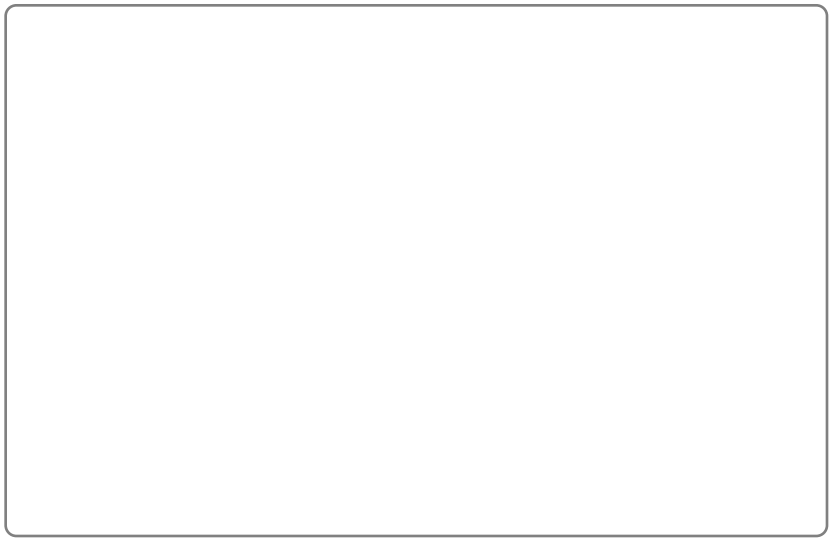
Subroutine Library Source Code

Version 2.0.1

For ORCA/M 2.0, ORCA/Pascal 1.4.2, ORCA/C 2.0
Apple IIGS Versions



By Mike Westerfield
and Barbara Allred

Limited Warranty - Subject to the below stated limitations, Byte Works, Inc., hereby warrants that the programs 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 are 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) interruption of service, loss of business, or anticipatory profits.

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

Use of Libraries - The enclosed subroutine libraries are fully copyrighted works. It is the policy of Byte Works, Inc., to license these libraries to purchasers of ORCA/M free of charge. Such licenses are generally restricted to include the libraries of binary files, and do not extend to use of the source code. A copy of the program, along with any documentation, and a list of the library subroutines used is required at the time of the licensing, and the document must give credit for using libraries from ORCA/M. For details, please contact the Byte Works, Inc.

Chapter 1 – Introduction

Overview

The ORCA Subroutine Libraries 2.0.1 is the source code for a collection of assembly-language subroutines which are used by the ORCA/M system macros and as run-time subroutines by ORCA/Pascal and ORCA/C. You are probably most familiar with these libraries as the ORCALIB, SYSLIB, SYSFLOAT and PASLIB files, located in the library prefix.

The reason for releasing this source code is partly due to our perceptions of what you need, and partly tradition. The source code for the subroutine libraries has always been available for ORCA languages; we consider it to be essential to the writing of efficient programs. To use the subroutine libraries with confidence, it is sometimes necessary to see exactly how they work. Then, too, bugs are not unheard of. If you have the source code to the libraries, it is easier for you to be sure that the bug is not in the libraries, or if it is, to track the bug down. Finally, it may be necessary to write a program for use on another computer, or in an environment other than ORCA. The libraries are optimized for the text environment of the Apple IIGS computer, and do not always work in other environments, or on other computers, or in ROM. With the source code, you can make the necessary modifications.

We felt that a concrete example of modifying a library subroutine would help you understand how libraries function in the ORCA environment. Chapter Two gives such an example.

The remainder of this chapter explains, in a general way, what is on the disk. All of the programs are thoroughly documented with internal comments, so the internal workings of the code is not elaborated on here.

The libraries can be reassembled using ORCA/M 2.0. The end product is five files: SYSLIB, used with ORCA/M, ORCA/Pascal and ORCA/C, contains the basic I/O subroutines and utility subroutines used across two or more of the languages. PASLIB, used with ORCA/Pascal, contains the subroutines used only in ORCA/Pascal. ORCALIB, used with ORCA/C, contains the subroutines used in ORCA/C. There are two versions of the SYSFLOAT library, which is called by ORCA/Pascal and ORCA/C programs for all calculations and libraries that involve floating-point calculations. The first of these versions of SYSFLOAT calls Apple's SANE tool set to perform floating-point calculations, while the second makes some direct calls to the Innovative Systems Floating Point Engine (FPE) and some calls to SANE.

