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	75MHz
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/pro/item?MUrl=&PartUrl=mc68ec060zu75

Section 3
Integer Unit

Section 4
Memory Management Unit

SECTION 2
SIGNAL DESCRIPTION

This section contains brief descriptions of the MC68060 signals in their functional groups (see Figure 2-1). Each signal's function is briefly explained, referencing other sections containing detailed information about the signal and related operations. Table 2-1 lists the MC68060 signal names, mnemonics, and functional descriptions of the signals. Timing specifications for these signals can be found in **Section 12 Electrical and Thermal Characteristics**.

NOTE

Assertion and *negation* are used to specify forcing a signal to a particular state. *Assertion* and *assert* refer to a signal that is active or true. *Negation* and *negate* refer to a signal that is inactive or false. These terms are used independently of the voltage level (high or low) that they represent.

Table 2-1. Signal Index

Signal Name	Mnemonic	Function
Address Bus	A31–A0	32-bit address bus used to address any of 4-Gbytes.
Cycle Long-Word Address	$\overline{\text{CLA}}$	Controls the operation of A3 and A2 during bus cycles.
Data Bus	D31–D0	32-bit data bus used to transfer up to 32 bits of data per bus transfer.
Transfer Type	TT1,TT0	Indicates the general transfer type: normal, MOVE16, alternate logical function code, and acknowledge.
Transfer Modifier	TM2–TM0	Indicates supplemental information about the access.
Transfer Line Number	TLN1,TLN0	Indicates which cache line in a set is being pushed or loaded by the current line transfer cycle.
User-Programmable Attributes	UPA1,UPA0	User-defined signals, controlled by the corresponding user attribute bits from the address translation entry.
Read/Write	R/$\overline{\text{W}}$	Identifies the transfer as a read or write.
Transfer Size	SIZ1,SIZ0	Indicates the data transfer size. These signals, together with A0 and A1, define the active sections of the data bus. Alternately, BS3–BS0 can be used for this function.
Bus Lock	$\overline{\text{LOCK}}$	Indicates a bus cycle is part of a read-modify-write operation and that the sequence of bus cycles should not be interrupted.
Bus Lock End	$\overline{\text{LOCKE}}$	Indicates the current bus cycle is the last in a locked sequence of bus cycles.
Cache Inhibit Out	$\overline{\text{CIOUT}}$	Indicates the processor will not cache the current bus transfer information.
Byte Select	$\overline{\text{BS3}}$–$\overline{\text{BS0}}$	Indicate which bytes within a long word are selected and which data bus bytes are valid.
Transfer Start	$\overline{\text{TS}}$	Indicates the beginning of a bus cycle.
Transfer in Progress	$\overline{\text{TIP}}$	Asserted for the duration of a bus cycle.
Starting Termination Acknowledge Signal Sampling	$\overline{\text{SAS}}$	Indicates the MC68060 will begin sampling the termination acknowledge signals.
Transfer Acknowledge	$\overline{\text{TA}}$	Asserted to acknowledge a bus transfer.

The operation of the instruction fetch unit (IFU) and the OEPs are decoupled by a 96-byte FIFO instruction buffer. The IFU prefetches instructions every processor clock cycle, stopping only if the instruction buffer is full or encountering a wait condition due to instruction fetch address translation or cache miss. The OEPs attempt to read instructions from the instruction buffer and execute them every clock cycle, stopping only if full instruction information is not present in the buffer or due to operand pipeline wait conditions.

3.2 INTEGER UNIT REGISTER DESCRIPTION

The following paragraphs describe the integer unit registers in the user and supervisor programming models. Refer to **Section 4 Memory Management Unit** for details on the MMU programming model and **Section 6 Floating-Point Unit** for details on the FPU programming model.

3.2.1 Integer Unit User Programming Model

Figure 3-2 illustrates the integer unit portion of the user programming model. The model is the same as for previous M68000 family microprocessors, consisting of the following registers:

- 16 General-Purpose 32-Bit Registers (D7–D0, A7–A0)
- 32-Bit Program Counter (PC)
- 8-Bit Condition Code Register (CCR)

3.2.1.1 DATA REGISTERS (D7–D0). Registers D7–D0 are used as data registers for bit and bit field (1- to 32-bit), byte (8-bit), word (16-bit), long-word (32-bit), and quad-word (64-bit) operations. These registers may also be used as index registers.

3.2.1.2 ADDRESS REGISTERS (A6–A0). These registers can be used as software stack pointers, index registers, or base address registers. The address registers may be used for word and long-word operations.

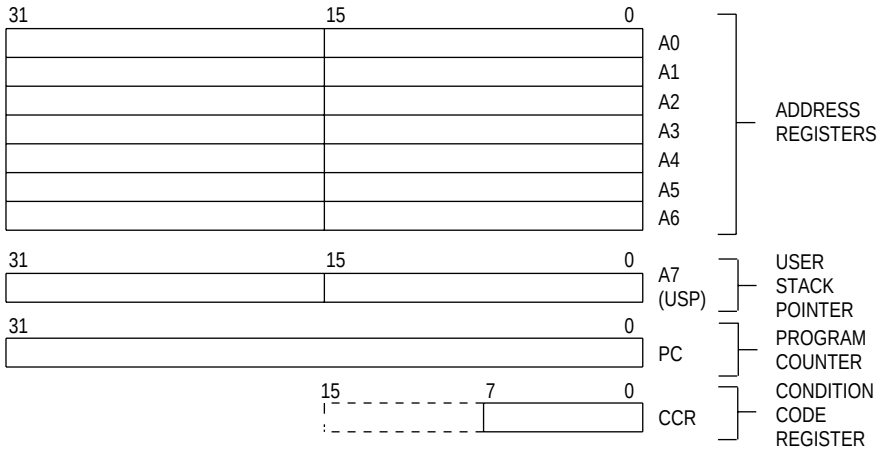


Figure 3-2. Integer Unit User Programming Model

3.2.1.3 USER STACK POINTER (A7). A7 is used as a hardware stack pointer during implicit or explicit stacking for subroutine calls and exception handling. The register designation A7 refers to the user stack pointer (USP) in the user programming model and to the

SECTION 4
MEMORY MANAGEMENT UNIT

NOTE

This section does not apply to the MC68EC060. Refer to **Appendix B MC68EC060** for details.

The MC68060 supports a demand-paged virtual memory environment. Demand means that programs request permission to use memory area by accessing logical addresses, and paged means that memory is divided into blocks of equal size, called page frames. Each page frame is divided into pages of the same size. The operating system assigns pages to page frames as they are required to meet the needs of the program.

The MC68060 memory management includes the following features:

- Independent Instruction and Data Memory Management Units (MMUs)
- 32-Bit Logical Address Translation to 32-Bit Physical Address
- User-Defined 2-Bit Physical Address Extension
- Addresses Translated in Parallel with Indexing into Data or Instruction Cache
- 64-Entry Four-Way Set-Associative Address Translation Cache (ATC) for Each MMU (128 Total Entries)
- Global Bit Allowing Flushes of All Nonglobal Entries from ATCs
- Selectable 4- or 8-Kbyte Page Size
- Separate Supervisor and User Translation Tables
- Two Independent Blocks for Each MMU Can Be Defined as Transparent (Untranslated)
- Three-Level Translation Tables with Optional Indirection
- Supervisor and Write Protections
- History Bits Automatically Maintained in Descriptors
- External Translation Disable Input Signal ($\overline{\text{MDIS}}$) for Emulator Support
- Caching Mode Selected on Page Basis
- Default Transparent Translation
- Default Cache Mode and User Attributes

The MMUs completely overlap address translation time with other processing activities when the translation is resident in the corresponding ATC. ATC accesses operate in parallel with indexing into the on-chip instruction and data caches. The MMU $\overline{\text{MDIS}}$ signal dynamically disables address translation for emulation and diagnostic support.

For 8-Kbyte pages, the five bits of the PGI field are multiplied by 4 (shifted to the left by two bits) and concatenated with the fetched pointer-level descriptor's upper 25 bits to produce the physical address of the 8-Kbyte page descriptor. The upper 19 bits of the page descriptor are the page frame's physical address. There are 32 8-Kbyte page descriptors in a page-level table.

Similarly, for 4-Kbyte pages, the six bits of the PGI field are multiplied by 4 (shifted to the left by two bits) and concatenated with the fetched pointer-level descriptor's upper 24 bits to produce the physical address of the 4-Kbyte page descriptor. The upper 20 bits of the page descriptor are the page frame's physical address. There are 64 4-Kbyte page descriptors in a page-level table.

Write-protect status is accumulated from each level's descriptor and combined with the status from the page descriptor to form the ATC entry status. The MC68060 creates the ATC entry from the page frame address and the associated status bits and uses this address and attributes to generate a bus access. Refer to **4.3 Address Translation Caches** for details on ATC entries.

If the descriptor from a page table is an indirect descriptor, the page descriptor pointed to by this descriptor is fetched. Invalid descriptors can be used at any level of the tree except the root. When a table search for a normal translation encounters an invalid descriptor, the processor takes an access error exception. The invalid descriptor can be used to identify either a page or branch of the tree that has been stored on an external device and is not resident in memory or a portion of the translation table that has not yet been defined. In these two cases, the exception routine can either restore the page from disk or add to the translation table. Figure 4-8 and Figure 4-9 illustrate detailed flowcharts of table search and descriptor fetch operations.

A table search terminates successfully when a page descriptor is encountered. The occurrence of an invalid descriptor or a transfer error acknowledge also terminates a table search, and the MC68060 takes an access error exception immediately on the data access and is delayed for instruction fetches until the instruction is ready to be executed. The exception handler should distinguish between anticipated conditions and true error conditions. The exception handler can correct an invalid descriptor that indicates a nonresident page or one that identifies a portion of the translation table yet to be allocated. An access error due to a system malfunction can require the exception handler to write an error message and terminate the task. The fault status long word (FSLW) of the access error stack frame provides detailed information regarding the cause of the exception. Refer to **Section 8 Exception Processing** for more information on exception handling.

The processor does not use the data cache when performing a table search. Therefore, translation tables must not be placed in copyback space, since the normal accesses which build the translation tables would be cached and not written to external memory, but the processor only uses tables in external memory. This is a functional difference between the MC68060 and the MC68040.

Table and page descriptors must not be left in a state that is incoherent to the processor. Violation of this restriction can result in an undefined operation. Page descriptors must not

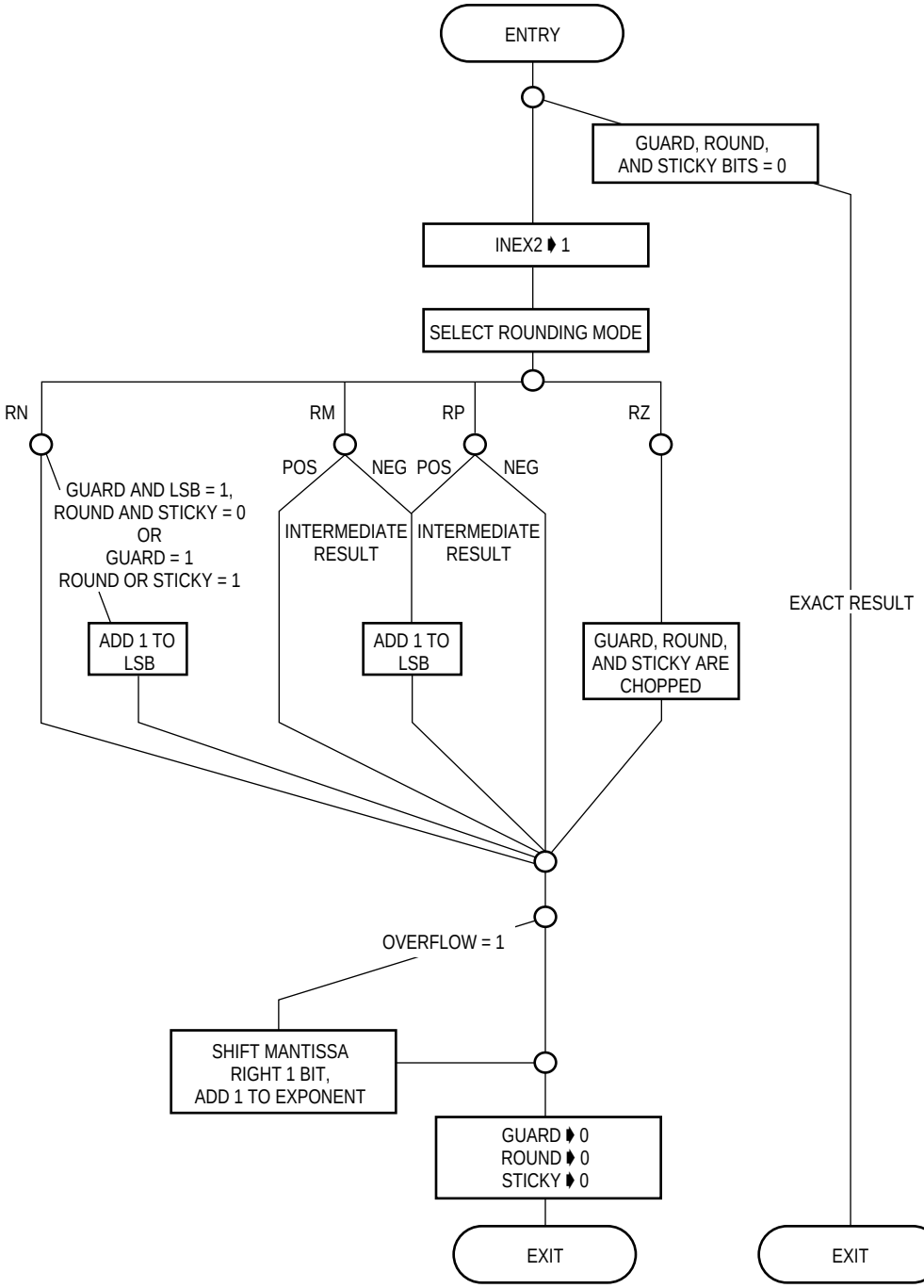


Figure 6-9. Rounding Algorithm Flowchart

Result	Integer	63-Bit Fraction	Guard	Round	Sticky
Intermediate	x	xxx…x00	1	0	0
Rounded-to-Nearest	x	xxx…x00	0	0	0

handled in this manner. If the destination data format is extended and there is a difference between the infinitely precise intermediate result and the round-to-nearest result, the relative difference is 2^{-64} (the value of the guard bit). This error is equal to one-half of the least significant bit's value and is the worst case error that can be introduced when using the RN

Table 6-12. Possible Operand Errors Exceptions

Instruction	Condition Causing Operand Error
Native to MC68060	
FADD	[(+∞) + (–∞)] or [(–∞) + (+∞)]
FDIV	(0 ÷ 0) or (∞ ÷ ∞)
FMOVE to B,W,or L	Integer overflow, source is nonsignaling NAN or ±∞
FMUL	One operand is 0 and other is +∞
FSQRT	(Source < 0) or (–∞)
FSUB	[(+∞) – (+∞)] or [(–∞) – (–∞)]
Non-Native to MC68060	
FACOS	Source is ±∞, > +1, or < –1
FASIN	Source is ±∞, > +1, or < –1
FATANH	Source is ±∞, > +1, or < –1
FCOS	Source is ±∞
FGETEXP	Source is ±∞
FGETMAN	Source is ±∞
FLOG10	Source is < 0 or –∞
FLOG2	Source is < 0 or –∞
FLOGN	Source is < 0 or –∞
FLOGNP1	Source is ≤ 1 or –∞
FMOD	Floating-point data register is ±∞ or source is 0, other operand is not a NAN
FMOVE to P	Source exponent > 999 (decimal) or k-factor > 17
FREM	Floating-point data register is ±∞ or source is 0, other operand is not a NAN
FSCALE	Source is ±∞, other operand not a NAN
FSGLDIV	(0 ÷ 0) or(∞ ÷ ∞)
FSGLMUL	One operand is 0, other operand is ∞
FSIN	Source is ±∞
FSINCOS	Source is ±∞
FTAN	Source is ±∞

6.6.3.1 TRAP DISABLED RESULTS (FPCR OPERR BIT CLEARED). For an FMOVE OUT instruction with the format S, D, or X, an OPERR is impossible. For an FMOVE OUT instruction with the format B, W, or L, an OPERR is possible only on an integer overflow, if the source is an infinity, or if the source is a NAN. On the integer overflow and infinity source cases, the largest positive or negative integer that can fit in the specified destination size (B, W, or L) is stored. On the NAN source case, the 8, 16, or 32 most significant bits of the NAN significand is stored in the B, W, or L destination.

