Welcome to E-XFL.COM

Understanding Embedded - Microprocessors

Embedded microprocessors are specialized computing chips designed to perform specific tasks within an embedded system. Unlike general-purpose microprocessors found in personal computers, embedded microprocessors are tailored for dedicated functions within larger systems, offering optimized performance, efficiency, and reliability. These microprocessors are integral to the operation of countless electronic devices, providing the computational power necessary for controlling processes, handling data, and managing communications.

Applications of Embedded - Microprocessors

Embedded microprocessors are utilized across a broad spectrum of applications, making them indispensable in

Details	
Product Status	Obsolete
Core Processor	68060
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	50MHz
Co-Processors/DSP	-
RAM Controllers	-
Graphics Acceleration	No
Display & Interface Controllers	-
Ethernet	-
SATA	-
USB	-
Voltage - I/O	3.3V
Operating Temperature	0°C ~ 70°C (TA)
Security Features	-
Package / Case	206-BPGA
Supplier Device Package	206-PGA (47.25x47.25)
Purchase URL	https://www.e-xfl.com/product-detail/nxp-semiconductors/mc68060rc50

PREFACE

The complete documentation package for the MC68060, MC68LC060, and MC68EC060 (collectively called M68060) consists of the M68060UM/AD, *M68060 User's Manual*, and the M68000PM/AD, *M68000 Family Programmer's Reference Manual*. The *M68060 User's Manual* describes the capabilities, operation, and programming of the M68060 superscalar 32-bit microprocessors. The *M68000 Family Programmer's Reference Manual* contains the complete instruction set for the M68000 family.

The introduction of this manual includes general information concerning the MC68060 and summarizes the differences among the M68060 family devices. Additionally, appendices provide detailed information on how these M68060 derivatives operate differently from the MC68060.

When reading this manual, disregard information concerning the floating-point unit in reference to the MC68LC060, and disregard information concerning the floating-point unit and memory management unit in reference to the MC68EC060.

The organization of this manual is as follows:

Section 1	Introduction
Section 2	Signal Description
Section 3	Integer Unit
Section 4	Memory Management Unit
Section 5	Caches
Section 6	Floating-Point Unit
Section 7	Bus Operation
Section 8	Exception Processing
Section 9	IEEE 1149.1 Test (JTAG) and Debug Pipe Control Modes
Section 10	Instruction Timings
Section 11	Applications
Section 12	Electrical and Thermal Characteristics
Section 13	Ordering Information and Mechanical Data
Appendix A	MC68LC060
Appendix B	MC68EC060
Appendix C	MC68060 Software Package
Appendix D	M68060 Instructions

Table 1-3. Instruction Set Summary (Continued)

Opcode	Operation	Syntax

NOTES:
1.Where d is direction, left or right.
2.Emulation support only, not supported in hardware.
3.Where r is rounding precision, single or double precision.
4.List refers to register.
5.List refers to control registers only.
6.MOVE16 (ax)+,(ay)+ is functionally the same as MOVE16 (ax),(ay)+ when ax = ay. The address register is only incremented once, and the line is copied over itself rather than to the next line.
7.Not available for the MC68EC060.
8.Emulation support for misaligned operands.
9.Emulation support for FMCVEM with dynamic register list.

1.10 NOTATIONAL CONVENTIONS

Table 1-4 lists the notation conventions used throughout this manual.

Table 1-4. Notational Conventions

Single- And Double-Operand Operations	
+	Arithmetic addition or postincrement indicator.
–	Arithmetic subtraction or predecrement indicator.
×	Arithmetic multiplication.
÷	Arithmetic division or conjunction symbol.
~	Invert; operand is logically complemented.
•	Logical AND
+	Logical OR
⊕	Logical exclusive OR
˘	Source operand is moved to destination operand.
–˘	Two operands are exchanged.
<op>	Any double-operand operation.
<operand>tested	Operand is compared to zero and the condition codes are set appropriately.
sign-extended	All bits of the upper portion are made equal to the high-order bit of the lower portion.
Other Operations	
TRAP	Equivalent to Format ÷ Offset Word ˘ (SSP); SSP – 2 ˘ SSP; PC ˘ (SSP); SSP – 4 ˘ SSP; SR ˘ (SSP); SSP – 2 ˘ SSP; (Vector) ˘ PC
STOP	Enter the stopped state, waiting for interrupts.
<operand>$_{10}$	The operand is BCD; operations are performed in decimal.
If <condition> then <operations> else <operations>	Test the condition. If true, the operations after “then” are performed. If the condition is false and the optional “else” clause is present, the operations after “else” are performed. If the condition is false and else is omitted, the instruction performs no operation. Refer to the Bcc instruction description as an example.
Register Specification	
An	Any Address Register n (example: A3 is address register 3)
Ax, Ay	Source and destination address registers, respectively.
BR	Base Register—An, PC, or suppressed.
Dc	Data register D7–D0, used during compare.
Dh, Dl	Data registers high- or low-order 32 bits of product.
Dn	Any Data Register n (example: D5 is data register 5)
Dr, Dq	Data register’s remainder or quotient of divide.
Du	Data register D7–D0, used during update.
Dx, Dy	Source and destination data registers, respectively.
MRn	Any Memory Register n.

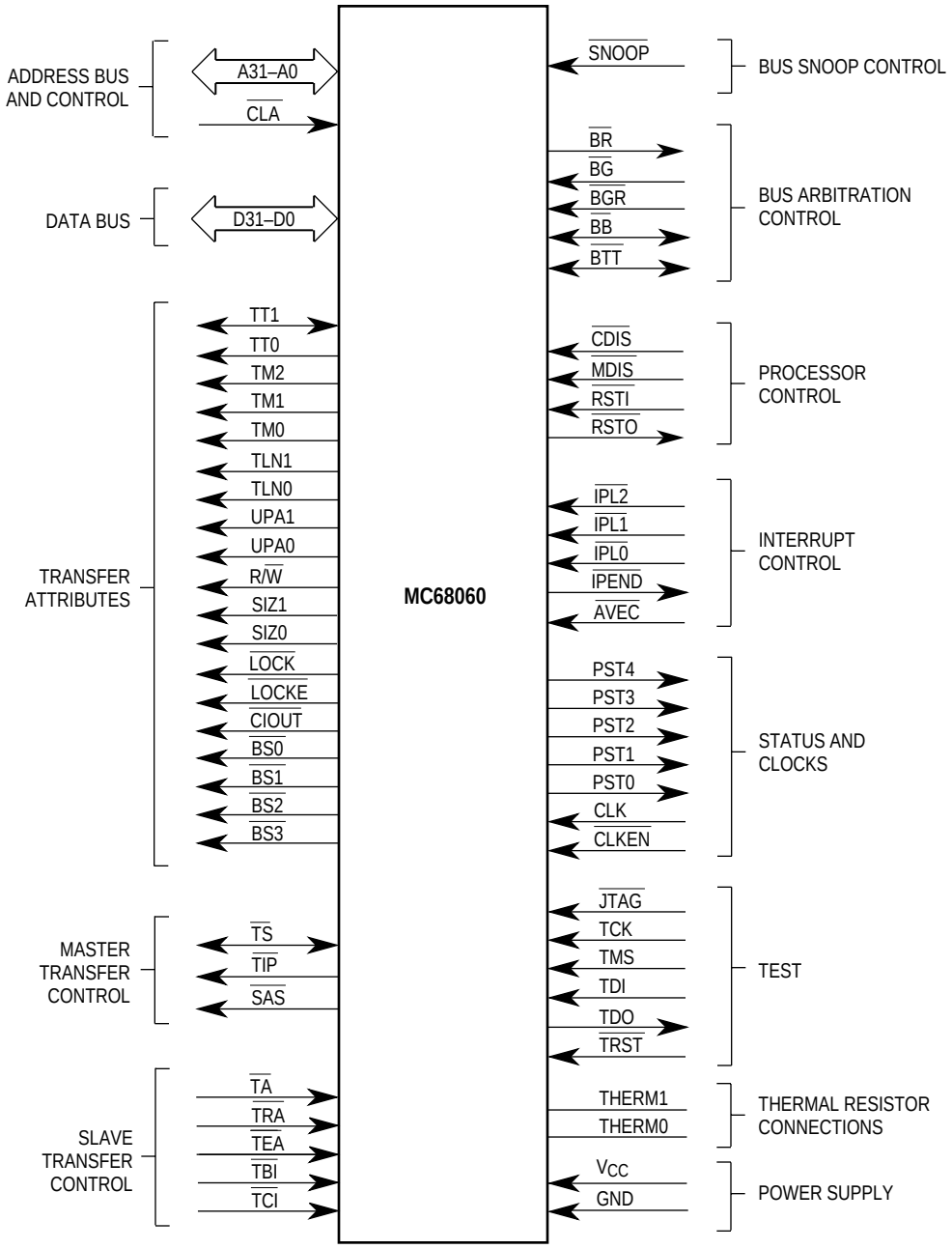


Figure 2-1. Functional Signal Groups

2.1 ADDRESS AND CONTROL SIGNALS

The following paragraphs describe the MC68060 address and control signals.

2.1.1 Address Bus (A31–A0)

These three-state bidirectional signals provide the address of the first item of a bus transfer (except for interrupt acknowledge transfers) when the MC68060 is the bus master. When an alternate bus master is controlling the bus and asserts the $\overline{\text{SNOOP}}$ signal, the address sig-

2.11.4 Test Data In (TDI)

This input signal provides a serial data input to the TAP. TDI should be tied to V_{CC} if it is not used and emulation mode is not to be used.

2.11.5 Test Data Out (TDO)

This three-state output signal provides a serial data output from the TAP. The TDO output can be placed in a high-impedance mode to allow parallel connection to board-level test data paths.

2.11.6 Test Reset ($\overline{\text{TRST}}$)

This input signal provides an asynchronous reset of the TAP controller. $\overline{\text{TRST}}$ should be grounded if 1149.1 operation is not to be used.

2.12 THERMAL SENSING PINS (THERM1, THERM0)

THERM1 and THERM0 are connected to an internal thermal resistor and provide information about the average temperature of the die. The resistance across these two pins is proportional to the average temperature of the die. The temperature coefficient of the resistor is approximately 1.2 Ω/°C with a nominal resistance of 400Ω at 25°C.

2.13 POWER SUPPLY CONNECTIONS

The MC68060 requires connection to a V_{CC} power supply, positive with respect to ground. The V_{CC} and ground connections are grouped to supply adequate current to the various sections of the processor. **Section 13 Ordering Information and Mechanical Data** describes the groupings of the V_{CC} and ground connections.

2.14 SIGNAL SUMMARY

Table 2-8 provides a summary of the electrical characteristics of the MC68060 signals.

4.3 ADDRESS TRANSLATION CACHES

The ATCs in the MMUs are four-way set-associative caches that each store 64 logical-to-physical address translations and associated page information similar in form to the corresponding page descriptors in memory. The purpose of the ATC is to provide a fast mechanism for address translation by avoiding the overhead associated with a table search of the logical-to-physical mapping of recently used logical addresses. Figure 4-19 illustrates the organization of the ATC.

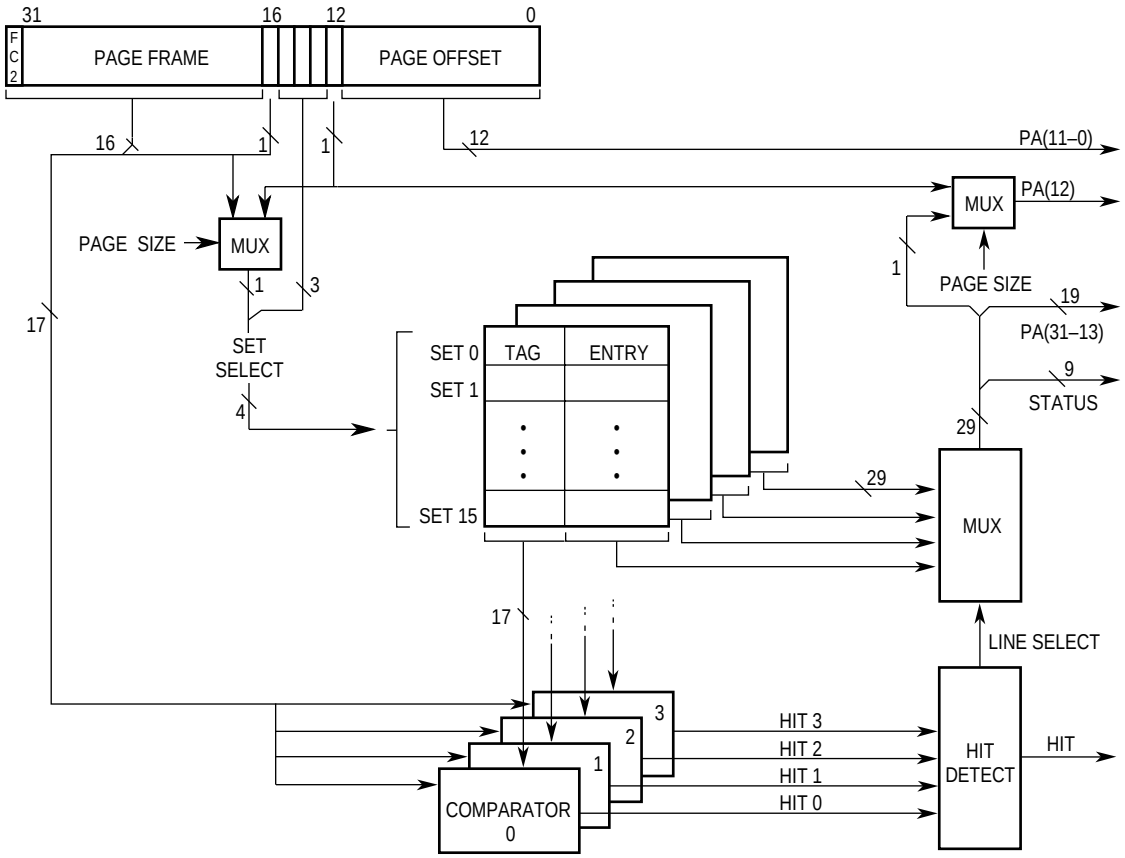


Figure 4-19. ATC Organization

Each ATC entry consists of a physical address, attribute information from a corresponding page descriptor, and a tag that contains a logical address and status information. Figure 4-20, which illustrates the entry and tag fields, is followed by field definitions listed in alphabetical order.

vide a mechanism that allows the instruction fetch pipeline to detect and change instruction streams before the change-of-flow instructions enter an operand execution pipeline.

The branch cache implementation is made up of a five-state prediction model based on past execution history, in addition to the current program counter/branch target virtual address association logic.

For each instruction fetch address generated, the branch cache is examined to see if a valid branch entry is present. If there is not a branch cache hit, the instruction fetch unit continues to fetch instructions sequentially. If a branch cache hit occurs indicating a “taken branch”, the instruction fetch unit discards the current instruction steam and begins fetching at the location indicated by the branch target address. As long as the branch cache prediction is correct, which happens a very significant percentage of the time, the change-of-flow of the instruction stream is “invisible” to the OEP and performance is maximized. If the branch cache prediction is wrong, the internal pipelines are “cancelled” and the correct instruction flow is established.

The branch cache must be cleared by the operating system on all context switches (using the MOVEC to CACR instruction), because it is virtually-mapped.

The branch cache is automatically cleared by the hardware as part of any cache invalidate (CINV) or any cache push and invalidate (CPUSH) instruction operating on the instruction cache.

Programs that use the TRAPF instruction extension word as a possible branch target destination intefere with proper operation of the branch target cache, resulting in an access error exception. This condition is indicated by the BPE bit in the FSLW of the access error stack.

5.12 CACHE OPERATION SUMMARY

The instruction and data caches function independently when servicing access requests from the integer unit. The following paragraphs discuss the operational details for the caches and present state diagrams depicting the cache line state transitions.

5.12.1 Instruction Cache

The integer unit uses the instruction cache to store instruction prefetches as it requests them. Instruction prefetches are normally requested from sequential memory locations except when a change of program flow occurs (e.g., a branch taken) or when an instruction that can modify the status register (SR) is executed, in which case the instruction pipe is automatically flushed and refilled. The instruction cache supports a line-based protocol that allows individual cache lines to be in either the invalid or valid states.

For instruction prefetch requests that hit in the cache, the long word containing the instruction is places onto the internal instruction data bus. When an access misses in the cache, the cache controller requests the line containing the required data from memory and places it in the cache. If available, an invalid line is selected and updated with the tag and data from memory. The line state then changes from invalid to valid by setting the V-bit. If all lines in the set are already valid, a pseudo round-robin replacement algorithm is used to select one

6.1.3.1 FLOATING-POINT CONDITION CODE BYTE. The FPCC byte (see Figure 6-4) contains four condition code bits that are set at the end of all arithmetic instructions involving the floating-point data registers. These bits are sign of mantissa (N), zero (Z), infinity (I), and NAN. The FMOVE FPm,<ea>, FMOVEM FPm, and FMOVE FPCR instructions do not affect the FPCC.

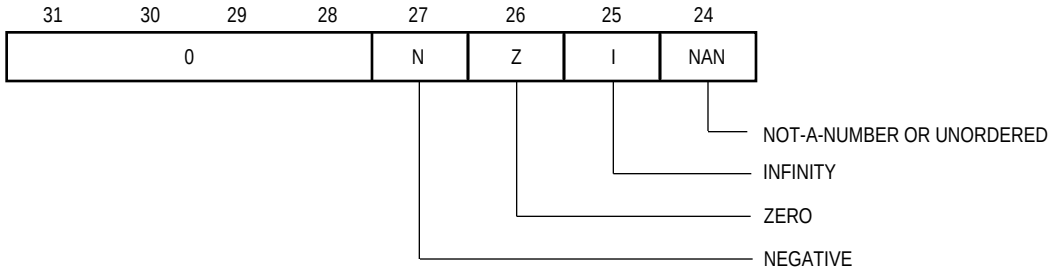


Figure 6-4. Floating-Point Condition Code (FPSR)

To aid programmers of floating-point subroutine libraries, the MC68060 implements the four FPCC bits in hardware instead of only implementing the four IEEE conditions. An instruction derives the IEEE conditions when needed. For example, the programmers of a complex arithmetic multiply subroutine usually prefer to handle special data types, such as zeros, infinities, or NANs, separately from normal data types. The floating-point condition codes allow users to efficiently detect and handle these special values.

6.1.3.2 QUOTIENT BYTE. The quotient byte (see Figure 6-5) provides compatibility with the MC68881/MC68882. This byte is set at the completion of the modulo (FMOD) or IEEE remainder (FREM) instruction, and contains the seven least significant bits of the unsigned quotient as well as the sign of the entire quotient.

The quotient bits can be used in argument reduction for transcendentals and other functions. For example, seven bits are more than enough to determine the quadrant of a circle in which an operand resides. The quotient field (bits 22–16) remains set until the user clears it.

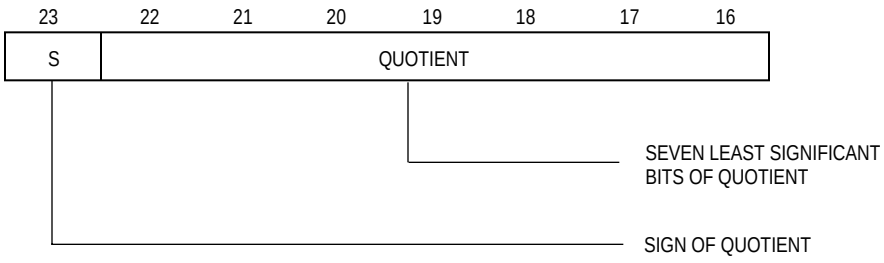


Figure 6-5. Floating-Point Quotient Byte (FPSR)

6.1.3.3 EXCEPTION STATUS BYTE. The EXC byte (see Figure 6-6) contains a bit for each floating-point exception that can occur during the most recent arithmetic instruction or move operation. The start of most operations clears this byte; however, operations that cannot generate floating-point exceptions (the FMOVEM and FMOVE control register instructions) do not clear this byte. An exception handler can use this byte to determine which floating-point exception(s) caused a trap.

