

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

M68060 Software Package Copyright © 1993, 1994 Motorola Inc. All rights reserved.

THE SOFTWARE is provided on an "AS IS" basis and without warranty. To the maximum extent permitted by applicable law, MOTOROLA DISCLAIMS ALL WARRANTIES WHETHER EXPRESS OR IMPLIED, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE and any warranty against infringement with regard to the SOFTWARE (INCLUDING ANY MODIFIED VERSIONS THEREOF) and any accompanying written materials.

To the maximum extent permitted by applicable law, IN NO EVENT SHALL MOTOROLA BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER PECUNIARY LOSS) ARISING OF THE USE OR INABILITY TO USE THE SOFTWARE. Motorola assumes no responsibility for the maintenance and support of the SOFTWARE.

You are hereby granted a copyright license to use, modify, and distribute the SOFTWARE

so long as this entire notice is retained without alteration in any modified and/or

redistributed versions, and that such modified versions are clearly identified as such.

No licenses are granted by implication, estoppel or otherwise under any patents or trademarks of Motorola, Inc.

~~~~~  
68060 INTEGER SOFTWARE PACKAGE (kernel version)  
-----

The file isp.sa contains the 68060 Integer Software Package. This package is essentially an exception handler that can be integrated into an operating system to handle the "Unimplemented Integer Instruction" exception vector #61. This exception is taken when any of the integer instructions not hardware implemented on the 68060 are encountered. The isp.sa provides full emulation support for these instructions.

The unimplemented integer instructions are:

- 64-bit divide
- 64-bit multiply
- movep
- cmp2
- chk2
- cas (w/ a misaligned effective address)
- cas2

Release file format:

-----  
The file isp.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 `isp.s` has also been included but only for documentation purposes.

Release file structure:

-----

(top of module)

```
-----
|                               | - 128 byte-sized section
(1) |   Call-Out               | - 4 bytes per entry (user fills these in)
|                               | - example routines in iskeleton.s
-----
|                               | - 8 bytes per entry
(2) |   Entry Point           | - user does a "bra" or "jmp" to this address
|                               |
-----
|                               | - code section
(3) ~ ~ ~ ~ ~
|                               |
-----
```

(bottom of module)

The first section of this module is the "Call-out" section. This section is NOT INCLUDED in `isp.sa` (an example "Call-out" section is provided at the end of the file `iskeleton.s`). The purpose of this section is to allow the ISP routines to reference external functions that must be provided by the host operating system. This section MUST be exactly 128 bytes in size. There are 32 fields, each 4 bytes in size. Each field corresponds to a function required by the ISP (these functions and their location are listed in "68060ISP call-outs" below). Each field entry should contain the address of the corresponding function RELATIVE to the starting address of the "call-out" section. The "Call-out" section must sit adjacent to the `isp.sa` image in memory.

The second section, the "Entry-point" section, is used by external routines to access the functions within the ISP. Since the `isp.sa` hex file contains no symbol names, this section contains function entry points that are fixed with respect to the top of the package. The currently defined entry-points are listed in section "68060 ISP entry points" below. A calling routine would simply execute a "bra" or "jmp" that jumped to the selected function entry-point.

For example, if the 68060 hardware took a "Unimplemented Integer Instruction" exception (vector #61), the operating system should execute something similar to:

```
bra    _060ISP_TOP+128+0
```

(`_060ISP_TOP` is the starting address of the "Call-out" section; the "Call-out" section is 128 bytes long; and the Unimplemented Integer ISP handler entry point is located 0 bytes from the top of the "Entry-point" section.)

The third section is the code section. After entering through an "Entry-point", the entry code jumps to the appropriate emulation code within the code section.

68060ISP call-outs: (details in `iskeleton.s`)

```

-----
0x000:    _060_real_chk
0x004:    _060_real_divbyzero
0x008:    _060_real_trace
0x00c:    _060_real_access
0x010:    _060_isp_done

0x014:    _060_real_cas
0x018:    _060_real_cas2
0x01c:    _060_real_lock_page
0x020:    _060_real_unlock_page

0x024:    (Motorola reserved)
0x028:    (Motorola reserved)
0x02c:    (Motorola reserved)
0x030:    (Motorola reserved)
0x034:    (Motorola reserved)
0x038:    (Motorola reserved)
0x03c:    (Motorola reserved)

0x040:    _060_imem_read
0x044:    _060_dmem_read
0x048:    _060_dmem_write
0x04c:    _060_imem_read_word
0x050:    _060_imem_read_long
0x054:    _060_dmem_read_byte
0x058:    _060_dmem_read_word
0x05c:    _060_dmem_read_long
0x060:    _060_dmem_write_byte
0x064:    _060_dmem_write_word
0x068:    _060_dmem_write_long

0x06c:    (Motorola reserved)
0x070:    (Motorola reserved)
0x074:    (Motorola reserved)
0x078:    (Motorola reserved)
0x07c:    (Motorola reserved)

```

68060ISP entry points:

```

-----
0x000:    _060_isp_unimp

0x008:    _060_isp_cas
0x010:    _060_isp_cas2
0x018:    _060_isp_cas_finish
0x020:    _060_isp_cas2_finish
0x028:    _060_isp_cas_inrange
0x030:    _060_isp_cas_terminate
0x038:    _060_isp_cas_restart

```

Integrating cas/cas2:

-----  
The instructions "cas2" and "cas" (when used with a misaligned effective address) take the Unimplemented Integer Instruction exception. When the 060ISP is installed properly, these instructions will enter through the \_060\_isp\_unimp() entry point of the ISP.

After the 060ISP decodes the instruction type and fetches the appropriate data registers, and BEFORE the actual emulated transfers occur, the package calls either the "Call-out" \_060\_real\_cas() or \_060\_real\_cas2(). If the emulation code provided by the 060ISP is sufficient for the host system (see isp.s source code), then these "Call-out"s should be made, by the system integrator, to point directly back into the package through the "Entry-point"s \_060\_isp\_cas() or \_060\_isp\_cas2().

Assignment Project Exam Help  
<https://powcoder.com>  
Add WeChat powcoder

One other necessary action by the integrator is to supply the routines `_060_real_lock_page()` and `_060_real_unlock_page()`. These functions are defined further in `iskeleton.s` and the 68060 Software Package Specification.

If the "core" emulation routines of either "cas" or "cas2" perform some actions which are too system-specific, then the system integrator must supply new emulation code. This new emulation code should reside within the functions `_060_real_cas()` or `_060_real_cas2()`. When this new emulation code has completed, then it should re-enter the 060ISP package through the "Entry-point" `_060_isp_cas_finish()` or `_060_isp_cas2_finish()`. To see what the register state is upon entering `_060_real_cas()` or `_060_real_cas2()` and what it should be upon return to the package through `_060_isp_cas_finish()` or `_060_isp_cas2_finish()`, please refer to the source code in `isp.s`.

Miscellaneous:

-----

`_060_isp_unimp:`

-----

- documented in 2.2 in spec.

- Basic flow:

exception taken ---> enter `_060_isp_unimp` --|

|

may exit through `_060_real_itrace` <----|

or |

may exit through `_060_real_chk` <----|

or |

may exit through `_060_real_divbyzero` <----|

or |

may exit through `_060_isp_done` <----|

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder