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	68020
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	25MHz
Co-Processors/DSP	-
RAM Controllers	-
Graphics Acceleration	No
Display & Interface Controllers	-
Ethernet	-
SATA	-
USB	-
Voltage - I/O	5.0V
Operating Temperature	0°C ~ 70°C (TA)
Security Features	-
Package / Case	100-BQFP
Supplier Device Package	100-QFP (14x20)
Purchase URL	https://www.e-xfl.com/product-detail/nxp-semiconductors/mc68ec020fg25

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

MC68020/EC020 ACRONYM LIST

BCD — Binary-Coded Decimal
CAAR — Cache Address Register
CACR — Cache Control Register
CCR — Condition Code Register
CIR — Coprocessor Interface Register
CMOS — Complementary Metal Oxide Semiconductor
CPU — Central Processing Unit
CQFP — Ceramic Quad Flat Pack
DDMA — Dual-Channel Direct Memory Access
DFC — Destination Function Code Register
DMA — Direct Memory Access
DRAM — Dynamic Random Access Memory
FPCP — Floating-Point Coprocessor
HCMOS — High-Density Complementary Metal Oxide Semiconductor
IEEE — Institute of Electrical and Electronic Engineers
ISP — Interrupt Stack Pointer
LMB — Lower Middle Byte
LRAR — Limited Rate Auto Request
LSB — Least Significant Byte
MMU — Memory Management Unit
MPU — Microprocessor Unit
MSB — Most Significant Byte
MSP — Master Stack Pointer
NMOS — n-Type Metal Oxide Semiconductor
PAL — Programmable Array Logic
PC — Program Counter
PGA — Pin Grid Array
PMMU — Paged Memory Management Unit
PPGA — Plastic Pin Grid Array
PQFP — Plastic Quad Flat Pack
RAM — Random Access Memory
SFC — Source Function Code Register
SP — Stack Pointer
SR — Status Register
SSP — Supervisor Stack Pointer
SSW — Special Status Word
UMB — Upper Middle Byte
USP — User Stack Pointer
VBR — Vector Base Register
VLSI — Very Large Scale Integration

When initiating a bus cycle, the MC68020 asserts $\overline{ECS}$ in addition to A1–A0, SIZ1, SIZ0, FC2–FC0, and R/$\overline{W}$. $\overline{ECS}$ can be used to initiate various timing sequences that are eventually qualified with $\overline{AS}$. Qualification with $\overline{AS}$ may be required since, in the case of an internal cache hit, a bus cycle may be aborted after $\overline{ECS}$ has been asserted. During the first MC68020 external bus cycle of an operand transfer, $\overline{OCS}$ is asserted with $\overline{ECS}$. When several bus cycles are required to transfer the entire operand, $\overline{OCS}$ is asserted only at the beginning of the first external bus cycle. With respect to $\overline{OCS}$, an "operand" is any entity required by the execution unit, whether a program or data item. Note that $\overline{ECS}$ and $\overline{OCS}$ are not implemented in the MC68EC020.

The FC2–FC0 signals select one of eight address spaces (see Table 2-1) to which the address applies. Five address spaces are presently defined. Of the remaining three, one is reserved for user definition, and two are reserved by Motorola for future use. FC2–FC0 are valid while $\overline{AS}$ is asserted.

The SIZ1 and SIZ0 signals indicate the number of bytes remaining to be transferred during an operand cycle (consisting of one or more bus cycles) or during a cache fill operation from a device with a port size that is less than 32 bits. Table 5-2 lists the encoding of SIZ1 and SIZ0. SIZ1 and SIZ0 are valid while $\overline{AS}$ is asserted.

The R/$\overline{W}$ signal determines the direction of the transfer during a bus cycle. When required, this signal changes state at the beginning of a bus cycle and is valid while $\overline{AS}$ is asserted. R/$\overline{W}$ only transitions when a write cycle is preceded by a read cycle or vice versa. This signal may remain low for two consecutive write cycles.

The $\overline{RMC}$ signal is asserted at the beginning of the first bus cycle of a read-modify-write operation and remains asserted until completion of the final bus cycle of the operation. The $\overline{RMC}$ signal is guaranteed to be negated before the end of state 0 for a bus cycle following a read-modify-write operation.

5.1.2 Address Bus

A31–A0 (for the MC68020) or A23–A0 (for the MC68EC020) define the address of the byte (or the most significant byte) to be transferred during a bus cycle. The processor places the address on the bus at the beginning of a bus cycle. The address is valid while $\overline{AS}$ is asserted. In the MC68EC020, A31–A24 are used internally, but not externally.

5.1.3 Address Strobe

$\overline{AS}$ is a timing signal that indicates the validity of an address on the address bus and of many control signals. It is asserted one-half clock after the beginning of a bus cycle.

5.1.4 Data Bus

D31–D0 comprise a bidirectional, nonmultiplexed parallel bus that contains the data being transferred to or from the processor. A read or write operation may transfer 8, 16, 24, or 32 bits of data (one, two, three, or four bytes) in one bus cycle. During a read cycle, the data is latched by the processor on the last falling edge of the clock for that bus cycle. For

5.2.6 Bus Operation

The MC68020/EC020 bus is used in an asynchronous manner allowing external devices to operate at clock frequencies different from the MC68020/EC020 clock. Bus operation uses the handshake lines ($\overline{AS}$, $\overline{DS}$, $\overline{DSACK0}$, $\overline{DSACK1}$, $\overline{BERR}$, and $\overline{HALT}$) to control data transfers. $\overline{AS}$ signals the start of a bus cycle, and $\overline{DS}$ is used as a condition for valid data on a write cycle. Decoding SIZ1, SIZ0, A1, and A0 provides byte enable signals that select the active portion of the data bus. The slave device (memory or peripheral) then responds by placing the requested data on the correct portion of the data bus for a read cycle or latching the data on a write cycle and by asserting the $\overline{DSACK0}/\overline{DSACK1}$ combination that corresponds to the port size to terminate the cycle. If no slave responds or the access is invalid, external control logic asserts $\overline{BERR}$ to abort or $\overline{BERR}$ and $\overline{HALT}$ to retry the bus cycle.

$\overline{DSACK1}/\overline{DSACK0}$ can be asserted before the data from a slave device is valid on a read cycle. The length of time that $\overline{DSACK1}/\overline{DSACK0}$ may precede data is given by parameter #31, and it must be met in any asynchronous system to ensure that valid data is latched into the processor. (Refer to **Section 10 Electrical Characteristics** for timing parameters.) Note that no maximum time is specified from the assertion of $\overline{AS}$ to the assertion of $\overline{DSACK1}/\overline{DSACK0}$. Although the processor can transfer data in a minimum of three clock cycles when the cycle is terminated with $\overline{DSACK1}/\overline{DSACK0}$, the processor inserts wait cycles in clock period increments until $\overline{DSACK1}/\overline{DSACK0}$ is recognized.

The $\overline{BERR}$ and/or $\overline{HALT}$ signals can be asserted after $\overline{DSACK1}/\overline{DSACK0}$ is asserted. $\overline{BERR}$ and/or $\overline{HALT}$ must be asserted within the time given (parameter #48), after $\overline{DSACK1}/\overline{DSACK0}$ is asserted in any asynchronous system. If this maximum delay time is violated, the processor may exhibit erratic behavior.

5.2.7 Synchronous Operation with $\overline{DSACK1}/\overline{DSACK0}$

Although cycles terminated with $\overline{DSACK1}/\overline{DSACK0}$ are classified as asynchronous, cycles terminated with $\overline{DSACK1}/\overline{DSACK0}$ can also operate synchronously in that signals are interpreted relative to clock edges. The devices that use these synchronous cycles must synchronize the responses to the MC68020/EC020 clock. Since these devices terminate bus cycles with $\overline{DSACK1}/\overline{DSACK0}$, the dynamic bus sizing capabilities of the MC68020/EC020 are available. In addition, the minimum cycle time for these synchronous cycles is three clocks.

To support systems that use the system clock to generate $\overline{DSACK1}/\overline{DSACK0}$ and other asynchronous inputs, the asynchronous input setup time (parameter #47A) and the asynchronous input hold time (parameter #47B) are provided in **Section 10 Electrical Characteristics**. (Note: although a misnomer, these “asynchronous” parameters are the setup and hold times for synchronous operation.) If the setup and hold times are met for the assertion or negation of a signal, such as $\overline{DSACK1}/\overline{DSACK0}$, the processor can be guaranteed to recognize that signal level on that specific falling edge of the system clock. If the assertion of $\overline{DSACK1}/\overline{DSACK0}$ is recognized on a particular falling edge of the clock, valid data is latched into the processor (for a read cycle) on the next falling clock edge provided the data meets the data setup time (parameter #27). In this case, parameter #31

State 4

MC68020/EC020—At the end of state 4 (S4), the processor latches the incoming data.

