cursor and places it in a special 'undo' buffer which is independent of the clipboard.

### ♁Escape (Exit Full Screen Editor]

This command will exit the Full Screen Editor to a line-oriented Command Mode, but only if the controlling parameter in the PARMS file has enabled this function. Unless you have a strong desire to use a Command Mode similar to that for Merlin 8, this should never be needed.

### ♁TAB (Insert line)

Pressing ♁TAB will insert a blank line at the cursor. Also see ♁I.

### ♁Down arrow (Move half-screen down)

Moves the cursor down 10 lines; that line then becomes the center line on the screen. This command has the effect of moving the current line to the top of the screen and then moving the cursor to what was the bottom line on the screen.

### ♁Up arrow (move half-screen up)

Moves the cursor up 10 lines; that line then becomes the center line on the screen. This command has the effect of moving the current line to the bottom of the screen and then moving the cursor to what was the 1st line on the screen.

### ♁Right arrow (Move one page down)

Moves the cursor down 24 lines; that line then becomes the center line on the screen.

### ♁Left arrow (move one page up)

Moves the cursor up 24 lines; that line then becomes the center line on the screen.

### ♁1 (PRTR 1 + Assemble)

This combination command issues a PRTR 1 followed by an ♁A to assemble. Thus ♁1 will activate the printer with the PRTR 1 command and then assemble and print the current listing. The default printer init string from the PARMS file, if any, is used with ♁1 and there will be no page break title unless the source file sets one with the TTL opcode. This command is provided for convenience.

### ⌘2 (PRTR 1 + USER + Assemble)

This combination command issues a PRTR 1 followed by USER and an ⌘A to assemble. ⌘2 will activate the printer with the PRTR 1 command, send the USER command, and then assemble and print the current listing. This command is provided for convenience. See the USER description in the Utility Programs section.

### ⌘3 (PRTR 3 + Assemble)

This combination command issues a PRTR 3 followed by an ⌘A to assemble. Thus ⌘3 will activate the screen with the PRTR 3 command and then assemble and print the current listing to the screen instread of the printer. This command is provided for convenience.

### ⌘4 (PRTR 3 + USER + Assemble)

This combination command issues a PRTR 3 followed by an USER and an ⌘A to assemble. ⌘4 will activate the screen with the PRTR 3 command, send the USER command, and then assemble and print the current listing to the screen instead of the printer. This command is provided for convenience. See the USER description in the Utility Programs section).

### ⌘6 (LINK for LINKER.GS only)

This command is the equivalent of typing LINK in the Command Box. ⌘6 can be used with the LINKER.GS only. This command is provided for convenience.

### ⌘8 (Asterisk)

Produces a line of 32 asterisks. Overstrikes existing line, if any. Undo with the Control-R command.

### ⌘9 (Box)

Produces an asterisk, 30 spaces, and then another asterisk. This and the ⌘8 command can be used to produce a large box for titles and other information. Overstrikes existing line, if any. Undo with the Control-R command.

### ⌥- (Hyphen)

Produces a line of 1 asterisk and 31 hyphens. Overstrikes existing line, if any. Undo with the Control-R command.

### ⌥= (Equal sign)

Produces a line of 1 asterisk and 31 equal signs. Overstrikes existing line, if any. Undo with the Control-R command.

### [Closed Apple]

This command speeds up cursor movement and keyboard input. For example, pressing and holding down the down arrow key will scroll the listing. If you press and hold down the Closed Apple key while you press the up or down arrow key, the scroll rate will be increased.

# MERLIN 16 COMMAND BOX

## GENERAL GUIDELINES FOR THE COMMAND BOX COMMANDS

The Command Box is used for various commands not directly related to editing the current listing. To open the Command Box from the Full Screen Editor, you would press ÓO, and then type in the desired command. For most of the Merlin 16 Command Box operations, only the first letter of the command is required, the rest being optional. This manual will show the required command characters in UPPER case and the optional ones in lower case.

## ABOUT THE COMMAND BOX DOCUMENTATION

For each of the commands available from the Merlin 16 Command Box, the documentation consists of three basic parts:

1) the name and syntax of the command
2) examples of the use of each available syntax
3) a description of the function of each command

When the syntax for each command is given:

PARENTHESES () indicate a required value
ANGLE BRACKETS <> indicate an optional value or character
SQUARE BRACKETS [] are used to enclose comments about the commmand

### Delimited Strings (or d-strings)

Several commands allow specification of a string. The string must be delimited by a non-numeric character other than the slash or comma. Such a string is called a delimited or d-string. The usual delimiter is single or double quote marks (' or "). Delimited string examples:

'this is a delimited string'
"this is a delimited string"
@this is another d-string@

Note that the slash '/' cannot be used as a delimiter since it is the character that delimits range lists in line number-related commands.

### Wild Card Characters in Delimited Strings

For all of the commands that use delimited strings (d-strings), the '^' character acts as a wild card character. For example, the d-string 'Jon^s' is equivalent to both 'Jones' and 'Jonas' d-strings.

### Upper and Lower Case Control

The shift and caps lock keys work as you would expect. While editing or entering a line of text, you can also use the Control-L command, described earlier in this section.

### Line Numbers in Command Box

With some commands, you must specify a line number, a range of line numbers, or a range list. A line number is just a number. A range is a pair of line numbers separated by a comma. A range list consists of several ranges separated by a slash ('/').

Line Number examples:

| | | |
|---|---|---|
| 10 | LINE # | [ a single line number ] |
| 10,30 | RANGE | [ the range of lines 10 to 30 ] |
| 10,30/50,60 | RANGE LIST | [ ranges 10 to 30 AND 50 to 60] |

If a line number in a range exceeds the number of the last line in the source, the editor automatically adjusts the specified line to the last line number. For example, if you wanted to List all the lines past 100 in a source listing, entering L100,9999 would do it.

# COMMAND BOX COMMANDS

Following are the commands recognized by Merlin 16 in the Command Box of the Full Screen Editor. The Command Box is opened by pressing ÓO from within the Full Screen Editor. Pressing Return alone will cancel and close the Command Box. Pressing Return after entering one of the following commands will execute the command.

NOTE: UPPER case characters shown in command are required, lower case characters are optional and are listed for purposes of clarity only.

### A (Add)

| | |
|---|---|
| A | [ only option for this command ] |

The Add command places you at the end of the existing source listing, if any, and is equivalent to the ÓN command.

### ASM (Assemble)

| | |
|---|---|
| ASM | [ only option for this command ] |

This passes control to the assembler, which then assembles the source file.

Assembly may be terminated at any point by pressing Control-C or Escape.

During the second pass of assembly, pressing Control-D will toggle the list flag, so that the listing will either stop or resume. The next LST opcode in the source overrides this, but another Control-D can be used again.

### FIX

| | |
|---|---|
| FIX | [ undoes the TEXT command ] |
| FIXS | [ same but removes multiple spaces except in comments and ASCII strings ] |

This undoes the effect of the TEXT command. It also does a number of technical housekeeping chores. It is recommended that FIX be used on all source files from external sources that are being converted to Merlin 16 source files, after which the file should be saved.

NOTE: FIX will truncate any lines longer than 255 characters.

## GET

GET ( obj adrs )

| | |
|---|---|
| GET | [ put object code in Main Memory at the address specified in the source's ORG ] |
| GET $4000 | [ put object code at location $4000 in Main Memory ] |

This command is used to move the object code, after an assembly, from its location in Auxiliary Memory to its ORG location in Main Memory. The address must be above the existing source file, if any, and it will not be allowed to overwrite DOS. You can do a NEW if you want to load it lower in memory than allowed, but you must remember to save the source first. You cannot use GET to put object code at memory locations lower than $901, but you can go to the Monitor afterwards and use it to move the object to any desired location. However, any such move using the Monitor may destroy your source or other data necessary to the assembler's operation. Caution should be used.

The GET command does not check if a valid object code has been assembled.

This command is supplied for convenience only. The recommended method for testing a program is to save the source code first, save the object code, and then run the program from BASIC or with the G command from the Monitor.

## HEX-DEC CONVERSION

128 = $0080
$80 = 128

If you type a positive or negative decimal number in the Command Box, the hex equivalent is returned. If you type a hex number using the $ prefix, the decimal equivalent is returned. All commands accept hex numbers.

## L (List)

List
List (line number)
List (range)
List (range list)

| | |
|---|---|
| L | [ list entire file ] |
| L 20 | [ list line 20 only ] |
| L 20,30 | [ list 20 through 30 ] |
| L 20,30/40,42 | [ list 20 through 30 and then list lines 40 through 42 ] |

Lists the source file with line numbers. Control characters in source are shown in inverse, unless the listing is being sent to a printer or other nonstandard output device.

The listing can be aborted by Control-C or with the slash (/) key. You may stop the listing by pressing the space bar and then advance a line at a time by pressing the space bar again. By holding down the space bar, the auto-repeat feature of the Apple will result in a slower listing. Any other key will resume the normal speed. This space bar technique also works during assembly and the symbol table printout. Any other key will restart it.

### MON (Monitor)

MON [ only option with this command ]

This exits to the Monitor. You may return to the Merlin 16 Main Menu by pressing Control-C, Control-B, or Control-Y. These commands re-establish important zero page pointers from a safe area inside Merlin 16. Thus Control-Y will give a correct entry even if you have damaged the zero page pointers while in the Monitor. DOS is not connected while using this entry to the Monitor.

You may return to the editor directly by typing 0G and pressing Return, but unlike the above commands, this uses the zero page pointers stored at $0A-$0F. Therefore, you must be sure that these pointers have not been altered. For normal usage, any of the three Control commands should be used to return to Merlin 16.

NOTE: When you exit to the Monitor with the MON command, the RAM-based $D000-$FFFF memory is enabled, and therefore, Merlin 16 and its symbol table if any. If you want to examine the ROM memory that would normally correspond to Applesoft and the F8 Monitor, you should quit Merlin 16 with the Main Menu Quit command, and enter the Monitor with Call -151. This procedure necessitates loading BASIC.SYSTEM which removes Merlin 16 from memory.

### NEW

NEW [ only option for this command ]

Deletes the present source file in memory. Also see UNNEW.

### P (Print without line numbers)

Print
Print (line number)
Print (range)
Print (range list)

| | |
|---|---|
| P | [ print entire file ] |
| P50 | [ print line 50 only ] |
| P50,100 | [ print lines 50 through 100 ] |
| P1,10/20,30 | [ print 1 through 10 and then print lines 20 through 30 ] |

This is the same as LIST except that line numbers are not added. See PRTR for formatted printouts.

### PRTR (Formatted printout)

PRTR (command)

| | |
|---|---|
| PRTR 1 | [ activate printer in slot 1 with no printer init string ] |
| PRTR 1 ""Page Title" | [ printer in slot 1, no printer init string, "Page Title" is the page header ] |
| PRTR 1 "<Control-I>80N" | [ as above, add Control-I80N to initialize the printer ] |
| PRTR 3 | [ send formatted listing to screen ] |

This command is for sending a listing to a printer with page headers and provision for page boundary skips. See the section on Configuration for details on setting up default parameters, also TTL in the Assembler section. The entire syntax of this command is:

PRTR # "(string)"<page header>"

If the page header is omitted, the header will consist of page numbers only.

*The initialization string may not be omitted if a page header is to be used.* If no special string is required, use a null string of two quotes (""), as in the example showing "Page Title" in which case a carriage return will be used. No output is sent to the printer until a LIST, PRINT, or ASM command is issued. See Control-O for information on inserting Control characters in the printer init string. The PRTR command only affects the next output command, and is canceled at the end of the listing.

### Q (Quit)

| | |
|---|---|
| Q | [ only option for this command ] |

Exits to Main Menu.

### SYM (Symbol table)

| | |
|---|---|
| SYM | [ print symbol table from last assembly ] |
| SYM'TR' | [ print only those labels starting with 'string' specified ] |
| SYM: | [ print local labels in symbol table ] |
| SYM:'TR' | [ print local labels with 'string' specified ] |

The SYM command will print the symbol table from the last assembly to be printed. Care should be taken to ensure that a valid symbol table exists before this command is used.

The SYM: command inverts the *print locals flag* and then prints the symbol table with local labels first. Thus, it is not necessary to change this bit in the PARMS file to see the local label list.

If a string is specified with either SYM and SYM:, only those labels starting with the string will be printed

NOTE: If the printing is aborted and the SYM command is reissued, only a partial listing will result. This is because symbol flagging is done during the sorting of the table. However, if the printing is allowed to continue to the end, the entire symbol table will be printed the next time SYM or SYM: is used. If aborted during an alphabetical printout, the SYM command will restart where it left off.

### TABS

TABS <number><, number><,...> <'tab character'>

| | |
|---|---|
| TABS | [ clear all tabs ] |
| TABS 10,20 | [ set tabs to 10 & 20 ] |
| TABS 10,20 " " | [ as above, space is tab character ] |

This sets up to 9 tabs for the editor, and has no effect on the assembler listing. The default tab character is a space, but any may be specified. Space is the only acceptable tab character for the separation of labels, opcodes, and operands. If you don't specify the tab character, then the last one used remains. Entering TABS and a Return will set all tabs to zero.

### TEXT

| | |
|---|---|
| TEXT | [ only option for this command ] |

This converts all spaces in a source file to inverse spaces. The purpose of this is for use on word processing type text files so that it is not necessary to remember to zero the tabs before printing such a file. This conversion has no effect on anything except the Editor's tabulation. The command FIX undoes the effect of the TEXT command.

### TROF (Truncate Off)

TROF [ only option for this command ]

This command returns to the default condition of the truncation flag which also happens automatically upon entry to the editor from the Main Menu or from the Assembler. All source lines when listed or printed will appear normal.

### TRON (Truncate On)

TRON [ only option for this command ]

This command sets a flag which, during LIST or PRINT, will suppress printing of comments that follow a semicolon. It makes reading of some source files easier. Typing TRON before using ÓE or ÓF also causes the search to ignore text within comments.

### UNNEW

UNNEW [ only option for this command ]

This command restores a text file cleared by a NEW command. Note that the first two characters of the source file cannot be restored, so some additional editing will be necessary. Also see NEW.

### USER

USER
USER 1 [ example for use with XREF ]
USER 0: FILENAME [ example for use with PRINTFILER ]

This does a JSR to the routine at $3F5. This is the location of the Applesoft ampersand vector which normally points to a RTS. USER is designed to connect the various utilities supplied with Merlin 16 and for user defined printer drivers. You must be careful that your printer driver does not use zero page addresses, with the exception of the I/O pointers and $60-$6F and $90-$9F, because this is likely to interefere with Merlin 16's heavy usage of zero page. Several supplied utilities operate through the USER command. Specifically, XREF is a USER type utility. Also see the description of USER files supplied with Merlin 8/16 as described in the Utilities section of this manual.

**VAL**

VAL "expression"

| | |
|---|---|
| VAL "LABEL" | [ gives the address (or value) of LABEL for the last assembly done or "unknown label" if not found. ] |
| VAL "$1000/2" | [ returns $0800 ] |

This will return the value of the expression as the assembler would compute it (or was used in the last assembly). All forms of label and literal expressions valid for the assembler are valid for this command.

**VID (VIDeo)**

VID (slot number)

| | |
|---|---|
| VID 3 | [ turns on 80 column display ] |

This command selects an 80 column display device. To turn off the display, use Escape Ctrl-Q for the Apple IIe, IIc or IIgs video; use Escape-0 for the Videx UltraTerm. VID 3 is required to re-activate the display if you have set the PARMS file to switch to 40 columns for the PRTR command.

**W (Where)**

Where (line number)

| | |
|---|---|
| W 50 | [ where is line 50 in memory ] |
| W 0 | [ where is end of source file ] |

This prints in hex the location in memory of the start of the specified line. *Where 0* or *W0* will give the location of the end of source.

**. [period]**

| | |
|---|---|
| . | [ only option for this command ] |

Lists starting from the beginning of the last specified range. For example, if you type *L10,100*, lines 10 to 100 will be listed. If you then use the period (.) command, the listing will start again at 10 and continue until stopped. The end of the range is not remembered.

**/ (slash)**

/ <line number>

| | |
|---|---|
| / | [ start to list at last line listed ] |
| /50 | [ start listing at line 50 ] |

This command continues the listing from the last line number listed, or, when a line number is specified, from that line. This listing continues to the end of the file or until it is stopped as in LIST.

# MERLIN 16 EDITOR COMMAND SUMMARY

## CONTROL KEY COMMANDS (line oriented)

The Control Key commands consist of cursor moves and line oriented commands.

Control-B ..... Moves cursor to beginning of line
Control-C ..... Cancel assembly and return to Editor at last line assembled
Control-D ..... Deletes character under the cursor
Control-F ..... Finds next occurrence on current line of next character typed
Control-I ..... Toggles insert and overstrike cursor
Control-L ..... Toggles lower case conversion
Control-N ..... Moves cursor to end of line
Control-O ..... Prefix key for typing optional characters
Control-R ..... Restores original line
Control-S ..... Displays status box
Control-T ..... Remember current line for recall by ÓT
Control-W..... Finds next occurrence of word in line
Control-X..... Exits global exchange, etc. while in progress
Control-Y ..... Delete characters to end of line
Arrows ..... Moves the cursor in the specified direction
Delete ..... Deletes character to left of cursor
Escape ..... Moves cursor to beginning of next line
Return ..... Moves cursor down and inserts blank line
TAB ..... Toggles insert and overstrike cursor

## OPEN-APPLE KEY COMMANDS (entire listing oriented)

The Open-Apple Key commands are global commands, which means they are generally oriented to the whole listing as opposed to just the current line (or a single character).

| | |
|---|---|
| ÓA | Assembles the current source listing |
| ÓB | Moves cursor to beginning of listing |
| ÓC | Start text selection/Copy selected text to clipboard |
| ÓD | Deletes line and places it in 'undo' buffer |
| ÓE | Global exchange (Find & Replace) |
| ÓF | Finds next occurrence of text entered |
| ÓH | Toggles half or split screen mode |
| ÓI | Inserts blank line at cursor |
| ÓL | Finds first occurrence of label or line |
| ÓN | Moves cursor to end of listing |
| ÓO | Opens Command Box |
| ÓQ | Quits Editor and returns to Main Menu |
| ÓR | Swaps current line with 'undo' buffer |
| ÓT | Goes to line of last Control-T |
| ÓV | Pastes contents of clipboard on current line |
| ÓW | Finds next occurrence of whole word |
| ÓX | Start text selection/Cut selected text to clipboard |
| ÓY | Selects text from current line to end of file |
| ÓZ | Current line becomes eleventh line on screen |
| ÓDelete | Deletes line above cursor; puts in 'undo' buffer |
| ÓTAB | Inserts a blank line at cursor |
| ÓDown | Moves cursor down 10 lines |
| ÓUp | Moves cursor up 10 lines |
| Ó1 | Issues 'PRTR 1' + 'Assemble' |
| Ó2 | Issues 'PRTR 1' + 'USER' + 'Assemble' |
| Ó3 | Issues 'PRTR 3' + 'Assemble' |
| Ó4 | Issues 'PRTR 3' + 'USER' + 'Assemble' |
| Ó6 | Issues 'LINK' command to LINKER.GS |
| Ó8 | Produces a line of 32 asterisks |
| Ó9 | Produces 1 asterisk, 30 spaces, and 1 asterisk |
| Ó- | Produces a line of 1 asterisk followed by 31 hyphens |
| Ó= | Produces a line of 1 asterisk followed by 31 equal signs |
| ◆ | Increases speed of cursor movement and keyboard input |

## COMMAND BOX COMMANDS

The Command Box commands are a series of commands not directly related to editing the listing. These commands are always preceded by an Open-Apple-O [O] which opens the Command Box. After entering the desired command, press Return. UPPER case characters are required, lower case characters are optional and are listed for purposes of clarity.

| | |
|---|---|
| Add | ..... Moves to the end of the current listing |
| ASM | ..... Assembles the current source file |
| FIX | ..... Undoes the effect of the TEXT command |
| GET | ..... Place object code in Main Memory at specified address |
| Link | ..... Assemble and link last source file worked on. |
| List | ..... Lists line or range of lines |
| MON | ..... Exits Editor and enters Monitor |
| NEW | ..... Clears current source file from memory |
| Print | ..... Prints listing without formatting |
| PRTR | ..... Prints formatted listing. See command for complete syntax |
| Quit | ..... Exits to Main Menu |
| SYM | ..... Prints symbol table of last assembly |
| TABS | ..... Sets tabs used by the Editor |
| TEXT | ..... Zeroes all tabs in text file |
| TROF | ..... Turns off truncation of comments field in printout |
| TRON | ..... Turns on truncation of comments field in printout |
| UNNEW | ..... Restores text file deleted by the NEW command |
| USER | ..... Connects various utilities and user defined printer drivers |
| VAL | ..... Returns value of expression or address of label |
| Where | ..... Prints memory location of specified line |
| . (period) | ..... Lists from beginning of last specified range |
| / (slash) | ..... Continues listing from last specified line |

## CUSTOMIZING MERLIN 16 COMMANDS

Almost all of the Merlin 16 Editor command keys can be changed if you have a preference for an alternate definition. See the description of the PARMS file for Merlin 16 in the Technical Information section of this manual.

## GENERAL REMARKS ON MERLIN 16 EDITOR

When you move the cursor between lines, its horizontal position may vary. This is because the cursor position is based on the actual position in memory in the line, and not on the screen position. If the tabs are zeroed you will not notice this, except for the fact that the cursor is never beyond the last character in the line.

The maximum line length is actually 192 but you can only edit the first 80 characters. *Lines longer than 80 characters will be truncated if they are edited.* This can be important if you are using TXTED as mentioned in the Utilities section.

An assembly will delete the contents of the clipboard.

### EDMAC, TXTED, and AUTO.EDIT

A macro program called EDMAC is also included on the Merlin 16 disk, which will automatically generate phrases like "LDA #$" in your source file with a single key ♦ command. TXTED is a modified full screen editor that breaks lines when you press Return, and can delete Returns as well. AUTO.EDIT is an automated text editor that can speed up making the same change in a number of places in a source file where the Find or Exchange commands would not be appropriate. See the description of these utilities in the Utilities section of this manual for more ionformation.

### OOPS

Virtually any editor action can be undone. You should remember that the proper undo command is of the same 'type' as the command you want to undo. Thus, any Control key command is undone by Control-R. This includes the ♦8, ♦9, ♦-, and ♦= commands which are considered line oriented commands for this purpose.

The line deletion commands ♦D and ♦Delete are undone by creating an empty line with ♦Tab followed by ♦R. If you forget to create the empty line, type another ♦R and then insert the empty line to receive the undo buffer contents.

The ♦R command undoes itself.

A Cut (♦X) is undone by a Paste (♦V) without moving the cursor off its line.

If you are entering a line of text in response to a prompt, such as a filename, PRTR initialization, or dialog box, you can press Control-C or Control-X to cancel the line.

## ASSEMBLING A MERLIN 16 FILE

Once you have entered and edited your source listing, you will want to assemble it. Just press Open-Apple-A. This passes control to the assembler, which attempts to assemble the source file.

If you wish to have a formatted printed listing of an assembly, just use the PRTR command immediately before using the Open-Apple-A command.

Pressing Control-C terminates the assembly and returns to the Editor with the cursor on the line where assembly was interrupted.

# THE ASSEMBLER

In Merlin 8/16, the Editor is used to create and edit the source listing from which the final program, or object code, will be assembled. The Assembler is that part of Merlin 8/16 which actually interprets your source code to create the final program.

The Assembler portion of Merlin 8/16 is distinct only in concept. In practice, both the Editor and Assembler are resident in the machine at all times, and thus both are available without having to be aware of which is in operation at any given time. This is in contrast to many other assemblers, in which the Editor and Assembler are completely separate programs, necessitating the switching between them by loading and running independent programs, and often requiring that you save the source file to disk before an assembly can even be done.

This section of the documentation explains the syntax of those commands, or directives, that can be used in the source listing itself, and which direct Merlin 8/16 to perform some function while assembling the object code. These are in contrast to the Editor commands which are used primarily to edit an existing line of text.

For example, in the simplest assembler possible, only commands like LDA, JSR, etc. would be recognized by the assembler. However, the first time you want to create a data table, an instruction is required by the assembler which will define one or more bytes that are a pure number value, as opposed to specific opcodes. This is addressed in virtually all assemblers by creating the assembler directive, or pseudo opcode HEX.

Thus the assembler can create a byte of data like this:

```
1 LABEL   HEX F7    ; STORES BYTE '$F7'
```

Now, suppose the data you wanted to store was an ASCII character string. With only the HEX directive, you'd have to look up all the ASCII character equivalents, and encode them in your program with individual HEX statements.

Wouldn't it be nice, though, if the assembler itself had a larger repetoire of new commands or directives that included ones for defining character strings? You bet! And Merlin 8/16 has a lot of them.

The simplest is ASC, and a typical line would look like this:

```
1  LABEL  ASC 'THIS IS A TEST' ; STORE ENTIRE CHARACTER STRING
```

When assembled, Merlin 8/16 would automatically look up the ASCII character equivalents, and store the bytes in memory at wherever that statement occurred in your program. Along with the Editor, the variety and power of assembler directives is the other biggest factor in determining the power of a given assembler. Merlin 8/16 is outstanding in this area with a wide complement of directives for every occasion.

This section of the documentation will explain the syntax to use in your source files for each directive, and document the features that are available to you in the assembler.

## ABOUT THE ASSEMBLER DOCUMENTATION

The assembler documentation is broken into three main sections:

1) Preliminary Definitions
2) Assembler Syntax Conventions
3) Assembler Pseudo Opcode Descriptions

The last two sections are each broken down further into the following:

Assembler Syntax Conventions:
1) Number Format
2) Source Code Format
3) Expressions Allowed by the Assembler
4) Immediate Data Syntax
5) Addressing Modes
6) Sweet 16 Opcodes
7) Native vs. Emulation mode in Merlin 16

Assembler Pseudo Opcode Descriptions:
1) Assembler Directives
2) Formatting Pseudo Ops
3) String Data Pseudo Ops
4) Data and Storage Allocation Pseudo Ops
5) Miscellaneous Pseudo Ops
6) Conditional Pseudo Ops
7) Pseudo Ops for Macros
8) Variables

The Assembler Syntax Conventions illustrate the syntax of a line of assembly code, the proper method to specify numbers and data, how to construct assembler expressions and the proper syntax to use to specify the different addressing modes allowed by the 65xx microprocessors. This section should be understood prior to using the assembler, otherwise it is will be difficult to determine the acceptable methods to construct a proper expression as the operand for a pseudo op.

The Assembler Pseudo Opcode Descriptions illustrate the functions of the many Merlin 8/16 pseudo ops, the correct syntax to use and examples of each pseudo op's use.

## PRELIMINARY DEFINITIONS

The type of operand for almost all of Merlin 8/16's pseudo ops and the 65xx microprocessors can be grouped into one of four categories:

1) Expressions
2) Delimited Strings (d-strings)
3) Data
4) Filenames or Pathnames

### Expressions

Expressions are defined in the Assembler Syntax Conventions section of this Chapter.

### Delimited Strings

Delimited Strings are defined in the Editor section of the manual, but that definition is repeated here for continuity.

Several of the pseudo opcodes, and some of the 65xx opcodes such as LDA, allow their operand to be a string. Any such string must be delimited by a non-numeric character other than the slash (/) or comma (,). Such a string is called a "d-string" or delimited string. The usual delimiter is a single or double quote mark ( " or ').

Examples:
"this is a d-string"
'this is another d-string"
@another one@
Zthis is one delimited by an upper case zZ
"A"
'A'

NOTE: Delimited strings that are used as the object of *any* 65xx opcode *must* be enclosed in single or double quotes. If not, the assembler will interpret the d-string to be a label, expression or data instead.

Take special note that some of the pseudo ops as well as the 6502 and 65C02 opcodes use the delimiter to determine the *hi-bit* condition of the resultant string. In such cases the delimiter should be restricted to the single or double quote.

### Data

Data is defined as raw hexadecimal data composed of the digits 0 through 9 and the letters A through F.

### Filenames (DOS 3.3 only)

Filenames are defined as the name of a DOS 3.3 file without any delimiters, e.g. no quotes surrounding the name. Source file names are suffixed with a .S while a T. is used as the prefix for Text files, USE files and PUT files. The applicable suffix or prefix *should not* be used as part of the filename when loading, saving, reading or writing.

### Pathnames (ProDOS only)

Pathnames are defined as ProDOS pathnames and are restricted to the definition of pathnames as described in the ProDOS User's Manual. Pathnames as used by Merlin 8/16 do not have delimiters, e.g. no quotes surrounding the pathname. The .S suffix is used for source, USE, and PUT pathnames. This suffix *should not* be used as part of the pathname when loading or saving.

## ASSEMBLER SYNTAX CONVENTIONS

### SOURCE CODE FORMAT

#### Syntax of a Source Code Line

A line of source code typically looks like:

```
    LABEL    OPCODE  OPERAND        ;COMMENT
```

and a few real examples:

```
1  START      LDA  #50              ;THIS IS A COMMENT
2  *    THIS IS A COMMENT ONLY LINE
3                                   ;TABBED BY EDITOR
```

A line containing only a comment can begin with an asterisk (*) as in line 2 above. Comment lines starting with a colon (;), however, are accepted and tabbed to the comment field as in 3 above. The assembler will accept an empty line in the source code and will treat it just as a SKP 1 instruction, except that the line number will be printed. See the section on pseudo opcodes for details.

The number of spaces separating the fields is not important, except for the editor's listing, which expects just one space.

#### Source Code Label Conventions

The maximum allowable LABEL length is 26 characters in Merlin 16 or 13 characters in Merlin 8, but more than 8 will produce messy assembly listings unless you change the tab settings. A label must begin with a character at least as large, in ASCII value, as the colon, and may not contain any characters less, in ASCII value, than the number zero. Note that periods (.) are not allowed in labels since the period is used to specify the logical OR in expressions.

A line may contain a label by itself. This is equivalent to equating the label to the current value of the address counter.

### Source Opcode and Pseudo Opcode Conventions

The assembler examines only the first 3 characters of the OPCODE, with certain exceptions such as macro calls and the DEND opcode. For example, you can use PAGE instead of PAG. However, because of the exception, the fourth letter should not be a D. The assembler listing will not be aligned with an opcode longer than five characters unless there is no operand or you change the tab settings.

### Operand and Comment Length Conventions

The maximum allowable combined OPERAND + COMMENT length is 64 characters. You will get an OPERAND TOO LONG error if you use more than this. A comment line by itself is also limited to 64 characters.

## LOCAL LABELS

A local label is any label beginning with a colon (:). A local label is attached to the last global label and can be referred to by any line from that global label to the next global label. You can then use the same local label in other segments governed by other global labels. You can choose to use a meaningless type of local label such as :1, :2, etc., or you can use meaningful names such as :LOOP, :EXIT, and so on.

Example of local labels:

```
1   START  LDY #0
2          LDX #0
3   :LOOP  LDA (JUNK),Y        ;:loop is local to start
4          STA (JUNKDEST),Y
5          INY
6          CPY #100
7          BNE :LOOP           ;branch back to :LOOP in 3
8   LOOP2  LDY #0
9   :LOOP  LDA (STUFF),Y       ;:loop is now local to loop2
10         STA (STUFFDEST),Y
11         INY
12         CPY #100
13         BNE :LOOP           ;branch back to :LOOP in 9
14         RTS
```

Some restrictions on use of local labels are:

1) Local labels cannot be used inside macros.
2) You cannot label a MAC, ENT or EXT with a local label and you cannot EQUate a local label.
3) The first label in a program cannot be a local label.

### Local Labels, Global Labels and Variables

There are three distinct types of labels used by the assembler. Each of these are identified and treated differently by Merlin.

| | |
|---|---|
| Global Labels | : labels not starting with "]" or ":" |
| Local labels | : labels beginning with ":" |
| Variables | : labels beginning with "]" |

Note that local labels do not save space in the symbol table, while variables do. Local labels can be used for forward and backward branching, while variables cannot. Good programming practice dictates the use of local labels as branch points, variables for passing data, etc.

## VARIABLES

Labels beginning with a right bracket (]) are regarded as variables. They can be redefined as often as you wish. The designed purpose of variables is for use in macros, but they are not confined to that use.

Forward reference to a variable is impossible, that is, with correct results, but the assembler will assign some value to it. Therefore, a variable should be defined before it is used.

It is possible to use variables for backwards branching, using the same label at numerous places in the source. This simplifies label naming for large programs and uses much less space than the equivalent once-used labels.

For example:

```
 1          LDY #0
 2 ]JLOOP   LDA TABLE,Y
 3          BEQ NOGOOD
 4          JSR DOIT
 5          INY
 6          BNE ]JLOOP       ;BRANCH TO LINE 2
 7 NOGOOD   LDX #-1
 8 ]JLOOP   INX
 9          STA DATA,X
10          LDA TBL2,X
11          BNE ]JLOOP       ;BRANCH TO LINE 8
```

## NUMBER FORMAT

The assembler accepts decimal, hexadecimal, and binary numerical data. Hex numbers must be preceded by the dollar sign ($) and binary numbers by the per cent sign (%), thus the following four numbers are all equivalent:

| Dec | Hex | Binary | Binary |
|---|---|---|---|
| 100 | $64 | %1100100 | %01100100 |

as indicated by the last binary number, leading zeros are ignored.

### Immediate Data vs. Addresses

In order to instruct the assembler to interpret a number as immediate data as opposed to an address, the number should be prefixed with a pound sign (#). The # here stands for number or data. For example:

```
LDA #100        LDA #$64       LDA #%1100100
```

These three instructions will all load the accumulator with the number 100, decimal.

A number not preceded by # is interpreted as an address. Therefore:

```
LDA 1000        LDA $3E8       LDA %1111101000
```

are equivalent ways of loading the accumulator with the byte that resides in memory location $3E8.

### Use of Decimal, Hexadecimal or Binary Numbers

We recommend that you use the number format that is appropriate for clarity. For example, the data table:

```
DA    $1
DA    $A
DA    $64
DA    $3E8
DA    $2710
```

is a good deal more mysterious than its decimal equivalent:

```
DA    1
DA    10
DA    100
DA    1000
DA    10000
```

Similarly,

```
ORA #$80
```

is less informative than

```
ORA #%10000000
```

which sets the hi-bit of the number in the accumulator.

## EXPRESSIONS ALLOWED BY THE ASSEMBLER

### Primitive Expressions

Expressions are built up from "primitive expressions" by use of arithmetic and logical operations. The primitive expressions are:

1. A label.
2. A number (either decimal, $hex, or %binary).
3. Any ASCII character preceded or enclosed by quotes or single quotes.
4. The asterisk character (*) which stands for the current address.

All number formats accept 16-bit data and leading zeros are never required. In case 3, the "value" of the primitive expression is just the ASCII value of the character. The hi-bit will be on if a quote (") is used and the value greater than $7F. The hi-bit will be off if an apostrophe (') is used and the value less than $80.

### Arithmetic and Logical Operations in Expressions

The assembler supports the four arithmetic operations: +, -, / (integer division), and * (multiplication). It also supports the three logical operations: ! (Exclusive OR), . (OR), and & (AND).

### Building Expressions

Expressions are built using the primitive expressions defined above, either with or without arithmetic and/or logical operations. This means that expressions can take the form of primitives or primitives operated on by other primitives using the arithmetic and logical operators.

Some examples of legal expressions are:

| | |
|---|---|
| #01 | (primitive expression = 1) |
| #$20 | (primitive expression = 32 dec) |
| LABEL | (primitive consisting of a label) |
| #"A" | (primitive consisting of letter "A") |
| * | (primitive = current value of PC) |

The following are examples of more complex expressions

| | |
|---|---|
| LABEL1-LABEL2 | (LABEL1 minus LABEL2) |
| 2*LABEL+$231 | (2 times LABEL plus hex 231) |
| 1234+%10111 | (1234 plus binary 10111) |
| "K"-"A"+1 | (ASCII "K" minus ASCII "A" plus 1) |
| "0"!LABEL | (ASCII "0" EOR LABEL) |
| LABEL&$7F | (LABEL AND hex 7F) |
| *-2 | (current address minus 2) |
| LABEL.%10000000 | (LABEL OR binary 10000000) |

By clever use of the simple arithmetic and logical operators in carefully designed expressions, you can create other functions that may not be immediately obvious.

For example, for a conditional assembly, you might want to see if one label had a value greater than that of another. Although Merlin 8/16 doesn't have a specific < or > function, you can still do the equivalent test using the division operator:

```
DO  LABEL1/LABEL2       ; 0 IF LABEL1 < LABEL2 = DON'T DO
                        ; 1 IF LABEL1 >= LABEL2 = DO
DO  LABEL1/5            ; DO IF LABEL1 >= 5
DO  LABEL1-1/-1         ; DO IF LABEL1 = 0
DO  5-1/LABEL1          ; DO IF LABEL1 < 5
DO  LABEL1/6            ; DO IF LABEL1 > 5
```

As another example, use of the AND operator (&) makes it simple to define a control character:

```
CHK  LDA  CHAR              ; INPUT CHARACTER
     CMP  #$9F&"A"          ; CONTROL-A
     BNE  NEXT
```

$9F AND $C1 ("A") = $81 ("Control-A)"

Checking for equality is a simple subtraction:

```
DO  LABEL1-5-1/-1       ; DO ONLY IF LABEL1 = 5
```

### Parentheses and Precedence in Expressions

Parentheses are not normally allowed in expressions. They are not used to modify the precedence of expression evaluation. All arithmetic and logical operations are evaluated left to right. Thus, 2+3*5 would assemble as 25 and not 17.

