Merlin™ to ORCA/M™

Source Code Translator
for ORCA/M
or the Apple IIGS Programmer's Workshop

Includes Source Code

By Barbara Allred

Limited Warranty - Subject to the below stated limitations, Byte Works Inc. hereby warrants that the program contained in this unit will load and run on the standard manufacturer's configuration for the computer listed for a period of ninety (90) days from date of purchase. Except for such warranty, this product is supplied on an "as is" basis without warranty as to merchantability or its fitness for any particular purpose. The limits of warranty extend only to the original purchaser.

Neither Byte Works, Inc. nor the authors of this program am liable or responsible to the purchaser and/or user for loss or damage caused, or alleged to be caused, directly or indirectly by this software and its attendant documentation, including (but not limited to) interniption of service, loss of business, or anticipatory profits.

To obtain the warranty offered, the enqlosed purchaser registration card must be completed and returned to the Byte Works Inc. within ten (10) days of purchase.

This product is sold for use on a single computer at at a single location. Contact the publisher for information regarding licensing for use at multiple-workstation or multiple-computer installations.

ORCA/M is a trademark of the Byte Works, Inc.

Table of Contents

Chapter 1 - Introduction

What Merlin-to-ORCA Is

The Merlin-to-ORCA Source Code Translator takes as input a Merlin assembly-language source file and translates the input to an ORCA assembly-language program. You should be aware that the translation is close, but not perfect. Lines which cannot be translated are clearly marked (as comments), so that you may change them by hand. The translator does not support DOS. If you will be converting Merlin DOS source files, you should first run your Merlin programs through the DOS-to-ProDOS CONVERT utility supplied with your ProDOS system disk, then pass the converted ProDOS Merlin source file through the Merlin-to-ORCA translator.

Throughout this manual, comments applying to ORCA/M 1.0 apply equally to the Apple IIGS Programmer's Workshop (APW).

There are four main differences between the Merlin and ORCA assembler packages. First, the linkers are different. At this point in time, the ORCA/M 4.1 linker does not produce REL files, nor does the assembler save portions of the object file to different names. The problem with REL files is moot on the Apple IIGS, as that is a "relocating" machine, and the ORCA/M 1.0 linker produces only relocatable executable output. (ORCA/M 1.0 provides a binary-file maker, installed as a utility named MAKEBIN, for programmers developing software on the Apple IIGS that is targeted for the eight-bit Apple // computers.)

Second, the macro languages used by the two assemblers are distinct. Macro translation is discussed at the end of Chapter Three.

Third, the directives defined for each assembler are unique. Some directives are translated directly, others use ORCA macros to accomplish translation, and some require extensive analysis in order to perform partial or full translation. A few of the directives are simply discarded. These either relate to the Merlin linker (EXT, REL), or serve some function which can be duplicated by the ORCA command processor (DAT, TYP), or simply cannot be translated (CHK, OBJ, PAU, SW, USR).

Finally, expression syntax differs between the two assemblers. The translator contains an expression evaluator which transforms Merlin expressions to ORCA expressions.

What You Should Have Received

The Merlin-to-ORCA Translator package consists of the following: one 5.25" diskette containing both the 8-bit and 16-bit versions of the program and source code, this manual explaining the

program, and a warranty registration card. Returning the warranty registration card allows you to receive update information and the Byte Works' newsletter, *ORCA NEWS.*

Side one of the disk contains the following subdirectories:

```
EXECUTE
HELP
MERLN.GS
MERLN.II
```

Side two of the disk contains the single subdirectory named COMMON.

Within the subdirectory EXECUTE are the executable files named MERLN.GS and MERLN.II. MERLN.GS is the translator which runs on the Apple IIGS computer. MERLN.II is the version which runs on the Apple][,][+, //e, or //e. The file named MERLIN.MACROS contains special ORCA macros which are used as a final step in the translation process. This file should be included when running a translated source program under ORCA/M, as explained in Chapter Two of this manual.

HELP contains two text files which briefly describe how to execute the two translators. They are provided in case you would like to install Merlin-to-ORCA as a utility, as detailed in Chapter Two.

The ProDOS 16 specific portions of the code are found in the subdirectory named MERLN.GS, in the files named MERLN.GS.FILE and MERLN.GS.MACROS. The ProDOS 8 specific source code is located in the subdirectory named MERLN.II, in the files named MERLN.H.FILE, MERLN.II.FILE2, and MERLN.II.MACROS. The source code contained in the .FILE modules includes the global data definitions, and the memory management, file handling, and terminal error routines. The MACROS files contain the macros used in writing the translator.

The source code which is shared by both versions of the program is located in the subdirectory named COMMON. All of the files in this directory are named MERLJN.x, where x is a letter specifying the first character of the first subroutine in that file.

The Byte Works' intent in releasing the source code to the Merlin-to-ORCA Translator is to allow you to make any changes to the program you want. The program is for your personal use only, however. Merlin-to-ORCA is copyrighted, and the Byte Works retains all rights to it.

Overview of the Manual

Chapter Two explains how to install and run the translator. Chapter Three explains the differences between the Merlin and ORCA assemblers, concentrating on such areas as labels, instructions, directives, macros, operands, and comments. Chapter Four discusses each Merlin directive, showing how the program translates each of them. It is not necessary that you read every word of the manual in order to use the translator effectively. If you are already somewhat

familiar with ORCA, then all you may need to read are the directions for running Merlin-to-ORCA, given in Chapter Two. If ORCA is somewhat mysterious to you, you may find Chapter Three helpful. Chapter Four, covering the directives, has been written so that you need only refer to those Merlin directives whose translation you are particularly interested in.

Chapter 2 - How to Use Merlin-to-ORCA

Installation

The Merlin-to-ORCA translator runs in the ORCA/M environment, and relies upon the ORCA shell and system libraries for correct execution. The translator can be installed as a utility, or executed as a stand-alone program.

> *PLEASE!!! Make a back-up copy of Merlin-to-ORCA and store the original in a safe place. Protect your investment !!!*

Installing As A Utility Under ORCA/M 1.0

To install Merfin-to-ORCA as a utility, copy the executable file named MERLN.GS, located in the translator subdirectory EXECUTE/, to the UTILITIES directory of your ORCA/M. system.. Now copy the text file named MERLN.GS, located in the translator's subdirectory HELP/, to the ORCA UTILITIES/HELP prefix. This will allow on-line help, and can be accessed by entering the command

```
HELP MERLN.GS
```

The final step in installing Merlin-to-ORCA as a utility is to modify the system command table. Enter the editor with the file named SYSCMND, located in the SYSTEM prefix of ORCA. Insert a blank line for the new utility in the correct place in the table. (The system command names are generally stored in sorted increasing alphabetical order.) Beginning in column one of the blank line, enter the name of the executable file you just copied to UTILITIES (MERLN.GS). Move to the next column in the table, on this same line, and enter a `U' for utility. An optional comment may be entered in the third column, describing the command's function.