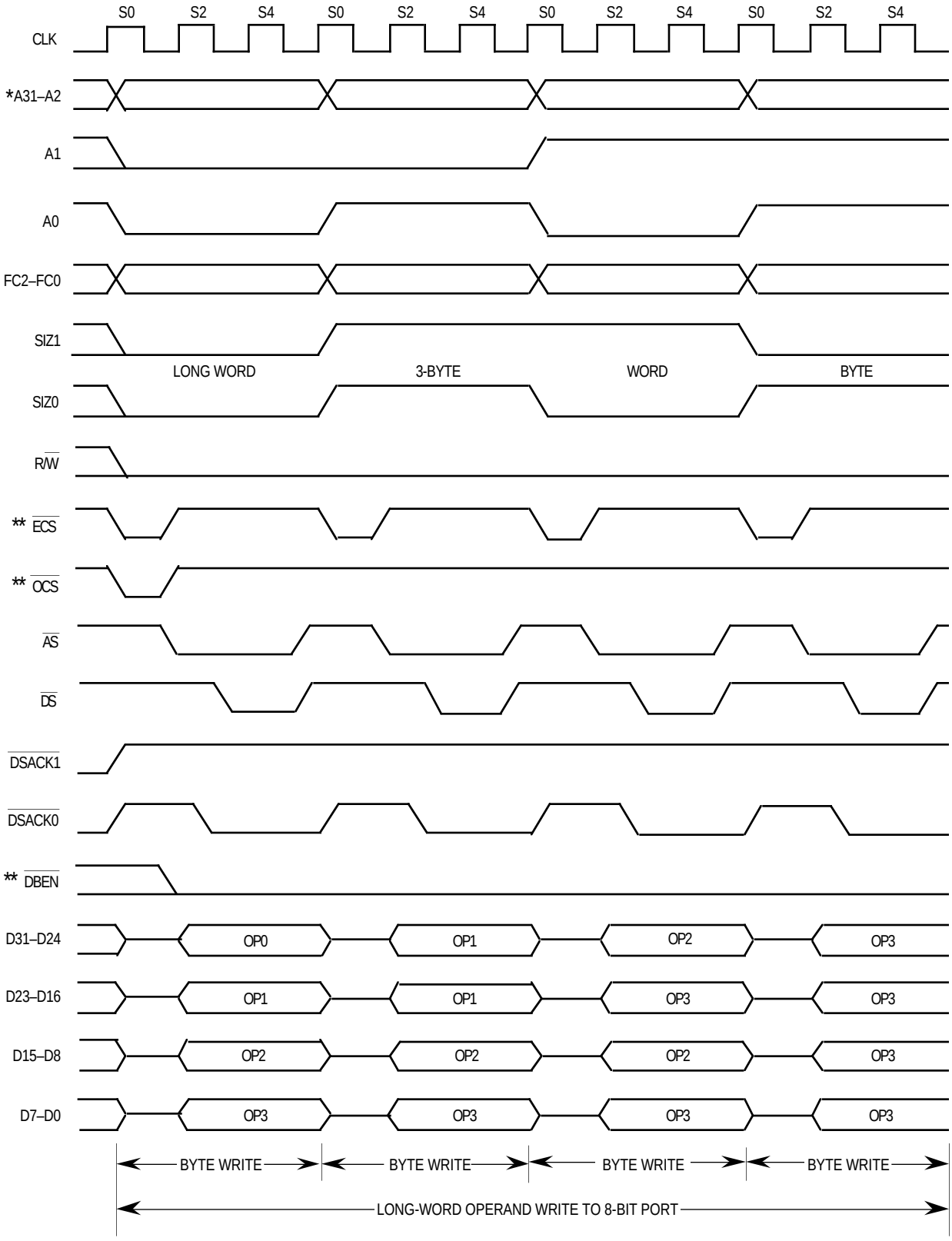
State 5

MC68020—The processor negates $\overline{AS}$, $\overline{DS}$, and $\overline{DBEN}$ during state 5 (S5). It holds the address valid during S5 to provide address hold time for memory systems. R/$\overline{W}$, SIZ1–SIZ0, and FC2–FC0 also remain valid throughout S5.

The external device keeps its data and $\overline{DSACK1}$/$\overline{DSACK0}$ signals asserted until it detects the negation of $\overline{AS}$ or $\overline{DS}$ (whichever it detects first). The device must remove its data and negate $\overline{DSACK1}$/$\overline{DSACK0}$ within approximately one clock period after sensing the negation of $\overline{AS}$ or $\overline{DS}$. $\overline{DSACK1}$/$\overline{DSACK0}$ signals that remain asserted beyond this limit may be prematurely detected for the next bus cycle.

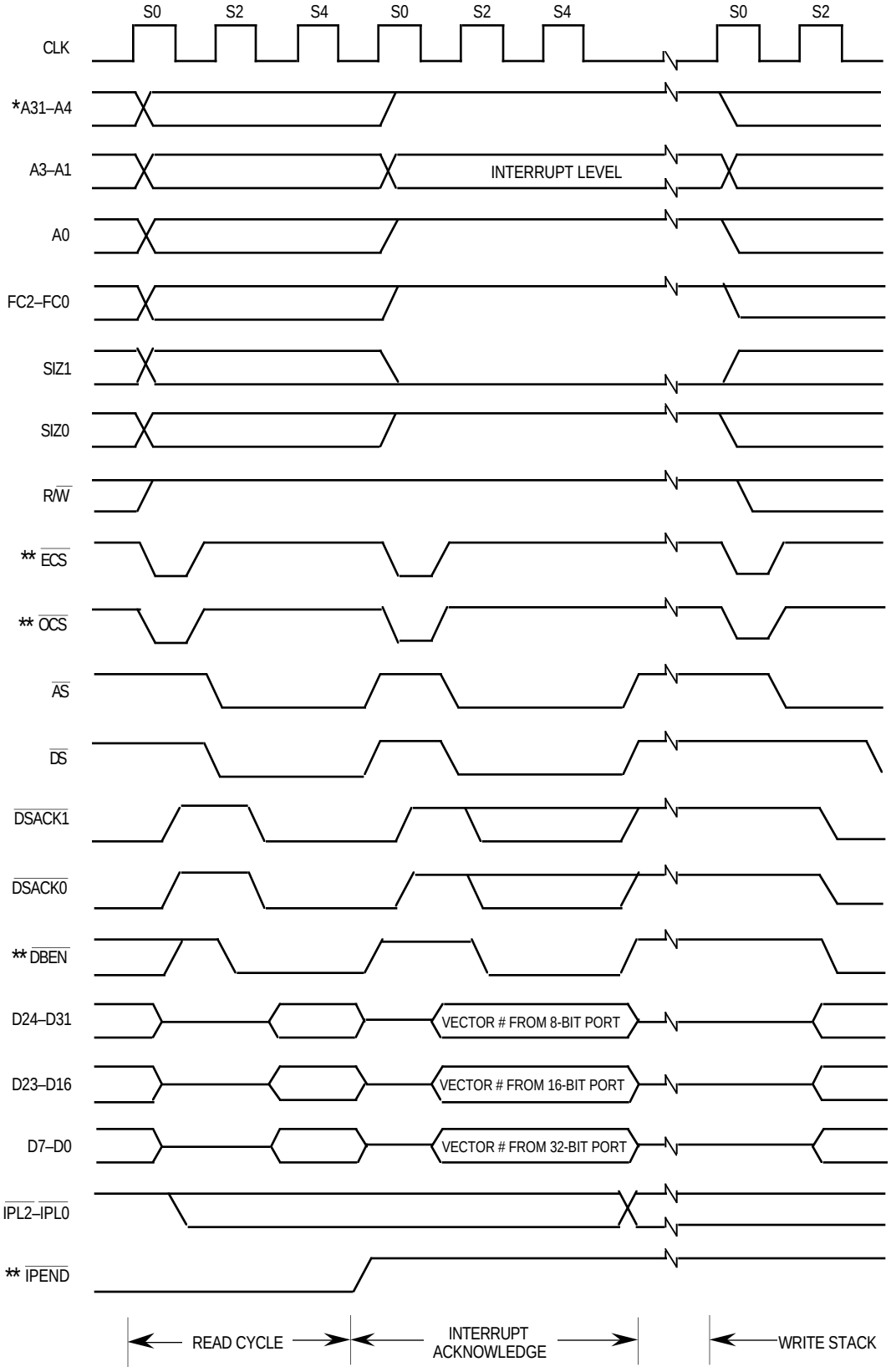
MC68EC020—The processor negates $\overline{AS}$ and $\overline{DS}$ during state S5. It holds the address valid during S5 to provide address hold time for memory systems. R/$\overline{W}$, SIZ1, SIZ0, and FC2–FC0 also remain valid throughout S5.