Parentheses are used to retrieve a value from the memory location specified by the value of the expression within the parentheses, much like indirect addressing. This use is restricted to certain pseudo ops, however.

For example:

```
DO  ($300)
```

will instruct the assembler to generate code if the value of memory location $300 is non-zero at the time of assembly.

### Example of Use of Assembler Expressions

The ability of the assembler to evaluate expressions such as LAB2-LAB1-1 is very useful for the following type of code:

```
COMPARE     LDX    #EODATA-DATA-1
LOOP        CMP    DATA,X
            BEQ    FOUND       ;found
            DEX
            BPL    LOOP
            JMP    REJECT      ;not found
DATA        HEX    CACFC5D9
EODATA      EQU    *
```

With this type of code, you can add or delete some of the DATA and the value which is loaded into the X index for the comparison loop will be automatically adjusted.

## IMMEDIATE DATA SYNTAX

For those opcodes such as LDA, CMP, etc., which accept immediate data, i.e. numbers as opposed to addresses, the immediate mode is signed by preceding the expression with #. An example is LDX #3. When programming the 65802 or 65816, the interpretation of an immediate data expression may depend on the assembler status of the M and X bits at that point in the assembly. In general:

| | |
|---|---|
| #expression | produces the low byte of the expression in 8 bit mode |
| #expression | produces the low word of the expression in 16 bit mode |
| #<expression | produces the low byte of the expression in 8 bit mode |
| #<expression | produces the low word of the expression in 16 bit mode |
| #>expression | produces the high byte of the expression in 8 bit mode |
| #>expression | produces the high word of the expression in 16 bit mode |
| #^expression | produces the bank byte of the expression always. |

## ADDRESSING MODES

Because of the different processors in use in the Apple IIe, IIc, and IIgs machines, Merlin 8/16 not only supports all the addressing modes of the 65xx microprocessors, but the different assemblers are customized to the systems they are most likely to be used on.

### Merlin 8

When either version of Merlin 8 is started, the assembler assumes a 6502 microprocessor state for assemblies. If you are assembling code for a 65C02 processor, *you must use the assembler XC opcode once* at the beginning of the listing anywhere before an actual 65C02 opcode is used. The requirement for the XC enable directive is a safety feature, so that you don't inadvertently use a 65C02 opcode like STZ or INC in a program designed for a 6502 machine. Because the assembler would otherwise generate no errors with the extended opcodes, a program that unwittingly used an improper code would be very hard to debug.

If you are using Merlin 8 to write a program for the 65802, *you must use two XC pseudo-ops* at the beginning of the source listing to enable the 65802 instructions. Because the 65802 does not support more than 64K of addressable memory, Merlin 8 does not feature any long addressing modes, although macros to duplicate these functions can be written.

### Merlin 16

Because Merlin 16 is most likely to be used on an Apple IIgs, the startup default for the assembler logic is to have 65816 opcodes enabled, thus avoiding the need for using the XC directives. If you will be using Merlin 16 to assemble code for Apple IIe compatible computers, you may want to consider changing the PARMS file for Merlin 16 to require the use of the XC directives as a safety device to prevent accidental use of 65816 opcodes in programs written for the Apple IIe. Even Apple Computer is not above these hazards, as evidenced by the use of a 65C02 instruction in a version of ProDOS that was intended to have been compatible with 6502 machines.

In regards to the M and X bits of the 65816, the assembler logic of Merlin 16 starts out assuming the processor will be in the Emulation Mode, since this is the power-up state of the microprocessor. Remember, this is an assumption about *your* source code, and has nothing to do with the actual program code of Merlin 16 itself. Like the XC default, you can change the PARMS file to start a source listing in any setting of the M and X bits you prefer.

When programming for the 65816, Merlin 16 uses the following definitions for address expressions:

```
LDA   expression      use one (low) byte of the expression
LDA   <expression     use one (low) byte of the expression
LDA   >expression     use two bytes (low word) of the expression
PEA   ^expression     use two bytes (high word) of the expression
LDA   |expression     use three bytes (complete address) of the expression
```

this last example is equivalent in Merlin 16 to:

```
LDAL  expression      use three bytes (complete address) of the expression
```

Merlin 16 allows the use of the 4th character in the opcodes ADC, AND, CMP, EOR, LDA, ORA, SBC and STA to force the long addressing modes, as shown by the previous example.

In addition, the instructions JML and JSL are always assembled in the long form, and JSR and JMP are always assembled in the short or 64K address form.

### Special Forced Non-Zero Page Addressing

There is no difference in syntax for zero page and absolute modes. The assembler automatically uses zero page mode when appropriate. Merlin 8/16 provides the ability to *force* non-zero page addressing. The way to do this is to add anything except D in Merlin 8, or L in Merlin 16, to the end of the opcode. Example:

```
LDA  $10   assembles as zero page (2 bytes: A5  10)
```

while,

```
LDA: $10   assembles as non-zero page (3 bytes: AD  10  00)
```

Also, in the indexed indirect modes, only a zero page expression is allowed, and the assembler will give an error message if the "expr" does not evaluate to a zero page address.

**NOTE:** The Accumulator Mode does not require an operand (the letter "A"). For example, to do an LSR of the accumulator, you can use:

```
1  LABEL  LSR          ; LOGICAL SHIFT RIGHT
```

Some assemblers perversely require you to put an "A" in the operand for this mode.

Merlin 8/16 will decide the legality of the addressing mode for any given opcode.

### Sweet 16 Opcodes (Merlin 8 only)

The Merlin 8 assembler accepts all Sweet 16 opcodes with the standard mnemonics. The usual Sweet 16 registers R0 to R15 do not have to be EQUated and the R is optional. For the SET opcode, either a space or a comma may be used between the register and the data part of the operands; that is, SET R3,LABEL is equivalent to SET R3 LABEL. It should be noted that the NUL opcode is assembled as a one-byte opcode the same as HEX 0D, and not a two byte skip, since this would be interpreted by ROM Sweet 16. This is intentional, and is done for internal reasons.

**NOTE:** The Sweet 16 opcodes will not be recognized by Merlin 8 unless the SW pseudo opcode has been previously assembled. This pseudo op will enable assembly of Sweet 16.

### 65C02 and 65802 Opcodes

The Merlin 8 and Merlin 16 assemblers accept all the 6502, 65C02 and 65802 opcodes with standard mnemonics. It also accepts BLT (Branch if Less Than) and BGE (Branch if Greater or Equal) as pseudonyms for BCC and BCS, respectively. The XC pseudo opcode activates these features. This opcode is discussed in the following section on pseudo ops.

You will have problems if you do not use the standard 65C02 opcodes as specified by GTE, NCR, and Rockwell. In creating the IIc Reference Manual, Apple apparently did not check with the manufacturers regarding the final set of opcodes. Thus, Apple refers to two *non-standard* opcodes, INA and DEA.

To increment and decrement the Accumulator, use INC and DEC with no operand, as is consistent with other Accumulator directed commands such as LSR, ASL, etc.

Branch on Bit Set (BBS) and Branch on Bit Reset (BBR) are also *non-standard* Rockwell opcodes and are not supported by the NCR and GTE chips.

Merlin 16 supports the alternate opcodes:

| Standard Opcode: | Alternate Opcode: |
|---|---|
| TCS | TAS |
| TSC | TSA |
| XBA | SWA |
| TCD | TAD |
| TDC | TDA |
| BCC | BLT |
| BCS | BGE |

# ASSEMBLER PSEUDO OPCODE DESCRIPTIONS

## EQU or (=) (EQUate)

```
Label EQU expression
Label  =  expression  (alternate syntax)
   START EQU $1000          [ equate START to $1000 ]
   CHAR  EQU "A"            [ equate CHAR to ASCII value of A ]
   PTR   =   *              [ PTR equals present address in the assembled source listing ]
   LABEL =   55             [ LABEL equals the decimal value of 55 ]
```

```
LABEL  EQU  $25
       LDA  LABEL
```

This will load the accumulator with the value stored in location $25.

```
LABEL  EQU  #$25
       LDA  LABEL
```

This will load the accumulator with the value of $25.

**IMPORTANT: Forgetting to include the # symbol to load an immediate value is probably the number-one cause of program bugs. If you're having a problem, double check immediate value syntax first!**

EQU is used to define the value of a label, usually an exterior address or an often used constant for which a meaningful name is desired. All EQUates should be located at the beginning of the program.

**NOTE:** The assembler will not permit an EQUate to a zero page number after the label equated has been used, since bad code could result from such a situation . Also see the section on Variables.

(1) For Example:

```
1  LABEL  LDA    #LEN
2  LABEL  DFB    $00
3         DFB    $01
4  LEN    EQU    * - LABEL
```

When assembled, this will give an "ILLEGAL FORWARD REFERENCE IN LINE 4" ERROR message. The solution is as follows:

```
1         LDA    #END - LABEL
2  LABEL  DFB    $00
3         DFB    $01
4  END
```

Note that labels are CASE SENSITIVE. Therefore, the assembler will consider the following labels as different labels:

```
START          [ upper case label ]
Start          [ mixed case label ]
start          [ lower case label ]
```

## EXT (EXTernal label)

```
label EXT          [ label is external labels name ]
   PRINT EXT       [ define PRINT as external ]
```

This pseudo op defines a label as an external label for use by the Linker. The value of the label, at assembly time, is set to $8000, but the final value is resolved by the Linker. The symbol table will list the label as having the value of $8000 plus its external reference number (0-$FE). See the Linker section of the manual for more information on this opcode.

## ENT (ENTry label)

```
label ENT
   PRINT ENT       [ define PRINT as entry label ]
```

This pseudo-op will define the label column as an ENTRY label. An entry label is a label that may be referred to as an EXTernal label by another REL code module, which may refer to the ENT label just as if it were an ordinary label. It can be EQUated, jumped to, branched to, etc. The true address of an entry label will be resolved by the Linker.

See *The Linker* section of the manual for more information on this opcode.

## ORG (set ORiGin)

```
ORG expression
ORG
   ORG  $1000            [ start code at $1000 ]
   ORG  START+END        [ start at value of expression ]
   ORG                   [ re-ORG ]
```

Establishes the address at which the program is designed to run, and where it will be automatically BLOADed in memory if it is a BINary type object file. This is not necessarily where Merlin 8/16 will actually assemble the code with the ASM command. ORG defaults to $8000. Ordinarily there will be only one ORG and it will be at the start of the program. If more than one ORG is used, the first one establishes the BLOAD address, while the second actually establishes a new origin for any code that follows it. This can be used to create an object file that would load to one address though it may be designed to run at another address.

NOTE: If you need to back up the object pointers you must use DS-1 . This *cannot* be done by ORG*-1.

ORG without an operand is accepted and is treated as a "REORG" type command. It is intended to be used to re-establish the correct address pointer after a segment of code which has a different ORG. When used in a REL file, all labels in a section between an "ORG address" and an "ORG noaddress" are regarded as absolute addresses. This should only be used in a section that is to be moved to an explicit address.

Example of ORG without an operand:

```
                    1             ORG  $1000
1000: A0 00         2             LDY  #0
1002: 20 21 10      3             JSR  MOVE       ;"MOVE" IS
1005: 4C 12 10      4             JMP  CONTINUE   ;NOT LISTED.
                    5             ORG  $300       ;ROUTINE TO
0300: 8D 08 C0      6  PAGE3      STA  MAINZP     ;BE MOVED
0303: 20 ED FD      7             JSR  COUT
0306: 8D 09 C0      8             STA  AUXZP
0309: 60            9             RTS
                    10            ORG            ;REORG
1012: A9 C1         11 CONTINUE   LDA  #"A"
1014: 20 00 03      12            JSR  PAGE3
```

Sometimes, you will want to generate two blocks of code with separate ORGs in one assembly. There are four ways of doing this involving four different directives. These are DSK, SAV, DS and REL. All four are described later in this manual, and are presented here in the interest of continuity.

## METHOD #1: USING THE DSK OPCODE

In this first example, two separate disk files are created with independent ORG values by using the DSK command. This command directs the assembler to assemble all code to disk *following* the DSK command. The file is closed when either the assembly ends or another DSK command is encountered.

```
 1  ******************
 2  * MULTIPLE ORG'S *
 3  *  SOLUTION # 1  *
 4  *  DSK COMMAND   *
 5  ******************
 6
 7       DSK FILEONE          ; CREATE 1ST FILE
 8       ORG $8000            ; DEFINE ITS LOAD ADDRESS
 9       LDA #0               ; SAMPLE PROGRAM LINE
10
11       DSK FILETWO          ; CLOSE 1ST FILE, START 2ND
12       ORG $8100            ; DEFINE ITS LOAD ADDRESS
13       LDY #1               ; ANOTHER PROGRAM, FILE CLOSED AT END
```

## METHOD #2: USING THE SAV OPCODE

In this second example, two separate disk files are again created with independent ORG values, but this time by using the SAV command. This command directs the assembler to save all code assembled *previous* to the SAV code disk.

```
 1  ******************
 2  * MULTIPLE ORG'S *
 3  *  SOLUTION # 2  *
 4  *  SAV COMMAND   *
 5  ******************
 6
 7       ORG $8000            ; LOAD ADDRESS FOR 1ST FILE
 8       LDA #0               ; SAMPLE PROGRAM LINE
 9       SAV FILEONE          ; SAVE 1ST FILE
10
11       ORG $8100            ; LOAD ADDRESS FOR 2ND FILE
12       LDY #1               ; SAMPLE PROGRAM LINE
13       SAV FILETWO          ; SAVE 2ND FILE
```

## METHOD #3: USING THE DS OPCODE

In this third example, just one file is created on disk, but the two blocks of code are separated by approximately a $100 byte gap, less the size of the first code block.

This might be useful, for example, if you wanted your code to skip over the Hi-Res page 1 area of memory. Please read the section on SAV for more information about multiple ORGs in a program.

```
1  ******************
2  * MULTIPLE ORG'S *
3  *  SOLUTION # 3  *
4  *   DS COMMAND   *
5  ******************
6
7       ORG $8000          ; LOAD ADDRESS OF FILE
8       LDA #0             ; SAMPLE PROGRAM LINE
9       DS \               ; FILL WITH $0 TO NEXT PG. BOUNDARY
10                         ; or could have been DS $8100-*
11      LDY #1             ; SAMPLE LINE OF 2ND SEGMENT
12                         ; THIS WILL START AT $8100
```

## METHOD #4: USING THE REL OPCODE

The REL directive is used to create relocatable files. The Linker use these REL files to create the final object code to run at a given location. The Merlin 16 Linker supports multiple output files, and so can be used to create two or more files with independent ORG values.

```
1  ******************
2  * MULTIPLE ORG'S *
3  *  SOLUTION #4A  *
4  *  REL COMMAND   *
5  ******************
6
7       REL                ; RELOCATABLE FILE TYPE (LNK)
8       DSK FILEONE.L      ; CREATE 1ST LNK FILE
9       LDA #0             ; SAMPLE PROGRAM LINE
```

```
1  ******************
2  * MULTIPLE ORG'S *
3  *  SOLUTION #4B  *
4  *  REL COMMAND   *
5  ******************
6
7       REL                ; RELOCATABLE FILE TYPE (LNK)
8       DSK FILETWO.L      ; CREATE 2ND LNK FILE
9       LDA #0             ; SAMPLE PROGRAM LINE
```

This example is for the Merlin 16 Linker only. These two files would be linked for the desired ORG addresses with a Link command file like this:

```
 1  ******************
 2  * MULTIPLE ORG'S *
 3  *      LINKER    *
 4  *  COMMAND FILE  *
 5  ******************
 6
 7       ORG $8000            ; SPECIFY 1ST ADDRESS
 8       LNK FILEONE.L        ; LINK 1ST FILE
 9       SAV FILE1            ; SAVE 1ST OBJECT FILE
10
11       ORG $8100            ; SPECIFY 2ND ADDRESS
12       LNK FILETWO.L        ; LINK 2ND FILE
13       SAV FILE2            ; SAVE 2ND OBJECT FILE
```

Although the Linker is normally used to *combine* several source files, or to communicate label values between programs, it can be used to assemble even unrelated files.

### REL (RELocatable code module)

REL
    REL [ only option for this opcode ]

This opcode instructs the assembler to generate code files compatible with the relocating linker. This opcode must occur *prior* to the use or definition of any labels. See the Linker section of this manual for more information on this opcode.

### OBJ (set OBJect)

OBJ expression
    OBJ $4000 [ use of hex address ]
    OBJ START [ use with a label ]

The OBJ opcode is accepted only *prior* to the start of the code and it only sets the division line between the symbol table and object code areas in memory, which defaults to $8000. *The OBJ address is accepted only if it lies between $4000 and $BFE0.* This may cause a problem if you try to assemble a listing OBJ'ed to $300, for example.

Nothing disastrous will happen if OBJ is out of range; when you return to the Main Menu to save your object file, no object file address and length values will be displayed on the screen, and Merlin 8/16 will simply beep at you if you try to save an object file.

The main reason for using OBJ is to be able to quit the assembler directly, test a routine in memory, and then be able to immediately return to the assembler to make any corrections. If you want to do this, simply use the GET command (Example: GET $300) in the DOS 3.3 version of Merlin 8 before quitting to BASIC.

In the ProDOS version of Merlin 8/16, this isn't an option because you can't temporarily quit Merlin 8/16 to BASIC. *For ProDOS, it is recommended that you disregard the use of OBJ entirely.* To test a program from the Main Menu, you should save the source code, save the object code, then quit to BASIC.SYSTEM. Then BLOAD the object file. The file will automatically load at the proper location.

Most people should never have to use OBJ. If the REL opcode is used then OBJ is disregarded. If DSK is used then you can, but may not have to, set OBJ to $BFE0 to maximize the space for the symbol table.

In Merlin 16, the address range of the symbol table is printed in hex, at the end of an assembly. This allows you to see when a new OBJ value may be needed. You can also use the DSK command should the object file become too big.

## PUT (PUT a text file in assembly)

PUT filename

DOS 3.3 Examples:

```
PUT  SOURCEFILE      [ PUTs file T.SOURCEFILE ]
PUT  !SOURCE         [ PUTs file SOURCE ]
PUT  !SOURCE,D2      [ PUTs file SOURCE from drive 2 ]
```

ProDOS Examples:

```
PUT  SOURCEFILE      [ PUTs file SOURCEFILE.S ]
PUT  /PRE/SOURCE     [ PUTs file SOURCE.S from subdirectory PRE ]
```

"PUT filename" reads the named file and inserts it at the location of the opcode.

Occasionally your source file will become too large to assemble in memory. This could be due to a very long program, extensive comments, dummy segments, etc. In any case, this is where the PUT opcode can make life easy. All you have to do is divide your program into sections, then save each section as a separate text file. The PUT opcode will load these text files and insert them in the "Master" source file at the location of the PUT opcode. This "Master" source file usually only contains equates, macro definitions (if used), and *all* of your PUT opcodes.

A Master source file might look something like this:

```
********************
*  Master Source   *
********************

* LABEL DEFINITIONS

  LABEL1    EQU  $00
  LABEL2    EQU  $02
  COUT      EQU  $FDED

* MACRO DEFINITIONS

  SWAP      MAC
            LDA ]1
            STA ]2
            <<<

* SAMPLE SOURCE CODE

            LDA  #LABEL1
            STA  LABEL2
            LDA  #/LABEL1
            STA  LABEL2+1
            LDA  LABEL1
            JSR  COUT
            RTS

* BEGIN PUTFILES

            PUT  FILE1       ; FIRST SOURCE FILE SEGMENT
            PUT  FILE2       ; SECOND SOURCE FILE SEGMENT
            PUT  FILE3       ; THIRD SOURCE FILE SEGMENT
```

NOTE: You cannot define macros from within a PUT file. Also, you cannot call the next PUT file from within a PUT file. All macro definitions and PUT opcodes must be in the Master source file. There are other uses for PUT files such as PUTting portions of code as subroutines, PUTting a file of ProDOS global page equates, etc. The possibilities are almost endless.

Here's an example of a Master program that uses 3 PUT files to create a final object file called FINAL.OBJ, which is called from an Applesoft BASIC program. The DSK command is not required when using PUT files, but may be needed for object files that are too large to fit in memory, or where a special filetype, other than BIN, is desired for the object file.

```
1  * MASTER CALLING PROGRAM
2
3  COUT    EQU  $FDED
4  HOME    EQU  $FC58
5
```

```
 6          ORG  $8000
 7
 8          DSK  FINAL.OBJ  ; OUTPUT FILE
 9          JSR  HOME
10          PUT  FILE1      ; Named "T.FILE1" on disk (Merlin 8, DOS 3.3)
11          PUT  FILE2      ; Named "T.FILE2" on disk
12          PUT  FILE3      ; Named "T.FILE3" on disk
13                          ; Named "FILE1.S, etc. on ProDOS disk)
```

And here are the text files that the Master program calls in by using the PUT commands:

```
 1  * FILE1
 2
 3          LDX  #0
 4  LOOP1   LDA  STRING1,X
 5          BEQ  FILE2
 6          JSR  COUT
 7          INX
 8          BNE  LOOP1
 9  STRING1 ASC "THIS IS FILE 1"
10          HEX  8D00
```

```
 1  * FILE2
 2
 3  FILE2   LDX  #0
 4  LOOP2   LDA  STRING2,X
 5          BEQ  FILE3
 6          JSR  COUT
 7          INX
 8          BNE  LOOP2
 9  STRING2 ASC "NOW ITS FILE 2"
10          HEX  8D00
```

```
 1  * FILE3
 2
 3  FILE3   LDX  #0
 4  LOOP3   LDA  STRING3,X
 5          BEQ  DONE
 6          JSR  COUT
 7          INX
 8          BNE  LOOP3
 9  DONE    RTS
10  STRING3 ASC "FINALLY FILE 3"
11          HEX 8D00
```

Each PUT file (FILE1, FILE2, FILE3) prints a message identifying which file is in operation.

The final assembly is tested by this Applesoft program:

```
10  TEXT : HOME
20  PRINT  CHR$ (4);"BLOAD FINAL.OBJ"
25  CALL 32768
30  VTAB 10: HTAB 10: PRINT "IT REALLY WORKS!"
40  VTAB 15: LIST : END
```

When this program is run, the following lines of text should appear on the screen:

```
THIS IS FILE 1
NOW ITS FILE 2
FINALLY FILE 3
IT REALLY WORKS!
```

**DOS 3.3 NOTE:** Drive and slot parameters are accepted in the standard DOS syntax. The "filename" specified must be a text file with the "T." prefix. If it doesn't have the "T." prefix in the disk catalog, the "filename" specified must start with a character less than "@" in ASCII value. This tells Merlin 8/16 to look for a file without the "T." prefix. The "!" character can be used for this purpose. For example:

Disk file name = T.SOURCE CODE [ name in catalog ]
PUT file name = SOURCE CODE [ name in PUT opcode ]

Disk file name = SOURCE CODE [ name in catalog ]
PUT file name = !SOURCE CODE [ name in PUT opcode ]

**ProDOS NOTE:** Drive and slot parameters are not accepted; pathnames must be used. Note that the above name conventions do not apply to ProDOS, since all source files under ProDOS are text files.

**NOTE:** "Insert" refers to the effect on assembly and not to the *location* of the source. The file itself is actually placed just following the main source. These files are only in memory one at a time, so a very large program can be assembled using the PUT facility.

There are two restrictions on a PUT file. First, there cannot be macro definitions inside a file which is PUT; they must be in the Master source file or in a USE file. Second, a PUT file may not call another PUT file with the PUT opcode. Of course, linking can be simulated by having the Master program just contain the macro definitions and call, in turn, all the others with the PUT opcode.

Any variables, such as ]LABEL, may be used as "local" variables. The usual local variables ]1 through ]8 may be set up for this purpose using the VAR opcode.

The PUT facility provides a simple way to incorporate often used subroutines, such as SENDMSG or PRDEC, in a program.

## USE (USE a text file as a macro library)

USE filename

```
USE MACRO LIBRARY    [DOS 3.3 example]
USE !MACROS          [DOS 3.3, no "T." prefix]
USE MACROS,S5,D1     [DOS 3.3 with slot/drive]
USE /LIB/MACROS      [ProDOS pathname]
```

This works similarly to PUT but the file is kept in memory. It is intended for loading a macro library that is USEd by the source file.

It can also be used for including a common library of equates in source files to avoid having to type them into every new program you write. For example, this equate file:

```
**********************
* COMMON EQUATE FILE *
**********************

HOME    EQU  $FC58      ; MONITOR CLEAR SCREEN ROUTINE
VTAB    EQU  $FC22      ; MONITOR VERTICAL TAB ROUTINE
CH      EQU  $24        ; HORIZ. CURSOR POSITION
 Etc...
```

Could be included in every program you write using the USE command:

```
**********************
*   SAMPLE PROGRAM   *
**********************

PTR    EQU  $06         ; POINTER FOR MY PROGRAM
       USE  EQUATES     ; USE PRE-DEFINED EQUATES
BEGIN JSR   HOME        ; CLEAR SCREEN (USE HOME LABEL)
Etc.
```

Normally, the assembled listing will print out all the labels defined in the EQUATES file, but you could use LST ON and LST RTN at the beginning and end of the EQUATES file to suppress the listing of just the defined labels.

## VAR (setup VARiables)

```
VAR expr;expr;expr...
   VAR 1;$3;LABEL          [ set up VAR's 1,2 and 3 ]
```

This is a convenient way to equate the variables ]1 - ]8. For example, VAR 3;$42;LABEL will set ]1 = 3, ]2 = $42, and ]3 = LABEL. This is designed for use just *prior* to a PUT. If a PUT file uses ]1 - ]8, except in lines for calling macros, there *must* be a previous declaration of these.

## SAV (SAVe object code)

```
SAV filename
   SAV  FILE               [ ProDOS or DOS 3.3 syntax ]
   SAV  /OBJ/PROG          [ ProDOS pathname syntax ]
```

SAV filename will save the current object code under the specified name. This acts the same as the Main Menu object saving command, but it can be done several times during assembly.

This pseudo-op provides a means of saving portions of a program having more than one ORG. It also enables the assembly of extremely large files. After a save, the object address is reset to the last specification of OBJ or to $8000 by default.

Files saved with the SAV command will be saved to BLOAD at the correct address.

SAV allows you to save sections of assembled object code during an assembly. It saves all assembled code in the current assembly at the point at which the SAV opcode occurs. This applies *only* to the first SAV in a source. With each additional SAV, Merlin 8/16 only saves the object code generated since the last SAV. This feature allows you to use one source file to assemble code and then SAV sections in separate files. Together with PUT and DSK, SAV makes it possible to assemble extremely large files.

For example, suppose you have a program that uses Hi-Res graphics and is located in memory at two different places. The first part is located at $800 and the second part is at $6000. Your program is divided this way because it is 16K bytes long and thus the Hi-Res pages fall in the middle of your program.

When you first assembled your program you didn't realize the Hi-Res pages were a problem. Your program worked for about two seconds, but when it cleared the Hi-Res screens, it bombed to the Monitor. Clearing the Hi-Res screens also cleared your program! What do you do now?

Just determine in your program where address $2000 is, since this is the start of the Hi-Res Page 1. Once you find this point, it is a simple matter to put in a JMP opcode, follow it immediately with an ORG to $6000, then reassemble the program. You look at the assembly listing and sure enough,

all of the code that used to reside at $2000 is shown at $6000. Then you run your program and it crashes again!

You go into the Monitor and find that none of your code is at $6000. It's just a bunch of hex garbage! The answer is that when more than one ORG statement is used, Merlin 8/16 *does not* physically move the generated code to the new address, it adds it to the end of the previous code. Therefore, the code that should have started at $6000 was *assembled* with all of its *addresses* correct for $6000, but its actual *location* was still down at $2000.

Merlin 8/16 SAV's the day! You need to assemble your source as one file since the two sections refer to each other, but each section needs to be put in different memory locations. The answer is to assemble the entire file with SAV's. Each section will be saved as a binary file with the proper load address. Thus in the following example, when the entire file is assembled, two binary files will be generated and saved. The first will be called FILE1 and will have a load address of $800. The second will be called FILE2 and will have a load address of $6000.

Therefore, *SAV is used to save sections of code to separate individual binary files during an assembly*. With SAV, you can assemble code that may not be continuous in memory but which must be assembled all at once because the sections refer to each other, and may share labels, data, and/or subroutines.

See the example of the multiple-ORG files using SAV at the beginning of this section for an illustration of the SAV command.

NOTE: The Linker provides an alternate way of acheiving this same result. A linker is used more often for large programs because the each segment can be individually created, assembled, and then linked into the final program without re-assembling the other segments, thus saving time during program development.

### TYP (set ProDOS file type for DSK and SAV)

TYP expression

```
TYP  $00          [ no file type ]
TYP  $06          [ binary file type ]
```

This sets the file type to be used by the DSK or SAV opcodes. The default is the BIN type. Valid file types for Merlin 8 are 0,6,$F0-$F7, and $FF (no type, BIN, CMD, user defined, SYS). In Merlin 16, there are no restrictions on the filetypes available.

## DSK (assemble directly to DiSK)

DSK filename (or pathname for ProDOS)

```
DSK PROG              [ DOS 3.3 or ProDOS ]
DSK /OBJ/PROG         [ ProDOS pathname example ]
```

"DSK filename" will cause Merlin 8/16 to open a file specified in the opcode and place all assembled code in that file. It is used at the *start* of a source file before any code is generated. Merlin 8/16 then writes all the following code directly to disk. If DSK is already in effect, the old file will be closed and the new one opened. This is useful primarily for extremely large files.

**NOTE:** Files intended for use with the Linker must be saved with the DSK pseudo op. See the REL opcode for details.

The DSK opcode has three basic purposes:

1) It allows you to assemble programs that result in object code larger than Merlin 8/16 can normally keep in memory.

2) It allows you to automatically put your object code on disk without having to remember to use the Main Menu Object save command.

3) It is used in conjunction with the TYP command to create object files with a filetype other than BIN.

The first purpose is the most often used reason for utilizing the DSK opcode.

**NOTE:** Using DSK will slow assembly significantly. This is because Merlin 8/16 will write a sector to disk every time 256 bytes of object code have been generated. If you don't need a copy of the object code on disk, you should not use the DSK opcode, or use a conditional to defeat it. This is illustrated in the APPLESOFT.S source also.

The assembly speed of source programs that use DSK, PUT, USE or SAV can be improved significantly by putting the referenced files on a RAM disk.

Here is an example listing of a program that creates two separate object files using the DSK command:

```
1  * DSK SAMPLE *
2            DSK   FILEONE     ;ASSEMBLE 'FILEONE' TO DISK
3            ORG   $300        ;'FILEONE' AT $300 (CALL 768)
4  COUT      EQU   $FDED
5  HOME      EQU   $FC58
6            JSR   HOME
7            LDX   #0
8  LOOP1     LDA   STRING1,X
```

```
 9              BEQ   DONE1
10              JSR   COUT
11              INX
12              BNE   LOOP1
13   DONE1      RTS
14   STRING1    ASC   "THIS IS ONE"
15              HEX   8D00
16
17              DSK   FILETWO     ;ASSEMBLE 'FILETWO' TO DISK
18              ORG   $8000       ;'FILETWO' AT $8000 (CALL 32768)
19               .
20              LDX   #0
21   LOOP2      LDA   STRING2,X
22              BEQ   DONE2
23              JSR   COUT
24              INX
25              BNE   LOOP2
26   DONE2      RTS
27   STRING2    ASC "NOW IT'S TWO"
28              HEX 8D00
```

This can be tested with the following Applesoft program.

```
10  PRINT  CHR$ (4);"BLOAD FILEONE"
15  CALL 768
20  PRINT  CHR$ (4);"BLOAD FILETWO"
25  CALL 32768
30  END
```

When run, the following text should appear on the screen:

```
THIS IS ONE
NOW IT'S TWO
```

## END (END of source file)

END
    END [ only option for this opcode ]

This rarely used or needed pseudo opcode instructs the assembler to ignore the rest of the source. Labels occurring after END will not be recognized.

## DUM (DUMmy section)

DUM expression
    DUM $1000 [ start DUMmy code at $1000 ]
    DUM LABEL [ start code at value of LABEL]
    DUM END-START [ start at val of END-START ]

This starts a section of code that will be examined for the values of labels but will produce no object code. The expression must give the desired ORG of this section. It is possible to re-ORG such a section using another DUMMY opcode or using ORG. Note that although no object code is produced from a dummy section, the text output of the assembler will appear as if code is being produced, so you can see the addresses as they are referenced.

## DEND (Dummy END)

DEND
DEND [ only option for this opcode ]

This ends a dummy section and re-establishes the ORG address to the value it had upon entry to the dummy section.

DUM and DEND are used most often to create a set of labels that will exist outside of your program, but that your program needs to reference. Thus, the labels and their values need to be available, but you don't want any code actually assembled for that particular part of the listing.

Sample usage of DUM and DEND:

```
 1          ORG  $1000
 2
 3 IOBADRS =     $B7EB
 4
 5          DUM  IOBADRS
 6 IOBTYPE DFB   1
 7 IOBSLOT DFB   $60
 8 IOBDRV  DFB   1
 9 IOBVOL  DFB   0
10 IOBTRCK DFB   0
11 IOBSECT DFB   0
12          DS   2          ;pointer to DCT
13 IOBBUF  DA    0
14          DA   0
15 IOBCMD  DFB   1
16 IOBERR  DFB   0
17 ACTVOL  DFB   0
18 PREVSL  DFB   0
19 PREVDR  DFB   0
20          DEND
21
22 START   LDA   #SLOT
23          STA  IOBSLOT
24 *  And so on
```

Note that no code is generated for lines 5 through 20, but the labels are available to the program itself, for example, on line 23.

# FORMATTING PSEUDO OPS

## AST (send a line of ASTerisks)

AST expression

| | |
|---|---|
| AST 30 | [ send 30 asterisks to listing ] |
| AST NUM | [ send NUM asterisks ] |

This sends a number of asterisks (*) to the listing equal to the value of the operand. The number format is base 10, so that AST10 will send decimal 10 asterisks, for example. The number is treated modulo 256 with 0 being 256 asterisks.

## CYC (calculate and print CYCle times for code)

CYC
CYC OFF
CYC AVE
CYC FLAGS

| | |
|---|---|
| CYC | [ print opcode cycles & total ] |
| CYC OFF | [ stop cycle time printing ] |
| CYC AVE | [ print cycles & average ] |
| CYC FLAGS | [ print cycles & current mx flag status - Merlin 16 only ] |

This opcode will cause a program cycle count to be printed during assembly. A second CYC opcode will cause the accumulated total to go to zero. CYC OFF causes it to stop printing cycles. CYC AVE will average in the cycles that are underterminable due to branches, indexed and indirect addressing.

The cycle times will be printed or displayed to the right of the comment field and will appear similar to any one of the following:

```
5  ,0326      or       5' ,0326     or     5'',0326
```

The first number displayed, the 5 in the example above, is the cycle count for the current instruction. The second number displayed is the accumulated total of cycles in decimal.

An apostrophe or single quote after the cycle count indicates a possible added cycle, depending on certain conditions the assembler cannot forsee. If this appears on a branch instruction then it indicates that one cycle should be added if the branch occurs. For non-branch instructions, the single quote indicates that one cycle should be added if a page boundary is crossed.

A double quote after the cycle count indicates that the assembler has determined that a branch would be taken and that the branch would cross a page boundary. In this case the extra cycle is displayed and added to the total.

The CYC opcode will also work for the extra 65C02 opcodes in Merlin 8/16. It will not work for the additional 65C02 opcodes present in the Rockwell 65C02, i.e. RMB#, SMB#, BBR# and BSS#. These opcodes are not supported by Merlin 8/16, except when USEing the ROCKWELL macro library. All of these unsupported opcodes are 5-cycle instructions with the usual possible one or two extra cycles for the branch instructions BBS and BBR.

In Merlin 8, the CYC opcode will also work for the 65802 opcodes, but it will *not* add the extra cycles required when M=0 or when X=0. In Merlin 16, there is an additional option, CYC FLAGS, that will print out the current assembler status of the registers sizes, M and X. This can be useful for verifying that register states are as you want them throughout a listing. The CYC function in Merlin 16 *does* correctly take the M and X bits into account when calculating cycle times.

## DAT (DATe stamp assembly listing - ProDOS only)

```
DAT
   DAT                    [ only option for this opcode ]
```

This prints the current date and time on the second pass of the assembler. Available only in the ProDOS versions of Merlin 8/16.

## EXP ON/OFF/ONLY (macro EXPand control)

```
EXP ON or OFF or ONLY
   EXP ON                 [ macro exapand on ]
   EXP OFF                [ print only macro call ]
   EXP ONLY               [ print only generated code ]
```

EXP ON will print an entire macro during the assembly. The OFF condition will print only the PMC pseudo-op. EXP defaults to ON. This has no effect on the object code generated. EXP ONLY will cause expansion of the macro to the listing omitting the call line and end of macro line. However, if the macro call line is labeled, it is printed. This mode will print out just as if the macro lines were written out in the source.

