Merlin 16+™ Supplementary Manual

GS/OS (ProDOS 16) Macro Assembler

by Glen Bredon

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
1M289

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., with all rights reserved. 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 no more than two computers. You may not sell or transfer copies of, or any part of, this product, nor rent or lease to others 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 HodgePodge are copyrighted programs of Apple Computer, Inc. licensed to Roger Wagner Publishing, Inc. to distribute for use only in combination with the Merlin assembler

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.

OUR GUARANTEE

This product carries the unconditional guarantee of satisfaction or your money back. Any product may be returned in resellable condition to the 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.

Table of Contents

Introduction

The Merlin Series of Assemblers

Overview

1 Memory Requirements
1 Xref, Sourceror, Linkers, Macgen
2 Support of ProDOS 16 "Message Center"
2 Hard Disk Installation

Main Menu

4 Main Menu Changes
4 Disk Commands from the Main Menu
6 Prefixes
8 APW Source Files

Full Screen Editor

10 Full Screen Editor
12 Command Box Commands

Assembler Changes

18 Assembler Changes

Changes to the Linkers

24 Changes to the Linkers
25 New Commands in the Linker Command Files
27 OMF Linker

EXE Files (and USR opcodes)

28 EXE files and USR opcodes
30 Apple protocol for EXE files

Files on the Merlin 16+ Disk

34 The /Merlin disk - main directory
34 Source subdirectory
35 Object subdirectory
35 Macro Library subdirectory
46 Subroutine Library subdirectory
46 Commands subdirectory
48 Help files subdirectory
49 Tool Equates subdirectory

Files on the Merlin Samples Disk

50 Merlin Samples - Main Directory
50 Command Source subdirectory
50 Subroutine Source subdirectory
51 Miscellaneous subdirectory
52 Generic Source subdirectory
53 HodgePodge subdirectory
53 Nifty List subdirectory
53 HP.Supermacros subdirectory
54 Converter subdirectory
54 Internals CDA subdirectory
55 New Additions to the Merlin 16+ and Merlin Sample Disks

APW vs. Merlin

56 Converting source files

Introduction

The Merlin series of assemblers has been, and will continue to be, an evolutionary process. The original Merlin assembler was based on one of the very first assemblers for the Apple II computers, back when 48K was considered a full-memory machine.

As the Apple II has grown, so has Merlin. This latest version of the classic Apple II assembler adds an enormous number of new features. In fact, Merlin has now reached the point where it is unlikely you will ever need each and every one of the features present in the system. However, they are all provided so that no matter what your next programming challenge is, Merlin will stand ready to make your job as easy as possible.

There are now four versions of Merlin in the Merlin 8/16 package: Merlin 8 for DOS 3.3 and ProDOS 8, Merlin 16 running under ProDOS 8, and now Merlin 16+, which runs under ProDOS 16, or GS/OS on the Apple IIGS.

Merlin 8 is provided for those programmer who work on a standard Apple IIe or IIc. It is quite sufficient for many programming tasks.

Merlin 16 runs under ProDOS 8, and also requires either a 65802 or 65816 microprocessor. Most often this will be run on an Apple IIGS, but this version has been kept in the Merlin 8/16 package because of its impressive performance on an Apple IIe or IIc when equipped with a 65802 microprocessor. Replacing the 65C02 with the 65802 is an easy and inexpensive operation, and will give the serious programmer twice the assembly speed, a command-based linker five times faster than the Merlin 8 counterpart, plus the power of a more advanced editor with split-screen operation, faster operation, and Apple IIGS-compatible code generation. In addition, Merlin 16 is SoftSwitch-compatible, which means you can use the Merlin 16 assembler, and have other utilities, or even test the program you're writing, and keep up to nine separate programs, including Merlin, all in memory at the same time!

For the Apple IIGS owner, however, Merlin 16+ is the ultimate form of the Merlin assembler. Taking full advantage of the increased memory on the Apple IIGS, Merlin 16+ is twice again as fast as Merlin 16, has an even more powerful linker, and adds a substantial list of new features and functions to the Merlin environment. Built-in file utilities including intelligent file copying, disk formatting, file and directory compares and more.

Many of the extra utilities in Merlin such as Macgen and Sourceror are now memory-resident. New functions include the ability to import and export APW-compatible files for use with other language compilers, and general compatibility with APW users. The editor is even faster than ever, with mouse control and a 4000-step undo function. This means you can edit literally hundreds of lines of code, then change your mind, and "run the clock backwards" until you are back where you want to be!

The following documentation assumes that you are already familiar with Merlin 16. If you are a new user of the Merlin system, we suggest you first use Merlin 16, and follow the tutorial in the main manual. Once you are comfortable with the basic Merlin system, the following pages will explain what has been added.

In general, every version of Merlin is a super-set of all previous versions. That is to say, even with Merlin 16+, all the familiar commands and assembler directives are still available, and you can still create ProDOS 8 (and even DOS 3.3 for that matter) applications in the Merlin 16+ environment. Remember, Merlin is simply the tool to create the code you want. The disk operating system it runs under imposes no restriction on your final output programs. The point here is that if you can run Merlin 16+, we heartily recommend you do so, regardless of the programs you are writing, with our desire being that you work with the most efficient and comfortable tools we can provide.

Overview of the Operation of Merlin 16+

Memory requirements

Merlin 16+ requires an Apple IIGS with at least one megabyte of expansion RAM. More than this is recommended if you wish to use RAM disk space. Merlin 16+ itself, together with the co-resident utility program, occupies in excess of 72k of memory just for code and uses an additional 576k of buffer space.

When Merlin 16+ starts up, it does a quick test of each memory bank allocated by the Memory Manager and gets another bank if the allocated bank tests out bad. Such bad banks will not be released to the system until Merlin is exited.

Former versions of Merlin used a memory-efficient, but time consuming, way of keeping its symbol table, and the number of labels that could be accomodated depended on their sizes. Because of the vastly increased memory on the GS, Merlin 16+ uses an advanced "hashing" scheme for its symbol table, which is memory wasteful but very fast. It is this scheme that is primarily responsible for the 3-fold speed increase of Merlin 16+ over Merlin-16. (Based on RAMdisk timings.) In this scheme, the size of the symbols has little importance; there is space for 4096 symbols of length less than 12 and for 2048 symbols of length 12 or over. This does not include local labels which are handled separately; indeed the "label count" shown at the end of assembly is of global labels only.

It is doubtful that one would run out of symbol table space, but if that does happen, a cure is to convert many symbols to local form, or to use the linker to break up the source into smaller pieces. (The amount of speed increase depends exponentially on the size of the source, so small sources will not see a 3-fold advantage, but very large ones could well see even more.) Merlin 16+ speed on typical source files from /RAM5 has been timed at around 40,000 lines per minute.

Xref, Sourceror, Linkers, Macgen

The XREF programs, the Linkers, Sourceror, and Macgen are all resident, and are accessed directly from the editor Command Box.

In addition, there is a new sub-menu, accessed from the Main Menu, called *Utilities*. This is a set of disk volume and file utilities. These functions are documented in detail later in this manual.

Support of ProDOS 16 "Message Center"

If the program selector you are using passes a pathname using the Message Center tool, then Merlin will use that name to automatically load a source file when it starts up. This would be the case, for example, when starting up Merlin 16+ by double-clicking on a Merlin source file document in the Finder. Note that for this feature to work, the Icons folder on your System disk must have the icon defined for a Merlin source file. There is a file included on the Merlin 16+ disk, called Merlin.Icons that can be move to the Icons folder on your system disk. Since the application pathname is imbedded in the icon file, Merlin.Sys16 must be located in a folder named Merlin on your startup disk. The only way to change the application pathname for a source file that launches Merlin.Sy16 is with an icon editor utility. Double-clicking on Merlin.Sys16 itself should work to launch Merlin 16+ no matter where it is located.

Hard Disk Installation

Installing Merlin 16+ on a hard disk is actually very easy - in fact, even easier than Merlin 16. This is because earlier versions of Merlin had a number of support files like the Linkers, etc. that were separate files from Merlin.System itself. With the new Merlin 16+, all the Linkers, utilities, Sourceror, Macgen, etc. are all within the same file, Merlin.Sys16.

The exact files required to install Merlin 16+ on a hard disk depend on how many of the other files, such as the macro libraries, etc. you wish to have available. Here are the options:

1) Minimum Merlin system:

Filename	**Function/Description**
Merlin.Sys16	Main file for Merlin itself.
PARMS	User-defined default values for system operation.
LABELS	Label file for Sourceror

That's it! These are the only two files you need on a disk to use Merlin itself. This will give you Merlin, the Linkers, Macgen, Sourceror, and the Utilities.

2) Source files:

PARMS.S	This is the source file for the PARMS file.

Add this file to your disk if you want to be able to change your default settings.

3) Shell Commands:

COMMANDS (directory) This directory contains all the EXE files that are used as a shell command from Merlin. The command file HELP must be in this directory if you want the on-line help function. Prefix 6 in the PARMS files must be set to this directory.

4) Help Files:

HELP.FILES (directory) This directory holds all the TXT files that will be available as on-line help while in Merlin. Prefix 7 in the PARMS files must be set to this directory. Within this directory is another directory, TOOLDOC, that contains all the TXT files for the TOOLHELP command.

5) Tool.Macros:

MACRO.LIBRARY (directory) This directory contains all the macro definition files needed for Macgen and the Apple IIGS Tool calls. Prefix 4 in the PARMS files must be set to this directory.

6) Other files:

See the section "Merlin 16+ Files" for more detailed information on all the files on the Merlin 16+ disk. You may install all the files from the Merlin 16+ and Merlin.Samples disks, or just those you want, at your descretion.

Main Menu Changes

IMPORTANT: There has been a change in the Main Menu input of file names for loading, saving, etc. The cursor no longer is placed at the end of input with a question mark asking whether to accept the default; and, more important, pressing Return at this point no longer cancels, but rather, ***accepts*** the default. This change was made for more consistency with other parts of the program, particularly the Utilities section. Users of former versions of Merlin may have difficulties with this at first.

There is, however, a bit in the PARMS file which will cause Return on the first character to be an abort as was done in the older Merlin 16. For more on the input routine, both at the Main Menu and elsewhere, see the section on "Line Input" in the upcoming text.

Merlin 16+ does not support printing the disk catalog by entering the number "1" as the pathname, as is done in Merlin 8 and 16.

RESET: In previous versions of Merlin, RESET was trapped, and returned you to the Main Menu. This continues to be the case with Merlin 16+, with one caveat: If you are running Merlin 16+ under GS/OS, RESET may leave the system in an unpredictable state. Therefore, if you do need to press RESET to regain control of the computer, you should immediately save your file, if necessary, to disk (preferrably an expendable one), and then re-boot the system. The RESET-trapping function can be turned off in the PARMS file. We believe it is better to give you at least a fighting chance to recover your data, rather than arbitrarily shut down the system.

Disk commands from the Main Menu

Line input: Input lines (such as giving a filename for a Load in the Main Menu or a directory name in the Utilities, or some command in the editor Command Box) have available some line editing features that are more or less standardized across the program. The DEL key acts as a delete key should and deletes the character to the left of the cursor. The Control-D key deletes the character under the cursor. The left and right arrows move the cursor. Escape cancels whatever the input is being done for. Return accepts the line as it stands (or truncates it at the cursor depending on the setting of one of the PARMS bits).

If the cursor is on the *first* character when Return is pressed, then the line is accepted as it stands, but one of the PARMS bits can change this to an abort as with the old input routine. If the Apple key is down while a key is pressed then that key is *inserted.* Also the Control-E key toggles the insert mode for those who prefer things to be as in AppleWorks. A cursor move, or any control character, turns insert mode off. The TAB key moves the cursor just beyond the next "/" or to the end of line. (In Utilities, TAB also places a "/" at the end of the line and it does not matter whether a "/" is left at the end of a line or not.)

The Apple-key, together with TAB, moves the cursor to the point just following the *previous* "/", or to the beginning of the line. Control-Y truncates the line at the cursor, for those opting to have the whole line accepted on Return.

BLOAD/BSAVE are no longer supported.

BRUN and - are the same and will execute SYS, S16, and EXE files. The - command will return to a selector, or the GS built-in dialog box, when the program quits. Note that this applies only to SYS, and S16 files. EXE files can be executed by either of these commands, but are treated specially in that Merlin remains active and passes certain information. A NEW is done before executing a BRUN, releasing the memory for the current source file, and clearing the source file name. This does not apply to EXE files where the source buffer is *not* released. (See the section on EXE files, and USR files in particular.)

= (Launch program): The new command = (as in =MYPROGRAM) is also a run command, but it differs in that it leaves Merlin's address on the system stack so that a quit from the program that is run will return to Merlin. When running a program with the = command from the disk command prompt, the prefixes are reset on return to the default values in the PARMS file.

SAVESYM: The new command SAVESYM <filename> will save the symbol table to a disk file. Local labels will be ignored. The purpose of this facility is for the possible use of symbolic debuggers. It should be used only just following an assembly, or it may be corrupted by parts of Merlin that use the same memory area for other purposes. The format of the resulting file will be the same as in earlier versions of Merlin 16.

FILETYPE: The new command, FILETYPE, is for changing the type of a file. The syntax is:

```
FILETYPE pathname,TYP
```

where TYP can be hex (starting with "$") or ASCII. For example, the command

```
FILETYPE MYFILE,EXE  or  FILETYPE MYFILE,$B5
```

will change the type of MYFILE to EXE.

AUXTYPE: Another new command, AUXTYPE, is for changing the auxtype of a file. The syntax is:

```
AUXTYPE pathname,$FFFF
```

where $FFFF can be any hex (starting with "$") value.

Shell Commands: Any unrecognized command (such as HELP) will be regarded as a "shell command" which means that an executable file (usually of filetype EXE) of that name will be looked for in the Prefix 6 directory (see next section), and executed if found. For example, type HELP from the disk command prompt or from the Command Box. You might also want to try TYPE <filename> or DUMP <filename> where <filename> is a file in the current prefix directory.

Ambitious Merlin users can create their own extended commands. The source files for the Merlin HELP, TOOLHELP, and TYPE commands, and other examples, are included on the Merlin Samples disk.

Prefixes

Merlin 16+ supports a total of *eight* individual, numbered ProDOS prefixes. All eight prefixes can be set up with default values using Merlin's PARMS file. (Note that as with other versions of Merlin, Merlin 16+ requires its own PARMS file. You cannot use the PARMS file from another version of Merlin.)