The external device keeps its data and $\overline{DSACK1}$/$\overline{DSACK0}$ signals asserted until it detects the negation of $\overline{AS}$ or $\overline{DS}$ (whichever it detects first). The device must remove its data and negate $\overline{DSACK1}$/$\overline{DSACK0}$ within approximately one clock period after sensing the negation of $\overline{AS}$ or $\overline{DS}$. $\overline{DSACK1}$/$\overline{DSACK0}$ signals that remain asserted beyond this limit may be prematurely detected for the next bus cycle.



* For the MC68EC020, A23–A2.
This signal does not apply to the MC68EC020.

Figure 5-27. Long-Word Operand Write—8-Bit Port



* For the MC68EC020, A23–A4.
** This signal does not apply to the MC68EC020.

Figure 5-33. Interrupt Acknowledge Cycle Timing

5.5.3 Halt Operation

When $\overline{\text{HALT}}$ is asserted and $\overline{\text{BERR}}$ is not asserted, the MC68020/EC020 halts external bus activity at the next bus cycle boundary. $\overline{\text{HALT}}$ by itself does not terminate a bus cycle. Negating and reasserting $\overline{\text{HALT}}$ in accordance with the correct timing requirements provides a single-step (bus cycle to bus cycle) operation. The $\overline{\text{HALT}}$ signal affects external bus cycles only; thus, a program that resides in the instruction cache and does not require use of the external bus may continue executing unaffected by $\overline{\text{HALT}}$.

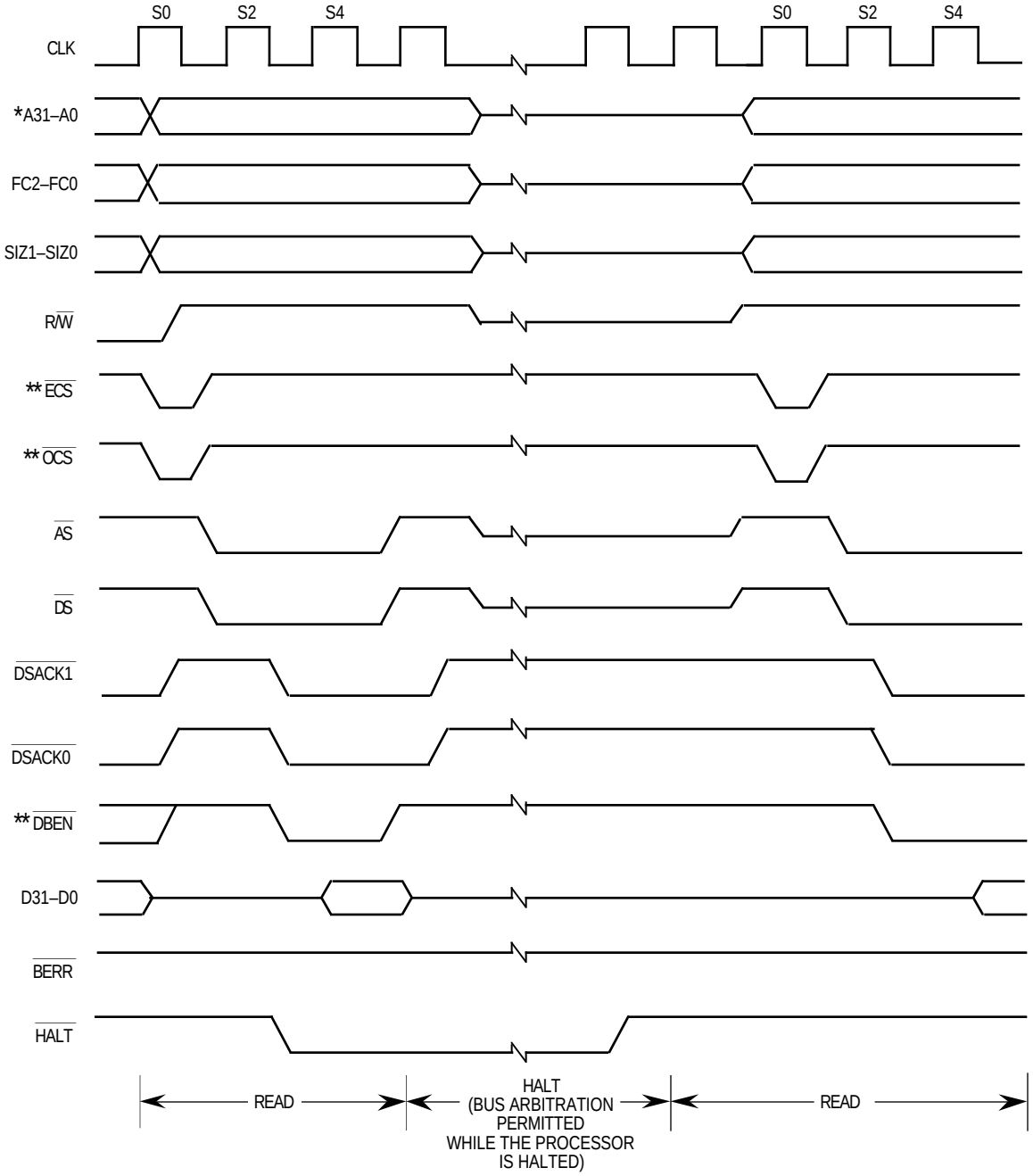
The single-cycle mode allows the user to proceed through (and debug) external processor operations, one bus cycle at a time. Figure 5-41 shows the timing requirements for a single-cycle operation. Since the occurrence of a bus error while $\overline{\text{HALT}}$ is asserted causes a retry operation, the user must anticipate retry cycles while debugging in the single-cycle mode. The single-step operation and the software trace capability allow the system debugger to trace single bus cycles, single instructions, or changes in program flow. These processor capabilities, along with a software debugging package, give complete debugging flexibility.

