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/pro/item?MUrl=&PartUrl=mc68lc060rc50

Table 2-7. PSTx Encoding

Hex	PST4	PST3	PST2	PST1	PST0	Internal Processor Status
$00	0	0	0	0	0	Continue Execution in User Mode
$01	0	0	0	0	1	Complete 1 Instruction in User Mode
$02	0	0	0	1	0	Complete 2 Instructions in User Mode
$03	0	0	0	1	1	—
$04	0	0	1	0	0	—
$05	0	0	1	0	1	—
$06	0	0	1	1	0	—
$07	0	0	1	1	1	—
$08	0	1	0	0	0	Emulator Mode Entry Exception Processing
$09	0	1	0	0	1	Complete Not Taken Branch in User Mode
$0A	0	1	0	1	0	Complete Not Taken Branch Plus 1 Instruction in User Mode
$0B	0	1	0	1	1	IED Cycle of Branch to Vector, Emulator Entry Exception
$0C	0	1	1	0	0	—
$0D	0	1	1	0	1	Complete Taken Branch in User Mode
$0E	0	1	1	1	0	Complete Taken Branch Plus 1 Instruction in User Mode
$0F	0	1	1	1	1	Complete Taken Branch Plus 2 Instructions in User Mode
$10	1	0	0	0	0	Continue Execution in Supervisor Mode
$11	1	0	0	0	1	Complete 1 Instruction in Supervisor Mode
$12	1	0	0	1	0	Complete 2 Instructions in Supervisor Mode
$13	1	0	0	1	1	—
$14	1	0	1	0	0	—
$15	1	0	1	0	1	Complete RTE Instruction in Supervisor Mode
$16	1	0	1	1	0	Low-Power Stopped State; Waiting for an Interrupt or Reset
$17	1	0	1	1	1	MC68060 Is Stopped Waiting for an Interrupt
$18	1	1	0	0	0	MC68060 Is Processing an Exception
$19	1	1	0	0	1	Complete Not Taken Branch in Supervisor Mode
$1A	1	1	0	1	0	Complete Not Taken Branch Plus 1 Instruction in Supervisor Mode
$1B	1	1	0	1	1	IED Cycle of Branch to Vector, Exception Processing
$1C	1	1	1	0	0	MC68060 Is Halted
$1D	1	1	1	0	1	Complete Taken Branch in Supervisor Mode
$1E	1	1	1	1	0	Complete Taken Branch Plus 1 Instruction in Supervisor Mode
$1F	1	1	1	1	1	Complete Taken Branch Plus 2 Instructions in Supervisor Mode

2.10.2 MC68060 Processor Clock (CLK)

CLK is the synchronous clock of the MC68060. This signal is used internally to clock or sequence the internal logic of the MC68060 processor and is qualified with $\overline{\text{CLKEN}}$ to clock all external bus signals.

Since the MC68060 is designed for static operation, CLK can be gated off to lower power dissipation (e.g., during low-power stopped states). Refer to **Section 7 Bus Operation** for more information on low-power stopped states.

2.10.3 Clock Enable ($\overline{\text{CLKEN}}$)

This input signal is a qualifier for the MC68060 processor clock (CLK) and is provided to support lower bus frequency MC68060 designs. The internal MC68060 bus interface controller will sample, assert, negate, or three-state signals (except for $\overline{\text{BB}}$ and $\overline{\text{TIP}}$ which can three-

FITC—1/2-Cache Mode (Instruction ATC)

0 = The instruction ATC operates with 64 entries.
1 = The instruction ATC operates with 32 entries.

DCO—Default Cache Mode (Data Cache)

00 = Writethrough, cachable
01 = Copyback, cachable
10 = Cache-inhibited, precise exception model
11 = Cache-inhibited, imprecise exception model

DUO—Default UPA bits (Data Cache)

These bits are two user-defined bits for operand accesses (see **4.2.2.3 Descriptor Field Definitions**).

DWO—Default Write Protect (Data Cache)

0 = Reads and writes are allowed.
1 = Reads are allowed, writes cause a protection exception.

DCI—Default Cache Mode (Instruction Cache)

00 = Writethrough, cachable
01 = Copyback, cachable
10 = Cache-inhibited, precise exception model
11 = Cache-inhibited, imprecise exception model

DUI—Default UPA Bits (Instruction Cache)

These bits are two user-defined bits for instruction prefetch bus cycles (see **4.2.2.3 Descriptor Field Definitions**)

Bit 0—Reserved by Motorola. Always read as zero.

4.2.2.3 DESCRIPTOR FIELD DEFINITIONS. The field definitions for the table- and page-level descriptors are listed in alphabetical order:

CM—Cache Mode

This field selects the cache mode and accesses serialization as follows:

00 = Cachable, Writethrough
01 = Cachable, Copyback
10 = Cache-Inhibited, Precise exception model
11 = Cache-Inhibited, Imprecise exception model

Section 5 Caches provides detailed information on caching modes.

Descriptor Address

This 30-bit field, which contains the physical address of a page descriptor, is only used in indirect descriptors.

