

SECTION 5 BUS OPERATION

This section provides a functional description of the bus, the signals that control it, and the bus cycles provided for data transfer operations. It also describes the error and halt conditions, bus arbitration, and reset operation. Operation of the bus is the same whether the processor or an external device is the bus master; the names and descriptions of bus cycles are from the point of view of the bus master. For exact timing specifications, refer to **Section 10 Electrical Characteristics**.

The MC68020/EC020 architecture supports byte, word, and long-word operands, allowing access to 8-, 16-, and 32-bit data ports through the use of asynchronous cycles controlled by the $\overline{\text{DSACK1}}$ and $\overline{\text{DSACK0}}$ input signals.

The MC68020/EC020 allows byte, word, and long-word operands to be located in memory on any byte boundary. For a misaligned transfer, more than one bus cycle may be required to complete the transfer, regardless of port size. For a port less than 32 bits wide, multiple bus cycles may be required for an operand transfer due to either misalignment or a port width smaller than the operand size. Instruction words and their associated extension words must be aligned on word boundaries. The user should be aware that misalignment of word or long-word operands can cause the MC68020/EC020 to perform multiple bus cycles for the operand transfer; therefore, processor performance is optimized if word and long-word memory operands are aligned on word or long-word boundaries, respectively.

5.1 BUS TRANSFER SIGNALS

The bus transfers information between the MC68020/EC020 and an external memory, coprocessor, or peripheral device. External devices can accept or provide 8 bits, 16 bits, or 32 bits in parallel and must follow the handshake protocol described in this section. The maximum number of bits accepted or provided during a bus transfer is defined as the port width. The MC68020/EC020 contains an address bus that specifies the address for the transfer and a data bus that transfers the data. Control signals indicate the beginning of the cycle, the address space and size of the transfer, and the type of cycle. The selected device then controls the length of the cycle with the signal(s) used to terminate the cycle. Strobe signals, one for the address bus and another for the data bus, indicate the validity of the address and provide timing information for the data.

The bus operates in an asynchronous mode for any port width. The bus and control input signals are internally synchronized to the MC68020/EC020 clock, introducing a delay. This delay is the time period required for the MC68020/EC020 to sample an input signal, synchronize the input to the internal clocks of the processor, and determine whether the

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.

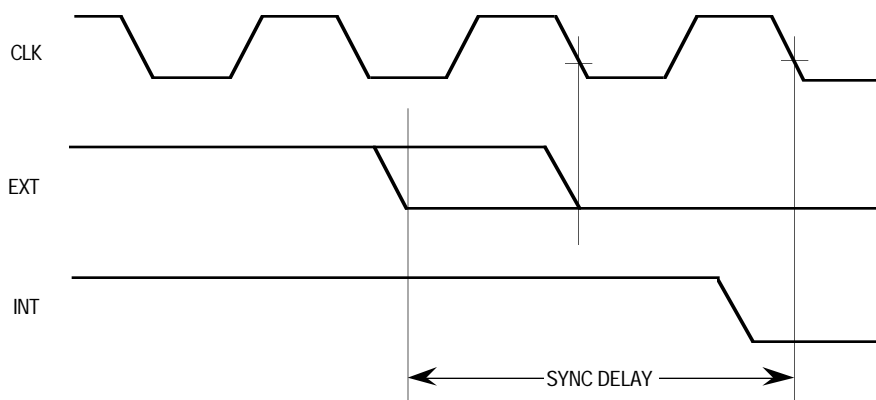


Figure 5-1. Relationship between External and Internal Signals

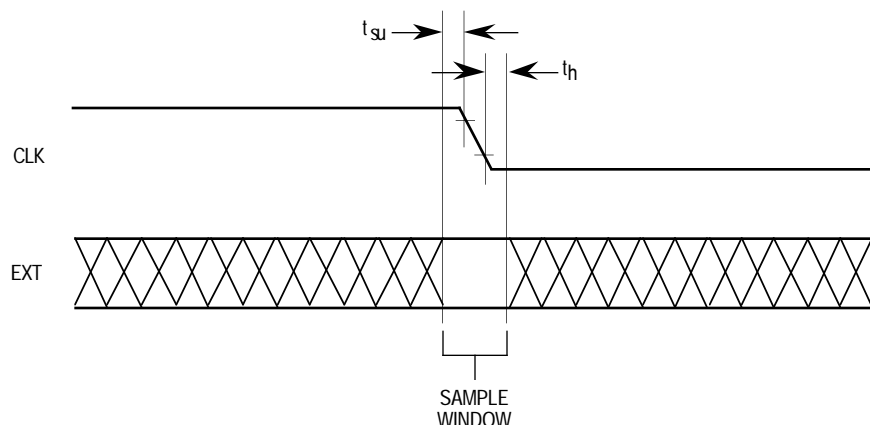


Figure 5-2. Input Sample Window

5.1.1 Bus Control Signals

The MC68020/EC020 initiates a bus cycle by driving the A1–A0, SIZ1, SIZ0, FC2–FC0, and $\overline{R/\overline{W}}$ 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.

When initiating a bus cycle, the MC68020 asserts \overline{ECS} in addition to A1–A0, SIZ1, SIZ0, FC2–FC0, and R/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/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/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

a write cycle, all 32 bits of the data bus are driven, regardless of the port width or operand size. The processor places the data on the data bus one-half clock cycle after \overline{AS} is asserted in a write cycle.

5.1.5 Data Strobe

\overline{DS} is a timing signal that applies to the data bus. For a read cycle, the processor asserts \overline{DS} to signal the external device to place data on the bus. \overline{DS} is asserted at the same time as \overline{AS} during a read cycle. For a write cycle, \overline{DS} notifies the external device that the data to be written is valid. The processor asserts \overline{DS} one full clock cycle after the assertion of \overline{AS} during a write cycle.

5.1.6 Data Buffer Enable

The MC68020 \overline{DBEN} signal is used to enable external data buffers while data is present on the data bus. During a read operation, \overline{DBEN} is asserted one clock cycle after the beginning of the bus cycle and is negated as \overline{DS} is negated. In a write operation, \overline{DBEN} is asserted at the time \overline{AS} is asserted and is held active for the duration of the cycle. Note that \overline{DBEN} is implemented in the MC68020 and is not implemented in the MC68EC020.

5.1.7 Bus Cycle Termination Signals

During bus cycles, external devices assert $\overline{DSACK1}/\overline{DSACK0}$ as part of the bus protocol. During a read cycle, $\overline{DSACK1}/\overline{DSACK0}$ assertion signals the processor to terminate the bus cycle and to latch the data. During a write cycle, the assertion of $\overline{DSACK1}/\overline{DSACK0}$ indicates that the external device has successfully stored the data and that the cycle may terminate. $\overline{DSACK1}/\overline{DSACK0}$ also indicate to the processor the size of the port for the bus cycle just completed, as shown in Table 5-1. Refer to **5.3.1 Read Cycle** for timing relationships of $\overline{DSACK1}/\overline{DSACK0}$.

The \overline{BERR} signal is also a bus cycle termination indicator and can be used in the absence of $\overline{DSACK1}/\overline{DSACK0}$ to indicate a bus error condition. It can also be asserted in conjunction with $\overline{DSACK1}/\overline{DSACK0}$ to indicate a bus error condition, provided it meets the appropriate timing described in this section and in **Section 10 Electrical Characteristics**. Additionally, the \overline{BERR} and \overline{HALT} signals can be asserted together to indicate a retry termination. Again, the \overline{BERR} and \overline{HALT} signals can be simultaneously asserted in lieu of, or in conjunction with, the $\overline{DSACK1}/\overline{DSACK0}$ signals.

Finally, the \overline{AVEC} signal can be used to terminate interrupt acknowledge cycles, indicating that the MC68020/EC020 should generate a vector number to locate an interrupt handler routine. \overline{AVEC} is ignored during all other bus cycles.

5.2 DATA TRANSFER MECHANISM

The MC68020/EC020 architecture supports byte, word, and long-word operands allowing access to 8-, 16-, and 32-bit data ports through the use of asynchronous cycles controlled by $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$. Byte, word, and long-word operands can be located on any byte boundary, but misaligned transfers may require additional bus cycles, regardless of port size.

5.2.1 Dynamic Bus Sizing

The MC68020/EC020 dynamically interprets the port size of the addressed device during each bus cycle, allowing operand transfers to or from 8-, 16-, and 32-bit ports. During an operand transfer cycle, the slave device signals its port size (byte, word, or long word) and indicates completion of the bus cycle to the processor with the $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$ signals. Refer to Table 5-1 for $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$ encodings and assertion results.

Table 5-1. $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$ Encodings and Results

| $\overline{\text{DSACK1}}$ | $\overline{\text{DSACK0}}$ | Result |
|----------------------------|----------------------------|--|
| Negated | Negated | Insert Wait States in Current Bus Cycle |
| Negated | Asserted | Complete Cycle—Data Bus Port Size is 8 Bits |
| Asserted | Negated | Complete Cycle—Data Bus Port Size is 16 Bits |
| Asserted | Asserted | Complete Cycle—Data Bus Port Size is 32 Bits |

For example, if the processor is executing an instruction that reads a long-word operand from a long-word-aligned address, it attempts to read 32 bits during the first bus cycle. (Refer to **5.2.2 Misaligned Operands** for the case of a word or byte address.) If the port responds that it is 32 bits wide, the MC68020/EC020 latches all 32 bits of data and continues with the next operation. If the port responds that it is 16 bits wide, the MC68020/EC020 latches the 16 bits of valid data and runs another bus cycle to obtain the other 16 bits. The operation for an 8-bit port is similar, but requires four read cycles. The addressed device uses the $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$ signals to indicate the port width. For instance, a 32-bit device always returns $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$ for a 32-bit port, regardless of whether the bus cycle is a byte, word, or long-word operation.

Dynamic bus sizing requires that the portion of the data bus used for a transfer to or from a particular port size be fixed. A 32-bit port must reside on D31–D0, a 16-bit port must reside on D31–D16, and an 8-bit port must reside on D31–D24. This requirement minimizes the number of bus cycles needed to transfer data to 8- and 16-bit ports and ensures that the MC68020/EC020 correctly transfers valid data. The MC68020/EC020 always attempts to transfer the maximum amount of data on all bus cycles; for a long-word operation, it always assumes that the port is 32 bits wide when beginning the bus cycle.

The bytes of operands are designated as shown in Figure 5-3. The most significant byte of a long-word operand is OP0; the least significant byte is OP3. The two bytes of a word-length operand are OP2 (most significant) and OP3. The single byte of a byte-length operand is OP3. These designations are used in the figures and descriptions that follow.

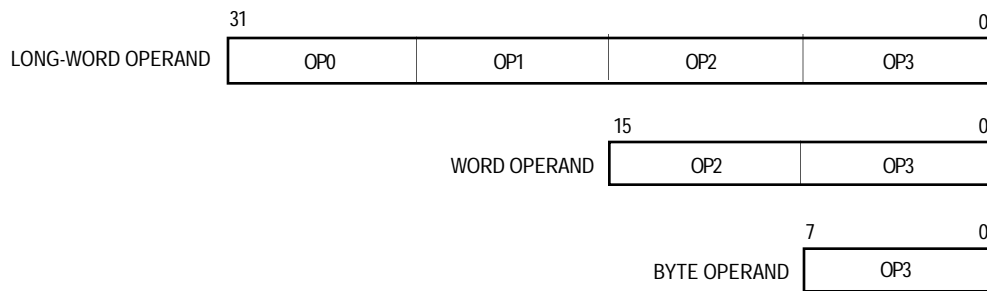


Figure 5-3. Internal Operand Representation

Figure 5-4 shows the required organization of data ports on the MC68020/EC020 bus for 8-, 16-, and 32-bit devices. The four bytes shown in Figure 5-4 are connected through the internal data bus and data multiplexer to the external data bus. This path is the means through which the MC68020/EC020 supports dynamic bus sizing and operand misalignment. Refer to **5.2.2 Misaligned Operands** for the definition of misaligned operand. The data multiplexer establishes the necessary connections for different combinations of address and data sizes.

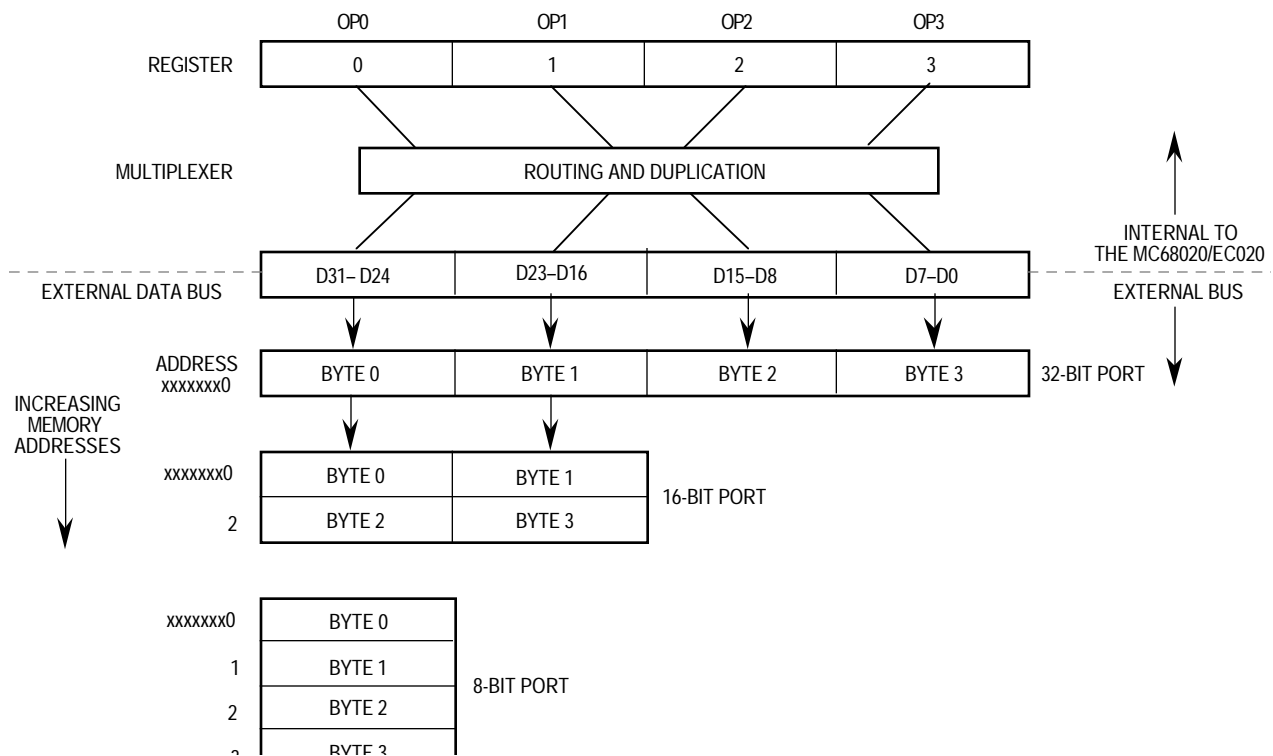


Figure 5-4. MC68020/EC020 Interface to Various Port Sizes

The multiplexer takes the four bytes of the 32-bit bus and routes them to their required positions. For example, OP0 can be routed to D31–D24, as would be the normal case, or it can be routed to any other byte position to support a misaligned transfer. The same is true for any of the operand bytes. The positioning of bytes is determined by the SIZ1, SIZ0, A1, and A0 outputs.

The SIZ1 and SIZ0 outputs indicate the remaining number of bytes to be transferred during the current bus cycle, as listed in Table 5-2.

Table 5-2. SIZ1, SIZ0 Signal Encoding

| SIZ1 | SIZ0 | Size |
|----------|----------|-----------|
| Negated | Asserted | Byte |
| Asserted | Negated | Word |
| Asserted | Asserted | 3 Bytes |
| Negated | Negated | Long Word |

The number of bytes transferred during a write or read bus cycle is equal to or less than the size indicated by the SIZ1 and SIZ0 outputs, depending on port width and operand alignment. For example, during the first bus cycle of a long-word transfer to a word port, the SIZ1 and SIZ0 outputs indicate that four bytes are to be transferred, although only two bytes are moved on that bus cycle.

A1–A0 also affect operation of the data multiplexer. During an operand transfer, A31–A2 (for the MC68020) or A23–A2 (for the MC68EC020) indicate the long-word base address of that portion of the operand to be accessed; A1 and A0 indicate the byte offset from the base. Table 5-3 lists the encodings of A1 and A0 and the corresponding byte offsets from the long-word base.

Table 5-3. Address Offset Encodings

| A1 | A0 | Offset |
|----------|----------|----------|
| Negated | Negated | +0 Bytes |
| Negated | Asserted | +1 Byte |
| Asserted | Negated | +2 Bytes |
| Asserted | Asserted | +3 Bytes |

Table 5-4 lists the bytes required on the data bus for read cycles. The entries shown as OP3, OP2, OP1, and OP0 are portions of the requested operand that are read or written during that bus cycle and are defined by SIZ1, SIZ0, A1, and A0 for the bus cycle.

Table 5-4. Data Bus Requirements for Read Cycles

| Transfer Size | Size | | Address | | Long-Word Port External Data Bytes Required | | | | Word Port External Data Bytes Required | | Byte Port External Data Bytes Required |
|---------------|------|------|---------|----|---|---------|--------|-------|--|---------|--|
| | SIZ1 | SIZ0 | A1 | A0 | D31–D24 | D23–D16 | D15–D8 | D7–D0 | D31–D24 | D23–D16 | D31–D24 |
| Byte | 0 | 1 | 0 | 0 | OP3 | | | | OP3 | | OP3 |
| | 0 | 1 | 0 | 1 | | OP3 | | | | OP3 | OP3 |
| | 0 | 1 | 1 | 0 | | | OP3 | | OP3 | | OP3 |
| | 0 | 1 | 1 | 1 | | | | OP3 | | OP3 | OP3 |
| Word | 1 | 0 | 0 | 0 | OP2 | OP3 | | | OP2 | OP3 | OP2 |
| | 1 | 0 | 0 | 1 | | OP2 | OP3 | | | OP2 | OP2 |
| | 1 | 0 | 1 | 0 | | | OP2 | OP3 | OP2 | OP3 | OP2 |
| | 1 | 0 | 1 | 1 | | | | OP2 | | OP2 | OP2 |
| 3 Bytes | 1 | 1 | 0 | 0 | OP1 | OP2 | OP3 | | OP1 | OP2 | OP1 |
| | 1 | 1 | 0 | 1 | | OP1 | OP2 | OP3 | | OP1 | OP1 |
| | 1 | 1 | 1 | 0 | | | OP1 | OP2 | OP1 | OP2 | OP1 |
| | 1 | 1 | 1 | 1 | | | | OP1 | | OP1 | OP1 |
| Long Word | 0 | 0 | 0 | 0 | OP0 | OP1 | OP2 | OP3 | OP0 | OP1 | OP0 |
| | 0 | 0 | 0 | 1 | | OP0 | OP1 | OP2 | | OP0 | OP0 |
| | 0 | 0 | 1 | 0 | | | OP0 | OP1 | OP0 | OP1 | OP0 |
| | 0 | 0 | 1 | 1 | | | | OP0 | | OP0 | OP0 |

Table 5-5 lists the combinations of SIZ1, SIZ0, A1, and A0 and the corresponding pattern of the data transfer for write cycles from the internal multiplexer of the MC68020/EC020 to the external data bus.

Table 5-5. MC68020/EC020 Internal to External Data Bus Multiplexer—Write Cycles

| Transfer Size | Size | | Address | | External Data Bus Connection | | | |
|---------------|------|------|---------|----|------------------------------|---------|--------|-------|
| | SIZ1 | SIZ0 | A1 | A0 | D31–D24 | D23–D16 | D15–D8 | D7–D0 |
| Byte | 0 | 1 | x | x | OP3 | OP3 | OP3 | OP3 |
| Word | 1 | 0 | x | 0 | OP2 | OP3 | OP2 | OP3 |
| | 1 | 0 | x | 1 | OP2 | OP2 | OP3 | OP2 |
| 3 Bytes | 1 | 1 | 0 | 0 | OP1 | OP2 | OP3 | OP0* |
| | 1 | 1 | 0 | 1 | OP1 | OP1 | OP2 | OP3 |
| | 1 | 1 | 1 | 0 | OP1 | OP2 | OP1 | OP2 |
| | 1 | 1 | 1 | 1 | OP1 | OP1 | OP2* | OP1 |
| Long Word | 0 | 0 | 0 | 0 | OP0 | OP1 | OP2 | OP3 |
| | 0 | 0 | 0 | 1 | OP0 | OP0 | OP1 | OP2 |
| | 0 | 0 | 1 | 0 | OP0 | OP1 | OP0 | OP1 |
| | 0 | 0 | 1 | 1 | OP0 | OP0 | OP1* | OP0 |

*Due to the current implementation, this byte is output but never used.

x = Don't care

NOTE: The OP tables on the external data bus refer to a particular byte of the operand that is written on that section of the data bus.

Figure 5-5 shows the transfer (write) of a long-word operand to a word port. In the first bus cycle, the MC68020/EC020 places the four operand bytes on the external bus. Since the address is long-word aligned in this example, the multiplexer follows the pattern in the entry of Table 5-5 corresponding to SIZ0, SIZ1, A0, A1 = 0000. The port latches the data on D31–D16, asserts $\overline{\text{DSACK1}}$ ($\overline{\text{DSACK0}}$ remains negated), and the processor terminates the bus cycle. It then starts a new bus cycle with SIZ1, SIZ0, A1, A0 = 1010 to transfer the remaining 16 bits. SIZ1 and SIZ0 indicate that a word remains to be transferred; A1 and A0 indicate that the word corresponds to an offset of two from the base address. The multiplexer follows the pattern corresponding to this configuration of SIZ1, SIZ0, A1, and A0 and places the two least significant bytes of the long word on the word portion of the bus (D31–D16). The bus cycle transfers the remaining bytes to the word-sized port. Figure 5-6 shows the timing of the bus transfer signals for this operation.