For FMOVE OUT with the format P (packed decimal), if the k-factor is greater than +17, the result returned is a packed decimal string that assumes a k-factor equal to +17. For packed decimal results where the absolute value of the exponent is greater than 999, the decimal string is returned with the three least significant exponent digits in EXP2, EXP1, and EXP0. The fourth digit, EXP3, is supplied in the most significant four bits of the third byte in the string.

For all other OPERR cases, the destination is a floating-point data register. An extended-precision non-signaling NAN is stored in the destination.

6.6.3.2 TRAP ENABLED RESULTS (FPCR OPERR BIT SET). For the FMOVE OUT cases, the destination is written as if the trap were disabled, and then control is passed to

When the user INEX exception handler has completed, the floating-point frame may be discarded. The RTE instruction must be executed to return to normal instruction flow.

NOTE

The IEEE 754 standard specifies that inexactness should be signaled on overflow as well as for rounding. The processor implements this via the INEX bit in the FPSR AEXC byte. However, the standard also indicates that the inexact exception should be taken if an overflow occurs with the OVFL bit disabled and the INEX bit enabled in the FPSR AEXC byte. Therefore, the processor takes the inexact exception if this combination of conditions occurs, even though the INEX1 or INEX2 bit may not be set in the FPSR EXC byte. In this case, the INEX bit is set in the FPSR AEXC byte, and the OVFL bit is set in both the FPSR EXC and AEXC bytes.

6.7 FLOATING-POINT STATE FRAMES

All floating-point arithmetic exception handlers must have FSAVE as the first floating-point instruction; any other floating-point instruction causes another exception to be reported. Once the FSAVE instruction has executed, the exception handler should use only the FMOVEM instruction to read or write to the floating-point data registers since FMOVEM cannot generate further exceptions or change the FPCR.

An FSAVE instruction is executed to save the current floating-point internal state for context switches and floating-point exception handling. When an FSAVE is executed, the processor waits until the FPU either completes the instruction or is unable to perform further processing due to a pending exception that must be serviced.

FSAVE operations always write a floating-point state frame containing three long words. The exception operand, is part of the EXCP frame. This exception operand retains its value when FRESTOREd as an EXCP frame into the processor and then FSAVEd at a later time. The FSAVE frame contents are shown in Figure 6-10 and the status word contents are shown in Figure 6-11.

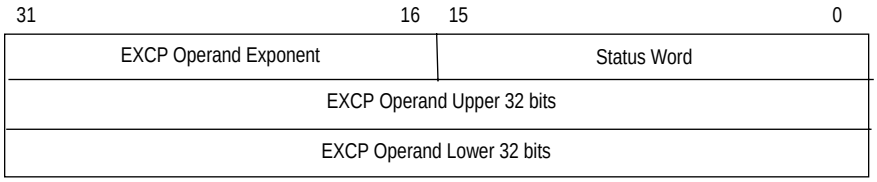
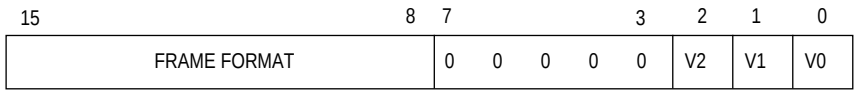


Figure 6-10. Floating-Point State Frame

Bits 15–8 of the first long word of the floating-point frame define the frame format. The legal formats for the MC68060 are:

$00	Null Frame (NULL)
$60	Idle Frame (IDLE)
$E0	Exception Frame (EXCP)



Frame Format
- $00—Null Frame
- $60—Idle Frame
- $E0—Exception Frame

V2–V0—Exception Vector
- 000—BSUN
- 001—INEX2 | INEX1
- 010—DZ
- 011—UNFL
- 100—OPERR
- 101—OVFL
- 110—SNAN
- 111—UNSUP

Figure 6-11. Status Word Contents

FSAVE on the MC68060 only generates one size frame (three long words), which creates a significant performance benefit, and one of these three frame types. An attempt to FRESTORE a frame format other than $00, $60, or $E0 results in a format error exception.

The format of the first long word of the MC68060 floating-point frame has changed from that of previous M68000 microprocessors. The MC68060 frame format (bits 15–8) is a consolidation of the version number and size format information (bits 31–16) on previous parts. In addition, on the MC68060, this information resides in the lower word of the long word while the upper word is used for the exception operand exponent in EXCP frames. Therefore, FRESTORE of a frame on an MC68060 created by FSAVE on a non-MC68060 microprocessor and FRESTORE of a frame on a non-MC68060 microprocessor created by FSAVE on an MC68060 will not guarantee a format error exception will be detected and thus must never be attempted.

When an FSAVE is executed, the floating-point frame reflects the state of the FPU at the time of the FSAVE. Internally, the FPU can be in the NULL, IDLE or EXCP states. Upon reset, the FPU is in the NULL state. In the NULL state, all floating-point registers contain nonsignaling NANs and the FPCR, FPSR, and FPIAR contain zeroes. The FPU remains in this state until the execution of an implemented floating-point instruction (except FSAVE). At this point, the FPU transitions from a NULL state to an IDLE state. An FRESTORE of NULL returns the FPU to the NULL state. The EXCP state is entered as a result of either a floating-point exception or an unsupported data type exception. V2–V0 indicates the exception types that are associated with the EXCP state.