## LST ON/OFF/RTN (LiSTing control)

LST ON or OFF or RTN

| | |
|---|---|
| LST ON | [ turn listing on ] |
| LST OFF | [ turn listing off ] |
| LST | [ turn listing on, optional ] |
| LST RTN | [ return LST state to that in effect before previous LST command. Merlin 16 only] |

This controls whether the assembly listing is to be sent to the Apple screen, or other output device, or not. For example, you may use this to send only a portion of the assembly listing to your printer. Any number of LST instructions may be in the source. If the LST condition is OFF at the end of assembly, the symbol table will not be printed.

The assembler actually only checks the third character of the operand to see whether or not it is a space. Therefore, LST will have the same effect as LST ON. The LST directive will have no effect on the actual generation of object code. If the LST condition is OFF, the object code will be generated much faster, but this is recommended only for debugged programs.

LST RTN is available in Merlin 16 only and will return the LST status to what it was previous to the last instance of LST. For example, if a macro library had LST OFF at the beginning and LST RTN at the end, the library would not be listed in an assembly, but in addition, the list status of the main source file, either on or off, would not be disturbed by the included LST commands of the macro library.

**NOTE:** Control-D from the keyboard toggles this flag during the second pass, and thus can be used to manually turn on or off the screen or printer listing during assembly.

## LSTDO or LSTDO OFF (LiST DO OFF areas of code)

LSTDO
LSTDO OFF

| | |
|---|---|
| LSTDO | [ list the DO OFF areas ] |
| LSTDO OFF | [ don't list DO OFF areas ] |

This opcode causes the listing of DO OFF areas of code to be printed in listings or not to be printed.

### PAG (new PAGe)

PAG
PAG [ only option for this opcode ]

This sends a formfeed, i.e. $8C, to the printer. It has no effect on the screen listing even when using an 80 column card.

### TTL (define Title heading - Merlin 16 only)

TTL string
TTL "Segment Title" [ only option for this opcode ]

This has the same syntax as the ASC pseudo op, and sets the page title in use by the PRTR command. This is used for changing the title at the top of the page during a source listing printout, and is usually followed by a PAG pseudo op.

### SKP (SKiP lines)

SKP expression
SKP 5 [ skip 5 lines in listing ]
SKP LINES [ skip "LINES" lines in listing]

This sends the number of carriage returns in "expression" to the listing. The number format is the same as in AST.

### TR ON/OFF (TRuncate control)

TR ON or OFF or ADR
TR ON [ limit object code printing ]
TR OFF [ don't limit object code print]
TR ADR [ suppress bank byte of addresses - Merlin 16 only ]

TR ON or just TR limits object code printout to three bytes per source line, even if the line generates more than three. TR OFF resets it to print all object bytes.

TR ADR can be used in Merlin 16 to suppress the bank byte part of the address listing at the left of an assembly listing.

# STRING DATA PSEUDO OPS

## GENERAL NOTES ON STRING DATA AND STRING DELIMITERS

Different delimiters have different effects. Any delimiter with an ASCII value less than the apostrophe (') will produce a string with the high-bits on, otherwise the high-bits will be off. For example, the delimiters !"#$%& will produce a string in *negative* ASCII, and the delimiters '()+? will produce one in *positive* ASCII. The quote (") and apostrophe (') are the usual delimiters of choice, but other delimiters provide the means of inserting a string containing the quote or apostrophe as part of the string. Example delimiter effects:

```
"HELLO"                 [ negative ASCII, hi bit set ]
!HELLO!                 [ negative ASCII, hi bit set ]
#HELLO#                 [ negative ASCII, hi bit set ]
&HELLO&                 [ negative ASCII, hi bit set ]
!ENTER "HELLO"!         [ string with embedded quotes - negative ASCII]
'HELLO                  [ positive ASCII, hi bit clear ]
(HELLO(                 [ positive ASCII, hi bit clear ]
'ENTER "HELLO"'         [ string with embedded quotes - positive ASCII ]
```

All of the opcodes in this section, except REV, also accept hex data after the string. Any of the following syntaxes are acceptable:

```
ASC "string"878D00
FLS "string",878D00
DCI "string",87,8D,00
STR "STRING",878D00
INV "string",878D00
```

### ASC (define ASCii text)

```
ASC d-string
   ASC  "STRING"          [ negative ASCII string ]
   ASC  'STRING'          [ positive ASCII string ]
   ASC  "Bye,Bye",8D      [ negative with added hex bytes]
```

This puts a delimited ASCII string into the object code. The only restriction on the delimiter is that it does not occur in the string itself.

### DCI (Dextral Character Inverted)

DCI d-string

```
DCI "STRING"          [ negative ASCII, except for the "G" ]
DCI 'STRING'          [ positive ASCII, except for the "G" ]
DCI 'Hello',878D      [ positive with two added hex bytes ]
```

This is the same as ASC except that the string is put into memory with the last character having the opposite high bit to the others. When a hex suffix is added, the bit inversion will still be on the last character of the *delimited* string. Thus, only the 'o' in Hello will have the high bit inverted.

### INV (define INVerse text)

INV d-string

```
INV "STOP!"           [ negative ASCII, inverse on printing]
INV 'END',878D        [ positive, added bytes ]
```

This puts a delimited string in memory in inverse format.

### FLS (define FLaShing text)

FLS d-string

```
FLS "The End"         [ negative ASCII, flash on printing]
FLS 'The End',8D00    [ positive, flash with added bytes ]
```

This puts a delimited string in memory in flashing format.

### REV (REVerse)

REV d-string

```
REV "Insert"          [ negative ASCII, reversed in memory ]
REV 'Insert"          [ same as above but positive ]
```

This puts the d-string backwards in memory. Example:

```
REV "DISK VOLUME"
```

gives EMULOV KSID (delimiter choice as in ASC). HEX data may *not* be added after the string terminator.

## STR (define a STRing with a leading length byte)

```
STR d-string
   STR  "/PATH/"          [ positive ASCII, (ProDOS pathname?)]
   STR  "HI"              [ result= 02 C8 C9 ]
   STR  'HI',8D           [ result= 02 48 49 8D ]
```

This puts a delimited string into memory with a leading length byte. Otherwise it works the same as the ASC opcode. This facility is mainly intended for use with ProDOS which uses this type of data extensively.

*Note that following HEX bytes, if any, are not counted in the length.* Thus, although the third example above will not generate an error, it should not be used since any hex bytes appended to the end of a defined string would not be printed or otherwise recognized by a routine using the length byte as part of the descriptor for the string data.

# DATA AND STORAGE ALLOCATION PSEUDO OPS

## DA or DW (Define Address or Define Word)

DA expression or DW expression

```
DA $FDF0          [ results: F0 FD in memory ]
DA 10,$300        [ results: 0A 00 00 03 ]
DW LAB1,LAB2      [ example of use with labels ]
```

This stores the two-byte value of the operand, usually an address, in the object code, low-byte first.

These two pseudo ops also accept multiple data separated by commas (such as DA 1,10,100).

## DDB (Define Double-Byte)

DDB expression

```
DDB $FDED+1       [ results: FD EE in memory ]
DDB 10,$300       [ results: 00 0A 03 00 ]
```

As above with DA, but places high-byte first. DDB also accepts multiple data (such as DDB 1,10,100).

## DFB or DB (DeFine Byte or Define Byte)

DFB expression or DB expression

```
DFB 10            [ results: 0A in memory ]
DFB $10           [ results: 10 in memory ]
DB >$FDED+2       [ results: FD in memory ]
DB LAB            [ example of use with label ]
```

This puts the byte specified by the operand into the object code. It accepts several bytes of data, which must be separated by commas and contain no spaces. The standard number format is used and arithmetic is done as usual.

The pound sign (#) is acceptable but ignored, as is the less-than sign (<). The greater-than sign (>) may be used to specify the hi-byte of an expression, otherwise the low-byte is always taken. The > should appear as the first character only of an expression or immediately after #. That is, the instruction DFB >LAB1-LAB2 will produce the hi-byte of the value of LAB1-LAB2.

For example:

```
DFB $34,100,LAB-LAB2,%011,>LAB1-LAB2
```

is a properly formatted DFB statement which will generate the hex object code:

34 64 DE 0B 09

assuming that LAB1=$81A2 and LAB2=$77C4.

### ADR (Define Long Address - 3 bytes - Merlin 16 only)

ADR expression

| | |
|---|---|
| ADR $01FDF0 | [ results: F0 FD 01 in memory ] |
| ADR 10,$020300 | [ results: 0A 00 00 00 03 02 ] |
| ADR LAB1,LAB2 | [ example of use with labels ] |

This stores the three-byte value of the operand, usually an address, in the object code, low-byte, hi-byte, then bank byte. This pseudo op also accepts multiple data separated by commas such as ADR 1,10,100).

### ADRL (Define Long Address - 4 bytes - Merlin 16 only)

ADRL expression

| | |
|---|---|
| ADRL $01FDF0 | [ results: F0 FD 01 00 in memory ] |
| ADRL 10,$020300 | [ results: 0A 00 00 00 00 03 02 00 ] |
| ADRL LAB1,LAB2 | [ example of use with labels ] |

This stores the four-byte value of the operand, usually an address, in the object code, low-byte, hi-byte, bank byte, then hi-byte of the high word. This pseudo op also accepts multiple data separated by commas (such as ADR 1,10,100).

The decision as to whether to use ADR or ADRL will depend largely on whether the addresses defined are to be used as an indirect pointer (for example, JSR [PTR]), or as a address to be accessed with two LDA type instructions (LDA LABEL, LDA LABEL+2).

### HEX (define HEX data)

HEX hex-data

| | |
|---|---|
| HEX 0102030F | [ results: 01 02 03 0F in memory ] |
| HEX FD,ED,C0 | [ results: FD ED C0 in memory ] |

This is an alternative to DFB which allows convenient insertion of hex data. Unlike all other cases, the $ is not required or accepted here. The operand should consist of hex numbers having two hex digits, thus you would use 0F, not F. They may be separated by commas or may be adjacent. An error message will be generated if the operand contains an odd number of digits or ends in a comma, or in any case, contains more than 64 characters.

## DS (Define Storage)

```
DS experesssion
DS expression1, expression2
DS \
   DS 10              [ zero out 10 bytes of memory ]
   DS 10,$80          [ put $80 in 10 bytes of memory ]
   DS \               [ zero memory to next memory page ]
   DS \,$80           [ put $80 in memory to next page ]
   DS \,expression2   [ put expresson2 in memory to next page ]
```

This reserves space for string storage data. It zeros out this space if the expression is positive. DS 10, for example, will set aside 10 bytes for storage.

Because DS adjusts the object code pointer, an instruction like DS-1 can be used to back up the object and address pointers one byte.

The first alternate form of DS, with two expressions, will fill expression1 bytes with the value of the low-byte of expression2, provided expression2 is positive. If expression2 is missing, 0 is used for the fill.

The second alternate form, DS \, will fill memory with zeroes until the next memory page. The "DS \,expression2" form does the same but fills using the low-byte of expression2.

### Notes for REL files and the Linker

The back slash (\) options are intended for use mainly with REL files and work slightly differently with these files. Any DS \ opcode occurring in a REL file will cause the linker to load the next file at the first available page boundary, and to fill with zeroes or the indicated byte. Note that for REL files, the location of this code has *no effect* on its action. *To avoid confusion, you should only use this code at the end of a file.*

## USING DATA TABLES IN PROGRAMS

Merlin's various data commands are used by the programmer to store pure data bytes, as opposed to executable program instruction bytes, in memory for use by the program. As an example, here is a program that prints the square of three numbers.

```
1    * DATA TABLE DEMO *
2
3             ORG   $8000
4
5    HOME     EQU   $FC58
6    COUT     EQU   $FDED
7
8    START    JSR   HOME        ;CLEAR SCREEN
9             LDY   #0          ;SET Y TO ZERO
10
11   PRINT1   LDA   DATA1,Y     ;PRINT NUMBER TO BE SQUARED
12            JSR   COUT
13            LDX   #0          ;SET X TO ZERO
14   LOOP1    LDA   DATA2,X     ;LOOP TO PRINT TEXT
15            BEQ   PRINT2
16            JSR   COUT
17            INX
18            BNE   LOOP1
19   PRINT2   LDA   DATA3,Y     ;PRINT SQUARED VALUE
20            JSR   COUT
21            LDA   #$8D
22            JSR   COUT
23            INY
24            CPY   #$03        ;ARE 3 LOOPS COMPLETED?
25            BCS   DONE        ;IF SO WE'RE DONE
26            JMP   PRINT1      ;IF NOT BEGIN AGAIN
27   DONE     RTS
28   DATA1    DFB   #177,178,179
29   DATA2    ASC   " SQUARED IS "
30            HEX   00
31   DATA3    DFB   #177,180,185
```

Notice how the data portion of the program (DATA1, DATA2, DATA3) is referenced in the main body of the program. Also notice that for the purpose of illustration several data definition styles have been used. The actual numbers printed by the program (example, 3 SQUARED IS 9) are stored in the program as defined bytes (DFB) on lines 28 and 31. This could just as easily been done with the ASC pseudo-op. The pseudo-op HEX is also used on line 30 to create the zero byte that terminates the string " SQUARED IS ".

# CONDITIONAL PSEUDO OPS

## DO (DO if true)

DO expression

```
DO 0            [ turn assembly off ]
DO 1            [ turn it on ]
DO LABEL        [ if LABEL<>0 then on ]
DO LAB1/LAB2    [ if LAB1<LAB2 then off ]
DO LAB1-LAB2    [ if LAB1=LAB2 then off ]
DO LABEL-1      [ if LABEL = 0, only if LABEL = 0 or 1 ]
```

This together with ELSE and FIN are the conditional assembly pseudo ops. If the operand evaluates to zero, then the assembler will stop generating object code (until it sees another conditional). See the section on "Building Expressions" for more examples of testing for certain values. Except for macro names, it will not recognize any labels in such an area of code. If the operand evaluates to a non-zero number, then assembly will proceed as usual. This is very useful for macros.

It is also useful for sources designed to generate slightly different code for different situations. For example, if you are designing a program to go on a ROM chip, you would want one version for the ROM and another with small differences as a RAM version for debugging purposes. Conditionals can be used to create these different object codes without requiring two sources.

Similarly, in a program with text, you may wish to have one version for Apples with mousetext characters and one for those without. By using conditional assembly, modification of such programs becomes much simpler, since you do not have to make the modification in two separate versions of the source code.

Every DO should be terminated somewhere later by a FIN and each FIN should be preceded by a DO. An ELSE should occur only inside such a DO/FIN structure. DO/FIN structures may be nested up to eight deep, possibly with some ELSE's between. If the DO condition is off, i.e. value 0, then assembly will not resume until its corresponding FIN is encountered, or an ELSE at this level occurs. Nested DO/FIN structures are valuable for putting conditionals in macros.

## ELSE (ELSE do this)

ELSE

```
ELSE            [ only option for this opcode ]
```

This inverts the assembly condition for the last DO. Thus, ON becomes OFF and OFF becomes ON.

### IF (IF so then do)

```
IF char,]var  (IF char is the first character of ]var)
IF MX plus expression
   IF (,]1                  [ if first char of ]1 is "(" then assemble following code]
   IF ",]TEMP               [ if first char is ", assem ]
   IF "=]1                  [ alternate use with "=" ]
   IF MX                    [ if MX = 1, 2 or 3; Merlin 16 only ]
```

This checks to see if *char* is the leading character of the replacement string for ]var. IF cannot be used for testing whether a label is equal to a value, etc. Use the DO pseudo-op for value tests.

NOTE: Position is important since the assembler checks the first and third characters of the operand for a match. If a match is found then the following code will be assembled. As with DO, this must be terminated with a FIN, with optional ELSEs between. The comma is not examined, so any character, such as the equal sign, may be used there. For example:

```
IF "=]1
```

could be used to test if the first character of the variable ]1 is a double quote (") or not, perhaps needed in a macro which could be given either an ASCII or a hex parameter.

In Merlin 16, IF can be used to check the status of the assembler M & X bits. MX is interpreted as though it has a value in the range 0-3, depending on the current MX flag. The MX can then be included in an expression to control a conditional assembly. This is intended for use in macros to determine register length. For example:

```
IF MX/2                 ; DO if M is short
IF MX/2-1               ; DO if M is long
IF MX&1                 ; DO if X is short
IF MX&1-1               ; DO if X is long
IF MX/3                 ; DO if both M and X are short
IF MX!3/3               ; DO if both M and X are long
IF MX-2/-1              ; DO if M is long and X is short
IF MX-3/-1              ; DO if M is short and X is long
IF MX+1&3               ; DO if either M or X or both are long
IF MX                   ; DO if either M or X or both are short
```

### FIN (FINish conditional)

```
FIN
   FIN                  [ only option for this opcode ]
```

This cancels the last DO or IF and continues assembly with the next highest level of conditional assembly, or it cancels ON if the FIN concluded the last or outer DO or IF.

## USING CONDITIONAL ASSEMBLY

Here's a short example that shows how different program segments can be controlled with conditional assembly:

```
*********************************
* CONDITIONAL ASSEMBLY EXAMPLE *
*********************************

HOME    EQU  $FC58        ; MONITOR CLEAR SCREEN ROUTINE
COUT    EQU  $FDED        ; MONITOR PRINT ROUTINE
BELL    EQU  $FBDD        ; MONITOR "BELL" ROUTINE

FLAG    EQU  1            ; FLAG = 1 = DO THIS VERSION
                          ; IN YOUR PROGRAMS, FLAG CAN HAVE ANY NAME AND
                          ; HAVE WHATEVER RANGE OF VALUES YOU NEED FOR THE
                          ; NUMBER OF POSSIBLE ASSEMBLIES YOU WISH.

BEGIN   JSR  HOME         ; CLEAR SCREEN - ALL PROGRAMS DO THIS

        DO   FLAG         ; ASSEMBLE THIS PART IF FLAG = 1

PART1   LDA  #"A"
        JSR  COUT         ; PRINT LETTER "A"

        ELSE              ; DO PART2 IF FLAG = 0

PART2   LDA  #"B"
        JSR  COUT         ; PRINT LETTER "B"

        FIN               ; END OF CONDITIONAL SEGMENT

BELL    JSR  BELL         ; RING BELL IN ALL VERSIONS

        DO   FLAG-1       ; DO NEXT PART IF FLAG = 0
                          ; THIS SHOWS HOW TO DO INVERSE LOGIC OF 'FLAG'
                          ; (ASSUMES FLAG = 0 OR FLAG = 1)

PART2A  LDA  #"b"
        JSR  COUT         ; PRINT LETTER "b"

        ELSE              ; DO THIS PART IF FLAG = 1

PART1A  LDA #"a"
        JSR  COUT         ; PRINT LETTER "a"

        FIN               ; END OF CONDITIONAL SEGMENT

DONE    RTS               ; ALL VERSIONS END HERE
```

Using IF to test the first character of a parameter passed to a macro lets you add a variety of possible addressing modes to a macro that will depend on the input parameters. Assume we start with a simple macro to move data from one location to another:

The MOV macro moves data from ]1 to ]2:

```
MOV     MAC
        LDA  ]1
        STA  ]2
        <<<
```

We can then construct a more sophisticated macro that uses MOV, but which supports a wide variety of addressing modes:

The MOVD macro moves data from ]1 to ]2 with many available syntaxes

```
MOVD    MAC
        MOV  ]1;]2
        IF   (,]1           ; Syntax MOVD (ADR1),Y;????
        INY
        IF   (,]2           ;  MOVD (ADR1),Y;(ADR2),Y
        MOV  ]1;]2
        ELSE                ;  MOVD (ADR1),Y;ADR2
        MOV  ]1;]2+1
        FIN
        ELSE
        IF   (,]2           ;Syntax MOVD ????;(ADR2),Y
        INY
        IF   #,]1           ;  MOVD #ADR1;(ADR2),Y
        MOV  ]1/$100;]2
        ELSE                ;  MOVD ADR1;(ADR2),Y
        MOV  ]1+1;]2
        FIN
        ELSE                ;Syntax MOVD ????;ADR2
        IF   #,]1           ;  MOVD #ADR1;ADR2
        MOV  ]1/$100;]2+1
        ELSE                ;  MOVD ADR1;ADR2
        MOV  ]1+1;]2+1
        FIN                 ;MUST close ALL
        FIN                 ;conditionals, Count DOs
        FIN                 ;& IFs, deduct FINs.  Must
        <<<                 ;yield zero at end.
```

*The call syntaxes supported by MOVD are:

```
MOVD ADR1;ADR2
MOVD (ADR1),Y;ADR2
MOVD ADR1;(ADR2),Y
MOVD (ADR1),Y;(ADR2),Y
MOVD #ADR1;ADR2
MOVD #ADR1;(ADR2),Y
```

```
MOVD #ADR1;ADR2
MOVD #ADR1;(ADR2),Y
```

Here's a macro that can be created for use with the 65816 to push an immediate value on the stack using PEA, or to first load the contents of another memory location, and then push that value on the stack with a PHA. This type of operation is very common when programming on the Apple IIgs.

```
PushWord    MAC             ; DEFINE MACRO
            IF #=]1         ; IF FIRST CHARACTER OF ]1 IS A '#'
            PEA  ]1         ; PUSH VALUE OF ]1 ON STACK
            ELSE            ; OTHERWISE
            LDA  ]1         ; GET contents OF ]1
            PHA             ; PUSH THAT ON STACK
            FIN             ; END OF CONDITIONAL PART OF MACRO
            EOM             ; END OF MACRO DEFINITION
```

Thus, the PushWord macro could be used in any of these forms:

```
            PushWord #$80     ; PUSH VALUE $80 ON STACK
            PushWord #LABEL   ; PUSH VALUE LABEL ON STACK
```

or,

```
            PushWord $80      ; PUSH CONTENTS OF LOCATION $80 ON STACK
            PushWord LABEL    ; PUSH CONTENTS OF LOCATION LABEL ON STACK
```

# MISCELLANEOUS PSEUDO OPS

## CHK (place CHecKsum in object code)

CHK
CHK [ only option for this opcode ]

This places a checksum byte into object code at the location of the CHK opcode. This is usually placed at the end of the program and can be used by your program at runtime to verify the existence of an accurate image of the program in memory.

The checksum is calculated with Exclusive-ORing each successive byte with the running result. That is, byte 1 is EORed with byte 2 and the result put in the accumulator. Then that value is EORed with byte 3 and the process continued until the last byte in memory has been involved in the calculation. It is not a foolproof error checking scheme, but is adequate for most uses. If you will be publishing your source listing in a magazine, or loading object code in any situation in which you want to assure that a functional copy of the object code has been loaded, then the use of the checksum pseudo-op is recommended.

The following program segment will confirm the checksum at run time:

```
1     STARTCHK LDA   #<STARTCHK
2              STA   PTR
3              LDA   #>STARTCHK
4              STA   PTR+1
5              LDY   #$00
6              LDA   #$00
7              PHA              ; PUSH ZERO ON STACK
8
9     LOOP     PLA              ; RETRIEVE CURRENT CHKSUM
10             EOR   (PTR),Y
11             PHA              ; PUT TEMP BACK
12             INC   PTR
13             BNE   CHK        ; WRAP AROUND YET?
14             INC   PTR+1      ; YEP
15    CHK      LDA   PTR+1
16             CMP   #>PROGEND  ; SEE IF WE'RE DONE YET...
17             BCC   LOOP       ; NOT YET...
18             LDA   PTR
19             CMP   #<PROGEND
20             BCC   LOOP       ; NOPE
21             BEQ   LOOP
22    CHKCS    PLA              ; RETRIEVE CALCULATED VALUE
23             CMP   CHKSUM     ; COMPARE TO MERLIN'S VALUE
24             BNE   ERROR      ; ERROR HANDLER....
25                              ; FALL THROUGH IF O.K.
26  REALSTART  ???              ; REAL PROGRAM STARTS HERE
27             ???
```

```
...

998  PROGEND  RTS              ; END OF FUNCTIONAL PROGRAM
999  CHKSUM   CHK              ; Merlin 8/16 CHECKSUM DIRECTIVE
```

## ERR (force ERRor)

ERR expression
ERR \expression

```
   ERR  ($300)-$80          [ error if $80 not in $300 ]
   ERR  *-1/$4100           [ error if PC > $4100 ]
   ERR  \$5000              [ error if REL code address exceeds $5000 ]
```

"ERR expression" will force an error if the expression has a non-zero value and the message "BREAK IN LINE ???" will be printed. This may be used to ensure your program does not exceed, for example, $95FF by adding the final line:

```
        ERR  *-1/$9600
```

**NOTE:** The above example would only alert you that the program is too long, and will not prevent writing above $9600 during assembly, but there can be no harm in this, since the assembler will cease generating object code in such an instance. The error occurs only on the second pass of the assembly and does not abort the assembly.

Another available syntax is:

```
        ERR ($300)-$4C
```

which will produce an error on the first pass and abort assembly if location $300 in main memory does not contain the value $4C. The primary purpose for this function is to allow your source file to check to see if a USR defined opcode routine has been loaded prior to the assembly. This *does not* check a memory location in the object code.

### NOTES ON REL FILES AND THE ERR PSEUDO OP

The "ERR \expression" syntax gives an error on the second pass if the address pointer reaches expression or beyond. This is equivalent to ERR *-1/expr, but when used with REL files, it instructs the Linker to check that the last byte of the current module does not extend to expression or beyond. The expression must be absolute. If the Linker finds that the current module *does* extend beyond expression, linking will abort with a message "Constraint error:" followed by the value of expression in the ERR opcode. You can see how this works by linking the PI files which are a series of sample file on the Merlin 8/16 disks. They should be linked to an address over $81C. Note that the position of this opcode in a REL file has no bearing on its action, so that it is best to put it at the end.

## KBD (define label from KeyBoarD)

label KBD
label KBD d-string
OUTPUT KBD [ get value of OUTPUT from keyboard ]
OUTPUT KBD "send to printer" [ prompt with the d-string for the value of OUTPUT ]

This allows a label to be equated from the keyboard during assembly. Any expression may be input, including expressions referencing previously defined labels, however a BAD INPUT error will occur if the input cannot be evaluated.

The optional delimited string will be printed on the screen instead of the standard "Give value for LABEL:" message. A colon is appended to the string.

KBD generated labels are used most often to control conditional assemblies. For example, this code segment asks the user to press 0 or 1 to signify which version of a program should be assembled:

```
FLAG    KBD  "Assemble Part 1 or Part 2? (0/1)"

        DO   FLAG-1          ; DO IF FLAG = 0
PART1   LDA  #"A"
        JSR  COUT            ; PRINT "A"
        FIN
        DO   FLAG            ; DO IF FLAG = 1
PART2   LDA  #"B"
        JSR  COUT            ; PRINT "B"
        FIN
DONE    RTS
```

Instead of pressing 0 or 1, you can use the fact that KBD will accept a label as input to accept a Y or N input:

```
N       EQU  0               ; NO = 0
Y       EQU  1               ; YES = 1

FLAG    KBD  "Assemble Part 1 or Part 2? (Y/N)"
```

## LUP (begin a loop)

```
LUP expression  (Loop)
--^             (end of LUP)
```

The LUP pseudo-opcode is used to repeat portions of source between the LUP and the --^ "expression" number of times. An example of this is:

```
LUP 4
ASL
--^
```

which will assemble as:

```
ASL
ASL
ASL
ASL
```

and will show that way in the assembly listing, with repeated line numbers.

Perhaps the major use of this is for table building. As an example:

```
]A      =    0
        LUP  $FF
]A      =    ]A+1
        DFB  ]A
        --^
```

will assemble the table 1, 2, 3, ...,$FF.

The maximum LUP value is $8000 and the LUP opcode will simply be ignored if you try to use more than this.

**NOTE:** The above use of incrementing variables in order to build a table *will not* work if used within a macro. Program structures such as this must be included as part of the main program source.

In a LUP, if the @ character appears in the label column, it will be increased by the loop count, thus A,B,C...etc. Since the loop count is a countdown, these labels will go backwards, i.e. the last label has the A. This makes it possible to label items inside a LUP. This will work in a LUP with a maximum length of 26 counts, otherwise you will get a BAD LABEL error and possibly some DUPLICATE LABEL errors.

## MX (long status Mode of 65802)

```
MX expression
   MX %00          [ M & X = 16 bit modes ]
   MX %01          [ M = 16 bits, X = 8 bits ]
   MX %10          [ X = 8 bits, M = 16 bits ]
   MX %11          [ X = 8 bits, M = 8 bits ]
   MX 3            [ same as MX %11 ]
```

This pseudo-op is used to inform Merlin 8/16 of the intended status of the long status of the 65802 or 65816 processor. In Merlin 8, it functions only when the assembler is in the 65802 mode, i.e. when two consetutive XC opcodes have been given. The assembler cannot determine if the processor is in 16 bit memory mode (M status bit=0) or 16 bit index register mode (X status bit=0). The purpose of the MX opcode is to inform the assembler of the current status of these bits.

Three of the above examples use binary expressions as the operand of the MX opcode. Note that any valid expression may be used as long as it is within the range of 0-3.

NOTE: This opcode *must* be used when using 65802 or 65816 instructions on either Merlin 8 or Merlin 16 to inform the assembler of the proper mode to use in order to insure proper assembly of immediate mode commands such as LDA #expression, etc.

At startup, Merlin 16 assumes the MX setting to be MX %11, that is, Emulation Mode with both Accumulator and Memory register sizes set to 8 bits, although this default can be changed in the Merlin 16 PARMS file.

## PAU (PAUse)

```
PAU
   PAU             [ only option for this opcode ]
```

On the second pass, PAU causes assembly to pause until a key is pressed. This can also be done from the keyboard by pressing the space bar. This is handy for debugging.

## SW (SWeet 16 opcodes - Merlin 8 only)

```
SW
     SW          [ only option for this opcode ]
```

This enables Sweet 16 opcodes available in Merlin 8 only. If SW, and similarly for XC, is not selected then those opcode names can be used for macros. Thus, if you are not using Sweet 16, you can use macros named ADD, SUB, etc.

## USR (USeR definable op-code)

USR optional expressions
USR expression [ examples depend on definition]

This is a user-definable pseudo-opcode. It does a JSR $B6DA. This location will contain an RTS after a boot, a BRUN MERLIN or BRUN BOOT ASM. To set up your routine you should BRUN it from the Main Menu as a disk command. This should just set up a JMP at $B6DA to the your main routine and then RTS.

The following flags and entry points may be used by your routine:

```
USRADS    = $B6DA  ;must have a JMP to your routine
PUTBYTE   = $E5F6  ;see below
EVAL      = $E5F9  ;see below
PASSNUM   = $2     ;contains assembly pass number
ERRCNT    = $1D    ;error count
VALUE     = $55    ;value returned by EVAL
OPNDLEN   = $BB    ;contains combined length of
                   ;operand and comment
NOTFOUND  = $FD    ;see discussion of EVAL
WORKSP    = $280   ;contains the operand and
                   ;comment in positive ASCII
```

Your routine will be called by the USR opcode with A=0, Y=0 and carry set. To direct the assembler to put a byte in the object code, you should JSR PUTBYTE with the byte in A.

PUTBYTE will preserve Y but will scramble A and X. It returns with the zero flag clear so that BNE always branches. On the first pass PUTBYTE *only* adjusts the object and address pointers, so that the contents of the registers are not important. You *must* call PUTBYTE the *same number of times* on each pass or the pointers will not be kept correctly and the assembly of other parts of the program will be incorrect!

If your routine needs to evaluate the operand, or part of it, you can do this by a JSR EVAL. The X register must point to the first character of the portion of the operand you wish to evaluate, thus set X=0 to evaluate the expression at the start of the operand. On return from EVAL, X will point to the character following the evaluated expression. The Y register will be 0, 1, or 2 depending on whether this character is a right parenthesis, a space, a comma, or the end of an operand.

Any character not allowed in an expression will cause assembly to abort with a BAD OPERAND or other error. If some label in the expression is not recognized then location NOTFOUND will be non-zero. On the second pass, however, you will get an UNKNOWN LABEL error and the rest of your routine will be ignored. On return from EVAL, the computed value of the expression will be in location VALUE and VALUE+1, low-byte first. On the first pass this value will be insignificant if NOTFOUND is non-zero.

Appropriate locations for your routine are $300-$3CF and $8A0-$8FF. You must not write to $900.

You may use zero page locations $60-$6F, but should not alter other locations. Also, you must not change any thing from $226 to $27F, or anything from $2C4 to $2FF. Upon return from your routine with an RTS, the USR line will be printed on the second pass.

When you use the USR opcode in a source file, it is wise to include some sort of check in source that the required routine is in memory.

If, for example, your routine contains an RTS at location $310 then:

```
ERR ($310)-$60
```

will test that byte and abort assembly if the RTS is not there. Similarly, if you know that the required routine should assemble exactly two bytes of data, then you can roughly check for it with the following code:

```
LABEL     USR  OPERAND
          ERR  *-LABEL-2
```

This will force an error on the second pass if USR does not produce exactly two object bytes.

It is possible to use USR for several different routines in the same source. For example, your routine could check the first operand expression for an index to the desired routine and act accordingly. Thus "USR 1, whatever" would branch to the first routine, "USR 2,stuff" to the second, etc.

In Merlin 16, the USR opcode has been extended to allow up to 10 USR opcodes, USR0 through USR9. The Merlin 8 USR is equivalent to USR0 and is upward compatible. The number 0-9 is doubled and placed in the X Register and then a JSR $B6DA is done, the standard USR vector. At $B6DA you can place a JMP (VECTORTBL,X) instruction, where VECTORTBL is a list of addresses of your routines placed at any free spot such as page 3. To use routines that would not fit on page 3, you could set the source address at $A,B higher though you may have to copy this to $E00A,$E00B, or you could set HIMEM lower. To do the latter, set both $C,D and $73,74 to the lower address.

USR routines are entered in native 8 bit mode and can be exited in any mode. The documented routines that can be called must be entered in native 8 bit mode.

An example source file with 3 USR routines is provided in the Merlin 16 file SOURCE/USR.EXAMPLE.S.

## XC (eXtended 65C02, 65802 and 65816 opCodes)

```
XC
   XC                      [ enable the 65C02 option ]
   XC (twice in a row)     [ enable the 65802/65816 option ]
```

NOTE: On Merlin 8, if XC is used at the beginning of the listing, the 65C02 opcodes are enabled. If XC is used twice, that is, if it is used on the first two lines of the listing, the 65802/65816 codes can also be assembled.

On Merlin 8, some of the 65802 long addressing codes are not enabled since they have no application on the 65802. In Merlin 16, all 65816 opcodes are enabled. *With Merlin 16, you will not have to use the XC pseudo-ops unless you have altered the PARMS file to require their use.*

The XC pseudo-op will not enable the extended BIT opcodes used on the Rockwell 65C02 chip. There is, however, a macro library file included on the Merlin disk that can be USEd to implement these additional codes.

To use Sourceror to disassemble 65C02 code with the older (unenhanced) IIe ROMs, you must first BRUN MON.65C02. See the section on Sourceror for details. This utility is not needed with the newer IIe (enhanced) or IIc (Unidisk 3.5 compatible) ROMs.

Whether you are using the ProDOS or DOS 3.3 version of Merlin 8, you *must* use the XC opcode as the very first line in your code. This serves as a flag to tell Merlin 8 that you are using the 65C02 or 65802 opcodes.

You may wonder why the XC opcode is needed. After all, if simply using it on a line within a source listing enables the extended opcodes within Merlin 8/16, surely the ability to assemble the opcodes are there all along. Why burden the user with an extra requirement? The reason is in the interest of efficient de-bugging and ultimately, your sanity. Merlin 8 does its best to alert you to possible errors in a source listing, but what happens if you use 65C02 opcodes on the older 6502 microprocessor? The 6502 will perform quite unpredictably, and yet Merlin 8 can't tell what system your program ultimately is destined for, so an error is not necessarily in order.

The solution is to make the programmer deliberately set a flag signifying that he knows he's using the extended codes. That way you're less likely to get in the habit of using codes like INC (Increment accumulator directly, available on the 65C02), and then accidentally use the same opcode on a 6502.

# MACROS

## WHY MACROS?

Macros represent a shorthand method of programming that allows multiple lines of code to be generated from a single statement, or macro call. They can be used as a simple means to eliminate repetitive entry of frequently used program segments, or they can be used to generate complex portions of code that the programmer may not even understand.

Examples of the first type of macro call are presented throughout this manual and in the files called MACROS.S and MACROS.816.S on the Merlin 16 disk, MACROS.S on the Merlin 8 ProDOS disk, and T.MACRO LIBRARY on the Merlin 8 DOS 3.3 disk. Examples of the second, more complex type, can be found in the FPMACROS.S on the Merlin 8 ProDOS disk and in the T.FP MACROS and T.RWTS MACROS libraries found on the DOS 3.3 disk.

Macros can also be used to simulate opcodes from other microprocessors such as the Rockwell 65C02 extended bit-related opcodes, as shown in the ROCKWELL.S file on the Merlin 8 ProDOS disk and the T.ROCKWELL MACROS file on the DOS 3.3 disk.