The FPU performs all floating-point internal operations in extended precision. It supports mixed-mode arithmetic by converting single- and double-precision operands to extended-precision values before performing the specified operation. The FPU converts all memory data formats to extended precision before using it in a floating-point operation or loading it in a floating-point data register. The FPU also converts extended-precision data formats in a floating-point data register to any data format and either stores it in a memory destination or in an integer data register.

If the external operand is a denormalized number or unnormalized number, the number is normalized before an operation is performed. However, an external denormalized number moved into a floating-point data register is stored as a denormalized number.

If an external operand is an unnormalized number, the number is normalized before it is used in an arithmetic operation. If the external operand is an unnormalized zero (i.e., with a mantissa of all zeros), the number is converted to a normalized zero before the specified operation is performed. The regular use of unnormalized inputs not only defeats the purpose of the IEEE 754 standard, but also can produce gross inaccuracies in the results.

6.3.1 Intermediate Result

Figure 6-8 illustrates the intermediate result format. The intermediate result's exponent for some dyadic operations (e.g., multiply and divide) can easily overflow or underflow the 15-bit exponent of the destination floating-point register. To simplify the overflow and underflow detection, intermediate results in the FPU maintain a 16-bit, twos-complement integer exponent. Detection of an overflow or underflow intermediate result always converts the 16-bit exponent into a 15-bit biased exponent before being stored in a floating-point data register. The FPU internally maintains the 67-bit mantissa for rounding purposes. The mantissa is always rounded to 64 bits (or less, depending on the selected rounding precision) before it is stored in a floating-point data register.

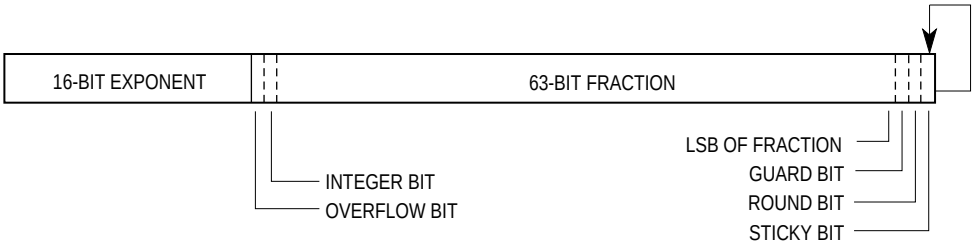


Figure 6-8. Intermediate Result Format

If the destination is a floating-point data register, the result is in the extended-precision format and is rounded to the precision specified by the FPCR PREC bits before being stored. All mantissa bits beyond the selected precision are zero. If the single- or double-precision mode is selected, the exponent value is in the correct range even if it is stored in extended-precision format. If the destination is a memory location, the FPCR PREC bits are ignored. In this case, a number in the extended-precision format is taken from the source floating-point data register, rounded to the destination format precision, and then written to memory.

If no underflow occurs, the intermediate result is rounded according to the user-selected rounding precision and rounding mode. After rounding, the INEX2 bit of the FPSR EXC byte is set accordingly. Finally, the magnitude of the result is checked to see if it is too large to be represented in the current rounding precision. If so, the OVFL bit of the FPSR EXC byte is set, and the MC68060 takes a nonmaskable overflow exception and executes the M68060SP overflow exception handler. The M68060SP returns a correctly signed infinity or a correctly signed largest normalized number, depending on the rounding mode in effect.

6.4.2 Conditional Testing

Unlike the integer arithmetic condition codes, an instruction either always sets the floating-point condition codes in the same way or it does not change them at all. Therefore, the instruction descriptions do not include floating-point condition code settings. The following paragraphs describe how floating-point condition codes are set for all instructions that modify condition codes. Refer to **6.1.3.1 Floating-Point Condition Code Byte** for a description of the FPCC byte.

The data type of the operation's result determines how the four condition code bits are set. Table 6-8 lists the condition code bit setting for each data type. The MC68060 generates only eight of the 16 possible combinations. Loading the FPCC with one of the other combinations and executing a conditional instruction can produce an unexpected branch condition.

Table 6-8. Floating-Point Condition Code Encoding

Data Type	N	Z	I	NAN
+ Normalized or Denormalized	0	0	0	0
– Normalized or Denormalized	1	0	0	0
+ 0	0	1	0	0
– 0	1	1	0	0
+ Infinity	0	0	1	0
– Infinity	1	0	1	0
+ NAN	0	0	0	1
– NAN	1	0	0	1

The inclusion of the NAN data type in the IEEE floating-point number system requires each conditional test to include the NAN condition code bit in its Boolean equation. Because a comparison of a NAN with any other data type is unordered (i.e., it is impossible to determine if a NAN is bigger or smaller than an in-range number), the compare instruction sets the NAN condition code bit when an unordered compare is attempted. All arithmetic instructions also set the FPCC NAN bit if the result of an operation is a NAN. The conditional instructions interpret the NAN condition code bit equal to one as the unordered condition.

The IEEE 754 standard defines four conditions: equal to (EQ), greater than (GT), less than (LT), and unordered (UN). In addition, the standard only requires the generation of the condition codes as a result of a floating-point compare operation. The FPU tests for these conditions and 28 others at the end of any operation affecting the condition codes. For purposes of the floating-point conditional branch, set byte on condition, decrement and branch on condition, and trap on condition instructions, the MC68060 logically combines the four FPCC bits to form 32 conditional tests. The 32 conditional tests are separated into two groups—16

Table 6-11. Unimplemented Instructions

Monadic Operations	
FACOS	FLOGN
FASIN	FLOGNP1
FATAN	FMOVECR
FATANH	FSIN
FCOS	FSINCOS
FCOSH	FSINH
FETOX	FTAN
FETOXM1	FTANH
FGETEXP	FTENTOX
FGETMAN	FTWOTOX
FLOG10	FLOG2
Dyadic Operations	
FMOD	FREM
FSCALE	—
Miscellaneous	
FTRAPcc	FDBcc
FScc	—
Unimplemented Effective Address	
FMOVEM.X (dynamic register list)	FMOVEM.L #immediate, list of 2 or 3 control registers
F<op>.X #immediate,FPn	F<op>.P #immediate,FPn

A floating-point unimplemented instruction exception occurs when the processor attempts to execute an instruction word pattern that begins with $F, the processor recognizes this bit pattern as an MC68881 instruction, the FPU is enabled via the processor control register (PCR), but the floating-point instruction is not implemented in the MC68060 FPU. This exception corresponds to vector number 11 and shares this vector with the floating-point disabled and the unimplemented F-line exceptions. A stack frame of type 2 is generated when this exception is reported. The stacked PC points to the logical address of the next instruction after the floating-point instruction. In addition, the effective address of the floating-point operand in memory (if any) is calculated and stored in the effective address field.