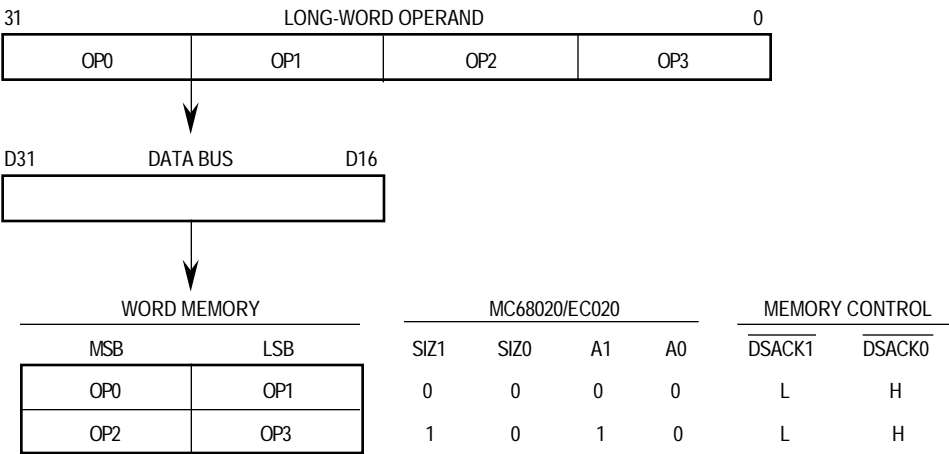
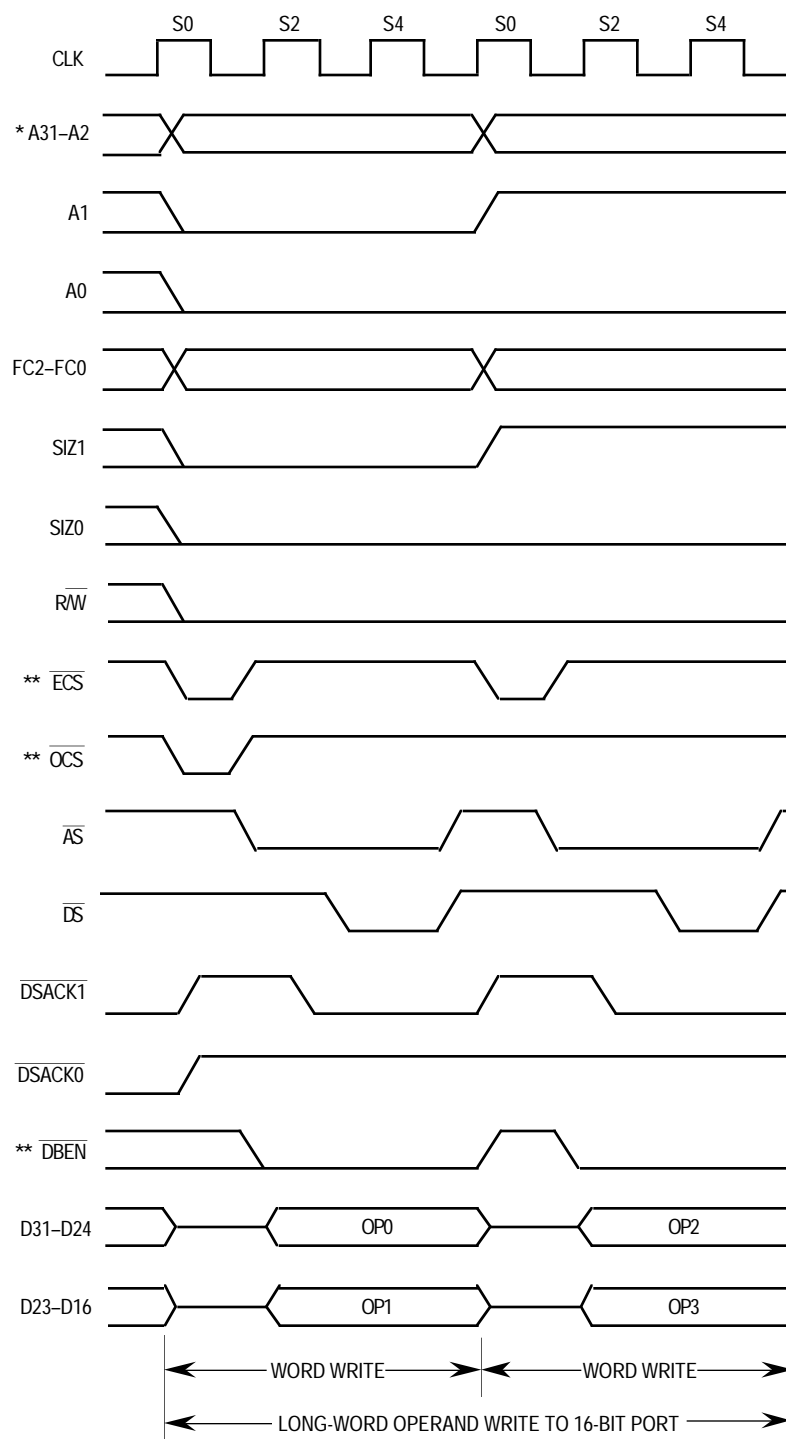


Figure 5-5. Long-Word Operand Write to Word Port Example



* For the MC68EC020, A23-A2.

** This signal does not apply to the MC68EC020.

Figure 5-6. Long-Word Operand Write to Word Port Timing

Figure 5-7 shows a word write to an 8-bit bus port. Like the preceding example, this example requires two bus cycles. Each bus cycle transfers a single byte. SIZ1 and SIZ0 for the first cycle specify two bytes; for the second cycle, one byte. Figure 5-8 shows the associated bus transfer signal timing.

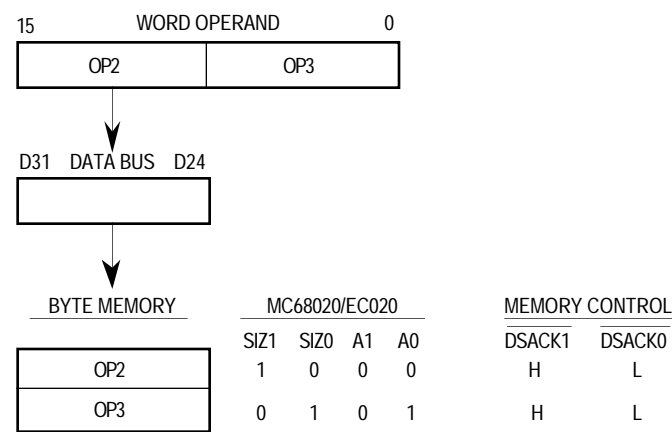
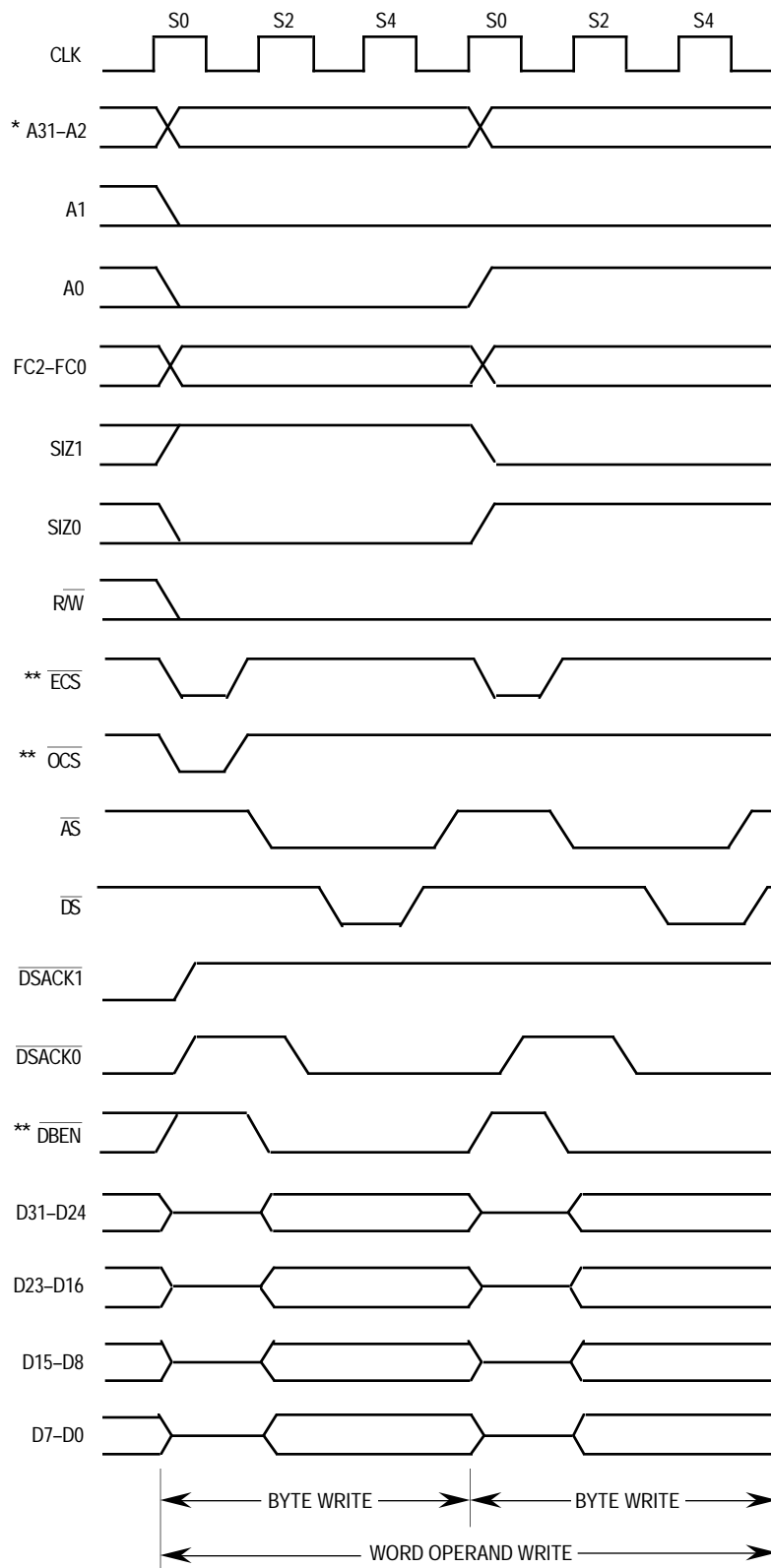


Figure 5-7. Word Operand Write to Byte Port Example



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

Figure 5-8. Word Operand Write to Byte Port Timing

5.2.2 Misaligned Operands

Since operands may reside at any byte boundary, they may be misaligned. A byte operand is properly aligned at any address; a word operand is misaligned at an odd address; a long word is misaligned at an address that is not evenly divisible by four. The MC68000, MC68008, and MC68010 implementations allow long-word transfers on odd-word boundaries but force exceptions if word or long-word operand transfers are attempted at odd-byte addresses. Although the MC68020/EC020 does not enforce any alignment restrictions for data operands (including PC relative data addresses), some performance degradation occurs when additional bus cycles are required for long-word or word operands that are misaligned. For maximum performance, data items should be aligned on their natural boundaries. All instruction words and extension words must reside on word boundaries. Attempting to prefetch an instruction word at an odd address causes an address error exception.

Figure 5-9 shows the transfer (write) of a long-word operand to an odd address in word-organized memory, which requires three bus cycles. For the first cycle, SIZ1 and SIZ0 specify a long-word transfer, and A2–A0 = 001. Since the port width is 16 bits, only the first byte of the long word is transferred. The slave device latches the byte and acknowledges the data transfer, indicating that the port is 16 bits wide. When the processor starts the second cycle, SIZ1 and SIZ0 specify that three bytes remain to be transferred with A2–A0 = 010. The next two bytes are transferred during this cycle. The processor then initiates the third cycle, with SIZ1 and SIZ0 indicating one byte remaining to be transferred with A2–A0 = 100. The port latches the final byte, and the operation is complete. Figure 5-10 shows the associated bus transfer signal timing. Figure 5-11 shows the equivalent operation for a data read cycle.

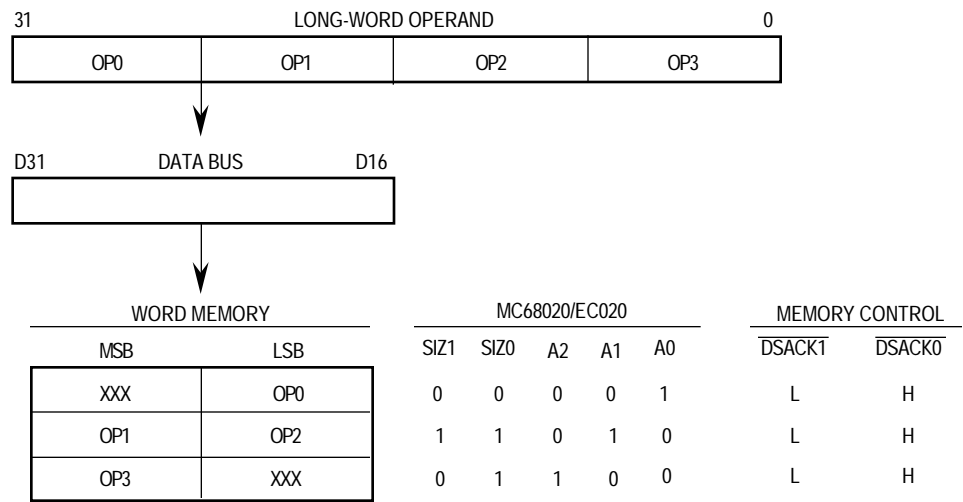
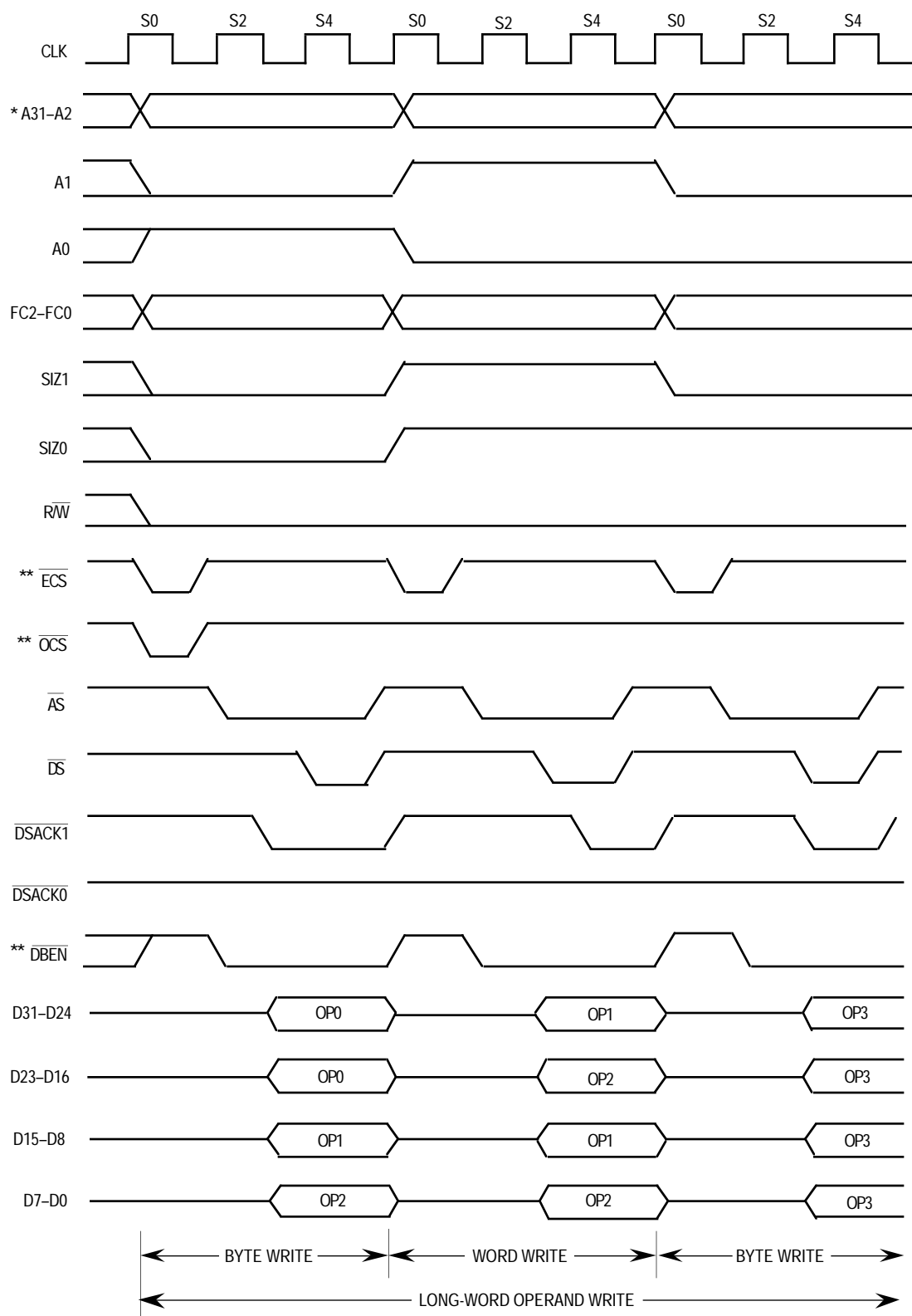


Figure 5-9. Misaligned Long-Word Operand Write to Word Port Example



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

Figure 5-10. Misaligned Long-Word Operand Write to Word Port Timing

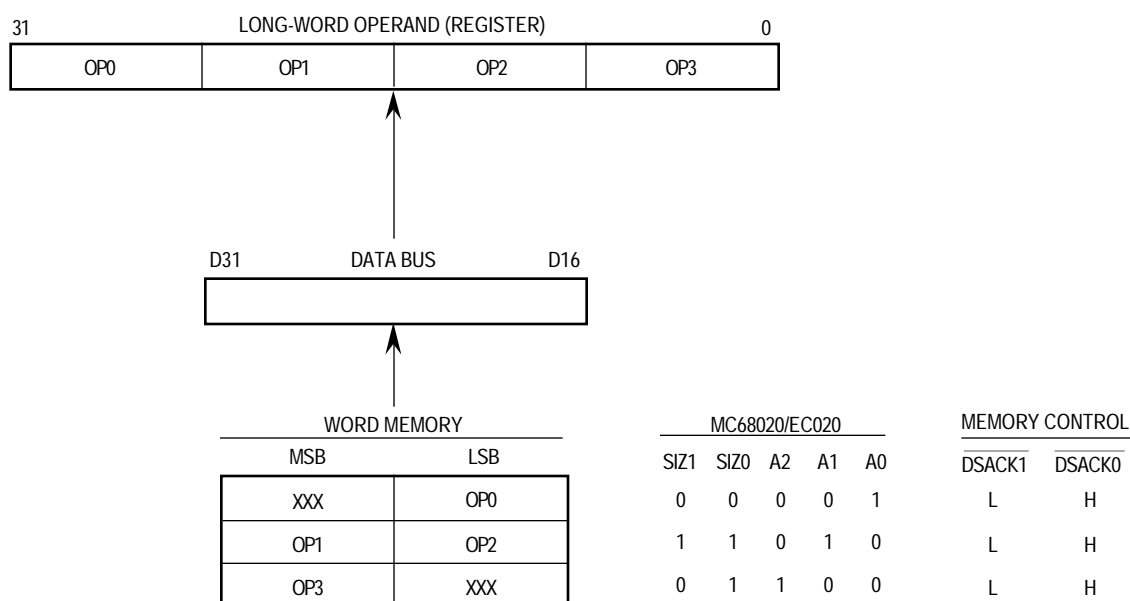


Figure 5-11. Misaligned Long-Word Operand Read from Word Port Example

Figures 5-12 and 5-13 show a word transfer (write) to an odd address in word-organized memory. This example is similar to the one shown in Figures 5-9 and 5-10 except that the operand is word sized and the transfer requires only two bus cycles. Figure 5-14 shows the equivalent operation for a data read cycle.

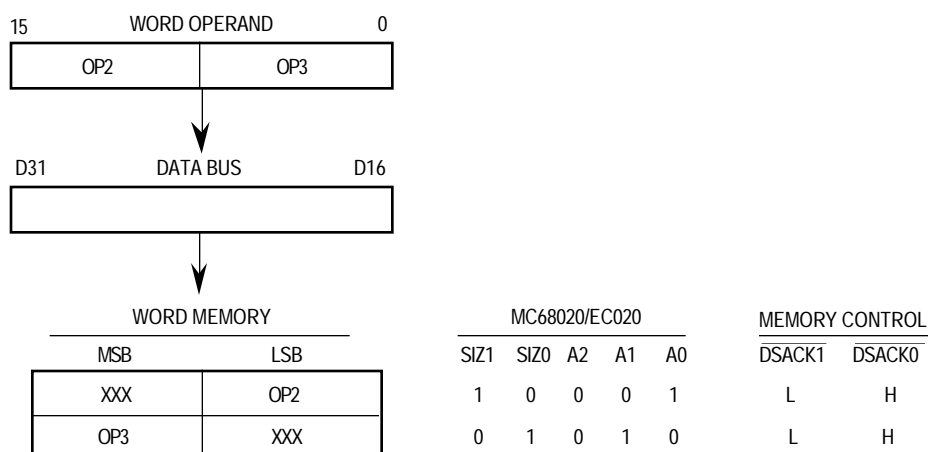
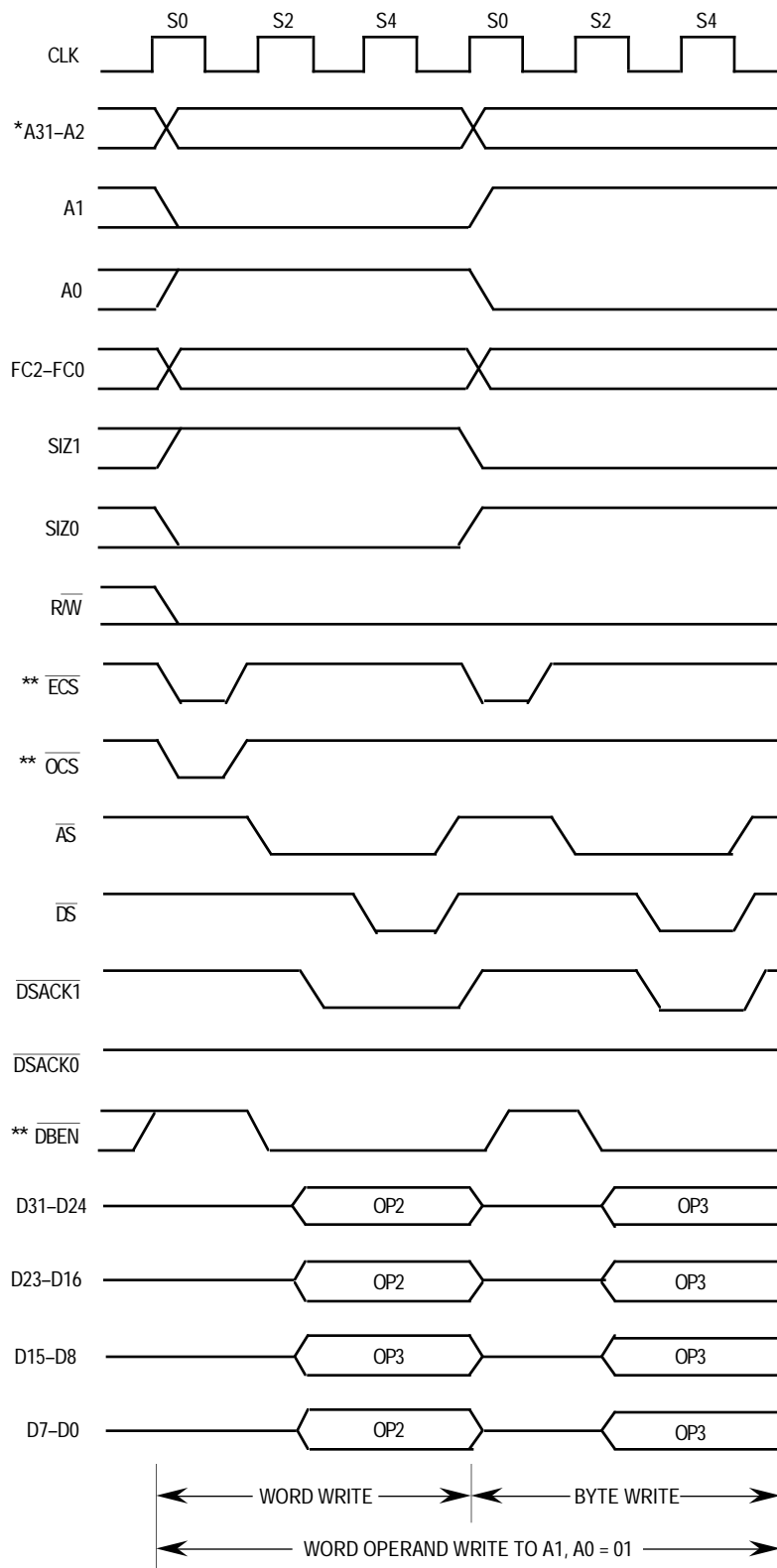


Figure 5-12. Misaligned Word Operand Write to Word Port Example



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

Figure 5-13. Misaligned Word Operand Write to Word Port Timing

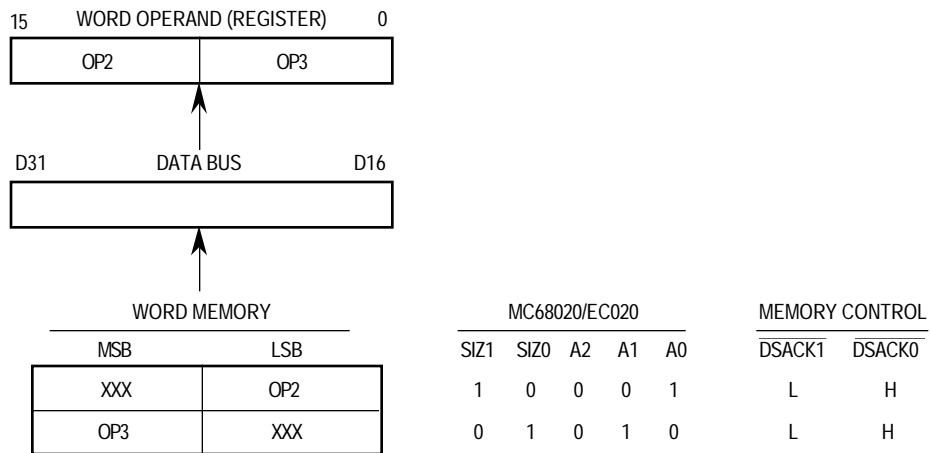


Figure 5-14. Misaligned Word Operand Read from Word Bus Example

Figures 5-15 and 5-16 show an example of a long-word transfer (write) to an odd address in long-word-organized memory. In this example, a long-word access is attempted beginning at the least significant byte of a long-word-organized memory. Only one byte can be transferred in the first bus cycle. The second bus cycle then consists of a three-byte access to a long-word boundary. Since the memory is long word organized, no further bus cycles are necessary. Figure 5-17 shows the equivalent operation for a data read cycle.

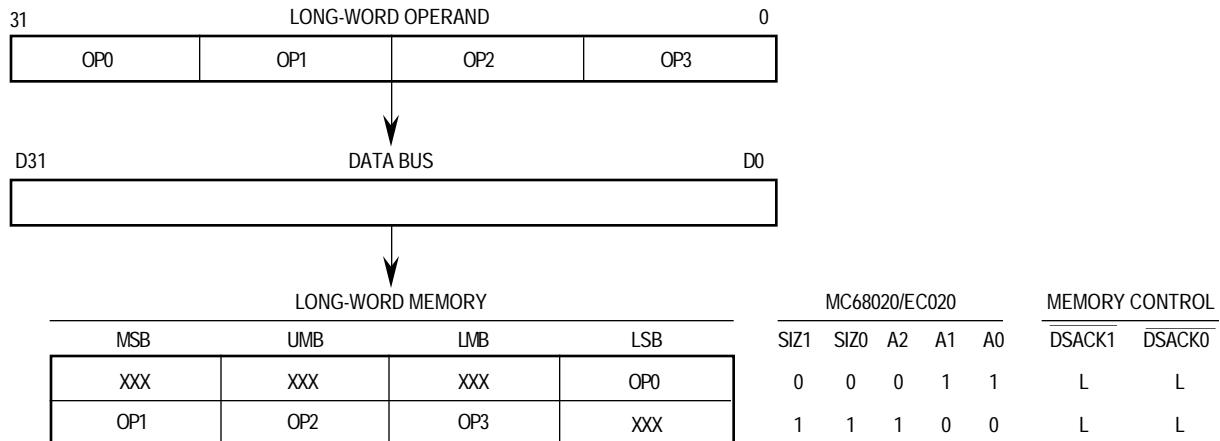
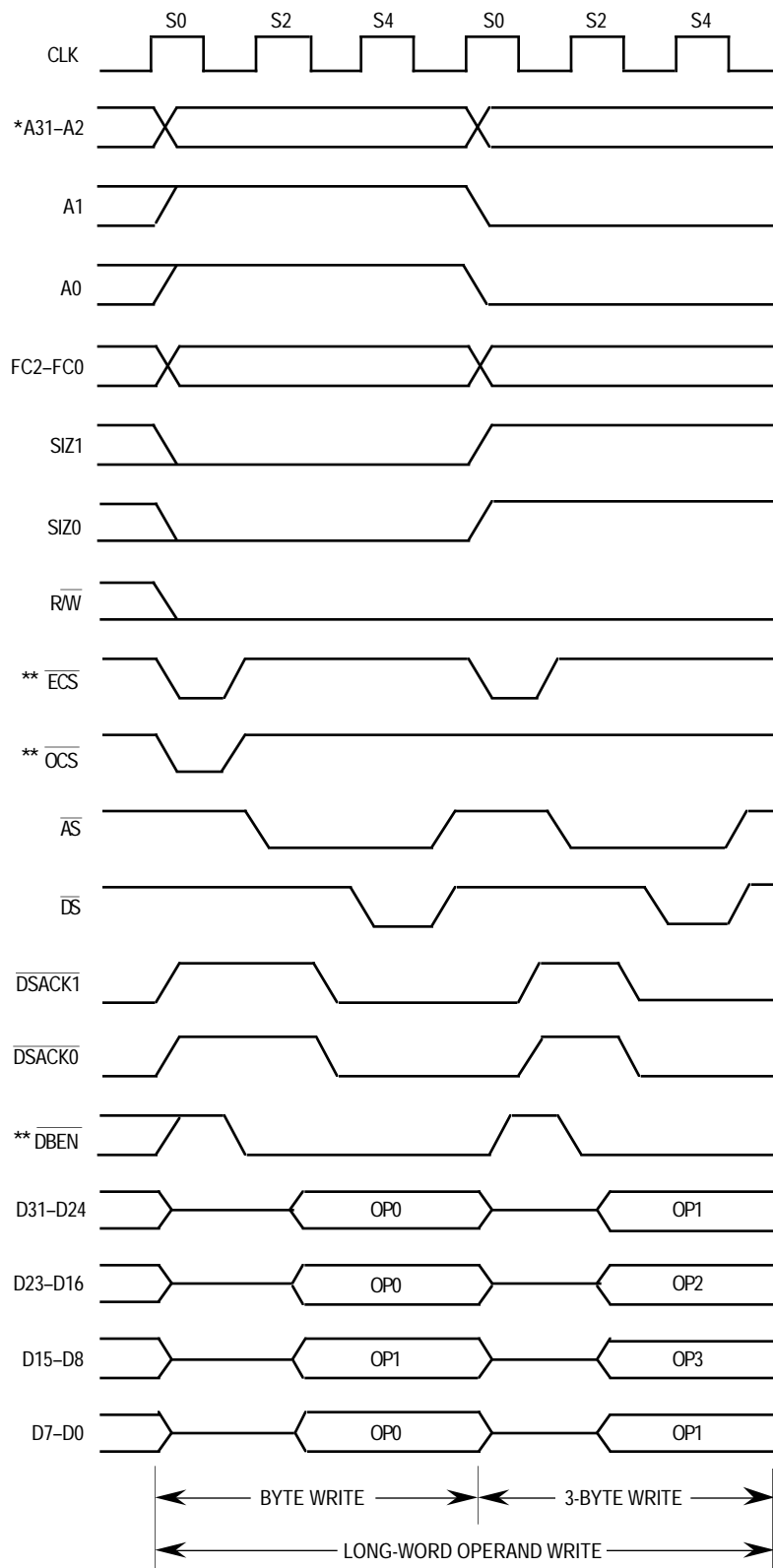


Figure 5-15. Misaligned Long-Word Operand Write to Long-Word Port Example



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

Figure 5-16. Misaligned Long-Word Operand Write to Long-Word Port Timing

Table 5-6 demonstrates that the processor always prefetches instructions by reading a long word from a long-word address ($A1, A0 = 00$), regardless of port size or alignment. When the required instruction begins at an odd-word boundary, the processor attempts to fetch the entire 32 bits and loads both words into the instruction cache, if possible, although the second one is the required word. Even if the instruction access is not cached, the entire 32 bits are latched into an internal cache holding register from which the two instructions words can subsequently be referenced. Refer to **Section 8 Instruction Execution Timing** for a complete description of the cache holding register and pipeline operation.

5.2.4 Address, Size, and Data Bus Relationships

The data transfer examples show how the MC68020/EC020 drives data onto or receives data from the correct byte sections of the data bus. Table 5-7 shows the combinations of the $SIZ1$, $SIZ0$, $A1$, and $A0$ signals that can be used to generate byte enable signals for each of the four sections of the data bus for read and write cycles if the addressed device requires them. The port size also affects the generation of these enable signals as shown in the table. The four columns on the right correspond to the four byte enable signals. Letters B, W, and L refer to port sizes: B for 8-bit ports, W for 16-bit ports, and L for 32-bit ports. The letters B, W, and L imply that the byte enable signal should be true for that port size. A dash (—) implies that the byte enable signal does not apply.

The MC68020/EC020 always drives all sections of the data bus because, at the beginning of a write cycle, the bus controller does not know the port size.

Table 5-7 reveals that the MC68020/EC020 transfers the number of bytes specified by $SIZ1$, $SIZ0$ to or from the specified address unless the operand is misaligned or unless the number of bytes is greater than the port width. In these cases, the device transfers the greatest number of bytes possible for the port. For example, if the size is four and $A1, A0 = 01$, a 32-bit slave can only receive three bytes in the current bus cycle. A 16- or 8-bit slave can only receive one byte. The table defines the byte enables for all port sizes. Byte data strobes can be obtained by combining the enable signals with the \overline{DS} signal. Devices residing on 8-bit ports can use the data strobe by itself since there is only one valid byte for every transfer. These enable or strobe signals select only the bytes required for write or read cycles. The other bytes are not selected, which prevents incorrect accesses in sensitive areas such as I/O.

Table 5-7. Data Bus Byte Enable Signals for Byte, Word, and Long-Word Ports

| Transfer Size | SIZ1 | SIZ0 | A1 | A0 | Data Bus Active Sections Byte (B), Word (W), Long-Word (L) Ports | | | |
|---------------|------|------|----|----|---|---------|--------|-------|
| | | | | | D31–D24 | D23–D16 | D15–D8 | D7–D0 |
| Byte | 0 | 1 | 0 | 0 | B W L | — | — | — |
| | 0 | 1 | 0 | 1 | B | W L | — | — |
| | 0 | 1 | 1 | 0 | B W | — | L | — |
| | 0 | 1 | 1 | 1 | B | W | — | L |
| Word | 1 | 0 | 0 | 0 | B W L | W L | — | — |
| | 1 | 0 | 0 | 1 | B | W L | L | — |
| | 1 | 0 | 1 | 0 | B W | W | L | L |
| | 1 | 0 | 1 | 1 | B | W | — | L |
| 3 Bytes | 1 | 1 | 0 | 0 | B W L | W L | L | — |
| | 1 | 1 | 0 | 1 | B | W L | L | L |
| | 1 | 1 | 1 | 0 | B W | W | L | L |
| | 1 | 1 | 1 | 1 | B | W | — | L |
| Long Word | 0 | 0 | 0 | 0 | B W L | W L | L | L |
| | 0 | 0 | 0 | 1 | B | W L | L | L |
| | 0 | 0 | 1 | 0 | B W | W | L | L |
| | 0 | 0 | 1 | 1 | B | W | — | L |

Figure 5-18 shows a logic diagram of one method for generating byte enable signals for 16- and 32-bit ports from the SIZ1, SIZ0, A1, and A0 encodings and the R/W signal.

5.2.5 Cache Interactions

The organization and requirements of the on-chip instruction cache affect the interpretation of $\overline{\text{DSACK1}}$ and $\overline{\text{DSACK0}}$. Since the MC68020/EC020 attempts to load all instructions into the on-chip cache, the bus may operate differently when caching is enabled. Specifically, on read cycles that terminate normally, the A1, A0, SIZ1, and SIZ0 signals do not apply.

The cache can also affect the assertion of $\overline{\text{AS}}$ and the operation of a read cycle. The search of the cache by the processor begins when the sequencer requires an instruction. At this time, the bus controller may also initiate an external bus cycle in case the requested item is not resident in the instruction cache. If an internal cache hit occurs, the external cycle aborts, and $\overline{\text{AS}}$ is not asserted.

For the MC68020, if the bus is not occupied with another read or write cycle, the bus controller asserts the $\overline{\text{ECS}}$ signal (and the $\overline{\text{OCS}}$ signal, if appropriate). It is possible to have $\overline{\text{ECS}}$ asserted on multiple consecutive clock cycles. Note that there is a minimum time specified from the negation of $\overline{\text{ECS}}$ to the next assertion of $\overline{\text{ECS}}$ (refer to **Section 10 Electrical Characteristics**). Instruction prefetches can occur every other clock so that if, after an aborted cycle due to an instruction cache hit, the bus controller asserts $\overline{\text{ECS}}$ on the next clock, this second cycle is for a data fetch. Note that, if the bus controller is executing other cycles, these aborted cycles due to cache hits may not be seen externally.

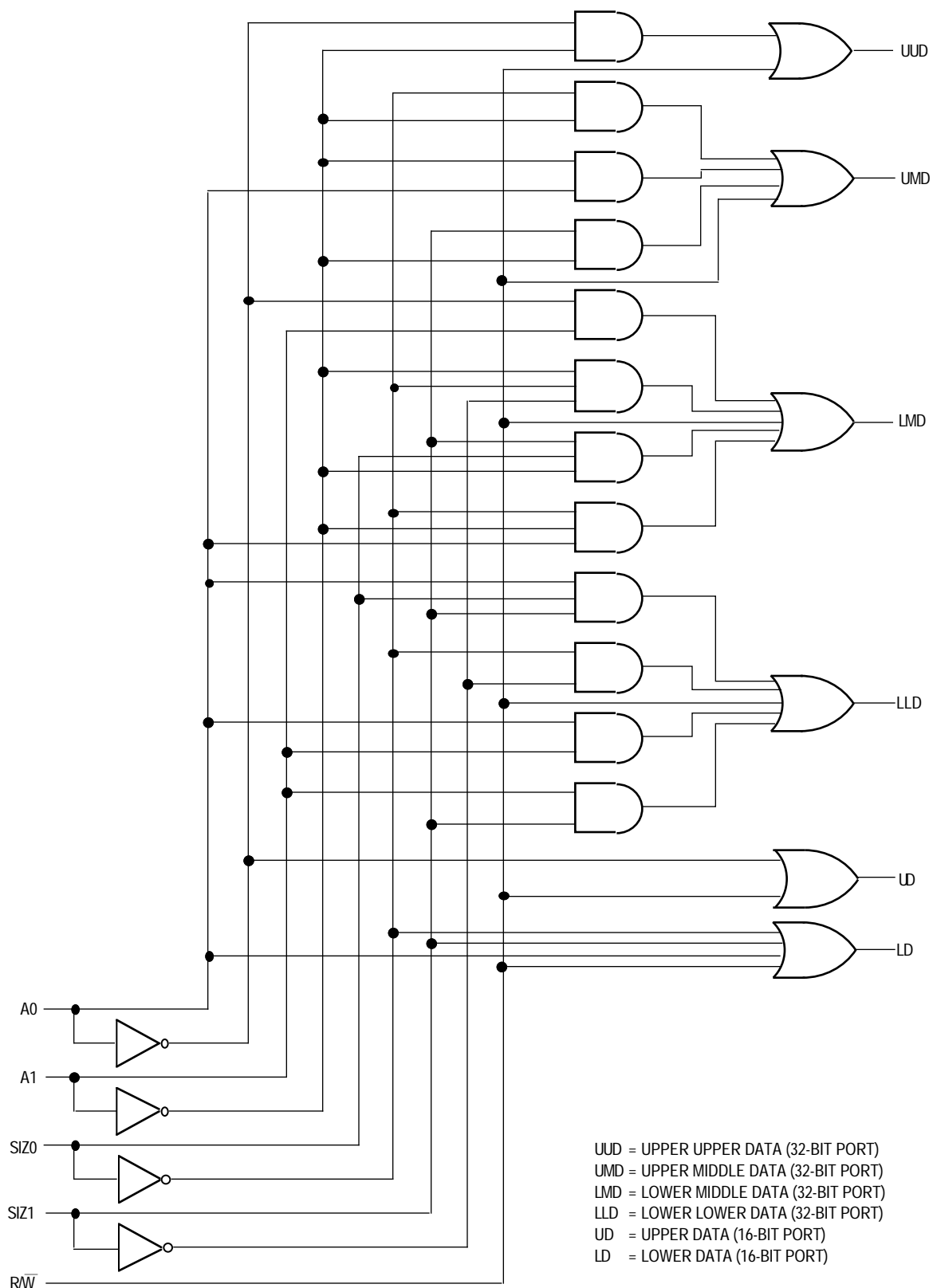


Figure 5-18. Byte Enable Signal Generation for 16- and 32-Bit Ports

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

for asynchronous operation can be ignored. All timing parameters referred to are described in **Section 10 Electrical Characteristics**. If a system asserts $\overline{\text{DSACK1/DSACK0}}$ for the required window around the falling edge of state 2 and obeys the proper bus protocol by maintaining $\overline{\text{DSACK1/DSACK0}}$ (and/or $\overline{\text{BERR/HALT}}$) until and throughout the clock edge that negates $\overline{\text{AS}}$ (with the appropriate asynchronous input hold time specified by parameter #47B), no wait states are inserted. The bus cycle runs at its maximum speed of three clocks per cycle for bus cycles terminated with $\overline{\text{DSACK1/DSACK0}}$.

To ensure proper operation in a synchronous system when $\overline{\text{BERR}}$ or $\overline{\text{BERR/HALT}}$ is asserted after $\overline{\text{DSACK1/DSACK0}}$, $\overline{\text{BERR}}$ (and $\overline{\text{HALT}}$) must meet the appropriate setup time (parameter #27A) prior to the falling clock edge one clock cycle after $\overline{\text{DSACK1/DSACK0}}$ is recognized. This setup time is critical, and the MC68020/EC020 may exhibit erratic behavior if it is violated.

When operating synchronously, the data-in setup (parameter #27) and hold (parameter #30) times for synchronous cycles may be used instead of the timing requirements for data relative to the $\overline{\text{DS}}$ signal.

