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	304-LBGA Exposed Pad
Supplier Device Package	304-TBGA (31x31)
Purchase URL	https://www.e-xfl.com/product-detail/nxp-semiconductors/mc68ec060zu50

Section 7
Bus Operation

List of Tables

The integer unit implements a subset of the MC68040 instruction set. The FPU implements a subset of the MC68881/2 coprocessor instruction set. The instruction and data memory units manage the ATCs and the instruction and data caches. The ATCs provide on-chip storage for the paged MMU's most recently used address translations. The data and instruction caches include the logic necessary to read, write, update, invalidate, and flush the caches. The bus controller manages the interface between the MMUs and the external bus. Snoop invalidation is supported to maintain cache consistency by monitoring the external bus when the processor is not the current master.

1.4.2 Integer Unit

The MC68060's integer unit carries out logical and arithmetic operations. The integer unit contains an instruction fetch controller, an instruction execution controller, and a branch target cache. The superscalar design of the MC68060 provides dual execution pipelines in the instruction execution controller, providing simultaneous execution.

The superscalar operation of the integer unit can be disabled in software, turning off the second execution pipeline for debugging. Disabling the superscalar operation also lowers performance and power consumption.

1.4.2.1 INSTRUCTION FETCH UNIT. The instruction fetch unit contains an instruction fetch pipeline and the logic that interfaces to the branch cache. The instruction fetch pipeline consists of four stages, providing the ability to prefetch instructions in advance of their actual use in the instruction execution controller. The continuous fetching of instructions keeps the instruction execution controller busy for the greatest possible performance. Every instruction passes through each of the four stages before entering the instruction execution controller. The four stages in the instruction fetch pipeline are:

1. Instruction Address Calculation (IAG)—The virtual address of the instruction is determined.
2. Instruction Fetch (IC)—The instruction is fetched from memory.
3. Early Decode (IED)—The instruction is pre-decoded for pipeline control information.
4. Instruction Buffer (IB)—The instruction and its pipeline control information are buffered until the integer execution pipeline is ready to process the instruction.

The branch cache plays a major role in achieving the performance levels of the MC68060. The concept of the branch cache is to provide a mechanism that allows the instruction fetch pipeline to detect and change the instruction stream before the change of flow affects the instruction execution controller.

The branch cache is examined for a valid branch entry after each instruction fetch address is generated in the instruction fetch pipeline. If a hit does not occur in the branch target cache, the instruction fetch pipeline continues to fetch instructions sequentially. If a hit occurs in the branch cache, indicating a branch taken instruction, the current instruction stream is discarded and a new instruction stream is fetched starting at the location indicated by the branch cache.

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.

3.2.2.4 ALTERNATE FUNCTION CODE REGISTERS. The alternate function code registers contain 3-bit function codes. Function codes can be considered extensions of the 32-bit logical address that optionally provides as many as eight 4-Gbyte address spaces. The processor automatically generates function codes to select address spaces for data and programs at the user and supervisor modes. Certain instructions use the SFC and DFC registers to specify the function codes for operations.

3.2.2.5 PROCESSOR CONFIGURATION REGISTER. The PCR is an 32-bit register which controls the operations of the MC68060 internal pipelines and contains a software readable revision number. The PCR is shown in Figure 3-5.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15 … 8	7	6 … 2	1	0
0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	Revision Number	EDEBUG	Reserved	DFP	ESS

Figure 3-5. Processor Configuration Register

Bits 31–16—Identification

These bits are configured with the value which identifies this device as an MC68060. These bits are ignored when writing to the PCR.

See **Appendix A MC68LC060** and **Appendix B MC68EC060** for MC68LC060 and MC68EC060, respectively, identification field values.

Bits 15–8—Revision Number

Bits 15–8 contain the 8-bit device revision number. The first revision is 00000000. These bits are ignored when writing to the PCR.

EDEBUG—Enable Debug Features

When this bit is set, the MC68060 outputs internal control information on the address bus (A31–A0) and data bus (D31–D0) during idle bus cycles. This capability is implemented to support debug of designs that include the MC68060. When this bit is cleared, operation proceeds in a normal manner and no internal information is output on idle bus cycles. This bit is cleared at reset.

Bits 6–2—Reserved by Motorola for future use and must always be zero.

DFP—Disable Floating-Point Unit

When this bit is set, the on-chip FPU is disabled and any attempt to execute a floating-point instruction generates a line F emulator exception. When this bit is cleared, the FPU executes all floating-point instructions. This bit is cleared at reset. Note that before this bit is set via the MOVEC instruction, an FNOP must be executed to ensure that all floating-point exceptions are caught and handled. This would prevent unexpected floating-point related exceptions to be reported when the FPU is re-enabled at a later time.

ESS—Enable Superscalar Dispatch

When this bit is set, the ability of the MC68060 to execute multiple instructions per machine cycle is enabled. When this bit is cleared, the ability to execute multiple instructions per cycle is disabled and the MC68060 operates at a slower rate with lower performance. This bit is cleared at reset.

have the V-bit and D-bit set, indicating that the line has valid entries that have not been written to memory. A cache line changes states from valid or dirty to invalid if the execution of the CINV or CPUSH instruction explicitly invalidates the cache line or if a snooped access hits the cache line. Both caches should be explicitly cleared using the CINVA instruction after a hardware reset of the processor since reset does not invalidate the cache lines.

Figure 5-4 illustrates the general flow of a caching operation. The caches use the physical addresses, and to simplify the discussion, the discussion of the translation of logical to physical addresses is omitted.

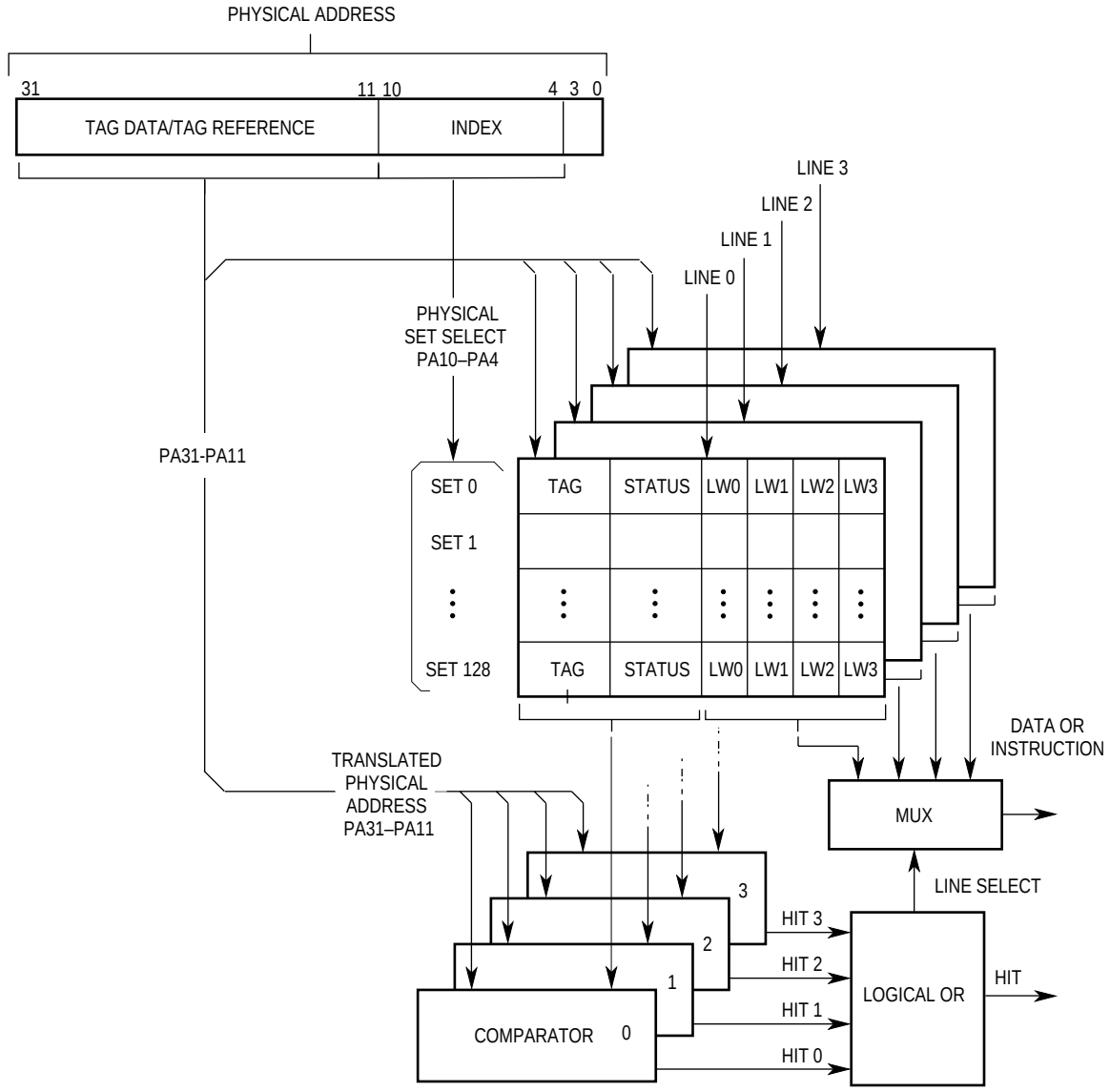


Figure 5-4. Caching Operation

To determine if the physical address is already allocated in the cache, the lower physical address bits 10–4 are used to index into the cache and select 1 of 128 sets of cache lines. Physical address bits 31–11 are used as a tag reference or to update the cache line tag

Bits 26–24—Reserved.

EBC—Enable Branch Cache

0 = The branch cache is disabled and branch cache information is not used in the branch prediction strategy.
1 = The on-chip branch cache is enabled. Branches are cached. A predicted branch executes more quickly, and often can be folded onto another instruction.

CABC—Clear All Entries in the Branch Cache

This bit is always read as zero.

0 = No operation is done on the branch cache.
1 = The entire content of the MC68060 branch cache is invalidated.

CUBC—Clear All User Entries in the Branch Cache

This bit is always read as zero.

0 = No operation is performed on the branch cache.
1 = All user-mode entries in the MC68060 branch cache are invailidated; supervisor-mode branch cache entries remain valid.

Bits 20–16—Reserved.

EIC—Enable Instruction Cache

0 = Instruction cache is disabled.
1 = Instruction cache is enabled.

NAI—No Allocate Mode (Instruction Cache)

0 = Accesses that miss in the instruction cache will allocate.
1 = The instruction cache will continue to supply instructions to the processor, but an access that misses will not allocate.

FIC—1/2 Cache Operation Mode Enable (Instruction Cache)

0 = The instruction cache operates in normal, full-cache mode.
1 = The instruction cache operates in 1/2-cache mode.

Bits 13–0—Reserved.

5.3 CACHE MANAGEMENT

The caches are individually enabled and configured by using the MOVEC instruction to access the CACR. A hardware reset clears the CACR, disabling both caches and removing all configuration information; however, reset does not affect the tags, state information, and data within the caches. The CINV instruction must clear the caches before enabling them. The MC68060 cannot cache page descriptors.

System hardware can assert the cache disable ($\overline{\text{CDIS}}$) signal to dynamically disable the both the instruction and data caches, regardless of the state of the enable bits in the CACR. The caches are disabled immediately after the current access completes. If $\overline{\text{CDIS}}$ is asserted during the access for the first half of a misaligned operand spanning two cache lines, the

Table 5-3. Data Cache Line State Transitions

Cache Operation	Current State					
	Invalid Cases		Valid Cases		Dirty Cases	
OPU Read Miss	(C,W)I1	Read line from memory and update cache; Supply data to OPU; Go to valid state.	(C,W)V1	Read new line from memory and update cache; supply data to OPU; Remain in current state.	CD1	Push dirty cache line to push buffer; Read new line from memory and update cache; Supply data to OPU; Write push buffer contents to memory; Go to valid state.
OPU Read Hit	(C,W)I2	Not possible.	(C,W)V2	Supply data to OPU; Remain in current state.	CD2	Supply data to OPU; Remain in current state.
OPU Write Miss (Copyback Mode)	CI3	Read line from memory and update cache; Write data to cache; Go to dirty state.	CV3	Read new line from memory and update cache; Write data to cache; Go to dirty state.	CD3	Push dirty cache line to push buffer; Read new line from memory and update cache; Write push buffer contents to memory; Remain in current state.
OPU Write Miss (Writethrough Mode)	WI3	Write data to memory; Remain in current state.	WV3	Write data to memory; Remain in current state.	WD 3	Write data to memory; Remain in current state.
OPU Write Hit (Copyback Mode)	CI4	Not possible.	CV$	Write data to cache; Go to dirty state.	CD4	Write data to cache; Remain in current state.
OPU Write Hit (Writethrough Mode)	WI4	Not possible.	WV4	Write data to memory and to cache; Remain in current state.	WD 4	Push dirty cache line to memory; Write data to memory and to cache; Go to valid state.
Cache Invalidate	(C,W)I5	No action; Remain in current state.	(C,W)V5	No action; Go to invalid state.	CD5	No action (dirty data lost); Go to invalid state.
Cache Push	(C,W)I6	No action; Remain in current state.	(C,W)V6	No action; Go to invalid state.	CD6	Push dirty cache line to memory; Go to invalid state or remain in current state, depending on the DPI bit the the CACR.
Alternate Master Snoop Hit	(C,W)I7	Not possible.	(C,W)V7	No action; Go to invalid state.	CD7	No action (dirty data lost); Go to invalid state.

The user BSUN exception handler must execute an FSAVE as its first floating-point instruction. FSAVE allows other floating-point instructions to execute without reporting the BSUN exception again, although none of the state frame values are useful in the execution of the user BSUN exception handler. The BSUN exception is unique in that the exception is taken before the conditional predicate is evaluated. If the user BSUN exception handler does not set the PC to the instruction following the one that caused BSUN exception when returning, the exception is executed again. Therefore, it is the responsibility of the user BSUN exception handler to prevent the conditional instruction from taking the BSUN exception again. There are four ways to prevent taking the exception again:

1. Incrementing the stored PC in the stack bypasses the conditional instruction. This technique applies to situations where a fall-through is desired. Note that accurate calculation of the PC increment requires detailed knowledge of the size of the conditional instruction being bypassed.
2. Clearing the NAN bit prevents the exception from being taken again. However, this alone cannot deterministically control the result's indication (true or false) that would be returned when the conditional instruction re-executes.
3. Disabling the BSUN bit also prevents the exception from being taken again. Like the second method, this method cannot control the result indication (true or false) that would be returned when the conditional instruction re-executes.
4. Examining the conditional predicate and setting the FPCC NAN bit accordingly prevents the exception from being taken again. This technique gives the most control since it is possible to predetermine the direction of program flow. Bit 7 of the F-line operation word indicates where the conditional predicate is located. If bit 7 is set, the conditional predicate is the lower six bits of the F-line operation word. Otherwise, the conditional predicate is the lower six bits of the instruction word, which immediately follows the F-line operation word. Using the conditional predicate and the table for IEEE nonaware test in **6.4.2 Conditional Testing**, the condition codes can be set to return a known result indication when the conditional instruction is re-executed.

Prior to exiting the user BSUN exception handler, the user exception handler discards the floating-point state frame before executing the RTE to return to normal program flow.

6.6.2 Signaling Not-a-Number (SNAN)

An SNAN is used as an escape mechanism for a user-defined, non-IEEE data type. The processor never creates an SNAN as a result of an operation; a NAN created by an operand error exception is always a nonsignaling NAN. When an operand is an SNAN involved in an arithmetic instruction, the SNAN bit is set in the FPSR EXC byte. Since the FMOVEM, FMOVE FPCR, and FSAVE instructions do not modify the status bits, they cannot generate exceptions. Therefore, these instructions are useful for manipulating SNANs.

6.6.2.1 TRAP DISABLED RESULTS (FPCR SNAN BIT CLEARED). If the destination data format is S, D, X, or P, then the most significant bit of the fraction is set to one and the resulting nonsignaling NAN is transferred to the destination. No bits other than the SNAN bit of the NAN are modified, although the input NAN is truncated if necessary. If the destination data format is B, W, or L, then the 8, 16, or 32 most significant bits of the SNAN significand, with the SNAN bit set, are written to the destination.

Table 6-15. Possible Divide-by-Zero Exceptions

Instruction	Operand Value
FDIV	Source operand = 0 and floating-point data register is not a NAN, ∞, or zero
FLOG10	Source operand = 0
FLOG2	Source operand = 0
FLOGN	Source operand = 0
FTAN	Source operand is an odd multiple of $\pm\pi \div 2$
FSGLDIV	Source operand = 0 and floating-point data register is not a NAN, ∞, or zero
FATANH	Source operand = ±1
FLOGNP1	Source operand = –1

6.6.6.1 TRAP DISABLED RESULTS (FPCR DZ BIT CLEARED). The destination floating-point data register is written with a result that is dependent on the instruction that caused the DZ exception.

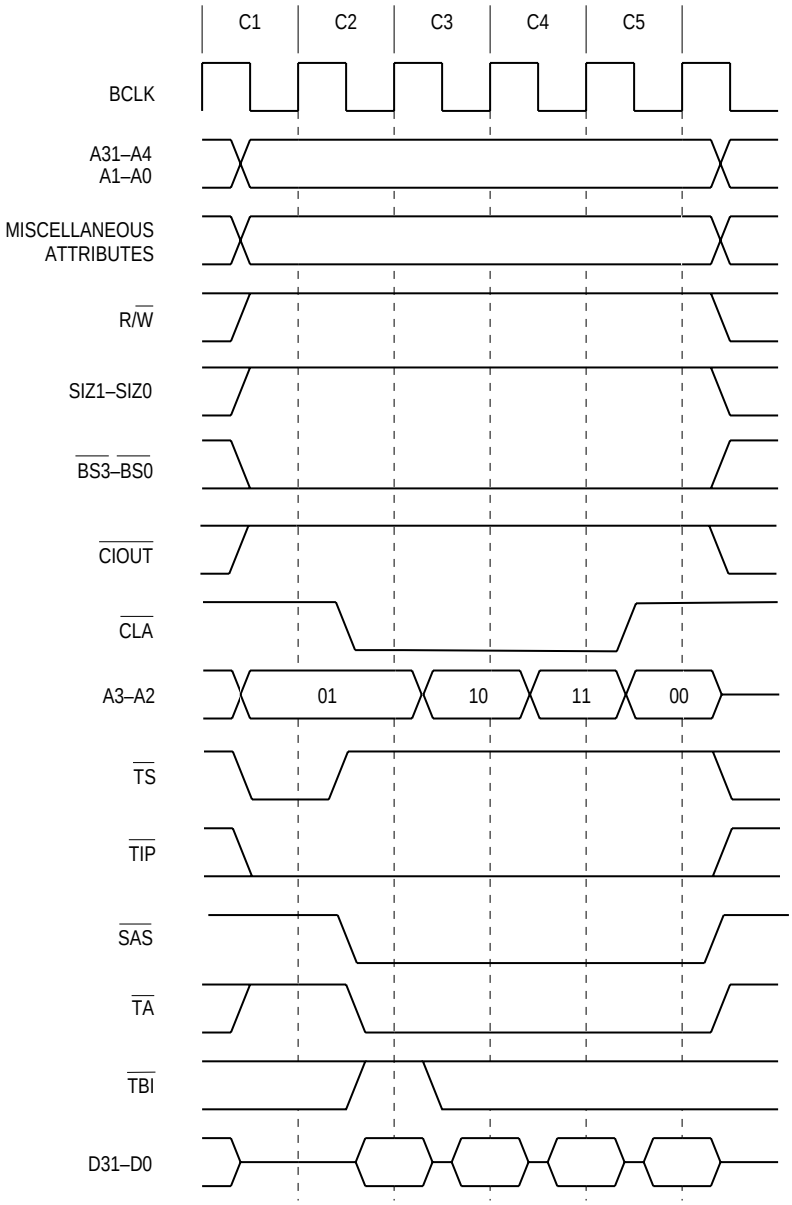
1. For the FDIV and FSGLDIV instructions, an infinity with the sign set to the exclusive OR of the signs of the input operands is stored in the destination.
2. For the FLOGx instructions, a $-\infty$ is stored in the destination.
3. For the FATANH instruction, a $+\infty$ is stored in the destination if the source operand is a –1, otherwise, a $-\infty$ is stored in the destination if the source operand is +1.

6.6.6.2 TRAP ENABLED RESULTS (FPCR DZ BIT SET). The destination floating-point data register is not modified. Control is passed to the user DZ handler as a pre-instruction exception when the next floating-point instruction is encountered. The user DZ handler must generate a result to store in the destination.

The user DZ handler must execute an FSAVE instruction as the first floating-point instruction to prevent further exceptions from reporting. The address of the instruction that causes the overflow is available to the user DZ handler in the FPIAR. By examining the instruction, the user DZ handler can determine the arithmetic operation type and destination location. The exception operand is stored in the floating-point state frame (generated by the FSAVE). The exception operand contains the source operand converted to the extended-precision format. When the user DZ exception handler has completed, the floating-point frame may be discarded. The RTE instruction must be executed to return to normal instruction flow.

6.6.7 Inexact Result

The processor provides two inexact bits in the FPSR EXC byte to help distinguish between inexact results generated by emulated decimal input (INEX1 exceptions) and other inexact results (INEX2 exceptions). These two bits are useful in instructions where both types of inexact results can occur (e.g., FDIV.P #7E-1,FP3). In this case, the packed decimal to extended-precision conversion of the immediate source operand causes an inexact error to occur that is signaled as INEX1 exception. Furthermore, the subsequent divide could also produce an inexact result and cause INEX2 to be set in the FPCR EXC byte. Note that only one inexact exception vector number is generated by the processor. If either of the two inexact exceptions is enabled, the processor fetches the inexact exception vector, and the user INEX exception handler is initiated. INEX refers to both exceptions in the following paragraphs.



NOTE: It is assumed that the acknowledge termination ignore state capability is disabled.

Figure 7-15. Line Read Transfer Timing

abled, $\overline{SAS}$ is asserted during C2 to indicate that the processor immediately begins sampling the terminations signals. Refer to **7.14.1 Acknowledge Termination Ignore State Capability** for details on this special mode.

Assuming that the acknowledge termination ignore state capability is disabled, the processor samples the level of $\overline{TA}$, $\overline{TBI}$, and $\overline{TCI}$ and registers the current value on the data bus at the end of C2. If $\overline{TA}$ is asserted, the transfer terminates and the data is passed to the appropriate memory unit. If $\overline{TA}$ is not recognized asserted, the processor ignores the data and inserts wait states instead of terminating the transfer. The processor continues to sample $\overline{TA}$, $\overline{TBI}$, and $\overline{TCI}$ on successive rising edges of BCLK until $\overline{TA}$ is recognized

is no longer needed. In an attempt to save time, the MC68060 negates $\overline{BR}$. If $\overline{BG}$ takes too long to assert, the MC68060 enters a disregard request condition.

The $\overline{BR}$ signal can be reasserted immediately for a different pending bus request, or it can stay negated indefinitely. If an external bus arbiter is designed to wait for the MC68060 to perform an active bus cycle before proceeding, then the system experiences an extended period of time in which bus arbitration is dead-locked. It must be understood that $\overline{BR}$ is a status signal which may or may not have any relationship to $\overline{BB}$, $\overline{BTT}$, or $\overline{BG}$.

When using the MC68060-arbitration protocol it is possible to determine bus tenure boundaries by observing $\overline{TS}$ and $\overline{BTT}$. An active bus tenure begins when a bus master asserts its $\overline{TS}$ for the first time. Once the bus tenure has started, the active bus master must end its tenure by asserting $\overline{BTT}$ (or a low-to-high transition of $\overline{BB}$). If a bus master is granted the bus, but does not start an active bus tenure by asserting $\overline{TS}$, no $\overline{BTT}$ assertion (or a low-to-high transition of $\overline{BB}$) is needed since no bus tenure was started. When reset is applied to the entire system, $\overline{TS}$ to all bus masters must be negated via a pullup resistor. In addition, the bus arbiter must grant the bus to a single bus master. Once the first bus master recognizes that $\overline{TS}$ is negated and that it has been granted the bus, it asserts its $\overline{TS}$ to establish its bus tenure and to inform other bus masters that its bus tenure has begun (this assumes that the $\overline{TS}$ signals of all bus masters in the system are tied together). All other bus masters will therefore detect an asserted $\overline{TS}$ ($\overline{TS}$ is asserted by the first bus master) immediately after reset. These bus masters must then wait for $\overline{BTT}$ to assert (or a low-to-high transition of $\overline{BB}$) before beginning their bus tenure when granted the bus.

Figure 7-43 illustrates an example of the processor requesting the bus from the external bus arbiter. During C1, the MC68060 asserts $\overline{BR}$ to request the bus from the arbiter, which negates the alternate bus master's $\overline{BG}$ signal and grants the bus to the processor by asserting $\overline{BG}$ during C2. During C2, the alternate bus master completes its current access and relinquishes the bus in C3 by three-stating all bus signals and negating $\overline{BB}$ and/or asserting $\overline{BTT}$. Typically, the $\overline{BB}$ and $\overline{BTT}$ signals require a pullup resistor to maintain a logic-one level between bus master tenures. The alternate bus master should negate these signals before three-stating to minimize rise time of the signals and ensure that the processor recognizes the correct level on the next BCLK rising edge. At the end of C3, the processor has already received bus grant and the alternate master has relinquished the bus. Hence, the processor assumes ownership of the bus and immediately begins a bus cycle during C4. During C6, the processor begins the second bus cycle for the misaligned operand and negates $\overline{BR}$ since no other accesses are pending. During C7, the external bus arbiter grants the bus back to the alternate bus master that is waiting for the processor to relinquish the bus. The processor negates $\overline{BB}$, asserts $\overline{BTT}$, and three-states bus signals during C8. Finally, the alternate bus master has the bus grant. The processor has relinquished the bus at the end of C8 and is able to resume bus activity during C9. Note that $\overline{BTT}$ is asserted only for one BCLK period and is negated for one BCLK period during C10. $\overline{BTT}$ is then three-stated in C10.

Further note that $\overline{BB}$ is only negated for one CLK (as opposed to BCLK) period before being three-stated, and the MC68040-arbitration protocol should not be used for full bus speed operation.

interrupting device using an interrupt acknowledge bus cycle with the interrupt level number output on the transfer modifier signals. For a device that cannot supply an interrupt vector, the autovector signal ($\overline{AVEC}$) must be asserted. In this case, the MC68060 uses an internally generated autovector, which is one of vector numbers 25–31, that corresponds to the interrupt level number (see Table 8-1). If external logic indicates a bus error during the interrupt acknowledge cycle, the interrupt is considered spurious, and the processor generates the spurious interrupt vector number, 24.

Once the vector number is obtained, the processor creates a stack frame of type 0. In this stack frame, the processor saves the exception vector offset, PC value, and the internal copy of the SR on the supervisor stack. The saved value of the PC is the logical address of the next instruction had the interrupt not occurred.

Unlike previous processors of the M68000 family, the MC68060 defers interrupt sampling from the beginning of exception processing of any exception, up to and until the first instruction of the exception handler. This allows the first instruction of any exception handler to raise the interrupt mask level and therefore execute the exception handler without interrupts (except level 7 interrupts).

Most M68000 family peripherals use programmable interrupt vector numbers as part of the interrupt acknowledge operation for the system. If this vector number is not initialized after reset and the peripheral must acknowledge an interrupt request, the peripheral usually returns the vector number for the uninitialized interrupt vector, 15.

8.2.10 Reset Exception

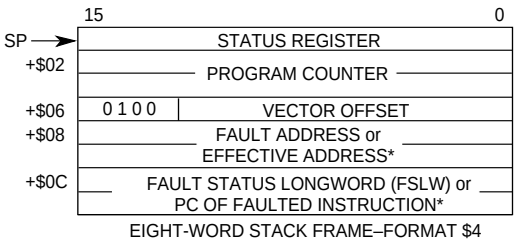
Asserting the reset in ($\overline{RSTI}$) input signal causes a reset exception. The reset exception has the highest priority of any exception; it provides for system initialization and recovery from catastrophic failure. Reset also aborts any processing in progress when $\overline{RSTI}$ is recognized; processing cannot be recovered. Figure 8-5 is a flowchart of the reset exception processing.

The reset exception places the processor the supervisor mode by setting the S-bit and disables tracing by clearing the T-bit in the SR. This exception also sets the processor's interrupt priority mask in the SR to the highest level, level 7. Next the VBR is initialized to zero ($00000000), and all bits in the cache control register (CACR) for the on-chip caches are cleared. The reset exception also clears the translation control register (TCR). It clears the enable bit in each of the four transparent translation registers (TTRs). It also clears the bus control register (BUSCR), and the PCR. The reset also affects the FPU. A quiet not-a-number (NAN) is loaded into each of the seven floating-point registers, and the floating-point control register (FPCR), floating-point status register (FPSR), and floating-point instruction address register (FPIAR) are cleared. If the processor is granted the bus, and the processor does not detect $\overline{TS}$ asserted (possibly by an alternate master), the processor then performs two long-word read bus cycles. The first long word, at address 0, is loaded into the SP, and the second long word, at address 4, is loaded into the PC. Reset exception processing concludes with the transfer of control to the memory location defined by the PC.

After the initial instruction is fetched, program execution begins at the address in the PC. The reset exception does not flush the ATCs or invalidate entries in the instruction or data caches; it does not save the value of either the PC or the SR. If an access error or address

8.4.4 Eight-Word Stack Frame (Format $4)

An eight-word stack frame is created for data and instruction access errors. It is also used for the floating-point disabled exception. Refer to **8.2.4 Illegal Instruction and Unimplemented Instruction Exceptions** for details on the use of this frame for the floating-point disabled exception. The following paragraphs describe in detail the format for this frame as used by for the access error and how the processor uses it when returning from exception processing.

Stack Frames	Exception Types	Stacked PC Points To
 EIGHT-WORD STACK FRAME–FORMAT $4 * Defined for the Floating-Point Disabled Exception	• Data or Instruction Access Fault (ATC Fault or Bus Error) • Floating-Point Disabled Exception	• See **8.4.4.1 Program Counter (PC)**, **8.4.4.2 Fault Address**, and **8.4.4.3 Fault Status Long Word (FSLW)** for additional information. • Next instruction; Effective Address Field has calculated <ea> of memory operand (if any); PC of Faulted Instruction points to the F-line instruction word of the floating-point instruction.

8.4.4.1 Program Counter (PC). On read access faults, the PC points to the instruction that caused the access error. This instruction is restarted when an RTE is executed, hence, the read cycle is re-executed. On read access errors on the second or later of misaligned reads, the read cycles that are successful prior to the access error are re-executed since the processor uses a restart model for recovery from exceptions.

Programs that rely on a read bus error to test for the existence of I/O or peripheral devices must increment the value of the PC prior to the execution of the RTE instruction. Incrementing the PC involves the calculation of the instruction length, which is dependent on the addressing mode used. To avoid having to calculate the instruction length, it is possible to use a NOP-TEST_WRITE-NOP instead of a TEST_READ of the I/O or peripheral device. The initial NOP causes all prior write cycles to complete. The TEST_WRITE causes the access error, and if the write cycle is to imprecise operand space, the stacked PC of the access error stack contains the address of the second NOP. When the RTE is executed, instruction execution resumes at the second NOP. The limitation of this method is that it works only if the I/O device is mapped to imprecise operand space. If the write is to a precise operand space, the processor does not increment the PC, and the stacked PC contains the instruction address of the TEST_WRITE.

On write access errors, the PC points to the instruction that causes the access error except for bus error ($\overline{\text{TEA}}$) on writes that involve the push and store buffers. Refer to **8.4.4.3 Fault Status Long Word (FSLW)** for specific information on these write cases. For these write cases, the PC does not point to the instruction that caused the access error. Hence the write cycle that incurred the bus error is lost. In general, bus errors on writes must be avoided. The processor provides little support for recovery on bus errored write cycles to imprecise operand spaces. For precise spaces, both the faulting PC and logical operand address are directly provided in the exception frame.

Table 10-22. Status Register (SR) Instruction Execution Times

Instruction	Execution Time
ANDI to SR	12(0/0)
EORI to SR	12(0/0)
MOVE from SR	1(0/1)[1]
MOVE to SR	12(1/0)[1]
ORI to SR	5(0/0)

[1] For these instructions, add the effective address calculation time.

Table 10-23. MOVES Execution Times

MOVES Function	Destination							
	Size	(An)	(An)+	–(An)	(d16,An)	(d8,An,Xi*SF)	(bd,An,Xi*SF)[1]	(xxx).WL
Source<SFC> -> Rn	Byte, Word	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	3(1/0)	2(1/0)
“	Long	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	3(1/0)	2(1/0)
Rn -> Dest <DFC>	Byte, Word	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	3(0/1)	2(0/1)
“	Long	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	3(0/1)	2(0/1)

[1] Add 2(1/0) cycles to the (bd,An,Xi*SF) time for a memory indirect address.

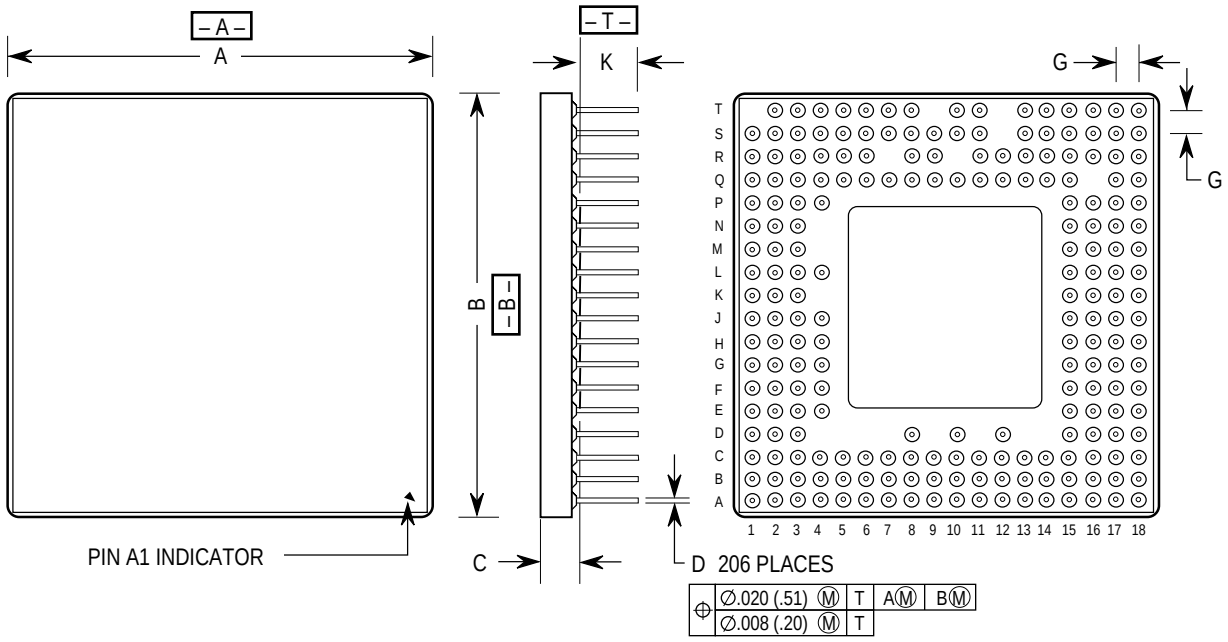
Table 10-24. Miscellaneous Instruction Execution Times

Instruction	Size	Register	Memory	Reg -> Dest	Source -> Reg
ANDI to CCR	Byte	1(0/0)	—	—	—
CHK	Word	2(0/0)	2(1/0) [1]	—	—
“	Long	2(0/0)	2(1/0) [1]	—	—
CINVA	—	—	<=17(0/0)	—	—
CINVL	—	—	<=18(0/0)	—	—
CINVP	—	—	<=274(0/0)	—	—
CPUSHA	—	—	<=5394(0/512)[2]	—	—
CPUSHL	—	—	<=26(0/1)[2]	—	—
CPUSHP	—	—	<=2838(0/256)[2]	—	—
EORI to CCR	Byte	1(0/0)	—	—	—
EXG	Long	1(0/0)	—	—	—
EXT	Word	1(0/0)	—	—	—
“	Long	1(0/0)	—	—	—
EXTB	Long	1(0/0)	—	—	—
LINK	Word	2(0/1)	—	—	—
“	Long	2(0/1)	—	—	—
LPSTOP	Word	15(0/1)	—	—	—
MOVE from CCR	Word	1(0/0)	1(0/1)[1]	—	—
MOVE to CCR	Word	1(0/0)	1(1/0)[1]	—	—
MOVE from USP	Long	1(0/0)	—	—	—
MOVE to USP	Long	2(0/0)	—	—	—
MOVEC (SFC,DFC, USP,VBR,PCR)	Long	—	—	12(0/0)	11(0/0)
MOVEC (CACR,TC, TTR,BUSCR,URP,SRP)	Long	—	—	15(0/0)	14(0/0)
NOP	—	9(0/0)	—	—	—
ORI to CCR	Byte	1(0/0)	—	—	—
PACK	—	2(0/0)	2(1/1)	—	—

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)

- Floating-point data register destination:
  - —exception stack frame: the four-word pre-instruction stack frame contains the PC of the next instruction.
  - —in the FSAVE frame: the exceptional operand which is the intermediate result mantissa rounded to extended precision, with an exponent bias of $3FFF+$6000 for underflow and $3FFF-$6000 for overflow rather than $3FFF. In cases of catastrophic overflow/underflow, the exceptional operand exponent is set to $0000. The user ovfl/unfl handler must execute an FSAVE to retrieve this value.
  - —at the destination location: the default underflow/overflow result.
  - —FPIAR: address of the instruction that underflowed/overflowed.
  - —FPSR: the bits are set according to the default result.

Note

Unlike the MC68040, the MC68060 FPU hardware does not provide the exceptional operand on overflow or underflow for use by an exception handler. Therefore, the M68060FPSP overflow and underflow handlers must emulate the entire faulted instruction in order to calculate the exceptional operand for the user enabled overflow or underflow handler.

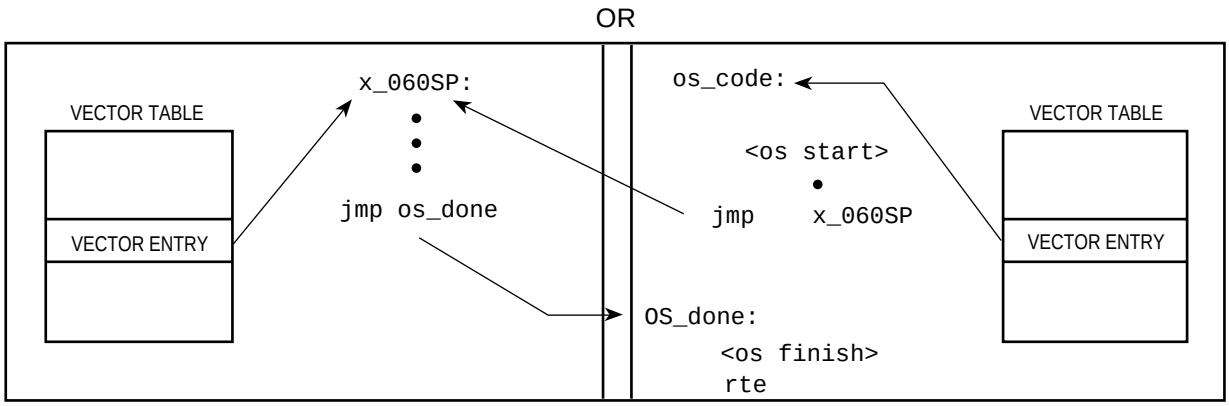
Finally, if the result of the floating-point multiplication unit is a normalized extended-precision number with a zero exponent, then the processor will incorrectly take an underflow exception. The M68060SP detects and corrects this case.

C.3.2.3.2 Signalling Not-A-Number, Operand Error. On the MC68060, the signalling not-a-number (SNAN) and operand error (OPERR) exceptions cause pre-instruction exceptions for opclass zero and two instructions and post-instruction exceptions for opclass three instructions. The processor takes exception vector number fifty-four for the SNAN exception and vector number fifty-two for the OPERR exception. The FSAVE frames for the exceptions are valid and contain the source operands converted to extended precision.

SNAN and OPERR were non-maskable exceptions on the MC68040 for opclass three instructions with byte, word, or long-word destination formats. The exceptions were non-maskable so that the MC68040FPSP software could provide the default SNAN or OPERR results when the exceptions were disabled. With the MC68060, as with the MC68881/882, SNAN and OPERR are entirely maskable since the default trap disabled results are provided by floating-point hardware.

ifying the call-out dispatch table, keep in mind that these need to be supplied and filled-in with **module-relative**, and not absolute addresses.

The next step is to prepare the exception vector table. The appropriate vector table entries must be filled with the addresses of the appropriate entry points. Since the modified pseudo-assembly module contains symbols that indicate the top of the module, the appropriate vector table entries must contain the symbol of the appropriate module top plus the pre-defined offset. Another alternative is to use the module code size information given in Table C-1 to concatenate the modules and use a single symbolic label to describe the combined module. Figure C-12 illustrates the relationship of the vector table to the M68060SP.



NOTE: X_060SP represents a generic M68060SP handler entry point and is not intended to imply a single shared handler entry point for all MC68060 exception handlers.

Figure C-12. Vector Table and M68060SP Relationship

The last step is to link everything. Be aware that the files must be linked such that the parts of the module that are in different files are kept together. Be aware that the included files are for a very simple installation procedure and may not be appropriate for all systems. For instance, the supplied _real_trace routine would be inappropriate for a system in which the trace vector table entry is dynamically changed. For that system, the _real_trace routine must include vector table query before jumping to the actual trace routine.

C.5.3 Release Notes and Module Offset Assignments

To obtain the most up-to-date offset assignments for the call-out dispatch table and the module Entry-point Dispatch Section assignments, four document files are provided with the M68060SP release. The files isp.doc, ilsp.doc, fpsp.doc, and fplsp.doc define the offsets for the unimplemented integer instruction exception handler, unimplemented integer subroutine, full (or partial) floating-point kernel and floating-point library modules respectively. The current module sizes are shown in Table C-1. If a module increases in code size or if additional entry points are made available in future releases, they will be documented in these four files.

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

The current implementation of the MC68060 supports the following four state frames:

NULL: This state frame has a frame format of $00. An FRESTORE operation with this state frame is equivalent to a hardware reset of the floating-point unit. The programmer's model is set to the reset state, with nonsignaling NANs in the floating-point data registers and zeros in the floating-point control register, floating-point status register, and floating-point instruction address register. (Thus, it is unnecessary to load the programmer's model before this operation.)

IDLE: This state frame has a frame format of $60. An FRESTORE operation with this state frame causes the floating-point unit to be restored to the idle state, waiting for the initiation of the next instruction, with no exceptions pending. The programmer's model is not affected by loading this type of state frame.

EXCP: This state frame has a frame format of $E0. An FRESTORE operation with this state frame causes the floating-point unit to be restored to an exceptional state. The exception vector field defines the type of exception that is pending. When in this state, initiation of any floating-point instruction with the exception of FSAVE or another FRESTORE causes the pending exception to be taken.The floating-point unit remains in this state until an FSAVE instruction is executed, then, it enters the idle state. The programmer's model is not affected by loading this type of state frame.

Floating-Point Status Register: Cleared if the state size is NULL; otherwise, not affected.

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