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	Active
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	5V
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=mc68ec060rc75

state on the rising edge of CLK regardless of the state of the $\overline{\text{CLKEN}}$) only on those rising edges of CLK which are spanned by the assertion of $\overline{\text{CLKEN}}$.

$\overline{\text{CLKEN}}$ may be used to allow the external bus to run at 1/2 or 1/4 the speed of the MC68060 processor clock which controls all internal operations. The MC68060 bus interface controller will not detect those rising edges of CLK which are spanned with the negation of $\overline{\text{CLKEN}}$. To operate the external bus at 1/2 or 1/4 the speed of CLK, $\overline{\text{CLKEN}}$ must be asserted and stable during the rising edges of CLK which coincide with the system clock running at 1/2 or 1/4 the frequency of the MC68060 processor clock. $\overline{\text{CLKEN}}$ must be negated and stable during all other rising CLK edges.

For full speed operation of the MC68060 processor, $\overline{\text{CLKEN}}$ must be continuously asserted.

Refer to **Section 7 Bus Operation** for more information on the MC68060 bus interface and controller. Refer to **Section 12 Electrical and Thermal Characteristics** for the timing specifications of CLK and $\overline{\text{CLKEN}}$.

2.11 TEST SIGNALS

The MC68060 includes dedicated user-accessible test logic that is fully compatible with the *IEEE 1149.1 Standard Test Access Port and Boundary Scan Architecture*. Problems associated with testing high-density circuit boards have led to the development of this standard under the IEEE Test Technology Committee and Joint Test Action Group (JTAG) sponsorship. The MC68060 implementation supports circuit board test strategies based on this standard. However, the JTAG interface is not intended to provide an in-circuit test to verify MC68060 operations; therefore, it is impossible to test MC68060 operations using this interface. **Section 9 IEEE 1149.1 Test (JTAG) and Debug Pipe Control Modes** describes the MC68060 implementation of IEEE 1149.1 and is intended to be used with the supporting IEEE document.

2.11.1 JTAG Enable ($\overline{\text{JTAG}}$)

This input signal is used to select between 1149.1 operation and debug emulation mode. The 1149.1 test access port (TAP) pins are remapped to emulation mode functions when this pin is negated. For normal 1149.1 operation, $\overline{\text{JTAG}}$ should be grounded.

2.11.2 Test Clock (TCK)

This input signal is used as a dedicated clock for the test logic. Since clocking of the test logic is independent of the normal operation of the MC68060, several other components on a board can share a common test clock with the processor even though each component may operate from a different system clock. The design of the test logic allows the test clock to run at low frequencies, or to be gated off entirely as required for test purposes. TCK should be grounded if it is not used and emulation mode is not to be used.

2.11.3 Test Mode Select (TMS)

This input signal is decoded by the TAP controller and distinguishes the principal operations of the test support circuitry. TMS should be tied to V_{CC} if it is not used and emulation mode is not to be used.

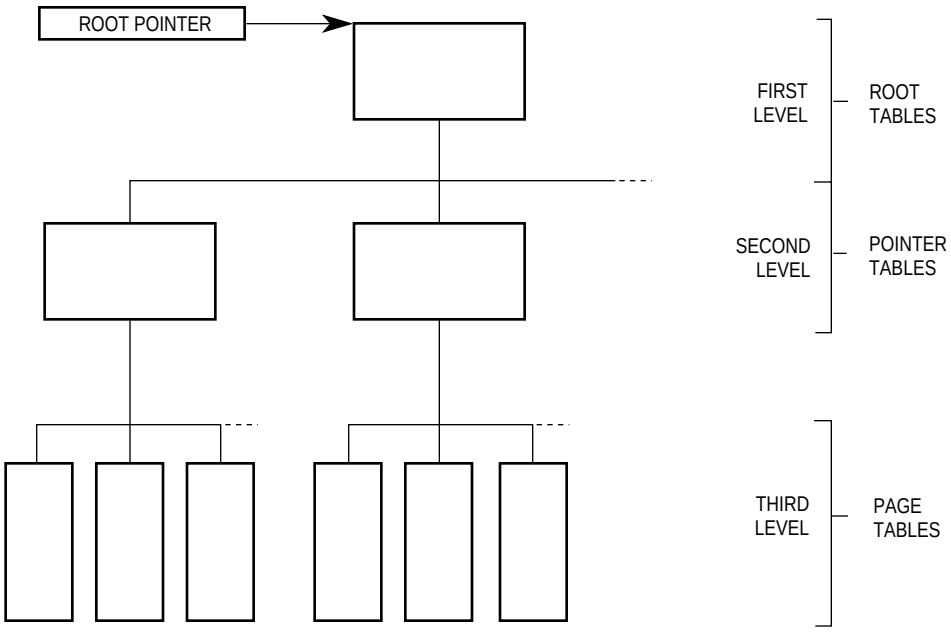


Figure 4-6. Translation Table Structure

descriptor, can be used when two or more logical addresses access a single page descriptor.

The table search uses logical addresses to access the translation tables. Figure 4-7 illustrates a logical address format, which is segmented into four fields: root index (RI), pointer index (PI), page index (PGI), and page offset. The first three fields extracted from the logical address index the base address for each table level. The seven bits of the logical address RI field are multiplied by 4 or shifted to the left by two bits. This sum is concatenated with the upper 23 bits of the appropriate root pointer (URP or SRP) to yield the physical address of a root-level table descriptor. Each of the 128 root-level table descriptors corresponds to a 32-Mbyte block of memory and points to the base of a pointer-level table.