When the processor completes a bus cycle with the $\overline{\text{HALT}}$ signal asserted, the data bus is placed in the high-impedance state, and the bus control signals ($\overline{\text{AS}}$, $\overline{\text{DS}}$, and, for the MC68020 only, $\overline{\text{ECS}}$ and $\overline{\text{OCS}}$) are negated (not placed in the high-impedance state); A31–A0 for the MC68020 or A23–A0 for the MC68EC020, FC2–FC0, SIZ1, SIZ0, and R/$\overline{\text{W}}$ remain in the same state. The halt operation has no effect on bus arbitration (refer to **5.7 Bus Arbitration**). When bus arbitration occurs while the MC68020/EC020 is halted, the address and control signals (A31–A0, FC2–FC0, SIZ1, SIZ0, R/$\overline{\text{W}}$, $\overline{\text{AS}}$, $\overline{\text{DS}}$, and, for the MC68020 only, $\overline{\text{ECS}}$ and $\overline{\text{OCS}}$) are also placed in the high-impedance state. Once bus mastership is returned to the MC68020/EC020, if $\overline{\text{HALT}}$ is still asserted, A31–A0 for the MC68020 or A23–A0 for the MC68EC020, FC2–FC0, SIZ1, SIZ0, and R/$\overline{\text{W}}$ are again driven to their previous states. The MC68020/EC020 does not service interrupt requests while it is halted (although the MC68020 may assert the $\overline{\text{IPEND}}$ signal as appropriate).

5.5.4 Double Bus Fault

When a bus error or an address error occurs during the exception processing sequence for a previous bus error, a previous address error, or a reset exception, a double bus fault occurs. For example, the processor attempts to stack several words containing information about the state of the machine while processing a bus error exception. If a bus error exception occurs during the stacking operation, the second error is considered a double bus fault. When a double bus fault occurs, the processor halts and asserts $\overline{\text{HALT}}$. Only an external reset operation can restart a halted processor. However, bus arbitration can still occur (refer to **5.7 Bus Arbitration**).

A second bus error or address error that occurs after exception processing has completed (during the execution of the exception handler routine or later) does not cause a double bus fault. A bus cycle that is retried does not constitute a bus error or contribute to a double bus fault. The processor continues to retry the same bus cycle as long as the external hardware requests it.



* For the MC68EC020, A23–A0.
** This signal does not apply to the MC68EC020.

Figure 5-41. Halt Operation Timing

5.6 BUS SYNCHRONIZATION

The MC68020/EC020 overlaps instruction execution—that is, during bus activity for one instruction, instructions that do not use the external bus can be executed. Due to the independent operation of the on-chip cache relative to the operation of the bus controller, many subsequent instructions can be executed, resulting in seemingly nonsequential instruction execution. When this is not desired and the system depends on sequential execution following bus activity, the NOP instruction can be used. The NOP instruction forces instruction and bus synchronization by freezing instruction execution until all pending bus cycles have completed.

An example of the use of the NOP instruction for this purpose is the case of a write operation of control information to an external register in which the external hardware attempts to control program execution based on the data that is written with the conditional assertion of $\overline{\text{BERR}}$. Since the MC68020/EC020 cannot process the bus error until the end of the bus cycle, the external hardware has not successfully interrupted program execution. To prevent a subsequent instruction from executing until the external cycle completes, the NOP instruction can be inserted after the instruction causing the write. In this case, bus error exception processing proceeds immediately after the write and before subsequent instructions are executed. This is an irregular situation, and the use of the NOP instruction for this purpose is not required by most systems.

5.7 BUS ARBITRATION

The bus design of the MC68020/EC020 provides for a single bus master at any one time: either the processor or an external device. One or more of the external devices on the bus can have the capability of becoming bus master. Bus arbitration is the protocol by which an external device becomes bus master; the bus controller in the MC68020/EC020 manages the bus arbitration signals so that the processor has the lowest priority.

Bus arbitration differs in the MC68020 and MC68EC020 due to the absence of $\overline{\text{BGACK}}$ in the MC68EC020. Because of this difference, bus arbitration of the MC68020 and MC68EC020 is discussed separately.

External devices that need to obtain the bus must assert the bus arbitration signals in the sequences described in **5.7.1 MC68020 Bus Arbitration** or **5.7.2 MC68EC020 Bus Arbitration**. Systems having several devices that can become bus master require external circuitry to assign priorities to the devices, so that when two or more external devices attempt to become bus master at the same time, the one having the highest priority becomes bus master first.