Macros literally allow you to write your own language and then turn that language into machine code with just a few lines of source code. Some people even take great pride in how many bytes of source code they can generate with a single macro call.

# MACRO PSEUDO OPS

### MAC (begin MACro definition)

Label MAC

This signals the start of a macro definition. It must be labeled with the macro name. The name is then reserved and cannot be referenced by anything other than that macro pseudo-op. For example, DA NAME will not be accepted if NAME is the label assigned to MAC.

### EOM (End of Macro
### <<< (Alternate form)

EOM
<<< (alternate syntax)

This signals the end of the definition of a macro. It may be labeled and used for branches to the end of a macro.

**PMC (Put Macro Call)**
**>>> (alternate form)**

PMC macro-name
>>> macro-name (alternate syntax #1)
macro-name (alternate syntax #2)

This instructs the assembler to assemble a copy of the named macro at the present location. It may be labeled.

## HOW A MACRO WORKS

A macro is simply a user-named sequence of assembly language statements. To create the macro, you indicate the beginning of a definition with the macro name in the label field, followed by the definition of the macro itself.

The macro definition ends with a terminator command in the opcode field of either EOM or the alternate form (<<<).

For example, suppose in your program that locations $06 and $07 need to be incremented by one, as in this listing:

```
1   INCR  INC  $06    ; INCREMENT LO BYTE
2         BNE  DONE
3         INC  $07    ; INCREMENT HIGH BYTE
4   DONE  ???         ; PROGRAM CONTINUES HERE...
```

If this is to be done a number of different times throughout the program, you could make the operation a subroutine, and JSR to it, or you could write the three lines of code every time you need it.

However, a macro could be defined to do the same thing like this:

```
1   INK   MAC         ; DEFINE A MACRO NAMED INK
2         INC  $06
3         BNE  DONE
4         INC  $07
5   DONE              ; NO OPCODE NEEDED
6         <<<         ; THIS SIGNALS THE END OF THE MACRO
```

Now whenever you want to increment bytes $06,07 in your program, you could just use the macro call:

```
100       INK         ; use the macro "INK"
```

You could also use either of these alternate forms:

```
100         PMC  INK   ; alternate form of macro call
```

or:

```
100         >>>  INK   ; alternate form of macro call
```

Now, suppose you notice that there are a number of different byte-pair locations that get incremented throughout your program. Do you have to write a macro for each one? Wouldn't it be nice if there was a way to include a variable within a macro definition? You could then define the macro in a general way, and when you use it, via a macro call, "fill in the blanks" left when you defined it. Here's a new example:

```
1   INK   MAC        ; define a macro named INK
2         INC  ]1    ; increment 1st location
3         BNE  DONE
4         INC  ]1+1  ; increment location + 1
5   DONE             ; NO OPCODE NEEDED
6         <<<        ; this signals the end of the macro
```

This can now be called in a program with the statement:

```
100         INK  $06
```

In the assembled object code, this would be assembled as:

```
100         INC  $06
100         BNE  DONE
100         INC  $07
100   DONE          ; NO OPCODE NEEDED
```

Notice that during the assembly, all the object code generated within the macro is listed with the same line number. Don't worry though, the bytes are being placed properly in memory, as will be evidenced by the addresses printed to the left in the actual assembly.

Later, if you need to increment locations $0A,0B, this would do the trick:

```
150         INK  $0A
```

In the assembled object code, this would be assembled as:

```
150         INC  $0A
150         BNE  DONE
150         INC  $0B
150   DONE            ; NO OPCODE NEEDED
```

As you can see, once a macro has been defined, you can use it just like any other assembler opcode.

Let's suppose you want to use several variables within a macro definition. No problem! Merlin 8/16 lets you use 8 variables within a macro, ]1 through ]8. Here's another example:

```
MOVE  MAC      ; define a macro named MOVE
      LDA ]1   ; load accum with variable ]1
      STA ]2   ; store accum in location ]2
      <<<      ; this signals the end of the macro
```

This is a macro that moves a byte or value from one location to another. In this example, the variables are ]1 and ]2. When you call the MOVE macro you provide a parameter list that "fills in" variables ]1 and ]2. What actually happens is that the assembler substitutes the parameters you provide at assembly time for the variables. The order of substitution is determined by the parameter's place in the parameter list and the location of the corresponding variable in the macro definition. Here's how MOVE would be called and then filled in:

```
MOVE  $00;$01
```

MOVE: macro being called
$00: takes place of ]1 (1st variable)
$01: takes place of ]2 (2nd variable)

Then, the macro will be expanded into assembly code:

```
MOVE  $00;$01
LDA   $00          {$00 in place of ]1}
STA   $01          {$01 in place of ]2}
```

It is very important to realize that *anything* used in the parameter list will be substituted for the variables. For example,

```
MOVE  #"A";DATA
```

would result in the following:

```
MOVE  #"A";DATA
LDA   #"A"
STA   DATA
```

You can get even fancier if you like:

```
MOVE  #"A";(STRING),Y
LDA   #"A"
STA   (STRING),Y
```

As illustrated, the substitution of the user supplied parameters for the variables is quite literal. It is also possible to get into trouble this way, but Merlin 8/16 will inform you with an error message if you get too carried away.

One common problem encountered is forgetting the difference between immediate mode *numbers* and *addresses*. The following two macro calls will do quite different things:

```
MOVE  10;20
MOVE  #10;#20
```

The first stores the contents of memory location 10 (decimal) into memory location 20 (decimal). The second macro call will attempt to store the *number* 10 (decimal) in the *number* 20! What has happened here is that an illegal addressing mode was attempted. If it were possible, the illegal macro call would have been expanded into something like this:

```
          MOVE  #10;#20        ; call the MOVE macro
          LDA   #10            ; nothing wrong here
          STA   #20            ; oops! can't do this!
*** BAD ADDRESS MODE ***       ; Merlin will let you know!
```

In order to use the macros provided with Merlin, or to write your own, study the macro in question and try to visualize how the required parameters would be substituted.

The number of values must match the number of variables used in the macro definition. A BAD VARIABLE error will be generated if the number of values is less than the number of variables used. No error message will be generated, however, if there are more values than variables.

Note that in giving the parameter list, the Macro is followed by a space, and then each parameter separated with a semicolon. When used in the opcode column, the macro name cannot be the same as any regular opcode or pseudo opcode, such as LDA, STA, ORG, EXP, etc. Also, it cannot begin with the letters DEND or POPD.

The PMC and >>> forms of a macro call are not subject to the above restrictions. In that case, the macro name will be in the operand column, and a comma is usually used to separate the macro from the parameter list. For example,

```
>>> MOVE,#10;#20
```

The assembler will accept some other characters in place of the comma between the macro name and the expressions in a macro call. You may use any of these characters:

. / , - ( and the space character

*The semicolons are required*, however, between the expressions, and no extra spaces are allowed.

NOTE: When the assembler sees a macro name in the opcode field like FIND, it first looks to see if there is a macro defined by that name. If, for example, the needed macro library was not included with the USE function, or wasn't defined at the beginning of the source listing, and thus the macro was not found, then the first three characters (FIN) are taken as the opcode. If this is a legal opcode or pseudo opcode, and in this example FIN is, then it is treated as such, and no error is generated. This can be a

source of confusion, but fortunately there are few such potential conflicts in the macro definitions. This can also be avoided by using, for example, an underscore as the first character of a macro, as in _FindControl.

Macros will accept literal data. Thus the assembler will accept the following type of macro call:

```
MUV  MAC              ;    MACRO DEFINITION
     LDA  ]1
     STA  ]2
     <<<

     MUV  (PNTR),Y;DEST
     MUV  #3;FLAG,X
```

with the resultant code from the above two macro calls being:

```
MUV  (PNTR),Y;DEST      ; macro call
LDA  (PNTR),Y           ; substitute first parm
STA  DEST               ; substitute second parm
```

and,

```
MUV  #3;FLAG,X          ; macro call
LDA  #3                 ; substitute first parm
STA  FLAG,X             ; substitute second parm
```

Variables passed can be used as the operand to pseudo-ops like ASC:

```
MACRO DEFINITION          RESULTANT CODE EXAMPLE

PRINT  MAC                PRINT "Example"
       JSR  SENDMSG       JSR   SENDMSG
       ASC  ]1            ASC   "Example"
       BRK                BRK
       <<<
```

Some additional examples of the PRINT macro call:

```
PRINT  !"quote"!
PRINT  'This is an example'
PRINT  "So's this, understand?"
```

**NOTE:** If such strings contain spaces or semicolons, they *must* be delimited by single or double quotes. Also, literals in macros such as PRINT "A" must have the final delimiter. This is only true in macro calls or VAR statements, but it is good practice in all cases.

## MORE ABOUT DEFINING A MACRO

A macro definition begins with the line:

```
Name    MAC     (no operand)
```

with Name in the label field. Its definition is terminated by the pseudo-op EOM or <<<. The label you use as Name cannot be referenced by anything other than a valid macro call: NAME, PMC NAME or >>> NAME.

Forward reference to a macro definition is not possible, and would result in a NOT MACRO error message. That is, the macro must be defined before it is called by NAME, PMC or >>>.

The conditionals DO, IF, ELSE and FIN may be used within a macro.

Although it is possible to define and invoke a macro at the same time, it is not recommended. The more common approach is to turn off assembly, define the macros to be used later, then turn assembly back on. The DO 0 ... FIN conditional assembly opcodes are used to enclose the macro definitions as follows:

```
****************** * SAMPLE PROGRAM * ******************
*
          DO 0              ;TURN OFF ASSEMBLY
*
ERROR     MAC               ;GIVE THE MACRO A NAME
          LDA #$88          ;ASCII CODE FOR CTRL-G (BELL)
          JSR COUT          ;PRINT TO SCREEN (CAUSES A BEEP)
          JSR COUT          ;DO IT AGAIN
          JSR COUT          ;ONE MORE TIME
          <<<               ;END OF MACRO *
          FIN               ;TURN ASSEMBLY BACK ON * * PROGRAM CONTINUES HERE ...
```

You can also give the EOM or <<< opcode a label so you could branch to it:

```
****************** * SAMPLE PROGRAM * ******************
*
          DO 0              ;TURN OFF ASSEMBLY
*
ERROR     MAC               ;GIVE THE MACRO A NAME
          LDA #$88          ;ASCII CODE FOR CTRL-G (BELL)
          LDY #$04
ERROR1    DEY               ;BEGIN COUNTDOWN
          BEQ FINISH        ;IF Y = 0 THEN EXIT
          JSR COUT          ;BEEP THE SPEAKER
          JMP ERROR1        ;GO BACK FOR ANOTHER
FINISH    <<<               ;END OF MACRO
*
          FIN               ;TURN ASSEMBLY BACK ON
*
* PROGRAM CONTINUES HERE ...
```

Labels inside macros are updated each time the macro NAME, PMC or >>> NAME is encountered. Error messages generated by errors in macros usually abort assembly because of possibly harmful effects.

**NOTE:** Such messages will usually indicate the line number of the macro call rather than the line inside the macro where the error occurred. Thus, if you get an error on a line in which a macro has been used, you should check the macro definition itself for the offending statement.

## NESTED MACROS

Macros may be nested to a depth of 15. Here is an example of a nested macro in which the definition itself is nested. This can only be done when both definitions end at the same place.

```
TRDB MAC
     TR  ]1+1;]2+1
TR   MAC
     LDA ]1
     STA ]2
     <<<
```

In this example TR LOC;DEST will assemble as:

```
     LDA  LOC
     STA  DEST
```

and TRDB LOC;DEST will assemble as:

```
     LDA  LOC+1
     STA  DEST+1
     LDA  LOC
     STA  DEST
```

A more common form of nesting is illustrated by these two macro definitions:

```
CH   EQU  $24
POKE MAC
     LDA  #]2
     STA  ]1
     <<<
HTAB MAC
     POKE  CH;]1
     <<<
```

The HTAB macro could then be used like this:

```
     HTAB 20            ; htab to column 20 decimal
```

and would generate the following code:

```
LDA  #20          ; ]2 in POKE macro
STA  CH           ; ]1 in POKE macro, 1st parm
                  ; in HTAB macro
```

### Flexible Variable Lists (Merlin 16 only)

Merlin 16 supports an additional macro variable, ]0, which returns the number of variables in the parameter list of the macro call. This lets you create macros with a flexible input. For example, here's a macro that uses the number of input variables to decide whether to store a value just pulled off the stack:

```
PullByte  MAC          ; MACRO DEFINITION
          PLA          ; PULL BYTE (OR WORD) OFF STACK
          DO   ]0      ; IF A LABEL IS GIVEN
          STA  ]1      ; STORE VALUE FROM STACK IN ]1
          FIN          ; END OF CONDITIONAL PART
          EOM          ; END OF MACRO DEFINITION
```

Thus, the macro call:

```
PullByte
```

could be used to pull a value off the stack and leave it in the accumulator, whereas

```
PullByte  LABEL
```

would pull the value off the stack and then store it in location LABEL. You could even get more fancy by adding the IF MX tests to see whether one or two PLAs and STAs were needed to get a two-byte word off the stack in the 8 bit mode, as opposed to a single PLA/STA pair in the 16 bit mode.

## MACRO LIBRARIES AND THE USE PSEUDO OP

There are a number of macro libraries on the Merlin 8/16 disks. These libraries are examples of how one could set up a library of often used macros.

The requirements for a file to be considered a macro library are:

1) Only Macro definitions and label definitions exist in the file.
2) The file is a text file.
3) If it is a DOS 3.3 library, the file name must be prefixed with "T."
4) The file must be accessable at assembly time, i.e. it must be in an active disk drive.

The macro libraries included with Merlin 8/16 include:

| DOS 3.3 | ProDOS | Macro Libary functions |
|---|---|---|
| T.MACRO LIBRARY | MACROS.S | Often used macros for general use. |
| | MACROS.816.S | General use macros for 65816 programs. |
| | TOOL.MACROS | Directory of IIgs Toolbox macros. Must be used with MACROS.816.S. |
| T.FPMACROS | FPMACROS.S | Allow easy access to Applesoft floating point math routines. |
| T.OUTPUT | <none> | To be used with SENDMSG. |
| T.PRDEC | PRDEC.S | Prints A,X in decimal. |
| T.ROCKWELL MACROS | ROCKWELL.S | Implements extended bit related opcodes on the Rockwell 65C02. |
| T.RWTS | <none> | Allow easy access to DOS 3.3's RWTS disk routines. |
| <none> | TOOL.EQUATES | Directory of IIgs label equates for use with toolset data structures. |

Any of these macro libraries may be included in an assembly by simply including a USE pseudo op with the appropriate library name. There is no limit to the number of libraries that may be in memory at any one time, except for available memory space. See the documentation on the USE pseudo op for a discussion on its use in a program.

# THE MERLIN 8/16 LINKERS

## WHY A LINKER?

The linking facilities built into Merlin offer a number of advantages over assemblers without this capability:

1) Extremely large programs may be assembled in one operation.

2) Large programs may be assembled much more quickly with a corresponding decrease in development time.

3) Libraries of subroutines, i.e. for disk access, graphics, screen/modem/printer drivers, etc., may be developed and linked to any Merlin program.

4) Programs may be quickly re-assembled to run at any address.

With a linker, you can write portions of code that perform specific tasks, such as general disk I/O handler, and perform whatever testing and debugging is required. When the code is correct, it is assembled as a REL file and placed on a disk. Whenever you need to write a program that uses disk I/O you won't have to re-write or re-assemble the disk I/O portion of your new program. Just link your general disk I/O handler to your new program and away you go. This technique can be used for a variety of often-used subroutines.

Wouldn't a PUT file or macro USES library serve the same purpose? A PUT file comes the closest to duplicating the utility of REL files and the linker, but there are a few rather large drawbacks for certain programs. First, using a PUT file to add a general purpose subroutine would result in slower assembly because the entire program has to be assembled even when changes are made only to the subroutine. Second, any label definitions contained in the PUT file would be global within the entire program. This means the person writing the subroutine would have to be careful not to use a label like LOOP that might occur in one of the other modules, or in the main program itself. With a REL file, only labels defined as ENTry in the REL file, and EXTernal in the current file, would be shared by both programs. There is no chance for duplicate label errors when using the Linker. Consider the following simple example:

A REL file has been assembled that drives a plotter. There are six entry points into the driver: PENUP, PENDOWN, NORTH, SOUTH, EAST, WEST. To further illustrate the value of a linker, assume the driver was written by a friend who has moved 2000 miles from you. Your job is to write a simple program to draw a box.

The code would look something like this:

```
 1              REL                    ;RELOCATABLE CODE
 2 PENUP        EXT                    ;EXTERNAL LABEL
 3 PENDOWN      EXT                    ;ANOTHER ONE
 4 NORTH        EXT
 5 SOUTH        EXT
 6 EAST         EXT
 7 WEST         EXT
 8
 9 BOX          LDY  #00               ; INITIALIZE Y
10              JSR  PENDOWN           ; GET READY TO DRAW
11 LOOP         JSR  NORTH             ; MOVE UP
12              INY                    ; INC COUNTER
13              CPY  #100              ; 100 MOVES YET?
14              BNE  LOOP              ; NOTICE LOCAL LABEL
15              LDY  #00               ; INIT Y AGAIN
16 LOOP2        JSR  EAST              ; NOW MOVE TO RIGHT
17              INY
18              CPY  #100
19              BNE  LOOP2             ; FINISH MOVING RIGHT
20 * YOU GET THE IDEA, DO SOUTH, THEN WEST, AND DONE!
```

This simple sample program illustrates some of the power of RELocatable, linked files. Your program doesn't have to concern itself with conflicts between its labels and the REL files labels, you don't concern yourself with the location of the EXTernal labels, your program listing is only 30 to 40 lines and it is capable of drawing a box on a plotter. Also, notice that you are free to use the label LOOP because it is *local* to your module, and will not be known to any of the other modules.

In addition, changes to your module will not require re-assembly of the plotter driver. In short, with REL files and a linker, changes to large programs can be made quickly and efficiently, greatly speeding the program development process.

## ABOUT THE LINKER DOCUMENTATION

There are three pseudo opcodes that deal directly with relocatable modules and the linking process. These are:

REL - instructs the assembler to generate relocatable files.
EXT - defines a label as external to the current file.
ENT - defines a label in the current file as accessable to other REL files.

There are two other pseudo opcodes that behave differently when used in a REL file, relative to a normal file. These are:

DS - define Storage opcode.
ERR - force an ERRor opcode.

Each of these five pseudo opcodes will be defined or redefined in this section as they relate to REL files. Also, an Editor command unique to REL files will be defined: LINK.

In order to use the Linker, the files to be linked must be specifed. The Linker uses a file containing the names of the files to be linked for this purpose. The format of this *linker name file* differs from DOS 3.3 and ProDOS. These differences will be illustrated here.

# PSEUDO OPCODES FOR USE WITH RELOCATABLE CODE FILES

**REL (generate a RELocatable code file)**

REL [ only options for this opcode ]

This opcode instructs the assembler to generate a relocatable code file for subsequent use with the relocating Linker.

This *must* occur prior to definition of any labels. You will get a BAD REL error if not. REL files are not compatible with the SAV pseudo-op and with the Main Menu's Save Object Code command. *To get an object file to the disk you must use the* DSK *opcode for direct assembly to disk.*

There are additional illegal opcodes and procedures that are normally allowed with standard files, but not with REL files. For example, an ORG at the start of the code is not allowed. The ORG address is specified at link time. A further restriction on REL files is that multiplication, division or logical operations can be appled to absolute expressions, but not to relative ones.

Examples of absolute expressions are:

- An EQUate to an explicit address
- The difference between two relative labels
- Labels defined in DUMMY code sections

Examples of relative expressions that are not allowed are:

- Ordinary labels
- Expressions that utilize the current Program Counter (PC), like: LABEL = *

The initial reference address of a REL file is $8000. Note that this is only a fictional address, since it will later be changed by the Linker. It is for this reason that no ORG opcode is allowed.

There are some restrictions with the Merlin 8 Linker involving use of EXTernal labels in operand expressions. No operand can contain more than one external. For operands of the following form:

```
   #>expression
or
   >expression
```

where the expression contains an external, the value of the expression must be within 7 bytes of the external labels' value.

For example:

```
LDA #>EXTERNAL+8        [ illegal expression ]
DFB >EXTERNAL-1         [ legal expression ]
```

Object files generated with the REL opcode are given the file type LNK under ProDOS. This is the type that will show if the disk is cataloged by Merlin 8/16. This type is file type $F8. These restrictions do not apply to the Merlin 16 Linkers.

**EXT (define a label EXTernal to the current REL module)**

```
label EXT
      EXT label1, label2, etc.
PRINT EXT                    [ define label PRINT as EXT ]
      EXT LABEL1, LABEL2 [ define LABEL1 and LABEL2 and entries - Merlin 16 only ]
```

This defines the label in the label column as an external label. This means that references to this label within the source will assume the value for LABEL is an as-yet undefined address, presumably found in another module that will be ultimately linked with this source file. Any external label must be defined as an ENTry label in its own REL module, otherwise it will not be reconciled by the Linker since the label would not have been found in any of the other linked modules. The EXTernal and ENTry label concepts are what allows REL modules to communicate and use each other as subroutines, etc.

The value of the label is set to $8000 and will be resolved by the Linker. In the symbol table listing, the value of an external will be $8000 plus the external reference number ($0-$FE) and the symbol will be flagged with an X.

In Merlin 16, the EXT and ENT opcodes accept the following syntax:

```
ENT LABEL1, LABEL2, LABEL3
```

This makes it possible to declare absolute symbols as entries. Thus, if LABEL1 was an equate instead of a location in the code, then it can still be used as an external by other modules. Thus, it does not have to be equated in all the files using it. For example, if all three modules in a linked system used the labels HOME, COUT and BELL, the first module could define these labels with the usual HOME EQU $FC58, etc. equates, and then use ENT HOME, COUT, BELL to make these equates available to the other modules being linked. This would avoid having to define the labels in each of the other modules.

With this particular syntax, you *must not* use a label in the label column. That will cause the assembler to assume you are using the more usual syntax.

**NOTE:** Using this function instead of putting all the common equates in all the source files, most easily accomplished by a common USE file, does take more space in the Linker symbol library, and hastens the time of a memory overflow.

### ENT (define a label as an ENTry label in a REL code module)

```
label ENT
      ENT  label1, label2, etc.
PRINT ENT                          [ define label PRINT as ENTry ]
      ENT LABEL1, LABEL2  [ define LABEL1 and LABEL2 and entries - Merlin 16 only ]
```

This defines the label in the label column as an ENTry label. This means that the label can be referred to by an EXTernal label in another source file somewhere. This facility allows other REL modules to use the label as if it were part of their source file. If a label is meant to be made available to other REL modules it must be defined with the ENT opcode, otherwise other modules wouldn't know it existed and the Linker would not be able to reconcile it.

The following example of a REL module segment illustrates the use of this opcode:

```
21            STA  POINTER        ;some meaningless code
22            INC  POINTER        ;for our example
23            BNE  SWAP           ;CAN BE USED AS NORMAL
24            JMP  CONTINUE
25  SWAP      ENT                 ;MUST BE DEFINED IN THE
26            LDA  POINTER        ;CODE PORTION OF THE
27            STA  PTR            ;MODULE AND NOT USED
28            LDA  POINTER+1      ;AS AN EQUated label
29            STA  PTR+1
30 * etc.
```

Note that the label SWAP is associated with the code in line 26 and that the label may be used just like any other label in a program. It can be branched to, jumped to, used as a subroutine, etc.

ENT labels will be flagged in the symbol table listing with an E.

## DS (Define Storage)

```
DS \
DS \expression
   DS \                    [ skip to next REL file, fill mem with zeros to next
                             page break ]
   DS \1                   [ skip to next REL file, fill mem with the value 1 to
                             next page ]
```

When this opcode is found in an REL file, it causes the Linker to load the next file in the *linker name file* at the first available page boundary, and to fill memory either with zeros or with the value specified by the expression. If used, this opcode should only be placed at the *end* of your source file. Notice that DS *expression* , for example DS 5, still has the usual function even in REL files.

## ERR (force an ERRor)

```
ERR \expression
   ERR \$4200              [ error if current code passes address $4200 ]
```

This opcode will instruct the Linker to check that the last byte of the current file does not extend to "expression" or beyond. The expression must be absoulute and not a relative expression.

If the Linker finds this is not the case, linking will abort with the message: CONSTRAINT ERROR: followed by the value of the expression in the ERR opcode.

The position of this opcode in a REL file has no bearing on its action. It is recommended that it be put at the end of a file.

You can see how this works by trying to link the sample PI files on the Merlin 8/16 disks to an address greater than $81C.

# THE MERLIN 8 LINKER

In Merlin 8, linking is done by using the LINK command in the Command Mode to specify the address at which the final file will be loaded, i.e. the non-linker equivalent to the ORG function, and the name of the file which contains a list of the REL files created by the individual assemblies of the files that make up the complete application. The Linker is part of the Editor/Assembler system, and is available at any time with the LINK command.

The general use of the LINK command looks like this:

```
LINK $1000 "NAMES          [ link files in NAMES - DOS 3.3 ]
LINK $2000 "/VOL/NAMES"    [ link files in NAMES - ProDOS ]
```

The LINK command invokes the linking loader. For example, suppose you want to link the object files whose names are held in a "linker name file" called NAMES. Suppose the start address desired for the linked program is $1000. Then you would type: LINK $1000 "NAMES" and press Return. If you are using the ProDOS version, this assumes the prefix has been set. The final quote mark in the name is optional. You can use other delimiters such as the apostrophe (') or colon (;). The specified start address has no effect on the space available to the Linker.

To provide space for the Linker, any source file must be removed from memory with the NEW command. This command is only accepted if there is no source file in memory.

## LINKER NAME FILES (DOS 3.3)

The linker name file is just a text file containing the file names of the REL object modules to be linked. It should be created with the Merlin editor and written to the disk with the Write Text File from the Main Menu. Remember to type a space to start the filename for the W command if you don't want the T. prefix appended to the start of the filename. Thus, if you want to link the object files named MYPROG.START, MYPROG.MID, and LIB.ROUTINE,D2, you would create a text file with these lines:

```
MYPROG.START
MYPROG.MID
LIB.ROUTINE,D2
```

In this example, you would write this to disk using the W command under the filename MYPROG.NAMES. You can use any filename you wish here; it is not required to call it NAMES. Then you would link these files with a start address of $1000 by typing NEW and then issuing the editor command as follows:

```
LINK $1000 "MYPROG.NAMES"
```

The Linker will not save the object file it creates. Instead, it sets up the object file pointers for the Main Menu Save Object Code command and returns directly to the Main Menu upon the completion of the linking process.

## LINKER NAME FILES (ProDOS)

The linker name file is just a specially formatted file containing the pathnames of the LNK files to be linked. This file is most easily created by assembling a source file with the proper format, as follows:

Each pathname in the source file should be given the form STR "pathname",00

NOTE: The 00 must be include at the end. The entire source file must end with a BRK, i.e. another 00. This tells the Linker that there are no more pathnames in the file. Thus, if you want to link the LNK files names /MYDISK/START, /MYDISK/MID, AND /OTHERDISK/END, you would make a source file containing these lines:

```
STR  "/MYDISK/START,00
STR  "/MYDISK/MID",00
STR  "/OTHERDISK/END",00
BRK
```

It is best to use full pathnames as shown, but this is not required. You should then assemble this file and save the object code as, for example, /MY DISK/MYPROG/NAMES. You can use any pathname you want here; it is not necessary to have NAMES in a subdirectory nor to call it NAMES. Then you can link these files to address $803 by typing NEW and then:

```
LINK $803 "/MYDISK/MYPROG/NAMES"
```

The file type used by the Save Object Code command is always the file type used in the last assembly. Thus it is BIN unless the last assembly had a TYP opcode and then it will be that type. This will be used by the Save Object Code command after you link a group of files. That is, the Linker does not change this type. If you make a mistake and the file gets saved under a type you did not want, just assemble an empty file, which would reset the object type to BIN ($06). You will, however, have to link the files again.

## THE LINKING PROCESS

Various error messages may be sent during the linking process See the ERRORS section of this manual for more information. If a DOS error occurs involving the file loading, then that error message will be seen and linking will abort. If the DOS error FILE TYPE MISMATCH occurs after the message "Externals:" has been printed then it is being sent by the Linker and means that the file structure of one of the files is incorrect and the linking cannot be done.

The messsage MEMORY IN USE may occur for two reasons. Either the object program is too large to accept, i.e. the total object size of the linked file cannot exceed about $A100, or the linking dictionary has exceeded its allotted space, i.e. it is greater than $B000 in length. Each of these possibilities is exceedingly remote.

After all files have been loaded, the externals will be resolved. Each external label referenced will be printed to the screen and will be indicated to have been resolved or not resolved. An indication is also given if an external reference corresponds to duplicate entry symbols. With both of these errors, the address of the one or two byte field effected is printed. This is the address the field will have when the final code is BLOADed.

If you use the TRON command prior to the LINK command, only the errors will be printed in the external list, i.e. NOT RESOLVED and DUPLICATE errors.

This listing may be stopped at any point using the space bar. The space bar may also be used to single step through the list. If you press the space bar while the files are loading then the Linker will pause right after resolving the first external reference.

The list can be sent to a printer by using the PRTR command prior to the LINK command. At the end, the total number of errors, i.e. external references not resolved and references to duplicate entry symbols, will be printed. After pressing a key, you will be sent to the Main Menu and can save the linked object file with the Save Object Code command, using any desired filename or pathname. You can also return to the Editor and use the GET command to move the linked code to main memory.

## USING MERLIN 8 LINKED FILES

The following example shows how two source files are used to generate the LNK files that will be combined by the Linker into a final application:

```
*********************************
*   RELOCATING LINKER SAMPLE    *
*         PART ONE              *
*********************************

         REL
         DSK   FILE1.L

HOME     EQU   $FC58
COUT     EQU   $FDED

BEGIN2   EXT               ; EXTERNAL LABEL

BEGIN    JSR   HOME        ; CLEAR SCREEN

         LDX   #$00        ; INITIALIZE COUNTER
LOOP     LDA   STRING,X    ; GET CHARACTER
         BEQ   DONE        ; END OF STRING
         JSR   COUT
         INX
         BNE   LOOP        ; ALWAYS

DONE     JMP   BEGIN2      ; JUMP TO 2ND SEGMENT...

STRING   ASC   "THIS IS FILE #1"
         HEX   8D,00       ; END OF STRING
         LST   OFF
```

And this is the second part:

```
*********************************
*   RELOCATING LINKER SAMPLE    *
*         PART TWO              *
*********************************

         REL
         DSK   FILE2.L

HOME     EQU   $FC58
COUT     EQU   $FDED

BEGIN2   ENT               ; ENTRY LABEL
         LDX   #$00        ; INITIALIZE COUNTER
LOOP     LDA   STRING,X    ; GET CHARACTER
         BEQ   DONE        ; END OF STRING
         JSR   COUT
```

```
         INX
         BNE   LOOP        ; ALWAYS

DONE     RTS               ; END OF PROGRAM

STRING   ASC   "THIS IS FILE #2"
         HEX   8D,00       ; END OF STRING
         LST   OFF
```

Having entered and assembled each of the source files, FILE1.L and FILE2.L will be created on the disk. These are the intermediate REL files that will be linked to create the final application program. Now you need to create the file containing the list of files to be linked. In this example, the file will be called NAMES and it will link the FILE1.L and FILE2.L files.

For the DOS 3.3 version of Merlin 8, you would just use the editor to type in the lines:

```
FILE1.L
FILE2.L
```

There are no leading spaces; the names FILE1.L and FILE2.L go in the *label* column of the Editor. This file is then saved as a text file using the Write Text File command from the Main Menu. Remember to add a space at the beginning of the filename to save under if you wish to avoid the T. prefix in the name.

To link the file, type NEW to clear the source workspace, then type LINK $8000 "NAMES" and press Return. The file NAMES will be read to determine which files to link. If there are no errors, you will be returned to the Main Menu when the link is complete. At that point, use the Save Object Code command to save the object file to disk under the name FINAL.OBJ.

To test the program, use this Applesoft BASIC program:

```
10 TEXT: HOME
20 PRINT CHR$(4);"BLOAD FINAL.OBJ"
30 CALL 32768: REM $8000
40 VTAB 12: HTAB 15: PRINT "IT REALLY WORKS!"
50 LIST: END
```

To create the names file for the ProDOS version of Merlin 8, you will need to create a separate source file for the names list. This is because the ProDOS Linker requires that the names list be in a special format. Remember that each name in the ProDOS Merlin 8 names list must be defined with the STR pseudo-op, terminated with a zero, and that the list itself is terminated with a zero also. To create the names list, type in this text:

```
*********************************
* MERLIN 8 LINKER NAMES FILE    *
*           ProDOS              *
*********************************

         DSK  "NAMES"          ; CREATE NAMES FILE
         STR, "FILE1.L",00     ; 1ST LINK FILE
         STR  "FILE2.L",00     ; 2ND LINK FILE
         BRK                   ; ZERO TO TERMINATE LIST
```

Assemble this file, which will create the actual NAMES file for the Linker, and save the source file under the file name NAMES. This will actually be saved on the disk as NAMES.S, so you needn't worry about any confusion when you use NAMES in the Linker.

To link the files under ProDOS, type NEW to clear the workspace, and then type LINK $8000 "NAMES" and press Return. When the link is complete, use the Save Object Code command at the Main Menu and save it under the name FINAL.OBJ. You can use the same Applesoft program shown for the DOS 3.3 example to test the program.

In looking at the example, notice how the ENT and EXT pseudo-ops are used to communicate the label BEGIN2 between the two programs. Also notice how there is no conflict over the use of the labels LOOP, DONE and STRING.

Compare this example to the sample program shown for the PUT directive. Notice how the same result of combining separate source files is achieved, but without the disadvantages of PUT files discussed at the beginning of this chapter.

# THE MERLIN 16 LINKERS

The Merlin 16 assembler supports three different linkers. These are separate and distinct from the built-in Linker in Merlin 8, and are as follows:

LINKER: This Linker combines multiple relocatable LNK files into a file that runs at a specified address, such as a ProDOS 8 BINary or SYStem type file. This is also referred to as the Absolute Linker, since the final output file must be run at a specified location.

LINKER.GS: This is a Linker specifically for creating Object Module Format (OMF) files to be run on the Apple IIgs under ProDOS 16 and the System Loader. On the Apple IIgs, applications have no way of knowing where in memory they will ultimately be loaded and run, and so must contain a relocation dictionary as part of the final output file. If you wish to write Apple IIgs ProDOS 16 programs, you should use Linker GS or Linker.XL, discussed next.

LINKER.XL: This is special version of Linker GS which makes two passes and links to disk to produce a multi-segment file. This feature is a trade-off with linking speed, that is, Linker XL is slower than Linker GS, and so you will normally want to use Linker GS unless you specifically require multi-segment files. A multi-segment file is where one program is broken up into multiple segments on the disk, but which ultimately all come together to form the final program in memory. This is different from a program which just happens to have other segments that it loads under its own control, such as printer drivers or program overlays.

Linker.GS is automatically loaded into memory when Merlin 16 is run, and so is available for immediate use. You can manually set up either of the other two Linkers by just typing -LINKER or -LINKER.XL as a Disk Command from the Main Menu. Alternatively, you can also change the PARMS.S file to load any version of the Linker that you wish. See the discussion of the PARMS file in the Technical Information section.

The Linkers use the same space as USER programs. LNK files that you may have created on Merlin Pro, an earlier version of Merlin 8, should be upward compatible as long as they do not have externals or entry declarations. If your programs have external or entry declarations, or you are in doubt or encounter difficulties, just re-assemble the source files with Merlin 16 to create updated LNK files.

As was discussed for the Merlin 8 Linker, creating an output file consists of several steps. First, source files are created using the Merlin editor. These are assembled and the output file (type $F8 = LNK) is created using the REL and DSK or SAV directives. This intermediate file is not usable in and of itself. Rather, it is to be used as the input for the next step, which is the actual linking of several LNK files to create the final object file. This file may be a stand-alone BIN or SYS file for ProDOS 8, or it may be an OMF file for ProDOS 16, such as a SYS16, CDA (Classic Desk Accessory), or any other ProDOS 16 loadable file.

The linking process in Merlin 16 is controlled by a *linker command file*. The linker command files are a more advanced form of the Merlin 8 NAME files, and have much more flexibility. They support comments, are able to do batch assemblies before linking, and are able to create multiple output files. Let's look at each Merlin 16 Linker:

## THE ABSOLUTE LINKER (LINKER)

To start the link, you must first delete any source file in memory to provide the memory for reading the Link command file. Remember to save the file first if necessary. To start the link, type Open-Apple-O to open the Command Box, and then type LINK "FILENAME" and press Return, where FILENAME is the name of the linker command file. The LINK command from the Merlin 16 editor has slightly different syntax from the Merlin 8 Linker in that you do not specify an address. Instead, the address is provided within the command file with an ORG directive.

NOTE: The command file is just a text file looking very much like a standard source file, *but you do not assemble it.*

The command file can have comments in the usual comment format for source files. Commands to the Linker are put in the opcode field. Commands supported are:

ASM, PUT, OVR, LNK (or LINK), ORG, ADR, SAV, TYP, EXT, ENT, DAT, LKV, END, and LIB.