An FSAVE instruction always clears the internal exception status bit at the completion of the FSAVE. An FRESTORE of EXCP may be used to place the FPU in the exception state.

The FRESTORE of an EXCP state is used in the M68060SP to provide to the user exception handler the illusion that the M68060SP handler never existed at all. The user exception handler is entered with the FPU in the proper exception state. The user

instruction boundary (following any higher priority exception). The IPEND signal negates after the interrupt acknowledge bus cycle.

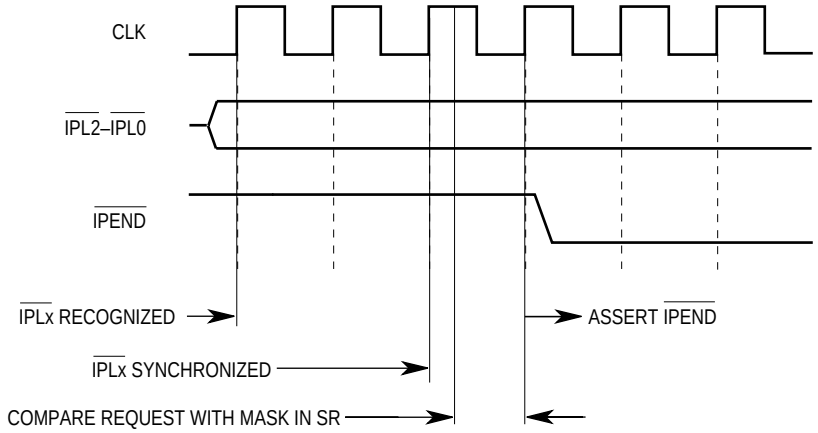


Figure 7-26. Assertion of IPEND

IPEND is intended to provide status information, and must not be used to replace the interrupt acknowledge cycle. As such, normal applications do not rely on IPEND to disable interrupts. Applications that use IPEND as a replacement for the interrupt acknowledge cycle are neither recommended nor supported.

The MC68060 takes an interrupt exception for a pending interrupt within one instruction boundary after processing any other pending exception with a higher priority. Thus, the MC68060 executes at least one instruction in an interrupt exception handler before recognizing another interrupt request. The following paragraphs describe the various kinds of interrupt acknowledge bus cycles that can be executed as part of interrupt exception processing. Table 7-4 provides a summary of the possible interrupt acknowledge terminations and the exception processing results. Note that TRA must always be negated for proper operation in the MC68040 acknowledge termination mode.

Table 7-4. Interrupt Acknowledge Termination Summary

Acknowledge Termination Mode	TA	TEA	TRA	AVEC	Termination Condition
Either	High	High	High	Don't Care	Insert Wait States
MC68040 Native-MC68060	High Don't Care	Low Low	High Don't Care	Don't Care Don't Care	Take Spurious Interrupt Exception
Either	Low	High	High	High	Register Vector Number on D7–D0 and Take Interrupt Exception
Either	Low	High	High	Low	Take Autovectored Interrupt Exception
MC68040 Native-MC68060	Low Don't Care	Low High	High Low	Don't Care Don't Care	Retry Interrupt Acknowledge Cycle
MC68040	Don't Care	Don't Care	Low	Don't Care	Illegal Combination, Unsupported

8.4 RETURN FROM EXCEPTIONS

Once the processor has completed processing of all exceptions, it must restore the machine context at the time of the initial exception before returning control to the original process.

Since the MC68060 is a complete restart machine, when the processor executes an RTE instruction, only three fields are referenced. The stack format is accessed (SP+6) and the frame type is first verified. If the format indicates an invalid type, a format error exception is signaled. Otherwise, the processor accesses the SR (SP) and PC (SP+2) fields from the top of the supervisor stack. If the PC value defines an odd address (least significant address bit is set), then an address error exception is signaled. Note that for the format error or the address error, the new stack frame will contain the SR value at the time the RTE's execution began, i.e., the SR has not been corrupted by the execution of the RTE. For either fault, the PC is the logical address of the RTE instruction.

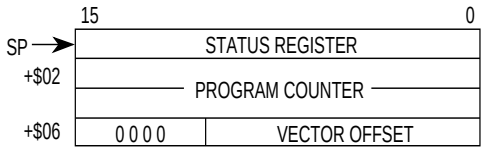
Given a valid stack format and a nonfaulting PC, the SR and PC are loaded with the stack operands, the SSP adjusted by the appropriate value determined by the format field, and control passed to the location defined by the new PC.

When the processor writes or reads a stack frame, it uses long-word operand transfers wherever possible. Using a long-word-aligned SP enhances exception processing performance. The processor does not necessarily read or write the stack frame data in sequential order. The following paragraphs discuss in detail each stack frame format.

Note that unlike any of the previous M68000 processors, the MC68060 RTE instruction treats the access error frame no differently from other frames.

8.4.1 Four-Word Stack Frame (Format $0)

If a four-word stack frame is on the stack and an RTE instruction is encountered, the processor updates the SR and PC with the data read from the stack, increments the stack pointer by eight, and resumes normal instruction execution

Stack Frames	Exception Types	Stacked PC Points To
 FOUR-WORD STACK FRAME–FORMAT $0	• Interrupt • Format Error • TRAP #N • Illegal Instruction • A-Line Instruction • F-Line Instruction • Privilege Violation • Floating-Point Pre-Instruction • Unimplemented Integer • Unimplemented Effective Address	• Next Instruction • RTE or FRESTORE Instruction • Next Instruction • Illegal Instruction • A-Line Instruction • F-Line Instruction • First Word of Instruction Causing Privilege Violation • Floating-Point Instruction • Unimplemented Integer Instruction • Instruction That Used the Unimplemented Effective Address

instruction is restarted, another table search is performed, and the instruction is executed successfully. If the access is not allowed, it is up to the system software designer to determine appropriate action.