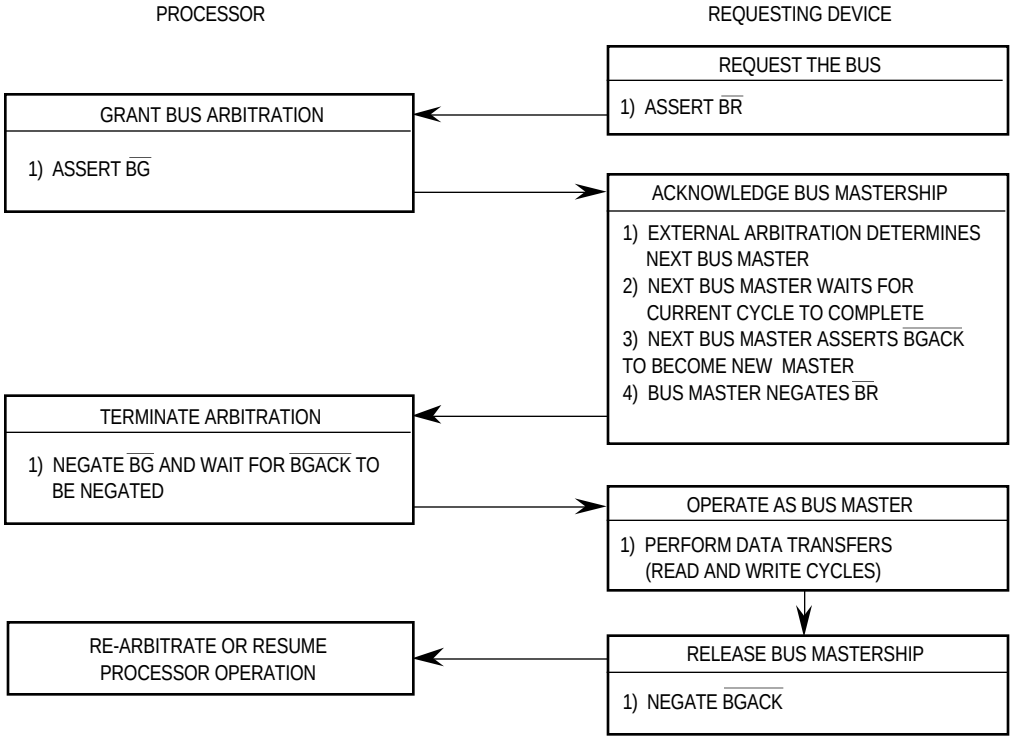


Figure 5-42. MC68020 Bus Arbitration Flowchart for Single Request

The timing diagram (see Figure 5-43) shows that $\overline{BR}$ is negated at the time that $\overline{BGACK}$ is asserted. This type of operation applies to a system consisting of the processor and one device capable of bus mastership. In a system having a number of devices capable of bus mastership, the $\overline{BR}$ line from each device can be wire-ORed to the processor. In such a system, more than one bus request can be asserted simultaneously.

The timing diagram in Figure 5-43 shows that $\overline{BG}$ is negated a few clock cycles after the transition of $\overline{BGACK}$. However, if bus requests are still pending after the negation of $\overline{BG}$, the processor asserts another $\overline{BG}$ within a few clock cycles after it was negated. This additional assertion of $\overline{BG}$ allows external arbitration circuitry to select the next bus master before the current bus master has finished with the bus. The following paragraphs provide additional information about the three steps in the arbitration process.

Bus arbitration requests are recognized during normal processing, $\overline{RESET}$ assertion, $\overline{HALT}$ assertion, and when the processor has halted due to a double bus fault.

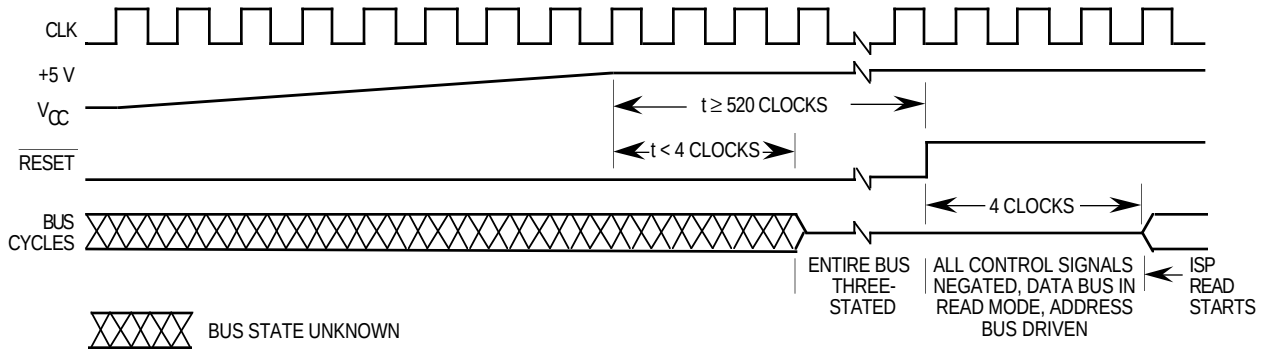


Figure 5-51. Initial Reset Operation Timing

Resetting the processor causes any bus cycle in progress to terminate as if $\overline{\text{DSACK1}}$/$\overline{\text{DSACK0}}$ or $\overline{\text{BERR}}$ had been asserted. In addition, the processor initializes registers appropriately for a reset exception. Exception processing for a reset operation is described in **Section 6 Exception Processing**.

When a RESET instruction is executed, the processor drives the $\overline{\text{RESET}}$ signal for 512 clock cycles. In this case, the processor resets the external devices of the system, and the internal registers of the processor are unaffected. The external devices connected to the $\overline{\text{RESET}}$ signal are reset at the completion of the RESET instruction. An external $\overline{\text{RESET}}$ signal that is asserted to the processor during execution of a RESET instruction must extend beyond the reset period of the instruction by at least eight clock cycles to reset the processor. Figure 5-52 shows the timing information for the RESET instruction.

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. The saved value of the PC is the logical address of the instruction following the instruction that caused the trap. For all instruction traps other than TRAP, a pointer to the instruction that caused the trap is also saved. Instruction execution resumes at the address in the exception vector after the required instruction prefetches.

6.1.5 Illegal Instruction and Unimplemented Instruction Exceptions

An illegal instruction is an instruction that contains any bit pattern in its first word that does not correspond to the bit pattern of the first word of a valid MC68020/EC020 instruction or a MOVEC instruction with an undefined register specification field in the first extension word. An illegal instruction exception corresponds to vector number 4 and occurs when the processor attempts to execute an illegal instruction.

An illegal instruction exception is also taken if a breakpoint acknowledge bus cycle (see **Section 5 Bus Operation**) is terminated with the assertion of the BERR signal. This implies that the external circuitry did not supply an instruction word to replace the BKPT instruction word in the instruction pipe.