Exit the editor, saving the modified SYSCMND file. To inform the system about its new command, use the COMMANDS command, described in your ORCA manual:

```
COMMMDS <pathname of SYSCMND>
```

or simply reboot ORCAJM.

Installing As A Utility Under ORCA/M 4.1

To install Merlin-to-ORCA as a utility, copy the executable file named MERLN.II, located in the translator subdirectory EXECUTE/, to the UTILITIES directory of your ORCA/M system. Now copy the text file named MERLN.II, located in the translator's subdirectory HELP/, to the ORCA

UTTLITIES/HELP prefix. This will allow on-line help, and can be accessed by entering the command

```
HELP MERLN.II
```

The final step in installing Merfin-to-ORCA as a utility is to modify the system command table. Enter the editor with the file named SYSCMND, located in the SYSTEM prefix of ORCA. Insert a blank line for the new utility in the correct place in the table. (The system command names are generally stored in sorted increasing alphabetical order.) Beginning in column one of the blank line, enter the name of the executable file you just copied to UTILITIES (MERLN.II). Move to the next column in the table, on this same line, and enter a 'U' for utility. An optional comment may be entered in the third column, describing the command's function.

Exit the editor, saving the modified SYSCMND file. To inform the system of its new command, use the COMMANDS command, described in your ORCA manual:

```
COMMANDS <pathname of SYSCMND>
```

or simply reboot ORCA/M.

Installing As A Utility Under ORCA/M 4.0

To install Merlin-to-ORCA as a utility, copy the executable file named MERLN.II, located in the translator subdirectory EXECUTE/, to the UTILITIES directory of your ORCA/M system. Now copy the text file named MIERLN.II, located in the translator's subdirectory HELP/, to the ORCA UTILITIES/HELP prefix. This will allow on-fine help, and can be accessed by entering the command

```
HELP MERLN.II
```

The final step in installing Merlin-to-ORCA as a utility is to modify the system configuration. Do this by calling the ORCA utility named SYSGEN:

```
SYSGEN
```

You will see a menu screen displayed. Select 'S' for system characteristics to change. You will now see a SYSTEMS PREFIXES menu screen. Select 'U' for utility and then enter a carriage return, followed by pressing the escape (ESC) key. This sequence of keystrokes returns you to the main screen. Enter 'Q' for quit. SYSGEN will ask if you want to make the change permanent. Enter 'Y' for yes, then the pathname of ORCA.HOST, which is typically /ORCA.

Running the Translator as a Utility

To explain how to use the translator, we will walk through a simple example. Let's assume that you have a Merlin source file named PROG1, and that the program is located on a volume

named /MY.PROGS, in a subdirectory named ASM. The pathname for PROG1 is then /MY.PROGS/ASM/PROG1. Let's also assume that you have installed Merhn-to-ORCA as a utility.

The translator program is executed by entering the name of its executable file installed in the UTILITIES prefix. The input file name is entered next on the command line. For now, we will assume that we are in the subdirectory named /MY. PROGS/ASM/. The command line would then be:

```
MERLN.GS    PROG1  (ProDOS 16 version)
MERLN.II    PROG1  (ProDOS 8 version)
```

The translation Of PROG1 will be written to standard output, which is defined as the monitor screen by default. We will probably want the translation to be placed in a file, however. We can redirect the output to a file by using the special symbol '>'. Let's assume that we want the output to go to a file named PROG1.O. The command line would then be:

```
MERLN.GS    PROG1  > PROG1.O        (ProDOS 16 version)
MERLN.II    PROG1  > PROG1.O        (ProDOS 8 version)
```

If we want to send the output to the printer instead of a file, we can redirect it by using the special device name.PRINTER:

```
MERLN.GS    PROG1  > .PRINTER       (ProDOS 16 version)
MERLN.II    PROG1  > .PRINTER       (ProDOS 8 version)
```

Input can also be redirected by using the special symbol '<'. In the following examples, the input still comes from the file PROG1, as above. We could then enter:

```
MERLN.GS    < PROG1                 send output to monitor screen
MERLN.II    < PROG1

MERLN.GS    < PROG1  > PROG1.0      send output to PROG1.O
MERLN.II    < PROG1  > PROG1.0

MERLN.GS    < PROG1  > .PRINTER     send output to the printer
MERLN.II    < PROG1  > .PRINTER
```

What if we're not in the same subdirectory as the program we want to translate? We can access the file by using its pathname:

```
MERLN.GS   /MYPROGS/ASM/PROG1
MERLN.II   /MYPROGS/ASM/PROG1
MERLN.GS   < /MYPROGS/ASM/PROG1
MERLN.II   < /MYPROGS/ASM/PROG1
MERLN.GS   /MYPROG2/ASM/PROG1   > .PRINTER
MERLN.II   /MYPROG2/ASM/PROG1   > FILE3
MERLN.GS   < /ASM/PROG1   > /ORCAPROGS/PROJ5/FILE56
```

All of the above commands are valid, based on our hypothetical disk. Note that the last example shows that you can redirect output to a file located in another subdirectory, or even on another volume.

Running the Translator as a Stand-Alone Program

If you do not want to install Merlin-to-ORCA as a utility, you can run the translator as a standalone program. ORCA executable programs (EXE files under ProDOS 16 or BIN files under ProDOS 8) are run by simply typing the pathname of the executable file. If you are in the directory where the executable file resides, then the filename by itself is sufficient to inform ORCA of the program you want to run. If you are in a directory other than the one where the executable file resides, then you can run it from the current directory by entering a pathname. The discussion above about supplying an input filename and redirection still apply. For example, suppose MERLN.H is in your current directory and you want to translate a file named MYFILE. The command fine to accomplish this is would be any of the following:

```
MERLN.II    MYFILE
MERLN.II    < MYFILE
```

Now suppose that MERLN.II is in a directory named AID, on a volume named /PROFILE. You could translate MYFILE, which is in your current prefix by any of the following command lines:

```
/PROFILE/AID/MERLN.II    MYFILE    > .PRINTER
/PROFILE/AID/MERLN.II    < MYFILE
/PROFILE/AID/MERLN.II    MYFILE    > OUTFILE
/PROFILE/AID/MERLN.II    < MYFILE    > /PROGS/P2/OSSS/OUTFILE3
```

Chapter 3 - Differences Between ORCA/M and Merlin Assembly Language Programs

Assembly-language source lines consist of the following fields:

```
LABEL     OPCODE    OPERAND             COMMENT
```

The opcode is an instruction defined in the target processor's instruction set, an assembler directive, or a macro call. Well-defined standards for the components of a language are generally prescribed for high-level language compilers, but this is not the case with assemblers. While ORCA and Merlin both provide the accepted instruction/operand syntax for the 65xx family of processors, they also uniquely define labels, directives, macros, and comments.