For the physical bus error cases, as long as it is not one of the non-recoverable write cases, the exception handler must fix the page descriptor to point to a different physical memory, so that when the restart of the instruction occurs, that bus error does not recur.

It is important to note that the MC68060 performs table searches in hardware, and does not use the fetch table and page descriptors from the cache. The descriptor tables must be placed in non-cachable memory so that when the exception handler touches these descriptors, that the physical image in memory is updated properly.

The sixth step is to handle the default TTR cases. The default TTR is indicated if none of these bits are set: TTR, PTA, PTB, IL and PF. At this point, only the following cases are possible:

- WP = 1 (write protection violation detected by default TTR)
- RE = 1 (bus error on read)
- WE = 1 (bus error on write)

These cases may be handled similarly to step three. If the exception handler has gotten to this point, but none of the WP, RE and WE bits are set, and if the BPE bit is set and has been handled by the first step, then execute an RTE.

8.4.6 Bus Errors and Pending Memory Writes

The MC68060 processor contains two different write buffers for pending memory write operations: the store buffer and the push buffer. The store buffer is used to optimize performance by deferring bus write operations in write through and imprecise cache modes, and the push buffer holds displaced copyback mode cache lines and line write data for the MOVE16 instruction.

The push buffer holds a displaced cache line destined for memory until the cache-miss bus read access that caused the push completes. Imprecise cache modes (cachable write-through and copyback, and cache inhibited, imprecise) use the write buffers of the MC68060 to optimize system performance. Cache inhibited precise mode provides a precise exception model for MC68060 operation, not utilizing the write buffers (store or push).

When the MC68060 detects an exception condition, all instruction execution is aborted and the exception processing state is entered. Upon entering this state, the pipeline stalls until both the store and push buffers are empty before beginning exception processing. If a $\overline{\text{TEA}}$ signal termination occurs during a memory write cycle while emptying the store buffer, ‘a bus error $\overline{\text{TEA}}$ on store buffer’ is recorded and the buffer sequences through all the remaining, pending writes. However, if a $\overline{\text{TEA}}$ signal termination occurs during a memory write cycle while emptying the push buffer, ‘a bus error $\overline{\text{TEA}}$ on push buffer’ is recorded and the memory write operation is aborted immediately.

Table 9-3. Boundary Scan Bit Definitions (Continued)

Bit	Cell Type	Pin/Cell Name	Pin Type
130	O.Pin	A5	I/O
131	I.Pin	A5	I/O
132	IO.Ctl	A5–A4 ena	—
133	O.Pin	A6	I/O
134	I.Pin	A6	I/O
135	O.Pin	A7	I/O
136	I.Pin	A7	I/O
137	IO.Ctl	A9–A6 ena	—
138	O.Pin	A8	I/O
139	I.Pin	A8	I/O
140	O.Pin	A9	I/O
141	I.Pin	A9	I/O
142	O.Pin	D31	I/O
143	I.Pin	D31	I/O
144	O.Pin	D30	I/O
145	I.Pin	D30	I/O
146	IO.Ctl	D31–D28 ena	—
147	O.Pin	D29	I/O
148	I.Pin	D29	I/O
149	O.Pin	D28	I/O
150	I.Pin	D28	I/O
151	O.Pin	D27	I/O
152	I.Pin	D27	I/O
153	O.Pin	D26	I/O
154	I.Pin	D26	I/O
155	IO.Ctl	D27–D24 ena	—
156	O.Pin	D25	I/O
157	I.Pin	D25	I/O
158	O.Pin	D24	I/O
159	I.Pin	D24	I/O
160	O.Pin	D23	I/O
161	I.Pin	D23	I/O
162	O.Pin	D22	I/O
163	I.Pin	D22	I/O
164	IO.Ctl	D23–D20 ena	—
165	O.Pin	D21	I/O
166	I.Pin	D21	I/O
167	O.Pin	D20	I/O
168	I.Pin	D20	I/O
169	O.Pin	D19	I/O
170	I.Pin	D19	I/O
171	O.Pin	D18	I/O
172	I.Pin	D18	I/O
173	IO.Ctl	D19–D16 ena	—
174	O.Pin	D17	I/O
175	I.Pin	D17	I/O
176	O.Pin	D16	I/O
177	I.Pin	D16	I/O
178	O.Pin	D15	I/O

```
attribute INSTRUCTION_CAPTURE of MC68060: entity is "0101";
attribute INSTRUCTION_PRIVATE of MC68060:entity is "PRIVATE";
attribute REGISTER_ACCESS of MC68060:entity is
  "BOUNDARY (LPSAMPLE)";

attribute IDCODE_REGISTER of MC68060: entity is
  "0001" &        -- version
  "000001" &      -- design center
  "0000110000" &  -- sequence number
  "00000001110" & -- Motorola
  "1";            -- required by 1149.1

attribute BOUNDARY_CELLS of MC68060:entity is
"BC_1, BC_2, BC_4";

attribute BOUNDARY_LENGTH of MC68060:entity is 214;

attribute BOUNDARY_REGISTER of MC68060:entity is
--num cell  port    function safe  ccell dsval rslt
"0   (BC_1, D(0),   input,    X),   " &
"1   (BC_2, D(0),   output3,  X,      4,   0,   Z), " &
"2   (BC_1, D(1),   input,    X),   " &
"3   (BC_2, D(1),   output3,  X,      4,   0,   Z), " &
"4   (BC_2, *,      control,  0),   " & -- d[3:0]
"5   (BC_1, D(2),   input,    X),   " &
"6   (BC_2, D(2),   output3,  X,      4,   0,   Z), " &
"7   (BC_1, D(3),   input,    X),   " &
"8   (BC_2, D(3),   output3,  X,      4,   0,   Z), " &
"9   (BC_1, D(4),   input,    X),   " &
"10  (BC_2, D(4),   output3,  X,     13,   0,   Z), " &
"11  (BC_1, D(5),   input,    X),   " &
"12  (BC_2, D(5),   output3,  X,     13,   0,   Z), " &
"13  (BC_2, *,      control,  0),   " & -- d[7:4]
"14  (BC_1, D(6),   input,    X),   " &
"15  (BC_2, D(6),   output3,  X,     13,   0,   Z), " &
"16  (BC_1, D(7),   input,    X),   " &
"17  (BC_2, D(7),   output3,  X,     13,   0,   Z), " &
"18  (BC_1, D(8),   input,    X),   " &
"19  (BC_2, D(8),   output3,  X,     22,   0,   Z), " &
--num  cell port    function safe  ccell dsval rslt
"20  (BC_1, D(9),   input,    X),   " &
"21  (BC_2, D(9),   output3,  X,     22,   0,   Z), " &
"22  (BC_2, *,      control,  0),   " & -- d[11:8]
"23  (BC_1, D(10),  input,    X),   " &
"24  (BC_2, D(10),  output3,  X,     22,   0,   Z), " &
"25  (BC_1, D(11),  input,    X),   " &
"26  (BC_2, D(11),  output3,  X,     22,   0,   Z), " &
"27  (BC_1, D(12),  input,    X),   " &
"28  (BC_2, D(12),  output3,  X,     31,   0,   Z), " &
"29  (BC_1, D(13),  input,    X),   " &
"30  (BC_2, D(13),  output3,  X,     31,   0,   Z), " &
"31  (BC_2, *,      control,  0),   " & -- d[15:12]
"32  (BC_1, D(14),  input,    X),   " &
"33  (BC_2, D(14),  output3,  X,     31,   0,   Z), " &
```