G—Global

When this bit is set, it indicates the entry is global which gives the user the option of grouping entries as global or nonglobal for use when PFLUSHing the ATC, and has no other meaning. PFLUSH instruction variants that specify nonglobal entries do not invalidate global entries, even when all other selection criteria are satisfied. If these PFLUSH variants are not used, then system software can use this bit.

M—Modified

This bit identifies a page which has been written to by the processor. The MC68060 sets the M-bit in the corresponding page descriptor before a write operation to a page for which the M-bit is clear, except for write-protect or supervisor violations in which case the M-bit is not set. The read portion of a locked read-modify-write access is considered a write for updating purposes. The MC68060 never clears this bit.

PDT—Page Descriptor Type

This field identifies the descriptor as an invalid descriptor, a page descriptor for a resident page, or an indirect pointer to another page descriptor.

00 = Invalid
This code indicates that the descriptor is invalid. An invalid descriptor can represent a nonresident page or a logical address range that is out of bounds. All other bits in the descriptor are ignored. When an invalid descriptor is encountered, an ATC entry is not created.

01 or 11 = Resident
These codes indicate that the page is resident.

10 = Indirect
This code indicates that the descriptor is an indirect descriptor. Bits 31–2 contain the physical address of the page descriptor. This encoding is invalid for a page descriptor pointed to by an indirect descriptor (that is, only one level of indirection is allowed).

updated before the MC68060 allows a page to be accessed. Table 4-1 lists the page descriptor update operations for each combination of U-bit, M-bit, write-protected, and read or write access type.

Table 4-1. Updating U-Bit and M-Bit for Page Descriptors

Previous Status		WP Bit	Access Type	Page Descriptor Update Operation	New Status	
U-Bit	M-Bit				U-Bit	M-Bit
0	0	X	Read	Locked RMW Access to Set U	1	0
0	1			Locked RMW Access to Set U	1	1
1	0			None	1	0
1	1			None	1	1
0	0	0	Write	Write to Set U and M	1	1
0	1			Write to Set U	1	1
1	0			Write to Set M	1	1
1	1			None	1	1
0	0	1		None	0	0
0	1			None	0	1
1	0			None	1	0
1	1			None	1	1

NOTE: WP indicates the accumulated write-protect status.

An alternate address space access is a special case that is immediately used as a physical address without translation. Because the MC68060 implements a merged instruction and data space, instruction address spaces (SFC/DFC = $6 or $2) using the MOVES instruction are converted into data references (SFC/DFC = $5 or $1). The data memory unit handles these translated accesses as normal data accesses. If the access fails due to an ATC fault or a physical bus error, the resulting access error stack frame contains the converted function code in the TM field for the faulted access. If the MOVES instruction is used to write instruction address space, then to maintain cache coherency, the corresponding addresses must be invalidated in the instruction cache. The SFC and DFC values and results for normal (TT = 0) and for MOVES (TT = 10) accesses are listed in Table 4-2.

Table 4-2. SFC and DFC Values

SFC/DFC Value	Results	
	TT	TM
000	10	000
001	00	001
010	00	001
011	10	011
100	10	100
101	00	101
110	00	101
111	10	111

4.2.6 Address Translation Protection

The MC68060 MMUs provide separate translation tables for supervisor and user address spaces. The translation tables contain both mapping and protection information. Each table and page descriptor includes a write-protect (W) bit that can be set to provide write protec-

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.

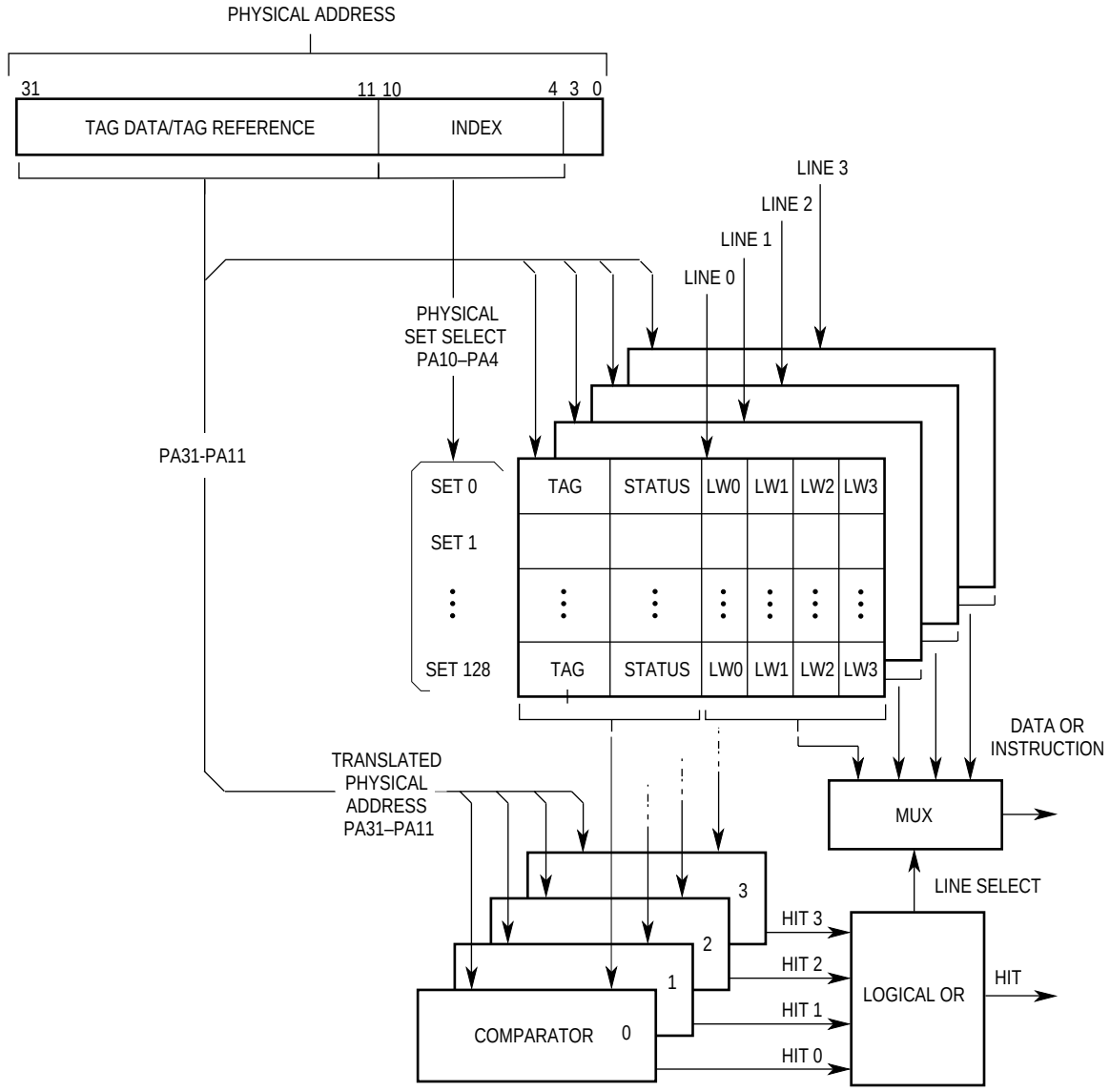


Figure 5-4. Caching Operation

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

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.

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.

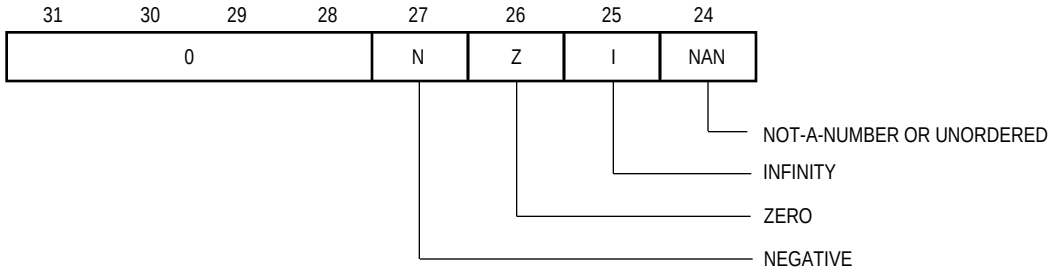


Figure 6-4. Floating-Point Condition Code (FPSR)

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.

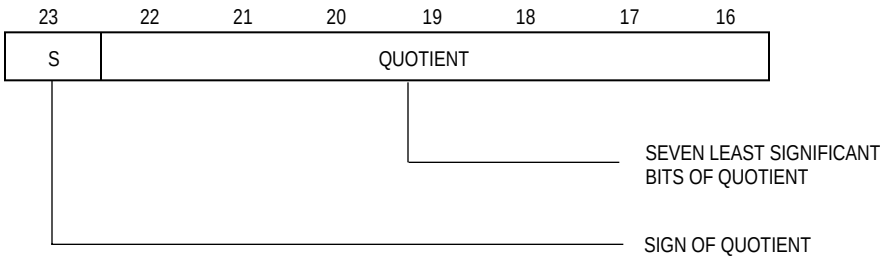


Figure 6-5. Floating-Point Quotient Byte (FPSR)

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.

cycles, the bus controller still treats the four transfers as a single line bus cycle and does not allow other unrelated processor accesses or bus arbitration to intervene between the transfers. $\overline{TBI}$ is ignored after the first long-word transfer.

Line reads to support cache line filling can be cache inhibited by asserting transfer cache inhibit ($\overline{TCI}$) with $\overline{TA}$ for the first long-word transfer of the line. The assertion of $\overline{TCI}$ does not affect completion of the line transfer, but the bus controller registers and passes it to the memory controller for use. $\overline{TCI}$ is ignored after the first long-word transfer of a line burst bus cycle and during the three long-word bus cycles of a burst-inhibited line transfer.

The address placed on the address bus by the processor for line bus cycle does not necessarily point to the most significant byte of each long word because A1 and A0 are copied from the original operand address supplied to the memory unit by the integer unit for line reads. These two bits are also unchanged for the three long-word bus cycles of a burst-inhibited line transfer. The selected device should ignore A1 and A0 for long-word and line read transfers.

The address of an instruction fetch will always be aligned to a long-word boundary ($xxxxxxx0, $xxxxxxx4, $xxxxxxx8, or $xxxxxxxC). This is unlike the MC68040 in which the prefetches occur on half-line boundaries. Therefore, compilers should attempt to locate branch targets on long-word boundaries to minimize branch stalls. For example, if the target of a branch is an instruction that starts at $1000000E, the following burst sequence will occur upon a cache miss: $1000000C, $10000000, $10000004, then $10000008. Figure 7-14 and Figure 7-15 illustrate a flowchart and functional timing diagram for a line read bus transfer.

Clock 1 (C1)

The line read cycle starts in C1. During C1, the processor places valid values on the address bus and transfer attributes. For user and supervisor mode accesses that are translated by the corresponding memory unit, the UPAx signals are driven with the values from the matching U1 and U0 bits. The TTx and TMx signals identify the specific access type. The R/$\overline{W}$ signal is driven high for a read cycle, and the size signals (SIZx) indicate line size. $\overline{CIOUT}$ is asserted for a MOVE16 operand read if the access is identified as noncachable. Refer to **Section 4 Memory Management Unit** for information on the MC68060 and MC68LC060 memory units and **Appendix B MC68EC060** for information on the MC68EC060 memory unit.

The processor asserts $\overline{TS}$ during C1 to indicate the beginning of a bus cycle. If not already asserted from a previous bus cycle, $\overline{TIP}$ is also asserted at this time to indicate that a bus cycle is active.

Clock 2 (C2)

During C2, the processor negates $\overline{TS}$. The selected device uses R/$\overline{W}$ and SIZx to place the data on the data bus. (The first transfer must supply the long word at the corresponding long-word boundary.) Concurrently, the selected device asserts $\overline{TA}$ and either negates $\overline{TBI}$ to indicate it can or asserts $\overline{TBI}$ to indicate it cannot support a burst transfer.

The MC68060 implements a special mode called the acknowledge termination ignore state capability to aid in high-frequency designs. In this mode, the processor begins sampling termination signals such as $\overline{TA}$ after a user-programmed number of BCLK rising

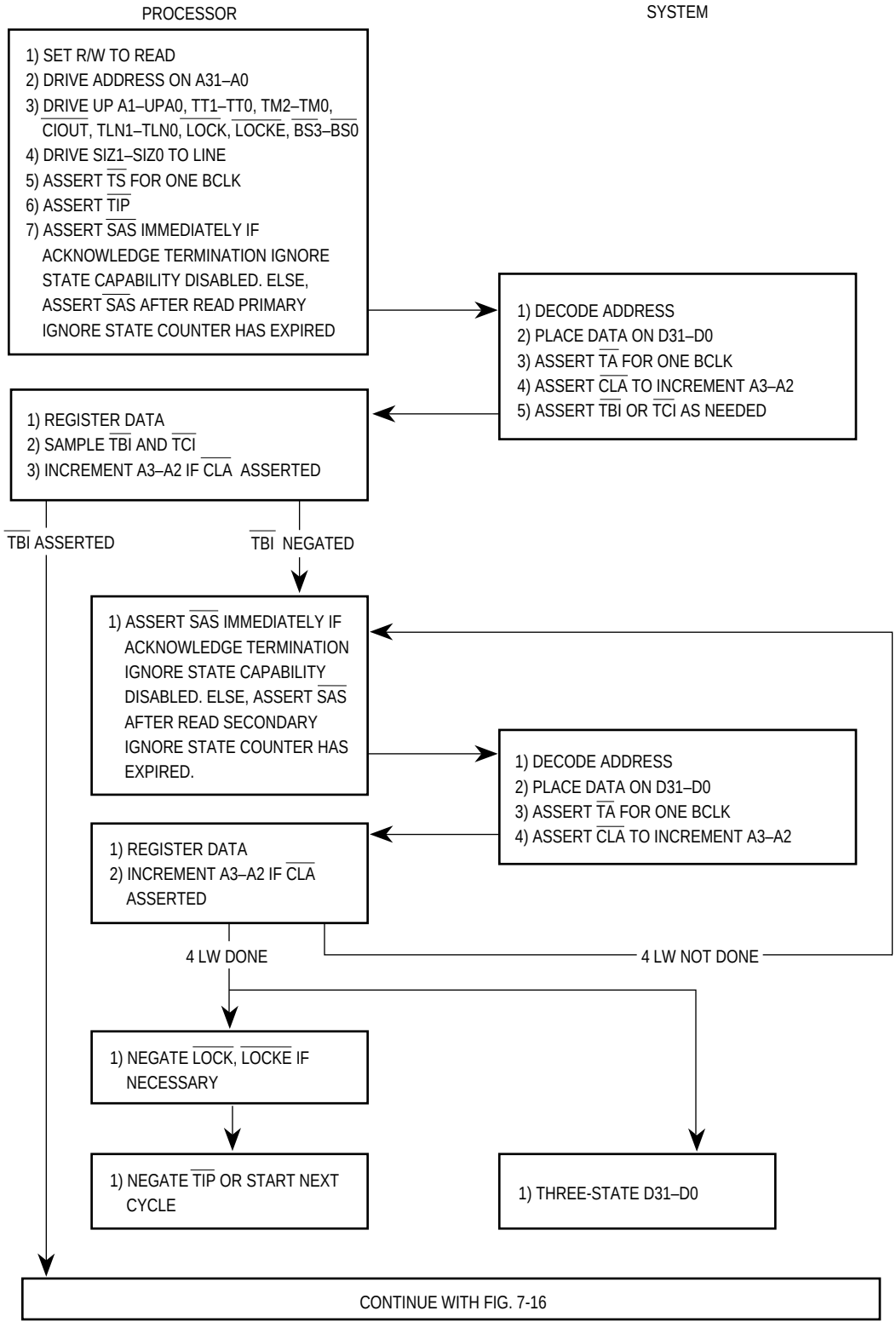


Figure 7-14. Line Read Cycle Flowchart

edges has expired. The signal SAS is provided as a status output to indicate which BCLK rising edge the processor begins to sample the termination signals. If this mode is dis-

7.8.1.1 INTERRUPT ACKNOWLEDGE CYCLE (TERMINATED NORMALLY). When the MC68060 processes an interrupt exception, it performs an interrupt acknowledge bus cycle to obtain the vector number that contains the starting location of the interrupt exception handler. Some interrupting devices have programmable vector registers that contain the interrupt vectors for the exception handlers they use. Other interrupting conditions or devices cannot supply a vector number and use the autovector bus cycle described in **7.8.1.2 Autovector Interrupt Acknowledge Cycle**.

The interrupt acknowledge bus cycle is a read transfer. It differs from a normal read cycle in the following respects:

- TT1 and TT0 = $3 to indicate an acknowledged bus cycle
- Address signals A31–A0 are set to all ones ($FFFFFFFF)
- TM2–TM0 are set to the interrupt request level (the inverted values of $\overline{\text{IPLx}}$).

The responding device places the vector number on the lower byte of the data bus during the interrupt acknowledge bus cycle, and the cycle is terminated normally with $\overline{\text{TA}}$. Figure 7-27 and Figure 7-28 illustrate a flowchart and functional timing diagram for an interrupt acknowledge cycle terminated with $\overline{\text{TA}}$.

Note that the acknowledge termination ignore state capability is applicable to the interrupt acknowledge cycle. If enabled, $\overline{\text{TA}}$ and other acknowledge termination signals are ignored for a user-programmed number of BCLK cycles.

7.8.1.2 AUTOVECTOR INTERRUPT ACKNOWLEDGE CYCLE. When the interrupting device cannot supply a vector number, it requests an automatically generated vector (autovector). Instead of placing a vector number on the data bus and asserting $\overline{\text{TA}}$, the device asserts the autovector ($\overline{\text{AVEC}}$) signal with $\overline{\text{TA}}$ to terminate the cycle. $\overline{\text{AVEC}}$ is only sampled with $\overline{\text{TA}}$ asserted. $\overline{\text{AVEC}}$ can be grounded if all interrupt requests are autovectored.

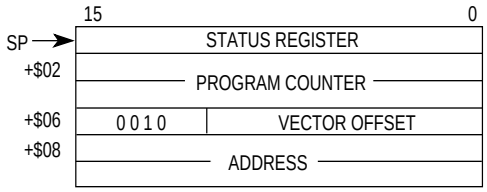
The vector number supplied in an autovector operation is derived from the interrupt priority level of the current interrupt. When the $\overline{\text{AVEC}}$ signal is asserted with $\overline{\text{TA}}$ during an interrupt acknowledge bus cycle, the MC68060 ignores the state of the data bus and internally generates the vector number, which is the sum of the interrupt priority level plus 24 ($18). There are seven distinct autovectors that can be used, corresponding to the seven levels of interrupts available with $\overline{\text{IPLx}}$ signals. Figure 7-29 illustrates a functional timing diagram for an autovector operation.

Note that the acknowledge termination ignore state capability is applicable to the interrupt acknowledge cycle. If enabled, $\overline{\text{AVEC}}$ and other acknowledge termination signals are ignored for a user-programmed number of BCLK cycles.

7.8.1.3 SPURIOUS INTERRUPT ACKNOWLEDGE CYCLE. When a device does not respond to an interrupt acknowledge bus cycle, spurious with $\overline{\text{TA}}$, or $\overline{\text{AVEC}}$ and $\overline{\text{TA}}$, the external logic typically returns the transfer error acknowledge signal ($\overline{\text{TEA}}$). In this case, the MC68060 automatically generates the spurious interrupt vector number 24 ($18) instead of the interrupt vector number. If operating in the MC68040 acknowledge termination mode,

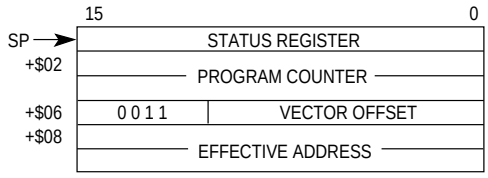
8.4.2 Six-Word Stack Frame (Format $2)

If a six-word stack frame is on the stack and an RTE instruction is encountered, the processor restores the SR and PC values from the stack, increments the SSP by $C, and resumes normal instruction execution.

Stack Frames	Exception Types	Stacked PC Points To; Address Field Has
 SIX-WORD STACK FRAME–FORMAT $2	• CHK, CHK2 (Emulated), TRAPcc, FTRAPcc(Emulated), TRAPV, Trace, or Zero Divide • Unimplemented Floating-Point Instruction • Address Error	• Next Instruction; Address field has the address of the instruction that caused the exception. • Next Instruction; Address field has the calculated <ea> for the floating-point instruction. • Instruction that caused the address error; Address field has the branch target address with A0=0.

8.4.3 Floating-Point Post-Instruction Stack Frame (Format $3)

In this case, the processor restores the SR and PC values from the stack and increments the supervisor stack pointer by $C. If another floating-point post-instruction exception is pending, exception processing begins immediately for the new exception; otherwise, the processor resumes normal instruction execution.

Stack Frames	Exception Types	Stacked PC Points To; Effective Address Field
 FLOATING-POINT POST-INSTRUCTION STACK FRAME–FORMAT $3	• Floating-Point Post-Instruction	• Next Instruction; Effective Address field is the calculated effective address for the floating-point instruction.

through at least five rising edges and the falling edge after the fifth rising edge. A JTAG reset will cause the TAP state machine to enter the test-logic-reset state (normal operation of the TAP state machine into the test-logic-reset state will also result in placing the default value of $5 into the instruction register). The shift register portion of the instruction register is loaded with the default value of $5 when in the Capture-IR state and a rising edge of TCK occurs.

9.1.2.6 CLAMP. The CLAMP instruction selects the bypass register while simultaneously forcing all output pins and bidirectional pins configured as outputs, to the fixed values that are preloaded and held in the boundary scan update registers. This instruction enhances test efficiency by reducing the overall shift path to a single bit (the bypass register) while conducting an EXTEST type of instruction through the boundary scan register. The CLAMP instruction becomes active on the falling edge of TCK in the update-IR state when the data held in the instruction shift register is equivalent to $6.

It is recommended that the boundary scan register bit equivalent to the $\overline{\text{RSTI}}$ pin be preloaded with the assert value for system reset prior to application of the CLAMP instruction. This will ensure that CLAMP asserts the internal reset for the MC68060 system logic to force a predictable benign internal state while isolating all pins from signals generated external to the part. However, if it is desired to hold the processor in the LPSTOP state when applying the CLAMP instruction, do not preload the boundary scan register bit equivalent to the $\overline{\text{RSTI}}$ pin with an assert value because this action forces the processor out of the LPSTOP state.

9.1.2.7 HIGHZ. The HIGHZ instruction is an IEEE 1149.1 option that is provided as a Motorola public instruction designed to anticipate the need to backdrive the output pins and protect the input pins from random toggling during circuit board testing. The HIGHZ instruction selects the bypass register, forces all output and bidirectional pins to the high-impedance state, and isolates all input signal pins except for CLK, IPL, and $\overline{\text{RSTI}}$. The HIGHZ instruction becomes active on the falling edge of TCK in the update-IR state when the data held in the instruction shift register is equivalent to $7.

It is recommended that the boundary scan register bit equivalent to the $\overline{\text{RSTI}}$ pin be preloaded with the assert value for system reset prior to application of the HIGHZ instruction. This will ensure that HIGHZ asserts the internal reset for the MC68060 system logic to force a predictable benign internal state while isolating all pins from signals generated external to the part.

9.1.2.8 BYPASS. The BYPASS instruction selects the single-bit bypass register, creating a single bit shift register path from the TDI pin to the bypass register to the TDO pin. This instruction enhances test efficiency by reducing the overall shift path when a device other than the MC68060 becomes the device under test on a board design with multiple chips on the overall IEEE-1149.1-defined boundary scan chain. The bypass register has been implemented in accordance with IEEE 1149.1 so that the shift register stage is set to logic zero on the rising edge of TCK following entry into the capture-DR state. Therefore, the first bit to be shifted out after selecting the bypass register is always a logic zero (this is to differentiate a part that supports an idcode register from a part that supports only the bypass register).

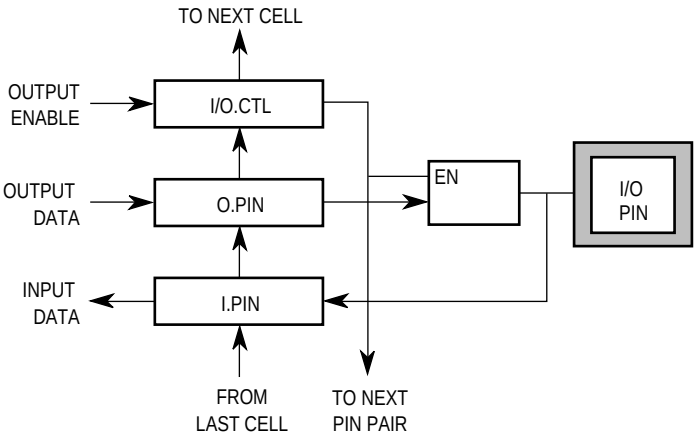


Figure 9-7. General Arrangement of Bidirectional Pin Cells

Table 9-3. Boundary Scan Bit Definitions

Bit	Cell Type	Pin/Cell Name	Pin Type
0	O.Pin	A31	I/O
1	I.Pin	A31	I/O
2	O.Pin	A30	I/O
3	I.Pin	A30	I/O
4	IO.Ctl	A31–A28 ena	—
5	O.Pin	A29	I/O
6	I.Pin	A29	I/O
7	O.Pin	A28	I/O
8	I.Pin	A28	I/O
9	O.Pin	A27	I/O
10	I.Pin	A27	I/O
11	O.Pin	A26	I/O
12	I.Pin	A26	I/O
13	IO.Ctl	A27–A24 ena	—
14	O.Pin	A25	I/O
15	I.Pin	A25	I/O
16	O.Pin	A24	I/O
17	I.Pin	A24	I/O
18	O.Pin	A23	I/O
19	I.Pin	A23	I/O
20	O.Pin	A22	I/O
21	I.Pin	A22	I/O
22	IO.Ctl	A23–A20 ena	—
23	O.Pin	A21	I/O
24	I.Pin	A21	I/O
25	O.Pin	A20	I/O
26	I.Pin	A20	I/O
27	O.Pin	A19	I/O
28	I.Pin	A19	I/O
29	O.Pin	A18	I/O
30	I.Pin	A18	I/O
31	IO.Ctl	A19–A16 ena	—

```
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), " &
```

Table 10-3. Superscalar Classification of M680x0 Privileged Instructions

Mnemonic	Instruction	Superscalar Classification
ANDI to SR	AND Immediate to Status Register	pOEP-only
CINV	Invalidate Cache Lines	pOEP-only
CPUSH	Push and Invalidate Cache Lines	pOEP-only
EORI to SR	Exclusive OR Immediate to Status Register	pOEP-only
MOVE from SR	Move from Status Register	pOEP-only
MOVE to SR	Move to Status Register	pOEP-only
MOVE USP	Move User Stack Pointer	pOEP-only
MOVEC	Move Control Register	pOEP-only
MOVES	Move Address Space	pOEP-only
ORI to SR	Inclusive OR Immediate to Status Register	pOEP-only
PFLUSH	Flush ATC Entries	pOEP-only
PLPA	Load Physical Address	pOEP-only
RESET	Reset External Devices	pOEP-only
RTE	Return from Exception	pOEP-only
STOP	Load Status Register and Stop	pOEP-only

Table 10-4. Superscalar Classification of M680x0 Floating-Point Instructions

Mnemonic	Instruction	Superscalar Classification
FABS, FDABS, FSABS	Absolute Value	pOEP-but-allows-sOEP[1]
FADD, FDADD, FSADD	Add	pOEP-but-allows-sOEP[1]
FBcc	Branch Conditionally	pOEP-only
FCMP	Compare	pOEP-but-allows-sOEP[1]
FDIV, FDDIV, FSDIV, FSGLDIV	Divide	pOEP-but-allows-sOEP[1]
FINT, FINTRZ	Integer Part, Round-to-Zero	pOEP-but-allows-sOEP[1]
FMOVE, FDMOVE, FSMOVE	Move Floating-Point Data Register	pOEP-but-allows-sOEP[1]
FMOVE	Move System Control Register	pOEP-only
FMOVEM	Move Multiple Data Registers	pOEP-only
FMUL, FDMUL, FSMUL, FSGLMUL	Multiply	pOEP-but-allows-sOEP[1]
FNEG, FDNEG, FSNEG	Negate	pOEP-but-allows-sOEP[1]
FNOP	No Operation	pOEP-only
FSQRT	Square Root	pOEP-but-allows-sOEP[1]
FSUB, FDSUB, FSSUB	Subtract	pOEP-but-allows-sOEP[1]
FTST	Test Operand	pOEP-but-allows-sOEP[1]

[1] These floating-point instructions are pOEP-but-allows-sOEP except for the following:
F<op>Dm,FPn
F<op>&imm,FPn
F<op>.x<mem>,FPn
which are classified as pOEP-only

The MC68060 superscalar architecture allows pairs of single-cycle standard operations to be simultaneously dispatched in the operand execution pipelines. Additionally, the design also permits a single-cycle standard instruction plus a conditional branch (Bcc) predicted by the branch cache to be dispatched in the OEP. Bcc instructions predicted as not taken allow another instruction to be executed in the sOEP. This also is true for forward Bcc instructions that are not predicted.

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)	—	—

10.16 EXCEPTION PROCESSING TIMES

Table 10-26 indicates the number of clock cycles required for exception processing. The number of clock cycles includes the time spent in the OEP by the instruction causing the exception, the stacking of the exception frame, the vector fetch, and the fetch of the first instruction of the exception handler routine. The number of operand read and write cycles is shown in parentheses (r/w).

Table 10-26. Exception Processing Times

Exception	Execution Time
CPU Reset	45(2/0)[1]
Bus Error	19(1/4)
Address Error	19(1/3)
Illegal Instruction	19(1/2)
Integer Divide By Zero	20(1/3)[2]
CHK Instruction	20(1/3)[2]
TRAPV, TRAPcc Instructions	19(1/3)
Privilege Violation	19(1/2)
Trace	19(1/3)
Line A Emulator	19(1/2)
Line F Emulator	19(1/2)
Unimplemented EA	19(1/2)
Unimplemented Integer	19(1/2)
Format Error	23(1/2)
Nonsupported FP	19(1/3)
Interrupt[3]	23(1/2)
TRAP Instructions	19(1/2)
FP Branch on Unordered Condition	21(1/3)
FP Inexact Result	19(1/3)[4]
FP Divide By Zero	19(1/3)[4]
FP Underflow	19(1/3)[4]
FP Operand Error	19(1/3)[4]
FP Overflow	19(1/3)[4]
FP Signaling NAN	19(1/3) [4]
FP Unimplemented Data Type	19(1/3)

[1] Indicates the time from when RSTI is negated until the first instruction enters the OEP.
[2] For these entries, add the effective address calculation time.
[3] Assumes either autovector or external vector supplied with zero wait states.
[4] For these entries, add the instruction execution time minus 1 if a post-exception fault occurs.

APPENDIX A
MC68LC060

The MC68LC060 is a derivative of the MC68060. The MC68LC060 has the same execution unit and MMU as the MC68060, but has no FPU. The MC68LC060 is 100% pin compatible with the MC68060. Disregard all information concerning the FPU when reading this manual. The following difference exists between the MC68LC060 and the MC68060:

- The MC68LC060 does not contain an FPU. When floating-point instructions are encountered, a floating-point disabled exception is taken.
- Bits 31–16 of the processor configuration register contain 0000010000110001, identifying the device as an MC68LC/EC060.

C.3.4 Module 5: Floating-Point Library (M68060FPLSP)

The M68060SP provides a library version of the unimplemented general monadic and dyadic floating-point instructions shown in Table C-3. These routines are System V ABI compliant as well as IEEE exception-reporting compliant. They are not, however, UNIX exception-reporting compliant. This library implementation can be compiled with user applications desiring the functionality of these instructions without having to incur the overhead of the floating-point unimplemented instruction" exception. The floating-point library contains floating-point instructions that are implemented by the MC68060. The floating-point library requires the partial floating-point kernel or full floating-point kernel to be ported to the system for proper operation.

In addition, the FABS, FADD, FDIV, FINT, FINTRZ, FMUL, FNEG, FSQRT, and FSUB functions are provided for the convenience of older compilers that make subroutine calls for all floating-point instructions. The code does *not* emulate these instructions in integer, but rather simply executes them.

All input variables must be pushed onto the stack prior to calling the supplied library routine. For each function, three entry points are provided, each accepting one of the three possible input operand data types: single, double, and extended precision. For dyadic operations both input operands are defined to have the same operand data type. For instance, for a monadic instruction such as the FSIN instruction, the functions are: _fsins(single-precision input operand), _fsind(double-precision input operand), _fsinx(extended-precision input operand). For dyadic operations such as the FDIV instruction, the entry points provided are: _fdivs(both single-precision input operands), _fdivd(both double-precision input operands, _fdivx(both extended-precision input operands).

To properly call a monadic subroutine, the calling routine must push the input operand onto the stack first. For instance:

```
* This example replaces the "fsin.x fp1,fp0" instruction
* Note that _fsinx is actually implemented as an offset from the
* top of the Floating-point Library Module.
fmove.x fp1,-(sp)   ; push operand to stack
bsr     _fsinx      ; result returned in fp0
add.w   #12,sp      ; clean up stack
```

To properly call a dyadic subroutine, the calling routine must push the second operand onto the stack before pushing the first operand onto the stack. For instance:

```
* This example replaces the "fdiv.x fp1,fp0" instruction
* Note that _fdivx is actually implemented as an offset from the
* top of the Floating-point Library Module.
fmove.x fp1,-(sp)   ; push 2nd operand to stack
fmove.x fp0,-(sp)   ; push 1st operand to stack
bsr     _fdivx      ; result returned in fp0
add.w   #24,sp      ; clean up stack
```

All routines return the operation result in the register fp0. It is the responsibility of the calling routine to remove the input operands from the stack after the routine has been executed. The result's rounding precision and mode, as well as exception reporting, is dictated by the value of the FPCR upon subroutine entry. The floating-point status register (FPSR) is set

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)