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	68020
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	20MHz
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	132-BCQFP
Supplier Device Package	132-CQFP (24x24)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mc68020fe20e

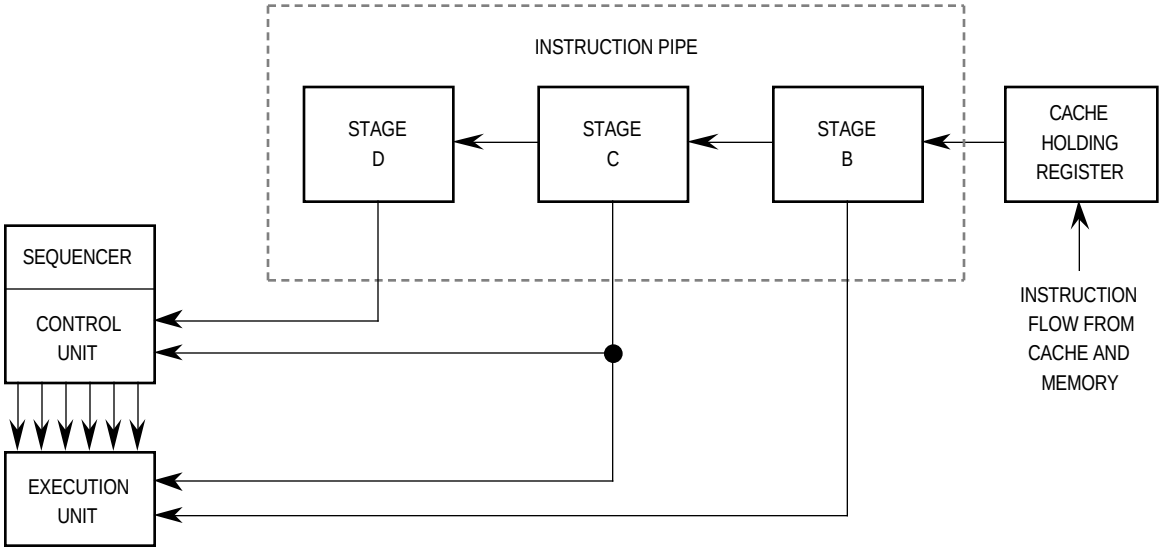


Figure 1-5. Instruction Pipe

The sequencer is either executing microinstructions or awaiting completion of accesses that are necessary to continue executing microcode. The bus controller is responsible for all bus activity. The sequencer controls the bus controller, instruction execution, and internal processor operations such as the calculation of effective addresses and the setting of condition codes. The sequencer initiates instruction word prefetches and controls the validation of instruction words in the instruction pipe.

Prefetch requests are simultaneously submitted to the cache holding register, the instruction cache, and the bus controller. Thus, even if the instruction cache is disabled, an instruction prefetch may hit in the cache holding register and cause an external bus cycle to be aborted.

1.7 CACHE MEMORY

Due to locality of reference, instructions that are used in a program have a high probability of being reused within a short time. Additionally, instructions that reside in proximity to the instructions currently in use also have a high probability of being utilized within a short period. To exploit these locality characteristics, the MC68020/EC020 contains an on-chip instruction cache.

The cache improves the overall performance of the system by reducing the number of bus cycles required by the processor to fetch information from memory and by increasing the bus bandwidth available for other bus masters in the system.

3.1 SIGNAL INDEX

The input and output signals for the MC68020/EC020 are listed in Table 3-1. Both the names and mnemonics are shown along with brief signal descriptions. Signals that are implemented in the MC68020, but not in the MC68EC020, have an asterisk (*) preceding the signal name in Table 3-1. Also, note that the address bus is 32 bits wide for the MC68020 and 24 bits wide for the MC68EC020. For more detail on each signal, refer to the paragraph in this section named for the signal and the reference in that paragraph to a description of the related operations.

Timing specifications for the signals listed in Table 3-1 can be found in **Section 10 Electrical Characteristics**.

3.2 FUNCTION CODE SIGNALS (FC2–FC0)

These three-state outputs identify the address space of the current bus cycle. Table 2-1 shows the relationships of the function code signals to the privilege levels and the address spaces. Refer to **Section 2 Processing States** for more information.

3.3 ADDRESS BUS (A31–A0, MC68020)(A23–A0, MC68EC020)

These three-state outputs provide the address for the current bus cycle, except in the CPU address space. Refer to **Section 2 Processing States** for more information on the CPU address space. A31 is the most significant address signal for the MC68020; A23 is the most significant address signal for the MC68EC020. The upper eight bits (A31–A24) are used internally by the MC68EC020 to access the internal instruction cache address tag. Refer to **Section 5 Bus Operation** for information on the address bus and its relationship to bus operation.

3.4 DATA BUS (D31–D0)

These three-state bidirectional signals provide the general-purpose data path between the MC68020/EC020 and all other devices. The data bus can transfer 8, 16, 24, or 32 bits of data per bus cycle. D31 is the most significant bit of the data bus. Refer to **Section 5 Bus Operation** for more information on the data bus and its relationship to bus operation.

3.5 TRANSFER SIZE SIGNALS (SIZ1, SIZ0)

These three-state outputs indicate the number of bytes remaining to be transferred for the current bus cycle. Signals A1, A0, DSACK1, DSACK0, SIZ1, and SIZ0 define the number of bits transferred on the data bus. Refer to **Section 5 Bus Operation** for more information on SIZ1 and SIZ0 and their use in dynamic bus sizing.

input is high or low. Figure 5-1 shows the relationship between the clock signal, a typical input, and its associated internal signal.

Furthermore, for all inputs, the processor latches the level of the input during a sample window around the falling edge of the clock signal. This window is illustrated in Figure 5-2. To ensure that an input signal is recognized on a specific falling edge of the clock, that input must be stable during the sample window. If an input transitions during the window, the level recognized by the processor is not predictable; however, the processor always resolves the latched level to either a logic high or logic low before using it. In addition to meeting input setup and hold times for deterministic operation, all input signals must obey the protocols described in this section.

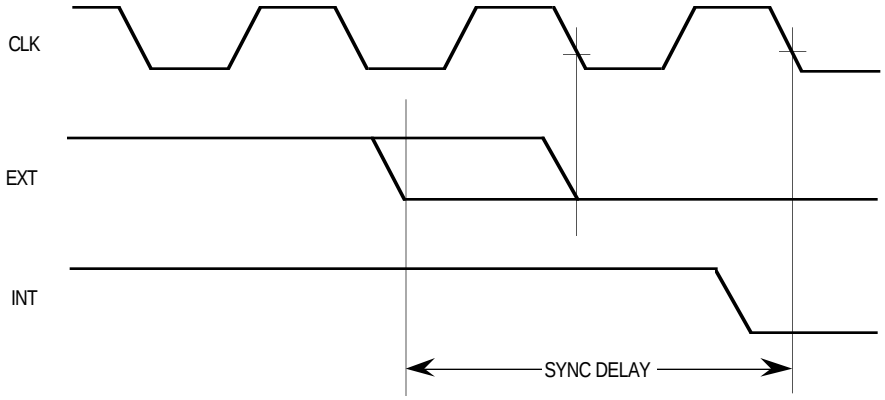


Figure 5-1. Relationship between External and Internal Signals

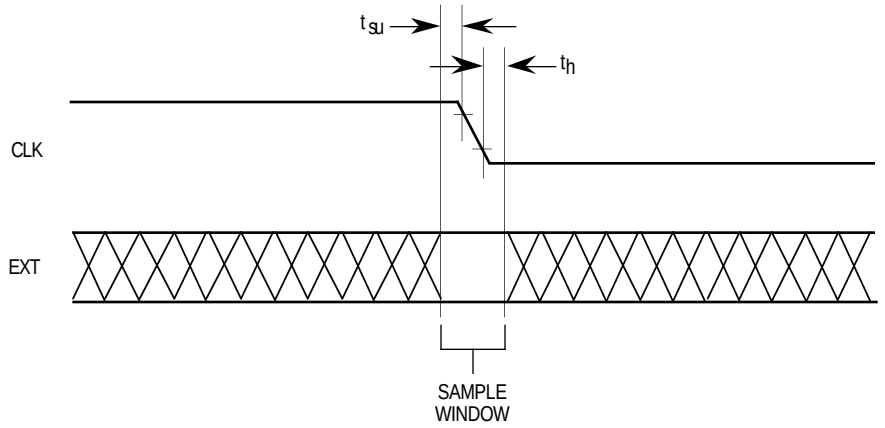


Figure 5-2. Input Sample Window

5.1.1 Bus Control Signals

The MC68020/EC020 initiates a bus cycle by driving the A1–A0, SIZ1, SIZ0, FC2–FC0, and R/$\overline{W}$ outputs. However, if the MC68020/EC020 finds the required instruction in the on-chip cache, the processor aborts the cycle before asserting the $\overline{AS}$.The assertion of $\overline{AS}$ ensures that the cycle has not been aborted by these internal conditions.

State 0

MC68020—The read cycle starts in state 0 (S0). The processor asserts $\overline{ECS}$, indicating the beginning of an external cycle. If the cycle is the first external cycle of a read operation, $\overline{OCS}$ is asserted simultaneously. During S0, the processor places a valid address on A31–A0 and valid function codes on FC2–FC0. The function codes select the address space for the cycle. The processor drives R/$\overline{W}$ high for a read cycle and negates $\overline{DBEN}$ to disable the data buffers. SIZ0 and SIZ1 become valid, indicating the number of bytes requested to be transferred.

MC68EC020—The read cycle starts in S0. During S0, the processor places a valid address on A23–A0 and valid function codes on FC2–FC0. The function codes select the address space for the cycle. The processor drives R/$\overline{W}$ high for a read cycle. SIZ0 and SIZ1 become valid, indicating the number of bytes requested to be transferred.

State 1

MC68020—One-half clock later in state 1 (S1), the processor asserts $\overline{AS}$, indicating that the address on the address bus is valid. The processor also asserts $\overline{DS}$ during S1. In addition, the $\overline{ECS}$ (and $\overline{OCS}$, if asserted) signal is negated during S1.

MC68EC020—One-half clock later in S1, the processor asserts $\overline{AS}$, indicating that the address on the address bus is valid. The processor also asserts $\overline{DS}$ during S1.

State 2

MC68020—During state 2 (S2), the processor asserts $\overline{DBEN}$ to enable external data buffers. The selected device uses R/$\overline{W}$, SIZ1–SIZ0, A1–A0, and $\overline{DS}$ to place its information on the data bus. Any or all of the bytes (D31–D24, D23–D16, D15–D8, and D7–D0) are selected by SIZ1–SIZ0 and A1–A0. Concurrently, the selected device asserts $\overline{DSACK1}$/$\overline{DSACK0}$.

MC68EC020—During S2, the selected device uses R/$\overline{W}$, SIZ1–SIZ0, A1–A0, and $\overline{DS}$ to place its information on the data bus. Any or all of the bytes (D31–D24, D23–D16, D15–D8, and D7–D0) are selected by SIZ1–SIZ0 and A1–A0. Concurrently, the selected device asserts $\overline{DSACK1}$/$\overline{DSACK0}$.

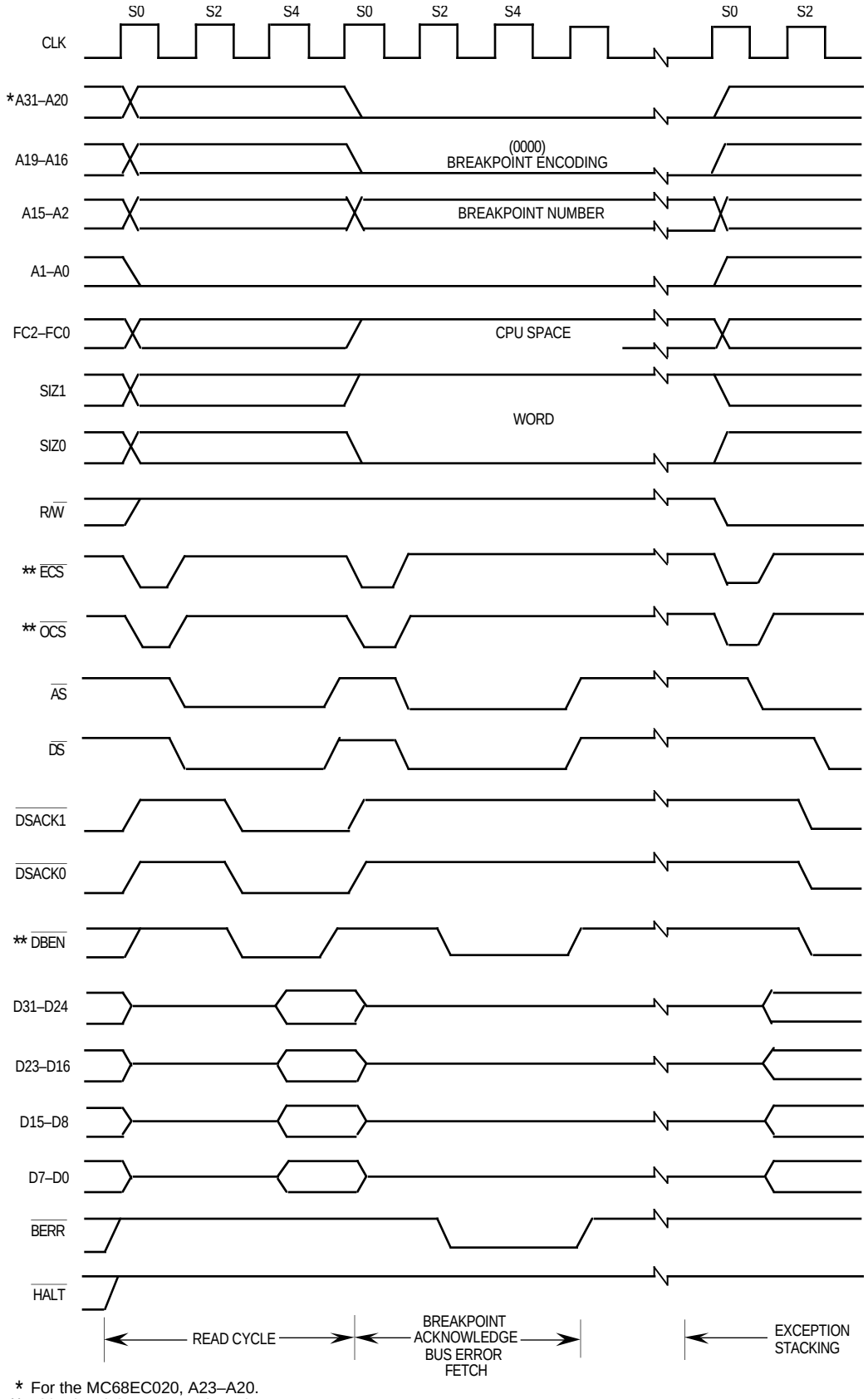
State 3

MC68020/EC020—As long as at least one of the $\overline{DSACK1}$/$\overline{DSACK0}$ signals is recognized by the end of S2 (meeting the asynchronous input setup time requirement), data is latched on the next falling edge of the clock, and the cycle terminates. If $\overline{DSACK1}$/$\overline{DSACK0}$ is not recognized by the start of state 3 (S3), the processor inserts wait states instead of proceeding to states 4 and 5. To ensure that wait states are inserted, both $\overline{DSACK1}$ and $\overline{DSACK0}$ must remain negated throughout the asynchronous input setup and hold times around the end of S2. If wait states are added, the processor continues to sample the $\overline{DSACK1}$/$\overline{DSACK0}$ signals on the falling edges of the clock until an assertion is recognized.

5.4.1.2 AUTOVECTOR INTERRUPT ACKNOWLEDGE CYCLE. When the interrupting device cannot supply a vector number, it requests an automatically generated vector or autovector. Instead of placing a vector number on the data bus and asserting DSACK1/DSACK0, the device asserts AVEC to terminate the cycle. The DSACK1/DSACK0 signals may not be asserted during an interrupt acknowledge cycle terminated by AVEC.

The vector number supplied in an autovector operation is derived from the interrupt level of the current interrupt. When AVEC is asserted instead of DSACK1/DSACK0 during an interrupt acknowledge cycle, the MC68020/EC020 ignores the state of the data bus and internally generates the vector number, the sum of the interrupt level plus 24 ($18). Seven distinct autovectors, which correspond to the seven levels of interrupt available with IPL2–IPL0, can be used. Figure 5-34 shows the timing for an autovector operation.

5.4.1.3 SPURIOUS INTERRUPT CYCLE. When a device does not respond to an interrupt acknowledge cycle with AVEC or DSACK1/DSACK0, the external logic typically returns BERR. In this case, the MC68020/EC020 automatically generates 24, the spurious interrupt vector number. If HALT is also asserted, the processor retries the cycle.



* For the MC68EC020, A23–A20.
** This signal does not apply to the MC68EC020.