5.3 DATA TRANSFER CYCLES

The transfer of data between the processor and other devices involves the following signals:

- Address Bus (A31–A0 for the MC68020) (A23–A0 for the MC68EC020)
- Data Bus (D31–D0)
- Control Signals

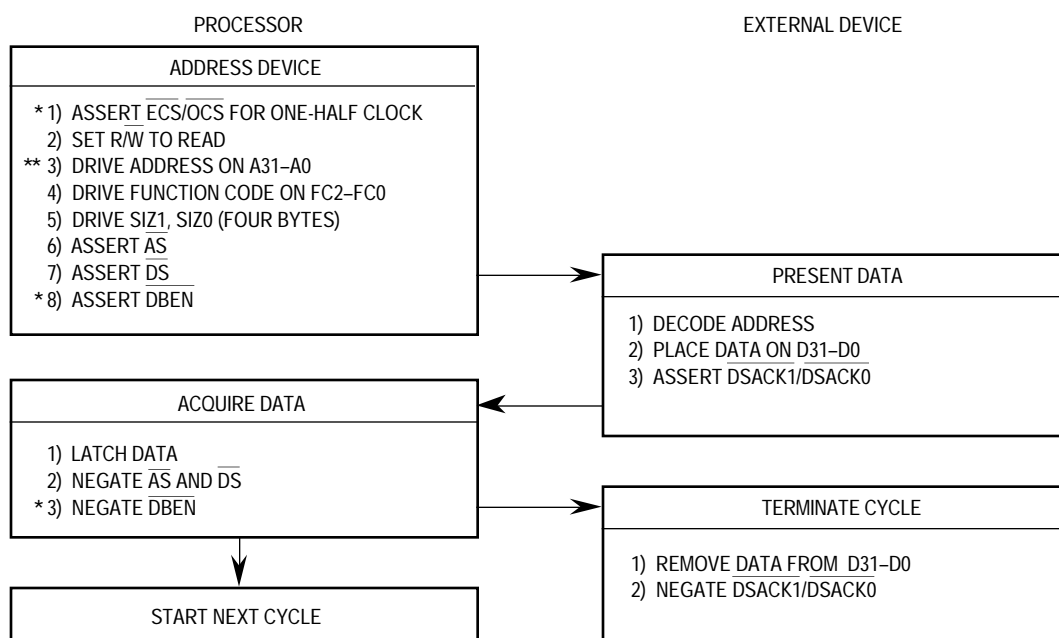
The address and data buses are both parallel, nonmultiplexed buses. The bus master moves data on the bus by issuing control signals, and the bus uses a handshake protocol to ensure correct movement of the data. In all bus cycles, the bus master is responsible for de-skewing all signals it issues at both the start and end of the cycle. In addition, the bus master is responsible for de-skewing $\overline{\text{DSACK1/DSACK0}}$, D31–D0, $\overline{\text{BERR}}$, $\overline{\text{HALT}}$, and, for the MC68020, $\overline{\text{DBEN}}$ from the slave devices. The following paragraphs define read, write, and read-modify-write cycle operations.

Each of the bus cycles is defined as a succession of states. These states apply to the bus operation and are different from the processor states described in **Section 2 Processing States**. The clock cycles used in the descriptions and timing diagrams of data transfer cycles are independent of the clock frequency. Bus operations are described in terms of external bus states.

5.3.1 Read Cycle

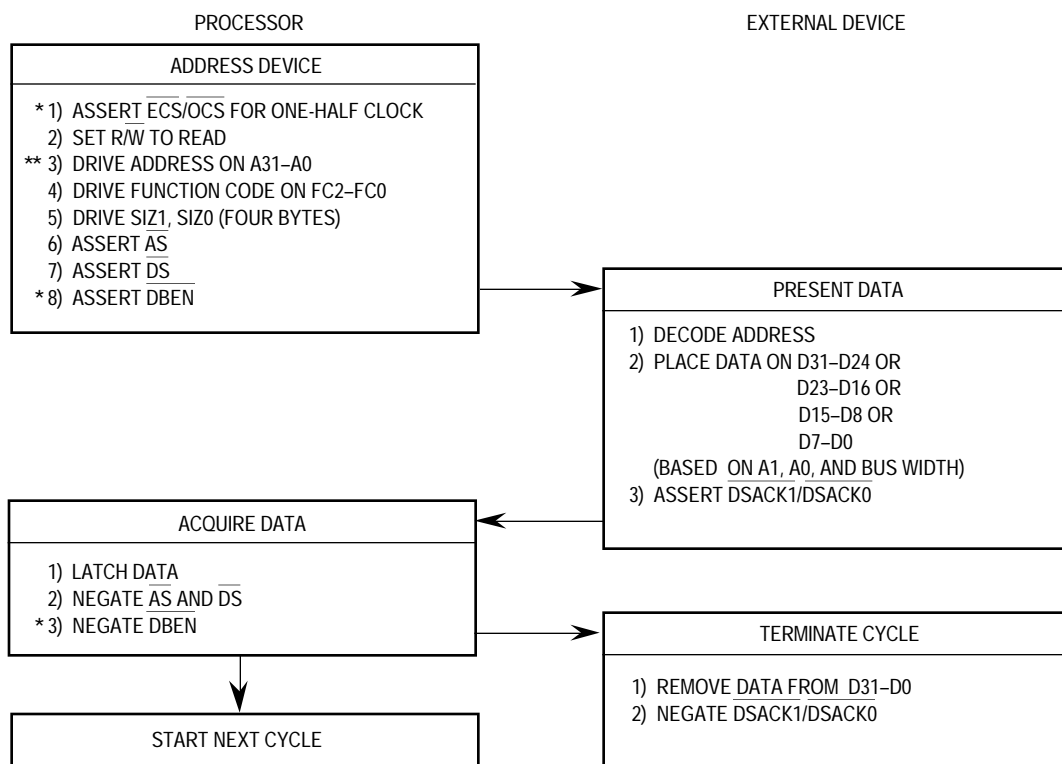
During a read cycle, the processor receives data from a memory, coprocessor, or peripheral device. If the instruction specifies a long-word operation, the MC68020/EC020 attempts to read four bytes at once. For a word operation, it attempts to read two bytes at once and for a byte operation, one byte. For some operations, the processor requests a three-byte transfer. The processor properly positions each byte internally. The section of the data bus from which each byte is read depends on the operand size, A1–A0, and the port size. Refer to **5.2.1 Dynamic Bus Sizing** and **5.2.2 Misaligned Operands** for more information on dynamic bus sizing and misaligned operands.

Figure 5-19 is a flowchart of a long-word read cycle. Figure 5-20 is a flowchart of a byte read cycle. Figures 5-21–5-23 are read cycle timing diagrams in terms of clock periods. Figure 5-21 corresponds to byte and word read cycles from a 32-bit port. Figure 5-22 corresponds to a long-word read cycle from an 8-bit port. Figure 5-23 also applies to a long-word read cycle, but from 16- and 32-bit ports.



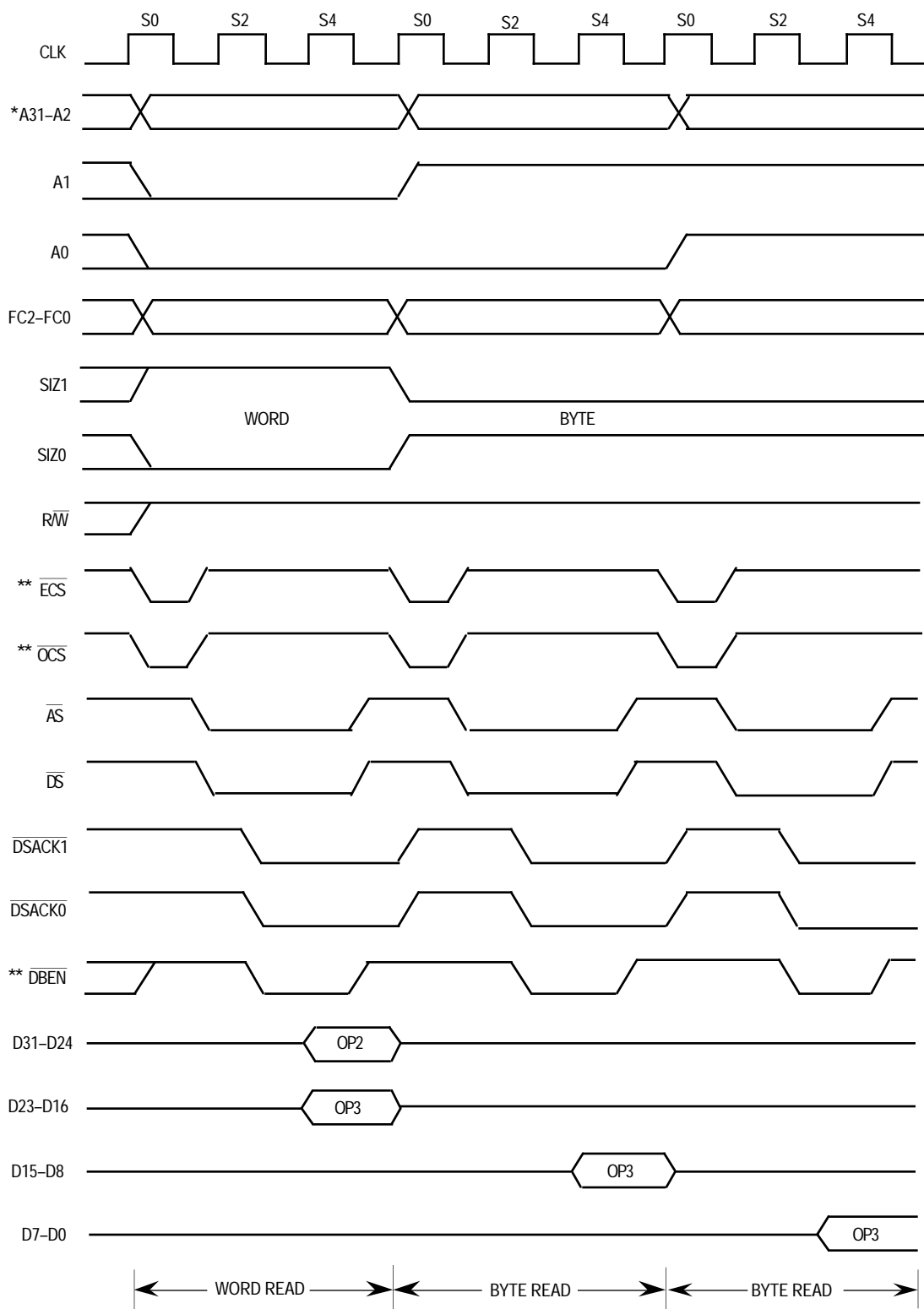
* This step does not apply to the MC68EC020.
For the MC68EC020, A23–A0.

Figure 5-19. Long-Word Read Cycle Flowchart



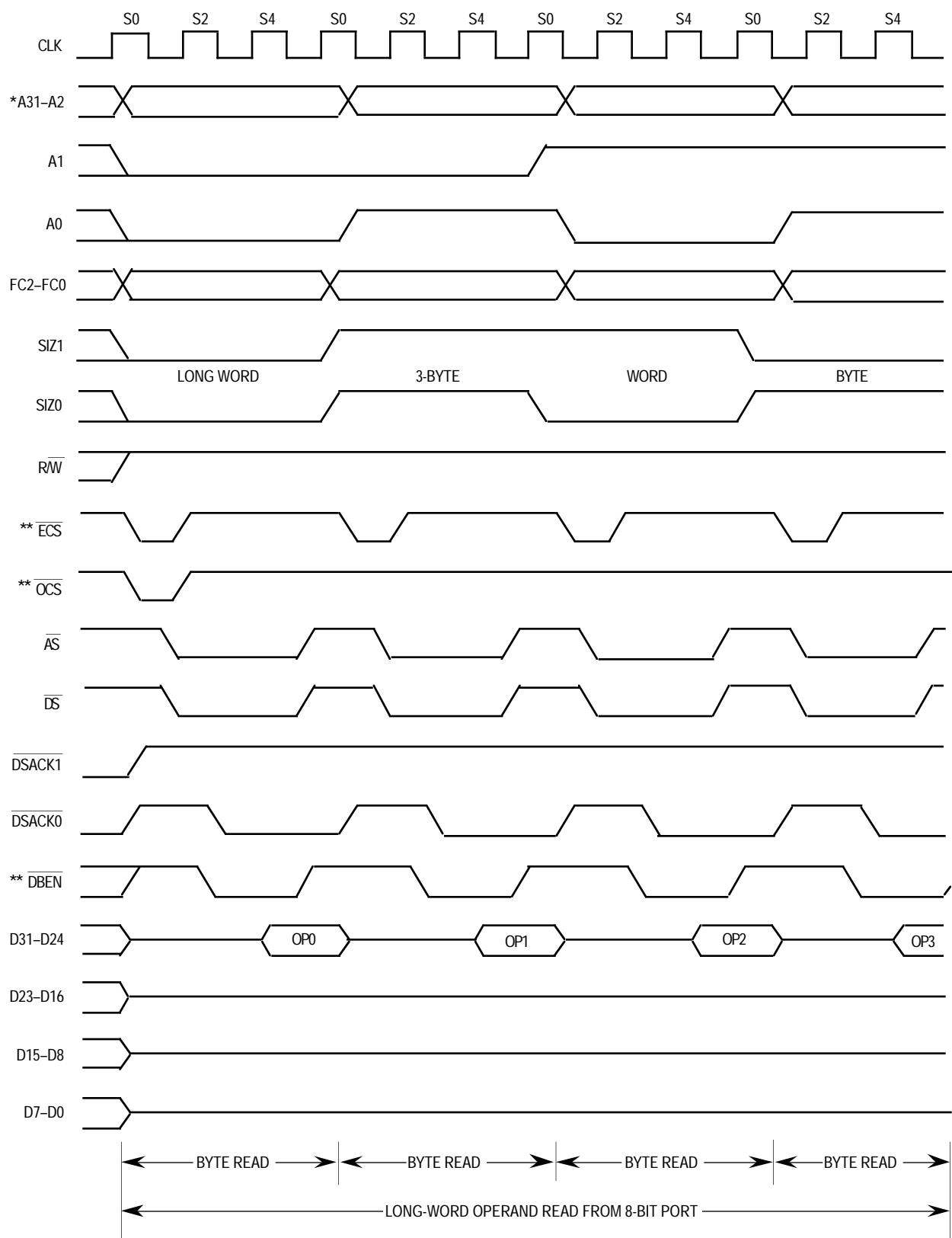
* This step does not apply to the MC68EC020.
 ** For the MC68EC020, A23-A0.

Figure 5-20. Byte Read Cycle Flowchart



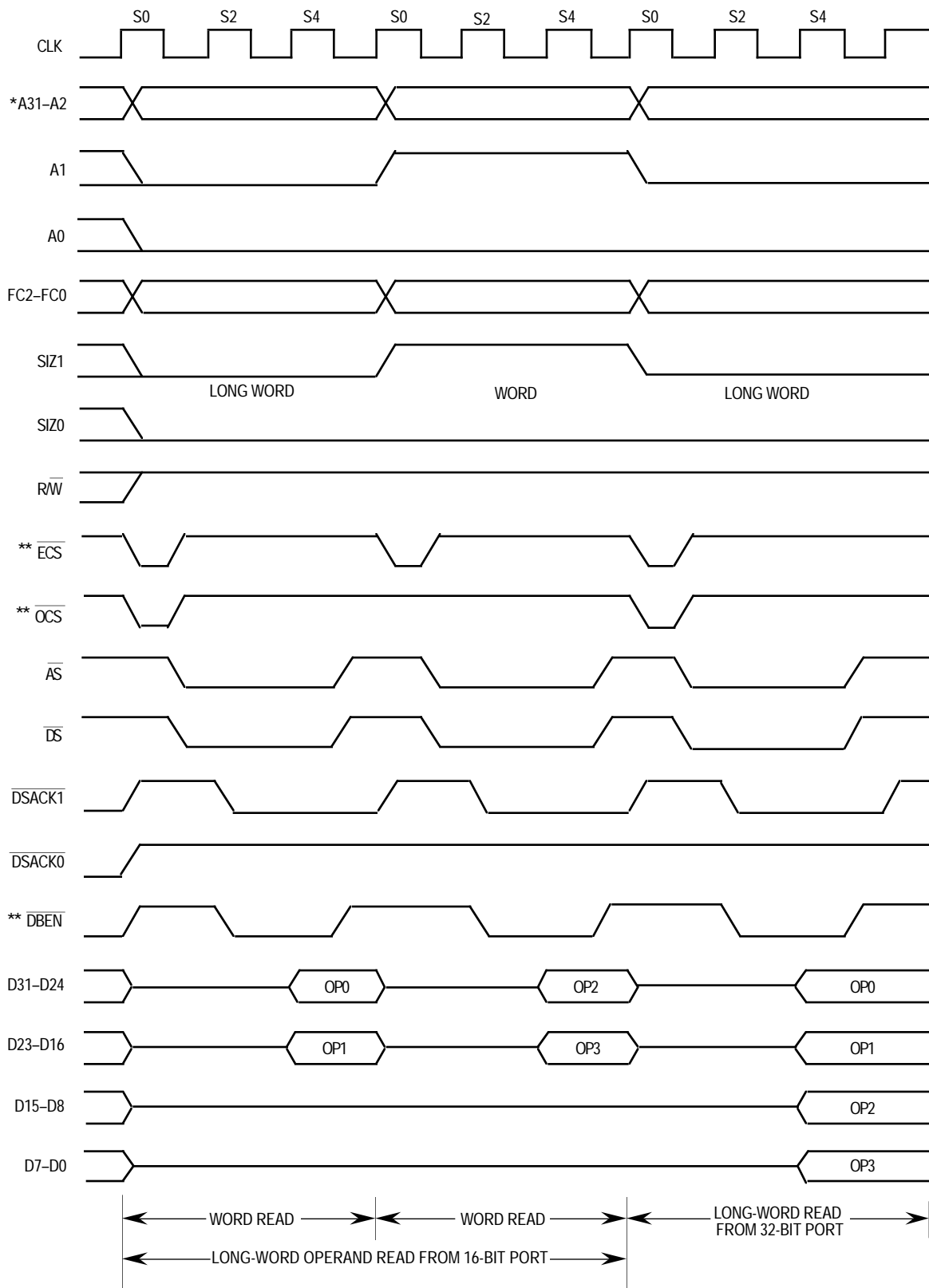
*

Figure 5-21. Byte and Word Read Cycles—32-Bit Port



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

Figure 5-22. Long-Word Read—8-Bit Port



* For the MC68EC020, A23-A2.

** This signal does not apply to the MC68EC020.

Figure 5-23. Long-Word Read—16- and 32-Bit Ports

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/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/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/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 $\overline{DSACK1/DSACK0}$.

MC68EC020—During S2, the selected device uses R/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 $\overline{DSACK1/DSACK0}$.

State 3

MC68020/EC020—As long as at least one of the $\overline{DSACK1/DSACK0}$ 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 $\overline{DSACK1/DSACK0}$ 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 $\overline{DSACK1/DSACK0}$ signals on the falling edges of the clock until an assertion is recognized.

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.

5.3.2 Write Cycle

During a write cycle, the processor transfers data to memory or a peripheral device. Figure 5-24 is a flowchart of a write cycle operation for a long-word transfer. Figures 5-25–5-28 are write cycle timing diagrams in terms of clock periods. Figure 5-25 shows two write cycles (between two read cycles with no idle time in between) for a 32-bit port. Figure 5-26 shows byte and word write cycles to a 32-bit port. Figure 5-27 shows a long-word write cycle to an 8-bit port. Figure 5-28 shows a long-word write cycle to a 16-bit port.