Prefix #0 is the default prefix at the Main Menu when Merlin is started up. If the #0 prefix is set to a null in the PARMS file, then Prefix #0 is set to the directory containing MERLIN.SYS16 (which is also Prefix #1). If Prefix #0 in the PARMS file is not null, then that prefix is used. This allows you to startup Merlin 16+ from one directory, but to have the default prefix (determined by the PARMs file) come up set to another directory, presumably containing your own source files.

When using the "disk command" PREFIX (or PFX), you can use the following:

PFX 3 sets prefix 0 to the current prefix 3 (null if prefix 3 was not set).
PFX 3=/name sets prefix 3 to "/name".
PFX 3stuff or PFX 3/stuff sets prefix 0 to 3/stuff if stuff does not start with "=".
PFX /stuff sets prefix 0 to /stuff.
PFX prints the list of prefixes.

In former versions the command PREFIX or PFX without a file name cut the prefix down to the volume name. Since this command does not seem to have been used much, the use of PREFIX (or PFX) alone has been changed to now print the list of the current 8 prefixes.

All the prefixes 0 through 7 are in the PARMS file and will be set on boot (*after* PARMS is loaded) accordingly. These prefixes can also be used as part of a pathname in the DSK, USE, etc. commands within a source file. For example, if you wanted to assemble a source file and have the object file saved in a location determined by your PARMS file, you could use the syntax:

```
DSK 1/FILE.OBJ
```

In the interest of convenience for common Merlin operations, the PARMS prefixes are used as the default location for pathnames. For example, in the Merlin 16 version of Macgen, you would normally have to type in the complete pathname for the location of the directory to scan for macro files. In Merlin 16+, the default is to use prefix #4 as the default pathname for your macro library.

Unless you have a strong desire to reserve the prefix number for another use, you may want to leave the prefixes set to the locations currently set in the PARMS file. Some of the default pathnames are used by Merlin 16+ as the default location to access certain files, others can be set at your convenience. The default pathames and uses are as follows:

Prefix 0 is usually the directory containing MERLIN.SYS16. Change this entry in the PARMS file if you want the default directory to be a particular location after starting up Merlin 16+. Prefix 0 is not used by Merlin for any defaults, so you can change this as you wish.

Prefix 1 is the directory containing MERLIN.SYS16. Regardless of the setting of Prefix 1, the PARMS file must always be in the same directory as MERLIN.SYS16. Currently Prefix 1 is used as a base for the other 6 Prefixes.

Prefix 2 is set to the SOURCE directory of the Merlin 16+ disk. Typing PFX 2 sets the prefix to the correct directory to load the PARMS.S file.

Prefix 3 is set to the OBJ.CODE directory on the Merlin Samples. This is the location of some example object files, but can be set to anything you like.

Prefix 4 is set to the MACRO.LIBRARY directory on the Merlin 16+ disk. This directory contains the macro definition files from both the GEN.MACROS and TOOLS.MACROS directories from the Merlin 16 disk. Prefix 4 is used as the default pathname for the directory to scan in the Macgen utility.

Prefix 5 is set to the SUBROUT.LIB directory on the Merlin 16+ disk. This directory contains some examples of useful subroutines that are in LNK file format, and can be linked into your own object files as subroutine resources. See the source files in the SOURCE directory for comments on the various files.

Prefix 6 set to the COMMANDS directory. This is where Merlin 16+ will look for the EXE files to execute its shell commands. This default prefix should only be changed if you have moved the shell commands to another directory.

Prefix 7 is set to the location of the Merlin HELP files, which are accessed by the shell command HELP in the Prefix 6 directory (usually COMMANDS). This also should not be changed unless you have put the help commands in a different directory.

Device Numbers vs. Slots:

In Merlin 16, and in particular ProDOS 8, Slot and Drive values were used to identify the device (via the SLOT command) that the next PFX= command would set the current prefix to.

In Merlin 16+, PFX= still sets prefix 0 to the volume name of the default device. The default device is the active device when Merlin 16+ was run. Under ProDOS 16, however, slot and drive numbers are no longer used. Instead, each disk device has a *device number*, which identifies it. In addition, devices have *device names*, which can be an easier way to identify a disk drive, especially if you have a number of different devices connected to your computer.

The device used by the PFX= (or PREFIX=) command can be changed by the disk command DEV# (where # = 1,2,3...), which takes the place of the SLOT command in previous versions.

Online:

Typing ONLINE as a disk command will show the various device numbers, and the device names, that are online. Similarly, the catalog function, given an equal sign ("=") as the "pathname", will set the prefix to the default device (volume) and then catalog it.

Typing PFX (or PREFIX) alone shows a list of the current settings for the eight prefixes.

APW Source Files

Export:

There is a "secret" Main Menu mode command "^" which is the Export command. This will write the current source file out as a type $B0 (SRC) file, stripping all high bits. This is for exporting a source file to APW. Nothing will be appended to the filename you select for the resulting file.

Import:

The other direction is trivial: the Main Menu Load command will load APW SRC type files (add a / to the end of the filename if it does not end in ".S"). After loading, you will probably want to do a FIXS command from the Command Box to get rid of all the excess spaces, and you may want to edit some of the semicolons at the start of comment lines to make them into asterisks for the correct tabbing.

The Converter program has also been upgraded, and is the recommended way to convert an APW-compatible file of any size to the Merlin 16 format.

Full Screen Editor

Mouse Control in Editor: The mouse is supported in the Merlin 16+ full screen editor, although it can be defeated as documented in the PARMS file. Moving the mouse moves the cursor accordingly. You can scroll the screen by moving the mouse to the top or bottom of the screen.

Both a single and double mouse click can be set in the PARMS file to be the equivalent of a given keyboard command. As the Merlin editor is normally set up, a single mouse click brings up a "Mouse control box", containing 16 commands that can be selected with the mouse. Here are the 16 commands, most of which are identical to some Apple-key command:

Cancel	Cancels selections or just exits the box.
Command Box	Opens Command Box.
Quit	Quits the Editor back to the Main Menu.
Assemble	Assembles the current source file.
Link	Does a "Quick Link" - equivalent to Apple-6.
Save	Brings up the "Quick Save" box - equivalent to Apple-S.
History	Shows current undo history - equiv. to two Ctrl-S's .
Goto	Brings up box for Goto label or line # - equiv. to Apple-L.
Select	Starts selection operation. If in select mode, that is canceled and a new one started. (Use to limit Find range, etc.)
Find	Equivalent to Apple-F. Subsequent mouse clicks will do the NEXT find until Find mode is cancelled by not finding or by Ctrl-X or Clear.
Find Word	Equivalent to Apple-W. Subsequent operation as for Find.
Exchange	Equivalent to Apple-E. Subsequent operation as for Find (a click is a "do it").
Paste	Paste text on clipboard - equivalent to Apple-V.
Cut	Cuts selected text. If no text is selected, first use starts select mode.
Copy	Copies selected text. If no text is selected, first use starts select mode.
Undo Mode	No key equivalent, but it sets an *Undo Mode* (indicated by the vertical end-of-line marker becoming a backslash (\). In this mode, mouse clicks become undo (OA-U) commands. The mode is automatically canceled when the undo buffer is empty, and it can be manually canceled by the Ctrl-X key or the CLEAR key.

A double-click with the mouse scrolls the listing one page forward.

The mouse button can be set up to issue any commands you please on a click and double-click by the appropriate setting in the PARMS file.

Note: The Mouse Control Box can also be activated by pressing Apple-?, at which point you can use the cursor keys and Return to make a selection.

Modified Command Keys: The editor key tables in the PARMS file now support changes to the Apple-command keys. That is, an Apple-command such as Apple-Z can now be changed to a non-Apple-key, usch as Control-Z. If a key is given in the PARMS file table with high bit off, then that key is an "Apple-key". Please note, however, that the cursor keys and Apple-cursor keys must not be changed.

INSERT MODE: In the editor, the default insert toggle key is now Control-E. The TAB key tabs the cursor to the next field. Pressing Apple-TAB does a backwards tab to the previous tab position.

TXTED: Apple-\ toggles the "TXTED" mode. TXTED mode is indicated by a short vertical bar marking "end of operand" on the first line of the screen, instead of the usual long vertical bar. In TXTED mode pressing Return will split a line, and a DELETE at the first character of a line will combine the line with the previous one.

Quick Save: A new command, Apple-S, brings up a box asking for the filename with which to save the current file. (Press Return on 1st character to accept default, ESC anywhere to abort.).

Backup Bit: There is a new PARMS bit to make the screen editor Apple-S command do a backup/save if set. In this case, if the filename is FILE.S then the file FILE.B, if any, is deleted, the current FILE.S is renamed FILE.B and the buffer is saved as FILE.S. Please note that this is for the Apple-S command *only* and is *not* done with the Main Menu Save command.

Keypad 1-9: The *keypad* keys 1-9 together with the Apple key move you to a spot in the source file proportional to the number key. There is a PARMS bit which will swap the keypad number key functions with those of the standard number keys.

Case Sensitive Searches: Searches in the editor are now case-sensitive only if PARMS indicates case sensitivity for labels in the assembler.

Return to Edit: The Editor now re-enters a source file with the cursor on the same line as it was when the editor was exited. This is true even when the file is saved, and re-loaded later. (There is a PARMS bit which disables this and always enters a source file with the cursor on the first line.)

4000-step Undo: The editor maintains a "history" of operations that remembers back over 4000 steps. The Undo command (Apple-U) will take the source file back along this path and can be issued as far as the history can take it. Of course there are some operations, such as cutting large chunks of the source out, that will limit the undo to less than the 4000 steps. When this happens, or the history is depleted, you get an "Undo buffer empty" message.

You can also see what the history looks like by pressing Control-S to get the Status box (which now shows the number of steps in the history) and then pressing Control-S again. This brings

up a large box with descriptions of the 12 most recent actions taken in the editor. You can use the up- and down-arrows to look at the remainder of the history. The history is forgotten if you leave the editor (e.g., if you do an assembly or issue a shell command like HELP).

There is a bit in the PARMS file which disables nondestructive things like cursor moves from being entered in the history and thus extending the necessary history much further. Even with the cursor moves, however, 4000 steps is quite satisfactory, and you might look at both options to see which you prefer. (Disabling is the present default.)

Shift-arrow commands: In the screen editor, the shift-arrow key combinations have special uses: Shift-right-arrow picks up the text under the cursor and saves it. Later, if shift-up-arrow is pressed, the picked up text is deposited at the position of the cursor. Shift-left-arrow is the same as shift-right arrow except that the text is deleted as it is being picked up (like a cut vs. copy). Shift-down-arrow empties the the buffer for this function. Shift-right-arrow (or left) will concatenate the text previously picked up with the current text if no shift-up-arrow or shift-down-arrow was used between.

Line Length Marker: In the screen editor, the vertical bar indicating end of line for assembly is now always shown on the screen, so that TXTED mode is visible. For very large line lengths for which the bar would ordinarily be off the screen, it is now printed at the last character of line 1 to the right of the line number.

The maximum line length in Merlin 16+ is 255 characters, and horizontal scrolling is supported in the Editor.

Keyboard macros:

The "example" keyboard macros program (with source file) now has three ways of learning temporary macros (permanent ones can just be put in the source and re-assembled). One way is a "learn by doing", another is via an editor box, and the third is a screen pick. Other enhancements are limited only by the user's imagination. See the source listing for more detailed instructions on how to use this useful addition to Merlin 16+.

Command Box Commands

GET: The GET command for moving the object code to main memory is no longer available in Merlin 16+.

A command terminated by the Enter key instead of Return will cause control to pass back to the command box so that another command can be issued without having to press Apple-O again.

Former USER commands (such as the XREF and SOURCEROR utilities) are memory resident and have their own commands. USER will be implemented for user-supplied routines only (except for the example source).

The Command Box (and the command line editor if enabled) now accepts a space as a delimiter, so you can use commands like LINK MYFILE.

All the "Disk commands" available from the Main Menu can also be given from the Command Box in the full screen editor (or from the command line editor if that is enabled). Anything not recognized as an editor or disk command is treated as a "shell command", which means that an executable file of the same name as the command is looked for in the Prefix 6 directory, and executed if found. If not found you will get an error message.

VAL: Because of the change in the way the symbol table is stored, the VAL command now only works with labels if the last assembly was done with listing off and no SYM command was issued. However, the value of a label is easily obtained using the SYM command with the syntax SYM LABEL. (The reason is that, at the end of assembly, the symbol table is modified, if listing is on, in order to make the symbol table listing more efficient. But, with the symbol table in another format, the assembler can no longer evaluate a label.)

SWAP and the Source buffers: Merlin 16+ maintains four separate text buffers. The main buffer is the source buffer, others are the PUT file buffer, the USE file buffer, and the clipboard. There are "Command Box" commands which will swap the source buffer with any of the other three. These commands are:

```
SWAPput   SWAPUse   and   SWAPClip
```

(lower case characters are optional). This makes it very easy to correct assembly errors in, for example, PUT files. When assembly gives such an error, abort it, note the PUT file line number, type SWAP from the command box, fix the error, go to the Main Menu to save the corrected file, return to the full screen editor, type SWAP from the command box again and reassemble; there is no need to reload the main source file. The name of the PUT file will be the default name for saving it, and will change back to the main source filename when the re-swap is done.

The XREF programs, the Linkers, Sourceror, and Macgen are all resident, and are used via the Command Box. In addition, there is a new Command Box command, "Convert".

Linkers:

To invoke the linkers, just type LINK filename. This goes into the GS linker by default. When using a linker command file, the LINKER command LKV $02 will now *switch* to LINKER.XL, and LKV $00 will enable the Absolute Linker, instead of issuing an error message.

Macgen:

Macgen is invoked by the MACgen command (eg. MAC *filename*). Note that in Merlin 16+, there is now a default pathname offered using Prefix #4. If you enter the name of a text file, rather than a directory, then Macgen will only scan that single text file for macros.

Xref:

The cross-referencers are now selectable by directives in the assembler. The directives are XREF and XREFA, the latter giving an address cross-reference. These both allow operands (optional) of 0 or 1, with 0 being the default. An operand of 1 causes DO OFF areas to be cross-referenced.

In contrast with the older versions, there is only the alphabetic list generated, and the cross-referencing is done even for areas with listing off. (Listing must be on, however, at the end of the assembly or the cross reference list will not be printed.) Also in contrast to the earlier versions, the cross-referencing has no effect on object code production and does not constrain memory.