Figure 5-38. Bus Error without DSACK1/DSACK0

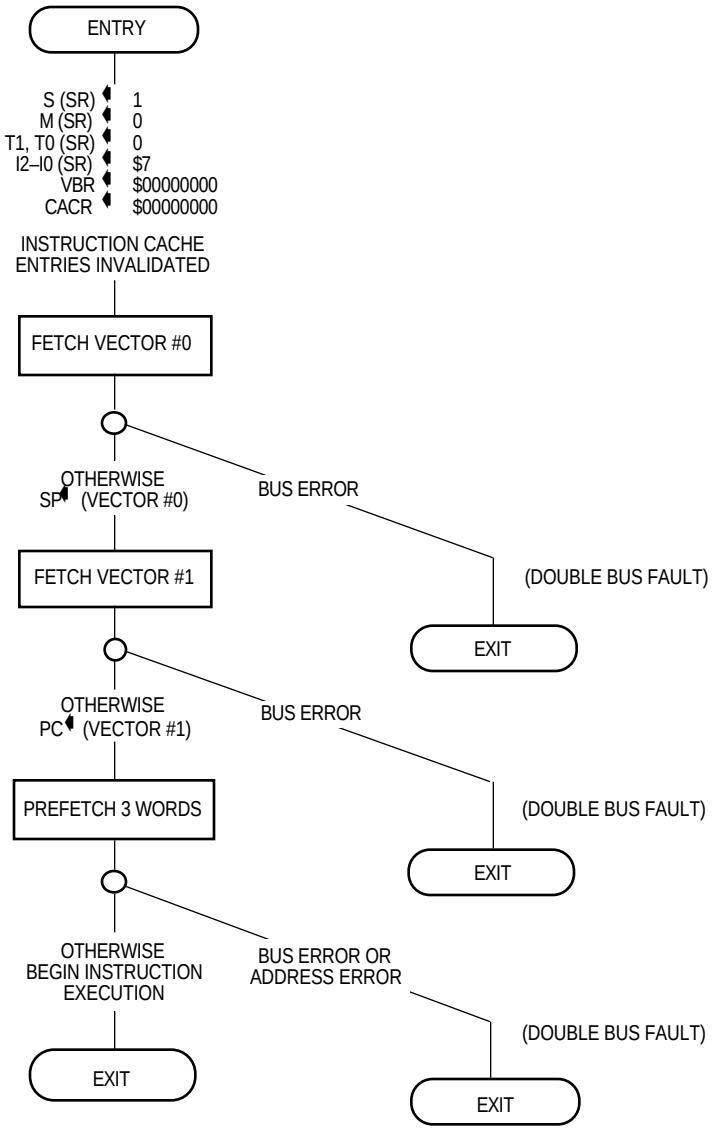


Figure 6-1. Reset Operation Flowchart

The processor begins exception processing for a bus error by making an internal copy of the current SR. The processor then enters the supervisor privilege level (by setting the S-bit in the SR) and clears the T1 and T0 bits in the SR. The processor generates exception vector number 2 for the bus error vector. It saves the vector offset, PC, and the internal copy of the SR on the active supervisor stack. The saved PC value is the logical address of the instruction that was executing at the time the fault was detected. This is not necessarily the instruction that initiated the bus cycle since the processor overlaps

6.1.10 Breakpoint Instruction Exception

To use the MC68020/EC020 in a hardware emulator, it must provide a means of inserting breakpoints in the emulator code and of performing appropriate operations at each breakpoint. For the MC68000 and MC68008, this can be done by inserting an illegal instruction at the breakpoint and detecting the illegal instruction exception from its vector location. However, since the VBR on M68000 family processors MC68010 and later allows arbitrary relocation of exception vectors, the exception address cannot reliably identify a breakpoint. The MC68020/EC020 processor provides a breakpoint capability with a set of breakpoint instructions, $4848–$484F, for eight unique breakpoints. The breakpoint facility also allows external hardware to monitor the execution of a program residing in the on-chip instruction cache without severe performance degradation.

When the MC68020/EC020 executes a breakpoint instruction, it performs a breakpoint acknowledge cycle (read cycle) from CPU space type $0 with address lines A4–A2 corresponding to the breakpoint number. Refer to **Section 5 Bus Operation** for a description of the breakpoint acknowledge cycle. The external hardware can return either BERR or DSACK1/DSACK0 with an instruction word on the data bus. If the bus cycle terminates with BERR, the processor performs illegal instruction exception processing. If the bus cycle terminates with DSACK1/DSACK0, the processor uses the data returned to replace the breakpoint instruction in the internal instruction pipe and begins execution of that instruction. The remainder of the pipe remains unaltered. In addition, no stacking or vector fetching is involved with the execution of the instruction. Figure 6-6 is a flowchart of the breakpoint instruction execution.

6.1.11 Multiple Exceptions

When several exceptions occur simultaneously, they are processed according to a fixed priority. Table 6-4 lists the exceptions grouped by characteristics. Each group has a priority from 4–0. Priority 0 has the highest priority.

As soon as the MC68020/EC020 has completed exception processing for a condition when another exception is pending, it begins exception processing for the pending exception instead of executing the exception handler for the original exception condition. Also, whenever a bus error or address error occurs, its exception processing takes precedence over lower priority exceptions and occurs immediately. For example, if a bus error occurs during the exception processing for a trace condition, the system processes the bus error and executes its handler before completing the trace exception processing. However, most exceptions cannot occur during exception processing, and very few combinations of the exceptions shown in Table 6-4 can be pending simultaneously.

When the MC68020/EC020 detects a protocol violation, it does not automatically notify the coprocessor of the resulting exception by writing to the control CIR. However, the exception handling routine may use the MOVES instruction to read the response CIR and thus determine the primitive that caused the MC68020/EC020 to initiate protocol violation exception processing. The main processor initiates exception processing using the midinstruction stack frame (refer to Figure 7-43) and the coprocessor protocol violation exception vector number 13. If the exception handler does not modify the stack frame, the main processor reads the response CIR again following the execution of an RTE instruction to return from the exception handler. This protocol allows extensions to the M68000 coprocessor interface to be emulated in software by a main processor that does not provide hardware support for these extensions. Thus, the protocol violation is transparent to the coprocessor if the primitive execution can be emulated in software by the main processor.

7.5.2.2 F-LINE EMULATOR EXCEPTIONS. The F-line emulator exceptions detected by the MC68020/EC020 are either explicitly or implicitly related to the encodings of F-line operation words in the instruction stream. If the main processor determines that an F-line operation word is not valid, it initiates F-line emulator exception processing. Any F-line operation word with bits 8–6 = 110 or 111 causes the MC68020/EC020 to initiate exception processing without initiating any communication with the coprocessor for that instruction. Also, an operation word with bits 8–6 = 000–101 that does not map to one of the valid coprocessor instructions in the instruction set causes the MC68020/EC020 to initiate F-line emulator exception processing. If the F-line emulator exception is either of these two situations, the main processor does not write to the control CIR prior to initiating exception processing.

F-line exceptions can also occur if the operations requested by a coprocessor response primitive are not compatible with the effective address type in bits 5–0 of the coprocessor instruction operation word. The F-line emulator exceptions that can result from the use of the M68000 coprocessor response primitives are summarized in Table 7-6. If the exception is caused by receiving an invalid primitive, the main processor aborts the coprocessor instruction in progress by writing an abort mask (refer to **7.3.2 Control CIR**) to the control CIR prior to F-line emulator exception processing.

Another type of F-line emulator exception occurs when a bus error occurs during the CIR access that initiates a coprocessor instruction. The main processor assumes that the coprocessor is not present and takes the exception.

When the main processor initiates F-line emulator exception processing, it uses the four-word preinstruction exception stack frame (refer to Figure 7-41) and the F-line emulator exception vector number 11. Thus, if the exception handler does not modify the stack frame, the main processor attempts to restart the instruction that caused the exception after it executes an RTE instruction to return from the exception handler.

If the cause of the F-line exception can be emulated in software, the handler stores the results of the emulation in the appropriate registers of the programming model and in the status register field of the saved stack frame. The exception handler adjusts the program

SECTION 8
INSTRUCTION EXECUTION TIMING

This section describes the instruction execution and operations (table searches, etc.) of the MC68020/EC020 in terms of external clock cycles. It provides accurate execution and operation timing guidelines but not exact timings for every possible circumstance. This approach is used since exact execution time for an instruction or operation is highly dependent on memory speeds and other variables. The timing numbers presented in this section allow the assembly language programmer or compiler writer to predict timings needed to evaluate the performance of the MC68020/EC020.