Instruction word patterns with bits 15–12 = 1010 are referred to as unimplemented instructions with A-line opcodes. When the processor attempts to execute an unimplemented instruction with an A-line opcode, an exception is generated with vector number 10, permitting efficient emulation of unimplemented instructions.

Instructions that have word patterns with bits 15–12 = 1111, bits 11–9 = 000, and defined word patterns for subsequent words, are legal PMMU instructions. Instructions that have bits 15–12 of the first words = 1111, bits 11–9 = 000, and undefined patterns in the subsequent words, are treated as unimplemented instructions with F-line opcodes when execution is attempted in the supervisor privilege level. When execution of the same instruction is attempted in the user privilege level, a privilege violation exception is taken. The exception vector number for an unimplemented instruction with an F-line opcode is 11.

The word patterns with bits 15–12 = 1111 and bits 11–9 $\neq$ 000 are used for coprocessor instructions. When the processor identifies a coprocessor instruction, it runs a bus cycle referencing CPU space type $2 (refer to **Section 2 Processing States**) and addressing one of eight coprocessors (0–7, according to bits 11–9). If the addressed coprocessor is not included in the system and the cycle terminates with the assertion of BERR, the instruction takes an unimplemented instruction (F-line opcode) exception. The system can emulate the functions of the coprocessor with an F-line exception handler. Refer to **Section 7 Coprocessor Interface Description** for more details.

The priority scheme is very important in determining the order in which exception handlers execute when several exceptions occur at the same time. As a general rule, the lower the priority of an exception, the sooner the handler routine for that exception executes. For example, if simultaneous trap, trace, and interrupt exceptions are pending, the exception processing for the trap occurs first, followed immediately by exception processing for the trace, and then for the interrupt. When the processor resumes normal instruction execution, it is in the interrupt handler, which returns to the trace handler, which returns to the trap exception handler. This rule does not apply to the reset exception; its handler is executed first even though it has the highest priority because the reset operation clears all other exceptions.

6.1.12 Return from Exception

After the MC68020/EC020 has completed exception processing for all pending exceptions, it resumes normal instruction execution at the address in the vector for the last exception processed. Once the exception handler has completed execution, the processor must return to the system context prior to the exception (if possible). The RTE instruction returns from the handler to the previous system context for any exception.

When the processor executes an RTE instruction, it examines the stack frame on top of the active supervisor stack to determine if it is a valid frame and what type of context restoration it requires. The following paragraphs describe the processing for each of the stack frame types; refer to **6.3 Coprocessor Considerations** for a description of the stack frame type formats.

For a normal four-word frame, 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.

For the throwaway four-word frame, the processor reads the SR value from the frame, increments the active stack pointer by eight, updates the SR with the value read from the stack, and then begins RTE processing again, as shown in Figure 6-7. The processor reads a new format word from the stack frame on top of the active stack (which may or may not be the same stack used for the previous operation) and performs the proper operations corresponding to that format. In most cases, the throwaway frame is on the interrupt stack and when the SR value is read from the stack, the S and M bits are set. In that case, there is a normal four-word frame or a ten-word coprocessor midinstruction frame on the master stack. However, the second frame may be any format (even another throwaway frame) and may reside on any of the three system stacks.

For the six-word stack frame, the processor restores the SR and PC values from the stack, increments the active supervisor stack pointer by 12, and resumes normal instruction execution.

If the coprocessor requires additional information to evaluate the condition, the cpDBcc instruction can include this information in extension words. These extension words follow the word containing the coprocessor condition selector field in the cpDBcc instruction format.

The last word of the instruction contains the displacement for the cpDBcc instruction. This displacement is a twos-complement 16-bit value that is sign-extended to long-word size when it is used in a destination address calculation.

7.2.2.3.2 Protocol. Figure 7-8 shows the protocol for the cpDBcc instructions. The MC68020/EC020 transfers the condition selector to the coprocessor by writing the word following the operation word to the condition CIR. The main processor then reads the response CIR to determine its next action. The coprocessor can use a response primitive to request any services necessary to evaluate the condition. If the coprocessor returns the true condition indicator, the main processor executes the next instruction in the instruction stream. If the coprocessor returns the false condition indicator, the main processor decrements the low-order word of the register specified by bits 2–0 of the F-line operation word. If this register contains minus one (–1) after being decremented, the main processor executes the next instruction in the instruction stream. If the register does not contain minus one (–1) after being decremented, the main processor branches to the destination address to continue instruction execution.

The MC68020/EC020 adds the displacement to the scanPC (refer to **7.4.1 ScanPC**) to determine the address of the next instruction. The scanPC must point to the 16-bit displacement in the instruction stream when the destination address is calculated.

7.2.2.4 TRAP ON COPROCESSOR CONDITION INSTRUCTION. The trap on coprocessor condition instruction allows the programmer to initiate exception processing based on conditions related to the coprocessor operation.

7.2.2.4.1 Format. Figure 7-13 shows the format of the trap on coprocessor condition instruction, denoted by the cpTRAPcc mnemonic.

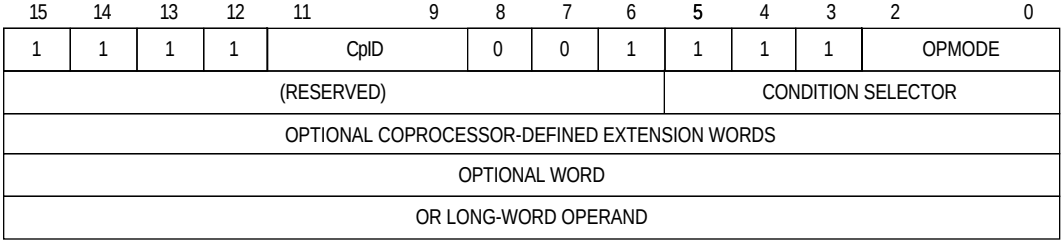


Figure 7-13. Trap on Coprocessor Condition Instruction Format (cpTRAPcc)

The first word of the cpTRAPcc instruction, the F-line operation word contains the CpID field in bits 11–9 and 001111 in bits 8–3 to identify the cpTRAPcc instruction. Bits 2–0 of the cpTRAPcc F-line operation word specify the opmode, which selects the instruction format. The instruction format can include zero, one, or two operand words.