When an unimplemented floating-point instruction is encountered, the processor waits for all previous floating-point instructions to complete execution. Pending exceptions are taken and handled prior to the execution of the unimplemented instruction.

The processor begins exception processing for the unimplemented floating-point instruction by making an internal copy of the current status register (SR). The processor then enters the supervisor mode and clears the trace bit. The processor creates a format $2 stack frame and saves the vector offset, PC, internal copy of the SR, and calculated effective address in the stack frame. The saved PC value is the logical address of the instruction that follows the unimplemented floating-point instruction. The processor generates exception vector number 11 for the unimplemented F-line instruction exception vector, fetches the address of the F-line exception handler from the processor's exception vector table, pushes the format $2 stack frame on the system stack, and begins execution of the exception handler after prefetching instructions to fill the pipeline.

provided for compatibility with existing MC68040-based ASICs and logic. This arbitration protocol uses the $\overline{BR}$, $\overline{BG}$, and $\overline{BB}$ signals. Bus tenure terminated ($\overline{BTT}$) must be ignored by the external arbiter and pulled high using a separate pullup resistor on the MC68060 pin when using this arbitration protocol.

In addition to the MC68040-arbitration protocol, a high speed MC68060-arbitration protocol is introduced to provide arbitration activity at higher frequencies. This arbitration protocol uses the $\overline{BR}$, $\overline{BG}$, $\overline{BTT}$, and $\overline{BGR}$ signals. $\overline{BB}$ must be ignored by the external arbiter and pulled high using a separate pullup resistor on the MC68060 when using this arbitration protocol.

In either arbitration protocol, the bus arbitration unit in the MC68060 operates synchronously and transitions between states in which CLK is enabled via $\overline{CLKEN}$ asserted (on the rising edge of BCLK). Either arbitration protocol allows arbitration to overlap with bus activity, but the MC68040-arbitration protocol should not be used at full bus speed. With either arbitration protocol, each master which can initiate bus cycles must have their $\overline{TS}$ signals connected together so that the MC68060 can maintain proper internal state. Note also, when using the MC68040-arbitration protocol, any alternate master which takes over bus ownership and initiates bus cycles with the assertion of $\overline{TS}$ must also assert $\overline{BB}$ for the time of its bus tenure.

7.11.1 MC68040-Arbitration Protocol ($\overline{BB}$ Protocol)

When using the MC68040-arbitration protocol, $\overline{BTT}$ must be pulled high through a resistor. Since $\overline{BTT}$ is also an output, a separate pullup resistor must be used exclusively for $\overline{BTT}$.

The MC68060 requests the bus from the external bus arbiter by asserting $\overline{BR}$ whenever an internal bus request is pending. The processor continues to assert $\overline{BR}$ for as long as it requires the bus. The processor negates $\overline{BR}$ at any time without regard to the status of $\overline{BG}$ and $\overline{BB}$. If the bus is granted to the processor when an internal bus request is generated, $\overline{BR}$ is asserted simultaneously with transfer start ($\overline{TS}$), allowing the access to begin immediately. The processor always drives $\overline{BR}$, and $\overline{BR}$ cannot be wire-ORed with other devices.

The external arbiter asserts $\overline{BG}$ to indicate to the processor that it has been granted the bus. If $\overline{BG}$ is negated while a bus cycle is in progress, the processor relinquishes the bus at the completion of the bus cycle, except on locked sequences in which $\overline{BGR}$ is negated. To guarantee that the bus is relinquished, $\overline{BG}$ must be negated prior to the rising edge of the BCLK in which the last $\overline{TA}$, $\overline{TEA}$, or $\overline{TRA}$ is asserted. Note that the bus controller considers the four long-word bus transfers of a burst-inhibited line transfer to be a single bus cycle and does not relinquish the bus until completion of the fourth transfer.

Unlike the MC68040 in which the read and write portions of a locked sequence is divisible, the MC68060 provides a choice via the $\overline{BGR}$ input. If $\overline{BGR}$ is asserted when $\overline{BG}$ is negated in the middle of a locked sequence, the MC68060 operates like the MC68040 and relinquishes the bus after the current bus cycle is completed. Otherwise, if $\overline{BGR}$ is negated when $\overline{BG}$ is negated, the MC68060 ignores the negated $\overline{BG}$, retains bus ownership, and completes all bus cycles of the locked sequence before giving up the bus. Systems may use the $\overline{BGR}$ input to assign severity of the $\overline{BG}$ negation. For instance, if bus arbitration is used to allow for DRAM refresh, it is okay to ignore locked sequences and force the MC68060 to

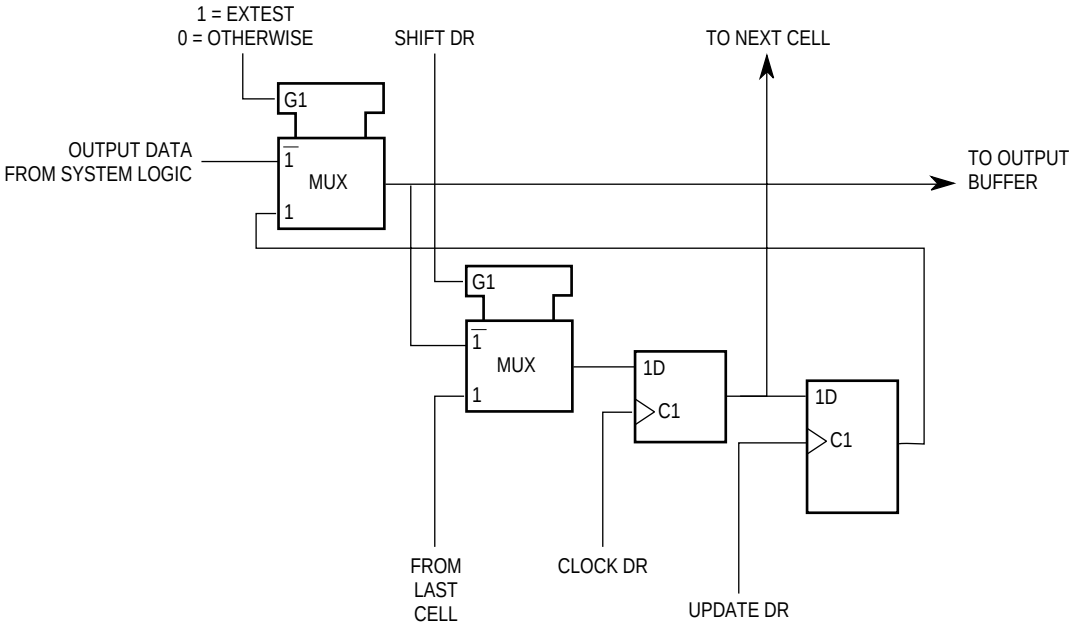


Figure 9-3. Output Pin Cell (O.Pin)