Structure

One of the most important differences between ORCA and Merlin is the general structure of an assembly-language program. Under ORCA/M 4.1, lines may be as long as 80 characters. The source code fields can be any length, as long as the combined field length does not exceed 80 characters. Note that labels are only significant through the first ten characters. Under the Apple IIGS version of ORCA/M, lines can be up to 255 characters in length. Labels can be any size, and all characters are significant, but an opcode must be included on a line containing a label. The translator will replace Merlin lines containing only a label with

```
LABEL     ANOP
```

where ANOP is the ORCA assembler directive "do nothing." In other words, ANOP can be used to define an address for a label.

Comments

ORCA comment lines are either completely blank, or begin with one of * ; ! in column one. Comments included at the end of a source line may begin one blank space after the last field of a line (either the opcode or the operand), or in column 4 1. They need not be prefixed with any special character, such as the ';'character required by Merlin. Care should be taken when commenting lines containing an optional operand. If the comment is not placed after column 40, the ORCA assembler will assume that the comment is the operand, resulting in an assembly-time error.

The translator replaces Merlin source lines which contain blanks followed by a comment denoted with ; with a semi-colon in column one, followed by the comment printed starting in column 41:

Merlin	ORCA
`; here is a comment`	`;  here is a comment`

Merlin comment lines which are all blanks or begin with an asterisk in column one are copied to the ORCA source file as is.

Segmentation

ORCA is one of the very few assemblers that provides true segmentation. That is, an assembly language program can be broken into segments, with each segment fulfilling a specific function. (This is very similar to subroutines found in high-level languages.) Segments are typically executed from other segments via the jump-to-subroutine instructions JSR and JSL. The labels within each segment are private to that segment, so that the same label name can be used in every segment of the program, if this is desired. An ORCA segment has the following structure

```
LABEL    START                    LABEL    DATA
         . . .        or                  . . .
         END                      END
```

where the first type of segment is used for code segments, and the other is used to define a block of data definitions. ORCA requires that a valid assembly-language program contain at least one segmenL

Only the labels on the START and DATA directives are global, like a procedure names in Pascal or C. They should be unique within the program. (Labels defined using the GEQU and ENTRY directives are also global; these are described in your ORCA/M manual.)

In creating an ORCA source file from a Merlin program, the translator places the following lines at the beginning of the output:

```
          MCOPY MERLIN.MACROS
          MCOPY MY.MACROS
TRANSLATE START
          LCLA &LUP
```

and, at the end of translation

```
          END
```

The file MERLIN.MACROS contains the translation of a few Merlin directives. These are discussed in the next chapter. The file MY.MACROS contains the translated macros you have defined in your Merlin source program. Macros are discussed at the end of this chapter. The line LCIA &LUP is for use with Merlin's LUP directive; it is covered in Chapter Four.

Labels

Merlin provides labels with limited scope - something that is done more effectively by ORCA through segmentation. A Merlin label beginning with the character ':' is defined as a local label and "attached" to the closest previous global label. The same local label can then be reused following a new global label.

ORCA and Merlin both allow the definition of variables. Merlin variables begin with the']' character, provide multi-purpose association with data, and can be redefined as often as desired. ORCA variables begin with the character '&' , are typed upon declaration (that is, there are three distinct variable types: arithmetic, boolean, and character), and can only be defined once for the segment or macm in which they appear.

The syntax of label names differs on the two systems. The maximum length of labels under Merlin is thirteen characters, while under ORCA/M 4.1 it is ten (significant) characters, and for ORCA/M 1.0 it is 255 characters. The syntax of a valid ORCA label is based upon. that for standard identifiers in high-level languages: a letter, optionally followed by zero or more letters or digits, up to the maximum presrcibed. An extension to the standard allows ORCA labels to start with or include the '~' and the '_' characters.

Merlin labels, on the other hand, are extremely free-format. They are required to begin with a character at least as great in ASCII value as ':', and cannot contain periods or characters less in value than an ASCII zero.

A major component, then, of Merlin-to-ORCA is label translation. One of the first things the translator does, in its initialization routine called PQT, is to allocate memory for a label table. All non-ORCA labels are stored in this table. Under ProDOS 8, which uses a simple bit-map technique for memory management, the translator allocates as large a buffer for the table as it can. When the buffer is exhausted, the program aborts. Under ProDOS 16, which contains a more sophisticated memory manager, the program asks ProDOS for a table buffer of initial size 1024 bytes. It then asks ProDOS to grow this buffer as needed. If the buffer reaches a point where it can no longer be grown, the program aborts.

Whenever the translator encounters a label, it first determines if its syntax is acceptable to ORCA. If it is not, it then searches the label table for a match. If a match occurs, the label translation routine, either TRANS_LABEL or TRANS_VAR, returns the unique ORCA-syntax label defined for that Merlin labef. lf no match occurred, the new label is added to the end of the table, and a unique translation is derived for the label. With the exception of variables, as discussed later in this section, Merlin labels are stored into the table in the foRowing manner: The first byte of the label's field is the size of the label - the number of characters of which it is comprised. The rest of the field contains the individual characters of the Merlin label. IIe end of the table is marked with a zero in the label-size field.

Merlin global labels are rewritten in the ORCA source file as the seven-byte string 'SYS'+ 'xxxx' where xxxx is an ASCII number corresponding to the index of the label in the table. For example, if the second non-ORCA label encountered in the Merlin source file was @AB, then the translator would return SYS0001. Note that zero is the first index into the table.

Local labels are transformed into the eight-byte string 'L' + 'xxxx' + 'yyy', where xxxx is an ASCII number corresponding to the index of the label in the table, and yyy is the current ASCII global label counter. The translator increments the global label counter with each occurrence of a global label found in the label field of an input source line. A unique local label is stored only once into the table. It is redefined during translation by appending the current global counter to it.

Merlin variables are translated into the eight-byte string 'V' + 'xxxx' + 'zzz' , where xxxx is an ASCII number corresponding to the index of the label in the table, and zzz is an ASCII count attached to this particular variable. Note that zzz is stored at the end of the label field of all variables in the label table, and that the size of a variable in the table is three more than the number of characters defining its name. Whenever the translator encounters a variable, it first searches the table for a match. If a match occurs, it checks whether the variable occurred in the label field of the input line. If this is the case, then the translator increments the variable counter at the end of the variable's field in the label table. If a variable occurred in the operand field, then the translator returns the counter as is from the table. New variables have their counter initialized to three ASCII zeroes.

To sum up label translation, the Merlin code segment on the left would produce the ORCA segment on the right:

Merlin			ORCA		
	. . .		. . .		
]1	LDA	NUM,X	V0000000	LDA	NUM,X
	BMI	@NEG		BMI	SYS0001
	INX			INX	
	BPL	]1		BPL	VOOOOOOO
@NEG	ADC	#3	SYS0001	ADC	#3
]1	CMP	#6	V0000001	CMP	#6
BEQ	@NEG		BEQ	SYS0001	
	ADC	#2		ADC	#2
	BNE	]1		BNE	V0000001
:LOOP	LDA	TABLE,Y	L0002001	LDA	TABLE,Y
	BEQ	OUT		BEQ	OUT
	INY			INY	
	BNE	:LOOP		BNE	L0002001
OUT	STA	VALUE	OUT	STA	VALUE
:LOOP	CMP	#2	L0002002	CMP	#2
	. . .		. . .		