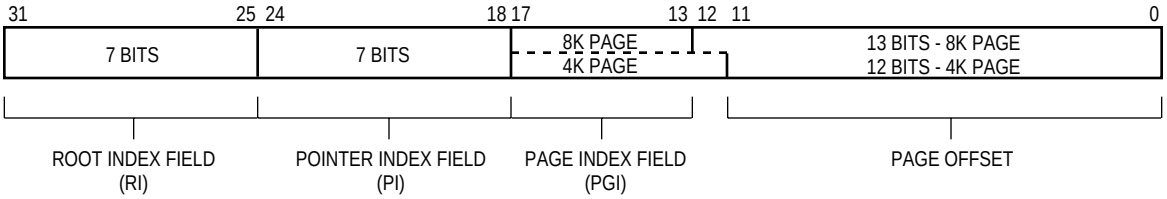


Figure 4-7. Logical Address Format

The seven bits of a logical address PI field are multiplied by 4 (shifted to the left by two bits) and concatenated with the fetched root-level descriptor's upper 23 bits to produce the physical address of the pointer-level table descriptor. Each of the 128 pointer-level table descriptors corresponds to a 256-Kbyte block of memory.

5.4.1.1 WRITETHROUGH MODE. Accesses to pages specified as writethrough are always written to the external address, although the cycle can be buffered (depending on the state of the ESB bit in the CACR). Writes in writethrough mode are handled with a no-write-allocate policy—i.e., writes that miss in the data cache are written to memory or the write buffer, but do not cause the corresponding line in memory to be loaded into the cache. Write accesses that hit always write through to memory and update matching cache lines. Specifying writethrough mode for the shared pages maintains cache coherency for shared memory areas in a multiprocessing environment. The cache supplies data to instruction or data read accesses that hit in the appropriate cache; misses cause a new cache line to be loaded into the cache, unless no-allocate mode is selected (NAD or NAI is set) via the CACR.

5.4.1.2 COPYBACK MODE. Copyback pages are typically used for local data structures or stacks to minimize external bus usage and reduce write access latency. Write accesses to pages specified as copyback that hit in the data cache update the cache line and set the corresponding D-bit without an external bus access. The dirty cached data is only written to memory if the line is replaced due to a miss, or a writethrough or cache-inhibited access which hits the dirty line, or a CPUSH which pushes the line. If a write access misses in the cache, then the needed cache line is read from memory and the cache is updated if the NAD bit in the CACR is clear. If a write miss occurs when the NAD bit is set, the cache is not updated. When a miss causes a dirty cache line to be selected for replacement, the current cache line data is moved to the push buffer. The replacement line is read into the cache, and the push buffer contents are written to external memory.

5.4.2 Cache-Inhibited Accesses

Address space regions containing targets such as I/O devices and shared data structures in multiprocessing systems can be designated cache inhibited. If a page descriptor's CM field indicates precise or imprecise, then the access is cache inhibited. The caching operation is identical for both cache-inhibited modes. The difference between these inhibited cache modes has to do with recovery from an exception (either external bus error, or interrupt).

If the CM field of a matching address indicates either precise or imprecise modes, the cache controller bypasses the cache and performs an external bus transfer. The data associated with the access is not cached internally, and the cache inhibited out ($\overline{\text{CIOUT}}$) signal is asserted during the bus cycle to indicate to external memory that the access should not be cached. If the data cache line is already resident in an internal cache and the current cache mode for that page becomes cache inhibited, either through an operating system change, or due to a shared physical page, then the caches provide additional support for cache coherency, by pushing the line if dirty or invalidating the line if it is valid.

If the CM field indicates precise mode, then the sequence of read and write accesses to the page is guaranteed to match the sequence of the instruction order. In imprecise mode, the operand pipeline allows read accesses that hit in the cache to occur before completion of a pending write from a previous instruction. Writes will not be deferred past operand read accesses that miss in the cache (i.e. that must be read from the bus). Precise operation forces operand read accesses for an instruction to occur only once by preventing the instruction from being interrupted after the operand fetch stage. Otherwise, if not in precise mode

5.9 STORE BUFFER

The MC68060 processor provides a four-entry store buffer (16 bytes maximum). This store buffer is a FIFO buffer that can be used for deferring pending writes to imprecise pages to maximize performance.

For operand writes destined for the store buffer, the operand execution pipeline incurs no stalls. The store buffer effectively provides a measure of decoupling between the pipeline's ability to generate writes (one write per cycle maximum) and the ability of the system bus to retire those writes (one write per two cycles minimum). When writing to imprecise pages, only in the event the store buffer becomes full and there is a write operation in the EX cycle of the operand execution pipeline will a stall be incurred.

If the store buffer is not utilized (store buffer disabled or cache inhibited, precise mode), system bus cycles are generated directly for each pipeline write operation. The instruction is held in the EX cycle of the operand execution pipeline (OEP) until bus transfer termination is received. This means each write operation is stalled for a minimum of five cycles in the EX cycle when the store buffer is not utilized.

A store buffer enable bit is contained in the CACR. This bit can be set and cleared via the MOVEC instruction. Upon reset, this bit is cleared and all writes are precise. When the bit is set, the cache mode generated by the MMU is used. The store buffer is utilized by the cachable/writethrough and the cache-inhibited/imprecise modes.

The store buffer can queue data up to four bytes in width per entry. Each entry matches a corresponding bus cycle it will generate; therefore, a misaligned long-word write to a writethrough page will create two entries if the address is to an odd word boundary, three entries if to an odd byte boundary—one per bus cycle.

A misaligned write access which straddles a precise/imprecise page boundary will use the store buffer for the imprecise portion of the write.

5.10 PUSH BUFFER AND STORE BUFFER BUS OPERATION

Once either the store buffer or the push buffer has valid data, the MC68060 bus controller uses the next available bus cycle to generate the appropriate write cycles. In the event that during the continued instruction execution by the processor pipeline another system bus cycle is required (e.g., data cache miss to process, address translation cache (ATC) tablesearch to perform), the pipeline will stall until both push and store buffers are empty before generating the required system bus transaction.