interrupts). The 32-bit instruction address of the first instruction of the emulator interrupt exception handler is derived as with other exceptions—the memory contents of address VBR + exception offset ($30).

The emulator mode entry from the breakpoint exception shares the same vector table entry (VBR + $30) as the emulator interrupt exception. However, the emulator mode entry from the breakpoint exception requires that the exception handler increment the stacked PC by two to point to the instruction following the breakpoint instruction. On the other hand, the emulator interrupt stack's PC already points to the next instruction.

9.3 SWITCHING BETWEEN JTAG AND DEBUG PIPE CONTROL MODES OF OPERATION

Since JTAG and the debug pipe control modes share the same set of pins, only one mode can be used at a time. Normally, the JTAG mode is used only during product testing, and the debug pipe control mode is used by the end user in conjunction with an in-circuit emulator. For this use, the board manufacturer normally designs in whatever JTAG functionality is required without regard to whether the board will eventually be used in the debug pipe control mode or not. The responsibility of allowing the processor to operate under the debug pipe control mode lies with the emulator vendor. The emulator vendor needs to ensure that the socket built to carry the processor has the target system's JTAG pins isolated from the processor to allow full control of these pins. Hence, under normal circumstances, dynamic switching between JTAG and debug pipe control modes is unnecessary.

However, for systems that need to switch between these modes can do so by following some guidelines. These guidelines are illustrated in Figure 9-12 and Figure 9-13. These figures illustrate how to transition between the JTAG mode and the debug pipe control mode.

10.9 SHIFT/ROTATE EXECUTION TIMES

Table 10-13 indicates the number of clock cycles required for execution of the shift and rotate instructions. The number of operand read and write cycles is shown in parentheses (r/w). Where indicated, the number of clock cycles and r/w cycles must be added to those required for effective address calculation.

Table 10-13. Shift/Rotate Execution Times

Instruction	Size	Register	Memory[1]
ASL, ASR	Byte, Word	1(0/0)	1(1/1)
“	Long	1(0/0)	—
LSL, LSR	Byte, Word	1(0/0)	1(1/1)
“	Long	1(0/0)	—
ROL, ROR	Byte, Word	1(0/0)	1(1/1)
“	Long	1(0/0)	—
ROXL, ROXR	Byte, Word	1(0/0)	1(1/1)
“	Long	1(0/0)	—

[1] For entries in this column, add the effective address calculation time. These operations are word-size only.

10.10 BIT MANIPULATION AND BIT FIELD EXECUTION TIMES

Table 10-14 and Table 10-15 indicate the number of clock cycles required for execution of the bit manipulation instructions. The execution times for the bit field instructions is shown in Table 10-16. The number of operand read and write cycles is shown in parentheses (r/w). Where indicated, the number of clock cycles and r/w cycles must be added to those required for effective address calculation.

Table 10-14. Bit Manipulation (Dynamic Bit Count) Execution Times

Instruction	Size	Register	Memory[1]
BCHG	Byte	—	1(1/1)
“	Long	1(0/0)	—
BCLR	Byte	—	1(1/1)
“	Long	1(0/0)	—
BSET	Byte	—	1(1/1)
“	Long	1(0/0)	—
BTST	Byte	—	1(1/0)
“	Long	1(0/0)	—

[1] For entries in this column, add the effective address calculation time.

13.2.2 MC68060, MC68LC060, and MC68EC060

PIN GROUPS	GND (VSS)	VCC (VDD)
Internal Logic	2, 17, 24, 26, 27, 31, 46, 53, 64, 79, 90, 106, 113, 128, 142, 155, 169, 183, 197, 199	1, 18, 32, 53, 63, 78, 89, 105, 114, 127, 141, 156, 170, 184, 198
Output Drivers	4, 10, 42, 49, 58, 67, 73, 84, 87, 95, 100, 109, 117, 122, 131, 136, 145, 150, 161, 166, 175, 180, 189, 194, 204	6, 12, 40, 50, 60, 69, 75, 82, 92, 97, 102, 111, 119, 124, 133, 138, 147, 152, 159, 164, 173, 178, 187, 192, 202

The condition code register upon return from all of the library routines is correct. Figure C-5 provides a C-code representation of the integer library routines in the M68060SP.

```
/* 64-bit (32x32 -> 64) unsigned multiply routine */
void _mulu64(multiplier,multiplicand,result)
       unsigned int multiplier;
       unsigned int multiplicand;
       unsigned int *result; /* array for result */

/* 64-bit (32x32 -> 64) signed multiply routine */
void _muls64(multiplier,multiplicand,result)
       int multiplier;
       int multiplicand;
       int *result; /* array for result */

/* 64-bit (32/32 -> 32r:32q) unsigned divide routine */
void _divu64(divisor,dividend_hi,dividend_lo,result)
       unsigned int divisor;
       unsigned int dividend_hi, dividend_lo;
       unsigned int *result; /* array for result */

/* 64-bit (32/32 -> 32r:32q) signed divide routine */
void _divs64(divisor,dividend_hi,dividend_lo,result)
       int divisor;
       int dividend_hi,dividend_lo;
       int *result; /* array for result */

/* CMP2 using an “A”ddress or “D”ata register. size = byte. */
void _cmp2_{D,A}b(rn,bounds)
       int rn;
       char *bounds; /* pointer to byte bounds array */

/* CMP2 using an “A”ddress or “D”ata register. size = word. */
void _cmp2_{D,A}w(rn,bounds)
       int rn;
       short *bounds; /* pointer to word bounds array */

/* CMP2 using an “A”ddress or “D”ata register. size = longword. */
void _cmp2_{D,A}l(rn,bounds)
       int rn;
       int *bounds; /* pointer to longword bounds array */
```

Figure C-5. C-Code Representation of Integer Library Routines

For example, to use a 64-bit divide instruction, do a “bsr” or “jsr” to the entry-point defined by the MC68060ILSP entry table. A compiler-generated code sequence for unsigned multiply could resemble Figure C-6.

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 is not lost while popping other items off of the stack. An example of using the CMP2 instruction is given in Figure C-7.

The unimplemented integer instruction library module contains no operating system dependencies and does not require a call-out dispatch table. If the instruction being emulated is a

Table D-3. Exception Vector Assignments for the M68000 Family

Vector Number(s)	Vector Offset (Hex)	Assignment
0	000	Reset Initial Interrupt Stack Pointer
1	004	Reset Initial Program Counter
2	008	Access Fault
3	00C	Address Error
4	010	Illegal Instruction
5	014	Integer Divide-by-Zero
6	018	CHK, CHK2 Instruction
7	01C	FTRAPcc, TRAPcc, TRAPV Instructions
8	020	Privilege Violation
9	024	Trace
10	028	Line 1010 Emulator (Unimplemented A-Line Opcode)
11	02C	Line 1111 Emulator (Unimplemented F-Line Opcode)
12	030	(Reserved)
13	034	Coprocessor Protocol Violation (Defined for MC68020 and MC68030)
14	038	Format Error
15	03C	Uninitialized Interrupt
16–23	040–05C	(Unassigned, Reserved)
24	060	Spurious Interrupt
25	064	Level 1 Interrupt Autovector
26	068	Level 2 Interrupt Autovector
27	06C	Level 3 Interrupt Autovector
28	070	Level 4 Interrupt Autovector
29	074	Level 5 Interrupt Autovector
30	078	Level 6 Interrupt Autovector
31	07C	Level 7 Interrupt Autovector
32–47	080–0BC	TRAP #0–15 Instruction Vectors
48	0C0	Floating-Point Branch or Set on Unordered Condition (Defined for MC68881, MC68882, MC68040, and MC68060)
49	0C4	Floating-Point Inexact Result (Defined for MC68881, MC68882, MC68040, and MC68060)
50	0C8	Floating-Point Divide-by-Zero (Defined for MC68881, MC68882, MC68040, and MC68060)
51	0CC	Floating-Point Underflow (Defined for MC68881, MC68882, MC68040, and MC68060)
52	0D0	Floating-Point Operand Error (Defined for MC68881, MC68882, MC68040, and MC68060)
53	0D4	Floating-Point Overflow (Defined for MC68881, MC68882, MC68040, and MC68060)
54	0D8	Floating-Point Signaling NAN (Defined for MC68881, MC68882, MC68040, and MC68060)
55	0DC	Floating-Point Unimplemented Data Type (Defined for MC68040 and MC68060)
56	0E0	MMU Configuration Error (Defined for MC68030 and MC68851)
57	0E4	MMU Illegal Operation Error (Defined for MC68851)
58	0E8	MMU Access Level Violation Error (Defined for MC68851)
59	0EC	(Unassigned, Reserved)
60	0F0	Unimplemented Effective Address (Defined for MC68060)
61	0F4	Unimplemented Integer Instruction (Defined for MC68060)
62–63	0F8–0FC	(Unassigned, Reserved)
64–255	100–3FC	User Defined Vectors (192)

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.