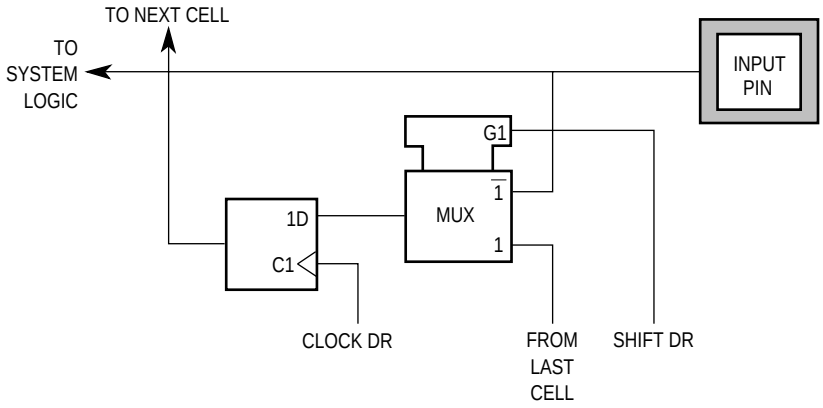


Figure 9-4. Observe-Only Input Pin Cell (I.Obs)

The $\overline{RAS}$ access time determines the number of wait states needed for the first memory access. The $\overline{RAS}$ access time is the time it takes between $\overline{RAS}$ being asserted and valid data coming out of the DRAM. The total available time for the first access is the time between the $\overline{TS}$ assertion and the first $\overline{TA}$ assertion. This time is equal to the clock period multiplied by the number of primary wait states. In addition to the $\overline{RAS}$ access time, the MC68060 input setup time and the $\overline{TS}$ to $\overline{RAS}$ propagation delay must also occur between the $\overline{TS}$ and $\overline{TA}$ signals. The following equation represents the number of wait states required for the primary memory access:

Wait States = ($\overline{RAS}$ propagation delay + $\overline{RAS}$ access time + Input Setup Time) / clock period

The following example assumes a $\overline{RAS}$ access time of 65 ns, an input setup time of 7 ns, and a $\overline{RAS}$ propagation delay of 5 ns. The processor is running at 50 MHz, so the clock period is 20 ns. The number of wait states required is (5ns + 65ns + 7ns) / 20 ns = 3.85 wait states. Therefore 4 wait states are required.

The $\overline{CAS}$ access time and the $\overline{CAS}$ precharge time determines the number of secondary wait states required. The $\overline{CAS}$ precharge time is the time that the $\overline{CAS}$ signal must remain negated between assertions. The total time available for the secondary access is the time between the first and second $\overline{TA}$ signals. This time is equal to the clock period multiplied by the number of secondary wait states. Since $\overline{CAS}$ must toggle during this time, two $\overline{CAS}$ propagation delays, the $\overline{CAS}$ precharge time, the $\overline{CAS}$ access time, and the MC68060 input setup time must occur during this time. Typically, the $\overline{CAS}$ precharge time is less than a clock period. Therefore an entire clock period is used to toggle $\overline{CAS}$. This leaves one $\overline{CAS}$ propagation delay time, a $\overline{CAS}$ access time, and the input setup time. This time must be less than the number of wait states less one multiplied by the clock period. The following equation represents the number of wait states required for the secondary memory accesses:

Wait States = [($\overline{CAS}$ propagation delay + $\overline{CAS}$ access time + input setup time) / clock period] + 1

The following example assumes a $\overline{CAS}$ access time of 20 ns, input setup time of 7 ns, and a $\overline{CAS}$ propagation delay of 5 ns. The clock period is 20 ns. The number of wait states required is [(5ns + 20ns + 7ns) / 20ns] + 1 = 2.6. Therefore three wait states are required. This first line burst transfer is a 5:3:3:3 transfer. For the primary transfer, an extra clock is added for the $\overline{TS}$ signal assertion.

In this example, a second line burst transfer occurs immediately following the first transfer. If the same DRAM chips are being accessed, $\overline{RAS}$ precharge time must be considered. $\overline{RAS}$ precharge time is the time that the $\overline{RAS}$ signal must remain high between assertions. In the example, $\overline{RAS}$ precharge time is 65 ns. Two additional wait states need to be added after the second $\overline{TS}$ to assure that the $\overline{RAS}$ precharge time is satisfied. Therefore, the second line burst transfer is a 7:3:3:3 transfer.