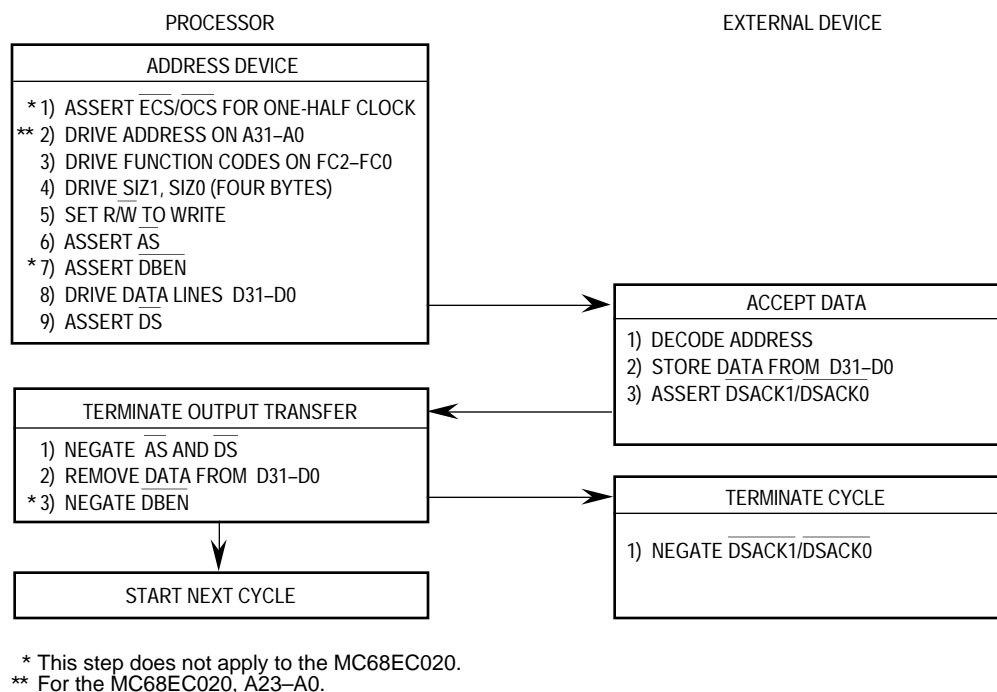
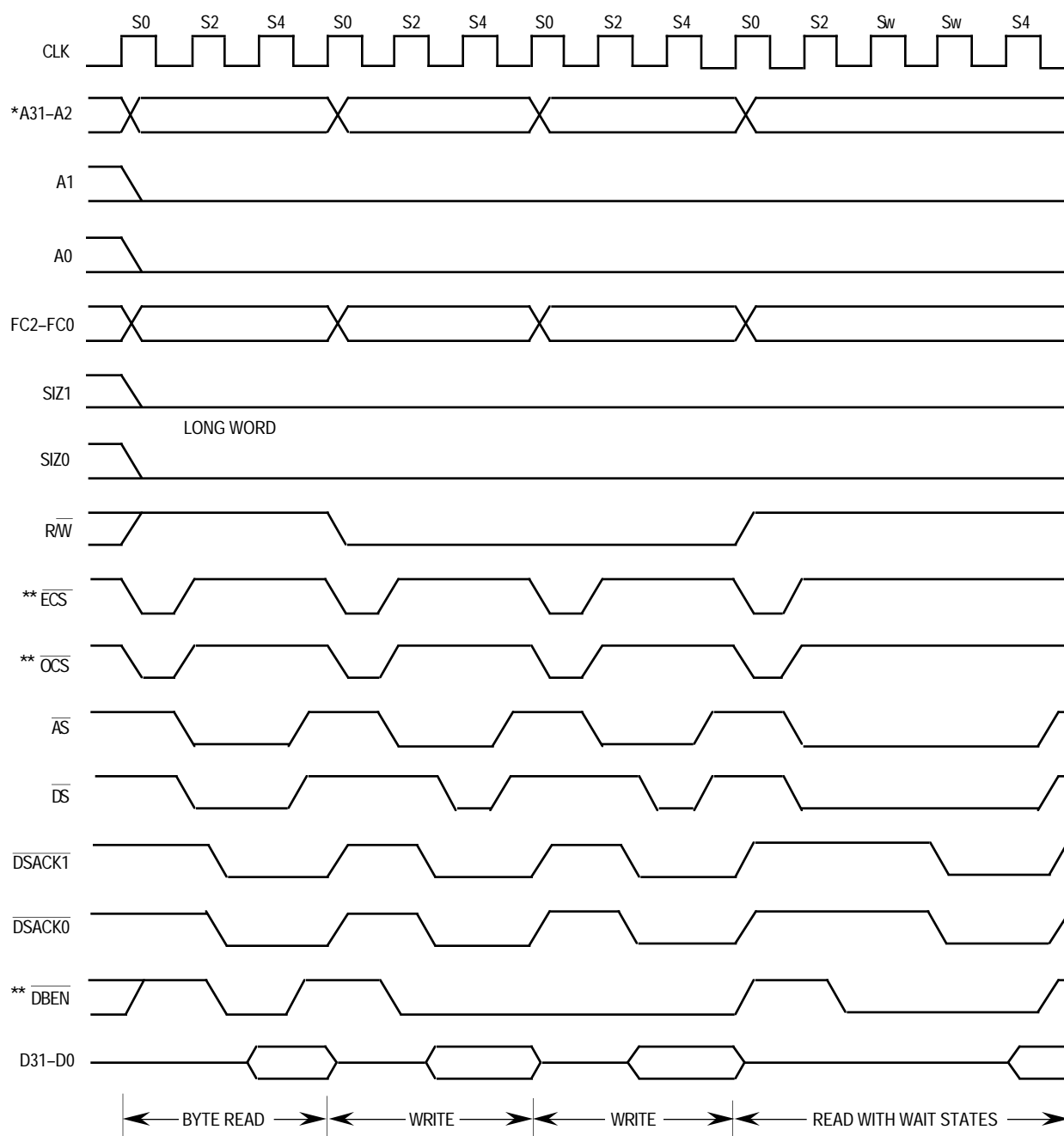


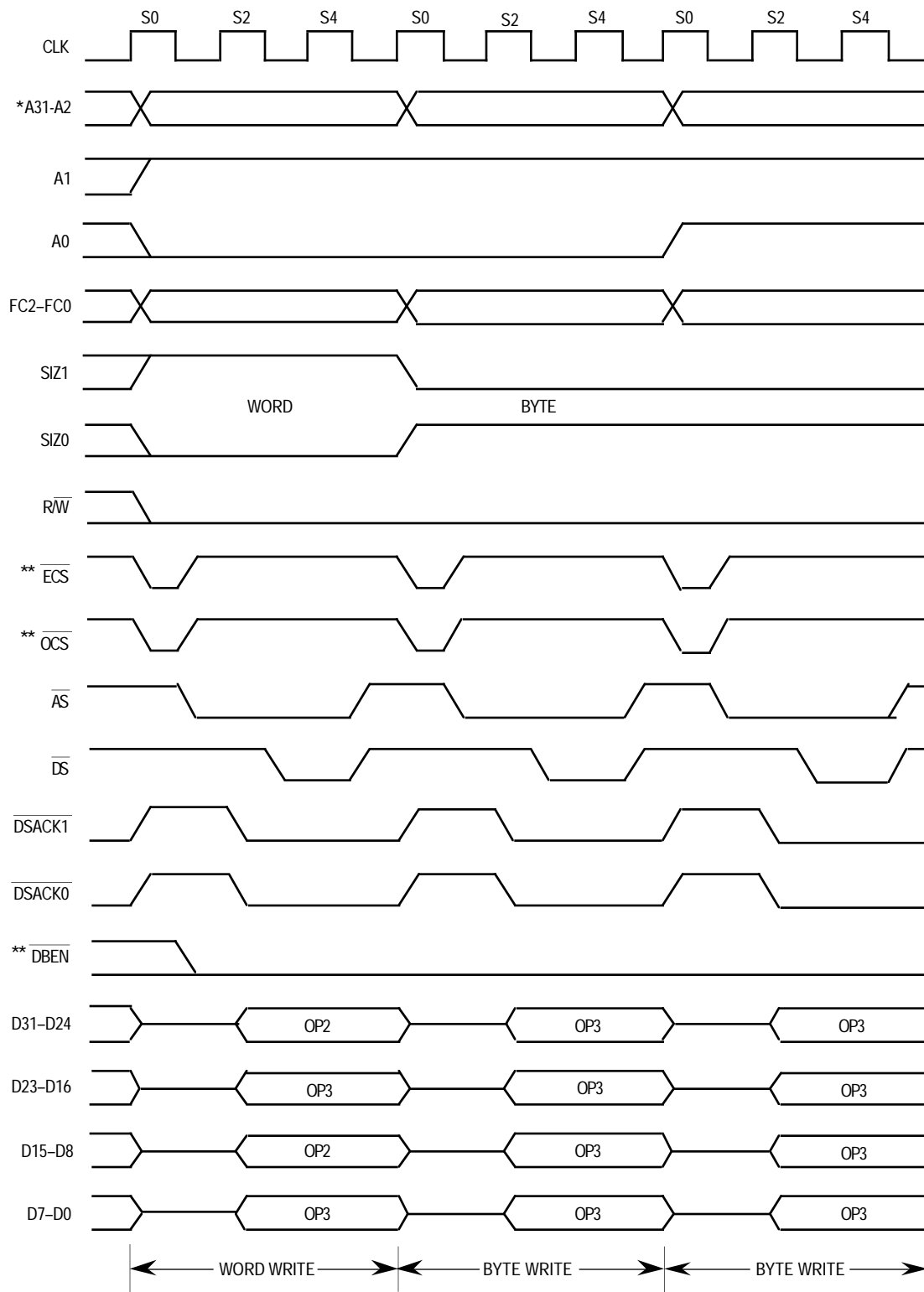
Figure 5-24. Write Cycle Flowchart



* For the MC68EC020, A23-A2.

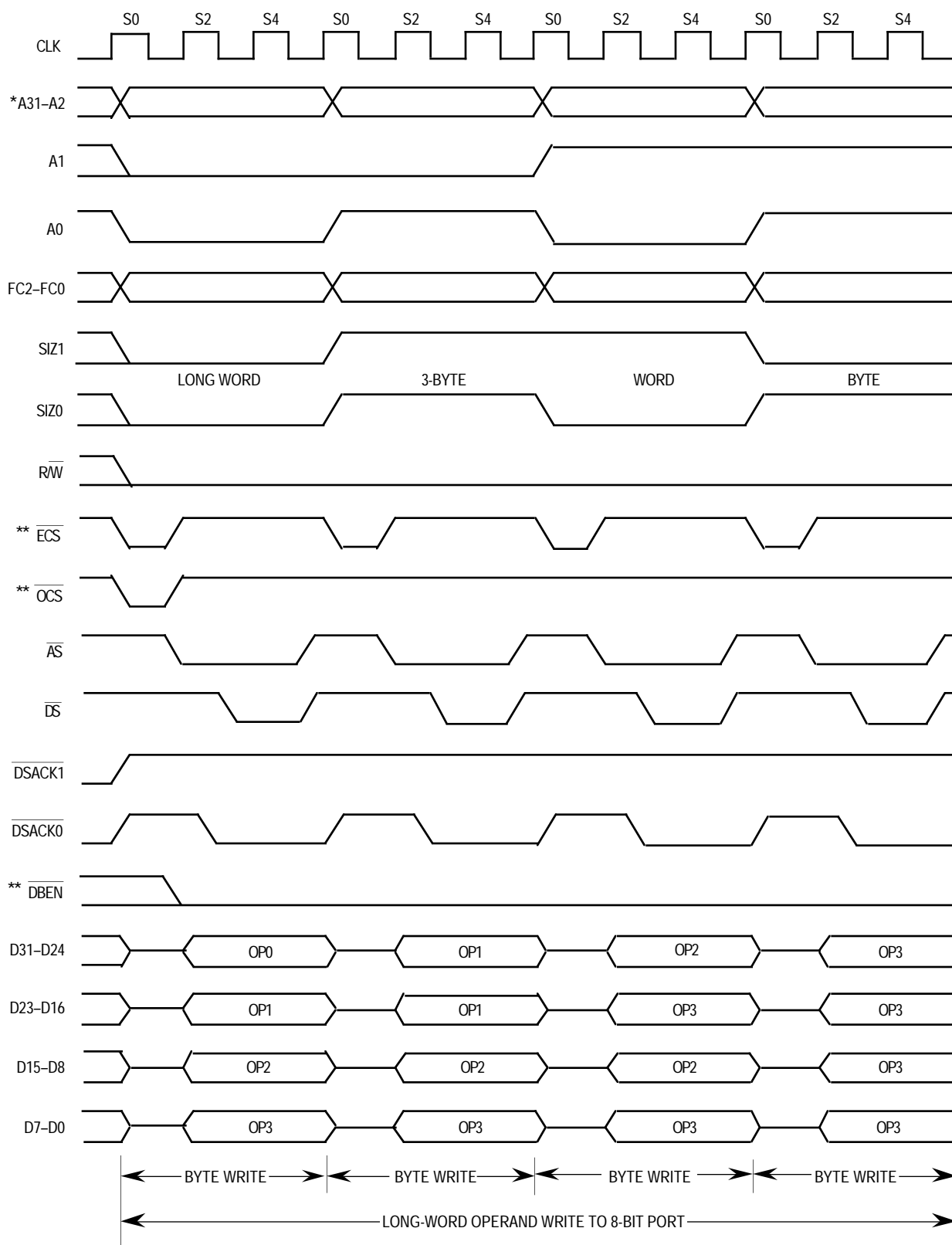
** This signal does not apply to the MC68EC020.

Figure 5-25. Read-Write-Read Cycles—32-Bit Port



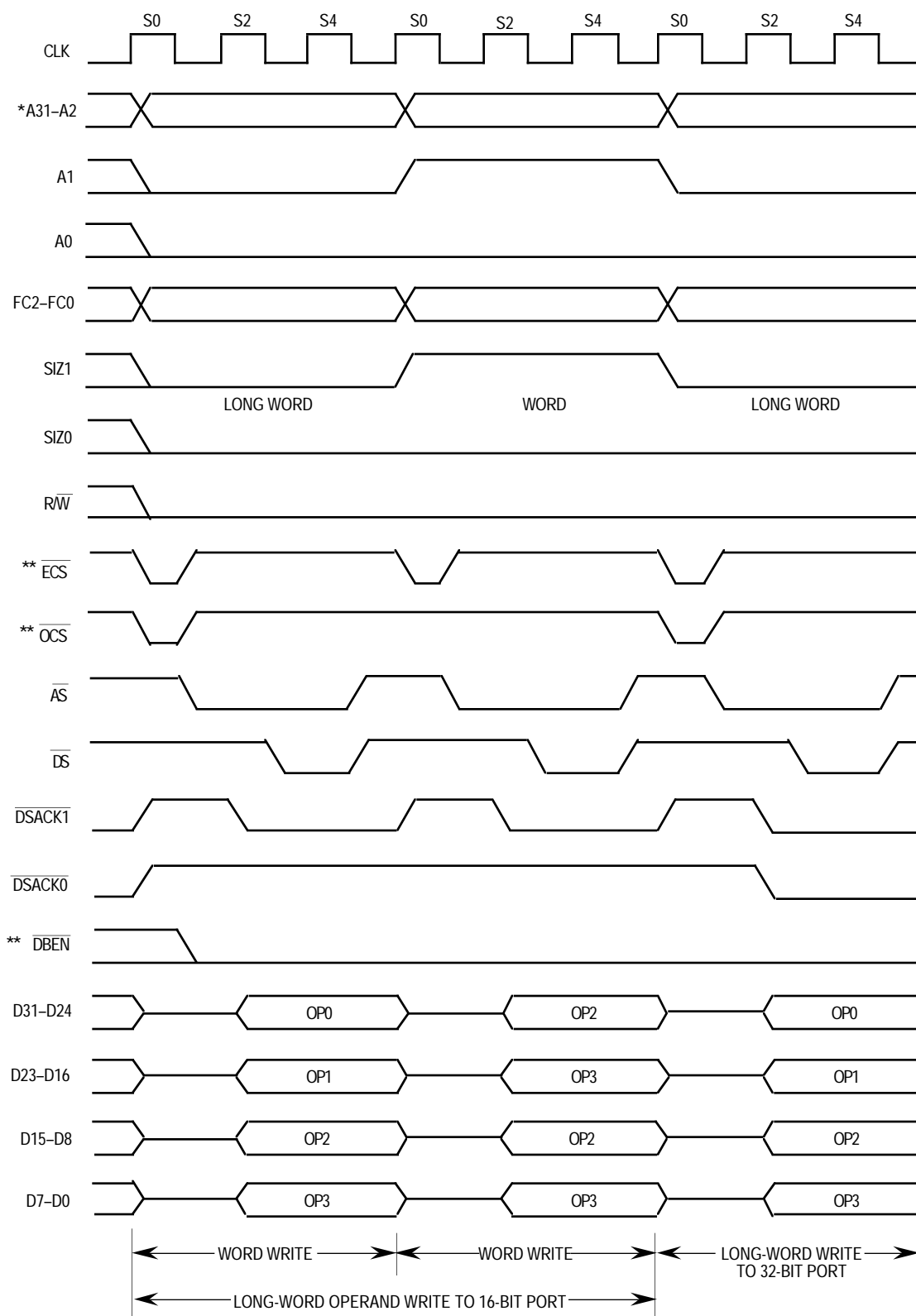
* For the MC68EC020, A23-A2.
This signal does not apply to the MC68EC020.

Figure 5-26. Byte and Word Write Cycles—32-Bit Port



* 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-A2.

** This signal does not apply to the MC68EC020.

Figure 5-28. Long-Word Operand Write—16-Bit Port

State 0

MC68020—The write cycle starts in S0. The processor negates \overline{ECS} , indicating the beginning of an external cycle. If the cycle is the first external cycle of a write 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/W low for a write cycle. SIZ1–SIZ0 become valid, indicating the number of bytes to be transferred.

MC68EC020—The write 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/W low for a write cycle. SIZ1, SIZ0 become valid, indicating the number of bytes to be transferred.

State 1

MC68020—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{DBEN} during S1, which can enable external data buffers. 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.

State 2

MC68020/EC020—During S2, the processor places the data to be written onto D31–D0. At the end of S2, the processor samples $\overline{DSACK1}/\overline{DSACK0}$.

State 3

MC68020/EC020—The processor asserts \overline{DS} during S3, indicating that the data on the data bus is stable. 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), the cycle terminates one clock later. If $\overline{DSACK1}/\overline{DSACK0}$ is not recognized by the start of S3, the processor inserts wait states instead of proceeding to S4 and S5. 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 one is recognized.

The external device uses R/W, \overline{DS} , SIZ1, SIZ0, A1, and A0 to latch data from the appropriate byte(s) of the data bus (D31–D24, D23–D16, D15–D8, and D7–D0). SIZ1, SIZ0, A1, and A0 select the bytes of the data bus. If it has not already done so, the device asserts $\overline{DSACK1}/\overline{DSACK0}$ to signal that it has successfully stored the data.

State 4

MC68020/EC020—The processor issues no new control signals during S4.

State 5

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

The external device must keep $\overline{DSACK1}/\overline{DSACK0}$ asserted until it detects the negation of \overline{AS} or \overline{DS} (whichever it detects first). The device must 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 S5. It holds the address and data valid during S5 to provide address hold time for memory systems. R/\overline{W} , $SIZ1$, $SIZ0$, and $FC2$ – $FC0$ also remain valid throughout S5.

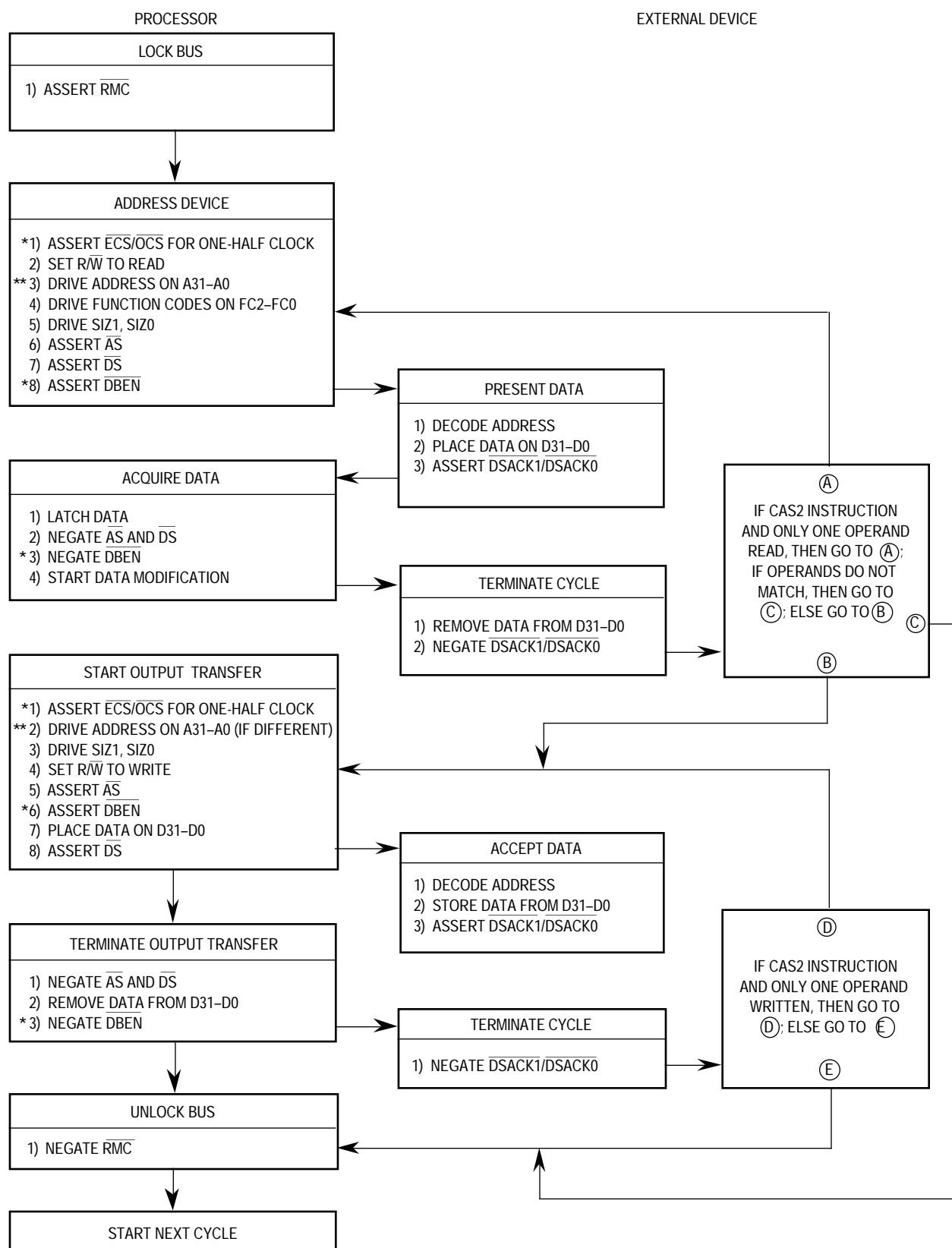
The external device must keep $\overline{DSACK1}/\overline{DSACK0}$ asserted until it detects the negation of \overline{AS} or \overline{DS} (whichever it detects first). The device must negate $\overline{DSACK1}/\overline{DSACK0}$ within approximately one clock period after sensing the negation of \overline{AS}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

5.3.3 Read-Modify-Write Cycle

The read-modify-write cycle performs a read, conditionally modifies the data in the arithmetic logic unit, and may write the data out to memory. In the MC68020/EC020, this operation is indivisible, providing semaphore capabilities for multiprocessor systems. During the entire read-modify-write sequence, the MC68020/EC020 asserts \overline{RMC} to indicate that an indivisible operation is occurring. The MC68020/EC020 does not issue a \overline{BG} signal in response to a \overline{BR} signal during this operation.

The TAS, CAS, and CAS2 instructions are the only MC68020/EC020 instructions that utilize read-modify-write operations. Depending on the compare results of the CAS and CAS2 instructions, the write cycle(s) may not occur.