In this section, instruction and operation times are shown in clock cycles, which eliminates clock frequency dependencies.

8.1 TIMING ESTIMATION FACTORS

The advanced architecture of the MC68020/EC020 makes exact instruction timing calculations difficult due to the effects of:

1. An On-Chip Instruction Cache and Instruction Prefetch
2. Operand Misalignment
3. Bus Controller/Sequence Concurrency
4. Instruction Execution Overlap

These factors make MC68020/EC020 instruction set timing difficult to calculate on a single instruction basis since instructions vary in execution time from one context to another. A detailed explanation of each of these factors follows.

8.1.1 Instruction Cache and Prefetch

The on-chip cache of the MC68020/EC020 is an instruction-only cache. Its purpose is to increase execution efficiency by providing a quick store for instructions.

Instruction prefetches that hit in the cache will occur with no delay in instruction execution. Instruction prefetches that miss in the cache will cause an external memory cycle to be performed, which may overlap with internal instruction execution. Thus, while the execution unit of the microprocessor is busy, the bus controller prefetches the next instruction from external memory. Both cases are illustrated in later examples.

8.2.8 Arithmetic/Logical Instructions

The arithmetic/logical instructions table indicates the number of clock periods needed for the processor to perform the specified arithmetic/logical operation using the specified addressing mode. It also includes, in worst case, the amount of time needed to prefetch the next instruction. Footnotes specify when to add either fetch address or fetch immediate effective address time. This sum gives the total effective execution time for the operation using the specified addressing mode. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Instruction			Best Case	Cache Case	Worst Case
*	ADD	EA,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	ADDA	EA,An	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	ADD	Dn,EA	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
*	AND	EA,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	AND	Dn,EA	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
*	EOR	Dn,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	EOR	Dn,Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
*	OR	EA,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	OR	Dn,EA	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
*	SUB	EA,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	SUBA	EA,An	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	SUB	Dn,EA	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
*	CMP	EA,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	CMPA	EA,An	**1**(0/0/0)	**4**(0/0/0)	**4**(0/1/0)
**	CMP2	EA,Rn	**16**(1/0/0)	**18**(1/0/0)	**18**(1/1/0)
*	MUL.W	EA,Dn	**25**(0/0/0)	**27**(0/0/0)	**28**(0/1/0)
**	MUL.L	EA,Dn	**41**(0/0/0)	**43**(0/0/0)	**44**(0/1/0)
*	DIVU.W	EA,Dn	**42**(0/0/0)	**44**(0/0/0)	**44**(0/1/0)
**	DIVU.L	EA,Dn	**76**(0/0/0)	**78**(0/0/0)	**79**(0/1/0)
*	DIVS.W	EA,Dn	**54**(0/0/0)	**56**(0/0/0)	**57**(0/1/0)
**	DIVS.L	EA,Dn	**88**(0/0/0)	**90**(0/0/0)	**91**(0/1/0)

*Add Fetch Effective Address Time

**Add Fetch Immediate Address Time

SECTION 9
APPLICATIONS INFORMATION

This section, which provides guidelines for using the MC68020/EC020, contains information on floating-point units, byte select logic, power and ground considerations, clock driver, memory interface, access time calculations, module support, and access levels.

9.1 FLOATING-POINT UNITS

Floating-point support for the MC68020/EC020 is provided by the MC68881 floating-point coprocessor or the MC68882 enhanced floating-point coprocessor. Both devices offer a full implementation of the *IEEE Standard for Binary Floating-Point Arithmetic* (754). The MC68882 is a pin- and software-compatible upgrade of the MC68881, with an optimized MPU interface that provides over 1.5 times the performance of the MC68881 at the same clock frequency.

Both coprocessors provide a logical extension to the integer data processing capabilities of the main processor. They contain a high-performance floating-point arithmetic unit and a set of floating-point data registers that are utilized in a manner that is analogous to the use of the integer data registers of the processor. The MC68881/MC68882 instruction set, a natural extension of all earlier members of the M68000 family, supports all addressing modes and data types of the host MC68020/EC020. The programmer perceives the MC68020/EC020 coprocessor execution model as if both devices are implemented on one chip. In addition to supporting the full IEEE standard, the MC68881 and MC68882 provide a full set of trigonometric and transcendental functions, on-chip constants, and a full 80-bit extended-precision real data format.

The interface of the MC68020/EC020 to the MC68881 or MC68882 is easily tailored to system cost/performance needs. The MC68020/EC020 and the MC68881/MC68882 communicate via standard asynchronous M68000 bus cycles. All data transfers are performed by the main processor at the request of the MC68881/MC68882; thus, memory management, bus errors, address errors, and bus arbitration function as if the MC68881/MC68882 instructions are executed by the main processor. The floating-point unit and the processor can operate at different clock speeds, and up to seven floating-point coprocessors can simultaneously reside in an MC68020/EC020 system.

Figure 9-1 illustrates the coprocessor interface connection of an MC68881/MC68882 to an MC68020/EC020 (uses entire 32-bit data bus). The MC68881/MC68882 is configured to operate with a 32-bit data bus when both its A0 and SIZE pins are connected to V_{CC}. Refer to the MC68881UM/AD, *MC68881/MC68882 Floating-Point Coprocessor User's Manual*, for configuring the MC68881/MC68882 for smaller data bus widths.

9.2 BYTE SELECT LOGIC FOR THE MC68020/EC020

The MC68020/EC020 architecture supports byte, word, and long-word operand transfers to any 8-, 16-, or 32-bit data port, regardless of alignment. This feature allows the programmer to write code that is not bus-width specific. When accessed, the peripheral or memory subsystem reports its actual port size to the controller, and the MC68020/EC020 then dynamically sizes the data transfer accordingly, using multiple bus cycles when necessary. The following paragraphs describe the generation of byte select control signals that enable the dynamic bus sizing mechanism, the transfer of differently sized operands, and the transfer of misaligned operands to operate correctly.

The following signals control the MC68020/EC020 operand transfer mechanism:

A1, A0 — Address signals. The most significant byte of the operand to be transferred is addressed directly.

SIZ1, SIZ0 — Transfer size signals. Output of the MC68020/EC020. These indicate the number of bytes of an operand remaining to be transferred during a given bus cycle.

R/W — Read/Write signal. Output of the MC68020/EC020. For byte select generation in MC68020/EC020 systems.