These have the following syntax and meanings:

ASM pathname
ASM FILENAME.S

Assemble the source file specified in pathname. The source should do a DSK or SAV to create the LNK file to ultimately be used by the Linker. All ASMs must be done *before* any other linker commands. The ASM command is a conditional operation, and only assembles those source files that have been changed since the last time the files were linked. This is a convenience feature that lets you create a command file to build a final application. Re-assemblies will only be done on just those parts of the application that have changed since your last linking.

To determine whether to do an assembly, the ASM command checks bit 0 of the "auxtype" of the source file. It does not do the assembly if this set. Otherwise it sets that bit and does the assembly. You can defeat this by zeroing the appropriate bit in the PARMS file. See the Technical Information section for details. This bit is cleared whenever you save a source file, so this will force assembly of that file. Also see the PUT command below.

PUT pathname
PUT FILENAME.S

Check and set auxtype bit 0 of this file, presumably a PUT file in the next source. If the file has been modified then force the assembly of the next ASM instruction. This should precede the ASM command for any files that use a particular PUT file, and is used to cause a re-assembly in the event you change a PUT file used by a particular master file. This command is not needed if your source files do not use PUT files, or you don't wish to use the feature.

OVR [ALL]
OVR or OVR ALL

Override the auxtype 0-bit check and force assembly of the next ASM instruction. This flag is reset by any ASM. With the OVR ALL syntax, it will force assembly of all files in the linker list, so you don't have to use OVR before each ASM instruction if you want all assemblies to be done.

LNK pathname
LNK FILENAME.L

Link the LNK file specified. Generally you will have several of these in a row.

ORG hex address
ORG $2000

Sets the run time address of the following LNKs. Must be used between each SAV and the next group of LNKs. The address will also be put in the auxtype if no ADR command follows.

NOTE: The Linkers have only hexadecimal address calculation ability. All addresses must be in hex and preceded with the $ sign.

ADR address
ADR $2000

Sets the load address of the next linked file. Must be used only after an ORG setting the run time address. The ORG automatically sets this address, so you don't have to have an ADR command if it is the same as the ORG. The *load address* is the address put in the auxtype of the file. For non-BIN files it could have some other meaning.

SAV pathname
SAV FINAL.SYSTEM

Saves the linked file. This *must* be in the command file or there will be no resulting linked file. It should come after all the LNKs for a given output file. The Absolute Linker supports up to 64 separate output files.

SAV must *only* be used after one or more LNKs. It is not to be used to save the object code after an ASM. The assembly source should do this with the SAV or DSK command, or DSK if linking is to be done and the Linker is not just being used to do batch assemblies. All ASMs must precede any LNKs and SAVs.

TYP byte
TYP $06

Sets the file type for the next linked file. If all the SAVs are to use the same type output file, this need only be used once in the command file.

EXT
EXT

Tells the Linker to print addresses of all resolved externals and not only the ones with errors. This is turned off after each SAV.

ENT
ENT

Tells the Linker to print the entry list. This should come after all the linking of all output files.

DAT
DAT

Causes the Linker to print the current date and time.

END
END

Marks end of linker command file. Optional.

LIB directory name
LIB TOOL.LIBR

If used, this should come after all LNKs and before the last SAV. It tells the Linker to look for any unresolved (at that point) external labels and search the given directory for corresponding files. The files must be LNK files of the *same name* as the entry label to which they correspond. Any such files found will be linked to the present module. Not finding a file will not cause an error because some other file linked this way may contain the entry in question. If not, an error will result when the final external resolution is done. This feature could have been made automatic, but was not because that would substantially impair performance when it is not needed. Making it an option in the command file provides more versatility.

For example, suppose one of the linked files has the label PRINT declared as an external. The Linker comes to the line LIB LIBRARY, and suppose that at that point, none of the linked files has an entry called PRINT. The Linker will look in the directory LIBRARY for a file called PRINT, and if that file is found, it will be linked to the present module. Then the Linker will search for further unresolved externals, including those from the file PRINT just linked, and act on them in a similar way.

LKV byte
LKV $02

Verifies that the cortrect version of the Linker in us. LINKER is version 0, LINKER.GS is version 1, and LINKER.XL is version 2. For example, if you want to guarantee that LINKER.XL is the Linker in use, you would put the command LKV $02 in the linker command file.

## THE LINKER COMMAND FILE

The linker command file is a standard Merlin 16 source file, i.e. BUILD.S, *but it is not assembled* itself. Instead, the linker command file is executed with the LINK command.

A simple linker command file would look like this:

```
ORG  $2000     ; DEFINE LOAD ADDRESS
LNK  FILE.L    ; SPECIFY LNK FILE
SAV  FILE      ; SAVE THE OBJECT FILE
```

This would take the LNK file FILE.L and adjust all internal address references, i.e. JMPs, JSRs, etc. for a load address of $2000. The final object file would be saved on the disk under the name FILE.

The default filetype for SAV is BIN, i.e. type $06. Thus, if you were writing a SYStem file, you would have to add a TYP command to tell the Linker to save the final object file with the appropriate file type:

```
ORG  $2000     ; DEFINE LOAD ADDRESS
LNK  FILE.L    ; SPECIFY LNK FILE
TYP  $FF       ; SYSTEM FILETYPE
SAV  FILE      ; SAVE THE OBJECT FILE
```

If TYP is used, all succeeding SAVs, within a given Link operation, use the current TYP value.

ASM is added to a command file to automatically re-assemble a file that might have been changed without a re-assembly of a new copy of the corresponding LNK file:

```
ORG  $2000     ; DEFINE LOAD ADDRESS
ASM  FILE.S    ; RE-ASSEMBLE IF CHANGED
LNK  FILE.L    ; SPECIFY LNK FILE
TYP  $FF       ; SYSTEM FILETYPE
SAV  FILE      ; SAVE THE OBJECT FILE
```

## USING MERLIN 16 LINKED FILES

Although linked files may be of any filetype and are not restricted to use by BASIC, here's an example that combines the output from two assemblies to create an object file that is loaded and called from BASIC. Compare this to a similar PUT file example. This is the first part of the program:

```
*********************************
*   RELOCATING LINKER SAMPLE    *
*         PART ONE              *
*********************************

         REL
         DSK   FILE1.L

HOME     EQU   $FC58
COUT     EQU   $FDED

BEGIN2   EXT               ; EXTERNAL LABEL

BEGIN    JSR   HOME        ; CLEAR SCREEN

         LDX   #$00        ; INITIALIZE COUNTER
LOOP     LDA   STRING,X    ; GET CHARACTER
         BEQ   DONE        ; END OF STRING
         JSR   COUT
         INX
         BNE   LOOP        ; ALWAYS
```

```
DONE     JMP   BEGIN2     ; JUMP TO 2ND SEGMENT...

STRING   ASC   "THIS IS FILE #1"
         HEX   8D,00       ; END OF STRING
         LST   OFF
```

And this is the second part:

```
*********************************
*   RELOCATING LINKER SAMPLE    *
*         PART TWO              *
*********************************

         REL
         DSK   FILE2.L

HOME     EQU   $FC58
COUT     EQU   $FDED

BEGIN2   ENT               ; ENTRY LABEL
         LDX   #$00        ; INITIALIZE COUNTER
LOOP     LDA   STRING,X    ; GET CHARACTER
         BEQ   DONE        ; END OF STRING
         JSR   COUT
         INX
         BNE   LOOP        ; ALWAYS

DONE     RTS               ; END OF PROGRAM

STRING   ASC   "THIS IS FILE #2"
         HEX   8D,00       ; END OF STRING
         LST   OFF
```

Having entered and saved these source listings, you would then enter and save this Linker command file:

```
*********************************
* MERLIN 16 LINKER NAMES FILE   *
*********************************

         LKV   $00         ; OPTIONAL CHECK FOR "LINKER"

         ASM   PART1.S     ; ASSEMBLE IF NEEDED
         ASM   PART2.S     ; ASSEMBLE IF NEEDED

         ORG   $8000       ; SPECIFY LOAD ADDRESS

         LNK   FILE1.L     ; LINK FILE
         LNK   FILE2.L     ; LINK FILE

         TYP   $06         ; BINARY
         SAV   FINAL.OBJ
```

These are assembled and linked together by typing LINK "FILENAME" where FILENAME is whatever name the command file has been saved under on the disk. Before linking the command file, remember to clear the workspace with NEW from the Command Box.

The generated object file, FINAL.OBJ could be tested with this Applesoft program:

```
10 TEXT: HOME
20 PRINT CHR$(4);"BLOAD FINAL.OBJ"
30 CALL 32768: REM $8000
40 VTAB 12: HTAB 15: PRINT "IT REALLY WORKS!"
50 LIST: END
```

Things to notice in the linker command file are the way that all the assemblies are done first, before any use of ORG or LNK instructions. In general, all ASMs should be done at once, followed by ORG and all LNKs.

The ORG $8000 specifies the load address of the final object file. This is used when linking the two LNK files called FILE1.L and FILE2.L into the final output file called FINAL.OBJ.

Both ASM and the check for the proper version of Linker with LKV are optional, and are not specifically required for this example. The main reason for including LKV in a command file is to make sure that the proper Linker is used for linking a particular command file. If you are developing both ProDOS 8 and ProDOS 16 applications at the same time, it would be easy to accidentally have the *wrong* Linker in the machine when you were trying to link a file for the *other* operating system.

In looking at the example, notice how the ENT and EXT pseudo-ops are used to communicate the label BEGIN2 between the two programs. Also notice there is no conflict over the use of the lables LOOP, DONE and STRING.

Compare this example to the sample program show for the PUT directive. The same result of combining separate source files is achieved, but without the disadvantages of PUT files discussed at the beginning of this Chapter.

## MULTIPLE OUTPUT FILES

For applications running under ProDOS 8, you can use the Absolute Linker to generate separate output files during the linking process. The advantage is that each output file has access to the values of any entry point definitions in the other modules. Thus, if you wanted to write a program with other modules that needed to know the address of entry points within the main program, you could use the Absolute Linker to generate all the files at the same time.

For multiple output files, start each group of LNKs with an ORG and end it with a SAV. External references will be resolved between such groups by a second linker pass.

For example:

```
*********************************
* MERLIN 16 LINKER NAMES FILE   *
*   MULTIPLE OUTPUT SAMPLE      *
*********************************

        LKV   $00         ; OPTIONAL CHECK FOR "LINKER"

        ASM   PROG1A.S    ; ASSEMBLE IF NEEDED
        ASM   PROG1B.S    ; ASSEMBLE IF NEEDED

        ORG   $2000       ; SPECIFY LOAD ADDRESS

        LNK   FILE1A.L    ; LINK FILE
        LNK   FILE1B.L    ; LINK FILE

        TYP   $FF         ; FILETYPE = SYSTEM
        SAV   PROGRAM1    ; SAVE 1ST OUTPUT FILE

        ASM   PROG2A.S    ; ASSEMBLE IF NEEDED
        ASM   PROG2B.S    ; ASSEMBLE IF NEEDED

        ORG   $8000       ; SPECIFY LOAD ADDRESS

        LNK   FILE2A.L    ; LINK FILE
        LNK   FILE2B.L    ; LINK FILE

        TYP   $06         ; FILETYPE = BINARY
        SAV   OVERLAY     ; SAVE 2ND OUTPUT FILE
```

## THE GS LINKER (LINKER.GS)

The GS Linker, on the Merlin 16 disk as LINKER.GS, works in the same way as the Absolute Linker except that it creates OMF (Object Module Format) files for the Apple IIgs only. The ORG is accepted, but should not be used, since memory is assumed to be assigned on an "as-available" basisin the Apple IIgs. However, you can use the ORG if you want to specify a specific load address for an object file.

**NOTE:** For Linker GS, the default SAV type is S16. Only one output file is supported, and there is a maximum length of the final file of about 32K for the code portion and another 32K for the relocation dictionary. LINKER.XL does not have these restrictions.

The main reason for using the GS Linker will be for creating application programs to run under ProDOS 16. Such files cannot be called from an Applesoft program, since ProDOS 16 is not available to Applesoft. In addition, the file format is such that a relocating dictionary is included as part of the final output object file. Thus, BLOADing it and listing it in the Monitor would reveal a certain amount of additional data saved with the file.

In addition to the linker commands of the standard Linker, the GS Linker has the following commands:

VER (version)
VER $01

This is used to specify the version of the OMF, i.e. the System Loader, that is to be used with the output object file. Versions 1 and 2 are supported. Thus, the operand must be $1 or $2. The VER instruction should come *before* any other linking instructions except ASMs, which are not dependent on the version of the Loader in use. ProDOS 16 version 1.2 uses version 2 of the loader format, and this is the default version used by the GS Linker if VER is not specified. You will have to decide what OMF version to use. If you specify OMF version 1, your file can be loaded by any version of ProDOS 16. If you specify OMF version 2, ProDOS 16 vers. 1.2 or later will be required.

KND address
KND $80
KND $8000

This specifies the value that you want put in the KIND location of the OMF header. You would use 1 byte for VER $1 and 2 bytes for VER $2. This must come *after* the VER so that the format is known to the Linker. If in doubt, ignore this command and accept the default.

ALI address
ALI $10000

This specifies the ALIGN field in the file header. It defaults to 0. Use only $10000 to align to a bank boundary or $100 to align to a page boundary. In most cases, you should leave this at the default 0.

**NOTE:** Our tests indicate that the bank align does not work on OMF version 1.

DS address
DS $2000

This tells the Linker to reserve this number of bytes to be zeroed by the loader at the END of the program. This number is put in the RESSPC field of the header. Using this instead of reserving space with a DS in the source file will result in smaller object files.

## QUICK LINK (LINKER.GS ONLY)

In LINKER.GS, but not in the other Linkers, the command LINK =, or simply LINK without a file name, from the Command Box will assemble the file given by the default file name, i.e. the name appearing when you give the Load or Save commands from the Main Menu, then will link the resulting file. This assumes that a LNK file was produced by the DSK and REL opcodes. The linked file will then be saved using the name of the LNK file with the last two characters cut off.

It is suggested that you add a .L suffix to the DSK filename, i.e. MYFILE.L, so the saved file will be MYFILE. For example, a source file with the lines:

```
REL
DSK          MYFILE.L
```

could be saved to disk, and then assembled and linked with the Editor LINK command, and the final object file will be saved on the disk under the name MYFILE.

NOTE: This command does not require a linker command file, and that it uses the defaults in the Link which produces an S16 filetype in object module format version 2, and of "kind" $1000, i.e. code segment that cannot be loaded to special memory. This syntax should not be used if you wish load preferences different than the defaults, or if you require the advanced features of a command file. The command LINK1 can be used to produce a file in object module format 1 of "kind" 0 and type S16.

If there is a source file in memory when this command is issued, you will be asked if it is OK to save the source file to disk using the current name. If you agree, that file will automatically be saved under the default file name *Caution: this can be dangerous.* If the workspace is empty when the LINK command is used, then the file given by the default file name will be loaded, assembled, linked and the object file saved.

## MULTIPLE LNK INPUT FILES

If you have multiple LNK files being linked to create the final object file, you should use a command file. However, Linker.GS does provide a short-cut way of linking up to 10 LNK files without requiring a command file. If the LNK file produced by the assembly has a name ending in ".x" where x is a digit from 0 to 9, i.e. MYFILE.3, then the Linker will not immediately link that file into the output file. Instead, it will look for the file with the same name but ending in .0, i.e. MYFILE.0, and will link that file with subsequently numbered LNK object files, i.e. MYFILE.1, MYFILE.2, MYFILE.3, etc., until it finds no more.

Up to 10 files can be linked without using a linker command file with this method. Note that only the default name source file is re-assembled. The source files for the other LNK files in the sequence are not assembled; only the LNK files are used as is, in the final object file. If a file is missing from the sequence, i.e. MYFILE.2 not present on the disk, the linking is terminated at that point, although no error is generated. Thus some care must be taken to make sure that all the needed files are on the disk.

## USING THE GS LINKER

Because the GS Linker is used most often for linking a single input file, and creating a ProDOS 16 application of some sort, this example will demonstrate a simple ProDOS 16 program that waits for a keypress, and then returns to whatever program launched it.

Notice that the characteristic ENT and EXT, etc. items do not show up, since the primary goal is to just produce a single OMF file that can be loaded and run by a ProDOS 16 program launcher.

```
 1 ********************************
 2 *    SIMPLE P16 SYSTEM FILE    *
 3 *      MERLIN 16 ASSEMBLER     *
 4 ********************************
 5
 6           MX    %00          ; FULL 16 BIT MODE
 7           REL                ; RELOCATABLE OUTPUT
 8           DSK   P16.SYSTEM.L
 9
10 PRODOS    EQU   $E100A8      ; PRODOS 16 ENTRY POINT
11 KYBD      EQU   $00C000
12 STROBE    EQU   $00C010
13 SCREEN    EQU   $000400      ; LINE 1 ON SCREEN
14
15 ENTRY     PHK                ; GET PROGRAM BANK
16           PLB                ; SET DATA BANK
17
18 PRINT     LDX   #$00         ; INIT X-REG
19 LOOP      LDA   MSSG,X       ; GET CHAR TO PRINT
20           BEQ   GETKEY       ; END OF MSSG.
21           STAL  SCREEN,X     ; "PRINT" IT
22           INX                ; NEXT TWO CHARS
23           INX                ; X = X + 2
24           BNE   LOOP         ; WRAP-AROUND PROTECT
25
26 GETKEY    LDAL  KYBD         ; CHECK KEYBOARD
27           AND   #$00FF       ; CLEAR HI BYTE
28           CMP   #$0080       ; KEYPRESS?
29           BCC   GETKEY       ; NOPE
30           STAL  STROBE       ; CLEAR KEYPRESS
31
32 QUIT      JSL   PRODOS       ; DO QUIT CALL
33           DA    $29          ; QUIT CODE
34           ADRL  PARMBL       ; ADDRESS OF PARM TABLE
35           BCS   ERROR        ; NEVER TAKEN
36           BRK   $00          ; SHOULD NEVER GET HERE...
37
38 PARMBL    ADRL  $0000        ; PTR TO PATHNAME
39 FLAG      DA    $00          ; ABSOLUTE QUIT
40
41 ERROR     BRK   $00          ; WE'LL NEVER GET HERE?
42
43 MSSG      ASC   "PLEASE PRESS A KEY -> " ; EVEN NUMBER OF CHARACTERS
```

```
44          DA    $0000       ; TWO ZEROS
45
46          LST   OFF
```

First, enter and save this source file. Do not worry about assembling it yet. Once it is saved, use Open-Apple-O to open the Command Box and type NEW to erase the source file.

To link the file, just press Open-Apple-6. The source will automatically be loaded, assembled, the intermediate REL file written to disk, and the final output file, P16.SYSTEM, written to disk.

To test your program, just type -P16.SYSTEM as a disk command from the Main Menu. When the program quits, it will return to whatever program launcher started Merlin.16. You can alternatively quit Merlin to either the Apple DeskTop, Finder, or the Program Launcher, and select the file P16.SYSTEM. When you press a key, control should return to the program selector. Note that either approach *requires* that you have first started up using a ProDOS 16 disk, so that the ProDOS 16 operating system is available.

That's all there is to writing and assembling an Apple IIgs ProDOS 16 program using Merlin 16. For more complex files that require several input REL files, a command file can be used with the GS Linker. Also, try loading the P16.SYSTEM file, and type Open-Apple-6 with the source file in memory. The GS Linker will automatically save the source file before starting the link process. This is so that you can make changes to a program, and then automatically do the whole save-assemble-link process with a single keystroke.

**NOTE:** If you have the Roger Wagner Publishing IIgs program switcher called SoftSwitch, you may also want to put Merlin 16 in one Workspace and the Apple DeskTop program selector in another. With this combination, it is possible to assemble and link a program, save Merlin 16 intact in a Workspace, and switch instantly to the DeskTop to launch your ProDOS 16 program. When the program quits and returns to the DeskTop, you can switch back to Merlin 16 with the proper ProDOS prefixes set, ready to load the source file for further changes. With SoftSwitch and Merlin 16, it is possible to assemble, link and test a ProDOS 16 program, and to be back in Merlin 16 making changes in less than 60 seconds for the total cycle time!

## LARGE FILE GS LINKER (LINKER.XL)

LINKER.XL is a version of LINKER.GS which makes two passes and writes the output file to disk to produce a multi-segment file. Use of the SAV command will cause the Linker to process the code to that point as one segment of a load file. This can produce much larger object files than the other Linker. However it is also much slower, so the other version is probably what you'll want to use most often. LINKER.XL does not have the command box LINK = capability of Linker.GS discussed earlier, so a command file is always required to use it. The maximum number of segments in the output file is 25, each with up to about 32K of code, excluding the relocation dictionary.

For LINKER.XL, the filename in the *first* SAV command is taken as the output file name. The rest, including the first, are placed in the *segment name* field of their respective segment header. If the name is more than the allowed 10 character space in the header, it is truncated to 10 characters.

LINKER.GS and LINKER.XL do not support the EXT or ENT linker commands that print the resolved address of labels, since these are generally meaningless on the Apple IIgs. The linker EXT and ENT commands should not be confused with the assembler EXT and ENT commands which are, of course, supported in all versions.

# TECHNICAL INFORMATION

The source is placed at START OF SOURCE when loaded, regardless of its original address.

The important pointers are:

| | | | |
|---|---|---|---|
| START OF SOURCE | in | $A,$B | (set to $901 unless changed) |
| HIMEM | in | $C,$D | (defaults to $9853 in DOS 3.3, defaults to $AA00 in ProDOS) |
| END OF SOURCE | in | $E,$F | |

Note that HIMEM does not change unless a USER routine or utility program changes locations $73, $74. Such a change will be copied automatically into locations $C, $D.

## GENERAL INFORMATION (DOS 3.3 ONLY)

When you exit to BASIC or to the Monitor, these pointers are saved on the RAM card at $E00A-$E00F. They are restored upon re-entry to Merlin 8/16.

Entry into Merlin 8/16 replaces the current I/O hooks with the standard ones and reconnects DOS. This is the same as typing PR#0 and IN#0 from the keyboard. Entry to the Editor disconnects DOS, so that you can use labels such as INIT without disastrous consequences. Re-entry to the Main Menu disconnects any I/O hooks that you may have established via the editor's PR# command, and reconnects DOS. Exit from assembly due to completion of assembly or Control-C also disconnects I/O hooks.

Re-entry after exit to BASIC is made by the ASSEM command. Simply use ASSEM wherever a DOS command is valid, for example, at the BASIC prompt. A BRUN MERLIN or a disk boot will also provide a warm re-entry and will not reload Merlin 8 if it is already there. A reload may be forced by typing BRUN BOOT ASM which would then be a cold entry, erasing any file in memory.

The DOS 3.3 version does not perform the same volume checking as the ProDOS version. However, it is possible to simulate this with the following code:

```
      LST
XXX   KBD "INSERT MYFILE DISK AND TYPE 0 <RETURN>"
      PAUSE
```

The assembler will stop at KBD on the first pass and assign a 0 value to XXX where XXX is any dummy label you desire. PAUSE will force a pause on the second pass and LST makes sure you will see the KBD line. On the second pass, assembly resumes when you press any key. It is not necessary to type 0 and press Return.

## GENERAL INFORMATION (ProDOS AND DOS 3.3)

In Merlin 8, if during assembly the object code exceeds usable RAM then the code will not be written to memory, but assembly will appear to proceed as normal and its output sent to the screen or printer. The only clue that this has happened, if not intentional, is that the Save Object Code command at Main Menu is disabled in this event. There is ordinarily a 16K space for object code, which can be changed with the OBJ opcode. In Merlin 16, an object code overflow generates an error message.

## SYMBOL TABLE

The symbol table is printed after assembly unless LST OFF has been used. It is displayed first sorted alphabetically and then sorted numerically. The symbol table can be canceled at any time by pressing Control-C. Stopping it in this manner will have no ill effect on the object code which was generated. The symbol table is flagged as follows:

MD = Macro Definition
M = Label defined within a Macro
V = Variable (symbols starting with "]")
? = A symbol that was defined but never referenced
X = External symbol
E = Entry symbol

Local labels are not shown in the symbol table listing. In Merlin 16, this can be enabled by changing the PARMS file.

When in EDIT mode, Merlin 8/16 takes total control of input and output. The effect of typing a control character will be as described in this manual and *not* as described in the manual for your 80 column card. For example, Control-L will not blank the screen, but is the case toggle. Control-A, which acts as a case toggle on many 80 column cards, will not do this in the Editor and simply produces a Control-A in the line being edited.

## ULTRATERM INFORMATION

The Ultraterm is an 80 column display card manufactured by Videx. If you do not have this card, skip to the next page. When in the Editor, the Ultraterm mode can be altered by the ESCAPE sequence given in the Ultraterm manual. Thus, the following commands give the indicated effects:

ESC 0 ..... 40 x 24 same effect as VID $10 or 16
ESC 1 ..... 80 x 24 standard character set
ESC 2 ..... 96 x 24
ESC 3 ..... 160 x 24
ESC 4 ..... 80 x 24 high quality character set
ESC 5 ..... 80 x 32

| | |
|---|---|
| ESC 6 | ..... 80 x 48 |
| ESC 7 | ..... 132 x 24 |
| ESC 8 | ..... 128 x 32 |

Exit to the Main Menu will return to the default state as set up in the HELLO program for DOS 3.3 or the PARMS file for ProDOS. The same is true of a VID 3 command.

Except for the normal 24 x 80 format, support for the Ultraterm depends on the card being in slot 3.

There may be problems if you try to send things to the printer while in some of the Ultraterm modes. It is recommended that you switch to 40 columns before doing this. Using a PRTR1"<Control-I>80N" command sometimes overcomes the problem.

## MEMORY ALLOCATION WITH MERLIN 8/16

The memory areas $300-$3EF in main memory and $800-$FFF in auxiliary memory are available for user supplied USER and USR routines. The page 3 area in main memory is intended for I/O interface routines. One cannot send a character to COUT, for example, from auxiliary memory. Merlin does not use these areas. Zero page locations $90-$9F are not used by Merlin and are reserved for USER routines (note that the XREF program uses these locations). Zero page locations $60-$6F are reserved for user supplied routines and may be used as you wish. No other zero page locations are available.

## CONFIGURING MERLIN 8 (ProDOS)

Configuration data is kept in a file called PARMS which is loaded when the assembler is run. To change the data just change the \+Italic\source\-Italic\ file PARMS.S and reassemble it.

To load the file, set the prefix to /MERLIN 8 and type L to load a source file. Then type SOURCE/PARMS at the prompt. When you are done making changes, reassemble the file. Use S to SAVE the source code as /MERLIN/SOURCE/PARMS. Remember that Merlin 8 adds the .S suffix automatically. Then save the object code as /MERLIN/PARMS by using the O command.

## CONFIGURING MERLIN 8 (DOS 3.3)

The data statements in the Applesoft boot program HELLO contain the configuation information. To change the data just LOAD HELLO, change the data in the DATA statements and SAVE HELLO.

## DATA DESCRIPTION FOR MERLIN 8 CONFIGURATIONS

| DATA # | DEFAULT | PURPOSE |
|---|---|---|
| 1 | 60 | Number of lines per page (for PRTR). |
| 2 | 0 | Lines to skip at page perforation (0 sends a form feed character. |
| 3 | 80 | Number of characters per line (for PRTR). |

| DATA # | DEFAULT | PURPOSE |
|---|---|---|
| 4 | $80 | Must be $80 if printer does its own CR at end of line, otherwise should be 0. |
| 5 | $83 | 80 column flag. Should be $80+3 if 80 column card is in slot 3 (or Apple 80 col card) is to be selected upon boot. Otherwise 0. MUST BE $83 WITH ProDOS. |
| 6,7 | $901 | Source file start address, must not be less than $901. |
| 8,9 | $AA00 | SHOULD NOT BE CHANGED. |
| 10,11 | $901 | End of source pointer. Must equal the Source file start address. |
| 12 | $DE "^" | The editor's wild card character. |
| 13 | 4 | Number of fields per line in symbol table printout. |
| 14 | $AF "/" | Character searched for by "UPDATE SOURCE" entry to assembler. If this is 0 the question will be bypassed. |
| 15,16,17 | 14,20,31 | The default tabs for editor and assembler, note that these values are relative to the left side of screen. |
| 18 | 8 | Number of object bytes/line after the first line. |
| 19 | 5 | Error/bell flag and Ultraterm start parameters. To disable the bell, set this value to 197. The high bit, if on, will force the assembler to pause forever for a keypress at an error; if off, a sound continues for 20 seconds and then assembly continues. The V bit, if set disables some bells. The low nibble determines the default mode of the Ultraterm if you are using that. The value 5 or $85 gives the 32X80 mode. |
| 20 | $40 | Cursor flag. Gives regular cursor if this is $40 and block cursor if 0. The Apple 80-col card must have the block cursor and this flag will be overridden if you are using that card. |
| 21 | 0 | LSTDO default: 0,1=LSTDO ON, >1=LSTDO OFF. Bit 0, if clear, causes shift to 40 columns when a PRTR command is issued. |
| 22 | 72 | Column at which the cycle count will be printed when using the CYC opcode. |
| 23 | $EC | Cursor type for Ultraterm. Must be changed if the Ultraterm mode is changed (see byte 19). |
| 24-44 | "$F1 to $F7" | File type names for the user defined file types $F1 through $F7. These names will be shown in the directory when cataloged by Merlin. ProDOS ONLY. |

## 64K MERLIN AND MERLIN 8/16 SOURCE FILES

Source files from the original 64k version of Merlin for the Apple II+ can be loaded directly into DOS 3.3 Merlin 8. To use 64k Merlin source files with ProDOS Merlin 8/16 you must use the CONVERT utility supplied with the ProDOS User's Disk. Some changes may be required to the source due to some of the missing pseudo opcodes in Merlin 8/16. If your program uses HIMEM: or SYM, they should be deleted. If your program uses the ERR opcode to check whether SYM or HIMEM: have been set, they should be deleted. If your program uses Sweet 16 then the enabling opcode SW will have to be inserted. Also, any OBJ opcodes will have to be removed since the meaning of this opcode has been changed.

## MERLIN 8/16 ProDOS NOTES

The ProDOS version uses TXT files exclusively for source files. This includes files intended for the PUT or USE opcodes, and all such files must have the .S suffix in the file name, which is automatically appended by Merlin 8/16 for all loads and saves. It is suggested that you keep files intended for PUT or USE in a subdirectory. For example, you could save a file named MYPUT under the pathname LIB/MYPUT. It would then be called in an assembly program by: PUT LIB/MYPUT, or PUT /PREFIX/LIB/MYPUT if the PUT file is in the volume called PREFIX.

If you save a file under a directory name that does not exist, a subdirectory will be created under that name. Suppose you want to save your current source SRC in the volume MYVOL and in the subdirectory SUB which does not exist in the MYVOL directory. Then type /MYVOL/SUB/SRC when the pathname is requested, or just SUB/SRC if /MYVOL/ is the prefix, and the subdirectory SUB will be automatically created and the file SRC placed in it.

It is wise to use a full pathname in operands of the SAV, USES and PUT opcodes, since otherwise the current prefix will be attached to the name and that may not be the prefix you want.

Slot and drive parameters are \+Italic\not\-Italic\ acceptable by any commands or opcodes. You \+Italic\must\-Italic\ use pathnames.

Since the ProDOS version of Merlin 8/16 runs under its own interpreter rather than the BASIC interpreter, there is no warm re-entry as with the DOS 3.3 version.

There is no equivalent of the BASIC CAT or CATALOG commands as Disk Commands. The interpreter automatically selects the catalog format for the C command according to whether you are in 40 or 80 column mode on Merlin 8.

The ProDOS volume /RAM/ is disconnected by Merlin 8/16 since it uses all of auxiliary memory.

If Merlin 8/16 cannot find a disk volume required for linking or assembly, it will ask for the correct volume to be inserted. This request can be aborted by pressing Control-C. This only applies to volumes, and not files. Thus, if you want a PUT opcode to prompt you to switch disks, you must use the full pathname with the PUT opcode. Note that this feature will not work with the Linker when using one disk drive.

If the present prefix does not correspond to any volume online, Merlin 8/16 will give a VOLUME NOT FOUND error.

## TRANSFERRING SOURCE FILES FROM DOS 3.3 TO ProDOS MERLIN 8/16

There are two methods of transferring files from the DOS 3.3 versions of Merlin to the ProDOS versions. Since the ProDOS version uses text files only, you could load files into the DOS 3.3 version and write them as text files and then transfer them with Apple's CONVERT program. Unfortunately, CONVERT is not a literal transfer, as it will clear the high bits in the file. The ProDOS version of Merlin 8/16 will set the high bits again, but the tabbing in the editor will be fouled up by this procedure. However, you merely have to type FIX in the editor and resave the source to remedy this problem. Files intended for PUT or USE should be resaved because, otherwise, assembly will be slowed.

Another method is to transfer the files as binary files from DOS 3.3 and use the fact that the ProDOS version of Merlin has the ability to load any type of file including binary files. This does not apply to saving. After loading a binary source file, it should be deleted and saved back as a TXT file. The Load command automatically permits loading of TXT or BIN files. Other types of files can be loaded by changing the byte used to designate source file type which is kept in location $BE5D which ordinarily holds a 4.

Since the ProDOS version of the assembler does not use the T. prefix of the DOS 3.3 version for PUT files, there will be some renaming of such files that will be necessary.

## MERLIN 8 AND SPEED UP CARDS

Merlin 8 will work either in main or auxiliary memory, aux being the default. If you are using the main memory version, you will get about a 1.6 times speed improvement with the SpeeDemon card, and about a 2x speed improvement with the Titan Accelerator the Applied Engineering Transwarp card. The difference is due to the speed up of auxiliary memory during assembly.

To select the main memory version of Merlin 8 with DOS 3.3, change the HELLO program to BLOAD MERLIN.X instead of MERLIN.

To select the main memory version of Merlin 8 with ProDOS, use a $C3 as the fifth byte in the PARMS file. The V-bit of that location is used as a flag to instruct the interpreter to make the main memory modifications.

A plus sign (+) after the Merlin 8 Version number on the Main Menu screen indicates the main memory version is active.

Some utilities do not work with the ProDOS main memory version. This is because ProDOS is moved to auxiliary memory. Programs that do not switch zero pages will work correctly. Programs designed to be run in 64K will most likely run properly. The Filer and Convert programs will run as long as the "-" command is used to run them, and all Merlin 8 utilities will function correctly. The QUIT command moves ProDOS back to main memory.

## MERLIN 8 and DOS 3.3 HARD DISKS or RAM disks

On the DOS 3.3 version of Merlin 8 are files called MERLIN.CORVUS and MERLIN.CORVUS.X. Normally, the DOS 3.3 version of Merlin 8 makes certain patches to DOS 3.3 to allow faster file loading, and the optional cancelling of a Catalog at the screen pause. With hard disks, RAM disks, DOS 3.3 for 3.5" disks, or any other custom version of DOS 3.3, the standard version of Merlin 8 may not work properly. The files MERLIN.CORVUS and MERLIN.CORVUS.X are versions of Merlin 8 which do not modify DOS 3.3 in any way, and are thus compatible with the Corvus and other custom DOS 3.3 software. To use the file called MERLIN.CORVUS, rename the file called MERLIN to TEMP.MERLIN and then rename MERLIN.CORVUS to MERLIN. MERLIN.CORVUS.X is a main memory version of MERLIN.CORVUS, which you may want to use if you are using the SpeeDemon accelerator card instead of MERLIN.X.

## CONFIGURING MERLIN 16

Merlin 16 can be customized by re-assembling the file PARMS.S in the SOURCE directory of the Merlin 16 diskette. The object file, PARMS, must be saved to the same directory that Merlin.System is located in.

