~~~~~~
MOTOROLA MICROPROCESSOR & MEMORY TECHNOLOGY GROUP
M68000 Hi-Performance Microprocessor Division
M68060 Software Package
Production Release P1.00 -- October 10, 1994

~~~~~~

68060 INTEGER SOFTWARE PACKAGE (Library version)

The file ilsp.s contains the "Library version" of the
68060 Integer Software Package. Routines included in this
module can be used to emulate 64-bit divide and multiply,
and the "cmp2" instruction. These instructions are not
implemented in hardware on the 68060 and normally take
exception vector #61 "Unimplemented Integer Instruction".

By re-compiling a program that uses these instructions, and
making subroutine calls in place of the unimplemented
instructions, a program can avoid the overhead associated with
taking the exception.

Release file format:

The file ilsp.sa is essentially a hexadecimal image of the
release package. This is the ONLY format which will be supported.
The hex image was created by assembling the source code and
then converting the resulting binary output image into an
ASCII text file. The hexadecimal numbers are listed
using the Motorola Assembly Syntax assembler directive "dc.l"
(define constant longword). The file can be converted to other
assembly syntaxes by using any word processor with a global
search and replace function.

To assist in assembling and linking this module with other modules,
the installer should add a symbolic label to the top of the file.

This will allow calling routines to access the entry points
of this package.

The source code ilsp.s has also been included but only for
documentation purposes.

Release file structure:

The file ilsp.sa contains an "Entry-Point" section and a
code section. The ILSP has no "Call-Out" section. The first section
is the "Entry-Point" section. In order to access a function in the
package, a program must "bsr" or "jsr" to the location listed
below in "68060ILSP Entry Points" that corresponds to the desired
function. A branch instruction located at the selected entry point
within the package will then enter the correct emulation code routine.

The entry point addresses at the beginning of the package will remain
fixed so that a program calling the routines will not have to be
re-compiled with every new 68060ILSP release.

For example, to use a 64-bit multiply instruction,
do a "bsr" or "jsr" to the entry point defined by
the 060ILSP entry table. A compiler generated code sequence
for unsigned multiply could look like:

```
# mulu.l <ea>,Dh:Dl
# mulu.l _multiplier,%d1:%d0

      subq.l      &0x8,%sp    # make room for result on stack
      pea   (%sp)       # pass: result addr on stack
      mov.l %d0,-(%sp)  # pass: multiplicand on stack
      mov.l _multiplier,-(%sp) # pass: multiplier on stack
      bsr.l _060LISP_TOP+0x18 # branch to multiply routine
      add.l &0xc,%sp    # clear arguments from stack
      mov.l (%sp)+,%d1  # load result[63:32]
      mov.l (%sp)+,%d0  # load result[31:0]
```


For a divide:

```
# divu.l <ea>,Dr:Dq
# divu.l _divisor,%d1:%d0

      subq.l      &0x8,%sp    # make room for result on stack
      pea   (%sp)       # pass: result addr on stack
      mov.l %d0,-(%sp)  # pass: dividend hi on stack
      mov.l %d1,-(%sp)  # pass: dividend hi on stack
      mov.l _divisor,-(%sp) # pass: divisor on stack
      bsr.l _060LISP_TOP+0x08 # branch to divide routine
      add.l &0xc,%sp    # clear arguments from stack
      mov.l (%sp)+,%d1  # load remainder
      mov.l (%sp)+,%d0  # load quotient
```

The library routines also return the correct condition code
register value. If this is important, then the caller of the library
routine must make sure that the value isn't lost while popping
other items off of the stack.

An example of using the "cmp2" instruction is as follows:

```
# cmp2.l <ea>,Rn
# cmp2.l _bounds,%d0

      pea   _bounds           # pass ptr to bounds
      mov.l %d0,-(%sp)  # pass Rn
```

```
      bsr.l _060LSP_TOP_+0x48 # branch to "cmp2" routine
      mov.w %cc,_tmp    # save off condition codes
      addq.l      &0x8,%sp    # clear arguments from stack

Exception reporting:
--------------------
If the instruction being emulated is a divide and the source
operand is a zero, then the library routine, as its last
instruction, executes an implemented divide using a zero
source operand so that an "Integer Divide-by-Zero" exception
will be taken. Although the exception stack frame will not
point to the correct instruction, the user will at least be able
to record that such an event occurred if desired.

68060ILSP entry points:
-----------------------
_060ILSP_TOP:
0x000:      _060LSP__idivs64_
0x008:      _060LSP__idivu64_

0x010:      _060LSP__imuls64_
0x018:      _060LSP__imulu64_

0x020:      _060LSP__cmp2_Ab_
0x028:      _060LSP__cmp2_Aw_
0x030:      _060LSP__cmp2_Al_
0x038:      _060LSP__cmp2_Db_
0x040:      _060LSP__cmp2_Dw_
0x048:      _060LSP__cmp2_Dl_
```