Should you wish to use any of the subroutines or portions of code from this package in a commercial program, please contact us. Due to the structure of the copyright laws, we must require that all such uses be licensed. Licensing is free of charge; we simply ask that the source of the routines be acknowledged. We have not had any reason to refuse to grant a license to date.

What You Should Have Received

The Subroutine Libraries package includes this manual and two 3.5" disks. The first disk is labeled "ORCALIB, PASLIB" and contains the source code for the ORCA/C library ORCALIB and the ORCA/Pascal library PASLIB. The source code for ORCALIB is located in a folder called ORCALIB, which is itself in a folder called SOURCE. Likewise, the source code for PASLIB is located in a folder called PASLIB. The second disk is labeled "SYSFLOAT, SYSFPEFLOAT, SYSLIB" and contains two directories, named SOURCE and EXAMPLES. The SOURCE subdirectory has three subdirectories: SYSLIB has the source code for the SYSLIB library, SYSFLOAT has the source code for the SANE version of the SYSFLOAT library, and SYSFPEFLOAT has the source code for the FPE version of the SYSFLOAT library.

The EXAMPLES subdirectory contains several source files that will be used in an example in Chapter Two. These programs will be explained in Chapter Two.

Rebuilding the Libraries

Each of the folders containing source code for one of the libraries has a script file called MAKE. You can rebuild any of the libraries by setting the current directory to the appropriate folder and typing MAKE from the text shell or the shell window of the PRIZM desktop development environment. For example, to rebuild ORCALIB, you would use the commands

```
prefix /lib.source1/source/orcalib
make
```

Of course, you should only do this with a copy of the distribution disks, never with the original!

Chapter 2 – Modifying the Libraries

Naming Conventions

The library subroutine names begin with the tilde (~) character. Names that begin with the ~ character are reserved for system use. This convention prevents naming conflicts between the libraries and your program. If you look at the GET4 macro, contained in the file M16.I.O in the ORCA MACROS prefix, you will see the instruction

```
        JSL   ~GET4
```

~GET4 is the library subroutine which receives a four-byte integer from standard input and places the value on the stack. It is a segment contained in the library source file named IO.ASM. The GET4 macro then calls another macro, named PL4, which removes the value from the stack and places it in the location specified when invoking the GET4 macro. Note that macros which are "helpers" for a main macro also begin with the tilde character.

An Example: Modifying MUL2

The following example demonstrates how to modify the subroutine libraries. Listed below is the macro MUL2, which performs a signed multiplication of the first two parameters. The result is stored at the location given by the third parameter if it is coded; otherwise, it is stored at the location specified by the first parameter. MUL2 is located in the file M16.INT2MATH, in the ORCA MACROS folder.

```
        MACRO
&LAB    MUL2  &N1,&N2,&N3
        AIF   C:&N3,.A
        LCLC  &N3
&N3     SETC  &N1
.A
&LAB    ~SETM
        LCLC  &C
&C      AMID  "&N2",1,1
        AIF   "{"="&C",.B
        AIF   "["="&C",.B
        ~OP   LDX,&N2
        AGO   .C
.B
        ~LDA  &N2
        TAX
.C
        ~LDA  &N1
        JSL   ~MUL2
        ~STA  &N3
        ~RESTM
        MEND
```

Note that the macros ~SETM, ~RESTM, ~OP, ~LDA, and ~STA are all special-purpose macros which are located at the end of the M16.INT2MATH file.

As you may know, signed arithmetic operations take longer than unsigned operations because of the sign manipulation involved. Suppose that you'd like to have an unsigned two-byte multiplication routine. You could obtain one very quickly by making some modifications to the multiply routine contained in the subroutine libraries. The segment named ~MUL2 in the library file I2.ASM performs the signed multiply; it is given below:

```
*************************************************************
*
*  ~MUL2 - Two Byte Signed Integer Multiply
*
*  Inputs:
*        A - multiplicand
*        X - multiplier
*
*  Outputs:
*        A - result
*        V - set if an overflow occurred
*
*  Notes:
*        1) Assumes long A and X on entry.
*
*************************************************************
*
~MUL2    START
         LONGA ON
         LONGI ON
NUM1     EQU   0
NUM2     EQU   4
SIGN     EQU   6

         TAY                           save value
         PHD                           set up local space
         TSC
         SEC
         SBC   #7
         TCD
         DEC   A
         TCS
         TYA                           restore value
         LDY   #0                      make all arguments positive
         BIT   #$8000                   start with A
         BEQ   ML1
         EOR   #$FFFF
         INC   A
         INY
ML1      STA   NUM1+2
         TXA                           now do X
         BPL   ML2
         DEY
         EOR   #$FFFF
         INC   A
ML2      STA   NUM2
         STY   SIGN

         LDY   #16                     do 16 bit multiply
         LDA   #0                      set up the high byte of the result
ML3      LSR   NUM1+2                  test the LSB
         BCC   ML4                     br if it is off
         CLC                           add in partial product
         ADC   NUM2
ML4      ROR   A                       multiply answer by 2
         ROR   NUM1
         DEY                           loop
         BNE   ML3
         TAX                           check for overflow
         BNE   OVFL
         LDA   NUM1
         BMI   OVFL

         LDY   SIGN                    if result is to be neg, reverse sign
         BEQ   ML5
         EOR   #$FFFF
         INC   A
ML5      TAY                           restore stack, DP
         TDC
         CLC
         ADC   #7
         TCS
         PLA
         TCD
         TYA
         CLV
         RTL

OVFL     TDC                           restore stack, DP
         CLC
         ADC   #7
         TCS
         PLA
         TCD
         SEP   #%01000000              SEV
         RTL
         END
```