Certain instructions and exception processing which synchronize the MC68060 processor pipeline guarantee both push and store buffers are empty before proceeding.

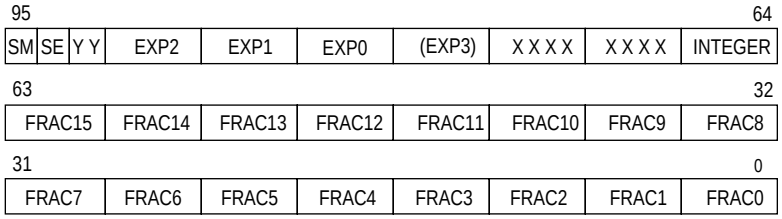
5.11 BRANCH CACHE

The branch cache plays a major role in achieving the performance levels of the MC68060 processor. The branch cache provides a table associating branch program counter values with the corresponding branch target virtual addresses. The fundamental concept is to pro-

Table 6-6. Extended-Precision Real Format Summary (Continued)

Approximate Ranges	
Maximum Positive Normalized	1.2×10^{4932}
Minimum Positive Normalized	1.7×10^{-4932}
Minimum Positive Denormalized	1.7×10^{-4951}

Table 6-7. Packed Decimal Real Format Summary



Data Type	SM	SE	Y	Y	3-Digit Exponent	1-Digit Integer	16-Digit Fraction
±Infinity	0/1	1	1	1	$FFF	$XXXX	$00…00
±NAN	0/1	1	1	1	$FFF	$XXXX	Nonzero
±SNAN	0/1	1	1	1	$FFF	$XXXX	Nonzero
+Zero	0	0/1	X	X	$000–$999	$XXX0	$00…00
–Zero	1	0/1	X	X	$000–$999	$XXX0	$00…00
+In-Range	0	0/1	X	X	$000–$999	$XXX0–$XXX9	$00…01–$99…99
–In-Range	1	0/1	X	X	$000–$999	$XXX0–$XXX9	$00…01–$99…99

NOTE: EXP3 is generated only during an FMOVE OUT if the source is too large to be represented with a three-digit exponent. Otherwise, it is a don't care.

6.3 COMPUTATIONAL ACCURACY

Whenever an attempt is made to represent a real number in a binary format of finite precision, there is a possibility that the number can not be represented exactly. This is commonly referred to as a round-off error. Furthermore, when two inexact numbers are used in a calculation, the error present in each number is reflected, and possibly aggravated, in the result. All FPU calculations use an intermediate result. When the MC68060 performs an operation, the calculation is carried out using extended-precision inputs, and the intermediate result is calculated as if to produce infinite precision. After the calculation is complete, the intermediate result is rounded to the selected precision and stored in the destination.

The FPCR RND and PREC encodings (see Table 6-1 and Table 6-2) provide emulation for devices that only support single and double precision. By setting the rounding precision to single, the MC68060 will perform all calculations as if only 24 bits of precision were available for the result. Setting the rounding precision to double does the same to 53 bits of precision. The execution speed of all instructions is the same whether using single- or double-precision rounding. When using these two forced rounding precisions, the MC68060 produces the same results as any other device that conforms to the IEEE 754 standard, but does not support extended precision. The results are the same when performing the same operation in extended precision and storing the results in single- or double-precision format.