DSACK1, DSACK0 — Data transfer and size acknowledge signals. Driven by an asynchronous port to indicate the actual bus width of the port.

The MC68020/EC020 assumes that 16-bit ports are situated on data lines D31–D16, and that 8-bit ports are situated on data lines D31–D24. This ensures that the following logic works correctly with the MC68020/EC020's on-chip internal-to-external data bus multiplexer. Refer to **Section 5 Bus Operation** for more details on the dynamic bus sizing mechanism.

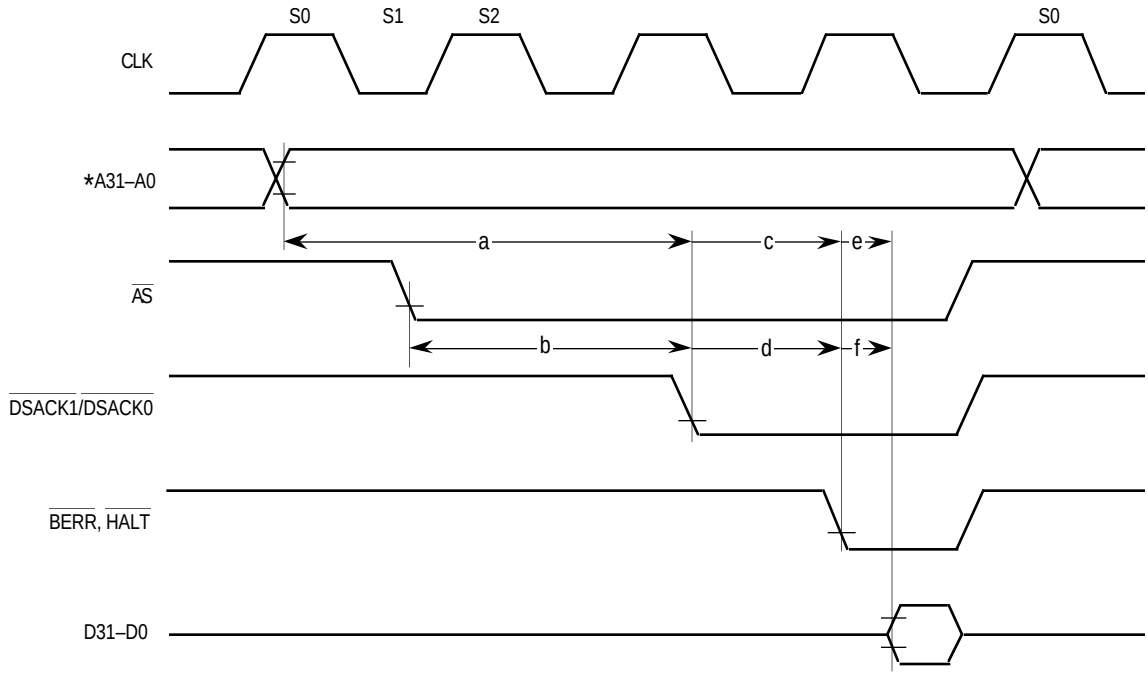
The need for byte select signals is best illustrated by an example. Consider a long-word write cycle to an odd address in word-organized memory. The transfer requires three bus cycles to complete. The first bus cycle transfers the most significant byte of the long word on D23–D16. The second bus cycle transfers a word on D31–D16, and the last bus cycle transfers the least significant byte of the original long word on D31–D24. To prevent overwriting those bytes that are not used in these transfers, a unique byte data strobe must be generated for each byte when using devices with 16- and 32-bit port widths.

For noncachable read cycles and all write cycles, the required active bytes of the data bus for any given bus transfer are a function of the SIZ1, SIZ0 and A1, A0 outputs (see Table 9-1). Individual strobes or select signals can be generated by decoding these four signals for every bus cycle. Devices residing on 8-bit ports can utilize DS or AS since there is only one valid byte for any transfer.

9.6 ACCESS TIME CALCULATIONS

The timing paths that are critical in any memory interface are illustrated and defined in Figure 9-9.

The type of device that is interfaced to the MC68020/EC020 determines exactly which of the paths is most critical. The address-to-data paths are typically the critical paths for static devices since there is no penalty for initiating a cycle to these devices and later validating that access with the appropriate bus control signal. Conversely, the address-strobe-to-data-valid path is often most critical for dynamic devices since the cycle must be validated before an access can be initiated. For devices that signal termination of a bus cycle before data is validated (e.g., error detection and correction hardware and some external caches), to improve performance, the critical path may be from the address or strobes to the assertion of BERR (or BERR and HALT). Finally, the address-valid-to-DSACK1/DSACK0-asserted path is most critical for very fast devices and external caches, since the time available between the address becoming valid and the DSACK1/DSACK0 assertion to terminate the bus cycle is minimal. Table 9-4 provides the equations required to calculate the various memory access times assuming a 50-percent duty cycle clock.



*For the MC68EC020, A23–A0.

Parameter	Description	System	Equation
a	Address Valid to DSACK1/DSACK0 Asserted	t_{AVDL}	9-3
b	AS Asserted to DSACK1/DSACK0 Asserted	t_{SADL}	9-4
c	Address Valid to BERR/HALT Asserted	t_{AVBHL}	9-5
d	AS Asserted to BERR/HALT Asserted	t_{SABHL}	9-6
e	Address Valid to Data Valid	t_{AVDV}	9-7
f	AS Asserted to Data Valid	t_{SADV}	9-8

Another way to optimize the CPU-to-memory access times in a system is to use a clock frequency less than the rated maximum of the specific MC68020/EC020 device. Table 9-5 provides calculated t_{AVDV} (see Equation 9-7 of Table 9-4) results for a 16 MHz MC68020/EC020 and a 25 MHz MC68020/EC020 operating at various clock frequencies. If the system uses other clock frequencies, the above equations can be used to calculate the exact access times.

Table 9-5. Calculated t_{AVDV} Values for Operation at Frequencies Less Than or Equal to the CPU Maximum Frequency Rating

Equation 9-7 t_{AVDV}		16-MHz MC68020/EC020		25-MHz MC68020/EC020		
Clocks Per (N) and Type Bus Cycle	Wait States	Clock at 12.5 MHz	Clock at 16.67 MHz	Clock at 16.67 MHz	Clock at 20 MHz	Clock at 25 MHz
3 Clock Asynchronous	0	181	121	131	101	71
4 Clock Asynchronous	1	261	181	191	151	111
5 Clock Asynchronous	2	341	241	251	201	151
6 Clock Asynchronous	3	421	301	311	251	191

9.7 MODULE SUPPORT

The MC68020/EC020 includes support for modules with the CALLM and RTM instructions. The CALLM instruction references a module descriptor. This descriptor contains control information for entry into the called module. The CALLM instruction creates a module stack frame and stores the current module state in that frame and loads a new module state from the referenced descriptor. The RTM instruction recovers the previous module state from the stack frame and returns to the calling module.