Another way is provided to invoke the cross-reference. This is by the Apple-2 and Apple-4 commands in the full screen editor. These either do the same as Apple-1 or Apple-3, or they combine those commands with XREF or XREFA; which of these is done is a PARMS selection (see the label called RTNMASK). [Note that Merlin-16 used these keys in a similar manner, but had them issue a USER, presumably to one of the XREF programs, so this is very close to the present action.]

Note that a XREF or XREFA pseudo-op in the source file will override these keys.

Sourceror:

Sourceror (invoked by the Command Box command DIS <filename>) now has commands X and C which, respectively, restrict recognition of opcodes to the 6502 codes and the 65C02 codes. Unrecognized opcodes will produce a ??? on the screen in the opcode column and will sound a bell. In the built source file they will produce HEX opcodes. These commands should be issued only as the first command to Sourceror, although they will be recognized at any point.

If the file to be disassembled is a Load file (type $Bx) then the "header" will be skipped and the first segment only will be loaded for disassembly. To disassemble other segments it will be necessary to extract the segment by other means, put it in another file (of BIN type) and disassemble that.

Macgen:

Macgen is invoked by the MACgen <filename> command from the editor's command box. The previous version sometimes generated duplicate macros, because of memory restrictions, and that has been improved. The default directory for the directory search is now the prefix #4 directory. Note that you can use another prefix for this (or for catalog) by just typing the prefix number and RTN.

Convert:

There is a new Command Box command, CONvert <pathname>, which takes an OBJ format file from APW or other languages producing such files, and converts it to a LNK-type file that is acceptable to the Merlin linkers. There are some things this conversion cannot handle, and which will produce error messages. The output LNK type file is written to the same directory as the original and its name is the same as the original with ".01" (etc.) appended. If the original file name is over 12 characters then it is truncated before the addition of the suffix.

The following record types are supported by CONvert, at least in part:

```
00         = end of segment
01-DF      = CONST
E0         = ALIGN
E1         = ORG (relative only)
E6         = GLOBAL
E7         = GEQU
EB,EC,ED   = EXPR
F1         = DS
F2         = LCONST
F3         = LEXPR
```

Convert handles files with up to 100 segments. Record types $E4 (USING) and $E5 (STRONG) are accepted, but do nothing. Record type $EF (LOCAL) is handled as though it were $E6 (GLOBAL), and type $F0 (EQU) is handled as though it were $E7 (GEQU), because Merlin 16 does not support analgous record types. Of course, these conversions could result in duplicate labels, and then the conversion would be useless. However, it would also be useless if these record types were not supported at all.

The following expression operands are supported:

```
80 = Value of location counter
81 = Absolute value
83 = Label value
84 = Label length
87 = Rel (displ. from start)
```

Supported expression operators are +, -, negation, bit shift (by -8 or -16 only), division (by $100 or $10000 only).

Some things that perhaps could be supported are not because they were not needed (or usable in REL files). Let us know if some unsupported item is important to you and we'll see what can be done about it.

CONvert also accepts a BIN file. In this case it converts the BIN file into a LNK file of pure data, and with one ENTRY label which is the filename. If the filename contains a period then it will NOT be put in as an ENTRY and thus the file will be pure data. In either case the file will be acceptable to the linkers. Note that there must be NO CODE in such a file, because it will not be relocated. This provision is for linking things like Hi-Res pictures. The output file suffix will be ".00" in this case.

Assembler Changes

XC OFF: In Merlin 8, the XC (eXtended opCodes) command could only *enable* the availability of the extra opcodes for the 65C02 and 65802/65816. This was provided as a "safety check" in Merlin 8 to prevent you from accidentally using the extra opcodes on a 6502 machines. This is because with no error message from Merlin, tracking down the apparent bug caused by a STZ instruction, for example, would be very difficult (the 6502, not recognizing the STZ would probably not crash, but rather carry out some unpredictable operation).

In Merlin 16, the extended opcodes are already enabled (i.e., the XC is not needed), with the assumption that since you are running a 65802/65816 system, the extra opcodes won't present a problem in your system. However, if you are writing programs for a 6502 machine (Apple II+), you might appreciate having the safety check available to prevent you from using them in certain programs. Merlin 16+ now allows the instruction XC OFF, which restores Merlin's extended code to OFF, i.e., 6502 codes accepted only.

IMPORTANT THOUGHT...

With the various versions of Merlin that are now around, and the wide variety of target machines and environments that any of these version of Merlin could be creating code for, the possibility exists that you could get a source file from someone, assemble it on your version of Merlin, and not get the same code output as they did. The reason for this would be that they had set the startup MX% status in their PARMS file differently than yours. Remember that both the MX% status, and the startup XC status can be set in the PARMS file.

We suggest that you make a habit of now beginning each source file with the needed MX% and XC instructions appropriate to a given program. For example, for a ProDOS 16 system file, NDA, etc., you would begin the source file with:

```
MX  %00         ; SET FULL 16-BIT MODE
XC              ; ENABLE 65C02 INSTRUCTIONS
XC              ; ENABLE 65816 INSTRUCTIONS
```

For a ProDOS 8 program, you would begin with:

```
MX  %11         ; SET 8-BIT MODE
XC  OFF         ; ENABLE 65C02 INSTRUCTIONS
```

Duplicate Labels: An optional setting in the PARMS file has been provided that, if on, will cause the assembler to ignore duplicate label errors provided they are EQUates with the same value. (This can be useful for often-used PUT files having some overlap in common equates.)

ERR: The ERR (..).. syntax in the assembler has little use in the GS version and so it is defeated *unless* the expression in parentheses evaluates to a zero-page location. Otherwise the opcode is ignored, so that old sources will still work. The main use of this opcode is to check for the presence of USR routines, and this is no longer needed since USR routines can be loaded automatically, via the PARMS file, and some routine is always active - removing the danger of calling non-existent code.

TYP: In both the assembler and the linker, the TYP directive will accept ASCII filetypes such as EXE and SYS. (This has no effect on the internals of the file, so one should not use inappropriate types such as S16 or AWP in the assembler.)

The assembler now supports TYP in REL files. Even though the saved file will have type LNK, the filetype specified in the TYP directive will be passed to the linker and will be the default file type of the final object file. This is mostly intended for use by the Apple-6 command to assemble and link without the use of a linker command file, and avoids the necessity of changing the file type of the final object file. For example, if the source has a TYP CDA line in it and is assembled with the Apple-6 command then the final file will be of CDA type.

ASC, ETC. The ASCII opcodes ASC, STR, REV, DCI, INV and FLS now can handle strings with hex data interspersed, and not just at the end as was formerly the case. INV used to map lower case characters to the $20-3F range; it now maps them to the $60-7F range which is correct for direct screen store with the alternate, or 80-column, character set. Empty strings are now accepted.

There is a new option for the ASC pseudo-op. If the operand starts with the # character then the value of the following expression is converted to ASCII and placed in the object code. If the expression follows # immediately or follows a single quote after the # then the object code will be in positive ascii, while a double quote will produce negative ASCII. In addition if the #, #', or #" is followed by > and then the expression, then the ASCII will be right justified in a field of 5 digits, otherwise it is left justified. In all cases the expression must evaluate to a 1- or 2-byte hex number.

Thus, for example:

```
ASC #123     produces object code:  31 32 33
ASC #'123    produces object code:  31 32 33
ASC #"123    produces object code:  B1 B2 B3
ASC #>123    produces object code:  20 20 31 32 33
ASC #'>123   produces object code:  20 20 31 32 33
ASC #">123   produces object code:  A0 A0 B1 B2 B3
```

DATE: The DATE opcode now accepts an (optional) operand. This has the following results:

```
Operand     Result
0 or none   Prints date/time to screen/printer, Merlin format.
  1         Date only to object code, Merlin format, 9 bytes.
            Ex:  29-DEC-88
  2         Date only to object code, Control panel format, 8 bytes.
            Ex:  12/29/88
  3         Date/time to object code, Merlin format, 22 bytes.
            Ex:  29-DEC-88   4:18:37 PM.
  4         Date/time to object code, Control panel format, 20 bytes.
            Ex:  12/29/88   4:18:37 PM.
 5-8        Same as 1-4 but data is in positive ascii.
```

Thus DATE alone is equivalent to the DATE directive in Merlin 16, and does not affect object code, while the other options do.

CAS: Merlin now allows you to set the case sensitivity within a particular source file. This will make it easier to exchange and assemble source files with others without having to re-set the bit in the PARMS file for case sensitivity. CAS SE sets case sensitive; CAS IN sets case insensitive. The setting applies only for that assembly, and otherwise defaults to the PARMS file setting.

SAV: The SAV opcode is no longer incompatible with REL assemblies. There can be only one SAV in such a source (it acts as an END) and it should be at the end. Using SAV instead of DSK has the advantage of somewhat faster operation and there is no writing to disk if an error occurs on the second pass.

Pathname Control: The pathname syntax ../PATHNAME is now supported and means: back up one directory level and append PATHNAME to that. To back up two levels use ../../PATHNAME etc. This syntax is supported only by assembler source files and linker command files. If the full pathname resulting from such syntax exceeds 64 characters then it will be defeated and a pathname error will result, because of the 64 character full pathname limit.

This syntax is useful for those cases where you don't want to have to specifically write the volume name into your source code, and would also like to be able to access a directory "parallel" to the one the source code is being assembled in, which might be the case for a library or equates file.

LST: The LST assembler directive has a new option. The syntax:

```
LST FILE,filename
```

will create (or open if already there) the file <filename> and the assembly listing output will be sent to the file as well as to the screen or printer. This replaces the Printfiler utility on older versions of Merlin. At this time there is no option to pack the output or to shut off screen output (listing must be on or you get nothing).

Unused Symbols: There is a bit in the PARMS file which will defeat listing of unused symbols in the symbol table printout. This does not affect the SYM command from the editor, so that you can see the unused labels even if you have this bit set.

STRL: New assembler directive STRL is the same as STR but makes a length *word* (two bytes) instead of a single byte. This is intended for use with GSOS for Class 1 strings.

Expression Evaluation: Operand evaluation now supports the comparison operators <, =, >, # (not =). These return 1 for true and 0 for false. Note that the characters <=> which used to be legal in labels, no longer are so.

The evaluation of "expressions" may now done algebraically, provided the expression is enclosed in braces (braces = {}). Without the braces, the expression is evaluated left-to-right as before for the reason of compatibility with old source files, and also because that is slightly faster. You can use nested braces just as if they are parentheses in traditional formulas. You can also intermix the two forms, keeping in mind that expressions in braces are evaluated first. For example, 1+{2*3}, {1+2*3}, and {1+{2*3}} all evaluate to 7; whereas 1+2*3 evaluates to 9 because the 1+2 is done before the multiplication.

Priorities of operations in an expression in braces are given by the following table:

```
< = > #
 + -          (- is subtraction here)
 * /
 & . !
 -            (unary negation operator)
```

This is from lowest priority to highest (done first), and operations on the same line have the same priority. In the case of two operations of the same priority, the left-most (in the expression) is done first. Thus 5-2-1 and {5-2}-1 evaluate to 2, while 5-2+1 and {5-2}+1 evaluate to 4.

There is a bit in PARMS which will cause all evaluation, not just in braces, to be algebraic. Present users of Merlin should consider, before adopting that option, on whether they have sources with instructions like:

```
LDA #LABEL1-LABEL2/2   or   ERR LABEL1-LABEL2/$100
```

and so on, which evaluate differently under the algebraic option. If many such items are used, it might be simpler to leave the algebraic option off. Otherwise braces will have to be inserted around the LABEL1-LABEL2.

NOTE: Because of the possibility of use of braces in future enhancements to the 65816, we reserve the case in which a brace is the first character of an expression, as in {1+2*3}. You can use this, but should be aware that its meaning may change. (Note that 1+{2*3} and +{1+2*3} give the same value.)

FLO: This new pseudo-opcode accepts an ASCII string in the operand represent a floating point number, and converts it to the 10-byte SANE extended floating point number which is placed in the object code. The string must be in the same format as for ASC. An example is:

```
FLO '2.3852e-10'
```

Also the **USR.EXAMPLE** file has been set up so that two of the functions are floating point (RPN) expression parsers. That allows putting constants like π = ATN(1)*4, and e = EXP(1), roots, and fractions in the object code, without having to convert them to the hex code and put that in by hand. That file also contains some floating point macros that are easier to use than the standard SANE macros.

In the documentation for **USR**, it should be noted that the zero page flag NOTFND has been changed to location $7C. None of the others have changed.

EXD: This is the same as EXT but declares the externals in the operand as direct page externals, and the assembler will generate direct page code for these labels. Presumably the corresponding entry labels will be direct page and defined with equates or dummy sections. If the entry is NOT direct page, then an error will be produced by the linker.

@ in LUP: The @ symbol is now handled in operands as it is in labels. That is, you can now use a statement like DFB "@" (which will be assembled as DFB "A", etc., just as you could use LABEL@ (which is assembled as LABELA, etc.)

DUM: The DUM opcode now accepts a DUM * (or any relative address, but others are not useful) in order to ease the definition of labels in a data area that is not put in the object code. This is intended to be at the end of the last linked module of a segment (before a SAV in the linker file) and the actual data area is intended to be specified in the linker file by a LINKER DS

statement before that SAV. (This does not result in any space in the object file, but creates an instruction telling the loader to reserve, and clear to zero, space at the end of the loaded program, in the same bank.)

Hints on assembler efficiency:

If you are assembling from a slow disk such as a 3.5" disk, then you can save substantial assembly time by arranging PUT and USE calls so that the second pass does no disk access. USE files are kept in memory and are never read again on second pass. PUT files are handled the same way, but if there is not enough memory to hold them all then they start overwriting one another and are *all* read again on second pass. Thus, if you find disk access (other than from DSK) on the second pass, you might consider converting some PUTs to USE. Other than this, PUTs are generally preferable because the last PUT file is always accessible after an assembly error, while USE files may not be if the USE space is nearly full (a SWAPU would just give an empty buffer). It is best to use USE only on the files least likely to need modifications.

Also SAV is somewhat more efficient than DSK, can handle files nearly 64k in length, and is now compatible with REL.

Changes to the Linkers

The default linker is LINKER.GS. To use one of the other linkers, you just have to (and *must*) put a LKV code at the start of the linker file with an operand of $00 for the absolute linker, $02 for the XL linker (two pass GS linker), and $03 for the new OMF linker (producing "OBJ" type files). An operand of $01 just leaves you in the default one-pass GS linker.