Instructions

Both ORCA and Merlin employ the standard set of three-character mnemonics to express the instruction set of the 65xx processor, and both allow the extensions BLT for BCC and BGE for BCS. Wherever possible, both assemblers resolve addresses to zero-page locations. The mechanism available to the programmer to override addressing assumptions is different, however. Under ORCA, the operand field begins with a '!' or '|' character to inform the assembler that the operand's address is two bytes long. Tle operand field can also begin with a '>' character to request long (3-byte) addressing. Under Merlin, any character except D' may be placed in the fourth byte of the opcode to request absolute addressing. Tberefore, an instruction whose fourth byte is not 'D'or'd' will be translated as !expression:

Merlin	ORCA
`LDA:  $3`	`LDA  !$3`

Expressions

Valid operands for 65xx instructions are identical in the two assemblers, with the exception of forced addressing described above, immediate data syntax, and expressions. The sequence #/ in Merlin specifies the high byte of the expression given in the operand. ORCA allows the same byte selection with the single symbol / . The symbols < to access the low byte of an expression and > to access the high byte are the same in both ORCA and Merlin. ORCA additionally recognizes ^ as the third byte of an expression.

Identical	Merlin	ORCA	Byte of Operand
`#<expression`			`low (1st) byte`
`#>expression`			`high (2nd) byte`
`#expression`			`low (1st) byte`
	`#/expression`	`/expression`	`high byte of immediate value`
		`#^expression`	`highest (3rd) byte of expression`

Valid arithmetic operators used in expressions are the same for the two assemblers: + - / * . The logical operators are dissimilar; they are summarized in the following table.

Operator	Merlin	ORCA
Addition	+	+
Subtraction	-	-
Multiplication	*	*
Division	/	/
Indirect address	( )	( )
Bitwise OR	.	
Bitwise AND	&	
Bitwise EOR	!	
Wordwise OR		.OR.
Wordwi3e AND		.AND.
Wordwise EOR		.EOR.
NOT		.NOT.
Grouping		(expression)
Bit shift		|
Equality		=
Not equal		<>
Greater than		>
Less than		<
Greater than or equal		>=
Less than or equal		<=

Rules of precedence also differ between Merlin and ORCA. Merlin assumes a strictly left-toright associativity, and does not permit parentheses within expressions except to denote an indirect address. ORCA defines precedence of operators according to the following rules:

Highest:	.NOT.	( )				
Next highest:	*	/	.AND.	|		
Next highest:	+	-	.OR.	.EOR.		
Lowest:	=	<>	<=	>=	<	>

For example, the expressions below would result in two different values being returned by ORCA and Merlin, because of the difference in precedence:

	Merlin	ORCA	
3+2*5	(3+2)*5 = 25	3+(2*5) = 13	
4-1/3	(4-1)/3 = 1	4-(1/3) = 4	integer division

In order to define Merlin expressions in ORCA's syntax, the translator uses parentheses to form groupings that produce an unambiguous left-to-right precedence. The start character for complex expressions is a '+.' Labels appearing within expressions are translated to ORCA labels as required. The examples below illustrate expression translation:

Merlin Expression	ORCA Expression
`#/LABl-LAB2`	`/LABl-LAB2`
`($32+3),Y`	`($32+3),Y`
`5-6+7*%0111/:LOOP`	`+(((5-6)+7)*%0111)/L0025003`
`"a"&$lF`	`** "a"&$1F ** ** Cannot translate bitwise operators **`

Note that numbers for both assemblers are coded using the standard 65xx conventions: a hex number begins with the character $, a binary number begins with % , and a decimal value contains the digits zero through nine. Pathnames are also coded identically, following the syntax rules accepted by ProDOS. The final example above shows that Merlin expressions conutining bitwise logical operators are flagged as errors by the translator.

Character strings are represented in different formats under ORCA and Merlin. ORCA strings can contain any printable characters, and are delimited by ' ' or " ". The delimiters must match, and both are required. A delimiter appearing within a string must be doubled. Delimiters have no bearing on the setting of the most significant bit of the characters wiffiin the string. A string containing only a pair of delimiters denotes the null (empty) string. Merlin strings, on the other hand, do not require matching delimiters, and the final delimiter may be omitted for single-character strings. A Merlin delimiter may be any non-numeric character other than / or , . The delimiter character used in Merlin strings has a special meaning. If it is less in value than an ASCII single-quote mark, it specifies that the characters within the string are to be represented internally with their high bits set. Otherwise, their high bits are to cleared.

Merlin strings are translated to ORCA strings by enclosing the entire string between singlequote delimiters. A delimiter found embedded in the string is doubled. If the Merlin delimiter selects high bit on, then the ORCA directive MSB ON is placed in the ORCA source file before the line containing the translated string, and the ORCA directive MSB OFF is placed in the source file after the line. Note that MSB is used to set the high bit of character data under ORCA/M.

Directives

Directives are highly individualized for any assembler, reflecting the basic structure of the assembler itself as well as the particular taste and biases of the assembler's author(s). Chapter Four of this manual discusses every Merlin directive, explaining the ORCA translation of each.

Macros

Both ORCA and Merlin support the use of macros, which are lines of assembly-language code that are inserted into a program when the assembler sees a macro name as an opcode. Both allow values to be passed to macros, and the values passed are associated with symbolic parameters. Merlin's symbolic parameters are predefined as]1,]2,...,]8. When used in a Merlin macro, 12,

for example, means to replace]2 by the second value given on the macro invocation line of the source file. ORCA macros permit symbolic parameters to be defined within the macro. The parameters that are to be sent to the macro have symbolic names, and these are defined on the macro model line (the line following the one containing the word 'MACRO.') ORCA macros also permit the definition of other symbolic variables within the macro. The example below shows some of the differences between ORCA and Merlin macros.

	Merlin			ORCA	
MACRO DEFINITION:					
ADD	MAC			MACRO	
	CLC		&LAB	ADD	&A,&B,&C
	LDA	#]1	&LAB	CLC	
	ADC	]2		LDA	#&A
	STA	]3		ADD	&B
	EOM			STA	&C
			MEND		
MACRO INVOCATION:					
	PMC	ADD.3;4;ADDR		ADD	3,4,ADDR

ORCA does not allow macros to be defined within a program. Macros are defined in a special file containing only macros, and the file is accessed from the program with the MCOPY and MLOAD directives. Whenever the translator finds a MAC directive, it writes the line containing the MAC and all other lines up to and including the next EOM directive to a file named MY.MACROS. ORCA allows macros to be neste4 but only to a level of four deep. Macro nesting is handled differently by the two assemblers. ORCA considers macros to be nested when one macro calls another macro. All macro definitions are complete--that is, they contain the definition of only one macro. The translator therefore separates nested Merlin macros.