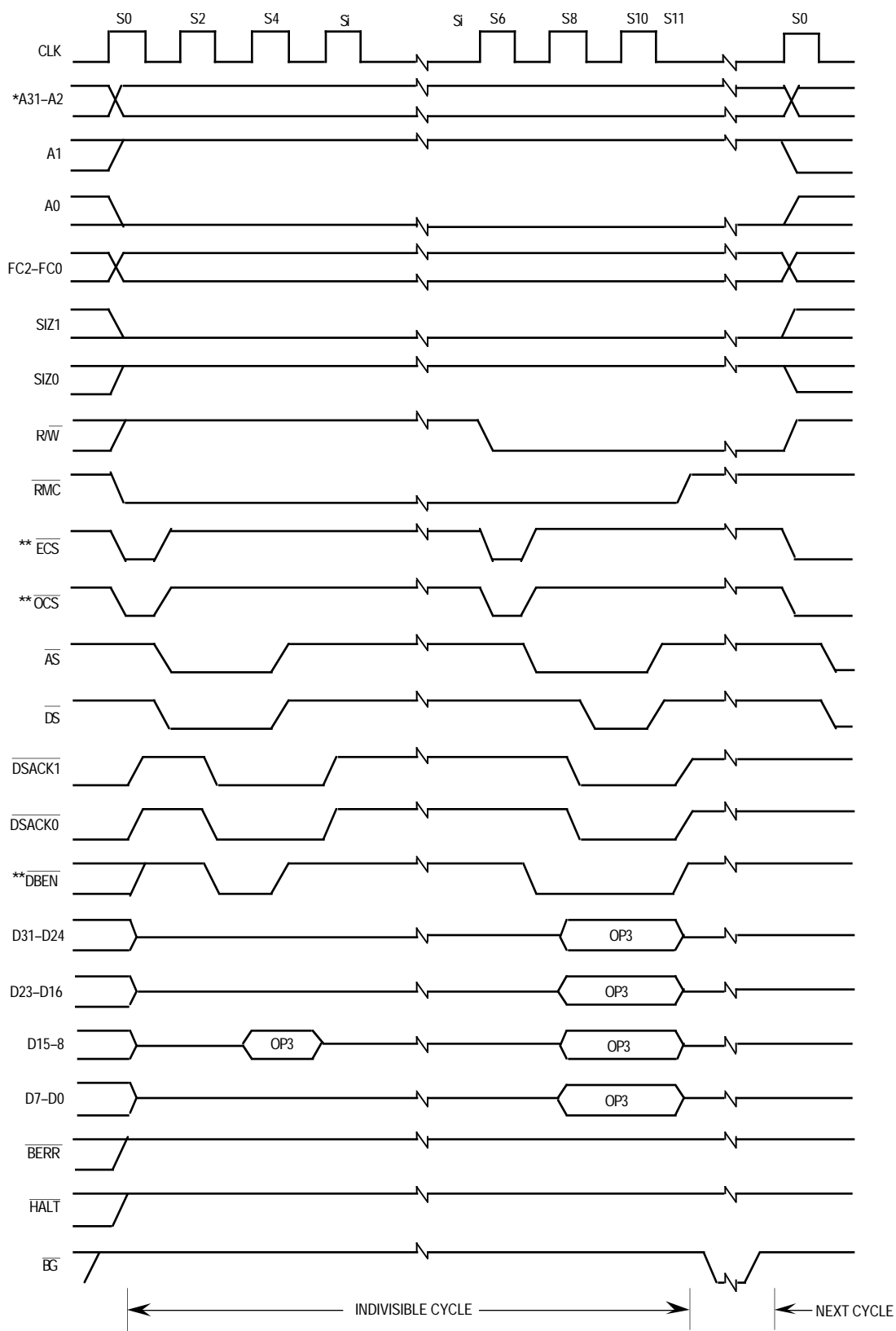
Figure 5-29 is a flowchart of the read-modify-write cycle operation. Figure 5-30 is an example timing diagram of a TAS instruction specified in terms of clock periods.



* This step does not apply to the MC68EC020.

** For the MC68EC020, A23-A0.

Figure 5-29. Read-Modify-Write Cycle Flowchart



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

Figure 5-30. Byte Read-Modify-Write Cycle—32-Bit Port (TAS Instruction)

State 0

MC68020—The processor asserts \overline{ECS} and \overline{OCS} in S0 to indicate the beginning of an external operand cycle. The processor also asserts \overline{RMC} in S0 to identify a read-modify-write cycle. 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 operation. SIZ1, SIZ0 become valid in S0 to indicate the operand size. The processor drives R/W high for the read cycle.

MC68EC020—The processor asserts \overline{RMC} in S0 to identify a read-modify-write cycle. 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 operation. SIZ1–SIZ0 become valid in S0 to indicate the operand size. The processor drives R/W high for the read cycle.

State 1

MC68020—One-half clock later in S1, the processor asserts \overline{AS} to indicate 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} to indicate that the address on the address bus is valid. The processor also asserts \overline{DS} during S1.

State 2

MC68020—During S2, the processor asserts \overline{DBEN} to enable external data buffers. The selected device uses R/W, SIZ1, SIZ0, A1, A0, and \overline{DS} to place 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, A1, and A0. Concurrently, the selected device may assert the $\overline{DSACK1/DSACK0}$ signals.

MC68EC020—During S2, the selected device uses R/W, SIZ1, SIZ0, A1, A0, and \overline{DS} to place 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, A1, and A0. Concurrently, the selected device may assert the $\overline{DSACK1/DSACK0}$ signals.

State 3

MC68020/EC020—As long as at least one of the $\overline{DSACK1/DSACK0}$ 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 $\overline{DSACK1/DSACK0}$ is not recognized by the start of S3, the processor inserts wait states instead of proceeding to S4 and S5. To ensure that wait states are inserted, both $\overline{DSACK0}$ and $\overline{DSACK1}$ 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 $\overline{DSACK1/DSACK0}$ signals on the falling edges of the clock until one is recognized.

State 4

MC68020/EC020—At the end of S4, the processor latches the incoming data.

State 5

MC68020—The processor negates \overline{AS} , \overline{DS} , and \overline{DBEN} during S5. If more than one read cycle is required to read in the operand(s), S0–S5 are repeated for each read cycle. When the read cycle(s) are complete, the processor holds the address, R/\overline{W} , and FC2–FC0 valid in preparation for the write portion of the cycle.

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 the 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 portion of the operation.

MC68EC020—The processor negates \overline{AS} , \overline{DS} , and \overline{DBEN} during S5. If more than one read cycle is required to read in the operand(s), S0–S5 are repeated for each read cycle. When the read cycle(s) is complete, the processor holds the address, R/\overline{W} , and FC2–FC0 valid in preparation for the write portion of the cycle.

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 the 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 portion of the operation.

Idle States

MC68020/EC020—The processor does not assert any new control signals during the idle states, but it may internally begin the modify portion of the cycle at this time. S6–S11 are omitted if no write cycle is required. If a write cycle is required, the R/\overline{W} signal remains in the read mode until S6 to prevent bus conflicts with the preceding read portion of the cycle; the data bus is not driven until S8.

State 6

MC68020—The processor asserts \overline{ECS} and \overline{OCS} in S6 to indicate that another external cycle is beginning. The processor drives R/\overline{W} low for a write cycle. Depending on the write operation to be performed, the address lines may change during S6.

MC68EC020—During S6, the processor drives R/\overline{W} low for a write cycle. Depending on the write operation to be performed, the address lines may change during S6.

State 7

MC68020—During S7, the processor asserts \overline{AS} , indicating that the address on the address bus is valid. The processor also asserts \overline{DBEN} , which can be used to enable data buffers. In addition, \overline{ECS} (and \overline{OCS} , if asserted) is negated during S7.

MC68EC020—During S7, the processor asserts \overline{AS} , indicating that the address on the address bus is valid.

State 8

MC68020/EC020—During S8, the processor places the data to be written onto the data bus.

State 9

MC68020/EC020—The processor asserts \overline{DS} during S9, indicating that the data on the data bus is stable. As long as at least one of the $\overline{DSACK1}/\overline{DSACK0}$ signals is recognized by the end of S8 (meeting the asynchronous input setup time requirement), the cycle terminates one clock later. If $\overline{DSACK1}/\overline{DSACK0}$ is not recognized by the start of S9, the processor inserts wait states instead of proceeding to S10 and S11. 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 S8. If wait states are added, the processor continues to sample $\overline{DSACK1}/\overline{DSACK0}$ signals on the falling edges of the clock until one is recognized.

The external device uses R/\overline{W} , \overline{DS} , $SIZ1$, $SIZ0$, $A1$, and $A0$ to latch data from the appropriate section(s) of the data bus ($D31-D24$, $D23-D16$, $D15-D8$, and $D7-D0$). $SIZ1$, $SIZ0$, $A1$, and $A0$ select the data bus sections. If it has not already done so, the device asserts $\overline{DSACK1}/\overline{DSACK0}$ when it has successfully stored the data.

State 10

MC68020/EC020—The processor issues no new control signals during S10.

State 11

MC68020/EC020—The processor negates \overline{AS} and \overline{DS} during S11. It holds the address and data valid during S11 to provide address hold time for memory systems. R/\overline{W} and $FC2-FC0$ also remain valid throughout S11.

If more than one write cycle is required, S6–S11 are repeated for each write cycle.

The external device keeps $\overline{DSACK1}/\overline{DSACK0}$ 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} .

5.4 CPU SPACE CYCLES

$FC2-FC0$ select user and supervisor program and data areas as listed in Table 2-1. The area selected by $FC2-FC0 = 111$ is classified as the CPU space. The interrupt acknowledge, breakpoint acknowledge, module operations, and coprocessor communication cycles described in the following paragraphs utilize CPU space.

The CPU space type is encoded on $A19-A16$ during a CPU space operation and indicates the function that the processor is performing. On the MC68020/EC020, four of the encodings are implemented as shown in Figure 5-31. All unused values are reserved by Motorola for future use.

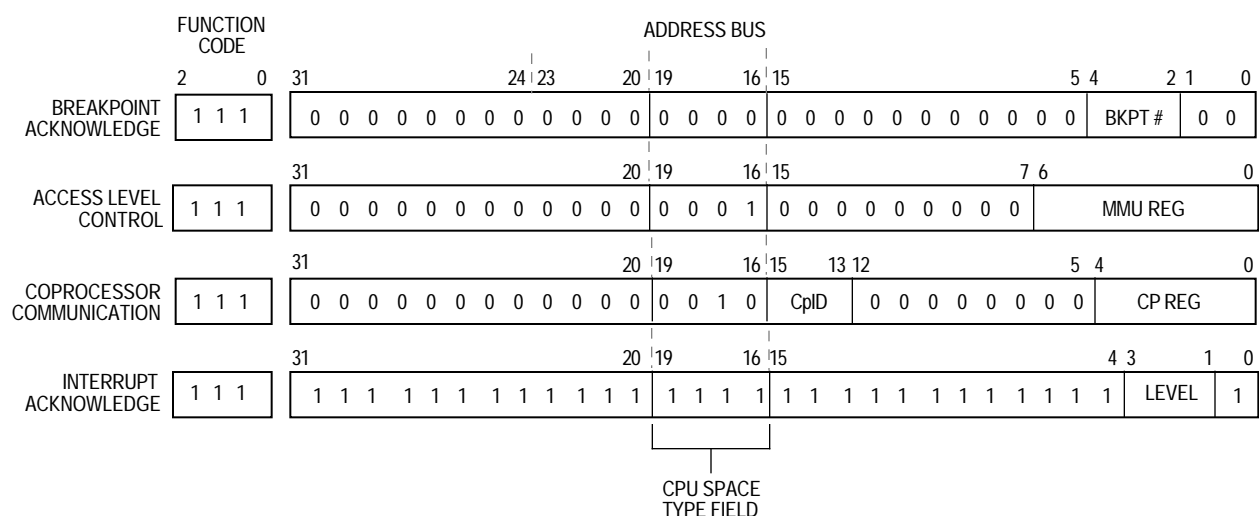


Figure 5-31. MC68020/EC020 CPU Space Address Encoding

5.4.1 Interrupt Acknowledge Bus Cycles

When a peripheral device signals the processor (with the $\overline{\text{IPL2}}\text{--}\overline{\text{IPL0}}$ signals) that the device requires service and when the internally synchronized value on these signals indicates a higher priority than the interrupt mask in the status register (or that a transition has occurred in the case of a level 7 interrupt), the processor makes the interrupt a pending interrupt. Refer to **Section 6 Exception Processing** for details on the recognition of interrupts.

The MC68020/EC020 takes an interrupt exception for a pending interrupt within one instruction boundary (after processing any other pending exception with a higher priority). The following paragraphs describe the various kinds of interrupt acknowledge bus cycles that can be executed as part of interrupt exception processing.

5.4.1.1 INTERRUPT ACKNOWLEDGE CYCLE—TERMINATED NORMALLY. When the MC68020/EC020 processes an interrupt exception, it performs an interrupt acknowledge cycle to obtain the number of the vector that contains the starting location of the interrupt service routine.

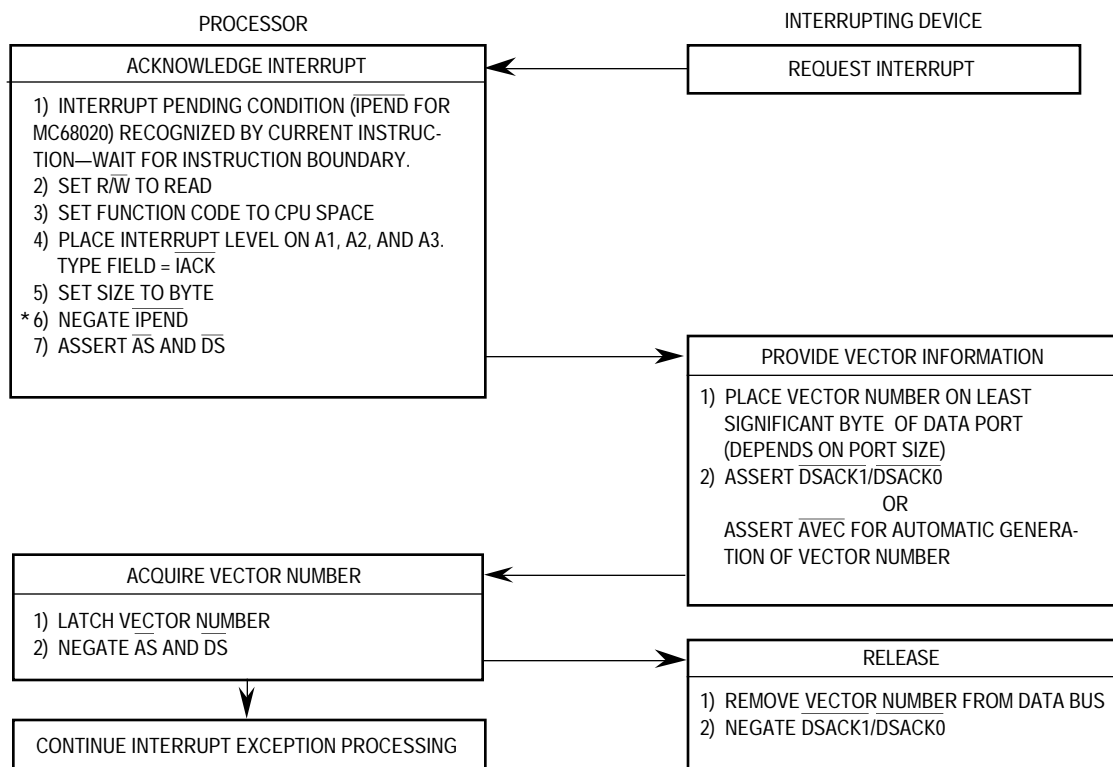
Some interrupting devices have programmable vector registers that contain the interrupt vectors for the routines they use. The following paragraphs describe the interrupt acknowledge cycle for these devices. Other interrupting conditions or devices cannot supply a vector number and use the autovector cycle described in **5.4.1.2 Autovector Interrupt Acknowledge Cycle**.

The interrupt acknowledge cycle is a read cycle. It differs from the read cycle described in **5.3.1 Read Cycle** in that it accesses the CPU address space. Specifically, the differences are:

1. FC2–FC0 are set 111 for CPU address space.
2. A3, A2, and A1 are set to the interrupt request level (the inverted values of $\overline{\text{IPL2}}$, $\overline{\text{IPL1}}$, and $\overline{\text{IPL0}}$, respectively).
3. The CPU space type field (A19–A16) is set to 1111, the interrupt acknowledge code.
4. Other address signals (A31–A20, A15–A4, and A0 for the MC68020; A23–A20, A15–A4, and A0 for the MC68EC020) are set to one.

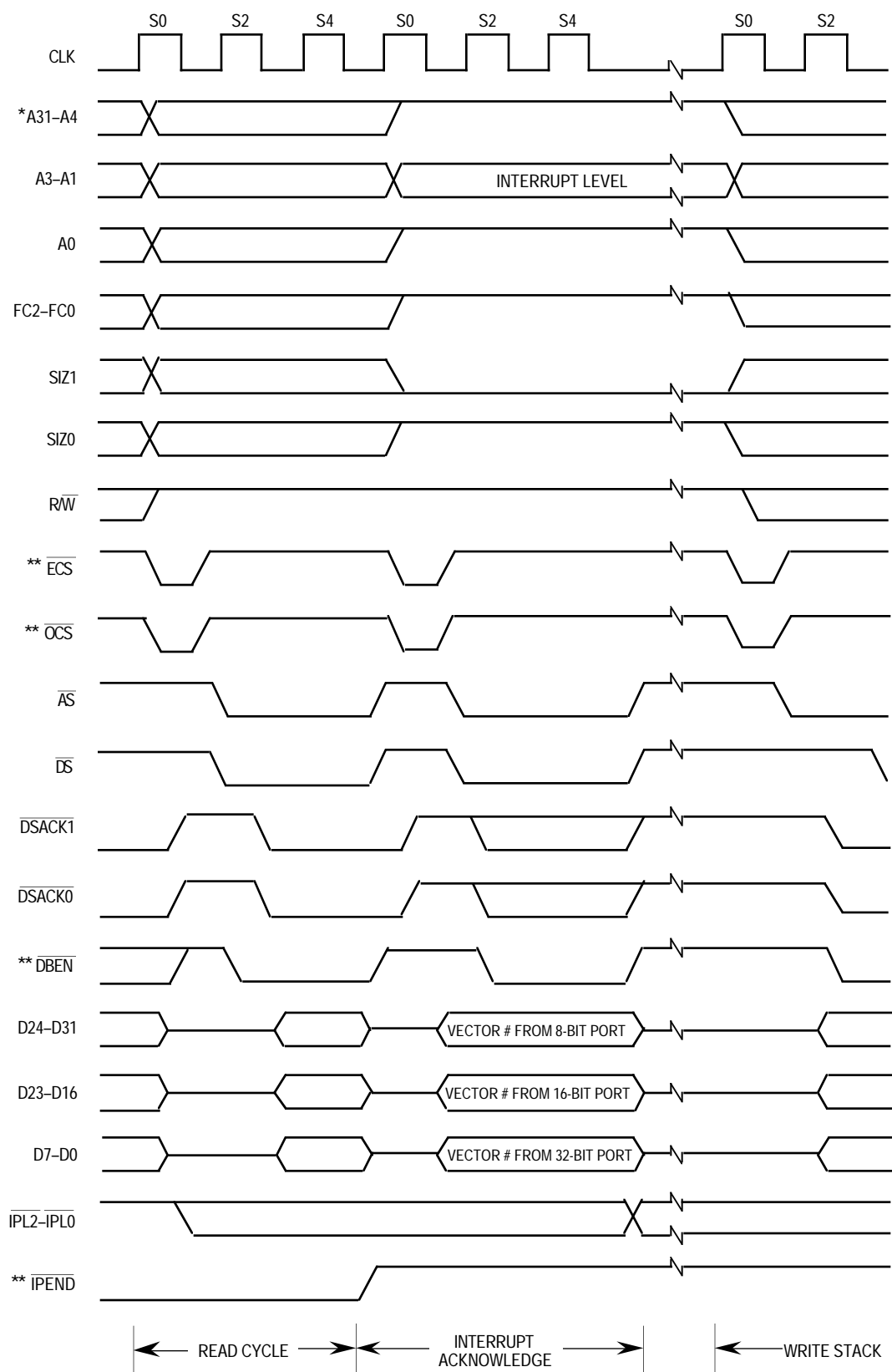
The responding device places the vector number on the data bus during the interrupt acknowledge cycle. Beyond this, the cycle is terminated normally with $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$. Figure 5-32 is the flowchart of the interrupt acknowledge cycle.

Figure 5-33 shows the timing for an interrupt acknowledge cycle terminated with $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$.



* This step does not apply to the MC68EC020.