The "linkers" have been more or less integrated, so that all commands are recognized by all linkers, but are not acted upon if irrelevant (such as KIND in the Absolute Linker). In addition, the linkers are 2-3 faster, depending on the number of labels in your files. The more labels you have, the better the speed improvement will be over the equivalent link process in the older Merlin 16.

Here is a list of all the current linker commands. An + indicates that this does something in the linker that is active, and a - indicates the opposite:

	Opcode	Type of operand if any	Linker: ABS	GS	XL	OMF
	LNK	Pathname of LNK file	+	+	+	+
	LIN	(alternative syntax to LNK)	+	+	+	+
	ORG	Decimal or hex constant	+	+	+	+
	ADR	Decimal or hex constant	+	+	+	+
	TYP	Hex byte or ascii filetype	+	+	+	-
	END	<none> (optional opcode)	+	+	+	+
	PUT	Pathname of PUT or USE file	+	+	+	+
	ASM	Pathname of source file	+	+	+	+
	OVR	<none> or "ALL" or "OFF"	+	+	+	+
	SAV	Pathname of obj file or segment	+	+	+	+
	DS	Decimal or hex constant	-	+	+	+
	KND	Hex address (byte for OMF #1)	-	+	+	+
	KIN	(alternative syntax to KND)	-	+	+	+
	VER	$01 or $02 (OMF version)	-	+	+	+
	ALI	Decimal or hex constant	-	+	+	+
	LKV	$00 through $03 (linker version)	+	+	+	+
	DAT	<none>	+	+	+	+
	LIB	Directory name	+	+	+	+
	ENT	<none>	+	+	+	+
LABEL	EXT	<none> (new meaning in this vers)	+	+	+	+
	NOL	<none> (turns ASM listing off)	+	+	+	+
	FAS	<none> (1 pass, 1 SAV)	+	-	-	-
	POS	ENTRY label name or <none>	+	+	+	-
	LEN	ENTRY label name or <none>	+	+	+	-

	Opcode	Type of operand if any	Linker: ABS	GS	XL	OMF
	IF	Pathname	+	+	+	+
	DO	Decimal or hex byte	+	+	+	+
	ELS	<none>	+	+	+	+
	FIN	<none>	+	+	+	+
LABEL	EQU	Decimal or hex constant	+	+	+	+
LABEL	GEQ	Decimal or hex constant	+	+	+	+
LABEL	=	Decimal or hex constant	+	+	+	+
LABEL	KBD	Optional message	+	+	+	+

New Commands in the Linker Command Files

NOL: There is a new command NOL (no list) which causes all assemblies to be done with a default of LST OFF instead of the usual ON default.

IF: There is a new command IF (operand a source file name, or PUT or USE file) which tests for a change in this file and, if it has changed, sets the linker to assemble all subsequent files in ASM lines. (I.e., if file has changed then OVR ALL is put into effect.)

DO, ELSE, FIN: There are new commands DO, ELSE, FIN which do as expected.

ENT: The ENT command is supported in all linkers and symbols in the entry list are now flagged with a ? if they are not used by an external from another module. The ENT code should not be used until after the last SAV.

OVR OFF: OVR OFF has been added to the linker commands. This turns off the effect of a previous OVR ALL or an IF.

Pathname Control: The pathname syntax ../PATHNAME is now supported and means: back up one directory level and append PATHNAME to that. To back up two levels use ../../PATHNAME etc. This syntax is supported only by assembler source files and linker command files. If the full pathname resulting from such syntax exceeds 64 characters then it will be defeated and a pathname error will result, because of the 64 character full pathname limit.

This syntax is useful for those cases where you don't want to have to specifically write the volume name into your source code, and would also like to be able to access a directory "parallel" to the one the source code is being assembled in, which might be the case for a library or equates file.

Labels: The linkers used to act on labels in operands only if they were defined by =. GEQ, EQU, and KBD have been added for more flexibility.

LABEL GEQ <Decimal or hex constant>
This defines a global absolute label that will be in effect for *all* subsequent ASMs. These labels must not be defined in the source files except in KBD lines or a duplicate label error will occur. If the label is defined in the source file by a KBD pseudo-op then its value is taken from this linker operand instead of from the keyboard and will *not* give a duplicate label error.

LABEL KBD <Optional string for display>
This is the same as GEQ but the label is defined via keyboard input, just as in the assembler directive of the same name.

LABEL EQU <Decimal or hex constant>
This is the same as GEQ but the label is active for the following assembly only and is discarded after that assembly.

There is no need for a local form of KBD since that can be done inside the asser ource files.

There is no duplicate label checking for these labels defined in the linker command file in the linker itself, but any subsequent assembly will give a "duplicate label error in line 0" error message on the first pass and will reject a REL opcode. In the case of the EQU command, you can use the same label with the same or different values for more than one assembly.

LABEL = <Decimal or hex constant>
This defines labels for the linker's internal use (eg., in DO statements). The label is not passed to the assembler. The linkers will also accept labels defined by GEQ and KBD, but not by EQU, in its own operands.

Note: the linker does not do arithmetic on expressions. Only hex, decimal or single labels are acceptable for linker operands.

EXT: The EXT linker command is now reenabled, but does a different task from the former one. If an EXT line has no label then it is ignored as before. If it has a label then the linker looks for an ENTry value for the label and assignes that value to the label for the linker's internal use. The label *must* be an ABSOLUTE label or a BAD ADDRESS error will occur. The purpose of this provision is to communicate the size of a buffer to the linker, so that the linker can reserve that space through a DS linker command. See the MACRO.EXAMPLE source file and MACRO.CMD linker file for an example. This linker command *must* come *after* the LNK of the module containing the label as an ENTry, and *before* the use of the label by the linker, because the lookup is done on the first linker pass.

Linker.XL: Linker.XL now supports 64 segments (it is 25 in Merlin-16).

The XL-linker (LKV 2) now supports dynamic segments (KIND $80 in OMF #1 or $8000 in OMF #2). [Please note: If you want to test this, or any other type of program, make sure that

all segments have at least 2 bytes of *executable code*. Versions before 1.4 of the loader crash if presented with a load segment of only one byte!

NOTE 1: Remember that a dynamic segment can be referred to only by a JSL, and, moreover, the JSL must be made in full 16-bit mode.

OMF Linker

There is a 4th linker (LKV $03) that produces "object files" in OMF from Merlin's LNK files. Use it in the same way as the other linkers (include the LKV $03 command at the start of the linker command file). It permits just one SAV and makes only a one-segment OMF file. This is not a real restriction since unresolved externals are permitted in such files. Such files are used as input to the APW linker, and the purpose is to be able to link with files produced by other languages. For pure assembly code, there is no reason to use this and many reasons not to use it (speed for one).

This linker will resolve all externals for which there are corresponding entry labels. Other external references are left unresolved. All entry labels are put into the object file as "global" labels.

Present versions of the APW and Orca linkers cannot handle object files in OMF version #2 format, so you must include the linker command VER $01 in linker files for this linker version until this situation changes. If you forget to use VER $01, then you will get the error "Terminal error, Linker version mismatch" from the APW linker.

EXE files (and USR opcodes)

EXE files can be run from the Main Menu disk command, by the -pathname command. An EXE ("executable") file is different from an S16 (ProDOS 16 System) file, in that it usually ends with an RTL, rather than the ProDOS Quit command. EXE files are also the form required for Merlin's shell commands, such as the HELP command. In the COMMAND.SOURCE directory on the Merlin Samples disk, you'll find several example source files for EXE files.

When an EXE file is launched from the Main Menu disk command, or by the Command Box as a shell command, it is executed in full 16-bit mode, with (as per Apple protocol) the X-Register holding the high word and Y the low word of the address of the "identifier" string, which is the string 'MERLINGS'. Following the string is the ASCII command (eg, -PROGRAM) that caused execution of the program. Also, as part of protocol, the A register holds the memory ID number for Merlin. In addition (not part of protocol), the stack holds, just above the return address, the low word (high word still in X) of an address table.

The table consists of low words (high word, same as before, in X) of special entry points in Merlin that can be used by the EXE file. At present the table has three entries for USR opcode support and a number of others for other purposes. See the USER.EXAMPLE source for documentation on the latter.

Generally, EXE files are expected to return via an RTL, but a quit command is also supported.

When executed, an EXE file will have its own stack and direct page established. Since some EXE files, particularly USR opcodes, would like to access Merlin's direct page, that address (word) is pushed onto the EXE file's stack just above the "address table" pointer word (see above) and the three return bytes from the JSL to the EXE file. In the case of USR opcode files, keep in mind that the subsequent entries via the USR opcode will have Merlin's stack and direct page up, so the USR routine will have to switch its own in and out in the unlikely event that that is needed. Locations $60-$6F in Merlin's direct page are free, as in previous versions, for use by any user supplied routine, and there is plenty of free space on the stack.

A special format is defined for Merlin utilities (optional for general EXE files). This is so that Merlin can identify special types of EXE programs and act accordingly. This format is as follows.

The program must begin (in full 16-bit mode) with the following list of 8 long branches:

```
START   BRL INIT
        BRL ENTRY1
        ...
        BRL ENTRY7
        DA  ID
        ASC 'MERLIN'
```

The remainder of the file is optional.

Files with ID 0, or EXE files that do not conform to this protocol are regarded as "shell commands" and will be run when the file name (e.g. HELP, TYPE and DUMP) is given from the "Disk command" prompt or the full screen editor's Command Box. Such files are shut down as soon as they return to Merlin via an RTL. Utilities that you do not want shut down upon return must not use this ID.

The ID 1 is for USR opcode files and will cause Merlin to link in the address of the BRL ENTRY1 to the USR opcode. [The entries ENTRY2 to ENTRY7 are not used at this time.] As formerly, the USR routine is entered in 8-bit mode with X = double the number after the USR. (I.e., USR2 causes a call to the BRL ENTRY1 with X=4.) In all cases, as formerly, USR routines are entered with A=Y=0 and carry set. One difference is that they are now called, of necessity, with a JSL and hence must exit with an RTL. The 3-word address table referred to above consists of the addresses:

```
DA  OPERAND    (where the operand is)
DA  PRCODE     (routine to put A in the object code, 1 byte)
DA  EVAL       (routine that evaluates the part of the operand
                pointed to by X)
```

The routines PRCODE and EVAL must be called by JSLs; register conditions are not important, except for X in EVAL. See older docs for other details. When the USR routine terminates, it should clear the carry and RTL if no error condition exists. If it wants to report an error, it should RTL with carry set and the error code in X. The error codes are:

```
X =
0 --- bad opcode
1 --- bad addressing mode
2 --- bad operand
3 --- bad label
4 --- bad input
5 --- unknown label
6 --- illegal forward reference
7 --- operand too long
8 --- illegal character in operand
```

See the source file USR.EXAMPLE for hints on how this facility can be used. Try it out! Just type -USR.EXAMPLE as a Disk command from the Main Menu, and then assemble some code with the USR (=USR0), USR1 and USR2 codes, as indicated in the source file.

EXE files with the special format and ID=2 are for editor keyboard macro programs (see the example source), and ID=3 is for the editor USER command use, but all former USER utilities are expected to be resident. The USER.EXAMPLE source shows how to use the latter and documents how a USER file can call the assembler or linker.

GS Toolsets:

The following tool sets are active when Merlin 16+ is up. Thus, EXE files may assume these tools to be available. They must not be shut down by the EXE file. Other tools needed by an EXE file must be started up on entry and shut down on exit. S16 programs must start up and shut down *all* tool sets they use.

Tool Locator
Misc Tools
Text Tools
Memory manager
SANE

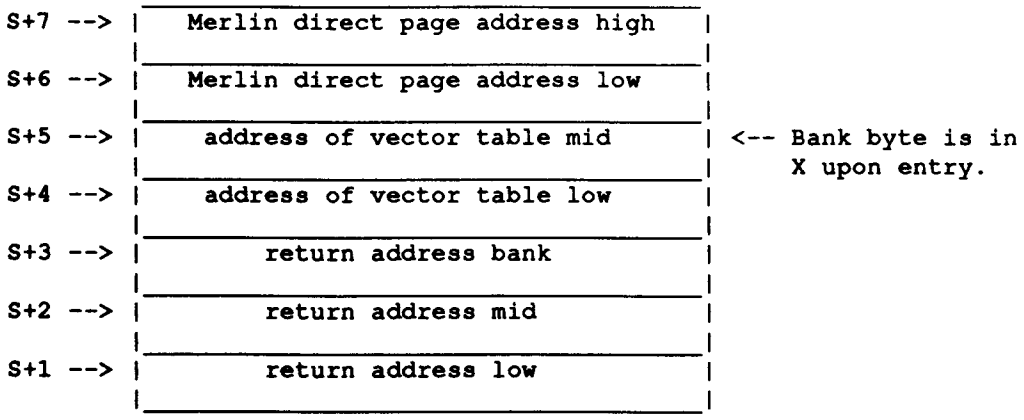
Apple protocol for EXE files:

File is executed in full 16 bit mode with X=high word of caller address (Merlin bank), Y=low word of identifier string, and A=caller ID.

Identifier string is 8 bytes (MERLINGS for Merlin), followed by the actual command string used in executing the file (eg, -FILE).

Program may exit with RTL (in any mode) or a "quit".

Stack at entry to EXE program is (execution from Merlin only):



Format of special-purpose EXE files:

EXE program must start with 8 long branches, the first one to an init routine (which is the one called when the file is executed), then an ID word, and an identifier string 'MERLIN':

```
START BRL INIT
      BRL ENTRY1
      BRL ENTRY2
      BRL ENTRY3
      BRL ENTRY4
      BRL ENTRY5
      BRL ENTRY6
      BRL ENTRY7
      DA  ID
      ASC 'MERLIN'
```

The IDs presently assigned to special purpose EXE files are:

0 = shell command (special structure is optional for these files)
1 = USR opcode support program
2 = keyboard macro program
3 = USER editor command program

Any EXE file with ID 0 or that does not conform to this added protocol will be treated as a shell command (or "transient" command). It is executed without releasing Merlin's memory, and the file is purged upon completion.

Any EXE files with a non-zero ID are NOT purged upon return to Merlin, but are purged when you quit from Merlin.

Vector table (see stack above). Vectors are words, bank is in X:

Byte offset	Name of vector	Purpose of vector
0	OPERAND	address of assembler's operand (for USR only).
2	PRCODE	put byte (A) in object code (for USR only).
4	EVAL	evaluate expression at OPERAND + X (USR only).
6	RNGLIST	address of range in USER command, also where linkers look for the command file name.
8	STRINGS	address of null terminated string given with the USER command.