To modify ~MUL2 so that it performs unsigned arithmetic, all that is necessary is to remove the portions of the code that deal with the sign. Since the amount of work space decreased, some changes to the set up code and exit code have also been made. The required changes are shown in boldface. Assuming that we'd like to retain our original ~MUL2, we could edit a copy ~MUL2 and use the new one in programs which perform unsigned multiplication. If ~MUL2 appears in our program, the linker will not include the ~MUL2 from the SYSLIB library. The sample program below tests our new ~MUL2:

```
            KEEP    STUFF
            MCOPY T.MAC
*************************************************************
*
*  TEST - Simple program to test if modified multiply routine
*         works.
*
*************************************************************
*
T           START
            PHK     Program & data in same bank
            PLB
            MUL2    #3,#4,ANS               Multiply 3 by 4; store in ANS
            PUTS    #'ANS: '                Echo multiply result
            PUT2    ANS,CR=T
            LDA     #0
            RTL

ANS         DS      2
            END

*************************************************************
*
*  ~MUL2 - Two-Byte Unsigned Integer Multiply
*
*  Inputs:
*        A - multiplicand
*        X - multiplier
*
*  Outputs:
*        A - result
*        V - set if an overflow occurred
*
*  Notes:
*        1) Assumes long A and X on entry.
*
*************************************************************
*
~MUL2       START
            LONGA   ON
            LONGI   ON
NUM1        EQU     1
NUM2        EQU     5

            PHD                             set up local data area on the
            PHA                              stack
            PHX
            PHX
            TSC
            TCD

            LDY     #16                     do 16 bit multiply
            LDA     #0                      set up the high byte of the result
ML3         LSR     NUM1+2                  test the LSB
            BCC     ML4                     br if it is off
            CLC                             add in partial product
            ADC     NUM2
ML4         ROR     A                       multiply answer by 2
            ROR     NUM1
            DEY                             loop
            BNE     ML3
            TAX                             check for overflow
            BNE     OVFL
            LDA     NUM1
            BMI     OVFL

ML5         PLY                             restore stack, DP
            PLY
            PLY
            PLD
            CLV
            RTL
```

```
OVFL     PLY                          restore stack, DP
         PLY
         PLY
         PLD
         SEP                          #%01000000
         RTL
         END
```

File T From EXAMPLES Folder

To see how much faster unsigned multiplications really are, let's run some benchmarks. The three programs below all run the same test. The test performs ten different multiplications, and these ten operations are repeated 10,000 times. The first test uses our modified ~MUL2. The second test is exactly the same as the first, except that it uses the library's version of ~MUL2. The final test calls the integer math toolkit. The call to the toolkit would generally be put into a macro, but we have included it here to help you understand how a call to the toolkit works. The EXAMPLES subdirectory also contains the file named TEST.MACROS, which includes the macros used in the three benchmark programs. It is included so that you can run the tests yourself, without having to use the MACGEN utility to build the macro file first.

```
         KEEP  UNSIGNED
         MCOPY TEST.MACROS
*************************************************************
*
*  This benchmark performs 10 multiplications, saving the result
*  of each.  The multiplications are repeated 1000 times.  The
*  UNMUL2 macro calls the routine ~UNMUL2, the unsigned version of
*  the library segment ~MUL2.
*
*  Written by Barbara Allred and Mike Westerfield
*
*  By The Byte Works Inc.
*  Copyright (c) 1986
*  All rights reserved.
*
*************************************************************
*
TEST     START
         PHK                          Program & data in same bank
         PLB

         PUTS  #'Start the clock',CR=T
         LDA   #10000
         STA   COUNT

TOP1     LDX   #10
         LDY   #0

TOP2     PHX                          save X and Y registers
         PHY

         LDA   M1,Y
         STA   ADDR1
         LDA   M2,Y
         STA   ADDR2

         UNMUL2  ADDR1,ADDR2,TEMP     multiply next pair

         PLY                          restore X and Y registers
         PLX
         LDA   TEMP                   save result of multiply
         STA   RES,Y

         INY                          update table index
         INY
         DBNE  X,TOP2

         DBNE  COUNT,TOP1             repeat inner loop 1000 times
         PUTS  #'Stop the clock',CR=T

         LDX   #18                    check results
TOP3     LDA   RES,X
         CMP   ANS,X
         BNE   ERROR
         DEX
         DBPL  X,TOP3
         LDA   #0
         RTL
```

```
ERROR    PUTS  #'Error in multiply routine',CR=T
         LDA   #$FFFF
         RTL

ADDR1    DS    2
ADDR2    DS    2
TEMP     DS    2
COUNT    DS    2
RES      DS    20
M1       DC    I2'0,11,22,33,44,55,66,77,88,99'
M2       DC    I2'5,76,34,123,654,41,92,18,12,99'
ANS      DC    I2'0,836,748,4059,28776,2255,6072,1386,1056,9801'
         END

************************************************************
*
*  ~UNMUL2 - Two-Byte Unsigned Integer Multiply
*
*  Inputs:
*        A - multiplicand
*        X - multiplier
*
*  Outputs:
*        A - result
*        V - set if an overflow occurred
*
*  Notes:
*        1) Assumes long A and X on entry.
*
************************************************************
*
~UNMUL2  START
         LONGA ON
         LONGI ON
NUM1     EQU   1
NUM2     EQU   5

         PHD                           set up local data area on the stack
         PHA
         PHX
         PHX
         TSC
         TCD

         LDY   #16                     do 16 bit multiply
         LDA   #0                      set up the high byte of the result
ML3      LSR   NUM1+2                  test the LSB
         BCC   ML4                     br if it is off
         CLC                           add in partial product
         ADC   NUM2
ML4      ROR   A                       multiply answer by 2
         ROR   NUM1
         DEY                           loop
         BNE   ML3
         TAX                           check for overflow
         BNE   OVFL
         LDA   NUM1
         BMI   OVFL

ML5      PLY                           restore stack, DP
         PLY
         PLY
         PLD
         CLV
         RTL

OVFL     PLY                           restore stack, DP
         PLY
         PLY
         PLD
         SEP   #%01000000              SEV
         RTL
         END
```

File UNTEST From EXAMPLES Folder

```
         KEEP  SIGNED
         MCOPY TEST.MACROS
************************************************************
*
*  This benchmark performs 10 multiplications, saving the result
*  of each.  The multiplications are repeated 1000 times.  The
*  MUL2 macro calls the library routine ~MUL2.
*
*  Written by Barbara Allred and Mike Westerfield
*
*  By The Byte Works Inc.
*  Copyright (c) 1986
*  All rights reserved.
*
************************************************************
*
T        START
         PHK                           Program & data in same bank
         PLB

         PUTS  #'Start the clock',CR=T
         LDA   #10000
         STA   COUNT

TOP1     LDX   #10
         LDY   #0

TOP2     PHX                           save X and Y registers
         PHY

         LDA   M1,Y
         STA   ADDR1
         LDA   M2,Y
         STA   ADDR2

         MUL2  ADDR1,ADDR2,TEMP        multiply next pair

         PLY                           restore X and Y registers
         PLX
         LDA   TEMP                    save result of multiply
         STA   RES,Y

         INY                           update table index
         INY
         DBNE  X,TOP2

         DBNE  COUNT,TOP1              repeat inner loop 1000 times
         PUTS  #'Stop the clock',CR=T

         LDX   #18                     check results
TOP3     LDA   RES,X
         CMP   ANS,X
         BNE   ERROR
         DEX
         DBPL  X,TOP3
         LDA     #0
         RTL

ERROR    PUTS  #'Error in multiply routine',CR=T
         LDA   #$FFFF
         RTL

ADDR1    DS    2
ADDR2    DS    2
TEMP     DS    2
COUNT    DS    2
RES      DS    20
M1       DC    I2'0,11,22,33,44,55,66,77,88,99'
M2       DC    I2'5,76,34,123,654,41,92,18,12,99'
ANS      DC    I2'0,836,748,4059,28776,2255,6072,1386,1056,9801'
         END
```