MY.MACROS is created at the beginning of the translation. If the file already exists when the translation starts, it is overwritten. This means that you will have to move or rename MY.MACROS between executions of the translator in the same directory. IIe directives which are specific to macro creation and handling are discussed on an individual basis in the next chapter.

Merlin variables that are used within a macro receive special translation. If the variable is one of the special Merlin variables]1 through]8, then it is translated to a corresponding ORCA symbolic parameter in the range &A through &H. If the variable is not one of]1 through]8, then it is converted to an ORCA label as described earlier in this chapter.

Chapter 4 - Translation of Merlin Directives

This chapter describes the substitutions that Merlin-to-ORCA makes when it encounters a Merlin directive. In general, a label occurring in a Merlin source line is handled as explained in Chapter 3 in the section "Labels." Operands used by different directives tend to be rather specialized. Therefore, operand translation is discussed in the context of the directive which operates upon it. Merlin directives which are said to be "rejected" are shown in the ORCA source file as a comment, followed by a commented line containing an error message:

Merlin	ORCA
`Label EXT`	`**  Label      EXT  **` `**  This directive is not translated  ***`

Directives Which Use Strings

The following Merlin directives accept delimited strings as operands: ASC, DCl, FLS, INV, KBD, REV, and STR. The translator handles them in a special manner, using macros supplied in the file MERLIN.MACROS. Each of these directives has two different macros associated with it, one for when the high bit is clear, and another for when the high bit is set. In order to determine which macro to use, the translator examines the Merlin delimiter to see if the high bit should be set. If it should, the translator uses the directive name for the macro name, with an 'H' appended to the name of the macro. For example,

Merlin	ORCA
`ASC  "string"`	`ASCH  'string'`

If the high bit is not to be set, the translator simply uses the name of the directive to invoke the macro for that particular directive:

Merlin	ORCA
`DCI  'hey hey'`	`DCI  'hey hey'`

The string contained in the operand is then put into ORCA format, using single quote marks as delimiters. Single or double quote marks found within the string are doubled in preparation for ORCA's macro-handling facilities. Hex data found at the end of the Merlin operand is compressed into a single string, delimited by single quotes:

Merlin	ORCA
`Label  INV  "HEY "FF`	`Label  INVH  'HEY ','FF'`

Note that you can use these macros in your own programs, provided you specify the operands in the expected format. The sections which describe the translation of directives which rely upon special macros give examples of Merlin code, followed by the translated ORCA code, and ending with a generalization of the code produced by the special macro.

ASC — Define ASCII String

ASC is translated to a DC character statement. Any hex data is converted to a DC hex statement:

Merlin	Translation	Code Produced by Macro
`LAB  ASC "hey hey "FF`	`LAB ASCH 'hey hey ','FF'`	`LAB DC C"hey hey "` `DC H"FF"`

AST — Print Asterisks in Assembly Listing

When found during translation, AST causes the number of asterisks given in the operand to be written to the ORCA source file. This means that the translator must make sense of the operand at the time it is encountered. For simplicity's sake, the translator requires that the operand be a decimal number. For example,

Merlin	ORCA
`AST  8`	`********`
`AST  value`	`**  AST  value  **`
	`**  Can only translate simple decimal operands`
`**`	

CHK — Compute Check Sums

ORCA has no means of computing check sums during assembly. This directive is therefore rejected.

CYC — Print CPU Cycle Times

CYC is translated as the ORCA directive INSTIPAE, which prints CPU cycle times in an extra column to the left of the source lines. Cycle times which are subject to change, such as because of crossing a bank boundary, are marked with an asterisk. INSTUVE does not permit totalling, averaging, or resetting to zero:

Merlin	ORCA
`CYC`	`INSTIME ON`
`CYC AVE`	`INSTIME ON`
`CYC OFF`	`INSTIME OFF`

DA — Define Address

This directive is entirely analogous to the ORCA DC A directive (define address constant), which defines two-byte integers for each value given in the operand. The ORCA directive requires that the operand be enclosed in quote marks.

Merlin	ORCA
`Lab DA $2000,198,Lab3`	`Lab DC A'$2000,198,Lab3'`

DAT — Date Stamp Listing

This directive is unnecessary under ORCA because ORCA date-stamps all assembly listings. You may wish to examine ORCA's TITLE directive, which lets you put page numbers and an optional title at the top of every page of an assembly listing.

DCI — Dextral Character Invert

The final character contained in the string is removed. ne first part of the string is translated as a DC character statement, as is the final character of the Merlin string. Its high bit is set opposite to that of the rest of the string, however. Any trailing hex data is converted to a DC hex statement:

Merlin	Translation	Code Produced by Macro

`MSG DCI (hey, you(,8f,aa`	`MSG DCIH 'hey, you','8faa'`	`MSG DC C"hey, yo"`
		`MSB ON`
		`DC C"u"`
		`MSB OFF`
		`DC H"8faa"`

DDB Define Double Byte, Reverse Order

DDB is translated using a macro named DDB, located in MERLIN.MACROS. The macro strips each value in the operand and writes it with its byte-order reversed, using ORCA DC I1 (define constant one-byte integer) directives. The macro DDB may be used in other ORCA source programs, provided that it is included in a macro file that will be used by the program. The operands for the DDB macro are coded without quotes.

Merlin	Translation	Code Produced by Macro
`LAB DDB $7FAB`	`LAB DDB '$7FAB'`	`LAB DC II>'$7FAB'` `DC Il'$7FAB'`

DFB, DB Define Byte

DFB is translated as the ORCA directive DC I1 (define constant one-byte integer). The individual values contained in the operand are passed through the translator's expression evaluator. When the high byte of a multiple-byte value is requested, the translator produces code to perform bit shifting:

Merlin	ORCA
`LAB1 DFB 98,>$7FAB`	`LAB1 DC I1'98,$7FAB|-8'`

The high byte of the second expression is obtained by bit-shifting the two-byte value backwards eight bits. This could also be written as

```
DC       I1>'$7FAB'
```

DO, ELSE, FIN

The Merlin conditional assembly language is translated to ORCA's conditional assembly language using the ORCA directives AT and AGO, and sequence symbols. When a DO directive is found, the Translator substitutes AIF for the opcode. The DO's operand is then passed through the translator's expression evaluator. The operand is enclosed in parentheses, and compared to zero as a whole. If the value is equal to zero, then the code between the DO and the next ELSE or FIN is not assembled. For example,

Merlin	ORCA

`DO 3+LAB`
`. . .`
`FIN`
	`AIF (3+LAB)=0,.I`
	`. . .`
	`.I`

The meaning of the AIF statement is: If (3+LAB) equals zero, then branch to the line beginning with the sequence symbol .I .