Figure 5-32. Interrupt Acknowledge Cycle Flowchart



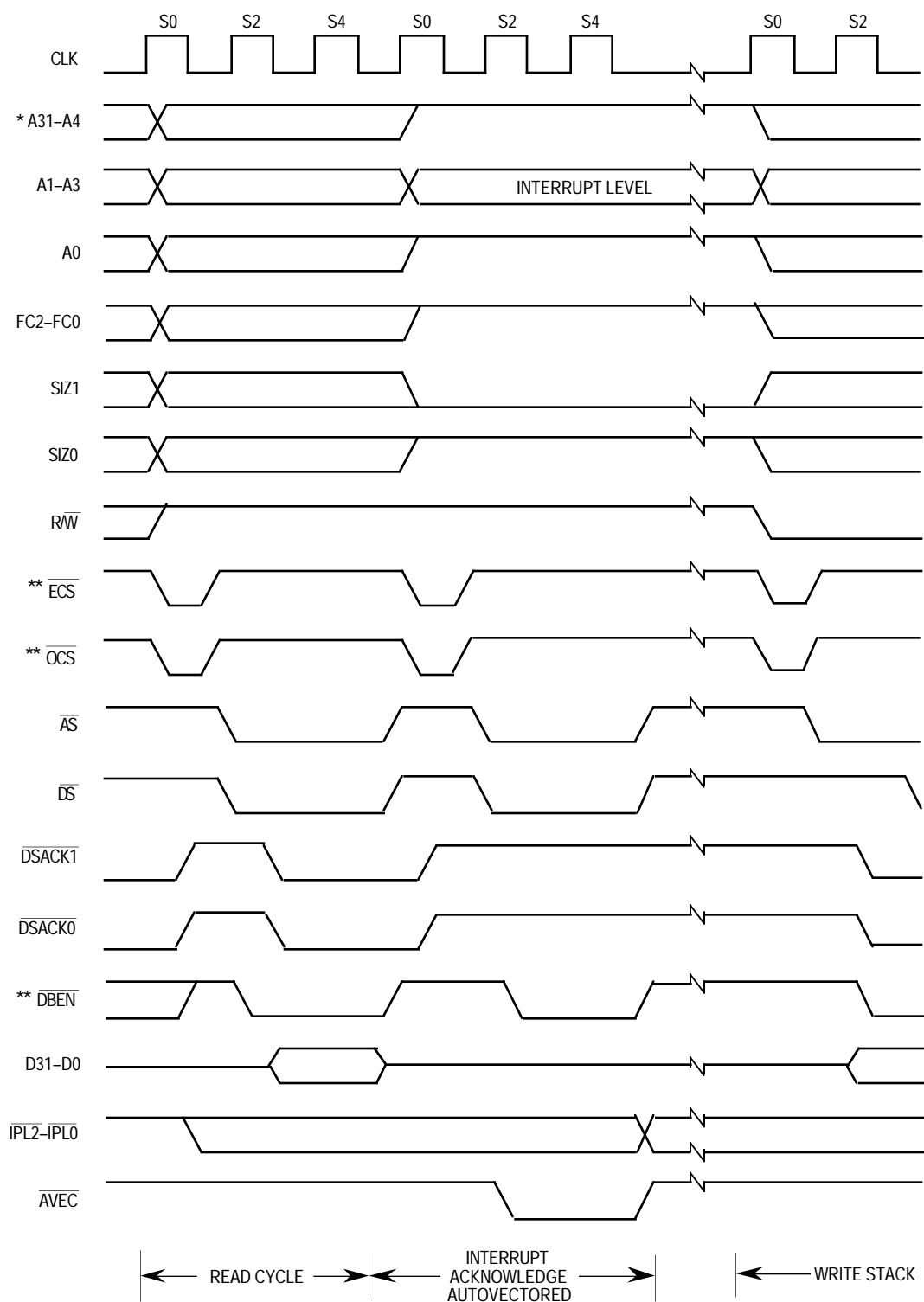
* For the MC68EC020, A23-A4.
 ** This signal does not apply to the MC68EC020.

Figure 5-33. Interrupt Acknowledge Cycle Timing

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 $\overline{\text{DSACK1/DSACK0}}$, the device asserts $\overline{\text{AVEC}}$ to terminate the cycle. The $\overline{\text{DSACK1/DSACK0}}$ signals may not be asserted during an interrupt acknowledge cycle terminated by $\overline{\text{AVEC}}$.

The vector number supplied in an autovector operation is derived from the interrupt level of the current interrupt. When $\overline{\text{AVEC}}$ is asserted instead of $\overline{\text{DSACK1/DSACK0}}$ 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 $\overline{\text{IPL2}}-\overline{\text{IPL0}}$, 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 $\overline{\text{AVEC}}$ or $\overline{\text{DSACK1/DSACK0}}$, the external logic typically returns $\overline{\text{BERR}}$. In this case, the MC68020/EC020 automatically generates 24, the spurious interrupt vector number. If $\overline{\text{HALT}}$ is also asserted, the processor retries the cycle.



* For the MC68EC020, A23-A4.

** This signal does not apply to the MC68EC020.

Figure 5-34. Autovector Operation Timing

5.4.2 Breakpoint Acknowledge Cycle

The breakpoint acknowledge cycle is generated by the execution of a BKPT instruction. The breakpoint acknowledge cycle allows the external hardware to provide an instruction word directly into the instruction pipeline as the program executes. This cycle accesses the CPU space with a type field of zero and provides the breakpoint number specified by the instruction on address lines A4–A2. If the external hardware terminates the cycle with $\overline{DSACK1}/\overline{DSACK0}$, the data on the bus (an instruction word) is inserted into the instruction pipe, replacing the breakpoint opcode, and is executed after the breakpoint acknowledge cycle completes. The BKPT instruction requires a word to be transferred so that if the first bus cycle accesses an 8-bit port, a second cycle is required. If the external logic terminates the breakpoint acknowledge cycle with \overline{BERR} (i.e., no instruction word available), the processor takes an illegal instruction exception. Figure 5-35 is a flowchart of the breakpoint acknowledge cycle. Figure 5-36 shows the timing for a breakpoint acknowledge cycle that returns an instruction word. Figure 5-37 shows the timing for a breakpoint acknowledge cycle that signals an exception.

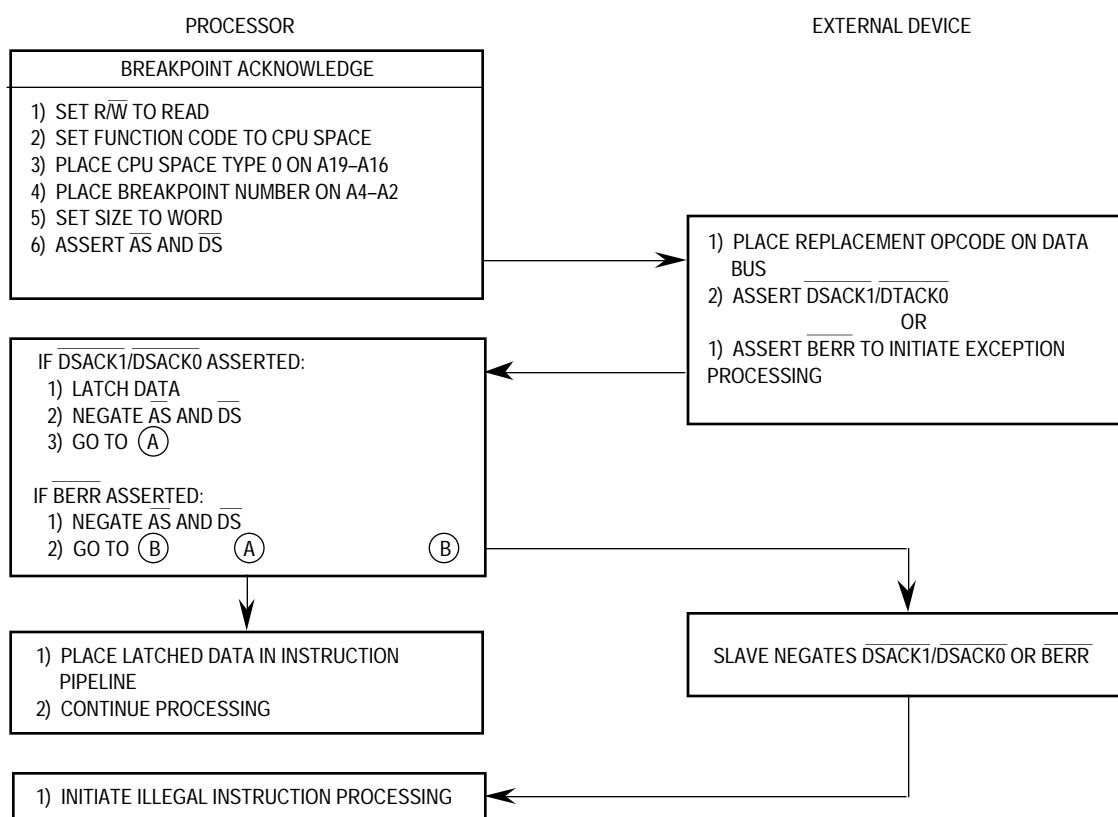


Figure 5-35. Breakpoint Acknowledge Cycle Flowchart

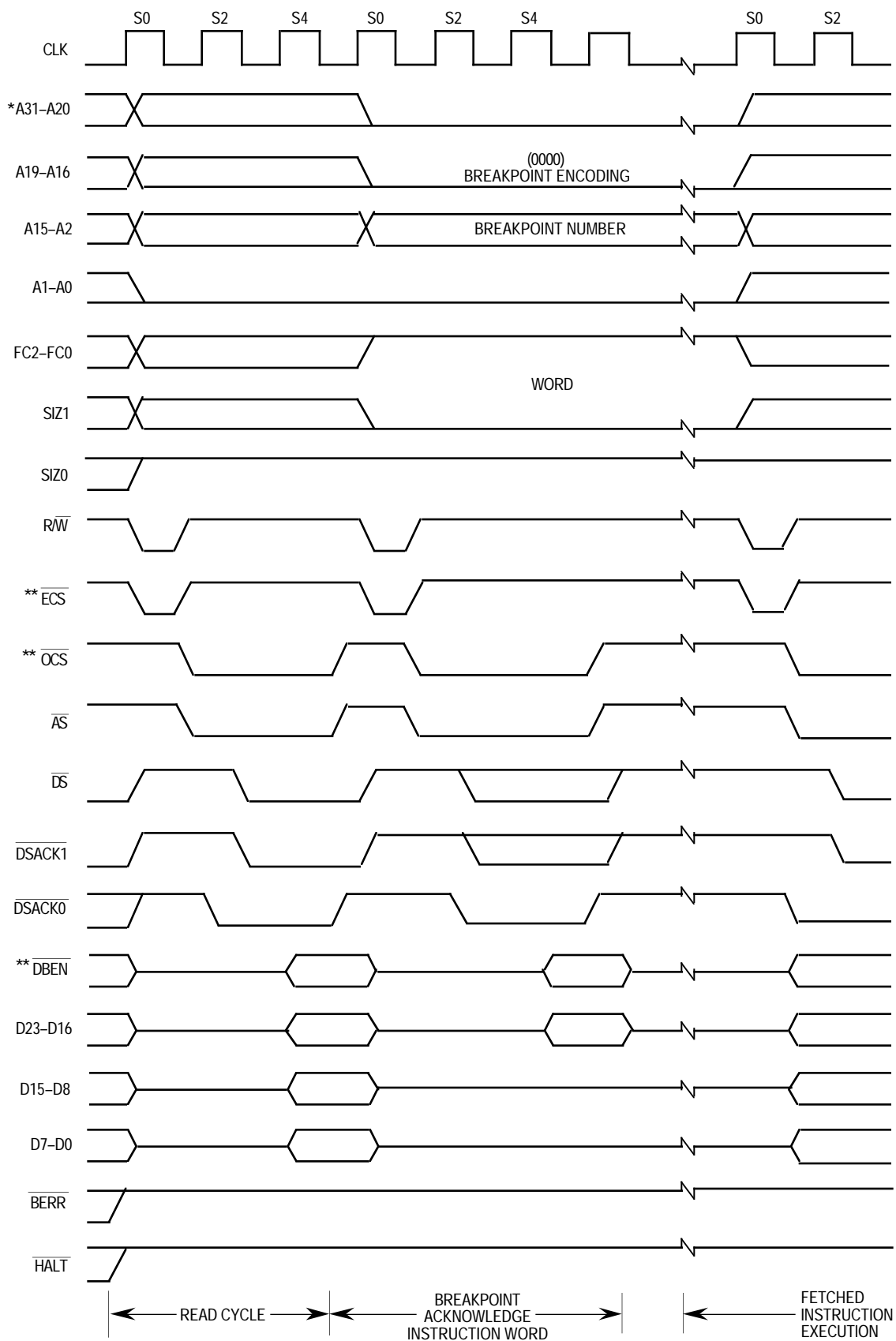
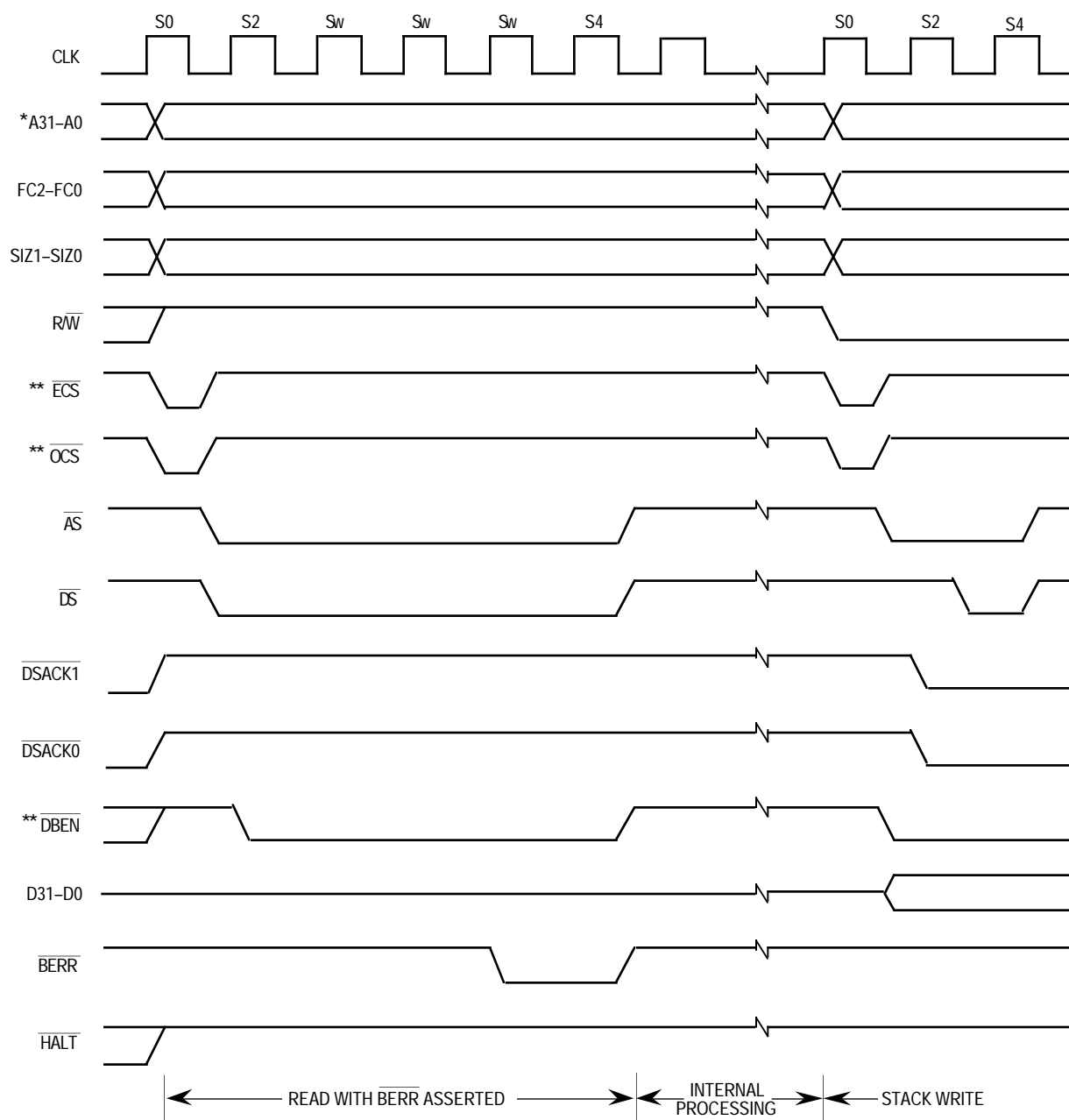


Figure 5-36. Breakpoint Acknowledge Cycle Timing



* For the MC68EC020, A23-A0.

** This signal does not apply to the MC68EC020.

Figure 5-37. Breakpoint Acknowledge Cycle Timing (Exception Signaled)

5.4.3 Coprocessor Communication Cycles

The MC68020/EC020 coprocessor interface provides instruction-oriented communication between the processor and as many as eight coprocessors. Coprocessor accesses use the MC68020/EC020 bus protocol except that the address bus supplies access information rather than a 32-bit address. The CPU space type field (A19–A16) for a coprocessor operation is 0010. A15–A13 contain the coprocessor identification number (CpID), and A5–A0 specify the coprocessor interface register to be accessed. The memory management unit of an MC68020/EC020 system is always identified by a CpID of zero and has an extended register select field (A7–A0) in CPU space 0001 for use by the CALLM and RTM access level checking mechanism. Refer to **Section 9 Applications Information** for more details.

5.5 BUS EXCEPTION CONTROL CYCLES

The MC68020/EC020 bus architecture requires assertion of $\overline{\text{DSACK1/DSACK0}}$ from an external device to signal that a bus cycle is complete. $\overline{\text{DSACK1/DSACK0}}$ or $\overline{\text{AVEC}}$ is not asserted if:

- The external device does not respond,
- No interrupt vector is provided, or
- Various other application-dependent errors occur.

External circuitry can assert $\overline{\text{BERR}}$ when no device responds by asserting $\overline{\text{DSACK1/DSACK0}}$ or $\overline{\text{AVEC}}$ within an appropriate period of time after the processor asserts $\overline{\text{AS}}$. Assertion of $\overline{\text{BERR}}$ allows the cycle to terminate and the processor to enter exception processing for the error condition.

$\overline{\text{HALT}}$ is also used for bus exception control. $\overline{\text{HALT}}$ can be asserted by an external device for debugging purposes to cause single bus cycle operation or can be asserted in combination with $\overline{\text{BERR}}$ to cause a retry of a bus cycle in error.

To properly control termination of a bus cycle for a retry or a bus error condition, $\overline{\text{DSACK1/DSACK0}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$ can be asserted and negated with the rising edge of the MC68020/EC020 clock. This procedure ensures that when two signals are asserted simultaneously, the required setup time (#47A) and hold time (#47B) for both of them is met for the same falling edge of the processor clock. (Refer to **Section 10 Electrical Characteristics** for timing requirements.) This or some equivalent precaution should be designed into the external circuitry that provides these signals.

The acceptable bus cycle terminations for asynchronous cycles are summarized in relation to $\overline{\text{DSACK1/DSACK0}}$ assertion as follows (case numbers refer to Table 5-8):

Normal Termination:

$\overline{\text{DSACK1/DSACK0}}$ is asserted; $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ remain negated (case 1).

Halt Termination:

$\overline{\text{HALT}}$ is asserted at same time or before $\overline{\text{DSACK1/DSACK0}}$, and $\overline{\text{BERR}}$ remains negated (case 2).

Bus Error Termination:

$\overline{\text{BERR}}$ is asserted in lieu of, at the same time, or before $\overline{\text{DSACK1/DSACK0}}$ (case 3) or after $\overline{\text{DSACK1/DSACK0}}$ (case 4), and $\overline{\text{HALT}}$ remains negated; $\overline{\text{BERR}}$ is negated at the same time or after $\overline{\text{DSACK1/DSACK0}}$.

Retry Termination:

$\overline{\text{HALT}}$ and $\overline{\text{BERR}}$ are asserted in lieu of, at the same time, or before $\overline{\text{DSACK1/DSACK0}}$ (case 5) or after $\overline{\text{DSACK1/DSACK0}}$ (case 6); $\overline{\text{BERR}}$ is negated at the same time or after $\overline{\text{DSACK1/DSACK0}}$; $\overline{\text{HALT}}$ may be negated at the same time or after $\overline{\text{BERR}}$.

Table 5-8. $\overline{\text{DSACK1/DSACK0}}$, $\overline{\text{BERR}}$, $\overline{\text{HALT}}$ Assertion Results

| Case No. | Control Signal | Asserted on Rising Edge of State | | Result |
|----------|---|----------------------------------|-------------|--|
| | | n | n+2 | |
| 1 | $\overline{\text{DSACK1/DSACK0}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$ | A N N | S N X | Normal cycle terminate and continue. |
| 2 | $\overline{\text{DSACK1/DSACK0}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$ | A N A/S | S N S | Normal cycle terminate and halt. Continue when $\overline{\text{HALT}}$ negated. |
| 3 | $\overline{\text{DSACK1/DSACK0}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$ | N/A A N | X S N | Terminate and take bus error exception, possibly deferred. |
| 4 | $\overline{\text{DSACK1/DSACK0}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$ | A N N | X A N | Terminate and take bus error exception, possibly deferred. |
| 5 | $\overline{\text{DSACK1/DSACK0}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$ | N/A A A/S | X S S | Terminate and retry when $\overline{\text{HALT}}$ negated. |
| 6 | $\overline{\text{DSACK1/DSACK0}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$ | A N N | X A A | Terminate and retry when $\overline{\text{HALT}}$ negated. |

Legend:

n—The number of current even bus state (e.g., S2, S4, etc.)

A—Signal is asserted in this bus state

N—Signal is not asserted and/or remains negated in this bus state

X—Don't care

S—Signal was asserted in previous state and remains asserted in this state

Table 5-8 lists various combinations of control signal sequences and the resulting bus cycle terminations. To ensure predictable operation, $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ should be negated according to parameters #28 and #57 in **Section 10 Electrical Characteristics**. $\overline{\text{DSACK1/DSACK0}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$ may be negated after $\overline{\text{AS}}$. If $\overline{\text{DSACK1/DSACK0}}$ or $\overline{\text{BERR}}$ remain asserted into S2 of the next bus cycle, that cycle may be terminated prematurely.

Example A:

A system uses a watchdog timer to terminate accesses to an unpopulated address space. The timer asserts $\overline{\text{BERR}}$ after timeout (case 3).

Example B:

A system uses error detection and correction on RAM contents. The designer may:

1. Delay $\overline{\text{DSACK1/DSACK0}}$ assertion until data is verified and assert $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ simultaneously to indicate to the processor to automatically retry the error cycle (case 5) or, if data is valid, assert $\overline{\text{DSACK1/DSACK0}}$ (case 1).
2. Delay $\overline{\text{DSACK1/DSACK0}}$ assertion until data is verified and assert $\overline{\text{BERR}}$ with or without $\overline{\text{DSACK1/DSACK0}}$ if data is in error (case 3). This configuration initiates exception processing for software handling of the condition.
3. Assert $\overline{\text{DSACK1/DSACK0}}$ prior to data verification. If data is invalid, $\overline{\text{BERR}}$ is asserted on the next clock cycle (case 4). This configuration initiates exception processing for software handling of the condition.
4. Assert $\overline{\text{DSACK1/DSACK0}}$ prior to data verification; if data is invalid, assert $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ on the next clock cycle (case 6). The memory controller can then correct the RAM prior to or during the automatic retry.