File LIB.TEST From EXAMPLES Folder

```
         KEEP  TOOLKIT
         MCOPY TEST.MACROS
************************************************************
*
*  This benchmark program performs 10 multiplications, saving
*  the result of each.  The 10 multiplications are repeated
*  1000 times.  The multiplication is performed using the integer
*  math toolkit.
*
*  Written by Barbara Allred and Mike Westerfield
*
*  By The Byte Works Inc.
*  Copyright (c) 1986
*  All rights reserved.
*
************************************************************
*
T        START
         PHK
         PLB

         PUTS  #'Start the clock',CR=T
         LDA   #10000
         STA   COUNT

TOP1     LDY   #0
         LDX   #10

TOP2     PHY                           save Y and X registers
         PHX

         PHA                           push room for output
         PHA
         LDA   M1,Y                    push first value
         PHA
         LDA   M2,Y                    push second value
         PHA

         LDX   #$090B                  call multiply tool
         JSL   $E10000

         PLA                           pull 32-bit result from stack
         STA   TEMP
         PLA
         STA   TEMP+2

         PLX                           restore X and Y registers
         PLY

         LDA   TEMP
         STA   RES,Y

         INY                           get next pair to multiply
         INY
         DBNE  X,TOP2

         DBNE  COUNT,TOP1              repeat inner loop 1000 times

         PUTS  #'Stop the clock',CR=T

         LDX   #18                     check results
TOP3     LDA   RES,X
         CMP   ANS,X
         BNE   ERROR
         DEX
         DBPL  X,TOP3
         LDA     #0
         RTL

ERROR    PUTS  #'Error in multiply routine',CR=T
         LDA   #$FFFF
         RTL

TEMP     DS    4
COUNT    DS    2
RES      DS    20
M1       DC    I2'0,11,22,33,44,55,66,77,88,99'
M2       DC    I2'5,76,34,123,654,41,92,18,12,99'
ANS      DC    I2'0,836,748,4059,28776,2255,6072,1386,1056,9801'
         END
```

File TOOL.TEST From EXAMPLES Folder

The benchmark times we obtained, in seconds, using a standard Apple IIGS computer, were:

	Raw	Corrected	Per Multiply
Loop itself, no multiplies:	2.9	0	0
Library Multiply:	24.4	21.5	215 microseconds
Unsigned Multiply:	22.7	19.8	198 microseconds
Toolkit Multiply:	31.8	28.9	289 microseconds

"Raw" is the time between the messages "Start the clock" and "Stop the clock." "Corrected" are the execution times minus the loop overhead. "Per multiply" are the average times for individual multiplications.

Adding Library Routines

Suppose we'd like to add our new ~MUL2 to the system library. First of all, let's rename it so that we may distinguish it from SYSLIB's original ~MUL2. Let's call it ~UNSMUL2, for unsigned two-byte multiplication. We can place our ~UNSMUL2 in the library file I2.ASM after ~MUL2 by using the copy and paste commands provided with the ORCA editor. To rebuild the library, execute the MAKE script, then copy the new SYSLIB library file to your LIBARIES folder, replacing the old SYSLIB.

One last detail needs to be addressed before we are done. How are we going to access ~UNSMUL2? The easiest way to do this is to modify our MUL2 macro. All we need to do is to rename the macro, change the segment name for the JSL instruction within the macro, and store the new macro in the ORCA MACROS prefix. Let's call the new macro UNSMUL2. The new macro is given below:

```
          MACRO
&LAB      UNSMUL2 &N1,&N2,&N3
          AIF     C:&N3,.A
          LCLC    &N3
&N3       SETC    &N1
.A
&LAB      ~SETM
          LCLC    &C
&C        AMID    "&N2",1,1
          AIF     "{"="&C",.B
          AIF     "["="&C",.B
          ~OP     LDX,&N2
          AGO     .C
.B
          ~LDA    &N2
          TAX
.C
          ~LDA    &N1
          JSL     ~UNSMUL2
          ~STA    &N3
          ~RESTM
          MEND
```

The changes are shown in boldface. Using the editor, we can add our new macro after MUL2 in the file MACROS/M16.INT2MATH. Now to use unsigned multiplication in a program, we can simply code something like:

```
          UNSMUL2 NUM1,NUM2
```