When an ELSE directive is found, the translator issues an unconditional branch to the next sequence symbol, followed by a line containing the previous sequence symbol. For example,

Merlin

```
DO   3
. . .
ELSE
```

ORCA

```
      AIF  (3)=0,.I
      . . .
      AGO  .J
.I
```

In the ORCA translation, the code between the DO and ELSE would only be executed if the DO's operand was true (i.e. not equal to zero). The AGO .J instruction at the end of the DO-when-true block causes the assembler to skip past the ELSE clause. If the DO's operand is false (i.e. equal to zero), the assembler will skip to the line marked by the sequence symbol .I, which marks the beginning of the ELSE clause.

A FIN directive causes the translator to issue the sequence symbol specified in the last DO or ELSE translation. In our example, it might be .J . Putting this all together,

Merlin

```
DO   (GradeA)
. . .
ELSE

. . .
FIN
```

ORCA

```
      AIF  (GradeA)=0,.I
      . . .
      AGO  .J
.I
      . . .
.J
```

Note that the translator allows for nesting of DOs up to eight levels deep. Attempting to nest deeper than eight levels results in an error message being issued by the translator.

DS — Define Storage

DS is translated with the aid of a macro named MDS, located in MERLIN.MACROS. If the first operand of the DS directive is a single '\', then the macro translates the source line as ALIGN 256, the ORCA directive which causes alignment to a specified boundary. The aligrunent called for by '\' is to the next page of memory, hence an alignment of 256. If a second operand follows the '\', the translator will issue a message that it is unable to fill memory with anything but zeroes.

If the first operand of the DS directive is a positive value, and has no second operand, it is translated as an ORCA DS directive, which functions just like the Merlin DS directive.

A second operand following a first positively-valued operand causes the macro to produce code which will fill memory with the value given in the second operand. The translator uses an ORCA DC I1 directive to do this.

If the first operand is a negative value, then the macro rewrites it as an ORCA ORG *-<value>, which is ORCA's method of backing up the current location counter. Examples:

Merlin	Translation	Code Produced by Macro
`DS  5`	`MDS  5`	`DS   5`
`DS  10,$F`	`MDS  10,$F`	`DC   10I1'$F'`
`DS  -5`	`MDS  -5`	`ORG  *-5`
`DS  \`	`MDS  \`	`ALIGN 256`
`DS  \,$F`	`MDS  \,$F`	`ALIGN 256`
		`**  Can only zero-fill`
`**`		

DSK, SAV — Save Object Module

ORCA does not allow saving portions of the object module to different disk files. It writes the complete object module to a single file, specified in the operand of the ORCA KEEP directive, or given from the command line with the KEEP parameter. The KEEP directive, if it is used, must appear before the first START of an ORCA assembly-language program. The translator uses an internal flag, named KEEP FLG, which is set when the first SAV or DSK directive is encountered. The first SAV or DSR is accepted, and translated to the ORCA directive KEEP. Subsequent SAVs or DSKs are flagged as errors. For example

Merlin	ORCA
`SAV  MYFILE.O`	`KEEP  MYFILE.O`
`. . .`	`. . .`
`DSK  FILE3`	`**  DSK  FILE3  **`
	`**  Only 1 SAV or DSK directive may be used  **`

DUM, DEND — Define "Dummy" Section

DUM is translated using the ORCA macro named DUM,'located in the file MERLIN.MACROS. The macro uses the ORCA directive OBJ to establish an ORG address for the dummy section. It also uses a global arithmetic variable named &DUM, setting it to the current location counter. Source lines occurring between DUM and DEND are translated in the usual manner. When the translator finds a DEND directive, it uses a macro named DEND, contained in the file MERLIN.MACROS, to finalize the translation of the dummy section. The DEND macro first issues an ORCA OBJEND directive to signal the end of the dummy section. It then calculates the length of the dummy section and issues an ORCA ORG directive with an operand to "back up" over the dummy section. Because macros are used to perform the translation, you will see

```
DUM      < operand >
. . .
DEND
```

in the ORCA source file.

DW Define Word

DW is the same directive as the Merlin directive DA. What's more, ORCA's DC A (define address constant) directive is just a mnemonic for its DC I (define two-byte integer constant) directive. As with DA, DW is translated directly.

Merlin

```
LB1   DW    $300,876
```

ORCA

```
LB1   DC    1'$300,876'
```

END End Program

This directive in the source file causes the translator to set its internal end-of-file flag to true, and wrap up the translation program.

ENT Define Entry Point

ENT is translated as the ORCA directive ENTRY. ENTRY defines an alternate entry-point into a segment. In the example below, you could enter the subroutine SEG2 at the beginning, or at the entry points ENT1 or ENT2.

```
Main  START                  SEG2  START
      . . .                        . . .
                             ENT1  ENTRY
      JSR   ENT1             ENT1  ENTRY
      JSR   SEG2
      . . .                        . . .
                             ENT2  ENTRY
      JSR   READIT                 JSR   READIT
      RTS                          RTS
      END                          END
```

EOM or <<< End of Macro Definition

The Merlin directive EOM signals the end of a macro definition. It is translated directly to the ORCA directive MEND, which also flags the end of a macro definition. MEND is written to MY.MACROS, and the translator's within-macro flag is set to false so that subsequent lines of source will be written to standard output rather than to MY.MACROS.

EQU or = Equate

Merlin and ORCA equates are used in the same way in a program, to define constants or addresses. The syntax is the same except for = , which is not recognized by ORCA. It is

important to note that the labels attached to equates are *case insensitive* under ORCA/M 4.1: StARt is the same as sTArT, for example. In translating equates, Merlin-to-ORCA will translate the label, then use 'EQU' as the opcode, and then pass the operand through its expression evaluator.

Merlin	ORCA
`Last  =     45*3`	`LAST  EQU   45*3`
`HERE  EQU   *`	`HERE  ANOP`

Under ORCA/M 1.0, you may use EQU * to set a label to the value of the current location counter. Under ORCA/M 4.1, this is not allowed. You may set a label to the value of the current location counter under either system by use of the ORCA directive ANOP, which means "no operation."

Under ORCA/M 1.0, equates can use values which have not yet been defined, and the assembler can be set to be case sensitive. You would use the ORCA directive CASE, with an operand of ON to request case sensitive handling of labels.

ERR — Check Error

There is no equivalent to ERR under ORCA. This directive is therefore rejected.

EXP — Expand Macros

EXP is akin to the ORCA directive GEN, both of which can be set to ON to display macro expansions. GEN does not support an operand such as ONLY, however. The translations are summarized below.

Merlin	ORCA
`EXP   ON`	`GEN   ON`
`EXP   OFF`	`GEN   OFF`
`EXP   ONLY`	`GEN   ON`

EXT — External Label

Merlin's REL files are roughly analogous to ORCA's library files. To ensure that the ORCA linker includes a particular subroutine in the final object module, you would use a DC R directive:

```
[Label] DC    R'Subrtn1[ [,Subrtn2] [,Subrtn3] . . . ] '
```

where the [ ] enclose optional entries. The named segments must reside in the ORCA LJBRARIES prefix. The files contained in this prefix are all in object-module format (i.e. the

intermediate code produced by an assembly of the original source program). Since REL and library files are not quite the same, this directive is rejected.