The value in the main processor scanPC at the time this primitive is received is saved in the scanPC field of the postinstruction exception stack frame. The value of the PC saved is the F-line operation word address of the coprocessor instruction during which the primitive is received.

When the MC68020/EC020 receives the take postinstruction exception primitive, it assumes that the coprocessor either completed or aborted the instruction with an exception. If the exception handler does not modify the stack frame, the MC68020/EC020 returns from the exception handler to begin execution at the location specified by the scanPC field of the stack frame. This location should be the address of the next instruction to be executed.

The coprocessor uses this primitive to request exception processing when it completes or aborts an instruction while the main processor is awaiting a normal response. For a general category instruction, the response is a release; for a conditional category instruction, it is an evaluated true/false condition indicator. Thus, the operation of the MC68020/EC020 in response to this primitive is compatible with standard M68000 family instruction related exception processing (for example, the divide-by-zero exception).

7.5 EXCEPTIONS

Various exception conditions related to the execution of coprocessor instructions may occur. Whether an exception is detected by the main processor or by the coprocessor, the main processor coordinates and performs exception processing. Servicing these coprocessor-related exceptions is an extension of the protocol used to service standard M68000 family exceptions. That is, when either the main processor detects an exception or is signaled by the coprocessor that an exception condition has occurred, the main processor proceeds with exception processing as described in **Section 6 Exception Processing**.

7.5.1 Coprocessor-Detected Exceptions

Coprocessor interface exceptions that the coprocessor detects, as well as those that the main processor detects, are usually classified as coprocessor-detected exceptions. Coprocessor-detected exceptions can occur during M68000 coprocessor interface operations, internal operations, or other system-related operations of the coprocessor.

Most coprocessor-detected exceptions are signaled to the main processor through the use of one of the three take exception primitives defined for the M68000 coprocessor interface. The main processor responds to these primitives as described in **7.4.18 Take Preinstruction Exception Primitive**, **7.4.19 Take Midinstruction Exception Primitive**, and **7.4.20 Take Postinstruction Exception Primitive**. However, not all coprocessor-detected exceptions are signaled by response primitives. Coprocessor-detected format errors during the cpSAVE or cpRESTORE instruction are signaled to the main processor using the invalid format word described in **7.2.3.2.3 Invalid Format Words**.

Table 7-6. Exceptions Related to Primitive Processing

Primitive	Protocol	F-Line	Other
Busy			
Null			
Supervisory Check* Other: Privilege Violation if S-Bit in the SR = 0			X
Transfer Operation Word*			
Transfer from Instruction Stream* Protocol: If Length Field Is Odd (Zero Length Legal)	X		
Evaluate and Transfer Effective Address Protocol: If Used with Conditional Instruction F-Line: If EA in Opword Is NOT Control Alterable	X	X	
Evaluate Effective Address and Transfer Data Protocol: 1. If Used with Conditional Instructions 2. Length Is Not 1, 2, or 4 and EA = Register Direct 3. If EA = Immediate and Length Odd and Greater Than 1 4. Attempt to Write to Unalterable Address Even if Address Declared Legal in Primitive F-Line: Valid EA Field Does Not Match EA in Opword	X	X	
Write to Previously Evaluated Effective Address Protocol: If Used with Conditional Instruction	X		
Take Address and Transfer Data*			
Transfer to/from Top of Stack* Protocol: Length Field Other Than 1, 2, or 4	X		
Transfer Single Main Processor Register*			
Transfer Main Processor Control Register Protocol: Invalid Control Register Select Code	X		
Transfer Multiple Main Processor Registers*			
Transfer Multiple Coprocessor Registers Protocol: 1. If Used with Conditional Instructions 2. Odd Length Value F-Line: 1. EA Not Control Alterable or (An)+ for CP to Memory Transfer 2. EA Not Control Alterable or –(An) for Memory to CP Transfer	X	X	
Transfer Status and ScanPC Protocol: If Used with Conditional Instruction Other: 1. Trace—Trace Made Pending if MC68020/EC020 in “Trace on Change of Flow” Mode and DR = 1 2. Address Error—If Odd Value Written to ScanPC	X		X
Take Preinstruction, Midinstruction, or Postinstruction Exception Exception Depends on Vector Supplies in Primitive	X	X	X

*Use of this primitive with CA = 0 will cause protocol violation on conditional instructions.

Abbreviations:

EA—Effective Address
CP—Coprocessor

Busy

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	PC	1	0	0	1	0	0	0	0	0	0	0	0	0	0

Transfer Multiple Coprocessor Registers

15	14	13	12	11	10	9	8	7	0
CA	PC	DR	0	0	0	0	1	LENGTH	

Transfer Status Register and ScanPC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	0	0	1	SP	0	0	0	0	0	0	0	0

Supervisor Check

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	PC	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Take Address and Transfer Data

15	14	13	12	11	10	9	8	7	0
CA	PC	DR	0	0	1	0	1	LENGTH	

Transfer Multiple Main Processor Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	0	1	1	0	0	0	0	0	0	0	0	0

Transfer Operation Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	0	1	1	1	0	0	0	0	0	0	0	0

Null

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	1	0	0	IA	0	0	0	0	0	0	PF	TF

Evaluate and Transfer Effective Address

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	1	0	1	0	0	0	0	0	0	0	0	0

Ñ DÑ

Ñ EÑ