10	FILENAME	address of default filename for source load/save.
12	HANDLES	address of table of 4-byte handles to buffers.
14	GOASM	address of routine to assemble file in buffer.
16	GOLINK	address of routine to link, cmd filename at RNGLIST.
18	PARMS	address of Merlin's parms ($200 long).

Handles pointed to by the HANDLES vector in vector table:

Byte offset	Handle points to	Remark
0	Source file buffer	File names are at byte 0 of buffers,
4	PUT file buffer	buffers start at relative byte $40,
8	USE file buffer	and are 0 terminated. All buffers are bank aligned.
12	Clipboard	Don't alter!!
16	Object file buffer	Starts at byte 0. (Also used by history)
20	Short symbol buffer	
24	Long symbol buffer & linker dictionary	Also used by history, etc.
28	Local labels	Also used by history, etc.
32	XREF buffer	Also used by history, etc.

Files on the Merlin 16+ Disks

The software associated with Merlin 16+ is contained in two disks, /Merlin and /Merlin.Samples. The following is an explanation of the various files on the two disks.

/MERLIN - Main Directory:

MERLIN.SYS16	Main system file for Merlin 16+.
PARMS	Parameters file for Merlin 16+. Loaded when Merlin 16+ is started, or the NEWPARMS command is used.
LABELS	File of labels used by Sourceror. Loaded when Merlin 16+ is started.
SOURCE	Source files for Merlin 16+ PARMS and LABELS files.
OBJECT	Output object files for USR.EXAMPLE, USER EXAMPLE, RND.TEST, and ERR.USR.
MACRO.LIBRARY	Macro library for use with Merlin source files. Includes GS Toolsets, Utility macros, ProDOS 8, ProDOS 16, and GSOS macros as well.
SUBROUT.LIB	Library of linkable subroutines that can be included in your own programs. See the corresponding source files in the COMMAND.SOURCE directory on the Merlin Samples disk.
COMMANDS	Merlin 16+ shell (EXE) commands. Source for these are in the COMMAND.SOURCE directory.
HELP.FILES	Text files of online help available in Merlin 16+.
TOOL.EQUATES	Equates files for GS programming.
ICONS	Contains the file MERLIN.ICONS, which can be used with the Finder.

SOURCE Subdirectory:

This directory contains just the source files for the PARMS and LABELS files used by Merlin.Sys16 when it starts up. Other source files are on the second disk, /Merlin.Samples.

PARMS.S	Source file to be changed and re-assembled if you want to change any of the startup parameters (defaults) of Merlin 16+. PARMS in the main directory must be re-saved from a new assembly for changes to take effect.
LABELS.S	Source file that defines the labels used by Sourceror. You can add your own label definitions as you wish. LABELS in the main directory must be re-saved from a new assembly for changes to take effect.

OBJECT Subdirectory:

Contains the object files for the USR.EXAMPLE, USER.EXAMPLE, and RND.TEST. Install the USR and USER examples by typing -USR.EXAMPLE and -USER.EXAMPLE from the main menu of Merlin 16+. Make sure the prefix has been set to the OBJECT directory before running. RND.TEST is a ProDOS 16 test program of the RANDOM function in the subroutine library. Type =RND.TEST from the main menu to try it. Pressing the Escape key returns you to Merlin 16+. Prefix 3 is normally set to this directory.

This also contains the file ERR.USR, which is a debugging aid automatically loaded by the current setup of Merlin 16+ when it starts up. This is used in the GENERIC.SOURCE and SKELETON example files on the Merlin Samples disk. See the comments in the documentation for the Sample disk, and in the source files themselves for more information.

MACRO.LIBRARY subdirectory:

This directory contains all the Apple IIGS Tool macros. The definitions also include the new "Super Macros", which allow you to significantly reduce the number of lines of source code needed for each tool call. In addition, each Super Macro call automatically places the proper bytes on the stack, which helps reduce those hard-to-find bugs caused by an improper toolcall setup. See the Merlin 16 Supplement for details on how to use the Super Macros. Prefix 4 is normally set to this directory, which is the default for scanning by the Macgen function. Other special files in the MACRO.LIBRARY include:

DOS.8.MACS.S — The "official" Apple Computer ProDOS 8 macro calls, modified to use another macro, DOS8. This is use to save space in the macro definition file. Use the syntax:

```
READ_BLOCK PARMADR
```

for example, where PARMADR is the address of the parameter table.

DOS.16.MACS.S — The "official" Apple Computer ProDOS 16 macro calls, again, modified to use another macro, DOS16, to save space in the macro definition file. Syntax is:

```
_SET_FILE_INFO PARMADR
```

for example, where the command name is followed by the address of the parameter table.

MACROS.S — General-purpose macros for use in 8-bit and 16-bit programs. These include:

ASL4 — Arithmetic Shift Left four bytes (Label...Label+3); assumes 16 bit mode. Equivalent to multiply by 2. Use the X-reg. for a shift count, or follow with a second parameter.

```
Ex: ASL4 Label          ; Multiply by 2 if X = 1
    ASL4 Label;3        ; Multiply by 8
```

INCD — Increments two bytes (Label, Label+1), regardless of MX.

```
Ex: INCD Label        ; Label = Label + 1
```

INC4 — Increments four bytes (Label...Label+3); assumes 16 bit mode.

```
Ex: INC4 Label        ; Label = Label + 1
```

DECD — Decrements two bytes (Label, Label+1) regardless of MX.

```
Ex: DECD Label        ; Label = Label - 1
```

DEC4 — Decrements four bytes (Label...Label+3); assumes 16 bit mode.

```
Ex: DEC4 Label        ; Label = Label - 1
```

DP — Creates 4-byte contstant. Equivalent to ADRL Label (for APW compat.)

```
Ex: DP    Label       ; Equivalent to ADRL Label
```

LSR4 — Logical Shift Right four bytes (Label...Label+3); assumes 16 bit mode. Equivalent to divide by 2. Use the X-reg. for a shift count, or follow with a second parameter.

```
Ex: LSR4 Label        ; Divide by 2 if X = 1
    LSR4 Label;3      ; Divide by 8
```

MOV — Moves byte or word from Label1 to Label2, depending on MX.

```
Ex: MOV  Label1;Label2  ; Move contents of Label1 to
                          Label2
```

MOVD — Moves word from Label1 to Label2, regardless of MX.

```
Ex: MOVD Label1;Label2  ; Move 2-byte contents of
                          Label1 to Label2
    MOVD (Label1),Y;Label2 ; Move contents of (Label1),Y
                          to Label2
    MOVD #Label1;Label2 ; Puts 2-byte value Label1 in
                          Label2
```

LDHI — Loads Acc. with hi-byte value if immediate (#), or hi-byte value from Label, (Label+1). Used by other macros; assumes 8-bit mode.

```
Ex: LDHI #Label1      ; Load hi-byte of value of Label1
    LDHI Label1       ; Load contents of Label+1
```

ADD — Adds one or two bytes (constant or label contents), depending on MX status. Stores in Label3 if given, Label2 if no 3rd label, "other" label if a constant is used.

```
Ex: ADD  #Label1;Label2 ; Adds value of Label1 to
                          contents of Label2, result left
                          in Label2.
```

```
ADD  #Label1;Label2;Label3  ; As above, but stores
                              result in Label3.
ADD  Label1;Label2  ; Adds contents of Label1 to
                      contents of Label2, result in
                      Label2.
ADD  Label1;Label2;Label3  ; As above,but result stored
                             in Label3
ADD  Label1;#Label2 ; Add contents of Label1 to
                      value of Label3, result in
                      Label1.
```

ADD4 Adds 4-byte constant or value to 4-byte constant or value that follows. If no third label, result is stored in second parameter, as long as that position is not an immediate value. If a third label is given, result is stored there, and an immediate value is allowed in the second position.

```
Ex: ADD4 #Label1;Label2 ; Add value of Label1 to
                          contents of Label2, result in
                          Label2.
    ADD4 #Label1;Label2;Label3 ; Add value of Label1 to
                                 contents of  Label2,
                                 result in Label3.
    ADD4 Label1;#Label2;Label3 ; Add contents of Label1 to
                                 value of  Label2,
                                 result in Label3.
```

SUB Subtracts one- or two-byte values, depending on MX status. Stores in Label3 if given, Label2 if no 3rd label, "other" label if a constant is used.

```
Ex: SUB  Label1;#Label2 ; Subtract value of Label2 from
                          contents of Label1. Result in
                          Label1.
    SUB  Label1;Label2  ; Subtract contents of Label2
                          from contents of Label1. Result
                          in Label2.
    SUB  #Label1;Label2 ; Subtract contents of Label2
                          from value of Label1. Result in
                          Label2.
    SUB  Label1;Label2;Label3 ; Subtract contents of
                                Label2 from contents of Label1.
                                Store in Label3.
    SUB  #Label1;Label2;Label3 ; As above, but use value
                                 of Label1.
```

SUB4 Subtracts 4-byte constant or value from another 4-byte constant or value that follows. If no third label, result is stored in second parameter, as long as that position is not an immediate value. If a third label is given, result is stored there, and an immediate value is allowed in the second position.

```
Ex: SUB4 #Label1;Label2 ; Subtr. contents of Label2
                          from value of Label1, result in
                          Label2.
    SUB4 #Label1;Label2;Label3 ; Subtr. value of Label2
                          from value of Label1, result in
                          Label3.
    SUB4 Label1;#Label2;Label3 ; Subtr value of Label2
                          from contents of Label1, result
                          in Label3.
```

ADDX Add X-Register to Label1 (contents or value), and store in Label2. Byte or word, depending on MX. If 8-bit mode, Label2+1 is incremented if needed.

```
Ex: ADDX Label1;Label2  ; Add X-reg to Label1, result in
                          Label2.
```

ADDA Add accumulator to Label1 (contents or value), and store in Label2. Byte or word, depending on MX. If 8-bit mode, Label2+1 is incremented if needed.

```
Ex: ADDA Label1;Label2  ; Add acc. to Label1, result in
                          Label2.
```

ADDY Add Y-Register to Label1 (contents or value), and store in Label2. Byte or word, depending on MX. If 8-bit mode, Label2+1 is incremented if needed.

```
Ex: ADDY Label1;Label2  ; Add Y-reg to Label1, result in
                          Label2.
```

ADDNUM Adds *value* of Label1 to contents of Label2. Result in Label2. Byte or word, depending on MX. If 8-bit mode, Label2+1 is incremented if needed.

```
Ex: ADDNUM Label1;Label2 ; Add value of Label1 to
                           contents of Label2,
                           result in Label2.
```

SWAP Swaps contents of Label1 with contents of Label2, using the stack. Byte or word, depending on MX.

```
Ex: SWAP Label1;Label2  ; Exchange contents of Label1
                          with contents of Label2.
```

COMPARE Compares 2-byte value or contents of Label1 with Label2. Follow with branch instruction or macro to complete test.

```
Ex: COMPARE Label1;Label2 ; Compare contents of Label1
                            with contents of Label2.
                            Follow with BCC, BGE, BCS, BLT,
                            etc.
```

```
COMPARE Label1;#Label2; Compare contents of Label1
                        with value of Label2.  Follow
                        with BCC, BGE, BCS, BLT, etc.
```

POKE — Stores 1- or 2-byte value of Label1 in Label2, depending on MX. Note that value stored is *second* parameter in macro!

```
Ex: POKE Label2;Label1  ; Store value of Label1 in
                          Label2.
```

STADR — Stores 2-byte value of Label1 in Label2, regardless of MX. Note that value stored is *second* parameter in macro!

```
Ex: STADR Label2;Label1 ; Store value of Label1 in
                          Label2.
```

SAVSTAT — Saves Acc., X-reg, Y-Reg, and status reg. on stack, in that order. Use at start of routine which might be called from unknown status and register lengths.

```
Ex: SAVSTAT             ; Save A,X,Y,and P
```

RESTORE — Restores registers and status. Assumes SAVSTAT has been used.

```
Ex: RESTORE             ; Restore P,Y,X, and A
```

The macros BLE ("Branch Less Than") and BGT ("Branch Greater Than") create branches that are not available on the 65816. All the additional branches that end in "L", for example, BLTL, BCCL, etc., are conditional long branches for use when the destination is out of range for a short branch.

BLE — Branch Less Than.
BGT — Branch Greater Than.

BLTL — Branch Less Than (long).
BCCL — Branch Carry Clear (long).
BGEL — Branch Greater or Equal (long).
BCSL — Branch Carry Set (long).
BPLL — Branch Plus (long).
BMIL — Branch Minus (long).
BEQL — Branch Equal (long).
BNEL — Branch Not Equal (long).
BVCL — Branch oVerflow Clear (long).
BVSL — Branch oVerflow Set (long).
BLEL — Branch Less than or Equal (long).
BGTL — Branch Greater Than (long).

The following macros are for use with text output, and expect the COUT subroutine to be linked in, or supplied as an external. PRINT expects the SENDMSG subroutine.

For easiest use, just declare COUT and SENDMSG as externals (EXT) in your program, and include the Linker command LIB 5 in the Linker command file. SENDMSG also requires an OUTPUT routine in your source declared as an entry (ENT). It can be as simple as:

```
OUPUT    ENT              ; Declare Entry label
         JMP COUT         ; Use COUT routine
```

SENDMSG and COUT are in the SUBROUTINE.LIB directory, and their source files are on the Merlin.Samples disk.