The module interface facilitates finer resolution of access control by external hardware. Although the MC68020/EC020 does not interpret the access control information, it communicates with external hardware when the access control is to be changed and relies on the external hardware to verify that the changes are legal.

9.7.1 Module Descriptor

Figure 9-10 illustrates the format of the module descriptor. The first long word contains control information used during execution of the CALLM instruction. The remaining locations contain data that can be loaded into processor registers by the CALLM instruction.

SECTION 10
ELECTRICAL CHARACTERISTICS

This section provides the thermal characteristics and electrical specifications for the MC68020/EC020. Note that the thermal and DC electrical characteristics are listed separately for the MC68020 and the MC68EC020. All other data applies to both the MC68020 and the MC68EC020 unless otherwise noted.

10.1 MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V_{CC}	–0.3 to +7.0	V
Input Voltage	V_{in}	–0.5 to +7.0	V
Operating Temperature Range Minimum Ambient Temperature	T_A	0	°C
Maximum Ambient Temperature PGA, PPGA, PQFP	T_A	70	°C
Maximum Junction Temperature CQFP	T_J	110	°C
Storage Temperature Range	T_{stg}	–55 to 150	°C

The device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, normal precautions should be taken to avoid application of voltages higher than maximum-rated voltages to these high-impedance circuits. Tying unused inputs to the appropriate logic voltage level (e.g., either GND or V_{CC}) enhances reliability of operation.

10.2 THERMAL CONSIDERATIONS

The average chip-junction temperature, T_J, in °C can be obtained from:

$$T_J = T_A + (P_D \cdot \theta_{JA}) \quad (10\text{-}1)$$

where:

T_A = Ambient Temperature, °C
θ_{JA} = Package Thermal Resistance, Junction-to-Ambient, °C/W
P_D = $P_{INT} + P_{I/O}$
P_{INT} = $I_{CC} \times V_{CC}$, Watts—Chip Internal Power
$P_{I/O}$ = Power Dissipation on Input and Output Pins—User Determined

For most applications, $P_{I/O} < P_{INT}$ and can be neglected.

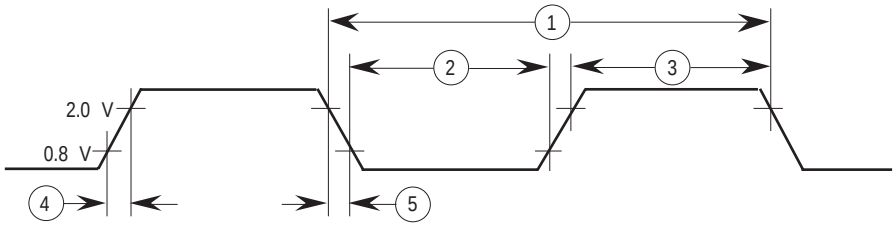
An approximate relationship between P_D and T_J (if $P_{I/O}$ is neglected) is:

$$P_D = K \div (T_J + 273°C) \quad (10\text{-}2)$$

AC ELECTRICAL CHARACTERISTICS—CLOCK INPUT (see Figure 10-2)

Num.	Characteristic	16.67 MHz		20 MHz		25 MHz*		33.33 MHz		Unit
		Min	Max	Min	Max	Min	Max	Min	Max	
	Frequency of Operation	8	16.67	12.5	20	12.5	25	12.5	33.33	MHz
1	Cycle Time	60	125	50	80	40	80	30	80	ns
2,3	Clock Pulse Width (Measured from 1.5 V to 1.5 V)	24	95	20	54	19	61	14	66	ns
4,5	Clock Rise and Fall Times	—	5	—	5	—	4	—	3	ns

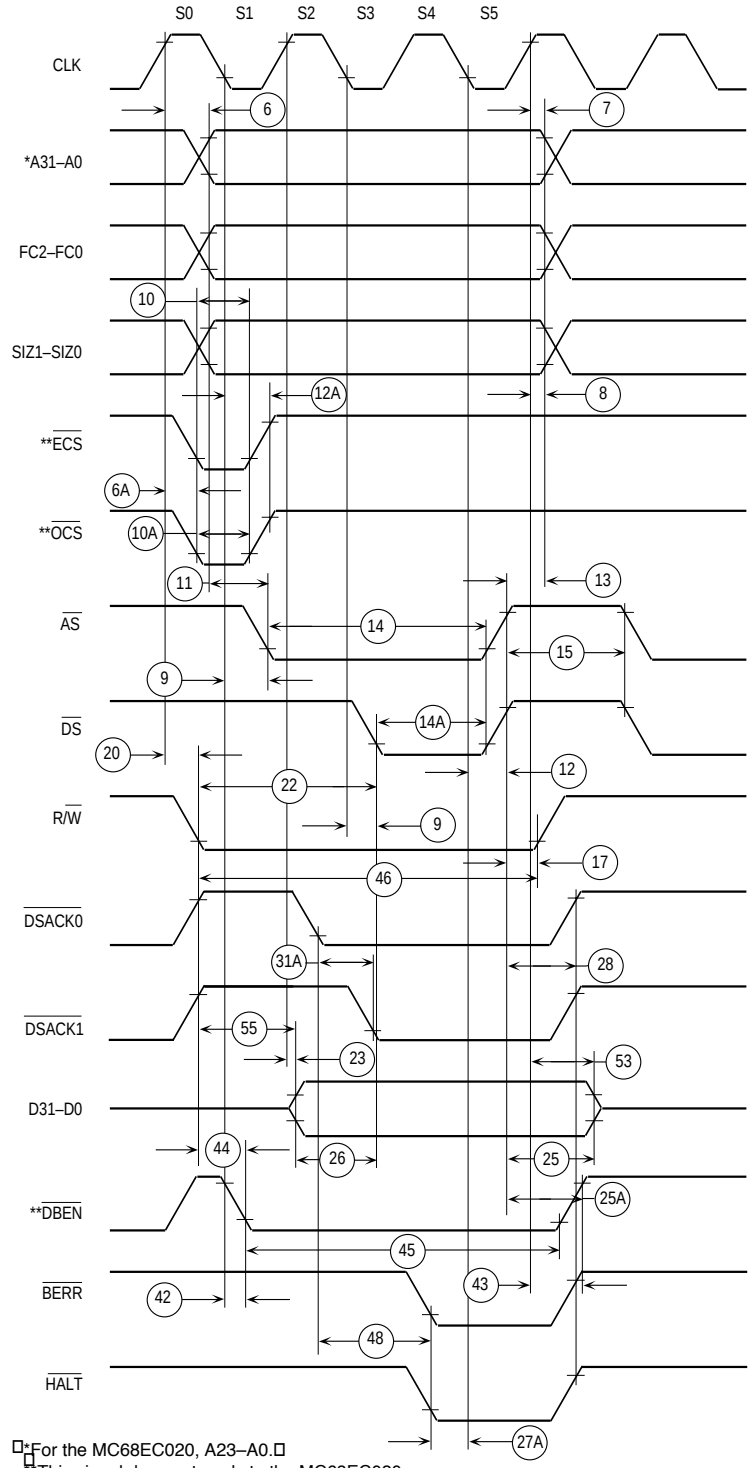
*These specifications represent an improvement over previously published specifications for the 25-MHz MC68020 and are valid only for products bearing date codes of 8827 and later.