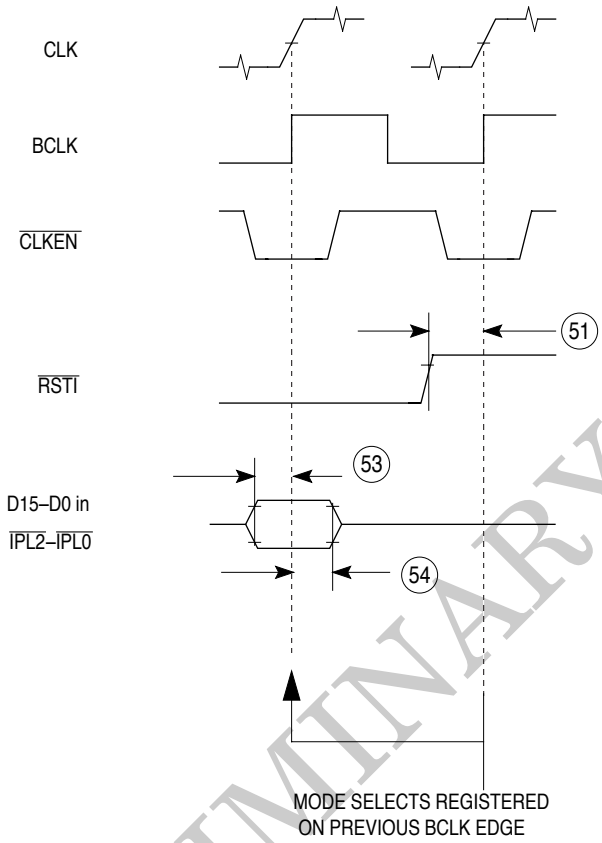
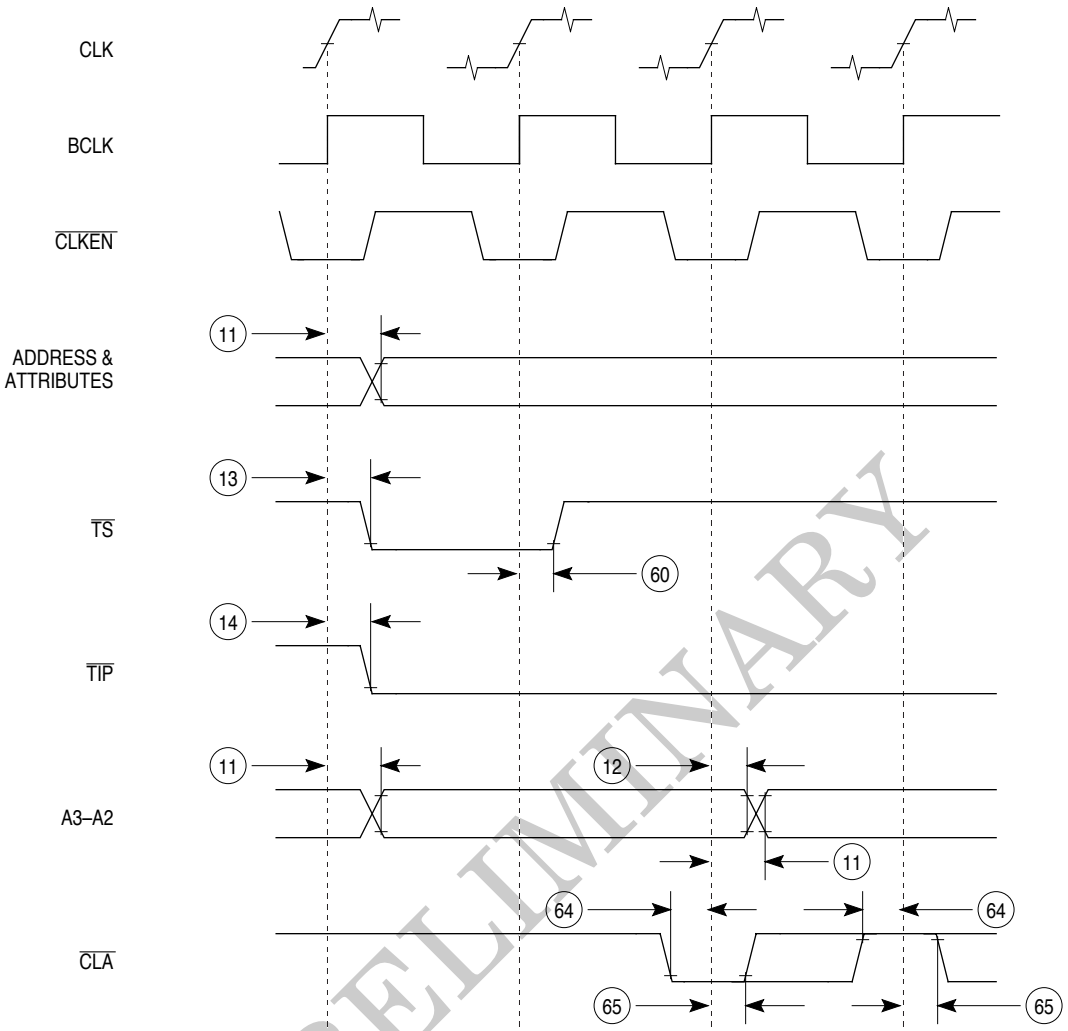


Figure 12-2. Reset Configuration Timing



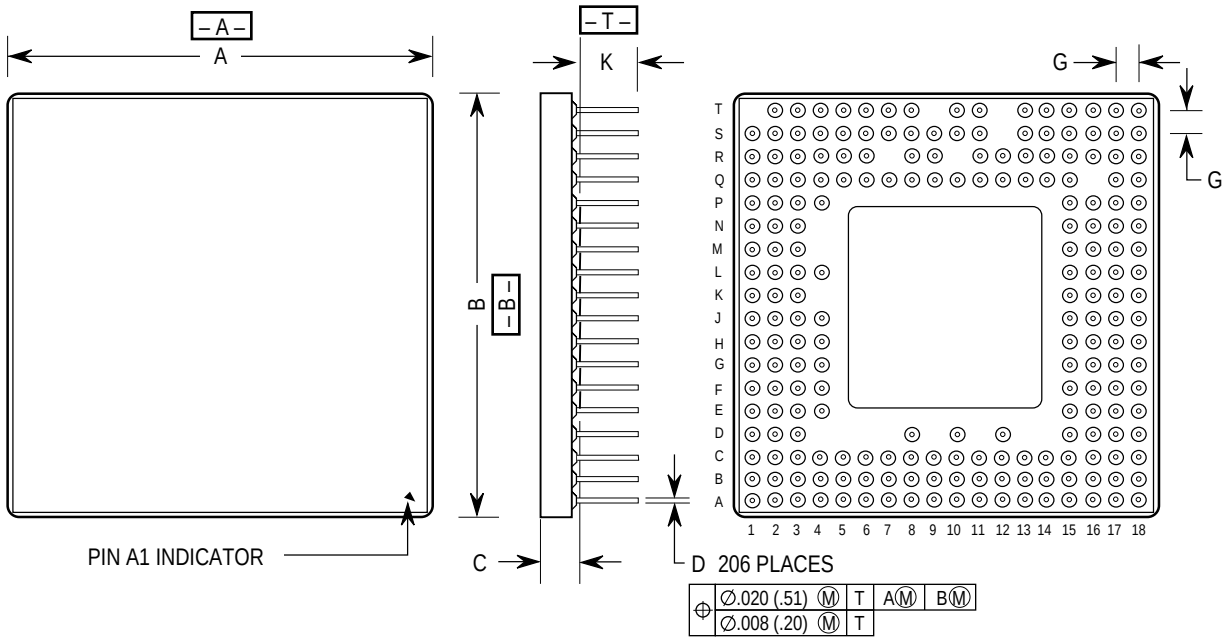
NOTE: Address and attributes refer to the following signals: A31-A0, SIZ1, SIZ0, R/W, TT1, TT0, TM2-TM0, TLN1, TLN

Figure 12-6. CLA Timing

13.3 MECHANICAL DATA

Figure 13-1 illustrates the MC68060, MC68LC060 and MC68EC060 PGA package dimensions. Figure 13-2 illustrates the MC68060, MC68LC060, and MC68EC060 CQFP package dimensions. Due to space limitation, Figure 13-2 is represented by a general (smaller) package outline rather than showing all 208 leads.