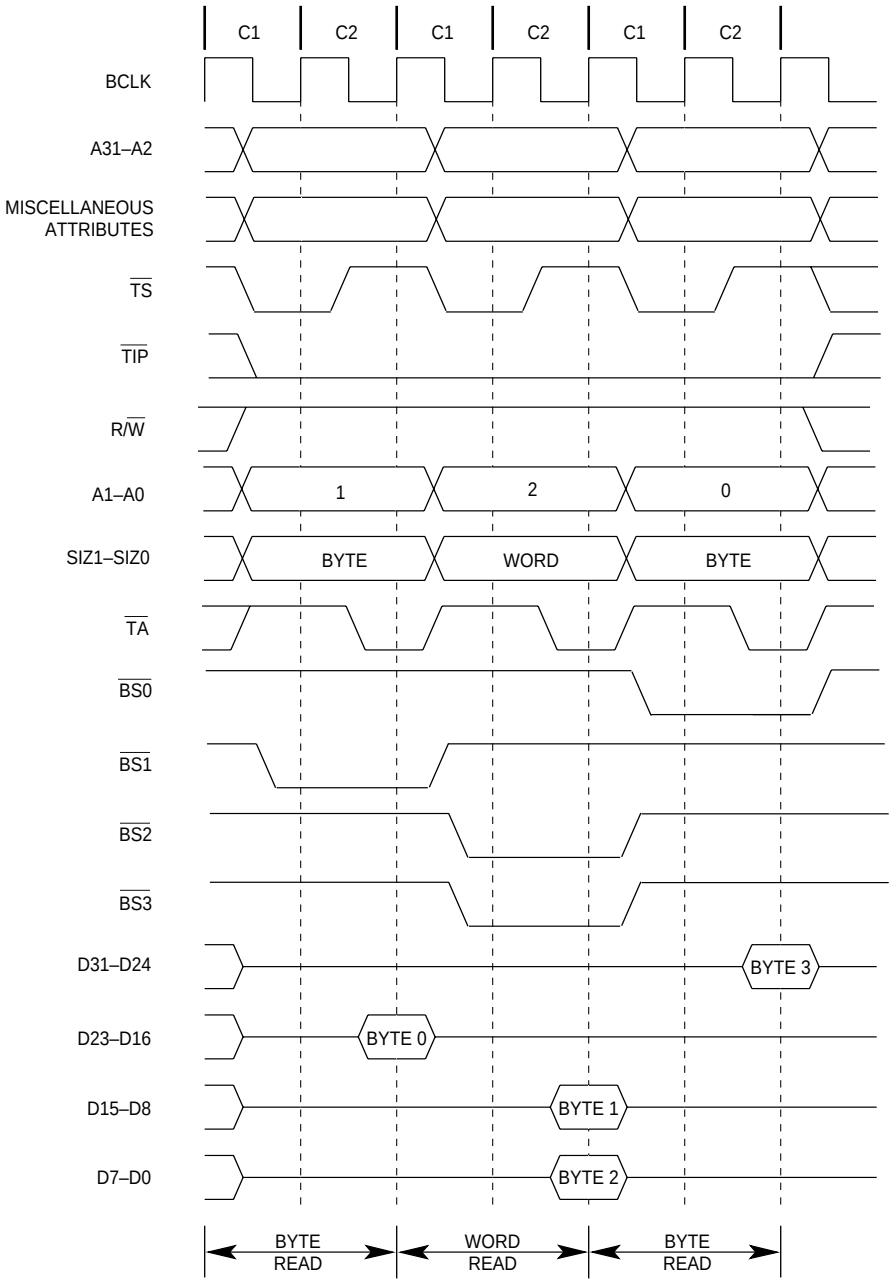


Figure 7-11. Misaligned Long-Word Read Bus Cycle Timing

The combination of operand size and alignment determines the number of bus cycles required to perform a particular memory access. Table 7-3 lists the number of bus cycles required for different operand sizes with all possible alignment conditions for read and write cycles. The table confirms that alignment significantly affects bus cycle throughput for non-cachable accesses. For example, in Figure 7-9 the misaligned long-word operand took three bus cycles because the byte offset = $1. If the byte offset = $0, then it would have taken one

asserted. The registered data and the value of $\overline{TCI}$ are then passed to the appropriate memory unit.

If $\overline{TBI}$ was negated with the assertion of $\overline{TA}$, the processor continues the cycle with C3. Otherwise, if $\overline{TBI}$ was asserted, the line transfer is burst inhibited, and the processor reads the remaining three long words using long-word read bus cycles. The processor increments A3 and A2 for each read, and the new address is placed on the address bus for each bus cycle. Refer to **7.7.1 Byte, Word, and Long-Word Read Transfer Cycles** for information on long-word reads. If no wait states are generated, a burst-inhibited line read completes in eight clocks instead of the five required for a burst read.

Clock 3 (C3)

The processor holds the address and transfer attribute signals constant during C3 if $\overline{CLA}$ is negated. The selected device must either increment A3 and A2 to reference the next long word to transfer, place the data on the data bus, and assert $\overline{TA}$, or alteratively assert the $\overline{CLA}$ input to request the processor to increment A3 and A2. Refer to **7.7.7 Using CLA to Increment A3 and A2** for details on $\overline{CLA}$ operation.

As in the description of C2, using acknowledge termination ignore state capability, the processor ignores any termination signal, such as $\overline{TA}$, until a user-programmable number of BCLK edges has expired. And, as in the description in C2, $\overline{SAS}$ indicates the first BCLK rising edge in which acknowledge termination signals become significant. If this mode is disabled, $\overline{SAS}$ stays asserted in C3 to indicate that the processor will sample $\overline{TA}$ immediately. Refer to **7.14.1 Acknowledge Termination Ignore State Capability** for details on this mode.

Assuming that the acknowledge termination ignore state capability is disabled, the processor samples the level of $\overline{TA}$ and registers the current value on the data bus at the end of C3. If $\overline{TA}$ is asserted, the transfer terminates and the second long word of data is passed to the appropriate memory unit. If $\overline{TA}$ is not recognized asserted at the end of C3, the processor ignores the latched data and inserts wait states instead of terminating the transfer. The processor continues to sample $\overline{TA}$ on successive rising edges of BCLK until it is recognized asserted. The registered data is then passed to the appropriate memory unit.

Clock 4 (C4)

This clock is identical to C3 except that once $\overline{TA}$ is recognized asserted, the registered value corresponds to the third long word of data for the burst.

Clock 5 (C5)

This clock is identical to C3 except that once $\overline{TA}$ is recognized, the registered value corresponds to the fourth long word of data for the burst. After the processor recognizes the last $\overline{TA}$ assertion and terminates the line read bus cycle, $\overline{TIP}$ remains asserted if the processor is ready to begin another bus cycle. Otherwise, the processor negates $\overline{TIP}$ during the next clock.

Figure 7-16 and Figure 7-17 illustrate a flowchart and functional timing diagram for a burst-inhibited line read bus cycle.

Table 7-9. MC68060-Arbitration Protocol State Description

BTTO	Bus Status	Own	State
Not Driven	Not Driven	No	Reset
Not Driven	Not Driven	No	Alternated Master Implicit Own
Not Driven	Not Driven	No	Alternate Master Explicit Own
Not Driven	Not Driven	Yes	Implicit Ownership
Not Driven	Driven	Yes	Explicit Ownership
Asserted for One BCLK, Negated for One BCLK then Three-Stated	Stops Being Driven at End of State	Yes	End Tenure
Not Driven	Not Driven	No	Alternate Master Own and Snooped

NOTE: BTTO represents the component of BTT as driven by the MC68060. BTT is normally three-stated but driven for one BCLK when asserted and one BCLK when negated.

The MC68060 can be in any one of seven bus arbitration states during bus operation: reset, AM-implicit own, AM-explicit own, snoop, implicit ownership, explicit ownership, and the end tenure state.

The reset state is entered whenever RSTI is asserted in any bus arbitration state, except the explicit ownership state. For that state, the end tenure state is entered prior to entering the reset state.This is done to ensure other bus masters are capable of taking the bus away from the processor when it is reset. When RSTI is negated, the processor proceeds to the implicit ownership state or alternate master implicit ownership state, depending on BG. If an alternate master asserts TS or has asserted TS in the past, the processor waits for BTT to assert (or alternatively for BB to go from being asserted to being negated) before taking the bus, even though BG may be asserted to the processor.

The AM-implicit own state denotes the MC68060 does not have ownership (BG negated) of the bus and is not in the process of snooping an access, and the alternate has not begun its tenure by asserting TS (alternate master TS or SNOOP negated). In the AM-implicit own state, the MC68060 does not drive the bus. The processor enters the AM-explicit own state when TS is asserted by the alternate master. Once in the AM-explicit own state, the processor waits for the alternate master to assert BTT before recognizing that a change of tenure has occurred. If BG is negated when BTT is asserted, the processor assumes that another master has taken implicit ownership of the bus. Otherwise, if BG is asserted when BTT is asserted, the processor assumes implicit ownership of the bus.

If an alternate master loses bus ownership when it is in implicit ownership state, the processor checks TS. If TS is sampled asserted, the processor interprets this as the alternate master transitioning to its explicit ownership state, and it does not take bus ownership. This operation is different from that of the MC68040 in that external arbiters are required to check for this boundary condition. However, in order for the processor to properly detect this boundary condition, it is imperative that the TS of all alternate bus masters be tied together with the processor's TS signal.

since an RTE or FRESTORE instruction can check for residency and trap before restoring the state.

A special case exists for systems that allow arbitration of the processor bus during locked transfer sequences. If the arbiter can signal a bus error of a locked translation table update due to an improperly broken lock, any pages touched by exception stack operations must have the U-bit set in the corresponding page descriptor to prevent the occurrence of the locked access during translation table searches.

8.2.2 Address Error Exception

An address error exception occurs when the processor attempts to prefetch an instruction from an odd address. An odd address is defined as an address in which the least significant bit is set. Some of the ways an address error exception is taken is as follows: RTS, RTD, RTR, or RTE in which the PC value in the stack is odd; a branch (conditional or unconditional), jump, or subroutine call in which the branch target address is odd; and an odd vector table entry (e.g., an odd reset vector).

A stack frame of type 2 is generated when this exception is reported.The stacked PC contains the address of the instruction that caused the address error. The address field in the stack contains the branch target address with A0 cleared.

If an address error occurs during the exception processing for a bus error, address error, or reset, a double bus fault occurs. The processor enters the halted state as indicated by the PST4–PST0 encoding $1C. In this case, the processor does not attempt to alter the current state of memory. Only an external reset can restart a processor halted by a double bus fault.

8.2.3 Instruction Trap Exception

Certain instructions are used to explicitly cause trap exceptions. The TRAP #n instruction always forces an exception and is useful for implementing system calls in user programs. The TRAPcc, TRAPV, and CHK instructions force exceptions if the user program detects an error, which can be an arithmetic overflow or a subscript value that is out of bounds. The DIVS and DIVU instructions force exceptions if a division operation is attempted with a divisor of zero.

As illustrated in Figure 8-1, when a trap exception occurs, the processor internally copies the SR, enters the supervisor mode, and clears the T-bit. The processor generates a vector number according to the instruction being executed. Vector 5 is for DIVx, vector 6 is for CHK, and vector 7 is for TRAPcc and TRAPV instructions. For the TRAP #n instruction, the vector number is 32 plus n. The stack frame saves the trap vector offset, the PC, and the internal copy of the SR on the supervisor stack.

A stack frame of type 0 is generated when a TRAP #n exception is taken. The saved value of the PC is the logical address of the instruction following the instruction that caused the trap. Instruction execution resumes at the address in the exception vector after the required instruction is prefetched.

For all instruction traps other than TRAP #n, a stack frame of type 2 is generated. The stacked PC contains the logical address of the next instruction to be executed. In addition to the stacked PC, a pointer to the instruction that caused the trap is saved in the address field of the stack frame. Instruction execution resumes at the address in the exception vector after the required instruction is prefetched.

8.2.4 Illegal Instruction and Unimplemented Instruction Exceptions

There are eight unimplemented instruction exceptions: unimplemented integer, unimplemented effective address, unimplemented A-line, unimplemented F-line, floating-point disabled, floating-point unimplemented instruction, floating-point unsupported data type, and illegal instruction.

The unimplemented integer exception corresponds to vector number 61 and occurs when the processor attempts to execute an instruction that contains a quad word operand (MULx producing a 64-bit product and DIVx using a 64-bit dividend), CAS2, CHK2, CMP2, CAS with a misaligned operand, and the MOVEP instruction. A stack frame of type 0 is generated when this exception is reported. The stacked PC points to the logical address of the unimplemented integer instruction that caused the exception.

The unimplemented effective address exception corresponds to vector number 60, and occurs when the processor attempts to execute any floating-point instruction that contains an extended precision immediate source operand (F<op>, #imm,FPx), when the processor attempts to execute an FMOVEM.L #imm,<control register list> instruction of more than one floating-point control register (FPCR, FPSR, FPIAR), when the processor attempts an FMOVEM.X instruction using a dynamic register list (FMOVEM.X Dn,<ea> or FMOVEM.X, <ea>,Dn). The stack frame of type 0 is generated when this exception is reported. The stacked PC points to the logical address of the instruction that caused the exception. The FPIAR is unaffected. Refer to **Section 6 Floating-Point Unit** for details.

An unimplemented A-line exception corresponds to vector number 10 and occurs when an instruction word pattern begins (bits 15–12) with $A. The A-line opcodes are user-reserved, and Motorola will not use any A-line instructions to extend the instruction set of any of Motorola’s processors. A stack frame of format 0 is generated when this exception is reported. The stacked PC points to the logical address of the A-line instruction word.

A floating-point unsupported data type exception occurs when the processor attempts to execute a bit pattern that it recognizes as an MC68881 instruction, the floating-point unit (FPU) is enabled via the processor configuration register (PCR), the floating-point instruction is implemented, but the floating-point data type is not implemented in the MC68060 FPU. This exception corresponds to vector number 55. A stack frame of type 0, 2, or 3 is generated when this exception is reported. The stacked PC points to the logical address of next instruction after the floating-point instruction. Refer to **Section 6 Floating-Point Unit** for details.

A floating-point unimplemented instruction exception occurs when the processor attempts to execute an instruction word pattern that begins with $F, the processor recognizes this bit pattern as an MC68881 instruction, the FPU is enabled via the PCR, but the floating-point instruction is not implemented in the MC68060 FPU. This exception corresponds to vector

When the MC68060 executes one of the breakpoint instructions, it performs a breakpoint acknowledge cycle (read cycle) with an acknowledge transfer type (TT=$3) and transfer modifier value of $0. Refer to **Section 7 Bus Operation** for a description of the breakpoint acknowledge cycle. After external hardware terminates the bus cycle with either $\overline{TA}$ or $\overline{TEA}$, the processor performs illegal instruction exception processing. Refer to **8.2.4 Illegal Instruction and Unimplemented Instruction Exceptions** for details on illegal instruction exception processing.

8.2.9 Interrupt Exception

When a peripheral device requires the services of the MC68060 or is ready to send information that the processor requires, it can signal the processor to take an interrupt exception using the $\overline{IPLx}$ signals. The three signals encode a value of 0–7 ($\overline{IPL0}$ is the least significant bit). High levels on all three signals correspond to no interrupt requested (level 0). Values 1–7 specify one of seven levels of interrupts, with level 7 having the highest priority. Table 8-2 lists the interrupt levels, the states of $\overline{IPLx}$ that define each level, and the SR interrupt mask value that allows an interrupt at each level.

Table 8-2. Interrupt Levels and Mask Values

Requested Interrupt Level	Control Line Status			Interrupt Mask Level Required for Recognition
	$\overline{IPL2}$	$\overline{IPL1}$	$\overline{IPL0}$	
0	Negated	Negated	Negated	No Interrupt Requested
1	Negated	Negated	Asserted	0
2	Negated	Asserted	Negated	0–1
3	Negated	Asserted	Asserted	0–2
4	Asserted	Negated	Negated	0–3
5	Asserted	Negated	Asserted	0–4
6	Asserted	Asserted	Negated	0–5
7	Asserted	Asserted	Asserted	0–7

When an interrupt request has a priority higher than the value in the interrupt priority mask of the SR (bits 10–8), the processor makes the request a pending interrupt. Priority level 7, the nonmaskable interrupt, is a special case. Level 7 interrupts cannot be masked by the interrupt priority mask, and they are transition sensitive. The processor recognizes an interrupt request each time the external interrupt request level changes from some lower level to level 7, regardless of the value in the mask. Figure 8-3 shows two examples of interrupt recognitions, one for level 6 and one for level 7. When the MC68060 processes a level 6 interrupt, the SR mask is automatically updated with a value of 6 before entering the handler routine so that subsequent level 6 interrupts and lower level interrupts are masked. Provided no instruction that lowers the mask value is executed, the external request can be lowered to level 3 and then raised back to level 6 and a second level 6 interrupt is not processed. However, if the MC68060 is handling a level 7 interrupt (SR mask set to level 7) and the external request is lowered to level 3 and than raised back to level 7, a second level 7 interrupt is processed. The second level 7 interrupt is processed because the level 7 interrupt is transition sensitive. A level comparison also generates a level 7 interrupt if the request level and mask level are at 7 and the priority mask is then set to a lower level (as

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.

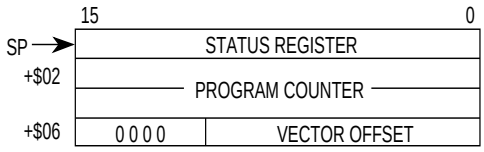
Given a valid stack format and a nonfaulting PC, the SR and PC are loaded with the stack operands, the SSP adjusted by the appropriate value determined by the format field, and control passed to the location defined by the new PC.

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

toggling (such as during in-circuit testing) by placing all system signal pins to a high impedance state.

NOTE

The IEEE standard 1149.1 test logic cannot be considered completely benign to those planning not to use this capability. Certain precautions must be observed to ensure that this logic does not interfere with system operation and allows full use of the LPSTOP function. Refer to **9.1.5 Disabling the IEEE 1149.1 Standard Operation**

9.1.1 Overview

Figure 9-1 illustrates the block diagram of the MC68060 implementation of the 1149.1 standard.The test logic includes several test data registers, an instruction register, instruction register control decode, and a 16-state dedicated TAP controller. The sixteen controller states are defined in detail in the in the IEEE 1149.1 standard, but eight are listed in Table 9-1 and included for illustration purposes:

Table 9-1. JTAG States

State Name	State Summary
Test-Logic-Reset	Places test logic in default defined reset state
Run-Test-Idle	Allows test control logic to remain idle while test operations occur
Capture-IR	Loads default IDCODE instruction into the instruction register
Shift-IR	Allows serial data to move from TDI to TDO through the instruction register
Update-IR	Applies and activates instruction contained in the instruction shift register
Capture-DR	Loads parallel sampled data into the selected test data register
Shift-DR	Allows serial data to move from TDI to TDO through the selected test data register
Update-DR	Applies test data contained in the selected test data register

The TAP consists of five dedicated signal pins which are controlled by a sixth dedicated compliance enable pin.

1. $\overline{\text{JTAG}}$—An active low JTAG enable pin that maps the TAP signals to either the 1149.1 logic or the emulation mode logic and meets the requirements set forth for a compliance enable pin. The TAP pins are described in the case of $\overline{\text{JTAG}}$ asserted.
2. TCK—A test clock input that synchronizes test logic operations.
3. TMS—A test mode select input with an internal pullup resistor that is sampled on the rising edge of TCK to sequence the TAP controller.
4. TDI—A serial test data input with an internal pullup resistor that is sampled on the rising edge of TCK.
5. TDO—A three-state test data output that is actively driven only in the shift-IR and shift-DR controller states and only updates on the falling edge of TCK.
6. $\overline{\text{TRST}}$—An active low asynchronous reset with an internal pullup resistor that forces the TAP controller into the test-logic-reset state.

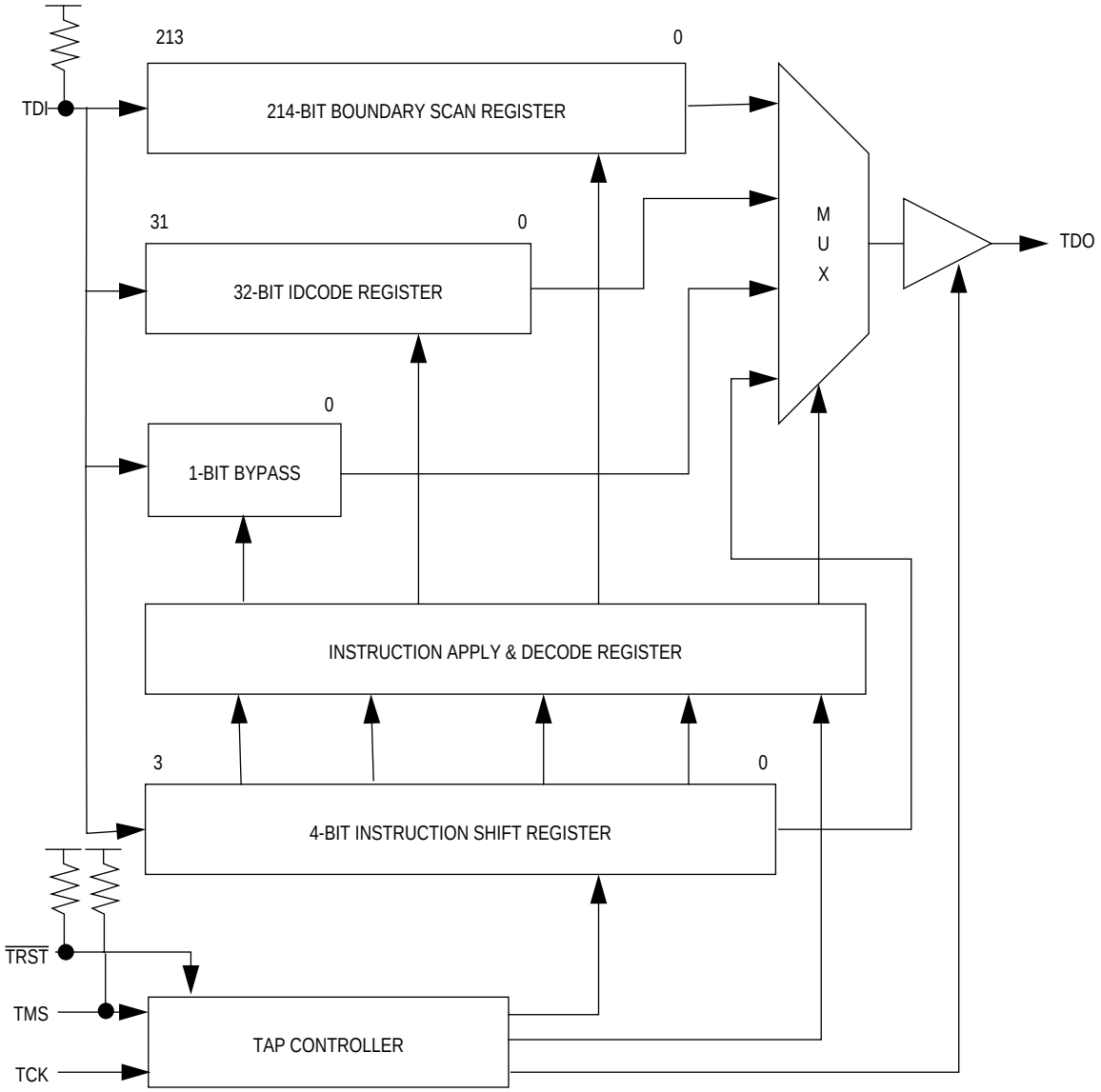


Figure 9-1. JTAG Test Logic Block Diagram

9.1.2 JTAG Instruction Shift Register

The MC68060 IEEE 1149.1 implementation uses a 4-bit instruction shift register without parity. The shift register transfers its value to a parallel hold register and applies one of sixteen possible instructions, seven of which are defined as public customer-usable instructions, on the falling edge of TCK when the TAP state machine is in the update-IR state (the other nine instructions are private instructions to support manufacturing test and can cause destructive behavior if used without proper understanding). The instructions may be loaded into the shift portion of the register by placing the serial data on the TDI signal prior to each rising edge of TCK. The most significant bit of the instruction shift register is the bit closest to the TDI signal and the least significant bit is the bit closest to the TDO pin.

The public customer-usable instructions that are supported are listed with their encodings in Table 9-2.

9.2.2 Debug Pipe Control Mode Commands

The following capabilities are provided by the debug pipe control mode:

- Halt and restart processor execution
- Forcing the processor into an emulator mode
- From a halted processor state, the following additional capabilities are provided:
  —Setting and resetting a non-pipelined execution mode in the processor
  —Override disable processor configuration features (instruction cache, data cache, address translation caches (ATCs), write buffer, branch cache, floating-point unit (FPU), superscalar dispatch)
  —Forcing insertion of cache and ATC control operations into the processor pipeline for execution (CINV all for instruction cache and data cache, CPUSH all for instruction cache and data cache, and PFLUSH all for ATCs)
  —Forcing all processor outputs into and out of a high-impedance state and disable all inputs
  —Setting and resetting modes that convert trace exceptions and breakpoint instructions into emulator mode entry

Table 9-5 provides a brief summary of the command functions that are made available through the debug pipe control mode. Most of the commands can only be issued only when the processor is halted.

10.11 BRANCH INSTRUCTION EXECUTION TIMES

Table 10-17, Table 10-18, and Table 10-19 indicate the number of clock cycles required for execution of the branch, jump, and return 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 10-17. Branch Execution Times

Instruction	Not Predicted, Forward, Taken	Not Predicted, Forward, Not Taken	Not Predicted, Backward, Taken	Not Predicted, Backward, Not Taken	Predicted Correctly as Taken	Predicted Correctly as Not Taken	Predicted Incorrectly
Bcc	7(0/0)	1(0/0)	3(0/0)	7(0/0)	0(0/0)	1(0/0)	7(0/0)
BRA	3(0/0)	—	3(0/0)	—	0(0/0)	—	—
BSR	3(0/1)	—	3(0/1)	—	1(0/1)	—	—
DBcc	3(0/0)	8(0/0)	3(0/0)	8(0/0)	2(0/0)	2(0/0)	8(0/0)
DBRA	3(0/0)	7(0/0)	3(0/0)	7(0/0)	1(0/0)	1(0/0)	7(0/0)
FBcc	8(0/0)	2(0/0)	8(0/0)	2(0/0)	2(0/0)	2(0/0)	8(0/0)

Table 10-18. JMP, JSR Execution Times[1]

Instruction	Not Predicted, Forward, Taken	Not Predicted, Forward, Not Taken	Not Predicted, Backward, Taken	Not Predicted, Backward, Not Taken	Predicted Correctly as Taken	Predicted Correctly as Not Taken	Predicted Incorrectly
JMP (d16,PC)	3(0/0)	—	3(0/0)	——	0(0/0)	—	—
JMP xxx.WL	3(0/0)	—	3(0/0)	—	0(0/0)	—	—
Remaining JMP	5(0/0)	—	5(0/0)	—	5(0/0)	—	—
JSR (d16,PC)	3(0/1)	—	3(0/1)	—	1(0/1)	—	—
JSR xxx.WL	3(0/1)	—	3(0/1)	—	1(0/1)	—	—
Remaining JSR	5(0/1)	—	5(0/1)	—	5(0/1)	—	—

[1] Add the effective address calculation time for each entry.

Table 10-19. Return Instruction Execution Times

Instruction	Execution Time
RTD	7(1/0)
RTE	17(3/0)
RTR	8(2/0)
RTS	7(1/0)

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

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

Figure C-6. MUL Instruction Call Example

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

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

Figure C-7. CMP2 Instruction Call Example

divide and the source operand is a zero, then the library routine (as it is last instruction) executes an implemented divide using a zero source operand so that an integer divide-by-zero exception will be taken. Although the exception stack frame will not point to the correct instruction, the user can at least be able to record that such an event occurred.

C.3 FLOATING-POINT EMULATION PACKAGE (MC68060FPSP)

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

MOVEC — Move Control Register — MOVEC

(MC68060, MC68LC060 and MC68EC060)

Operation: If Supervisor State
Then Rc ➧ Rn or Rn ➧ Rc
Else TRAP

Assembler Syntax: MOVEC Rc,Rn
MOVEC Rn,Rc

Attributes: Size = (Long)

Description: Moves the contents of the specified control register (Rc) to the specified general register (Rn) or copies the contents of the specified general register to the specified control register. This is always a 32-bit transfer, even though the control register may be implemented with fewer bits. Unimplemented bits are read as zeros.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	1	0	1	dr
A/D	REGISTER			CONTROL REGISTER											

Instruction Fields:

dr field—Specifies the direction of the transfer.
0—Control register to general register.
1—General register to control register.

A/D field—Specifies the type of general register.
0—Data Register
1—Address Register

PLPA — Load Physical Address (MC68060, MC68LC060) — PLPA

Operation: If Supervisor State
Then Logical Address {DFC,An} translated to Physical Address ➧ An
Else TRAP

Assembler Syntax:
PLPAR (An)
PLPAW (An)

Attributes: Unsized

Description: Translates the logical address defined by the contents of the destination function code register (DFC2–DFC0) and the address register (An31–An0), using full paged MMU functionality including TTRs, and generates a 32-bit physical address, which is loaded into An. All access error checks are performed during the translation, including in the checks the read/write instruction type, and an access error exception will be taken for faulting conditions.

PLPA is a privileged instruction; attempted execution in user mode will result in a privilege violation exception.

As with normal address translation activity:

If Data TTR hit
Then Use TTR translation and An stays the same
Else if E bit of TC Register = 0 or $\overline{\text{MDIS}}$ pin asserted
Then Use Default TTR translation and An stays the same
Else if E bit of TC Register =1 and $\overline{\text{MDIS}}$ pin negated and Data ATC hit
Then use ATC translation and An = Physical Address
Else if E bit of TC Register =1 and $\overline{\text{MDIS}}$ pin negated and Data ATC miss
Then Tablewalk
If Valid Page Descriptor Encountered
Then update Data ATC and An = Physical Address
Else Take Access Error Exception
EndIF

Condition Codes:
Not affected.