PRINT — Print ASCII text.

```
Ex: PRINT "This is a test"  ; Print message
    PRINT 8D8D"text"8D"more"8D; Print 2 crs, "text",
                                cr,"more",cr
```

OUT — Output Label contents or value through COUT routine.

```
Ex: OUT  #"T"          ; Output letter "T"
    OUT  Label1        ; Output contents of Label1
    OUT  #Label1       ; Output value of Label1
    OUT  (Label1),Y    ; Output contents of (Label1),Y
```

HOME — Clear 80 column screen. Outputs Control-L.

```
Ex: HOME               ; Clear 80 col. screen.
```

NORMAL — Set normal text. Outputs Control-N.

```
Ex: NORMAL Set normal text output.
```

INVERSE — Set inverse text. Outputs Control-O.

```
Ex: INVERSE            ; Set inverse text output.
```

MOUSEON — Turn mousetext characters on. Outputs Escape,Control-O.

```
Ex: MOUSEON            ; Turn on Mousetext char. set.
```

MOUSEOFF — Turn mousetext characters off. Outputs Escape,Control-N.

```
Ex: MOUSEOFF           ; Turn off Mousetext char. set.
```

BELL — Ring bell. Outputs Control-G

```
Ex: BELL               ; Ring bell
```

CLEOL — Clear from cursor position to end of line. Outputs Control-].

```
Ex: CLEOL              ; Clear to end of line.
```

CLEOP — Clear from cursor to end of page (screen). Outputs Control-K.

```
Ex: CLEOP              ; Clear to end of page.
```

OUTCR Output a carriage return (Control-M).

```
Ex: OUTCR              ; Print carriage return.
```

GOTOXY Uses Pascal protocol for positioning screen cursor.

```
Ex: GOTOXY #55,#10          ; HTAB 55, VTAB 10
    GOTOXY Label1,Label2     ; Use contents of
                               labels.
```

CURSXY Uses CH ($24) and 'n' carriage returns to position cursor. (BASIC output.)

```
Ex: CURSXY #55,#10          ; HTAB 55, VTAB 10
    CURSXY Label1,Label2     ; Use contents of
                               labels.
```

UTIL.MACROS General-purpose macros specifically oriented to ProDOS 16 application programming. Utility Macros include:

Push4 "Push 4" Pushes four *words* (8 bytes) on the stack.
Ex: Push4

PushPtr "Push Pointer" Pushes immediate 4-byte value of Label.

```
Ex: PushPtr Label
```

PushLong "Push Long" Pushes 4 bytes immediate or contents of Label.

```
Ex: PushLong #Label or PushLong Label
```

PushWord "Push Word" Pushes 2 bytes immediate or contents of Label.

```
Ex: PushWord #Label or PushWord Label
```

PullLong "Pull Long" Pulls 4 bytes from stack. If label given, stores result there.

```
Ex: PullLong or PullLong Label
```

PullWord "Pull Word" Pulls 2 bytes from stack. If label given, stores result there.

```
Ex: PullWord or PullWord Label
```

MoveLong "Move Long" LDA/STA operation for 4 bytes using source & target labels.

```
Ex: MoveLong Label1;Label2
```

MoveWord "Move Word" LDA/STA operation for 2 bytes using source & target labels.

```
Ex: MoveWord Label1;Label2
```

MoveBlock "Move Block" Moves block of memory using the MVN or MVP opcodes.
```
Ex:  MoveBlock  Firstbyte;Lastbyte;Destination
```

CmpLong "Compare Long" Compares two 4-byte immediate values, or contents of labels.
```
Ex:  CmpLong  #$01234;Label
```

The next group are macros used to set the register length of the accumulator and/or index registers. If the processor is not already in the appropriate mode, the macro will generate the needed REP or SEP instruction.

LONGM	"Long Memory" Sets M-bit to 0, i.e., long (2-byte) memory/accumulator.
LONGACC	"Long Accumulator" Same as above.
LONGX	"Long X" Sets X-bit to 0, i.e., long (2-byte) index registers.
LONGXY	"Long X,Y" Same as above.
LONG	"Long" Sets *both* M and X bits to 0, i.e., full 16-bit mode for all registers.
LONGAX	"Long A and X" Same as above.
SHORTM	"Short Memory" Sets M-bit to 1, i.e., short (1-byte) memory/accumulator.
SHORTACC	"Short Accumulator" Same as above.
SHORTX	"Short X" Sets X-bit to 1, i.e., short (1-byte) index registers.
SHORTXY	"Short X and Y" Same as above.
SHORT	"Short" Sets *both* M and X bits to 1, i.e., 8-bit mode for all registers.
SHORTAX	"Short A and X" Same as above.

The LONGI and LONGA macros are provided to correspond to an equivalent APW function, LONGI ON, LONGI OFF, etc. These control the register length of the assembler (not the processor), and are equivalent to a given MX % command. Unlike APW, Merlin does not require that you manually set the MX mode after using a REP, SEP, or XCE instruction, but its correct (redundant) use, won't hurt anything, and may make the source code easier to follow.

In Merlin, the only legitimate use of the MX-control macros, would be to begin those subroutines, for example, that are in 8-bit mode, but which are called from some other part of your program, and would otherwise appear to the assembler as being in 16-bit mode.

LONGI "Long I" Sets assembler mode for index registers. APW uses LONGI ON to set 16-bit mode, LONGI OFF for 8-bit mode. Use numeric values in Merlin, i.e., LONGI 0 for 8-bit mode, LONGI 1 for 16-bit mode. The Converter program automatically makes this conversion for both LONGI and LONGA.

LONGA	"Long A" Sets assembler mode for the accumulator. APW uses LONGA ON to set 16-bit mode, LONGA OFF for 8-bit mode. Use numeric values in Merlin, i.e., LONGA 0 for 8-bit mode, LONGA 1 for 16-bit mode.

The following group of macros is designed to replace other APW assembler directives, but may be useful in original Merlin files as well.

M65816	APW uses this instruction to specify the current language. In Merlin, this is implemented as XC,XC,MX%00, which sets the full 16-bit mode for the assembly, with 65816 opcodes enabled. It can also be used as M65816 1 or 0, with 1 being "on", and 0 being "off", i.e., the 6502 8-bit mode.
Native	APW equivalent to set the processor to full 16-bit mode.
Native Long	Same as above.
Native Short	If operand following "Native" is short (or anything other than "Long"), processor is set to native mode, but with short (8-bit) registers.
Emulation	Sets processor to 65C02 (8-bit) mode. Personally, we dislike the term "emulation" because it implies the processor is in a separate mode that doesn't recognize 65816 instructions like MVN, etc. This is very much *not* the case. All the "emulation" mode does is to prevent use of the REP or SEP instructions. Our preference would be to call the E-bit the "Enable bit", as it is a more accurate description to say that it simply enables the option of 16-bit registers.
Expmac	Enables expansion of macros within source code. Replaces APW's GEN function. Expmac 0 = GEN OFF (no macro expansion); Expmac 1 = GEN ON (expand macros).

The following macros are provided to make the definition of menu items in the desktop environment much easier. Normally when writing a GS program, the menu items are individually numbered in the source code. In addition, when a menu event is returned by GetEvent or Taskmaster, your program usually does a comparison against a hard-coded number to see what menu item has been chosen. The problem is that with hard-coded numbers, it is difficult to insert new menu items in your program. The Item macro automatically increments a menu item number for each menu entry, and lets you set program labels equal to the current item number. This means you can insert and delete menu items at any time with no need to re-code existing parts of your program.

Item — Allows the definition of a single menu item. Also allows you use easier-to-remember words like Bold, Divide, etc. when creating a menu item. Item automatically increments the variable]inum after each item. A label can also be set to]inum *after* the menu item for use in later comparisons with a GetEvent or Taskmaster value. Example:

```
]inum  =  255          ; 1st item will be #256
       Item 'Choice1 ';Kybd;'Bb';Check ; menu item
                       with checkmark and kybd equiv.
Ch1    =    ]inum   ; set Ch1 = current item
       Item 'Choice2';Disable;'';Kybd;'Cc'
       Item 'Choice3';Divide;''
```

See the source file comments for more details on this macro.

Menu — This is a macro specifically for incrementing the menu id #. The variable]mnum should be initialized to -1 prior to using this macro.

```
]mnum =    0    ; menu item #
      Menu '@';X;defines Apple menu (color hiliting)
      Menu 'Menu Title   ; defines menu #1
```

MItem — This is a simpler form of the Item macro. It doesn't allow the automatic creation of the modifier characters, but they can be added manually as a second parameter to the macro. Like the Item macro, the variable]inum must be initialized at the startup value *minus 1*.

The Tool macro is a special-purpose macro, required as a foundation macro for all of the GS Tool macros. Its purpose is to replace the LDX #$1234, JSL $E10000 in every tool call with a simple Tool $1234 equivalent.

Tool — Required macro for the GS Tool library.

The following macros are macros to read and write characters from the current text input and output device (usually the keyboard and 80 column text screen).

WriteCh — Writes ASCII character. Examples:

WriteCh	Print character in Acc.
WriteCh Label	Print contents of location Label (1 char).
WriteCh #Label	Print ASCII character of Label value.
WriteCh Label,Y	Use Label with indexed addressing.

ReadCh — Get single character. If label follows, then store at that location.

WriteStr — Similar to WriteCh, but prints string with starting length byte. Examples:

WriteStr	String pointed to by A,Y (address,bank).
WriteStr Label	String located at Label.
WriteStr #"string"	Immediate value of string data.

WriteLn — Similar to WriteStr, but terminates line with a carriage Return. WriteLn with no label prints only a carriage return.

The MLI16 macro is designed to be used with the CALLDOS subroutine, which is in the SUBROUTINE.LIB directory. The CALLDOS subroutine is linked into your program by a linker Command File. It also uses a set of constants values equated in the file MLI.CODES.S, also in the SUBROUTINE.LIB directory. See the CALLDOS source file on the /Merlin.Samples disk for more details.

MLI16 — Macro for easily doing ProDOS 16 calls. Sample syntax:

```
MLI16 close;CLSPARMS
```

where close is a label equated in the MLI.CODES file, and CLSPARMS is the address of the parameter table for the close call. When doing Class 0 calls, you may find the HELP file MLI.CALLS helpful: this shows the parm table structure for Class 0 ProDOS 16 calls. For Class 0 calls, use the HELP file CLASS.1.CALLS.

The following group of macros in the Util.Macros is provided specifically as part of the "Super Macro" in the Merlin 16 macro library. The macros such as PHWL, etc., are used to often-done operations in macro form. There are many times within the GS Tool operation, for example, that one would want to push a Word (2-byte) value, followed by pushing a Long (4-byte) value. PHWL does this with one "command", and is used in many of the "Super Macros". Chances are, you will not need these macros in your own code, but descriptions are provided here for your information.

PHWL — "Push Word/Long" Pushes Word value, *then* pushes Long value.

```
Ex:   PHWL   Label1;Label2
```

PHLW — "Push Long/Word" Pushes Long value, then Word value.

PxW — "Push x Words" Pushes up to four Word values, depending on how many labels follow the macro call.

PxL — "Push x Longs" Pushes up to four Long values, depending on how many labels follow the macro call.

P2SL — "Push 2 word Space/Long" Pushes 2 Words for space on stack (4 bytes) followed by Long value given by label.

```
Ex:  P2SL  Label1
```

P1SL — "Push 1 word Space/Long" Pushes 1 Word for space on stack (2 bytes) followed by Long value given by label.

```
Ex:  P1SL  Label1
```

PHL	"Push Long" Equivalent to Push Long (see earlier macros)
P2SW	"Push 2 space/Word" Pushes 2 Words for space on stack (4 bytes) followed by Word value given by label.
P1SW	"Push 1 space/Word" Pushes 1 Word for space on stack (2 bytes) followed by Word value given by label.
PHW	"Push Word" Equivalent to Push Word (see earlier macros)
PushSpace	"Push Space" Pushes specified number of bytes or words, depending on current status of the MX flag. If 8-bit, pushes x number of bytes. If 16 bit, pushes x number of words. `Ex: PushSpace 4`
PHS	"Push Space" Equivalent to PushSpace, above.
PUSH1	Pushes 1 byte on stack, regardless of MX. Provided for APW compatibility.
PUSH3	Pushes 3 bytes on stack, regardless of MX. Provided for APW compatibility.
PULL1	Pulls 1 byte from stack, regardless of MX. Provided for APW compatibility.
PULL3	Pulls 3 bytes from stack, regardless of MX. Provided for APW compatibility.

SUBROUT.LIB Subdirectory:

This contains the link files created by the source files in the SUBROUT.SOURCE directory on the /Merlin.Samples disk. See the descriptions in that section later for a general overview of the functions of each subroutine. Also see the comments in each source file for more details. Prefix 5 is normally set to this directory.

COMMANDS Subdirectory:

This contains the EXE files that execute each shell command from Merlin GS. The source files for each of these are located in the COMMANDS.SOURCE directory on the /Merlin.Samples disk. Also see the comments in each source file on the /Merlin.Samples disk for more details. Prefix 6 is normally set to this directory.

ASCII	This prints an ASCII chart with characters and hex values, complete with Mousetext. It also shows how the subroutine libraries can be easily used in a program with the EXT command (example: EXT PRBYTE, PRBL, SENDMSG, COUT), and how an entry point OUTPUT in the user program is declared for use by the library subroutine (example: ENT OUTPUT).
COMMANDS	Prints a list of the EXE-file commands currently available in Prefix 6. The source file for this is an example of not only how to write a shell command, but how to use the ProDOS 16 GetDirEntry call. Very useful!
DUMP	Prints ("dumps") the contents of the specified file in both hex and ASCII representation. Any key starts and stops the display. If a S16 or EXE file, other interesting facts about the file are provided as well. Pressing any key starts and stops the listing. Use the Utilities menu Dump command for a scrollable file display.
HELP	If a name following HELP is given, that text file in Prefix 7 (HELP.FILES) is opened and the contents displayed. If no name follows HELP, it displays a list of all the available files in the Help directory.
MACRO.EXAMPLE	This installs a keyboard macro function in the Merlin editor. The source file is a fascinating example of a Merlin 16 utility. Macros can be predefined, or "recorded" on the fly. See the source file for directions on using the macros.
NEWPARMS	Re-loads the PARMS files and re-initializes the settings in Merlin without having to re-run Merlin. (Note: This does *not* re-initialize the Prefix settings).
PURGE.MEM	This lists the current handle information while you are in Merlin, and then purges all unlocked purgable memory blocks. It shows the owner of each block, and is very enlightening. It may also be useful if you're having trouble getting enough memory for an application to run properly!
SHOW.PARMS	Shows the current setting of the active Merlin 16+ parameters. Good for checking that the change you have made to the PARMS.S file has been put into effect when Merlin 16+ was started up. The PARMS settings can also be updated without re-starting Merlin 16+ by using the NEWPARMS command.

TEST	This is a test of the INPUT, SENDMSG, and COUT library routines and shows how a shell command can do text screen I/O.
TOOLHELP	This command typed by itself lists all the available on-line GS Tool reference topics. Typing, for example, TOOLHELP WINDOWS brings up the reference for the Windows toolset. Press a letter key to move to the first entry that starts with that letter, or press the arrow keys to move forward or back. When an alpha key is pressed, it searches from that point *forward*. Hold the Apple key down to search backwards. Pressing Escape takes you back to Merlin.
TYPE	The TYPE command simply does an ASCII screen dump of the specified file. Handy if you want to quickly see the lines of a text file. Use the Utilities menu Type command for a more full-featured operation.
WHERE	This is a *very* useful tool that prints out all the current memory blocks and their owners. An excellent starting point for a CDA or NDA that could be used while debugging a program to verify memory usage.