## THE MERLIN 16 PARMS FILE

Here is a listing of the PARMS.S file for Merlin 16. By examining the comments, you can see which attributes of the assembler can be changed.

```
 1 *=======================
 2 *  PARMS for Merlin.16
 3 *-----------------------
 4
 5          TR    ADR
 6          TR
 7
 8 Y        EQU   1
 9 y        EQU   1
10 N        EQU   0
11 n        EQU   0
12
13 SAVOBJ   KBD   "Save object code? (Y,N)"
14
15          ORG   $8000
16
17 DATA     ORG   $E4F3
18
19          DFB   60          ;# lines/page for PRTR
20          DFB   0           ;Page skip (formfeed if 0)
21          DFB   80          ;# printer columns
22          DFB   $80         ;- if printer does CR at
```

```
23                              ; end of # columns
24          ORG
25          ORG   $E009
26
27          DFB   $83           ;80 col flag (DO NOT CHANGE
28                              ; except V-bit which will
29                              ; cause ProDOS to be moved
30                              ; to aux memory and Merlin
31                              ; to load into main memory.
32                              ; I.e., use $C3 for this.)
33
34 *=======================================
35 * Source address must be $901 or above.
36 * It can be set higher to protect an area
37 * of RAM above $900 for any purpose:
38 *---------------------------------------
39
40 SOURCE    =     $901
41
42          DA    SOURCE      ;Start of source
43          DA    $9E00       ;Reserved
44          DA    SOURCE      ;End of source
45
46          DFB   0           ;main menu accepts RTN as "Y"
47                            ; if this is set to $FF
48          DFB   4           ;# of symbol columns
49
50 *================================================
51 * Following flags byte has all bits significant:
52 *
53 * Bit 7 = print date on page header of assembly.
54 * Bit 6 = return to main menu after a key press
55 *   at end of an assembly, or to full screen editor
56 *   at point of error if an assembly abort occurs
57 *   (or also if REL is active).  If this bit is 0,
58 *   and bit 0 is 1, then it goes to the command line
59 *   editor.
60 * Bit 5 = linkers should check auxtype bit 0 of
61 *   source files to decide whether to do assemblies.
62 * Bits 4,3 = default XC mode:
63 *    00 = defaults to 6502
64 *    10 = 65C02 mode
65 *    11 = 65816 mode
66 *    (Do not use other values.)
67 * Bits 2,1 = default MX mode on entry:
68 *    00 = full 16-bit mode on entry
69 *    10 = short M, long X
70 *    01 = long M, short X
71 *    11 = 8 bit mode
72 * Bit 0, if set, enables command line editor access.
73 *------------------------------------------------
74
75          DFB   %11111110   ;Misc flags, see above
```

```
 76
 77          DFB   9,15,26     ;Default tabs
 78
 79          DFB   4           ;# obj bytes/line after 1st
 80
 81 *================================================
 82 * Following flags byte has 3 high bits significant:
 83 *
 84 * Bit 7 = wait forever for key upon assembly error.
 85 * Bit 6 = defeat most bells if set.
 86 * Bit 5 = do not pause or sound alarm on an assembly
 87 *       error (must also have bit 7 clear for this).
 88 *
 89 * Low nibble is Ultraterm entry mode:
 90 *
 91 * Eg., $05, $45, $85, etc give 32x80 interlace mode.
 92 *------------------------------------------------
 93
 94          DFB   $05         ;Bell flags & UT mode.
 95
 96          DFB   %01000000   ;Upper case convert mode
 97                            ; when entering full screen
 98                            ; editor if negative.
 99                            ; (Conversion is done for
100                            ; label, opcode and operand
101                            ; fields only and only when
102                            ; tabs are not zeroed.)
103                            ;V-bit = default cursor
104                            ; mode, 1=insert cursor
105
106 *================================================
107 * Following flags byte has 5 significant bits:
108 *
109 * Bit 7 = assembler is label case insensitive if on.
110 * Bit 6 = defeat screen ed screen blank, if on.
111 * Bit 5 = enable list of local labels, if on.
112 * Bit 1 = LSTDO default, DO off areas not listed if
113 *       this bit is on (and default not overridden).
114 * Bit 0 = defeat shift to 40 columns on PRTR1, if
115 *       bit is on (shift only occurs when Ultraterm
116 *       is active with more than 24 rows, so it is
117 *       best to leave this bit as is (off)).
118 *------------------------------------------------
119
120          DFB   %01000000   ; case senst., no blanking
121
122          DFB   80-8        ;Column for cycle count
123
124          ORG
125          ORG   $B23E
126
127          DFB   $9F&"["     ;Catalog abort key, now ESC
128
```

```
129           ORG
130
131           ERR    *-DATA-23  ;23 data bytes to here.
132
133 * User file type names:
134
135           ORG    $B6B6      ;Adrs subject to change
136
137           ASC    "$F1"
138           ASC    "$F2"
139           ASC    "$F3"
140           ASC    "$F4"
141           ASC    "$F5"
142           ASC    "$F6"
143           ASC    "$F7"
144
145           ORG
146
147           ERR    *-DATA-44
148
149 *=======================================
150 * Screen editor variable parameters:
151 * Cursors are: insert, find in insert mode,
152 *  overstrike, find in overstrike mode.
153 *---------------------------------------
154
155           ORG    $DFBC
156
157 CURSORS   INV    'IF F'
158
159           DFB    $A0        ;Cursor blink rate
160
161 *=======================================
162 * Screen editor cmd chars:
163 * The cursor keys (although here) must NOT
164 * be changed or the editor will not work
165 * correctly.
166 *---------------------------------------
167
168           DFB    $9F&"U"    ;Don't change
169           DFB    $88        ; "
170           DFB    $9F&"I"    ;Tab key insert toggle
171           DFB    $9F&"T"    ;^T  (remember this place)
172           DFB    $9F&"B"    ;Go to line beginning
173           DFB    $9F&"N"    ;Go to line end
174           DFB    $9F&"R"    ;Cancel changes
175           DFB    $9F&"S"    ;Status box
176           DFB    $9F&"F"    ;Find char
177           DFB    $9F&"W"    ;Next word
178           DFB    $9F&"L"    ;Toggle uc/lc auto shift
179           DFB    $9F&"D"    ;Delete char under cursor
180           DFB    $FF        ;Delete previous char
181           DFB    $9F&"Y"    ;Clear to end-of-line
```

```
182           DFB   $9F&"O"     ;Accept next key literal
183
184 * Open-apple key cmds, must be upper case:
185
186           DFB   #"X"        ;Cut
187           DFB   #"C"        ;Copy
188           DFB   #"V"        ;Paste
189           DFB   #"F"        ;Find
190           DFB   #"W"        ;Find word
191           DFB   #"E"        ;Exchange text
192           DFB   #"T"        ;Go to ^T selected point
193           DFB   $9F&"I"     ;Insert line at cursor
194           DFB   #"I"        ; "
195           DFB   $9F&"["     ;Go to cmd line ed if enabled
196           DFB   #"B"        ;Go to beginning
197           DFB   #"N"        ;Go to end
198           DFB   $DF         ;Delete preceding line (DEL)
199           DFB   #"L"        ;Locate text
200           DFB   #"Z"        ;Center line with cursor
201           DFB   $9F&"J"     ;Don't change
202           DFB   $9F&"K"     ; "
203           DFB   #"Y"        ;Select from here on
204           DFB   $9F&"U"     ;Don't change
205           DFB   $9F&"H"     ; "
206           DFB   #"D"        ;Delete this line
207           DFB   #"8"        ;Line of asterisks
208           DFB   #"9"        ;Line bordered by asterisks
209           DFB   #"-"        ;Line of dashes
210           DFB   #"="        ;Line of equal signs
211           DFB   #"1"        ;PRTR1 + assemble
212           DFB   #"2"        ;PRTR1 + USER + assemble
213           DFB   #"3"        ;PRTR3 + assemble
214           DFB   #"4"        ;PRTR3 + USER + assemble
215           DFB   #"A"        ;ASM command
216           DFB   #"Q"        ;Quit to main menu
217           DFB   #"H"        ;Toggle split screen
218           DFB   #"O"        ;Open command box
219           DFB   #"6"        ;LINK (for LINKER.GS only)
220           DFB   #"R"        ;Replace line
221
222           ORG
223
224           ERR   *-DATA-99
225
226 *===================================
227 * Printer init string, used when PRTR
228 * issued with empty first string.
229 * A CR is always issued after the
230 * string, so none need be here.
231 *-----------------------------------
232
233           ORG   $DBF0
234
```

```
235           DS    15          ;PRTR init default string
236
237           ORG
238
239           ERR   *-DATA-114
240
241  *========================
242  *  Default STARTUP file:
243  *------------------------
244
245           ORG   $2006
246
247  STUP     STR   "LINKER.GS" ;You can change this string
248
249           DS    $40-*+STUP ; but not this line
250
251           ORG
252
253           ERR   *-DATA-178
254
255  *================================
256  * The PARMS file must be in the
257  * SAME DIRECTORY as MERLIN.SYSTEM
258  *--------------------------------
259
260           DO    SAVOBJ
261           SAV   PARMS
262           FIN
```

### Screen Blanking

The PARMS file for Merlin 16 supports an optional setting for telling Merlin 16 to blank the screen if nothing is typed for an extended period. Pressing any key, such as Return, will restore the normal screen display.

This is intended to protect your monitor screen in situations where the computer is left unattended for long periods of time, and Merlin 16 is the only program generally run on that computer. Any video image will tend to "burn" itself into a monitor screen if left on continuously for several hours a day, every day, for a period of many months. Note this is not a specific problem to Merlin 8/16.

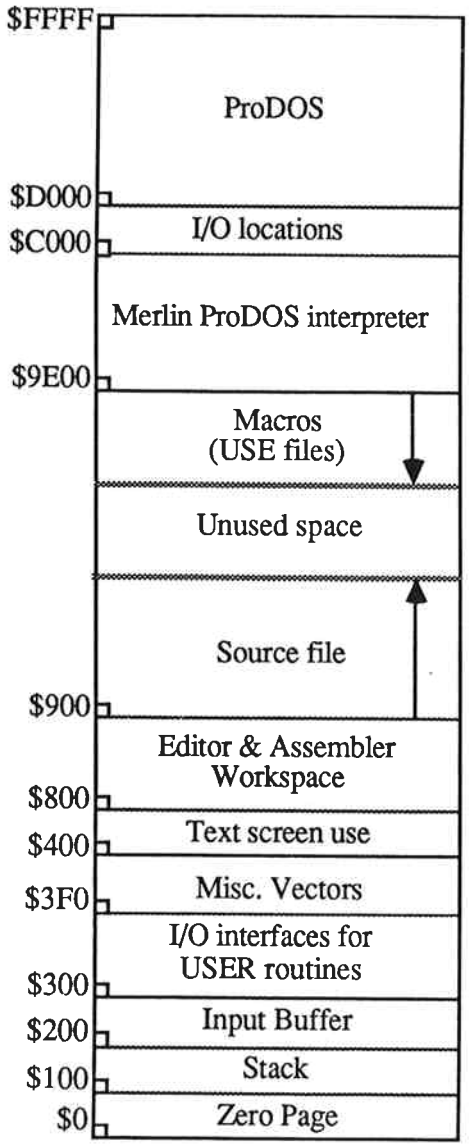
You should not concern yourself with this unless you are in a working environment where the Merlin 16 menu is left on all day when the computer is unattended.

Because it can be disconcerting to have the screen suddenly go blank if you are not aware of this feature, Merlin 16 is shipped with this feature disabled. To enable screen blanking, set the high bit of the flag near line 120 of the Merlin 16 PARMS.S file. See the comments in the source listing for details.

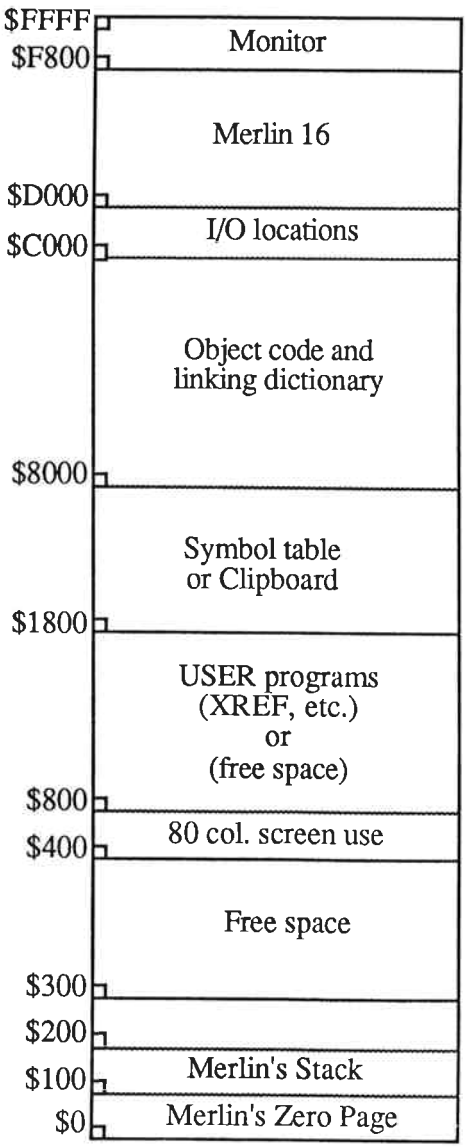
## MERLIN 8/16 MEMORY MAPS

## Merlin 16 - ProDOS

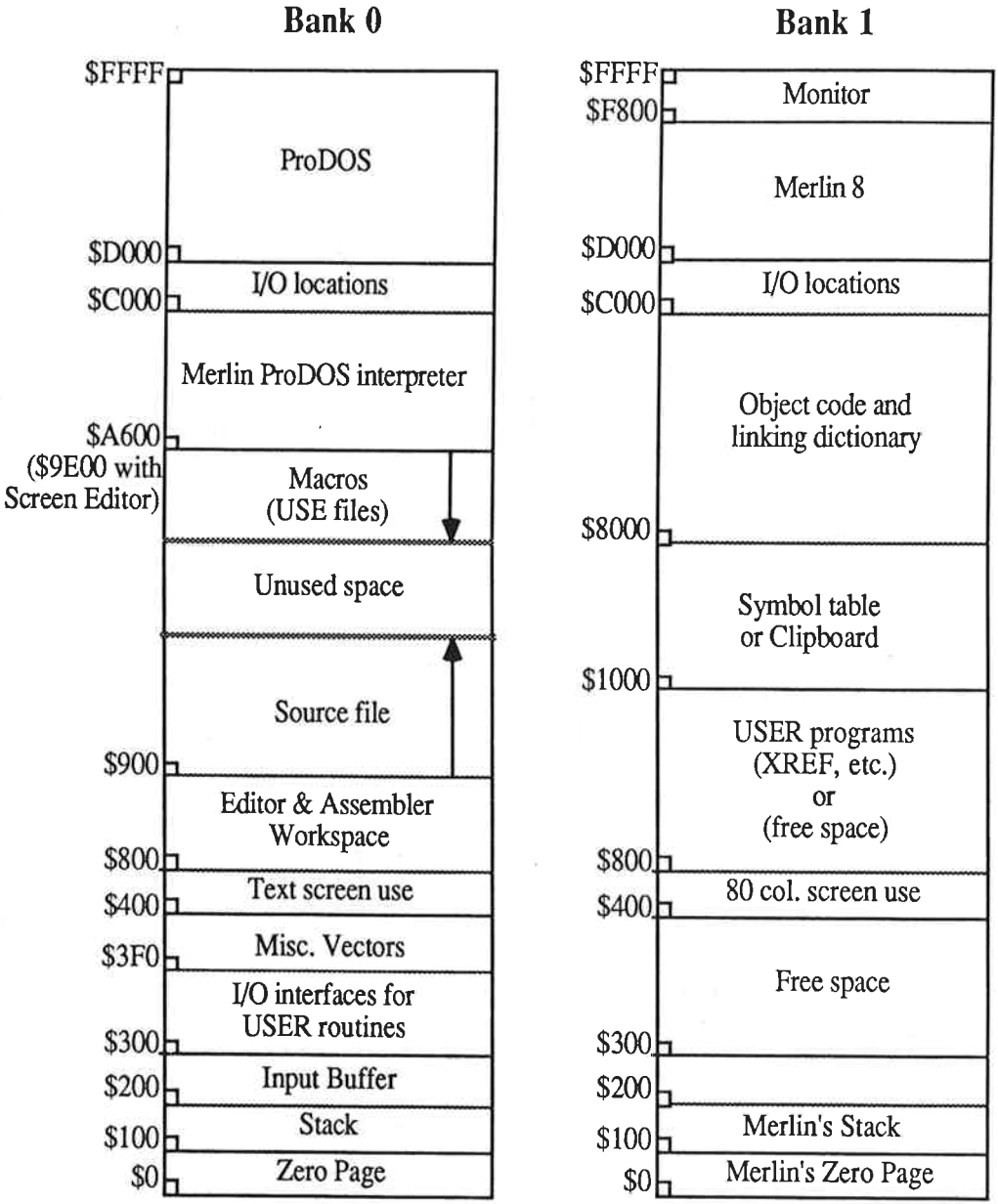
### Bank 0



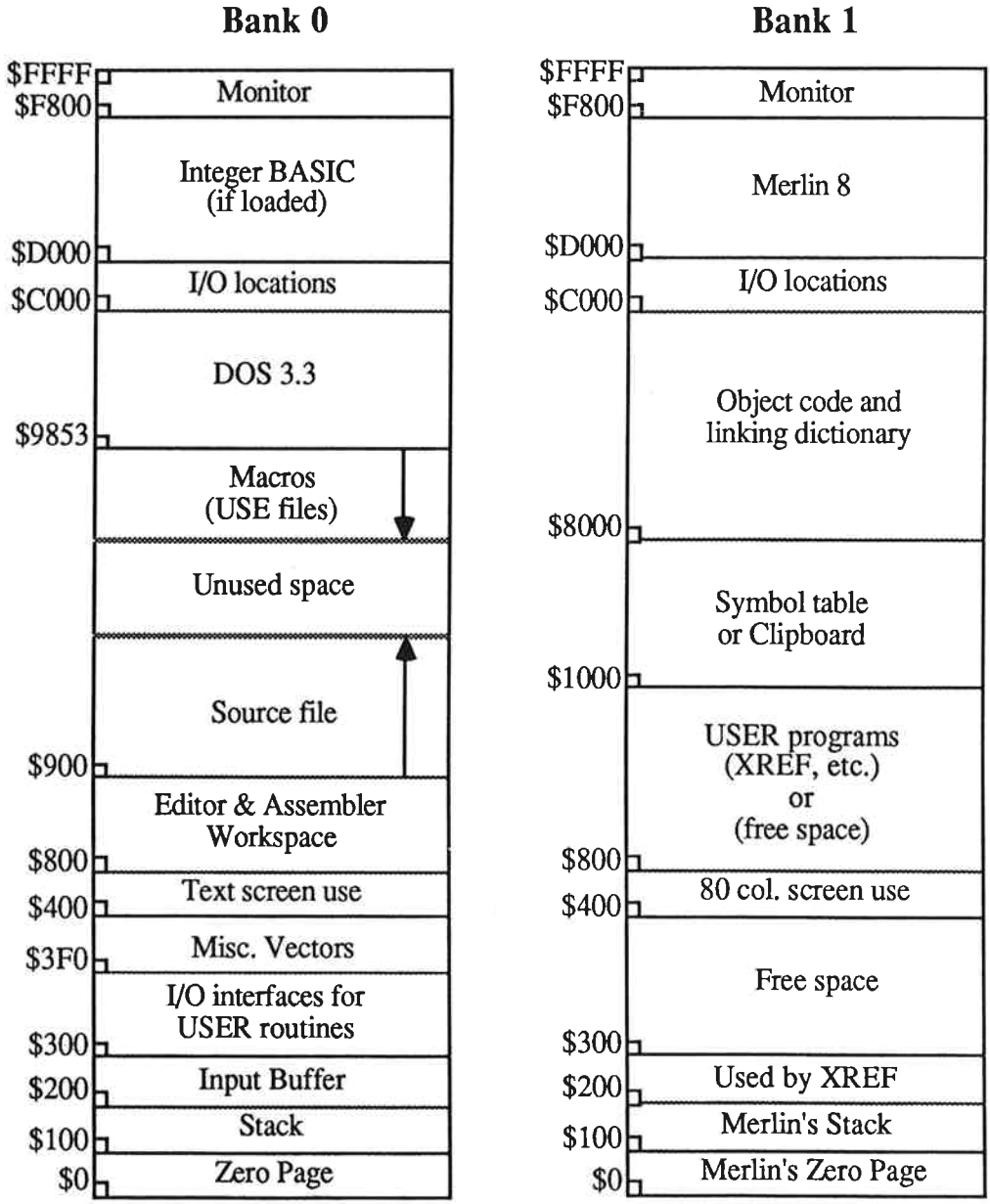

### Bank 1



# Merlin 8 - ProDOS



# Merlin 8 - DOS 3.3



# ERROR MESSAGES

## BAD ADDRESS MODE

The addressing mode is not a valid 6502 instruction; for example, JSR (LABEL) or LDX (LABEL),Y.

## BAD BRANCH

A branch (BEQ, BCC, etc.) to an address that is out of range, i.e. further away than +127 bytes.

**NOTE:** Most errors will throw off the assembler's address calculations. Bad branch errors should be ignored until previous errors have been resolved.

## BAD EXTERNAL

EXT or ENT in a macro or an equate of a label to an expression containing an external, or a branch to an external (use JMP).

## BAD INPUT

This results from either no input, i.e. Return alone or an input exceeding 37 characters in answer to the KBD opcode's request for the value of a label.

## BAD LABEL

This is caused by an unlabeled EQU, MAC, ENT or EXT, or a label that is greater than 13 characters, or one containing illegal characters. A label must begin with a character at least as large in ASCII value as the colon and may not contain any characters less than the digit zero.

## BAD OBJ

An OBJ after code start or OBJ not within $4000 to $BFE0.

## BAD OPCODE

Occurs when the opcode is not valid, or misspelled, or the opcode is in the label column.

## BAD ORG

Results from an ORG at the start of a REL file.

### BAD PUT

This is caused by a PUT inside a macro or by a PUT inside another PUT file.

### BAD REL

A REL opcode occurs after some labels have been defined.

### BAD SAV

This is caused by a SAV inside a macro or a SAV after a multiple OBJ after the last SAV.

### BAD VARIABLE

This occurs when you do not pass the number of variables to a macro that the macro expects. It can also occur for a syntax error in a string passed to a macro variable, such as a literal without the final quote.

### BREAK

This message is caused by the ERR opcode when the expression in the operand is found to be non-zero.

### DICTIONARY FULL

Overflow of the relocation dictionary in a REL file.

### DUPLICATE SYMBOL

On the first pass, the assembler finds two identical labels.

### FILE TYPE MISMATCH

A file specified to be loaded, such as a source file, or REL file during a Link, does not have the expected file type.

### FORMAT ERROR

A command has been incompletely typed, for example, typing PRTR instead of PRTR 1.

### ILLEGAL CHAR IN OPERAND

A non-math character occurs in the operand where the assembler is expecting a math operator. This usually occurs in macro calls with improper syntax resulting from the textual substitution.

### ILLEGAL FILE TYPE (ProDOS version only)

TYP opcode used with an illegal operand.

### ILLEGAL FORWARD REFERENCE

A label equated to a zero page address after it has been used. This also occurs on the first pass when an unknown label is used for some things that must be able to calculate the value on the first pass, e.g. ORG< OBJ DUM. It also occurs if a label is used before it is defined in a DUM section on zero page.

### ILLEGAL RELATIVE ADRS

In REL mode a multiplication, division or logical operation occurs in a relative expression. In Merlin 8, this also occurs for an operand of the type #>expr or a DFB >expr when the expr contains an external and the offset of the value of the expr from that of the external exceeds 7.

### MEMORY FULL Errors

There are three common causes for the MEMORY FULL error message.

**MEMORY FULL IN LINE: xx.** Generated during assembly.

CAUSE #1: Too many symbols in the symbol table, causing it to exceed available space.
REMEDY #1: Make the symbol table larger by setting OBJ to $BFE0 and use DSK to assemble directly to disk.

CAUSE #2: If the combined size of the source file and a PUT file is too large.
REMEDY #2: Split either file into two smaller files.

**ERR:MEMORY FULL.** Generated immediately after you type in one line too many.

CAUSE: The source code is too large and has exceeded available RAM.
REMEDY: Break the source file up into smaller sections and bring them in when necessary by using the PUT pseudo-op.

**ERROR MESSAGE: None**, but no object code will be generated. There is no OBJECT information displayed on the Main Menu.

CAUSE: Object code generated from an assembly would have exceeded the available 16K space.
REMEDY: Set OBJ to an address less than its $8000 default or use the DSK psuedo-op.

### MEMORY IN USE

This error results from trying to load a file into a part of memory already in use. This could be by loading a tool-large PUT file, appending a file without sufficient remaining memory, or loading a USE library. It can also occur from loading a USER function when one is already in memory. In Merlin 8, the error occurs if you try to load a utility like SOURCEROR when the Editor (ED) is already loaded. Use REMOVE.ED before loading another utility.

### MISALIGNMENT - MERLIN 16

This means that the value of a label on the second pass differs from what it was on the first pass and should indicate a forward reference that the assembler could not resolve. The most common cause is a forward reference to a zero page equate or dummy section. Misalignment may also be caused by a previous error. A misalignment error is given only once in order to avoid the error routine on all subsequent labels, all of which are likely to be misaligned.

### NESTING ERROR

Macros nested more than 15 deep or conditionals nested more than 8 deep will generate this error.

### NOT A MACRO

A Macro name has been used that has not been previously defined.

### NOT RESOLVED

This is a Linker error that alerts you to the fact that an EXT label reference in one of the linked files could not be not be found in any of the other linked files.

### OBJECT SPACE OVERFLOW - MERLIN 16 ONLY

If the available object file space has been exceeded, the assembler prints this message. This disables the Object file save command at the Main Menu.

### OUT OF MEMORY

This is a memory error from the Full Screen Editor. It is caused by trying to cut or copy more text to the clipboard, or to paste more text into the document than there is free memory to accomodate.

### RANGE ERROR

When saving a source file larger than 32K, you may get this error, in particular if are using a standard DOS 3.3 and have not booted on the Merlin 8 DOS 3.3 diskette. This is due to a limitation of DOS itself. The problem can be temporarily fixed however by changing byte $A964 in memory from $7F to $FF. (For those experienced with a disk sector editor, you can change Track 1, Sector 8, Byte $64 from $7F to $FF to make this change permanent on whatever DOS 3.3 disk you wish).

### SYNTAX ERROR

The proper syntax has not been used in a command, either because of typing errors, or extra or omitted characters or parameters.

### TWO EXTERNALS

Two or more externals in an operand expression.

### UNKNOWN LABEL

Your program refers to a label that has not been defined. This also occurs if you try to reference a MACRO definition by anything other than PMC or >>>. It can also occur if the referenced label is in an area with conditional assembly OFF. The latter will not happen with a MACRO definition.

### 256 EXTERNALS

The file has more than 255 externals.

**NOTE:** When an error occurs that aborts assembly, the line containing the error is printed to the screen. This may not have the same form as it has in the source, since it shows any textual substitutions that may have occurred because of macro expansion. If it is in a macro call, the line number will be that of the call line and not of the line in the macro which is unknown to the assembler.

# SOURCEROR

Sourceror is a sophisticated and easy to use co-resident disassembler designed to create Merlin 8/16 source files out of binary programs, usually in a matter of minutes. There are two versions of Sourceror in Merlin 8/16. Sourceror on the Merlin 16 disk disassembles 6502, 65C02, 65802 and 65816 object code. Sourceror on the Merlin 8 diskettes disassembles 6502, 65C02, and 65802 object code and is not designed for use on the IIgs.

## USING SOURCEROR ON MERLIN 16

To use Sourceror on the Merlin 16 diskette, follow these steps:

1. From the Main Menu, press D for Disk Command.
2. At the Disk Command prompt type: BRUN SOURCEROR/OBJ
3. Press F to enter the Editor.
4. Press Open-Apple-O to open the Command Box.
5. Type DIS "MYFILE" where MYFILE is the name of the object file to be disassembled.

The DIS command is available in Merlin 16 only and uses the following syntaxes:

| | |
|---|---|
| DIS "MYFILE" | [ disassemble MYFILE at the original running address ] |
| DIS $1000"MYFILE" | [ disassemble MYFILE at the specified address of $1000 ] |

SYS files are always assumed to have a running address of $2000. If a non-zero address is specified in the DIS command, that address is taken as the initial running address. If the address is not specified, the Merlin 16 Sourceror uses the information in the AUXTYPE field for the running address.

## USING SOURCEROR ON MERLIN 8

NOTE: On the ProDOS version of Merlin 8, the Full Screen Editor uses the same part of memory as Sourceror. Therefore you will have to temporarily deactivate the Full Screen Editor. This is not necessary with the DOS 3.3 version of Merlin 8.

To deactivate the Full Screen Editor on Merlin 8 ProDOS version:

1. From the Main Menu, press D for Disk Command.

2. At the Disk Command prompt type: BRUN /MERLIN.8/UTILITIES/REMOVE.ED

To use Sourceror on the Merlin 8 diskette, follow these steps:

1. Merlin 8 DOS 3.3 version
   From the Main Menu, press C to Catalog.
   At the Command prompt, type BRUN SOURCEROR and press Return.

   Merlin 8 ProDOS version
   After deactivating the editor as shown above, you should be at the Disk Command prompt.
   At the Disk Command prompt, type BRUN /MERLIN.8/SOURCEROR/OBJ and press Return.

2. Merlin 8 DOS 3.3 version
   Press E to enter the Editor.

   Merlin 8 ProDOS version
   Press Return to go back to the Main Menu.
   Press E to enter the Editor.

3. Press Escape Control-Q to set the screen to 40 columns.

   The USER command will be ignored and Sourceror will not run if the 80 column screen is in effect. Do not use Escape 4 to change to 40 columns.

4. From the 40 column screen, type USER and press Return.

5. You will be prompted to "Load an object file?"

   If you have loaded the object file prior to using Sourceror, press N.

   If you have not loaded an object file yet, press Y and enter the filename. It will be loaded showing the load address and end of program address. After you have noted these addresses, press any key.

NOTE: If you type Control-S after the filename to be loaded, files using a RAM version of SWEET 16 can be disassembled.

7. If you have loaded a file prior to using Sourceror you will be prompted regarding the location of the object code.

   Press Return if the program to be disassembled is at its original (running) location.

   If not, you must specify in hex the present location of the code to be disassembled. You will also be prompted to give the original location of that program.

8. The Sourceror Main Menu appears.

   Your first command must include a hex address. Thereafter this is optional, as explained shortly. You may now start disassembling or use any of the other commands.

NOTE: When disassembling, you must use the *original* address of the program, not the address where the program currently resides. It will appear that you are disassembling the program at its original location, but Sourceror is actually disassembling the code at its present location and translating the addresses.

## 65C02 OPCODES ON OLDER IIE/IIC COMPUTERS

To disassemble 65C02 codes with the older IIe/IIc ROMs, you must first BRUN the file called MON.65C02 on the Merlin 8 diskette.

If you are using Merlin 8 DOS 3.3, before using Sourceror, you must quit to BASIC from the Main Menu, then type BRUN MON.65C02 and press Return.

For the ProDOS version of Merlin 8, type in D for Disk Command at the Main Menu, and then BRUN /MERLIN/UTIL/MON.65C02 and press Return.

See the information on using the XC opcode for details on assembling 65C02 programs.

This utility is not needed with the enhanced IIe or IIc (Unidisk 3.5 compatible) ROMs.

## DISASSEMBLY COMMANDS

Your first command must include a hex address. Therefore, if you wanted to list the first 20 lines and the starting address was $8000, at the $ prompt you would type 8000L and press Return. All commands accept a 4-digit hex address before the command letter. If this number is omitted after the first command, the disassembly continues from its present address.

If you specify a number greater than the present address, a new ORG will be created.

More commonly, you will specify an address less than the present default value. In this case, the disassembler checks to see if this address equals the address of one of the previous lines. If so, it simply backs up to that point. If not, then it backs up to the next used address and creates a new ORG. Subsequent source lines are erased. It is generally best to avoid new ORGs when possible. If you get a new ORG and don't want it, try backing up a bit more until you no longer get a new ORG upon disassembly.

This backup feature allows you to repeat a disassembly if you have, for example, used a HEX or other command, and then change your mind.

### H (Hex)

This creates the HEX data opcode. It defaults to one byte of data. If you insert a one byte hex number using one or two digits after the H, that number of data bytes will be generated.

### L (List)

This is the main disassembly command. It disassembles 20 lines of code. It may be used in multiples, thus 2000LLL will disassemble 60 lines of code starting at $2000. In the Merlin 8 Sourceror, if a JSR to the SWEET 16 interpreter is found, disassembly is automatically switched to the SWEET 16 mode.

The L command always continues the present mode of disassembly, SWEET 16 or normal.

NOTE: If an illegal opcode is encountered, the bell will sound and opcode will be printed as three question marks in flashing format. This is only to call your attention to the situation. In the source code itself, unrecognized opcodes are converted to HEX data, but are not displayed on the screen.

### N (Normal - Merlin 8 only)

This is the same as L, but forces disassembly to start in normal 6502 mode.

### O (Org - Merlin 16 only)

This command can be used to change the ORG on the fly during disassembly. It is useful for programs that move code to other locations after loading. The syntax for this command is address O newaddress. Do not use any spaces in this command syntax, and note that it uses the letter O and not zero.

ADRSONEWADRS [ backup to ADRS and place and ORG to NEWADRS there ]

Therefore a command of 1045O300 followed by a Return would tell Sourceror to backup to address 1045 and place an ORG $300 there.

### Q (Quit)

This ends disassembly and goes to the final processing which is automatic. If you type an address before the Q, the address pointer is backed to, but does not include, that point before the processing. For example, if at the end of the disassembly, the lines included:

```
2341-  4C  03  E0        JMP  $E003
2344-  A9  BE  94        LDA  $94BE,Y
```

and the last line was garbage, you could type 2344Q and press Return. This would cancel the last line, but retain all the previous lines.

### R (Read)

In Merlin 8, this allows you to look at memory in a format that makes imbedded text stand out. For example, to look at the data from $1000 to $10FF, you would type 1000R and press Return. After that, R and Return will bring up the next page of memory. The numbers you use for this command are totally independent of the disassembly address.

You may also disassemble, then use (address)R and Return, then L and Return, and the disassembly will proceed just as if you never used R at all. If you don't intend to use the default address when you return to disassembly, it may be wise to note where you wanted to resume, or to use the / command before the R command. Merlin 16 prints ASCII during disassembly.

### S (SWEET 16 - Merlin 8 only)

This is similar to L, but forces the disassembly to start in SWEET 16 mode. SWEET 16 mode returns to normal 6502 mode whenever the SWEET 16 RTN opcode is found.

### T or TT (Text)

This attempts to disassemble the data at the current address as an ASCII string. Depending on the form of the data, this will automatically be disassembled under the pseudo-opcode ASC, DCI, INV or FLS. The appropriate delimiter ( " or ') is automatically chosen. The disassembly will end when the data encountered is inappropriate, or when 62 characters have been treated, or when the high bit of the data changes. In the last condition, the ASC opcode is automatically changed to DCI.

Sometimes the change to DCI is inappropriate. This change can be defeated by using TT instead of T in the command.

Occasionally, the disassembled string may not stop at the appropriate place because the following code looks like ASCII data to Sourceror. In this event, you may limit the number of characters put into the string by inserting a one or two digit hex number after the T command.

T or TT may also have to be used to establish the correct boundary between a regular ASCII string and a flashing one. It is usually obvious where this should be done.

### W, WW, or W- (Word)

This disassembles the next two bytes at the current location as a DA opcode. Optionally, if the command WW is used, these bytes are disassembled as a DDB opcode.

If W- is used as the command, the two bytes are disassembled in the form DA LABEL-1. The latter is often the appropriate form when the program uses the address by pushing it on the stack. You may detect this while disassembling, or after the program has been disassembled. In the latter case, it may be to your advantage to do the disassembly again with some notes in hand.

### / (Cancel)

This essentially cancels the last command. More exactly, it re-establishes the last default address (the address used for a command not necessarily attached to an address). This is a useful convenience which allows you to ignore the typing of an address when a backup is desired.

As an example, suppose you type T to disassemble some text. You may not know what to expect following the text, so you can just type L to look at it. Then if the text turns out to be followed by some HEX data such as $8D for a carriage return, simply type / to cancel the L and type the appropriate H command.

## MERLIN 16 SOURCEROR NOTES

Sourceror on the Merlin 16 disk is fully 65816 compatible. It attempts to follow the flow of the program in assigning the length of immediate operands, thus changing REP, SEP, and XCE when appropriate. Since this cannot always be successful, you can reset this mode by using an address with the L command. When an address is used with the L command, the M and X status bits are taken from the byte following the L. The default is MX = 00. Thus, 1012L01 and Return would restart the disassembly at $1012 with a long M and a short X, and MX would be set to 01.

The reversed DDB is not supported in the Merlin 16 version of Sourceror. Instead, you can use the WW command to generate a 4-byte long address using the psuedo-op ADRL.

The Merlin 16 Sourceror only uses the 80 column screen.

## SOURCEROR XL

For large object files, there is an alternative version of Sourceror on the Merlin 16 disk. To load it from the Main Menu Disk Command prompt, you would type BRUN SOURCEROR/XL and press Return. This version disassembles to disk instead of memory. The files are placed in the current prefix directory active at the time the DIS command is issued.

## FINAL PROCESSING

After the Q command, the program does some last minute processing of the assembled code. If you press Reset at this time, you will return to Merlin 8/16 and lose the disassembled code.

The processing may take from a second or two for a short program and up to several minutes for a long one. Be patient.

When the processing is done, you are returned to Merlin 8/16 with the newly created source in the Editor. You can use the Editor to edit or assemble the listing. After a successful assembly, you can save the new code with the Main Menu Save Source and Save Object Code commands.

NOTE: If you are using the ProDOS version of Merlin 8, you can enter the Editor and type USER1 to get rid of Sourceror and free up the memory used by the disassembler. This is not necessary in the DOS 3.3 version.

## MODIFYING THE FINISHED SOURCE

In most cases, after you have some experience and assuming you used reasonable care, the source will have few, if any, defects.

You may notice that some DA's would be more appropriate in a DA LABEL-1 or a DDB LABEL format. In such cases, it may be best to do the disassembly again with some notes in hand. The disassembly is so quick and painless that it is often much easier than trying to alter the source directly.