NOTE: Timing measurements are referenced to and from a low voltage of .08 V and a high voltage of 2.0 V, unless othervise noted. The voltage swing through this r should start outside and pass through the range such that the rise or fall between 0.8 V and 2.0 V.

FIGURE 10-2
MC68020UM

Figure 10-2. Clock Input Timing Diagram



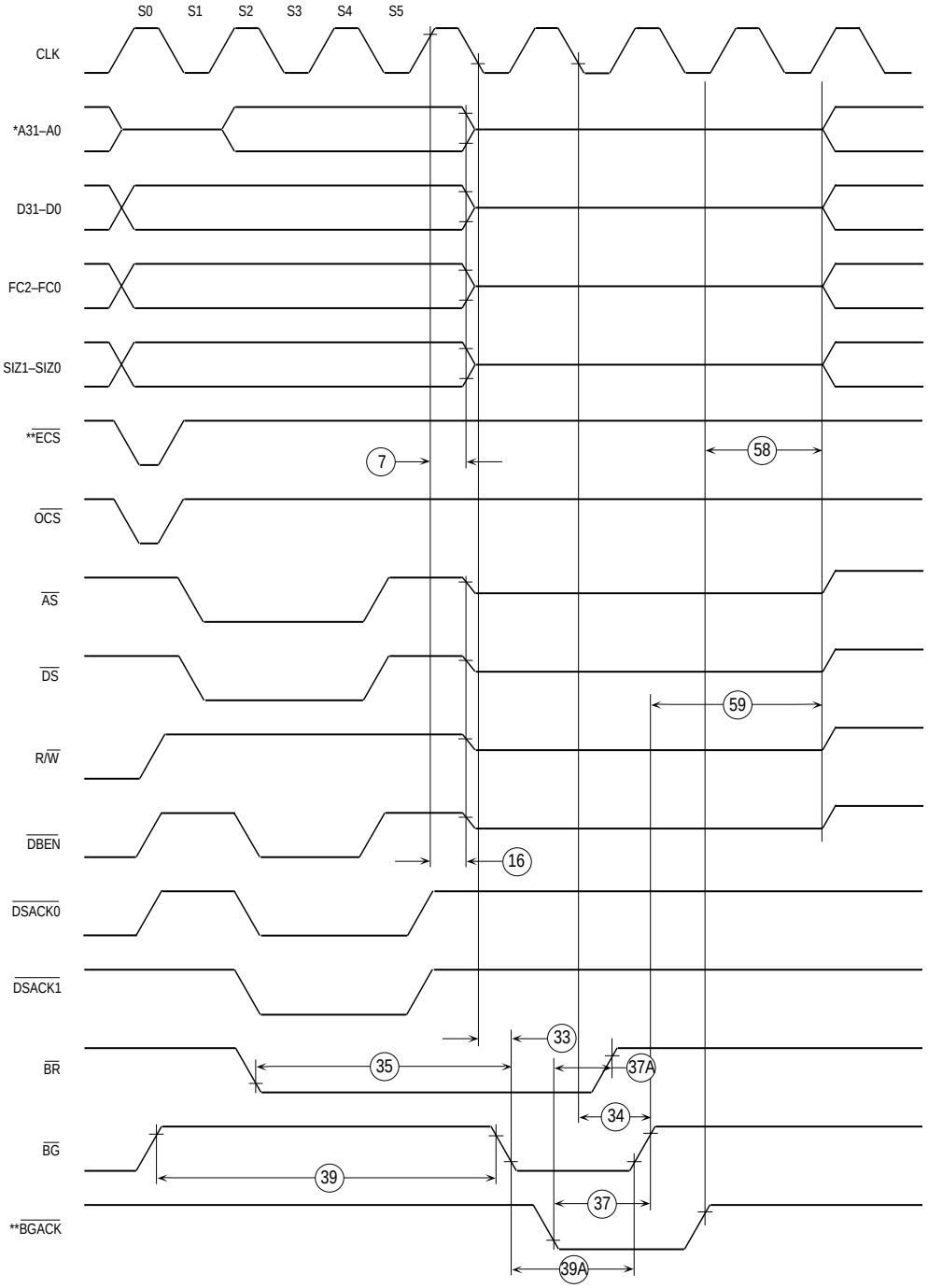
*For the MC68EC020, A23–A0.

**This signal does not apply to the MC68EC020.

NOTE: Timing measurements are referenced to and from a low voltage of 0.8 V and a high voltage of 2.0 V, unless otherwise noted. The voltage swing through this range should start outside and pass through the range such that the rise or fall will be linear between 0.8 V and 2.0 V.

FIGURE 10-4
MC68020UM

Figure 10-4. Write Cycle Timing Diagram



*For the MC68EC020, A23–A0.

**This signal does not apply to the MC68EC020.

NOTE: Timing measurements are referenced to and from a low voltage of 0.8 V and a high voltage of 2.0 V, unless otherwise noted. The voltage swing through this range should start outside and pass through the range such that the rise or fall will be linear between 0.8 V and 2.0 V.

FIGURE 10-5
MC68020UM

Figure 10-5. Bus Arbitration Timing Diagram