HELP.FILE Subdirectory:

This contains the text files used by the Merlin 16+ HELP command. Typing HELP alone lists all the files in this directory, which is normally set by Prefix 7. Typing HELP FILENAME prints the contents of the specified help file. There is also a subdirectory, TOOLDOC, in the HELP.FILE subdirectory, that has the files used by the TOOLHELP command.

Help files include:

APW	Converting source and object files between APW and Merlin.
APW.CMDS	A listing of the more-common APW commands and their Merlin equivalents.
ASM	How to assemble a source file.
ASSEMBLE	Alternate entry for HELP ASM.
BOOLEAN	List of Merlin Boolean (logical) operators (or, and, eor).
CLASS.1.CALLS	Listing of GS/OS Class 1 call parameters.
COMMANDS	A list of all the Merlin 16+ Command Box commands.
COMPRESS	A reference to the APW Compress utility.

CONVERT	Help on the Convert command, which converts an APW object file (for example. FILENAME.ROOT, to a Merlin-usable link file).
DISK.COMMANDS	List of legal disk commands (Main Menu or Command Box).
EXE	Information and requirements for Merlin-executed EXE files.
HELP	A description of Merlin's HELP files, and how to create them.
KEYMAC	Information on installing and using the keyboard macros program.
KIND	Listing of bit significance in KIND byte for OMF 1 and 2.
LINKERS	Description of the Merlin linkers, and Command File commands.
MACGEN	Description of the Macgen function.
MACROS	An example macro definition.
MLI.CALLS	Parameter information for ProDOS 16 Class 0 calls.
NEWPARMS	Description of the NEWPARMS command.
OPCODES	Addressing modes and available opcodes for 6502, 65C02, 65816.
PSEUDO.OPS	List of Merlin directives and pseudo-ops.
SHELL.COMMANDS	Description of Merlin shell commands.
SOURCEROR	Description of the Sourceror utility.
TOOLS	Description of the TOOLHELP command.
TRANSIENT.CMDS	Alternate name for a shell command.
UNDO	Description of Merlin's Undo function.
USER	Description of a USER function.

TOOL.EQUATES Subdirectory:

This contains a set of standard "Equates" files for the Apple IIGS Tools as defined by Apple Computer Co., Inc. They are used occasionally by sample programs like HodgePodge and others, but are not required for any particular program per se.

Merlin.Samples Files

/MERLIN.SAMPLES - Main Directory:

This disk contains the source files for many of the object files on the Merlin 16+ disk (/Merlin), and also other demonstration programs, and useful utilities. Following is a description of each subdirectory on this disk:

COMMAND.SOURCE Subdirectory:

This directory contains the source file for the shell or "transient" commands of Merlin 16+. These are excellent examples of not only how to write your own shell commands, but also how to perform other interesting and useful operations on the Apple IIGS in assembly language.

Each command has three files with it: A linker command file for creating the EXE object file, the source file, and the intermediate link file, for example: ASCII.CMD.S, ASCII.S, and ASCII.L.

SUBROUT.SOURCE Subdirectory:

These source files are example source files for the routines stored in the SUBROUT.LIB directory. The files create linkable subroutine files that can be called from within your own programs. See the source files themselves for a more detailed explanation of their function and use.

CALLDOS.S	Subroutine for making ProDOS 16 calls. It is used with the MLI16 macro within the UTIL.MACS file. Sample syntax: MLI16 close;CLSPARMS
COUT.S	This is a general-purpose ProDOS 16 text screen output routine.
INPUT.S	A general-purpose ProDOS 16 text screen keyboard input routine. It is a single-line input, with support for Delete key, insert/overstrike, default accept and more. Requires the COUT routine or equivalent.
MLI.CODES.S	This is an alternate set of ProDOS 16 equates. It can be used with the CALLDOS subroutine, or in other ways.
PRBL.S	Prints a number of blanks to the text screen in ProDOS 16. Requires the COUT routine or equivalent.

PRBYTE.S	Prints the 8-bit contents of the Accumulator in hex. Requires the COUT subroutine or equivalent.
PRDEC.S	Prints the decimal equivalent of a number in the Accumulator and optionally (8-bit mode) X register. Requires the COUT routine or equivalent.
RANDOM.S	Returns a random number in the Acc. Usable in 8- or 16-bit mode and is a very good number generator.
SENDMSG.S	Prints a string terminated by a null using the COUT routine or equivalent. The advantage of this over the tool call is that the character output routine can be customized to do special processing.

MISC.SOURCE Subdirctory:

These source files are example files for the USER and USR functions of Merlin 16+, and also a short demo of the RANDOM routine in the subroutine library.

RND.TEST	Shows how to generate a random number from assembly language using the RANDOM library routine.
USER.EXAMPLE	This is an example of the USER funtion. A USER file is usually a utility to create some sort of extension to the Merlin system. This example converts all the opcodes within a given range of the source code to upper case characters. Type -USER.EXAMPLE from the main menu to install the USER routine. Then type USER 12,34 (for example) from the Command Box to see how it works. The only advantage of the USER command over a shell command is that the USER routine is always in memory, whereas a shell command requires disk access for its execution.
USR.EXAMPLE.S	This is an example of the USR feature. USR is a user-definable pseudo-op in Merlin. This example shows different forms of USR, and custom math functions (floating point numbers) created by a USR pseudo-op.

GENERIC.SOURCE Subdirectory:

This contains two source files that can be combined and used as a generic starting point for creating your own desktop-style ProDOS 16 applications. The first file, Generic.Start.S, is based on a program listing that appeared in the "Sourceror's Apprentice", which is a monthly newsletter on assembly language programming on the Apple II. The Generic.Start source file is used to start up and shut down the Apple IIGS toolset in a standardized way. This source file can be used as a USE file at the beginning of your own programs to give you a "leg up" on the program-writing process.

The second file, Skeleton.S, builds on Generic.Start, and gives you an example desktop program, with a menu bar, that you can then add to to create your own applications.

Included in this program is a macro for easily adding and deleting menu items in a program. There is also a handy USR function that can be used while debugging a program. It displays the line number in the Merlin source file that corresponds to where the error in the running program occurred. Choose "Force an Error" (SHR or TEXT) from the Apple menu in the Skeleton program to see how this works.

GENERIC.START.S	Source file for generic startup for desktop-style application program. This program starts up the tools needed for windows and pull-down menus, and also sets up the Control-Y vector as described in Roger Wagner's "Apple IIGS: Machine Language for Beginners". It also includes the macros and subroutines for the ERR.USR debugging code.
ERR.USR.S	Source file for debugging aid. When added to your programs, this allows use of the USR function to create an error-handler much better than the Fatal Error handler seen in other sample source files. ERR.USR, when called, puts up a dialog box that displays the line number in the Merlin source file that corresponds to where the error occurred. See the program Skeleton for an example of this routine in use.
ERR.USR.L	REL file from ERR.USR.S.
SKELETON.S	This is a "skeleton" program that you can use as the starting point for your own programs. It uses the Generic.Start file as well. This uses the ERR.USR function, and demonstrates the Menu macros described under UTIL.MACS. Choose the "Force Error SHR" or "Force Error Text" in the Apple menu to see what the output looks like when an error occurs.

	IMPORTANT: The ERR.USR file *must* be installed for this program to assemble correctly! This is currently done automaticall by the Merlin PARMS file, but if you change the startup file, be sure to install ERR.USR yourself before assembling this program.
SKELETON.L	The REL file from SKELETON.S.
SKEL.MACS.S	Macro library used by SKELETON.S.

HODGEPODGE Subdirectory:

This subdirectory holds the Merlin version of the example program, HodgePodge, that is used in the Addison-Wesley book, "Programmer's Introduction to the Apple IIGS". Assemble and link the file (use Apple-6 to do it all at once) HP.ASM.S to create the final object file.

NIFTY.LIST Subdirectory:

Nifty List is a CDA by David Lyons that provides a number of very useful utility functions to the Apple IIGS programmer. To use this, copy the files NLIST.RWP and NLIST.DATA to the DESK.ACCS folder of the SYSTEM folder of your startup disk. To print the documentation, load the files NLIST.DESC and NLIST.REV into the Merlin editor, and print them with the "P" command from the Command Box.

Nifty List's primary function is to disassemble machine code in memory with the names of tool calls, ProDOS 8 and 16 calls displayed. There are also commands for visiting the Monitor, displaying toolset versio numbers and other toolset information, and displaying memory status information and other system values.

Nifty List is normally a "shareware" product, which means that you would send David Lyons a fee if you want to continue using the software. However, by special arrangement with Roger Wagner Publishing, Inc., the user fee for *this version only* of Nifty List has been paid for you, and you may use this version of Nifty List on your own system. If you wish to update to future versions of Nifty List, the NLIST.DESC file gives the address for David Lyons, and updates are available for the very reasonable fee of just $10.00 each.

HP.SUPERMACS Subdirectory:

This is nearly identical to the HODGEPODGE subdirectory, except that the source files have been further edited to use Merlin's Super Macros in place of the usual Apple macro calls. This has the effect of shortening the source listing, in addition to making it more readable. Also, the object file from this version is slighlty shorter since Merlin's PushLong macro uses the shorter PEA instruction, as compared tothe APW macro for PushLong, which does a LDA/PHA .

CONVERTER Subdirectory:

This contains the new version of the Converter program, which converts APW or ORCA/M source files to the Merlin format. In particular, you may want to look at the file FREND.S (for "Front End") that provides a useful interface for navigating subdirectories in ProDOS 8, and selecting files.

CONVERTER	This is an updated Converter program and its associated files. This replaces the Converter program and files on the Merlin 16 disk.

INTERNALS.CDA Subdirectory:

This is a special "bonus" CDA, written by Ken Kashmarek, one of the country's leading Apple II programmers. Install INTERNALS by copying the file INTERNALS to the DESK.ACCS folder of the SYSTEM folder of your startup disk. With this installed, you can interrupt a running program at any point by going to the Desk Accessory Menu, and choosing INTERNALS.

Internals displays, among other things, information about the current tool status, current interrupt vectors, information about QuickDraw, memory use, Battery RAM use, Loader pathnames and segment tables, CDA and NDA memory usage, and much more. This is another terrific aid when debugging programs!

New Additions to the Merlin 16+ and Merlin Samples Disks...

Even as this manual is going to press, additions continue to be made to the Merlin 16+ and Merlin Samples disks.

Located on one or the other of these disks (depending on where we could find some room!) is a file called READ.ME.S. Be sure to load and read this file to see what has been added to the Merlin system that is not described in this manual. We also recommend you print it out, and put a copy with your manual. Use the PRTR command and type P (for Print) instead of ASM to get a printed copy with formatted pages.

As always, we really do apprciate your support, and welcome any and all comments and suggestions you may have regarding the Merlin assembler. This latest version of Merlin 16+ includes over 100 new features compared to Merlin 16, many of which are the direct result of user requests and comments. Let us know what you want - we'll do our best to make it happen!

APW to MERLIN

The following text provides a discussion of the differences in source code between the APW and Merlin assemblers. This is provided to help you translate listings that are given in the APW format into a form usable by Merlin.

Although this list of differences may at first appear imposing, keep in mind that that majority of any given source listing is the same in either the APW or Merlin assemblers. That is, a LDA LABEL, or RTS is the same in either one. The differences are largely in what are called *assembler directives*. A directive is an instruction for the assembler itself. For example, in APW the directive TITLE is used to specify the title text to be used at the top of each page in a printout. In Merlin, the directive used is TTL. Knowing this, changing an APW source listing using TITLE 'My Listing' is easy - just make it TTL 'My Listing'.

The other area of difference is in the specifying of data within the source listing. For example, to define a string in APW, they would use:

```
MSSG    DC      C'This is a string.'
```

In Merlin, this would appear as:

```
MSSG    ASC     'This is a string.'
```

APW uses the DC (for "Define Constant") assembler code, followed by a letter ("C" for characters, "H" for hex, etc). Merlin has a specific code for each data form.

All of the assembler differences are discussed in the next few pages. It will not be necessary for you to use all the information to convert a single listing. Rather, the following information should be used like a reference. The best way to use it is to just scan all the way through once to get an overview of the differences, and to then just go back to a particular section as you need it.

Two Ways to Convert an APW-style listing to the Merlin Format:

We'll suppose that you have a listing for a program from a magazine, book, or even on disk that is currently in the APW format (i.e., has directives like START, DC, etc.). There are two ways to convert this to a format more usable by Merlin:

1) Converter. If the source file is already on disk, the easiest way is to just use the Convert utility provided on the Merlin 16 disk. This will automatically convert 90-100% of the source listing to a form compatible with Merlin.

If the listing is in a book or magazine, you might still want to type the listing in exactly as shown (you can do this in the Merlin editor), save the source file to disk, and then use the Converter program to convert the entire program at once. If you are new to assembly language programs, this might also be easier for your first few programs.

Once converted, you can then compare the output from the Converter program to the original APW-style source listing, and see what has changed. Compare this also to the discussion on the following pages. After conversion, some parts of the listing may require further editing before final assembly. The text of this section will help explain any remaining changes that will need to be made.

Once familiar with the individual differences between APW and Merlin, you can then use method two:

2) Manual Changes. Once you recognize the Merlin equivalents of APW statements like DC, MCOPY, etc., (and there really aren't that many to deal with), you can just manually make the changes as you enter listings presented in books and magazines.

ASSEMBLER DATA SYNTAX

Both assemblers use the (Parentheses) to enclose 2-byte operands, and [Brackets] to enclose 3-byte operands for indirect addressing. Immediate mode expressions use the # symbol. This prefix can be modified:

APW	Merlin	Meaning
#Label	#Label	Signifies low-byte (or word) of value of Label.
#<Label	#<Label	" "
#>Label	#>Label	Signifies high-byte of value of Label.
#/Label	#/Label	" "
#Label1-8	#>Label	" "
#^Label	#^Label	Signifies the high-word (or bank-byte) of the value of Label.
#Label1-16	#^Label	" "

Whether a byte or word value is used depends on the current register length in the assembler. In APW, this is controlled by the directives LONGA and LONGI (see description of these later). In Merlin, the MX directive can be used to set the register length, although Merlin does automatically keep the correct register length after REP, SEP and XCE instructions.

When dealing with an address directly (as opposed to an immediate, or constant, value, APW also allows the user to force short or long addressing.