The source will have all the exterior or otherwise unrecognized labels at the end in a table of equates. You should look at this table closely. It should not contain any zero page equates except ones resulting from DA's, JMP's or JSR's. This is almost a sure sign of an error in the disassembly (yours, not Sourceror's). It may have resulted from an attempt to disassemble a data area as regular code.

NOTE: If you try to assemble the source under these conditions, you will get an error as soon as the equates appear. If, as eventually you should, you move the equates to the start of the program, you will not get an error, but the assembly *may not be correct.*

It is important to deal with this situation immediately since trouble could occur if, for example, the disassembler finds the data AD008D. It will disassemble it correctly, as LDA $008D. The assembler always assembles this code as a zero page instruction, giving the two bytes A5 8D. Occasionally you will find a program that uses this form for a zero page instruction. In that case, you will have to insert a character after the LDA opcode to have it assemble identically to its original form. Since it was data in the first place rather than code, it must be dealt with to get a correct assembly.

## THE MEMORY FULL MESSAGE

When the source file reaches within $600 bytes of the end of its available space you will see MEMORY FULL and be prompted to HIT A KEY. Sourceror will then go directly to the final processing. The reason for the $600 byte gap is that Sourceror needs a certain amount of space for this processing. There is an optional override provision at the memory full point. If you press Control-O for override, then Sourceror will return for another command. You can use this to specify the desired ending point. You can also use it to go a little further than Sourceror wants you to, and disassemble a few more lines. Obviously, you should not carry this to extremes. If you get too close to the end of available space, Sourceror will no longer accept this overide and will automatically start the final processing.

## CHANGING SOURCEROR'S LABEL TABLES

The label tables used by Sourceror are just assembled Merlin 8/16 source files. The source file is called LABEL.S and is on the Merlin 8/16 diskettes. It can be modified directly by the user. It must be assembled and saved under the same name as the previous label file, i.e. you have to replace the old existing file.

If you have several label tables you wish to use, you may want to just rename them. For example, you could keep TABLE.DOS, TABLE.PRODOS, etc. on the disk, and then just rename the file as you needed it to LABELS.

# APPLESOFT SOURCE LISTING

## SOURCEROR.FP

A fully labeled and commented source listing of Applesoft BASIC can be generated by the program called Sourceror.FP which is on side 2 of the Merlin 8 ProDOS diskette. Please note that *Sourceror* and *Sourceror.FP* are two entirely different programs. Sourceror is the disassembler in Merlin 8/16; Sourceror.FP is a separate program that produces the source listing of Applesoft BASIC. If you are looking for details on the disassembler, see the section on Sourceror.

Sourceror.FP works by scanning the resident copy of Applesoft present in your computer and generating text files called Aplsoft.A, Aplsoft.B, Aplsoft.C, and Aplsoft.D.

To conserve space, these files contain macros that are defined in another file on the disk entitled Applesoft.S. This file, when assembled using the PRTR command, will print out a nicely formatted disassembly of Applesoft, automatically bringing in and using the Apsoft files as necessary. Exact details on doing this are outlined below.

NOTE: This is not an official source listing from Apple Computer, Inc., but rather a product of the Author's own research and interpretation of the original Applesoft ROM. Apple Computer, Inc. was not in any way involved in the preparation of this data, nor was the final product reviewed for accuracy by that company. Use of the term Apple should not be construed to represent any endorsement, official or otherwise, by Apple Computer, Inc.

Additionally, Roger Wagner Publishing, Inc. makes no warranties concerning the accuracy or usability of this data. It is provided solely for the entertainment of users of the Merlin 8/16 assembler.

**WARNING: Sourceror.FP and some temporary work files will be deleted when Sourceror.FP is BRUN. For this reason, you should make a backup copy of the SOURCEROR.FP side of the Merlin 8 disk before proceeding. Use the backup copy to make the Applesoft listing as explained next.**

## PRINTING THE APPLESOFT SOURCE LISTING

To get a printed source listing of Applesoft BASIC, follow these steps:

1. Start the Merlin 8 ProDOS diskette.

2. The Merlin 8 Full Screen Editor uses the same part of memory as Sourceror.FP. To use Sourceror.FP, you will have to temporarily deactivate the Full Screen Editor.

   Press D for Disk Command.

3. At the Disk Command prompt type: BRUN /MERLIN.8/UTILITIES/REMOVE.ED and press Return.

4. Turn the disk over to the Sourceror.FP side.

5. At the Disk Command prompt type: BRUN /APPLESOFT/SOURCEROR.FP and press Return.

   A prompt appears to remind you that files will be deleted if you proceed (see warning above). If you are using a backup copy of Sourceror.FP, press Y to continue.

6. When SOURCEROR.FP finishes, type L to LOAD a source file.

7. At the Load prompt type: /APPLESOFT/APPLESOFT and press Return.

8. To format the listing type: PRTR1""APPLESOFT LISTING and press Return.

9. Type ASM and press Return.

The assembler will prompt you to answer several questions about the format of the printout. You have the options of:

a) PRINT THE DO OFF AREAS? (Y/N). Depending on whether you select the new ROM version (see next question), this choice gives you the option of printing those parts that are not actually assembled within the listing. Since the completed listing with XREF listings is on the order of 150 pages long, you probably won't want to print this listing too many times. We recommend answering Y to this first option.

b) ASSEMBLE NEW ROM VERSION? (Y/N). If you answer N, you'll get the Apple II+ version of Applesoft. If you answer Y, you will be prompted with:

c) ASSEMBLE IIC VERSION? (Y/N). If you have an Apple IIgs, IIc or Enhanced IIe, you will want to answer Y to this. If you want the original IIe version, answer N.

For each question, remember to just press the Y or N keys. You do not have to type "Yes" or "No."

In the example above, the PRTR command will send output to slot 1 and will print "APPLESOFT LISTING" as a header at the top of every page.

Merlin 8 will then ask "GIVE VALUE FOR SAVEOBJ :" This refers to whether or not you want to save object code generated by the assembly. It is recommended that you answer Ø. This is all you need to do to begin the printing process. If you answer 1 instead, you will save object code at the cost of slowing down the system. Saved object code allows you to verify it against where it was taken from.

Merlin 8 will now execute the first assembler pass. The disk will be accessed a few times, sometimes with long periods between accesses. This is normal. The entire first pass takes about 2 minutes.

Merlin 8 will then begin to print out a completely disassembled and commented listing of Applesoft. It will take about 105 pages including the symbol tables and nearly an hour and a half to print out at a printer rate of 80 characters per second.

# APPLESOFT SOURCE CROSS REFERENCE LISTING

Although 105 pages of Applesoft source would seem like enough to keep one busy for at least a year, Merlin 8 also offers another source of Applesoft internal information - Applesoft internal address, subroutine and zero page cross references. By using the XREFA utility with the Applesoft source you can produce a listing of every subroutine, zero page address and where they are used and called. This is invaluable information for the programmer who desires to make use of the routines inside Applesoft in his own programs.

Assume, for example, that a user program is called by a running Applesoft program. Also assume that the programmer makes calls to some internal Applesoft routines and that the programmer wishes to use zero page locations $50 and $51 as temporary registers or pointers. This cross reference will immediately inform the programmer whether or not the routines that his program uses will destroy the contents of these two locations and cause difficult to find bugs in his program.

**IMPORTANT**: You *must* produce the source files before they can be cross-referenced. This is done by following the instructions on pages 192-194. Do this now if you have not already done so.

Steps to print an Applesoft cross reference:

1. Start the Merlin 8 ProDOS diskette.

2. Insert the Sourceror.FP diskette.

3. Press L to LOAD, then type: /APPLESOFT/APPLESOFT and press Return.

4. Type Q to quit the Editor, and return to the Main Menu.

5. Insert the Merlin 8 ProDOS diskette.

6. Press D for Disk command, then type: BRUN /MERLIN.8/UTILITIES/XREFA and press Return.

7. Enter the Editor, the type: PRTR1""APPLESOFT XREF and press Return.

8. Then type: USER 3 and press Return.

9. Type ASM and press Return.

You'll be prompted as follows:

Print DO OFF areas ? (Y/N)

You may answer Y or N. {See the previous section for an explanation of these questions...}

Assemble new ROM version ? (Y/N)

If you answer N, you'll get the Apple II+ version.
If you answer Y, you'll be prompted with:

Assemble //c version ? (Y/N)
Answer Y for the IIgs, IIc and Enhanced IIe version.
Answer N for the original IIe version.

11) Insert the Sourceror.FP diskette at the prompt:
"Insert /APPLESOFT/APLSOFT.A.S"

12) Press Return to start the cross-reference process.

The Applesoft source will again be assembled. This time, however, the XREFA program will limit your printed output to the cross reference table. Note that this process also takes quite a bit of time prior to printing.

# UTILITY PROGRAMS

## AUTO EDIT (MERLIN 16 ONLY)

This is a powerful utility but caution should be used with Auto Edit. It can be used from the Main Menu by typing BRUN UTIL/AUTO.EDIT and pressing Return. Another version called AUTO.EDIT.2 can also be loaded in a similar manner. It is more interactive and less dangerous but it is also much slower.

This utility creates an Open-Apple-\ command which is only accepted if a range is currently selected. This command lets you input an auto-edit string which will act on all lines of the selected range except comment lines. You can use ^ and an alpha character for control character commands. For example, the string ^U^U^D01 will move the cursor over twice, delete the next character and then insert the 01 characters, assuming the insert cursor is active.

Auto Edit 2 changes each line according to the edit string and then waits for a keypress. Pressing Escape, Control-C, or Control-X will cancel the entire process. Any non-control character such as Space will accept the change and move to the next line. Any control character such as Return will reject the change and move to the next line in the selected text.

## CLASSIC DESK ACCESSORIES

### Calendar - Notepad - Rational Calculator

The Merlin 16 disk contains commented source files for three Classic Desk Accessories. These are Calendar, Notepad, and Rational Calculator and the various files are located in the Library subdirectory. Please note these files are only offered as demonstrations of Classical Desk Accessories on the IIgs. You may use these files in your own programs but they *may not* be used in any program intended for commercial distribution. As noted in the source files for these programs, Glen Bredon reserves the commercial rights and copyrights for these and all related files.

## CLOCK - MERLIN 8 (ProDOS only)

This Merlin 8 utility is an interrupt driven software clock designed for the IIc which lacks a clock to do the time stamping available in ProDOS. It requires the IIc because it uses the VBLINT interrupt provision. THIS UTILITY SHOULD BE USED WITH CAUTION! If it is overwritten, anything can happen and probably will. Press Reset to turn off interrupts. The source files are provided in the SOURCE directory on the Merlin 8 ProDOS version.

### CONV.LNK.REL (MERLIN 8 ProDOS ONLY)

This makes the Merlin 8 REL files compatible with Apple's RLOAD and RBOOT programs. It will convert a Merlin 8 LNK file to Apple's REL format only if there are no externals. You can BRUN it from the Main Menu. If there is a source file in memory already, it will just return. Thus, enter NEW first in the Editor if necessary before using it. You will be prompted for the pathname of the file to be converted. The program will do the conversion and set up the converted file for the Merlin 8 object save command. The CONV.LNK.REL utility does not write anything to disk and does not delete or otherwise damage the original file.

You will be prompted for the pathname of the file you want to convert. The program will do the conversion and set up the converted file for the Merlin 8 object save command. The CONV.LNK.REL utility does not write anything to disk and does not delete or otherwise damage the original file.

### CONVERTER: APW Source Files to Merlin 16 Format

Source files created by the APW (Apple Programmer's Workshop) and ORCA/M assemblers can be loaded directly into the Merlin 16 editor, but they usually require some editing before assembly. This is primarly due to differences in the pseudo-ops used by each assembler to define things such as data storage. For example, Merlin 16 can define a hex byte with the statement:

```
LABEL   HEX  FF
```

whereas APW would use:

```
LABEL   DC   I1'$FF'
```

to achieve the same result.

If the file name of the APW source file does not have the ".S" suffix, remember to put the reverse slash (/) at the end of the name you enter at the Main Menu of Merlin 16. For example, if the source file you wanted to load was named FILE.SRC, you would enter FILE.SRC/ as the file to load.

For short files, you can load the APW source file into the Merlin 16 editor, and edit or use the search & replace (Ó E) or find (Ó F) functions to change any offending syntax in each line. In the case of the above example, you could use the (Ó E) command to replace DC I1' with HEX. Notice the space is included in both the search and replace strings. You will have to hold down the Open-Apple key when you press Return to start the search & replace to tell Merlin 16 to do the search on parts of words in the source file. In addition, in this particular case, you will want to go back through and remove the single quote from the end of the $FF operand.

In addition, APW source files expand the space between each field in the source files with as many spaces as it takes to fill the gap. In Merlin 16, a single space character automatically tabs the text to the next field. Thus, source files in APW are up to 50% larger than the equivalent file in Merlin 16.

To remove the extra spaces in an APW source file, use the Command Box (ÓO) command FIXS to remove unneeded spaces in the file.

Finally, save the converted source file to disk under a new name. Merlin 16 will automatically add the ".S" suffix. For example, entering FILE as the name to save under would appear as FILE.S in the disk directory.

## Using CONVERTER

Rather than manually editing an entire source file, it is usually easier to use a utility provided on the Merlin 16 disk called CONVERTER. This utility will automatically make many of the changes necessary to convert an APW source file to a form more acceptable to Merlin 16. Converter is an Applesoft BASIC program that can be run from either a file selector like ProSel, the DeskTop, etc., or by first setting the prefix to /MERLIN.16/UTILITIES at the BASIC prompt, and then typing RUN CONVERTER.

When the Converter runs, the first step is to place the disk containing the APW file to be converted into a disk drive. Then choose menu item #1, Select a file from disk.

The program will then present a list of all the on-line disk volumes. Press the number key or use the arrow keys to select the disk you want to look at, and press Return.

The program will then display all the files in the main directory of that disk. Subdirectories are indicated by a slash at the end of the filename, for example, UTILITIES/. Selecting a subdirectory will then list all the files in that subdirectory. You can back out of a given subdirectory by pressing the Escape key.

When you see the file that you want to convert, press the number key and press Return.

The program will then load the existing source file into memory, make the necessary changes, and then write a new file back to disk. The suffix ".S" will automatically be added to the new filename.

After the conversion, you may either select another file to change, or you can press Open-Apple-Escape to go back to the Main Menu. From there, you can either exit to BASIC, a program selector, or select another file to change.

Most Apple IIgs programs make extensive use of the Apple IIgs Toolset macros. At the beginning of a converted program you'll probably find an instruction similar to USE FILE.MACROS. This tells the assembler to use a library of macro definitions stored on the disk. Rather than try to convert a macro file for a given program, it is better to use another Merlin 16 utility, MACGEN, to create a new library of the macros needed for the program you're converting. The next section describes how to use MACGEN.

## Using MACGEN

The Merlin 16 disk contains a complete set of Apple IIgs Toolset macro definitions, along with other macro libraries such as MACROS.816 and UTIL.MACS. The brute-force way of using these libraries is to add whatever lines are necessary to use the particular macros you need. For example:

```
USE   /MERLIN.16/TOOL.MACROS/MEM.MACS
USE   /MERLIN.16/TOOL.MACROS/MISC.MACS
etc.
```

The disadvantage to this approach is that the entire macro library is read into memory, leaving little remaining free memory for your actual program.

Usually you would write the program using whatever macro calls you wish, and then run the program utility MACGEN. MACGEN looks through an entire source file, and makes a list of every macro call. It then compares this with all the calls in any existing macro libraries, and creates a new and condensed macro file of just those definitions used in your program. MACGEN must be used when converting an APW source file in order to create the condensed macro file needed by a particular program.

When converting an APW source file, or when using MACGEN in your own program, here's what to do:

1) MACGEN will need to be installed in Merlin 16. This is done by typing:

```
-/MERLIN.16/UTILITIES/MACGEN
```

from the Main Menu.

2) Now load the source file to be scanned for macros. For an Apple IIgs, ProDOS 16 program, the first two actual instructions in your program, disregarding comment lines, should be:

```
REL
DSK   FILENAME.L
```

where FILENAME is the name of the final object file you wish to create, and FILENAME.L will be the intermediate relocatable link file. This LNK file will be used by the linker to create the final object file, presumably named FILENAME.

Following the REL and DSK instruction will be the USE FILENAME instruction that loads the condensed macro definition file, which would look something like this:

```
USE   FILE.MACROS
```

For now, this should be either deleted, or you can add a semi-colon (;) to temporarily make it a comment, since the file has not been created yet. After making these changes, save the file to disk.

3) The editor workspace must be empty for MACGEN to operate, so type NEW in the Command Box to erase the source listing. Then, go to the Command Box again and type:

```
MAC "FILENAME"
```

where FILENAME.S is the source file to be scanned.

4) The source file will be read from the current disk directory, and the Macro Generator menu will appear:

```
[S]earch directory for macros.
[D]isplay unresolved macro names.
[C]atalog directory.
[Q]uit macro generator.

Select:
```

MACGEN has already made a list of all the macro calls in your program, and pressing D will display them. This step is not required, but it may be of interest to you. You will now tell MACGEN where the macro libraries are that you want reconciled with the source listing.

Press S to start the search. MACGEN will ask what directory you want it to search within for existing macro libraries. In most cases, you will probably enter:

```
/MERLIN.16/TOOL.MACROS
```

but this can be any directory that has macro definition files in it.

As the macro files are read and compared to the source file, each filename will be printed on the screen. When the directory is processed, the menu above will be repeated. Since you will probably want to use MACROS.816 also, press S again and type in:

```
/MERLIN.16/GEN.MACROS
```

At this point, all macro definitions that could be found are copied into a new source file in the Merlin 16 editor. Before quitting MACGEN, you may want to press D to display any macros definitions that were not found. You can either make a note of them to add later, or you can press S again to scan another directory for more macros. Note that unresolved macros may be due to UPPER/lower case use that is inconsistent with the actual definitions. For example, _TLStartup might not be resolved because the actual name is _TLStartUp. Setting the PARMS file so that Merlin 16 is case insensitive is one way to eliminate this source of unresolved macros.

When you quit MACGEN, you'll want to save the condensed macro file for use by the converted APW program. For example, for the program SAMPLE, you might want to save the macro file under the name SAMPLE.MACROS. This would be referenced in the source program with a line like this:

```
USE    SAMPLE.MACROS
```

Once the macro file is created, you can load the source file, and put in the active USE instruction to include the macro file in your source, or remove the semi-colon you put in to temporarily make that instruction a comment.

**Note #1**: Upper/lower case of the filename is not important in Merlin 16 opcodes and filenames.

**Note #2**: The Merlin 16 pseudo-op USE cannot be used in a Put file. It can only be called from within the "master" calling program. If the program you are assembling has a USE macro library reference, try moving it to the master calling program. See the Merlin 8/16 manual regarding Put files.

**Nested Macros**: MACGEN looks within macros to see if other macros are called. Thus, if the macro _QDStartUp uses the macro Tool, the name Tool will be added to the list of unresolved macros. Usually, the macro will be found in other files, and so will be removed from the unresolved macro names before you even see it. However, if a macro definition is encountered in a library before it is referenced from within a later macro, you will have to make a second pass through that directory so that MACGEN will pick it up the second time. In practice, this is rarely needed.

### Final Editing

There are certain further substitutions and editing that will need to be done to complete the conversion process. The easiest way to find the lines that need changes is to try to assemble the file (ÓA), and see where the errors occur. Remember that before you can use Linker.GS, you will have to re-install Linker.GS by typing "-/MERLIN.16/LINKER.GS" from the Main Menu, since MACGEN displaces Linker.GS when it is loaded.

Some APW functions are duplicated in macro functions included in the UTIL.MACS and MACROS.816 files. Other functions in APW may have no equivalents, and so are converted to comments by the Converter program. Following is a list of points to look for:

**Comment/Line length**: The Converter program automatically shortens lines to the 64 character limit of the Merlin editor. Any text that is too long is turned into a comment with an asterisk or a semi-colon, and is made into a new line. This may make title boxes in the listing look uneven.

**Case Sensitivity**: In its original condition, Merlin 16 is case sensitive in its handling of labels. That is, it considers the label NAME to be different from Name or name. Case sensitivity gives you more versatility in labels, for example, using the macro QUIT for ProDOS 16 and Quit for ProDOS 8. It also results in faster assembly speeds. APW, however, is not specifically case sensitive unless the CASE directive is used. In addition, many sample programs available for the Apple IIgs written with APW use a wide variety of conflicting-case labels. For example, a single program may use pulllong, PULLLONG, Pulllong and PullLong in different parts of the program, all referring to the same macro.

When converting an APW source file, you may either manually edit any conflicting labels to be consistent, or you may alternatively change the PARMS file for Merlin 16 to make Merlin 16 case insensitive (about line #120 in PARMS.S).

**Duplicate Labels:** APW blocks groups of labels in each START/END segment and considers all labels in that segment to be local, unless declared "known" to another segment by the APW using pseudo-op. In Merlin 16, all labels are global unless preceded with a colon, as in :LOOP. In assembling a program, you may get "Duplicate Label" errors as Merlin 16 encounters labels that were used repeatedly in many different segments. To resolve these, you may either rename offending labels, i.e. LOOP1, LOOP2, etc., or you may put a colon in front of the offending label, i.e. :LOOP.

The main caution has to do with when a label is declared "known" to one segment, but may be local in all others. For example, suppose you have a program with three program segments, and the label LOOP was used in segments #2 and #3, but only the LOOP in segment #3 was known to segment #1 through APW's using pseudo-op. Since Converter turns it into a comment, you would have to pay particular attention to the preserved using instruction and to the corresponding START/END segment it referenced when editing the source listing. In this example, you might leave the label LOOP in segment #3 alone, and change LOOP in segment #2 to :LOOP. Note that references to local labels such as :LOOP cannot cross global labels. Thus, the following code segment would not work:

```
                BNE     :DONE

TEST2           CMP     #$7F
                BEQ     AGAIN

:DONE           RTS
```

**Processor Mode:** Merlin 8 and Merlin 16, as configured in the original package, start assembling a file assuming that the microprocessor is in the 8-bit mode (MX %11). In the case of Merlin 16, this is done to minimize errors when assembling source files written with earlier (8-bit) versions of Merlin, such as Merlin, Merlin Pro, Merlin 8 or Big Mac. When assembling programs to run under ProDOS 16, it is very important that you make sure the assembler starts assembling the source in the full 16-bit (MX %00) mode. There are two ways to handle this. The first is to just include the Merlin pseudo-op MX %00 on one of the first lines of each source file. You may alternatively use the macro M65816 1, as defined in the Macros.816 file, which accomplishes the same function. The Converter program automatically puts the proper instructions in when converting an APW program, so this is only a concern when manually converting a program, or writing a ProDOS 16 program from scratch.

The other choice is to permanently change the appropriate bit in the flag in the Merlin 16 PARMS file (approx. line #75) so that Merlin 16 always starts in the 16-bit mode. Important: If you do make this change, you will have to be careful to remember to use an MX %11 (8-bit mode) instruction in any of your older ProDOS 8, etc. source files before assembling them with the altered Merlin 16.

In addition, here are some of the APW pseudo-ops that are automatically handled by the Converter, but which may be of some interest:

**65816 ON** In APW, this enables the 65816 assembly mode. In Merlin 16, this is equivalent to two XC instructions, plus an MX %00. This is automatically added to converted APW source files by Converter, but there is also a macro named M65816 in MACROS.816 that can be used.

**LONGI ON/OFF** Merlin 16 uses 0 and 1 as substitutes for APW's ON and OFF operands. LONGI and LONGA have equivalent macros in MACROS.816.

**START/END** APW delimits each program segment within a listing with a START and END pseudo-op. All labels within a segment are considered local by APW except for the label that is associated with the START instruction. Local labels in APW within a segment may be made known to other segments with the APW "using label" instruction, which is not the same as Merlin 16's "using" instruction. The Converter program changes START to ENT so that the particular label on that line can be referenced by other Merlin 16 source files, if needed. END is converted to a comment, since END in Merlin 16 would signify the end of the entire source file. This structure may have to be considered when handling local/global labels and label conflicts.

**SYMBOL ON/OFF** In APW this determines whether the symbol table should be printed at the end of the source listing. In Merlin 16 this is controlled by the presence of LST ON/OFF at the end of the listing.

**GEN ON/OFF** In APW, GEN ON tells the assembler to print the expanded form of macros. EXPMAC is an equivalent function macro in Merlin 16 that duplicates this. Note: EXP is the actual Merlin 16 directive. See EXP in the Merlin 8/16 manual.

**Programmer's Note:** The Converter uses the file DICTIONARY in the UTILITIES subdirectory in part of the conversion process. The Dictionary file contains search & replace entries that translate APW opcodes to Merlin 16 equivalents. For example, the file Dictionary has the entry:

```
COPY#PUT
```

This tells the converter to replace the opcode COPY with PUT. Thus the APW source line:

```
COPY   HIRES.STUFF
```

would become:

```
PUT   HIRES.STUFF
```

The pound sign (#) is the delimiter between the search and replacement strings. Replacement strings may include spaces and semi-colons to move the new statements to the comment field, as is done with END becoming ;END. A pseudo-op can be removed completely by leaving the replacement field blank.

You can edit the Dictionary file with Merlin 16 to add new search & replace definitions. Remember to use the filename DICTIONARY/ when loading or saving. You will probably also want to turn off the formatting (tabbing to assembler fields) with the Command Box TAB command.

The Converter changes ON and OFF arguments to certain APW directives, such as absaddr, 65816, case, longa, longi, etc., to 1 and 0 so that the ON/OFF argument can be handled by an equivalent Merlin 16 macro. These macros are presently defined in the file MACROS.816 in the GEN.MACROS subdirectory on the Merlin.16 disk.

The source files for the Converter utility are also included on the Merlin.16 disk. Although we do not provide specific technical support for making alterations to this program, the source files are well-commented, and you may wish to make changes to the program for your own use.

**Sample APW Program:** A sample program, called SIMP.SRC, is included on the Merlin 16 disk in the directory SAMPLES.APW. You may want to try the procedure described here to practice converting an APW source file before tackling other files. A copy of the SIMP.SRC in converted Merlin format, and the final output file, SIMP, is included in the SAMPLES.M16 directory. SIMP is a ProDOS 16 S16 type file that can be launched from the Finder or DeskTop.

As a general exercise in assembling, linking and running a ProDOS 16 application that uses many Apple IIgs tools, there is also a program in the Samples.M16 directory called BRICKOUT. This file can be assembled, linked and saved by linking the command file GAME.CMD.S. This file was converted from a listing originally in the APW format.

## CROSS REFERENCE PROGRAMS

### Xref - Xrefa

These utilities provide a convenient means of generating a cross-reference listing of all labels used within a Merlin 8/16 source program.

Such a listing can help you quickly find, identify and trace values throughout a program. This becomes especially important when attempting to understand, debug or fine tune portions of code within a large program.

Merlin 8/16 provides a printout of its symbol table only at the end of a successful assembly, provided that you have not defeated this feature with the LST OFF pseudo op code. While the symbol table allows you to see what the actual value or address of a label is, it does not allow you to follow the use of the label through the program.

This is where the XREF programs come in.

XREF gives you a complete alphabetical and numerical printout of label usage within an assembly language program. XREFA gives a cross reference table by address. This is more useful for large sources containing lots of PUT files. It also does not use as much space for its cross-reference data and therefore can handle larger source files than XREF.

### Sample Symbol Table Printout

```
Symbol table - alphabetical order:

   ADD      =$F786       BC       =$F7B0       BK       =$F706

Symbol table - numerical order:

   BK       =$F706       ADD      = $F786      BC       =$F7B0
```

### Sample Xref Printout

```
Cross referenced symbol table - alphabetical order:

     ADD      =$F786  101   185*
     BC       =$F7B0  90    207*
     BK       =$F706  104   121*

Cross referenced symbol table - numerical order:

     BK       =$F706  104    121*
     ADD      =$F786  101    185*
     BC       =$F7B0  90     207*
```

As you can see from the above example, the definition or actual value of the label is indicated by the equal (=) sign, and the line number of each line in the source file that the label appears in is listed to the right of the definition. In addition, the line number where the label is either defined or used as a major entry point is suffixed or flagged with an asterisk (*).

An added feature is a special notation for additional source files that are brought in during assembly with the PUT pseudo opcode. For example, 134.82 indicates line number 134 of the main source file is the line containing the PUT opcode and line number 82 of the PUT file is where the label is actually used.

### Using Xref

1. From the Main Menu, make sure you have saved the file and Merlin 8/16 is in the currently selected drive.

2. Merlin 16: Press D for Disk Command.
   At the Disk Command prompt type BRUN /MERLIN.16/UTILITIES/XREF and press Return.

   Merlin 8 ProDOS version: Press D for Disk Command.
   At the Disk Command prompt type BRUN /MERLIN.8/UTIL/XREF and press Return.

   Merlin 8 DOS 3.3 version: From the Main Menu press C to catalog the disk. At the Command prompt type BRUN XREF and press Return.

3. Enter the Editor, then type the appropriate USER command:

   USER 0 - Print assembly listing and alphabetical cross reference only. USER has the same effect as USER 0.

   USER 1 - Print assembly listing and both alphabetical an numerically sorted cross reference listings.

   USER 2 - Do not print assembly listing but print alphabetical cross reference only.

   USER 3 - Do not print assembly listing but print both alphabetical and numerical cross reference listings.

   For example, to print a cross-reference listing only to your printer, you could type in:

```
PRTR 1""
USER 3
ASM
```

   USER commands 0-3 cause labels within conditional assembly areas with the DO condition OFF to be ignored and not printed in the cross reference table.

   You can also use USER commands 4-7 which are identical to USER 0-3 except that they cause labels within conditional assembly areas to be printed regardless of the DO setting. The only exception to this is that labels defined in such areas and not elsewhere will be ignored.

   **NOTE:** You may change the USER command as many times as you wish, i.e. from USER 1 to USER 2 and so on. The change is not permanent until you enter the ASM command.

4. Merlin 16: Press Open-Apple-A to begin the assembly and printing process.

   Merlin 8: Type ASM and press Return to begin the assembly and printing process.

Since the XREF programs require assembler output, code in areas with LST OFF will not be processed and labels in those areas will not appear in the table. In particular, it is essential to the proper working of XREF that the LST condition be ON at the end of assembly since the program also intercepts the regular symbol table output. For the same reason, the Control-D flush command must not be used during assembly. The program attempts to determine when the assembler is sending an error message on the first pass and it aborts assembly in this case, but this is not 100% reliable.

**NOTE:** When using macros with XREF, labels defined within macro definitions have no global meaning and therefore are not cross-referenced.

```
DEF       MAC                 <---Macro definition
          CMP  #]1
          BNE  DONE
          ASL
DONE      <<<
          ----------------------   <---Beg. of program
          DEF  GLOBAL         <---Macro call
```

In the above example, variable GLOBAL will be cross referenced, but local label DONE will not.

### XREFA Notes

This is an *address* cross reference program and is handy when you have lots of PUT files. Since this program needs only four bytes per cross reference instead of six, it can handle considerably larger sources. The "where defined" reference is not given here because it would equal the value of the label. The exception is EQUated labels where it would indicate the address counter when the equate is done. This also saves considerable space in the table for a larger source.

## FORMATTER

This program is provided to enhance the use of Merlin 8/16 as a general text editor. It will automatically format a file into paragraphs using a specified line length. Paragraphs are separated by empty lines in the original file.

To use Formatter, you should first BRUN it from Main Menu. Formatter will then load itself into high memory.

This will simply set up the editor's USER vector. To format a file which is in memory, issue the USER command from the editor.

The formatter program will request a range to format. If you just specify one number, the file will be formatted from that line to the end. Then you will be asked for a line length, which must be less than 250. Finally, you may specify whether you want the file justified on both sides, rather than just on the left.

The first thing done by the program is to check whether or not each line of the file starts with a space. If not, a space is inserted at the start of each line. This is to be used to give a left margin using the editor's TAB command before using the PRINT command to print out the file.

Formatter uses inverse spaces for the fill required by two-sided justification. This is done so that they can be located and removed if you want to reformat the file later. It is important that you do not use the FIX or TEXT commands on a file after it has been formatted unless another copy has been saved. For files coming from external sources, it is desirable to first use the FIX command on them to make sure

they have the form expected by Formatter. For the same reason, it is advisable to reformat a file using only left justification prior to any edit of the file.

Don't forget to use the TABS command before printing out a formatted file.

## KEYBOARD MACRO FILES

### Edmac - Keymac

Edmac and Keymac are keyboard macro files for use with the Merlin 8/16 Editors. Edmac is for use with Merlin 8 and Merlin 16 full screen editors. Keymac is for the Merlin 8 line editor. A macro definition lets you type one key, and get a string of characters on the screen. This should not be confused with the assembler macros that Merlin 8/16 also supports.

An assembler macro is a definition of a set of assembler instructions, usually with variables, that you define within a given source listing. When the program is assembled, the assembler replaces the macro call with the series of lines that have been assigned to that macro.

A keyboard macro is only a substitute for a small amount of typing that you might do while you're in the editor itself.

For example, you've probably typed LDA many times in assembly language programs. With Edmac or Keymac installed, you could type Solid-Apple-3 (Option-3 on the Apple IIgs) and the characters LDA # would appear on the screen.

To install Edmac or Keymac, just BRUN it from the Main Menu, then enter the Editor and type USER from the Merlin 16 Command Box or Merlin 8 Command Mode. When you type one of the Edmac or Keymac Solid-Apple (Option- on the GS) commands shown on the following pages, the corresponding text will be inserted.

# KEYBOARD MACRO EQUIVALENT CHART

```
What You Type:     To Get:                Comments

 A                 1     AND
 B                 2     BVC
 C                 3     CMP
 D                 4     DFB
 E                 5     EOR
 F                       N/A              Keyboard Macro Not Assigned...
 G                       N/A              Keyboard Macro Not Assigned...
 H                 6     HEX
 I                       N/A              Keyboard Macro Not Assigned...
 J                 7     JSR
 K                 8     JMP
 L                 9     LDA
 M                10     BMI
 N                11     BNE
 O                12     ORA
 P                13     BPL
 Q                14     BEQ
 R                15     RTS             ; PLUS CARRIAGE RETURN
 S                16     STA
 T                       N/A              Keyboard Macro Not Assigned...
 U                       N/A              Keyboard Macro Not Assigned...
 V                17     BVS
 W                       N/A              Keyboard Macro Not Assigned...
 X                18     LDX #
 Y                19     LDY #0          ; LEAVES IN INSERT MODE
 Z                20     LD_ #0          ; LEAVES IN INSERT MODE
 1                       N/A              Keyboard Macro Not Assigned...
 2                       N/A              Keyboard Macro Not Assigned...
 3                21     LDA #
 4                22  LABEL  =  $        ; FOR HEX EQUATES
 5                23  LABEL  DFB %       ; FOR BINARY EQUATES
 6                24     N/A              Keyboard Macro Not Assigned...
 7                25          " "&$9F    ; USED TO DEFINE A CONTROL CHAR.
 8                26  *-------------------------------
 9                27          (),Y       ; WITH INSERT MODE ON...
 0                28          (,X)       ; WITH INSERT MODE ON...

 ^J               29     PLA             ; SAVE A,X,Y ON STACK
 (down arrow)     30     TAY             ; ALL 5 LINES WITH ONE MACRO.
                  31     PLA             ; (MACRO KEY = DOWN ARROW)
                  32     TAX
                  33     PLA

 ^K               34     PHA             ; RETRIEVE A,Y,Y FROM STACK
 (up arrow)       35     TXA             ; ALL 5 LINES WITH ONE MACRO.
                  36     PHA             ; (MACRO KEY = UP ARROW)
                  37     TYA
                  38     PHA
```

```
 '          39     ASC ''          ; SINGLE QUOTE WITH CURSOR BETWEEN QUOTES
 "          40     ASC ""          ; DOUBLE QUOTE WITH CURSOR BETWEEN QUOTES
 \          41     ERR \           ; FOR 'ERR' CHECK VALUE
 .          42     MAC             ; DEFINE A MACRO.
 ,          43     EOM             ; FINISH DEFINITION.
 =          44     ADC             ; = (SAME KEY AS '+')
 -          45     SBC             ; "-" FOR SUBTRACTION.
 ]          46     ]LOOP           ; VARIABLE FOR A LOOP.
 [          47     []              ; INDIRECT LONG ADDRESSING MODE.
```

Macros can be added or edited by adding or changing the data statements in KEYMAC.S or EDMAC.S. The only requirement for each definition is that it begin with the macro key itself, and that the assigned string end with a "high bit off" (value < $80) ASCII character.

## MAKE DUMP

This Merlin 8/16 program will make a hex dump text file from an object file after a valid assembly of the source. Before using Make Dump, you should have already assembled and saved the source code. This is important because the existing source code in memory is destroyed when Make Dump is used. To use Make Dump, BRUN it from the Main Menu. Then load the desired source file and assemble it. Finally, type USER from the Command Box or Command Mode of the Editor.

The current source in memory will be replaced with a hex dump of the previously assembled object file. The last line of the file will contain a BSAVE with the correct address and length but *without a filename*. You can use the Editor to insert the desired filename. You can then save the file for use as an EXEC file. If you forget to insert the filename, Merlin 8/16 will give you a SYNTAX ERROR when the file is EXECed.