MC68060 PGA
CASE NUMBER: 993A-01



DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	46.74	47.75	1.840	1.880
B	46.74	47.75	1.840	1.880
C	2.79	3.05	0.110	0.140
D	0.41	0.51	0.016	0.020
G	2.54 BSC		0.100 BSC	
K	3.81	4.32	0.150	0.170

NOTES:
1. DIMENSIONS AND TOLERANCING PER ANSI Y14.5M 1982.
2. CONTROLLING DIMENSION: INCH

Figure 13-1. PGA Package Dimensions (RC Suffix)

C.2.2.1 UNIMPLEMENTED INTEGER INSTRUCTION EXCEPTION MODULE ENTRY POINTS. The _isp_unimp function is implemented such that the unimplemented integer instruction exception vector table entry typically points directly to _isp_unimp. If the system software chooses to perform operations prior to entering the _isp_unimp function, it may do so as long as the system stack points to the exception stack frame at the time of entry.

C.2.2.2 UNIMPLEMENTED INTEGER INSTRUCTION EXCEPTION MODULE CALL-OUTS. The call-outs _real_trace, _real_chk, _real_divbyzero, and _real_access are defined to provide the system integrator a choice of either having the module point directly to the actual trace, chk, divide-by-zero, and access error handler, or to an alternate routine that would fetch the address of the exception handler from the vector table prior to jumping to the actual handlers. The direct implementation is ideal for systems that do not anticipate any changes to the vector table and performance is more critical. The indirect approach of consulting the vector table is more accurate in that if the instruction were implemented, the actual handler's address is fetched from the appropriate vector table entry before branching there.

Other call-outs which are common to the floating-point kernel module are discussed in **C.4 Operating System Dependencies**. These call-outs include the discussion of the _real_access and other operating-system-dependent functions.

The _isp_done call-out is provided as a means for the system to do any clean-up, if any is necessary, before executing the RTE instruction to return to normal instruction execution. The unimplemented integer instruction exception handler will either branch to this call-out or create an appropriate exception frame and branch to the call-outs _real_trace, _real_chk, _real_divbyzero, or _real_access routines as outlined previously.

C.2.2.3 CAS MISALIGNED ADDRESS AND CAS2 EMULATION-RELATED CALL-OUTS AND ENTRY POINTS. The CAS instruction with misaligned address and CAS2 emulation is the most system dependent of all MC68060ISP code. The emulation code may require interaction with a system's interrupt, paging and access error recovery mechanisms. The emulation algorithm uses the MOVEC of the BUSCR register to assert the $\overline{\text{LOCK}}$ and $\overline{\text{LOCKE}}$ signals during emulation. The following is a description of the main steps in the emulation process:

1. Decode instruction and fetch all data from registers as necessary. In addition, if any of the operand pages are non-resident, then they must be paged in and not be allowed to be paged out or marked invalid for any reason until the emulation process ends. For each operand address, the MC68060ISP calls _real_lock_page() which must be provided by the host operating system to "lock" the pages. This routine should also check to see if the address passed is valid and writable. If not, then an error result should be returned to the MC68060ISP.
2. The MC68060ISP then calls the "core" emulation code for either "cas" or "cas2". The MC68060ISP references the "core" routines by calling either the _real_cas() or _real_cas2() call-outs. If the emulation code provided is sufficient for a given system, then the system integrator can make these call-outs immediately re-enter the package by calling either _isp_cas() or _isp_cas2() entry points.These entry points will perform the required emulation. If the "core" routines provided need to be replaced by a more

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

       subq.l  #$8,sp               ; make room for result on stack
       pea     (sp)                 ; pass: result addr on stack
       move.l  d0,-(sp)            ; pass: multiplicand on stack
       move.l   _multiplier,-(sp)   ; pass: multiplier on stack
       bsr.l   _060LISP_TOP+$18     ; branch to multiply routine
       add.l   #$c,sp              ; clear arguments from stack
       move.l  (sp)+,d1            ; load result[63:32]
       move.l  (sp)+,d0            ; load result[31:0]
```

Figure C-6. MUL Instruction Call Example

```
* cmp2.l <ea>,Rn
* cmp2.l _bounds,d0

       pea     _bounds              ; pass ptr to bounds
       move.l   d0,-(sp)             ; pass Rn
       bsr.l   _060LSP_TOP_+$48     ; branch to “cmp2” routine
       add.l   #$8,sp               ; clear arguments from stack
```

Figure C-7. CMP2 Instruction Call Example

divide and the source operand is a zero, then the library routine (as it is 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 can at least be able to record that such an event occurred.

C.3 FLOATING-POINT EMULATION PACKAGE (MC68060FPSP)

The MC68060 does not implement some floating-point instructions, addressing modes, and data types on-chip in order to streamline internal operations. This results in an overall system performance improvement at the expense of software emulation of these unimplemented instructions, addressing modes, and data types. The M68060SP provides three separate modules that are related to floating-point operations. The first floating-point module is the full floating-point kernel module. This module is used for applications that require emulation of the full MC68881 floating-point instruction set, data-types, and IEEE-754 exception handling. The second floating-point module is the floating-point library. This library is provided as a separate module for applications that need to avoid the latency incurred by the F-line exception processing for unimplemented floating-point instructions. However, this method requires recompiling of existing software to implement subroutine calls. The third floating-point module, the partial floating-point kernel module, is optional and is used primarily in systems that also integrate the floating-point library. The partial floating-point kernel module is similar in function to the full floating-point kernel except that it does not contain the unimplemented floating-point instruction exception handler. This module is provided for the purpose of saving memory space. Otherwise, the full floating-point kernel module must be used instead.

FRESTORE Restore Internal Floating-Point State (MC68060 only) FRESTORE

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	1	0	1	EFFECTIVE ADDRESS MODE REGISTER					

Instruction Field:

Effective Address field—Determines the addressing mode for the state frame. Only postincrement or control addressing modes can be used as listed in the following table:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
–(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

PLPA

Test a Logical Address

(MC68060, MC68LC060)

PLPA

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	1	1	R/W	0	0	1	ADDRESS REGISTER		

Instruction Fields:

R/W field—Specifies simulating a read or write bus transfer.
0—Write
1—Read

Register field—Specifies the address register containing the effective address for the instruction.