FLS Display String in Flashing Video

The FLS macros examine each character in the string, causing each to be in the range $40 to $7F. A new string is built from the input string, and translated into an ORCA DC 1-byte integer string. Trailing hex data is rewritten as an ORCA DC hex statement:

Merlin

```
LBL  FLS  'ERR!'
I1'$45,$52,$52'
```

Translation

```
LBL  FLS  'ERR!'
```

Code Produced by Macro

```
LBL  DC
```

HEX Define Hex Data

BEX is translated directly into an ORCA DC H statement. All of the hex values given in the operand are compressed into a single string of hex digits. Commas within the Merlin hex string are converted to blanks for readability.

Merlin

```
HEX   AB,CD,EF
```

ORCA

```
HEX   DC    H'AB CD EF'
```

INV Display String in Inverse Video

The INV macros examine each character in the string, causing each to be in the range $3F to $7F. A new string is built from the input string and translated into an ORCA DC 1-byte integer string. Trailing hex data is rewritten as an ORCA DC hex statement:

Merlin

```
Lab  INV  &hey&,7FAA

I1'$28,$25,$39'
```

Translation

```
Lab  INVH  'hey','7FAA'
```

Code Produced by Macro

```
     MSB   ON
Lab  DC

     MSB   OFF
     DC    H'7FAA'
```

KBD — Assembly Input from Keyboard

This directive is translated using a macro named KBD, found in MERLIN.MACROS. The macro first inputs a value from the keyboard during assembly using ORCA's AINPUT directive. It then equates the label defined in the Merlin source line with the value input. The KBD macro may be used in another ORCA program by including it in a macro file that the program will be using.

Merlin

```
Label KBD
```

ORCA

```
      LCLC  &C            define string variable
&C    AINPUT "Give value for Label: "
Label EQU   &C
```

LST — Generate Assembly Listing

This directive is similar to the ORCA directive LIST, which is the substitution used by the translator. Since LIST requires an operand of either ON or OFF, a missing LST operand is converted to ON. Note that ORCA does not support keyboard input during assembly, except through the use of its AINPUT directive or when pressing special keys to abort the assembly. That is, ORCA does not allow "toggling" of LIST ON during assembly. For example,

Merlin

```
LST   OFF
LST
```

ORCA

```
LIST  OFF
LIST  ON
```

LSTDO — List "DO OFF" Areas

ORCA has no provision for displaying lines occurring within a DO OFF block. The only way to see these lines during assembly is to set the DO's operand to true. ORCA lines containing conditional assembly variables and directives are not displayed in the assembly listing unless TRACE ON is specified. For that reason, LSTDO is translated as TRACE:

Merlin

```
LSTDO OFF
LSTDO
```

ORCA

```
TRACE ON
TRACE OFF
```

LUP — Loop

It is now time to explain the third line of the translated source file:

```
      LCLA  &LUP
```

&LUP is the name of an arithmetic symbolic parameter. It is defined as such with the ORCA directive LCLA. When the translator sees the Merlin directive LUP, it sets up a loop using

ORCA's conditional assembly language. The first step is to decide the number of times to perform the loop. This is accomplished by setting &LUP to the value found in the operand of the LUP statement:

```
&LUP   SETA  <expression>
```

Note that <expression> is passed to the translator's expression evaluator to put it in ORCA's format. The bottom of the loop will be defined by the Merlin directive --^. In order to have a place to branch, the Wmslator uses a special ORCA label called a sequence symbol:

```
&LUP  SETA  <expression>
.LUP
```

A sequence symbol is a label whose first character is a period. If the assembler is not looking for a place to branch, a line beginning with a sequence symbol is treated as a comment.

The lines between LUP and --^ are translated as usual. When the translator encounters --^ it decrements the loop counter &LUP, and, if the value of &LUP is still positive, branches up to the sequence symbol .LUP:

Merlin

```
LB1   LUP   3
      ADC   #3
      --^
```

ORCA

```
LB1   ANOP       define label before looping
&LUP  SETA  3
.LUP
      ADC   #3
&LUP  SETA  &LUP-1
      AIF   &LUP>0,.LUP
```

The conditional assembly directive AIF has a two-part operand. The first part is a logical expression. If it is true (not equal to zero), then a branch to the sequence symbol given in the second part of the operand is executed.

MAC — Define Macro

MAC signals the beginning of a macro definition. If this is a new definition (i.e. not part of another macro), then the translator writes an opcode of 'MACRO' to the MY.MACROS file. On the next line it writes the name of the macro in the opcode field, and a string of eight symbolic parameters in the operand field. The symbolic parameters correspond to Merlin's eight predefined macro variables. If more parameters are coded than are needed, no harm is done.

MAC also causes the translator's internal within-macro flag to be set to true so that subsequent Merlin source lines which are part of the macro are also written to MY.MACROS. If MAC is encountered within another macro, the current macro in MY.MACROS is ended and a new one begun. Macros are ended under ORCA with the 'MEND' directive. Macro nesting ala Merlin is possible under ORCA, but the practice is not recommended. Also, you should be aware that

macros cannot be nested more than four levels deep under ORCA. By nesting we mean macro A calls macro B which calls macro C ...

Merlin

```
SUB   MAC
ADD   MAC
```

ORCA

```
      MACRO
&LAB  SUB   &A,&B,&C,&D,&E,&F,&G,&H
      MEND
      MACRO
&LAB  ADD   &A,&B,&C,&D,&E,&F,&G,&H
```

Note that the macro's symbolic parameters, which are defined on its macro model line, are local to the macro; other macros may also use the same names without conflim

MX Set Accumulator and Index Register Sizes

The trmslator uses a macro, named MX and located in NIERLIN.MACROS, to translate MX. This macro may be used in another ORCA program by including it in a macro file that the program will be using.

ORCA uses the directives LONGA and LONGI to inform the assembler of the sizes of the accumulator and index registers, respectively. LONGA ON specifies that loads and stores involving the accumulator are 16 bits. Eight-bit registers are denoted with OFF as an operand. The default register size under ORCA/M 4.1 is eight bits; under ORCA/M 1.0 it is sixteen bits. Normally the LONG and SHORT macros are used to set register sizes in ORCA. See your reference manual for details.

Merlin

```
MX    %0

MX    1
```

ORCA

```
LONGA ON
LONGI ON

LONGI ON
LONGA OFF
```

OBJ Set Division Between Object Code and Symbol Table

Since ORCA is a disk-based assembler, there is no counterpart to OBJ under ORCA. The translator therefore rejects OBJ. Note that there is an ORCA directive named OBJ, but it is used for a different purpose.

PAG This is exactly the same as the ORCA directive EJECT:

Merlin

```
PAG
```

ORCA

```
EJECT
```

PAU — Pause

No counterpart to PAU exists under ORCA. The translator rejects PAU.