## PRINTFILER

Printfiler is a utility designed to save an assembled listing to disk as a sequential text file. It optionally allows you to also select "file packing" for smaller space requirements and allows you to turn video output off for faster operation.

Text files generated by Printfiler can include the object code portion of a disassembled listing, something not normally available when saving a source file. This allows a complete display of an assembly language program and provides the convenience of not having to assemble the program to see the object code.

Printfiler can also be used to save a source listing to disk as a text file with the high bit clear (Merlin source files normally have the high bit set), and with the option of each tab field filled with spaces for compatibility with the APW (ORCA) assembler.

### Printfiler Applications

Other examples of where Printfiler might be used include:

- Incorporating the assembled text file in a document being prepared by a word processor.
- Sending the file over a telephone line using a modem.
- Mailing the file to someone such as a magazine editor who wants to work with the complete disassembly without having to assemble the program.

### Using Printfiler

1. Make sure that you've saved the source file before using Printfiler.

   Merlin 16 version: Press D for Disk Command.
   At the Command prompt type BRUN /MERLIN.16/UTILITIES/PRINTFILER and press Return.

   Merlin 8 ProDOS version: Press D for Disk Command.
   At the Command prompt type BRUN /MERLIN.8/UTILITIES/PRINTFILER and press Return.

   Merlin 8 DOS 3.3 version: Press C to catalog the Merlin 8 disk.
   At the Command prompt type BRUN PRINTFILER and press Return.

2. Press L to load the desired source file.

3. From the Main Menu select the drive to save the assembly to if necessary.

   Merlin 16 version: Enter the Editor and press Open-Apple-O to open the Command Box. From the Command Box type USER "FILENAME" where FILENAME is the name of your file. You may also use the PRTR command if you wish page headers to be sent with your listing. In that case enter the following instead of the USER command: PRTR 1 "T.MYFILE" MY PAGE HEADER and press Return.

   Merlin 8 version: Enter the Editor by pressing E from the Main Menu. From the Command Mode prompt (:) type USER "FILENAME" where FILENAME is the name of your file. You may also use the PRTR command if you wish page headers to be sent with your listing. In that case enter the following instead of the USER command: PRTR 1 "T.MYFILE" MY PAGE HEADER and press Return.

   There will be a short disk access and the name of the file used for the text save will appear on the screen.

4. Merlin 16 version: Press Open-Apple-A to begin the assembly and Printfiler will automatically assemble the source file directly to disk. Note that you will not see anything on your video screen because Printfiler is preconfigured to operate with the video output turned off for faster operation.

   Merlin 8 version: From the Command Mode prompt type ASM and Printfiler will automatically assemble the source file directly to disk. Note that you will not see anything on your video screen because Printfiler is preconfigured to operate with the video output turned off for faster operation.

   NOTE: You should not use a filename that already exists on the disk as an object file for the output file. Printfiler uses the exact filename with no prefix or suffix that you enter. Thus, if you had the source file PROGRAM.S and the object file PROGRAM on a disk, you would not want to use the name PROGRAM for the PRINTFILE name. PROGRAM.TEXT would be the recommended name.

   Printfiler will also print the output from the Merlin 16 "L" (list with line numbers) and "P" (Print without line numbers) commands. To output a text file for the APW (ORCA) assembler with each tab field automatically filled with spaces, just use the "P" command in the command box after opening the text file to print the source file to a formatted text file with the high bit clear. This file can then be directly loaded by APW.

### Changing Printfiler Options

Printfiler has two options that you may change: file packing and video output or echoing. In addition, you can make the change temporary or permanent.

File packing reduces the size of the text file saved to disk by replacing blanks from the source file with a single character with its high bit turned off. A listing of a packed file will display the packed blank characters as an inverse letter. Thus inverse A=1 blank, inverse B=2 blanks, inverse C=3 blanks, etc.

Unpacking means restoring the text file to its original appearance. Note that while you cannot assemble such a file, you can at least read it.

Echoing means printing on the screen what is sent to the disk. The time it takes to do this can slow Printfiler down.

The process of turning off video output or echoing makes Printfiler run approximately 25% faster. Additional speed can be gained by using packed files.

In addition, unpacked files are nearly twice as large as packed files and nearly three times the size of the original source file.

### Changing Printfiler Options (temporarily)

To temporaily change the Printfiler options you can go to the Monitor from the Merlin 8/16 Editor with the MON command. Enter the following:

```
300:00 00   for packed, video off, or.
300:00 80   for packed, video on, or
300:80 00   for unpacked, video off, or
300:80 80   for unpacked, video on
```

The Printfiler standard values are 300:80 00 (unpacked, video off)

Press Return, then Control-Y, then Return again to go back to Merlin 8/16. The values you select will stay in effect until you BRUN PRINTFILER again.

### Changing Printfiler Options (permanently)

1. Load Printfiler as above and assemble with it. During assembly, it will prompt you with the following questions:

2. GIVE VALUE FOR FORMAT
   Press 0 to turn on the Pack option.. Press 1 to turn off the Pack option.

3. GIVE VALUE FOR MONITOR
   Press 0 to turn video output or echoing off. Press 1 to turn echoing on.

   Printfiler will then immediately assemble into object code.

4. Quit the editor and save the Object code. Any time you BRUN this object code, it will use the values you specified in steps 2 and 3. Thus, it is possible to use different versions of Printfiler instead of setting options.

## TXTED (MERLIN 16 ONLY)

This is an alternate version of the Merlin 16 Full Screen Editor which will break a line upon a carriage return, and carriage returns are deletable. TXTED can be substituted from the Main Menu by typing BRUN UTIL/TXTED and pressing Return.

## TYPE.CHANGER

This utility will change the ProDOS file type of any file on a disk. It is an Applesoft BASIC program that can be run from either a file selector like ProSel, the DeskTop, etc., or by first setting the prefix to /MERLIN.16/UTILITIES at the BASIC prompt, and then typing RUN TYPE.CHANGER.

When Type.Changer runs, the first step is to place the disk containing the file whose type is to be changed into a disk drive. Then do the following:

1) Choose menu item #1, Select a file from disk.

2) The program will then present a list of all the on-line disk volumes. Press the number or use the arrow keys to select the disk you want to look at, and press Return.

3) The program will then display all the files in the main directory of that disk. Subdirectories are indicated by a slash at the end of the filename, i.e. UTILITIES/. Selecting a subdirectory will then list all the files in that subdirectory. You can back out of a given subdirectory by pressing the Escape key.

4) When you see the file whose type you want to change, press the number key and press Return.

5) The program will then display the current filetype of that file showing both abbreviated file code and hex filetype. For example, an Applesoft BASIC file would be listed with a file code BAS and a filetype of $FC.

6) You can enter the new filetype you want as either the "official" file code, such as BIN, S16, etc., or the hex code, such as $FF. If entering the hex code, you must include the dollar sign at the beginning of the number.

7) After the change, the new file code and type will be displayed. Pressing a key will return to the Main Menu. At that point you can either exit to BASIC, a program selector, or select another file to change.

NOTE: The file codes and types are all contained in DATA statements within the Type.Changer program. These may be added to or edited as you desire to support new and/or custom file types.

## ADDITIONAL MERLIN 8/16 RESOURCE FILES

The Merlin 8/16 disks contain a number of source files for your reference including PUT file examples, general macros that can be used in any program, Classic Desk Accessories, and Toolbox macros for the IIgs. These files are provided as resource material and can be helpful in understanding how these tools are designed and implemented.

# INDEX

"!", Exclusive OR 81
"&", Logical AND 81
"/ ( , - .", in Macros 131
".",
  as a logical OR 77, 81
"/",
  to abort a Change 38
  Division operator 81
  Editor - to List from last line 46, 69, 72
  Editor - to abort List 42
  in file names 18
  to abort a Find 39
  Line Range Delimiter 36, 37, 61, 62
  in Macros 131
  Sourceror (Cancel) 189
* for Comments 5, 77, 78
. (period),
  Listings 46, 69, 72
  in Macros 131
256 Externals 183

A

A:Append File 19
About the Assembler Documentation 75
About the Command Mode Documentation 36
About the Command Box Documentation 61
About the Linker Documentation 138
Absolute addresses, and Linker 140, 144, 151
Accumulator mode addressing 83-85
Add/Insert Mode Editing Commands 5-7, 8-9, 37, 63, 72
Addition operation, in operands 81
Addressing Modes 83-85
ADR 111
  Linker command 152
ADRL 111
ALI 159
All text, to select 29, 34, 57, 71
AND operation, in operands 81-82
Angle brackets in Documentation 36, 61
Append a file 19
Apple Keys (Ó)
  ÓA 9, 52, 71
  ÓB 27, 34, 52, 71
  ÓC 52, 71
  ÓD 27, 34, 53, 71
  ÓE 27, 34, 53, 71
  ÓF 28, 34, 54, 71
  ÓH 54, 71
  ÓI 28, 34, 55, 71
  ÓL 28, 34, 55, 71
  ÓN 29, 34, 55, 71
  ÓO 8, 55, 71
  ÓQ 8, 12, 29, 34, 55, 58, 71
  ÓR 29, 34, 56, 71
  ÓT 30, 34, 56, 71
  ÓV 30, 34, 56, 71
  ÓW 30, 34, 56, 71
  ÓX 30, 34, 57, 71
  ÓY 31, 34, 57, 71
  ÓZ 30, 34, 57, 71
  Ó Up 31, 34, 58, 71
  Ó Down31, 34, 58, 71
  Ó Right71
  Ó DEL 31, 34, 57, 71
  Ó TAB 31, 34, 55, 58, 71
  Ó - 31, 34, 60, 71
  Ó = 31, 34, 60, 71
  Ó1 58, 71
  Ó2 59, 71
  Ó3 59, 71
  Ó4 59, 71
  Ó6 59, 71
  Ó8 32, 34, 59, 71
  Ó9 32, 34, 71
  ◆ 60, 71
Applesoft source listing 192
APW to Merlin Converter 198-205
Arithmetic and Logical Expressions 81-82
ASC 107

Assembler,
to Pause 121, 123
Pseudo Opcode Descriptions 87
Syntax Conventions 77, 83
Assembling large files, and PUT, SAV, DSK 93-96, 98-99, 100-101, 147-148, 152, 153, 155-157
Assembly 9-11, 47, 52, 63, 73
ASM,
Command 9-11, 47, 63, 72
and PRTR 43-44, 47, 66, 72
Linker command file 151, 152
AST 103
Asterisks (*),
Comments 5, 77, 78
Line of in a comment 31, 34, 59, 71, 103
Auto edit 73, 197

B

Back-up copies of Merlin 3
Backing up Program Counter (DS) 89, 112
Backwards DELETE, in EDIT mode 26, 33, 50, 70 (see also Control-D)
Bad,
Address mode 179
Branch 179
External 179
Input 179
Label 179
OBJ 179
Opcode 179
ORG 179
PUT 180
REL 180
SAV 180
Variable 180
Beginning, move to,
of line of text 24, 33, 48
of source file 27, 34, 52
Bell, turning off 167, 172
BINary files 12, 13, 17
Binary numbers 80
BGE opcode 86
BLOAD 14, 19, 20, 21, 89
Block cursor 24, 26, 49, 167
BLT opcode 86
Branching,
to Variables and Local Labels 79, 86
BRK 180
BRUN 19, 20, 21
BSAVE 19
Bugs, common cause of 87
Building Expressions 81-82

C

C: Catalog 17-18
pause 17-18
CALL 14
Case sensitive labels 88
Center screen 30, 57
Change 38
Change drive 19
Change ProDOS filetype 214-215
Change Word 39
Changing Printfiler's Options 213-214
Changing Sourceror's Label Table 191
Character case change 24, 33, 49, 62
Character insert mode 24, 26, 33, 49, 70
Characters per line 34, 63, 73, 77, 167, 170, 172
Checksum, in object code 119-120
CHK 119-120
Classic Desk Accessories 197
Clipboard 30, 34, 52, 73
Clock 197
Command box 8, 55, 61
Command mode (Merlin 8) 15, 34, 36
Command summary 33-34, 70-72
Comments 5, 77
Comment length 34, 63, 73
Conditional Assembly 82-83, 116-118, 151, 152

Conditional Pseudo Ops 114-118
Configuration 4, 72, 166
Control characters 25, 33, 49, 70
Control-B (go to line begin) 24, 33, 70
Control-C,
during Catalog Command 17
to abort assembly 33, 47, 48, 70, 73
to abort List 42
to abort a Change 38
to abort a Find 39
or Control-X (to cancel lines) 34
Control-D (delete) 9, 24, 33, 49, 70
LST ON/OFF 47
Control-F (find) 24, 33, 49, 70
Control-I (insert) 24, 26, 33, 49, 70
Control-L 24, 33, 49, 70
Control-N (go to line end) 25, 33, 49, 70
Control-O (insert "other" character) 25, 33, 49, 70
Control-R (restore line) 25, 33, 70
Control-S (status box) 25, 50, 70
Control-T 25, 33, 50, 70
Control-W 25, 33, 50, 70
Control-X (to Cancel global exchange) 25, 33, 50, 70
Control-Y 25, 33, 50, 70
Control-Y, to return to Main Menu 15
Conv.lnk.rel 198
Converter, APW to Merlin source 198-205
Copy,
Command Mode, Editor 38
Full Screen Editor 52, 71
Copying Merlin 3
Cursor keys 26, 31
Cursors,
appearance/types 24, 26, 49, 70, 167
moving 26, 33, 50
Customizing Merlin 4, 72, 166
Cut 30, 33, 56, 57
CW (Change Word) 39
CYC 103-104
Cycle times, 103-104
column to print 167, 172

D

Disk Commands 19
D:Drive Change 19
DA 110
DAT 104, 153
Data,
definition of 76
immediate 76, 83
string 74, 107-109
storage 110-113
tables in programs 113
Data and Storage Allocation Pseudo Ops 76
Date, to set 22
DB 110
DCI 108
DDB 110
Decimal numbers 80
Defining,
a keyboard macro 209-211
a local label 78-79
a local variable 79
a Macro 127, 128-132, 133-134
Delete,
characters in a line 9, 24, 26, 33, 49, 50, 70
lines in Editor Command Mode 39
lines in Full Screen Editor 9, 27, 52
to end of line 25, 33, 50
entire Source File (New) 21, 43, 65
files on disk 19
Delete key 26, 33, 50, 70
Delimited Strings, 37, 61, 76
as an operand 76
DEND 102
DFB 110
Dictionary full 180
Disassembling,
raw object code (Sourceror) 184-191

Disk files,
names - see "filenames"
renaming 19
Division operation, in operands 81
DO 114
DOS 3.3 technical information 164
Drive change 19
DS 90, 112
and Linker 112, 139, 143, 159
DSK 90, 100
and the Linker 147, 149, 155-156
DUM 101
Duplicate symbol 146, 180
Duplicating Merlin disk 3
DW 110

E

Edit command 38
Edit Mode Commands 33-34, 70-72
Editor 5-8, 21
Edmac 35, 73, 209-211
ELSE 114
END 101, 153
End of line,
marker 23, 48
move to 25, 33, 49
End of source, move to 29, 34, 55
ENT 88, 141, 142, 153
EOM or <<< 127
EQU (=) 87
ERR 120
and Linker 120-121, 139, 143
ERR: Memory Full 181
Error Messages (general) 179-183
Escape 26, 33, 50, 70
Evaluation of expressions 45, 69, 72, 81
Exchange, global 27, 34, 53
EXCLUSIVE OR operation, in operands 81-82
EXP ON/OFF/ONLY 104
EXT 141, 153
Expressions Allowed by the Assembler 81-83

F

Fast keys 50
Filenames,
ASM 151
DSK 90, 91, 100-101, 147, 155-156, 161
LIB 154
Link 144
Linker command files 154-155
Linker name files 144-145, 148-149
LNK 147, 152, 155-156
Multiple LNK files 160
object 12, 13, 14, 19, 2
OVR 152
PUT 77, 93-96, 152
Quick Link 159-160
REL 147, 155-156, 161
SAV 90, 92, 98-99, 153
source 12, 13, 18, 19, 21, 22, 77
USE 77, 97, 136
Filetype mismatch error 146, 180
FIN 115
Find,
a character 24, 33, 49, 70
a string 25, 28, 39, 54, 71
a word 25, 28, 30, 39, 40, 50, 54, 56
Find and Replace (global exchange) 25, 27, 34, 38, 53
FIX 40, 63, 72
FLS 108
Forced Assembly Errors 120, 121, 143
Formfeed, printer 166, 170
FW (Find Word) 40
Format error 180
Formatter 208-209
Formatting Pseudo Ops 103-106
Full Screen Editor,
commands 8-9
Merlin 8 23-35
Merlin 16 48-60
entering 5, 21, 23, 48
quitting 7, 44, 66, 71

G

GET 15, 41, 64, 72
GET Command, and Linked files 146
Global Exchange 25, 27, 34, 38, 53
GS Linker 158, 161-162, 163

H

Half-screen editing 54
H (Hex) 187
Hard disks 170
HEX 111
Hex-Dec Conversion 41, 64
Hex data 80, 107, 110

I

IF 115
Illegal,
  char in operand 180
  file type 181
  forward reference 181
  relative adrs 181
Initialization string, for printer 43, 66, 175
Insert,
  with TAB key 24, 26, 28, 49, 50
  character mode 9, 26, 49, 50
  lines 8, 26, 28, 55, 58
  control characters 25, 49
Integer division, in operands 81
Inverse spaces 44, 67
Immediate Data Syntax 80, 83
Immediate Data vs. Addresses 80
INV 108

K

KBD 121
Keyboard input during assembly 121
Keymac 35, 209-211
KND 159

L

L (list - Sourceror) 187
L: Load Source 18
Labels,
  proper form of 6, 77
  length 77
  tables, changing Sourceror's 191
  case sensitivity 88
  jump to 28, 34, 55, 70
LENgth 42
LIB 154
Line length 34, 73, 78
Line numbers,
  in Command box 62
  in Command mode 36
Lines of text,
  to delete 9, 27, 31, 34, 39, 52, 57, 70
  to insert 9, 18, 28, 33, 34, 55, 70
  to replace 25, 29, 30, 34, 50, 56, 70
Lines per page 166, 170
Link 72, 139
Linker 137-163
  and DS opcode 139, 142
  and DSK opcode 147-149
  and ENT opcode 138, 142
  and ERR opcode 139, 143
  and EXT opcode 138, 141
  and ORG opcode 144, 152
  and REL opcode 138, 140-141
  and SAV opcode 153
  File Names 144-145
Linking process 146
List 8, 41, 64, 72
  and PRTR 43
  from last listed line 46, 69, 72
  from last specified line 46, 72
  to printer, formatted 41, 64, 72
  to screen, formatted 41, 64, 72
  without line numbers 43, 66, 72
  to abort 41
  to pause 41

Listing,
CYCLE times 103-104, 167, 172
DO OFF 114, 167, 172
Macros 104
limiting bytes printed in 167, 172
LKV 154
LNK 152
Load a source file 18
Local Labels, Global Labels & Variables 78-80
Local Variables,
defining 79-80
and PUT files 96
Locate a label or line 28, 34, 55
Lock file on disk 19
Logical operations, in operands 81
Lower case/upper case,
to change 24
in labels 88
LST ON/OFF 105
LSTDO or LSTDO OFF 105
LSTDO, configuring 167, 172
LUP 122

M

MAC 127
Macgen (utility) 200-202
Macros 127-136
defining 127, 128-132, 133-134
and PUT files 96
libraries, and USE opcode 97, 136
libraries, provided with Merlin 136
listings 104
nested 134
Pseudo Ops for, 127-129
variables 129-133
Main menu 5, 17, 55
Making Back-up Copies of Merlin 3
Make dump 211
Markers 28-29, 55, 56
Maximum,
length of comments 34, 63, 73
length of labels 77
Memory,
IN USE 146, 182
full 181
full error, in Sourceror 191
status 25, 33, 50, 70
use by Merlin 166
Merlin,
Memory Map 176-178
internal entry points 124
Misalignment error 182
Miscellaneous Pseudo Ops 119-126
Mistakes, fixing 25, 29, 34 (see also Control-R and ÓR)
MONitor 42, 65, 72
MOVE 42 (see Cut, Copy & Paste)
Moving,
the cursor 26, 33, 48, 49, 50
text 29, 42
Multiple input LNK files 160
Multiple output files 157
Multiplication operation, in operands 81
MX 84, 123

N

N (Normal - Sourceror) 187
Nesting error 182
Nested Macros 134
New 21, 43, 65, 72
Not macro error 182
Not resolved error 146
Number Format (Binary, Decimal, Hex) 80

O

O: Object Save Command 12, 21
command, and Linker 146, 153
O (ORG - Sourceror) 187
OBJ 92
Object code 13, 21
Online command 19, 20

Opcode and Pseudo Opcode Conventions 78
Operand and Comment Length Conventions 78
OR operation, in operands 81-82
ORG 6, 21, 89
  and the Linker 91, 152
Out of Memory 21, 182

P

PAG 106
Page Header, in listing 106
Parentheses,
  and Precedence in Expressions 82
PARMS file 4, 34, 55, 72, 170
Paste 30, 34, 56
Pathnames 77
PAU 123
Personalizing Merlin 4, 72, 166
PFX 19, 20
PMC or >>> 128
Precedence, in operand expressions 82
Prefix, ProDOS 13, 19, 20
Preliminary Definitions 75-76
Primitive expressions 81
Print command,
  and PRTR 43, 59, 66
  without line numbers 43, 66
Printer,
  slot#, in PRTR command 43, 66
  string, in PRTR command 43, 175
  to print Catalog 17
Printfiler 211-214
PRinTeR 43, 59, 66, 175 (see also TTL, pg 106; PAG, pg 106)
ProDOS technical information 168
PUT 93-96, 137
  linker command 152

Q

Q (Quit),
  to BASIC or program selector 22, 66
  to Main Menu 7, 44, 72
  Sourceror 187
Quick Link 159

R

R: Read Text File 21
R (Read - Sourceror) 188
Range Error 183
REL Files 91, 92, 140
  and the ERR Pseudo Op 143
  and the Linker 112
  and OBJ opcode 91
Relative expressions, and Linker 140
Renaming disk files 19
Replace 29, 56
Restoring lines in Edit mode 25
Return key 18, 33, 70
REV 108
Reversed string data 108
RTS return to Merlin 6
Running a program 14

S

S: Save source file 10, 13
SAV 90, 98, 153
".S",
  suffix in file names 10, 18, 96, 97
Select all text 29
Semicolon,
  for comments 6
  in macros 131
SKP 106
Slot 19, 20
Source code format 7, 77-78
Sourceror,
  Merlin 8 184-191
  Merlin 16 189
  XL version 189
Spaces in a text line 7
Special macro variables 129-133

Split screen editing 54
Status, memory 25, 33, 50, 70
STR 109
String Delimiters 37, 61
Suggested reading 2-3
SW 123
Sweet 16 85, 123
SYM 67, 72
Symbol table, description of 10-11, 165
Syntax error 183
Syntax, Source code 7, 75, 77
System requirements 1

T

T (Text - Sourceror) 188
TAB Key 18, 24, 26, 33, 50, 70
Tabs,
  to set Tabs 44, 67, 72, 167, 172
  to zero Tabs 21, 67, 72
  and word processing text files 44, 67
Technical Information 164-178
TEXT 44, 67, 172
Text files 17, 21
Text, select all 29, 57
TR ON/OFF/ADR 106
TRuncOFf 44, 68, 72
TRuncON 45, 68, 72
TTL 106 (see PRTR also)
Two Externals 183
Txted 73, 214
TYP 99, 153
Type.Changer (utility) 214-215

U

Ultraterm information 165
Undo (fixing mistakes) 29, 34, 56, 73
Unknown Label 183
Unlocking files 19
Unnew 68, 72
Upper case / lower case,
  to change 24, 37
  labels 88
USE, 97
  and Macro Libraries 97
USER 45, 68, 72
USR 97, 124-125
Using Sourceror 184-191
Utilities 197-215

V

VAL 45, 69, 72
Value of labels 45
VAR 98
Variables 79-80, 98, 115, 122, 131, 135
VER 159
Vertical Tabs 30, 33, 34, 56, 70, 71
VIDeo 45, 69

W

W: Write Text File, 21
W (Word - Sourceror) 189
W 0 command 46, 69
Word find 25, 28, 30, 39, 40, 50, 56
Where 45, 69, 72
Why Macros? 127
Why a Linker? 137
Wild Cards,
  in Delimited Strings 37
  character, changing the 167
Word processing text files 44, 67

X

XC 84, 123
XREF 205-207
XREFA 208

## Z

Zero Page Addresses used by Merlin for USR commands 124
Zero page addressing, forced 85

# ASCII Character Chart

| Hex | Dec | Binary | Hex | Dec | Binary | ASCII Character | | |
|---|---|---|---|---|---|---|---|---|
| $00 | 0 | 0000 0000 | $80 | 128 | 1000 0000 | ^@ | NUL | Null |
| $01 | 1 | 0000 0001 | $81 | 129 | 1000 0001 | ^A | SOH | Start of heading |
| $02 | 2 | 0000 0010 | $82 | 130 | 1000 0010 | ^B | STX | Start of text |
| $03 | 3 | 0000 0011 | $83 | 131 | 1000 0011 | ^C | ETX | End of text |
| $04 | 4 | 0000 0100 | $84 | 132 | 1000 0100 | ^D | EOT | End of transmission |
| $05 | 5 | 0000 0101 | $85 | 133 | 1000 0101 | ^E | ENQ | Enquiry |
| $06 | 6 | 0000 0110 | $86 | 134 | 1000 0110 | ^F | ACK | Acknowledge |
| $07 | 7 | 0000 0111 | $87 | 135 | 1000 0111 | ^G | BEL | Bell |
| $08 | 8 | 0000 1000 | $88 | 136 | 1000 1000 | ^H | BS | Backspace |
| $09 | 9 | 0000 1001 | $89 | 137 | 1000 1001 | ^I | HT | Horiz. TAB |
| $0A | 10 | 0000 1010 | $8A | 138 | 1000 1010 | ^J | LF | Linefeed |
| $0B | 11 | 0000 1011 | $8B | 139 | 1000 1011 | ^K | VT | Up Arrow |
| $0C | 12 | 0000 1100 | $8C | 140 | 1000 1100 | ^L | FF | Formfeed |
| $0D | 13 | 0000 1101 | $8D | 141 | 1000 1101 | ^M | CR | Return |
| $0E | 14 | 0000 1110 | $8E | 142 | 1000 1110 | ^N | SO | Shift out |
| $0F | 15 | 0000 1111 | $8F | 143 | 1000 1111 | ^O | SI | Shift in |
| $10 | 16 | 0001 0000 | $90 | 144 | 1001 0000 | ^P | DLE | Data link escape |
| $11 | 17 | 0001 0001 | $91 | 145 | 1001 0001 | ^Q | DC1 | XON |
| $12 | 18 | 0001 0010 | $92 | 146 | 1001 0010 | ^R | DC2 | AUXON |
| $13 | 19 | 0001 0011 | $93 | 147 | 1001 0011 | ^S | DC3 | XOFF |
| $14 | 20 | 0001 0100 | $94 | 148 | 1001 0100 | ^T | DC4 | AUXOFF |
| $15 | 21 | 0001 0101 | $95 | 149 | 1001 0101 | ^U | NAK | Negative Ack. (Right Arrow) |
| $16 | 22 | 0001 0110 | $96 | 150 | 1001 0110 | ^V | SYN | Synchronous file |
| $17 | 23 | 0001 0111 | $97 | 151 | 1001 0111 | ^W | ETB | End of transmission block |
| $18 | 24 | 0001 1000 | $98 | 152 | 1001 1000 | ^X | CAN | Cancel line |
| $19 | 25 | 0001 1001 | $99 | 153 | 1001 1001 | ^Y | EM | End of medium |
| $1A | 26 | 0001 1010 | $9A | 154 | 1001 1010 | ^Z | SUB | Substitute |
| $1B | 27 | 0001 1011 | $9B | 155 | 1001 1011 | ^[ | ESC | Escape |
| $1C | 28 | 0001 1100 | $9C | 156 | 1001 1100 | ^\ | FS | File or form separator |
| $1D | 29 | 0001 1101 | $9D | 157 | 1001 1101 | ^] | GS | Group separator |
| $1E | 30 | 0001 1110 | $9E | 158 | 1001 1110 | ^^ | RS | Record separator |
| $1F | 31 | 0001 1111 | $9F | 159 | 1001 1111 | ^_ | US | Unit separator |
| $20 | 32 | 0010 0000 | $A0 | 160 | 1010 0000 | space | | |
| $21 | 33 | 0010 0001 | $A1 | 161 | 1010 0001 | ! | | |
| $22 | 34 | 0010 0010 | $A2 | 162 | 1010 0010 | " | | |
| $23 | 35 | 0010 0011 | $A3 | 163 | 1010 0011 | # | | |
| $24 | 36 | 0010 0100 | $A4 | 164 | 1010 0100 | $ | | |
| $25 | 37 | 0010 0101 | $A5 | 165 | 1010 0101 | % | | |
| $26 | 38 | 0010 0110 | $A6 | 166 | 1010 0110 | & | | |
| $27 | 39 | 0010 0111 | $A7 | 167 | 1010 0111 | ' | | |
| $28 | 40 | 0010 1000 | $A8 | 168 | 1010 1000 | ( | | |
| $29 | 41 | 0010 1001 | $A9 | 169 | 1010 1001 | ) | | |
| $2A | 42 | 0010 1010 | $AA | 170 | 1010 1010 | * | | |
| $2B | 43 | 0010 1011 | $AB | 171 | 1010 1011 | + | | |
| $2C | 44 | 0010 1100 | $AC | 172 | 1010 1100 | , | | |
| $2D | 45 | 0010 1101 | $AD | 173 | 1010 1101 | - | | |
| $2E | 46 | 0010 1110 | $AE | 174 | 1010 1110 | . | | |
| $2F | 47 | 0010 1111 | $AF | 175 | 1010 1111 | / | | |
| $30 | 48 | 0011 0000 | $B0 | 176 | 1011 0000 | 0 | | |
| $31 | 49 | 0011 0001 | $B1 | 177 | 1011 0001 | 1 | | |
| $32 | 50 | 0011 0010 | $B2 | 178 | 1011 0010 | 2 | | |
| $33 | 51 | 0011 0011 | $B3 | 179 | 1011 0011 | 3 | | |
| $34 | 52 | 0011 0100 | $B4 | 180 | 1011 0100 | 4 | | |
| $35 | 53 | 0011 0101 | $B5 | 181 | 1011 0101 | 5 | | |
| $36 | 54 | 0011 0110 | $B6 | 182 | 1011 0110 | 6 | | |
| $37 | 55 | 0011 0111 | $B7 | 183 | 1011 0111 | 7 | | |
| $38 | 56 | 0011 1000 | $B8 | 184 | 1011 1000 | 8 | | |
| $39 | 57 | 0011 1001 | $B9 | 185 | 1011 1001 | 9 | | |
| $3A | 58 | 0011 1010 | $BA | 186 | 1011 1010 | : | | |
| $3B | 59 | 0011 1011 | $BB | 187 | 1011 1011 | ; | | |
| $3C | 60 | 0011 1100 | $BC | 188 | 1011 1100 | < | | |
| $3D | 61 | 0011 1101 | $BD | 189 | 1011 1101 | = | | |
| $3E | 62 | 0011 1110 | $BE | 190 | 1011 1110 | > | | |
| $3F | 63 | 0011 1111 | $BF | 191 | 1011 1111 | ? | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| $40 | 64 | 0100 0000 | $C0 | 192 | 1100 0000 | @ |
| $41 | 65 | 0100 0001 | $C1 | 193 | 1100 0001 | A |
| $42 | 66 | 0100 0010 | $C2 | 194 | 1100 0010 | B |
| $43 | 67 | 0100 0011 | $C3 | 195 | 1100 0011 | C |
| $44 | 68 | 0100 0100 | $C4 | 196 | 1100 0100 | D |
| $45 | 69 | 0100 0101 | $C5 | 197 | 1100 0101 | E |
| $46 | 70 | 0100 0110 | $C6 | 198 | 1100 0110 | F |
| $47 | 71 | 0100 0111 | $C7 | 199 | 1100 0111 | G |
| $48 | 72 | 0100 1000 | $C8 | 200 | 1100 1000 | H |
| $49 | 73 | 0100 1001 | $C9 | 201 | 1100 1001 | I |
| $4A | 74 | 0100 1010 | $CA | 202 | 1100 1010 | J |
| $4B | 75 | 0100 1011 | $CB | 203 | 1100 1011 | K |
| $4C | 76 | 0100 1100 | $CC | 204 | 1100 1100 | L |
| $4D | 77 | 0100 1101 | $CD | 205 | 1100 1101 | M |
| $4E | 78 | 0100 1110 | $CE | 206 | 1100 1110 | N |
| $4F | 79 | 0100 1111 | $CF | 207 | 1100 1111 | O |
| $50 | 80 | 0101 0000 | $D0 | 208 | 1101 0000 | P |
| $51 | 81 | 0101 0001 | $D1 | 209 | 1101 0001 | Q |
| $52 | 82 | 0101 0010 | $D2 | 210 | 1101 0010 | R |
| $53 | 83 | 0101 0011 | $D3 | 211 | 1101 0011 | S |
| $54 | 84 | 0101 0100 | $D4 | 212 | 1101 0100 | T |
| $55 | 85 | 0101 0101 | $D5 | 213 | 1101 0101 | U |
| $56 | 86 | 0101 0110 | $D6 | 214 | 1101 0110 | V |
| $57 | 87 | 0101 0111 | $D7 | 215 | 1101 0111 | W |
| $58 | 88 | 0101 1000 | $D8 | 216 | 1101 1000 | X |
| $59 | 89 | 0101 1001 | $D9 | 217 | 1101 1001 | Y |
| $5A | 90 | 0101 1010 | $DA | 218 | 1101 1010 | Z |
| $5B | 91 | 0101 1011 | $DB | 219 | 1101 1011 | [ |
| $5C | 92 | 0101 1100 | $DC | 220 | 1101 1100 | \ |
| $5D | 93 | 0101 1101 | $DD | 221 | 1101 1101 | ] |
| $5E | 94 | 0101 1110 | $DE | 222 | 1101 1110 | ^ |
| $5F | 95 | 0101 1111 | $DF | 223 | 1101 1111 | _ |
| $60 | 96 | 0110 0000 | $E0 | 224 | 1110 0000 | ` |
| $61 | 97 | 0110 0001 | $E1 | 225 | 1110 0001 | a |
| $62 | 98 | 0110 0010 | $E2 | 226 | 1110 0010 | b |
| $63 | 99 | 0110 0011 | $E3 | 227 | 1110 0011 | c |
| $64 | 100 | 0110 0100 | $E4 | 228 | 1110 0100 | d |
| $65 | 101 | 0110 0101 | $E5 | 229 | 1110 0101 | e |
| $66 | 102 | 0110 0110 | $E6 | 230 | 1110 0110 | f |
| $67 | 103 | 0110 0111 | $E7 | 231 | 1110 0111 | g |
| $68 | 104 | 0110 1000 | $E8 | 232 | 1110 1000 | h |
| $69 | 105 | 0110 1001 | $E9 | 233 | 1110 1001 | i |
| $6A | 106 | 0110 1010 | $EA | 234 | 1110 1010 | j |
| $6B | 107 | 0110 1011 | $EB | 235 | 1110 1011 | k |
| $6C | 108 | 0110 1100 | $EC | 236 | 1110 1100 | l |
| $6D | 109 | 0110 1101 | $ED | 237 | 1110 1101 | m |
| $6E | 110 | 0110 1110 | $EE | 238 | 1110 1110 | n |
| $6F | 111 | 0110 1111 | $EF | 239 | 1110 1111 | o |
| $70 | 112 | 0111 0000 | $F0 | 240 | 1111 0000 | p |
| $71 | 113 | 0111 0001 | $F1 | 241 | 1111 0001 | q |
| $72 | 114 | 0111 0010 | $F2 | 242 | 1111 0010 | r |
| $73 | 115 | 0111 0011 | $F3 | 243 | 1111 0011 | s |
| $74 | 116 | 0111 0100 | $F4 | 244 | 1111 0100 | t |
| $75 | 117 | 0111 0101 | $F5 | 245 | 1111 0101 | u |
| $76 | 118 | 0111 0110 | $F6 | 246 | 1111 0110 | v |
| $77 | 119 | 0111 0111 | $F7 | 247 | 1111 0111 | w |
| $78 | 120 | 0111 1000 | $F8 | 248 | 1111 1000 | x |
| $79 | 121 | 0111 1001 | $F9 | 249 | 1111 1001 | y |
| $7A | 122 | 0111 1010 | $FA | 250 | 1111 1010 | z |
| $7B | 123 | 0111 1011 | $FB | 251 | 1111 1011 | { |
| $7C | 124 | 0111 1100 | $FC | 252 | 1111 1100 | \| |
| $7D | 125 | 0111 1101 | $FD | 253 | 1111 1101 | } |
| $7E | 126 | 0111 1110 | $FE | 254 | 1111 1110 | ~ |
| $7F | 127 | 0111 1111 | $FF | 255 | 1111 1111 | Delete/rubout |