APW	Merlin	Meaning
<Label	Label	Forces direct-page (zero-page) addressing in APW.

Since Merlin does this automatically when the label equates to less than $100, this is not needed. However, if you want to force direct-page addressing, regardless of the value of Label (that is, you expect Label to be greater than $FF), you can force a one-byte value by using the expression Label&$FF.

APW	Merlin	Meaning
|Label	|Label	Forces two-byte addressing. In Merlin, this is the norm, so the vertical bar is probably not needed. The bar can also be used to force two-byte addressing in those cases where Label evaluates to less than $100.
>Label	>Label	Forces long addressing.

For forced long addressing, there is the alternate form in Merlin of Opcode+"L", for example, LDAL Label. The opcodes AND, ADC, CMP, EOR, LDA, ORA, SBC, and STA all have this alternate way of getting long addressing.

APW	Merlin	Meaning
ASL A	ASL	The Accumulator mode of addessing common to APW (such as ASL A) is different in Merlin. Merlin doesn't require that suffix. ASL alone is sufficient.

EXPRESSIONS

Merlin supports the standard for arithmatic expressions: +, -, / (Division),* (Multiplication). And the standard three logical operators: ! (Exclusive OR), . (OR), & (AND).

APW also supports +, -, /, *. The logical operators are .EOR., .OR., .NOT., or .AND. The vertical bar (or !) at the end of a label or expression is a bit shift operator, and is usually seen in the form:

```
LDA LABEL|-16
```

which is an instruction to load the bank byte of Label. In Merlin, this is equivalent to

```
LDA ^Label
```

APW supports the logical operators <>, =, <,>,<=,>=. These functions are directly supported in Merlin 16+, but can easily performed in other versions of Merlin by using other operators, as described in the Merlin 8/16 manual.

Both Merlin and APW support all three number formats: Binary, Hex, and Decimal. For **Binary**, the symbol '%' preceeds a number composed of 0's and 1's. %10001000. For **Hex**, the number (any combination of digits 0-9 and letters A-F) is preceded by '$'. $FA90.

Decimal numbers (composed of any digits, and no letters) are the default number type. 1234. APW also supports one more, exceedingly rare, number type: **Octal** (any combination of digits from 0-7), and is preceded by '@'.

APW

```
dc  b'10001101'          ; binary number form
dc  h'FF'                ; hex number form
dc  d'25'                ; decimal number form
```

Merlin

```
db   %10001101           ; binary number
hex  FF                  ; hex number form
db   25                  ; decimal number form
```

APW also has a syntax where a number in front of the number type character can be used as a repeat count. For example:

APW	Merlin	Meaning
dc 5h'FF'	HEX FFFFFFFF	Repeat $FF five times.

Hi-bit ASCII Control:

APW uses either single quotes (') or double qoutes (") to define a character constant. Any character can be within these delimiters. Merlin also uses these qoutes, but Merlin uses these delimiters to turn 'off' and "on" high-bit ASCII. So how does APW set and clear hi bits in data statements? APW has the **MSB ON** (or **MSB OFF**) command to set and clear the high bit ASCII. The CONVERT utility on the Merlin disk doesn't specifically handle the MSB codes in an APW listing, although MSB OFF is definitely the preferred mode for virtually all ProDOS 16 programs. This means the default conversion which leaves all ASC statements in Merlin with single quote delimiters will probably work just fine.

If the program you're converting does use MSB ON, just use double-quotes in Merlin from that point on (or until the next MSB OFF) for all your ASC statements.

Comments:

Both assemblers allow **comments**. APW allows three characters as comment delimiters- !, *, or ;. Merlin only supports the last two. The ; creates a comment field on the right, usually after the opcode/operand part of the line. The * is used at the start of the line, to set aside all of the upcoming line as comment space.

ASSEMBLER CONTROL

APW has a number of special commands that are only needed to define the code segments. Blocks of data are defined as such with the **DATA** and **END** commands. **START** is placed at the beginning of a code segment, while **END** defines the end. These are not required in Merlin, and all of these can be replaced, in Merlin, with blank space in the opcode field.

APW cannot tolerate a blank space following a label, and so requires the do-nothing pseudo-op **ANOP**. Merlin has no such restriction, so you can replace ANOP with a blank space.

USING: To quote the APW manual: "The USING directive makes local data segment labels visible to the code segment in which the USING directive appears." Normally, in APW, all the labels between a START and END (a segment) are local in that they are "known" only to that segment, and not to other parts of the source listing. The directive USING DIALOG3 would make all the labels in the segment DIALOG3 available in the section that had the USING directive.

There is no exact equivalent of this data structure in the Merlin assembler, but in practice, few listings make such heavy use of the same label that the local vs. global status of that label matters. In cases where the same label, like Done, is used in a lot of segments, simply converting the duplicate labels to Merlin-style local labels, for example, :Done, solves the problem. See the discussion of dealing with duplicate labels in the text describing the conversion of the APW form of the HodgePodge program, in the Merlin 16 Supplement booklet, that is included in the Merlin 8/16 package.

In APW, **PRIVATE** and **PRIVDATA** commands force all labels to be local within those areas up to the next **END** statement. The same effect can be achieved in Merlin one of two ways: 1) Use local labels (:Label), or, 2) Break these into separate segments, and link them all together. Within a linkfile, all labels are local to that file unless specified as an ENT label. Files that use those labels would declare them as externals with Merlin's EXT command. See the Merlin manual and explanation of the Linkers for more details.

DEFINED DATA

APW has only two types of data statements: **DC** (Define Constant) and **DS** (Define Space). For the DC command, each data number or character is preceded by a data type. In APW, this can be the letters a,b,c,d,e,f,h,i,r, or s, and have the following meanings:

APW

```
dc  a1'Label'    ; one-byte "address".
dc  a2'Label'    ; two-byte "address".
dc  a3'Label'    ; three-byte "address".
dc  a4'Label'    ; four-byte "address".
dc  b'10001101'  ; binary number form.
```

```
dc  c'test'      ; ASCII string.
dc  d'2.5'       ; 8-byte floating pt number.
dc  e'2.5'       ; 10-byte extended fp num. (SANE).
dc  f'2.5'       ; Applesoft-compatible fp number.
dc  h'ff'        ; hex number form.
dc  i1'Label'    ; 1-byte value.
dc  i'Label'     ; 2-byte value.
dc  i2'Label'    ; 2-byte value.
dc  i3'Label'    ; 3-byte value.
dc  i4'Label'    ; 4-byte value.
dc  s2'Label'    ; 2-byte space to hold a value.
```

Merlin has the data commands HEX, DFB, ASC, STR, INV, FLS, DCI, REV, etc., which are different, but functionally equivalent, forms of the APW data statements:

Merlin

```
db  Label        ; one-byte "address".
da  Label        ; two-byte "address".
adr Label        ; three-byte "address".
adrl Label       ; four-byte "address".
db  %10001101    ; binary number form (da,adr,adrl ok too!).
asc 'test'       ; ASCII string.
                 ; 8-byte floating pt number not avail.
flo 2.5          ; 10-byte extended fp num. (SANE).
                 ; Applesoft fp number not avail.
hex $ff          ; hex number form.
db  Label        ; 1-byte value.
da  Label        ; 2-byte value.
adr Label        ; 3-byte value.
adrl Label'      ; 4-byte value.
ds  2            ; 2-byte space to hold a value.
```

In addition, Merlin offers additional data-defining codes including STR, STRL, DW, DFB, INV, FLS, DCI and REV for greatest flexibility. See your Merlin manual for additional details.

The DS is used the same in both.

Another APW number format is **IEEE**, which generates IEEE format floating point numbers. If ON, this generates SANE type numbers. If OFF, this generates Applesoft-compatible numbers. Merlin 16+ has the FLO data command, which allows data in the form of the SANE floating point number. There is no Applesoft equivilent.

Related to IEEE is the **NUMSEX** command. This reverses the order of how the floating point numbers are stored. There Since the SANE convention is least significant, most significant, this is rarely needed.

ASSEMBLER DIRECTIVES

Within APW, **EQU** is a local label known only to the segment it appears in. **GEQU** is used to make a label known to all other segments. In Merlin, all EQU's within a source file and any USE or PUT files are known to all other parts of the listing, unless a local label (example: :Label) is specifically used. For the most part, you should be able to just use EQU in your Merlin listing where the APW listing uses either EQU or GEQU.

When using the Linker, you can produce the equivalent logical structure of APW's use of EQU and GEQU by your use of the EXT and ENT commands, although it is unlikely you will ever have to deal with a conversion that is this complex.

APW has four commands to force object code to behave in special ways. **ORG** designates the fixed location for the code. **OBJ** also designates a fixed location for the code, with the difference that it doesn't force the code to load there. **OBJEND** ends the OBJ segment of code. **ALIGN** forces your code align to a page boundry.

Merlin uses ORG exactly the same way, but all the rest are different. ORG with no address acts as a ORGEND, restoring the ORG to what it was before the first ORG. Thus, there's no need for OBJ. And ALIGN, in Merlin, can be simulated by the DS \ command, which pads the end of the file to the next page boundry. You can also use the ALI command in the Merlin linker for ProDOS 16 programs.

Both Merlin and APW come with the ability to turn on and off certain commands. Both generally use the syntax CMD + " OFF" or CMD + " ON". APW has the additional ability to read the settings of these commands. The sequence "S:" + CMD" returns either a 1 (meaning it is ON) or a 0. However, these are generally format-oriented commands, and are not required as part of the conversion process for a source file.

APW has three commands to control how the assembly is to be done. **COPY** (transfer processing to that file and come back), **KEEP** (Save assembled code on disk), and **APPEND** (transfer processing to that file but don't come back). Merlin also has three commands, although they aren't exactly the same. PUT is the equivilant of COPY, and KEEP is the equivilant of SAV or DSK. There is no equivilent of APPEND, although it can be simulated by performing multiple PUTs.

APW	Merlin	Meaning
COPY FILENAME	PUT FILENAME	Include source file.
MCOPY FILENAME	USE FILENAME	Bring in macros.
KEEP FILENAME	DSK FILENAME.L	Save object file (.L is dropped in final file).
APPEND FILENAME		(Use multiple PUTs).

MCOPY and **MLOAD** tell APW which macro file is needed to get the macro you're specifically using. Merlin uses the USE command to add Macros to source code. APW also has another command, **MDROP**, to remove that macro list from the current definitions.

APW has the **KIND** command, which is used to set the kind field of the object segment header. Merlin also has this command, but it's used in the linker command files and it's called KND.

APW has the **LONGA** and **LONGI** commands which set the length of the Accumulator and the Index register. Merlin controls the same effect with MX%, and macros have been included in the UTIL.MACS file on the Merlin disk to replace the APW functions.

APW	Merlin	Meaning
LONGA ON	LONGA 1	16-bit Acc.
LONGA OFF	LONGA 0	8-bit Acc.
LONGI ON	LONGI 1	16-bit index regs.
LONGI OFF	LONGI 0	8-bit index regs.

There is also quite a compliment of other macros in the UTIL.MACS file that don't require a numeric operand. For example, LONGM, SHORTA, SHORTAX, etc. You can also use the MX instruction in Merlin to set the register length directly.

MERR is an APW command that has no equivalent in Merlin. It allows you to set the maximum number of errors you'll accept before it stops assembling. Then there is **ERR**, a another error pseudo-op, which allows you to turn on and off the error messages from the assembler.

RENAME is another directive not in Merlin. It allows you to change the name of an existing opcode so you can use another that might have the same name. Interesting.

Then there is **CODECHK** (which tells linker to check Jump instructions to make sure they are within the current load segment), **DYNCHK** (which checks to make sure you only JSL to dynamic segments), and **DATACHK**, (which checks the code for 16 bit data references outside the current load segment). These are not available in Merlin, although Merlin's linker will report a variety of errors during the link process.

DIRECT, followed by an address, sets up error checking and handling for direct page usage. It does this by adding whatever value is at the address to the zero-page address described. If this is OFF, then code errors when no direct-page addressing is available. In Merlin, when you specify a zero-page adress, Merlin doesn't argue with you about it.

Another APW assembler option in APW is **CASE**, which specifies case sensitivity in labels. There's even a special CASE called **OBJCASE** which only activates case sensitivity within OBJ sections of code. Merlin uses CAS SE (for cases sensitive) and CAS IN (for insensitive). You can also set your startup preference for case sensitivity in Merlin using the approprate bit in the PARMS file.

APW LISTING CONTROL

INSTIME ("instruction time") does the task within APW of displaying each action's cycle times. Merlin also has that ability with the CYC command.

TRACE ON activates the displaying of lines of conditional code. **TRACE OFF** (the norm) only displays the resulting lines of code. Merlin uses the EXP (ON, OFF, or ONLY) to achieve the same effect.

SETCOM lets APW users set the horizontal position of the comment field. In Merlin, all the Tabs are individually setable within the PARMS file, or with the Command Box TABS command.

The commands **65C02** and **65816** are used by APW to enable the 65C02 and 65816 opcodes. Merlin 16+ defaults to having all opcodes enabled. XC OFF restricts legal opcodes to only those of the 6502. XC OFF, followed by XC enables just the 65C02 opcodes. The Converter changes 65816 to M65816, which is a simple macro in the UTIL.MACS file that enables 65816 opcodes with the XC instructions, regardless of your PARMS settings.

ABSADDR makes APW listings nicer looking by adding another column - the absolute address is displayed for the first byte of each line. Merlin does that already, using a default value of $8000 instead of the $0000 of APW.

APW has a couple of commands that help with listing programs. **EJECT** is APW's command for advancing to the next page. Merlin uses PAG. **EXPAND** turns on and off the listing of data created by the DC commands. Merlin uses TR ON to truncate data (any kind of data) to only the first three bytes. That makes a nice listing, too.

LIST, in APW, turns on and off the ability to list. Merlin has the LST ON and LST OFF commands that do exactly the same thing, plus a variety of other LST commands for greater flexibility. **SYMBOL** enables or disables symbol table production. Merlin can also control this with the LST OFF or ON at the end of the source listing.

PRINTER, within an APW listing, sends the output to the printer. Merlin use the PRTR command from the Command Box, or Apple-1 when doing an assembly.

TITLE generates a header on each subsequent page in the header up there right along with the page number. Merlin has the TTL command, to initialize the title.

That's it! This should be sufficient to provide you with the information you need when converting any APW source file. However, if you do have any problems not addressed by this documentation, feel free to call our technical support department for additional assistance.