PMC or >>> — Put Macro in Source Code Stream

Under ORCA, macros are invoked in a program by using the macro!s name as the opcode, and placing any values to be passed to the macro in the operand field. When the translator encounters PMC, it extracts the macro's name from the operand field and places this in the opcode field of the translated line. It then processes each value to be assigned to a different symbolic variable, and encloses the translated value in double quote marks. Note that macros invoked from other macros are coded the same way as in a program under ORCA.

Merlin	ORCA
`PMC  ADD,13;3;NUM`	`ADD  "13","3","NUM"`

The operands are enclosed in quotes in case one of the values to be passed to the macro contains indirect addressing, as in (ADDR),X.

ORG — Set Origin

The first ORG of a Merlin program is exactly equal to the first ORG of an ORCA program: they specify the load address of the program. ORCA additionally allows its ORG operand to contain the constructs *+expression or *-expression to move the location counter forward or backward. The first ORCA ORG must be coded before the first START directive of the program.

Merlin	ORCA
`ORG  $4000` `DS  -10`	`ORG  $4000` `ORG  *-10`

Merlin ORGs containing operands and appearing after the first ORG of a program define the beginning of code segments that will be moved (to the address given in the operand). The main program is loaded at the location specified in the first ORG. An ORG without an operand defines the end of the "moving" code segment. ORCA provides the same capabilities with the directives OBJ and OBJEND. The following example explains the translation:

Merlin

```
      ORG   $3000
      LDX   #END1-LB1
TOP   LDA   LBI,X
      STA   $200,X
      DEX
      BNE   TOP
      JMP   $200
      ORG   $200
LB1   LDA   #3
      . . .
END1  RTS
      ORG
      . . .
```

ORCA

```
      ORG   $3000
MAIN  START
      LDX   #END1-LB1
TOP   LDA   LB1,X
      STA   $200,X
      DEX
      BNE   TOP
      JMP   $200
LB1   ANOP
OBJ   $200
      LDA   #3
      . . .
END1  RTS
      OBJEND
      . . .
```

PUT — Insert Contents of Source File into Code Stream

PUT is very similar to the ORCA directive COPY. Both accept a pathname as an operand, and insert the file named in the operand at the current location in the file being assembled. For example

Merlin

```
PUT   /MYFILES/FILE1
```

ORCA

```
COPY  /MYFILES/FILE1
```

REL — Define Relocatable Module

As mentioned in chapter one, the ORCA/M 4.1 hnker does not produce relocatable modules, while the ORCA/M 1.0 linker produces only relocatable modules. For this reason, the translator rejects the REL directive.

REV — Reverse Character Order of String

The REV macros work on the input string from the end to the beginning, producing a new string which is the reverse of the input:

Merlin

```
LAB  REV   'hey'
```

Translation

```
LAB   REV   'hey'
```

Code Produced by Macro

```
LAB   DC    C'yeh'
```

SKP Write Blank Lines to Listing

When found during translation, SKP causes the number of blank lines given in the operand to be written to the ORCA source file. This means that the translator must make sense of the operand at the time it is encountered. For simplicity's sake, the translator requires that the operand be a decimal number. For example,

Merlin	ORCA
`SKP 3`	
`SKP value`	`** SKP value **`
	`** Can only translate simple decimal operands`
`**`	

STR Define String with Leading Count Byte

The STR macros compute the length of the input string, converting the length to a DC one-byte integer statement. The string is entered next, using a DC character statement. (Note that this can be accomplished in ORCA with its standard DW macro.) Trailing hex data is converted to a DC hex statement:

Merlin	Translation	Code Produced by Macro

`LB1 STR 'Hey !'F9 !"'`	`LB1 STR 'Hey !','F9'`	`LB1 DC I1'L:"Hey`
		`DC C'Hey !'`
		`DC H'F9'`

SW Allow "Sweet 16" Opcodes

The "Sweet 16" opcodes were not implemented in versions of the Apple II which followed the Apple][computer. For this reason, ORCA does not support these opcodes. Please consult the macro sections at the end of the ORCA reference manual for full details on using the ORCA macros which provide 16-bit integer math operations.

TR — Truncate Listing Lines

TR is roughly equivalent to ORCA's EXPAND directive. EXPAND, when set to ON, causes up to 16 bytes of each DC statement (a data definition directive, like Merlin's ASC, DCI, and DDB directives) to be displayed in the assembly listing. When set to OFF, only the first four bytes of each DC statement are shown in the listing. The number of bytes displayed on other types of source lines cannot by controlled by the user. Examples:

Merlin	ORCA
`TR  ON`	`EXPAND OFF`
`TR  OFF`	`EXPAND ON`
`TR`	`EXPAND OFF`

TYP — Set Filetype of Object Module

No counterpart to TYP exists in the ORCA assembler, so the translator rejects it However, both ORCA/M 4.1 and 1.0 provide a shell command named FILETYPE which allows you to change the type of any file.

USE — Access Macro File

USE is comparable to the ORCA directive MCOPY, which places the first four blocks of the file named in the operand into a macro buffer. Subsequent macro calls in the source code cause the assembler to search the macro buffer for the invoked macro. Portions of the macro file are swapped in and out of the buffer as necessary. For example

Merlin	ORCA
`USE  MY.MACROS`	`MCOPY MY.MACROS`

USR — Define User Opcode

This directive has no counterpart under ORCA and is therefore rejected. Note that you can provide tailored routines using macros and/or libraries.

VAR — Define Variables

For each of the values given in the operand of VAR, the translator produces a variable label and EQUates it to the value given. The labels displayed by the translator are the mnslations of the Merlin variables]1,]2, . . . up to the number of values given in the VAR operand. (Variable translation is discussed in chapter two.) The values appearing in the operand am sent to the translator's expression evaluator prior to being written to the ORCA source file. For example

Merlin	ORCA

`VAR  3+6*7;"A;Label`	`V0001002 EQU  +(3+6)*7`
	`MSB  ON`
	`V0004000 EQU  'A'`
	`MSB  OFF`
	`V0005002 EQU  LABEL`

XC — Extend Instruction Set

Under ORCA 4.1, the set of opcodes recognized by the assembler defaults to 6502 instructions only. The assembler can be told to recognize 65C02 opcodes by using the 65CO2 directive, with an operand of ON. It can be commanded to recognize 65816 opcodes by using the directive 65816, with an operand of ON.

Under ORCA 1.0, the assembler's opcode set defaults to 65816 instructions. The default can be overrridden to access, for instance, only 6502 instructions by using the directives

```
65816 OFF
65CO2 OFF
```

The translator has a flag which is set to false at the beginning of translation. When the first XC directive is seen, the translator converts it to 65CO2 ON. The next XC directive in the source file is rewritten as 65816 ON.

Merlin	ORCA
`XC`	`65C02 ON`
`XC`	`65816 ON`