5.5.1 Bus Errors

The $\overline{\text{BERR}}$ signal can be used to abort the bus cycle and the instruction being executed. $\overline{\text{BERR}}$ takes precedence over $\overline{\text{DSACK1/DSACK0}}$, provided it meets the timing constraints described in **Section 10 Electrical Characteristics**. If $\overline{\text{BERR}}$ does not meet these constraints, it may cause unpredictable operation of the MC68020/EC020. If $\overline{\text{BERR}}$ remains asserted into the next bus cycle, it may cause incorrect operation of that cycle.

When $\overline{\text{BERR}}$ is issued to terminate a bus cycle, the MC68020/EC020 may enter exception processing immediately following the bus cycle, or it may defer processing the exception. The instruction prefetch mechanism requests instruction words from the bus controller and the instruction cache before it is ready to execute them. If a bus error occurs on an instruction fetch, the processor does not take the exception until it attempts to use that instruction word. Should an intervening instruction cause a branch or should a task switch occur, the bus error exception does not occur.

$\overline{\text{BERR}}$ is recognized during a bus cycle in any of the following cases:

1. $\overline{\text{DSACK1/DSACK0}}$ and $\overline{\text{HALT}}$ are negated and $\overline{\text{BERR}}$ is asserted.
2. $\overline{\text{HALT}}$ and $\overline{\text{BERR}}$ are negated and $\overline{\text{DSACK1/DSACK0}}$ is asserted. $\overline{\text{BERR}}$ is then asserted within one clock cycle ($\overline{\text{HALT}}$ remains negated).
3. $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ are asserted (see **5.5.2 Retry Operation**).

When the processor recognizes a bus error condition, it terminates the current bus cycle in the normal way. Figure 5-38 shows the timing of a bus error for the case in which $\overline{\text{DSACK1/DSACK0}}$ is not asserted. Figure 5-39 shows the timing for a bus error for the case in which $\overline{\text{BERR}}$ is asserted after $\overline{\text{DSACK1/DSACK0}}$. Exceptions are taken in both cases. (Refer to **Section 6 Exception Processing** for details of bus error exception processing.) When $\overline{\text{BERR}}$ is asserted during a read cycle that supplies an instruction to the on-chip cache, the instruction in the cache is marked invalid.

When $\overline{\text{BERR}}$ is asserted after $\overline{\text{DSACK1/DSACK0}}$, $\overline{\text{BERR}}$ must be asserted within parameter #48 (refer to **Section 10 Electrical Characteristics**) for purely asynchronous operation, or it must be asserted and remain stable during the sample window, defined by parameters #27A and #47B, around the next falling edge of the clock after $\overline{\text{DSACK1/DSACK0}}$ is recognized. If $\overline{\text{BERR}}$ is not stable at this time, the processor may exhibit erratic behavior. In this case, data may be present on the bus, but may not be valid. This sequence may be used by systems that have memory error detection and correction logic and by external cache memories.

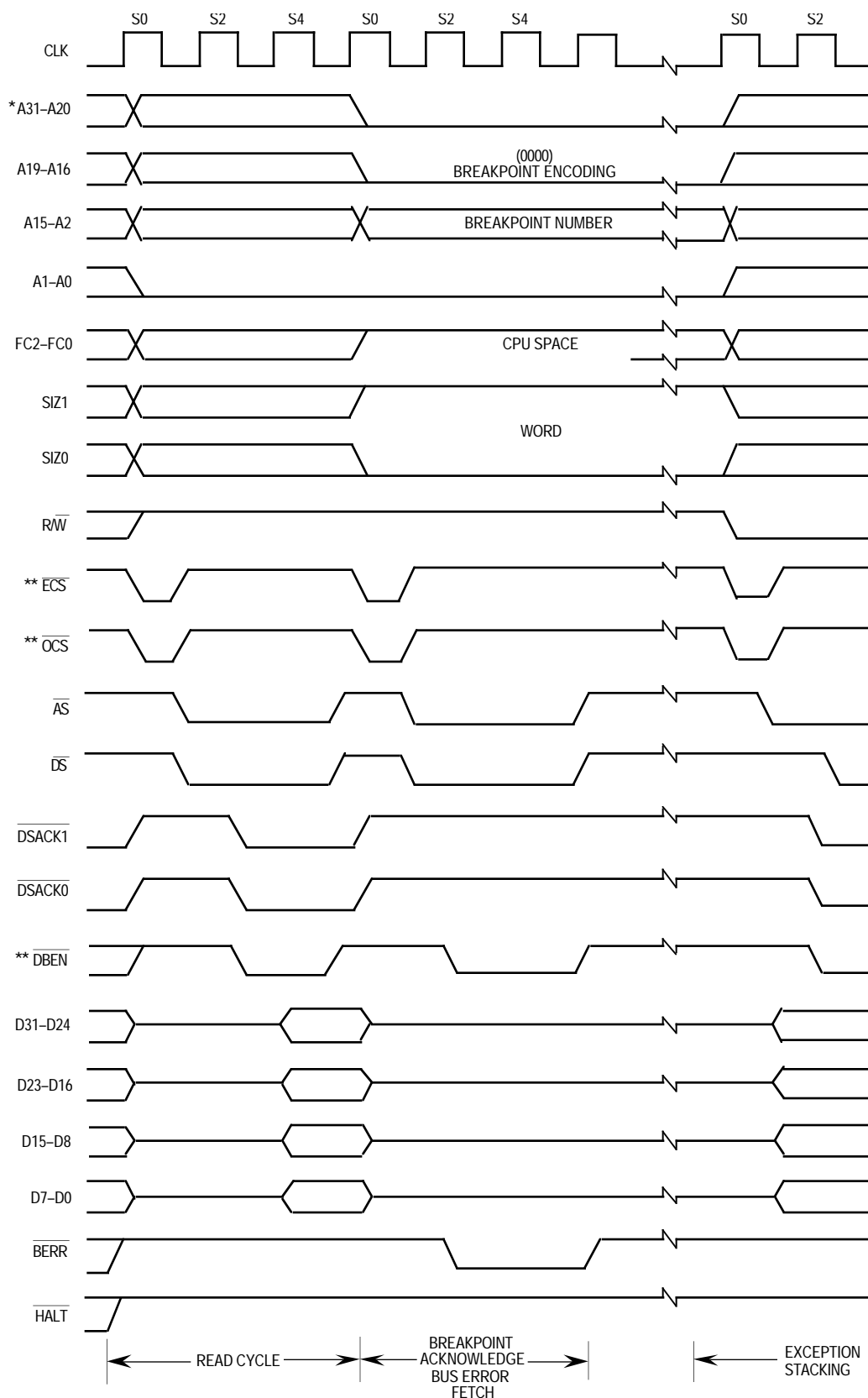
5.5.2 Retry Operation

When $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ are asserted simultaneously by an external device during a bus cycle, the processor enters the retry sequence. A delayed retry similar to the delayed $\overline{\text{BERR}}$ signal described previously can also occur.

The processor terminates the bus cycle, negates the control signals ($\overline{\text{AS}}$, $\overline{\text{DS}}$, $\text{R}/\overline{\text{W}}$, SIZ1 , SIZ0 , $\overline{\text{RMC}}$, and, for the MC68020 only, $\overline{\text{ECS}}$ and $\overline{\text{OCS}}$), and does not begin another bus cycle until the $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ signals have been negated by external logic. After a synchronization delay, the processor retries the previous cycle using the same access information (address, function code, size, etc.) The $\overline{\text{BERR}}$ signal should be negated before S2 of the read cycle to ensure correct operation of the retried cycle. Figure 5-40 shows a late retry operation of a cycle.

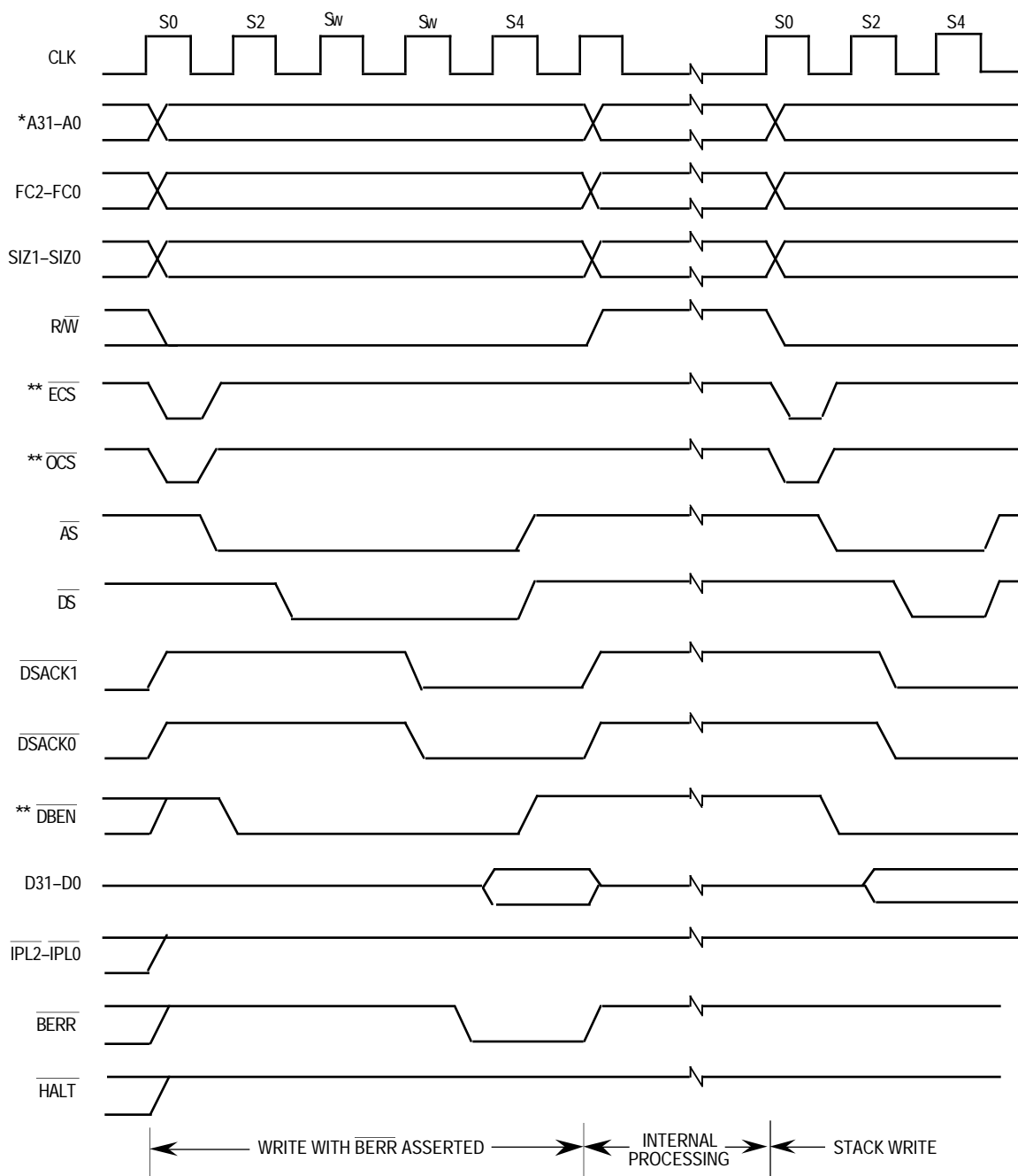
The processor retries any read or write cycle of a read-modify-write operation separately; $\overline{\text{RMC}}$ remains asserted during the entire retry sequence.

Asserting $\overline{\text{BR}}$ along with $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ provides a relinquish and retry operation. The MC68020/EC020 does not relinquish the bus during a read-modify-write operation. Any device that requires the processor to give up the bus and retry a bus cycle during a read-modify-write cycle must assert $\overline{\text{BERR}}$ and $\overline{\text{BR}}$ only ($\overline{\text{HALT}}$ must not be included). The bus error handler software should examine the read-modify-write bit in the special status word (refer to **Section 6 Exception Processing**) and take the appropriate action to resolve this type of fault when it occurs.



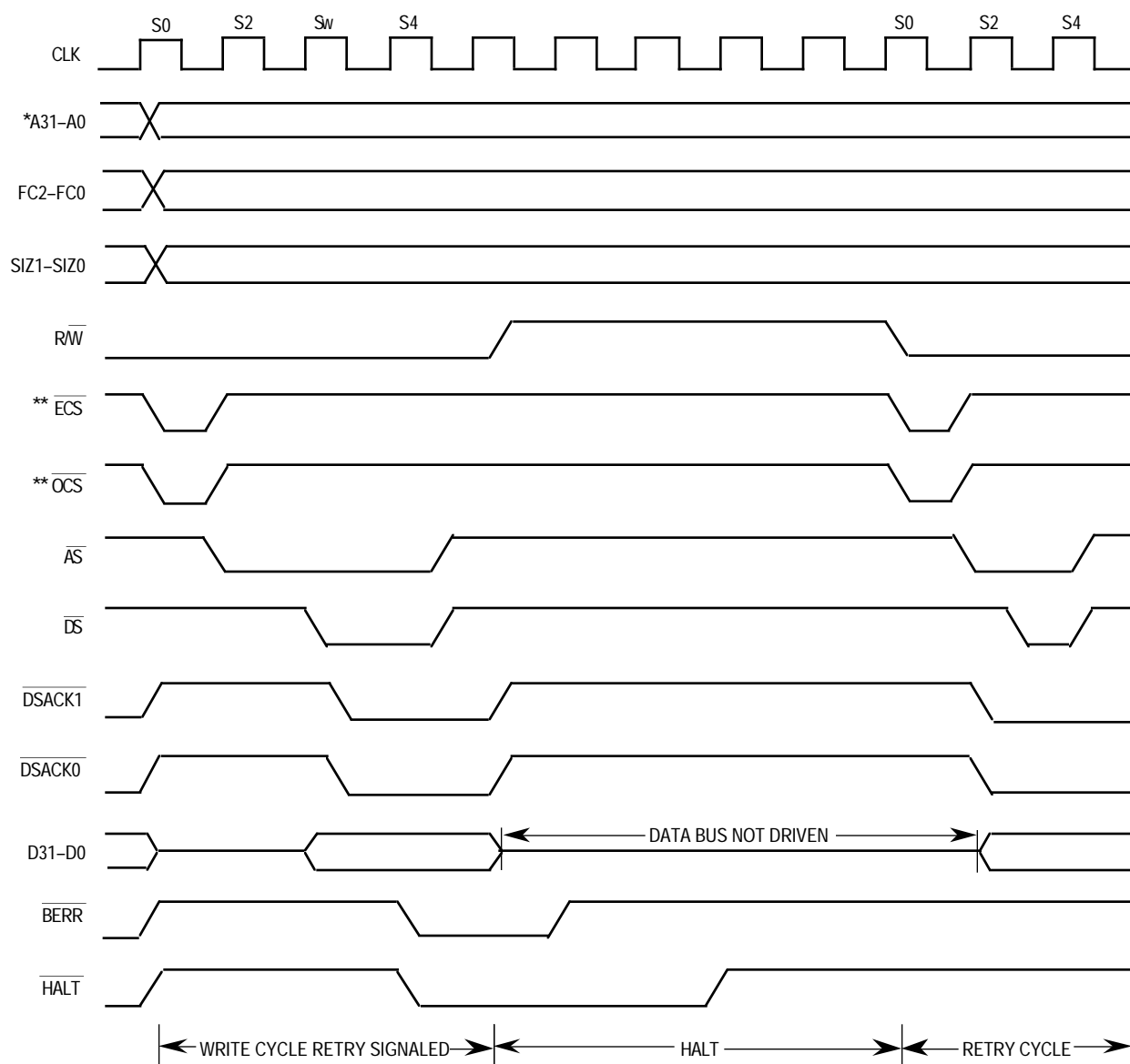
* For the MC68EC020, A23-A20.
 ** This signal does not apply to the MC68EC020.

Figure 5-38. Bus Error without DSACK1/DSACK0



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

Figure 5-39. Late Bus Error with DSACK1/DSACK0



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

Figure 5-40. Late Retry

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/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/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/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.

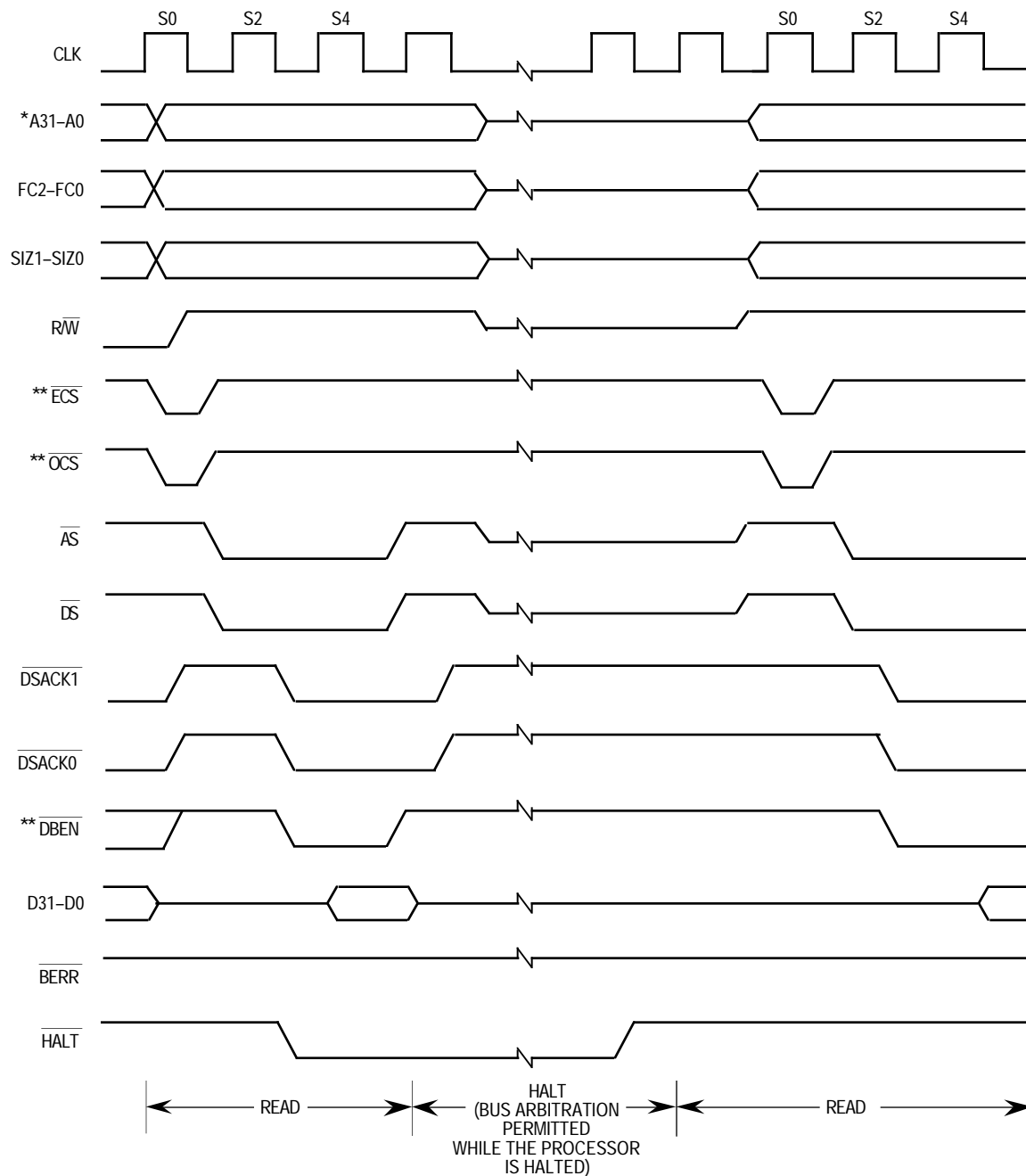


Figure 5-41. Halt Operation Timing

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.

5.7.1 MC68020 Bus Arbitration

The sequence of the MC68020 bus arbitration protocol is as follows:

1. An external device asserts the \overline{BR} signal.
2. The processor asserts the \overline{BG} signal to indicate that the bus will become available at the end of the current bus cycle.
3. The external device asserts the \overline{BGACK} signal to indicate that it has assumed bus mastership.

\overline{BR} may be issued any time during a bus cycle or between cycles. \overline{BG} is asserted in response to \overline{BR} ; it is usually asserted as soon as \overline{BR} has been synchronized and recognized, except when the MC68020 has made an internal decision to execute a bus cycle. Then, the assertion of \overline{BG} is deferred until the bus cycle has begun. Additionally, \overline{BG} is not asserted until the end of a read-modify-write operation (when \overline{RMC} is negated) in response to a \overline{BR} signal. When the requesting device receives \overline{BG} and more than one external device can be bus master, the requesting device should begin whatever arbitration is required. The external device asserts \overline{BGACK} when it assumes bus mastership, and maintains \overline{BGACK} during the entire bus cycle (or cycles) for which it is bus master. The following conditions must be met for an external device to assume mastership of the bus through the normal bus arbitration procedure:

- The external device must have received \overline{BG} through the arbitration process.
- \overline{AS} must be negated, indicating that no bus cycle is in progress, and the external device must ensure that all appropriate processor signals have been placed in the high-impedance state (by observing specification #7 in **Section 10 Electrical Specifications**).
- The termination signal ($\overline{DSACK1}/\overline{DSACK0}$) for the most recent cycle must have been negated, indicating that external devices are off the bus (optional, refer to **5.7.1.3 Bus Grant Acknowledge (MC68020)**).
- \overline{BGACK} must be inactive, indicating that no other bus master has claimed ownership of the bus.

Figure 5-42 is a flowchart of MC68020 bus arbitration for a single device. Figure 5-43 is a timing diagram for the same operation. This technique allows processing of bus requests during data transfer cycles.

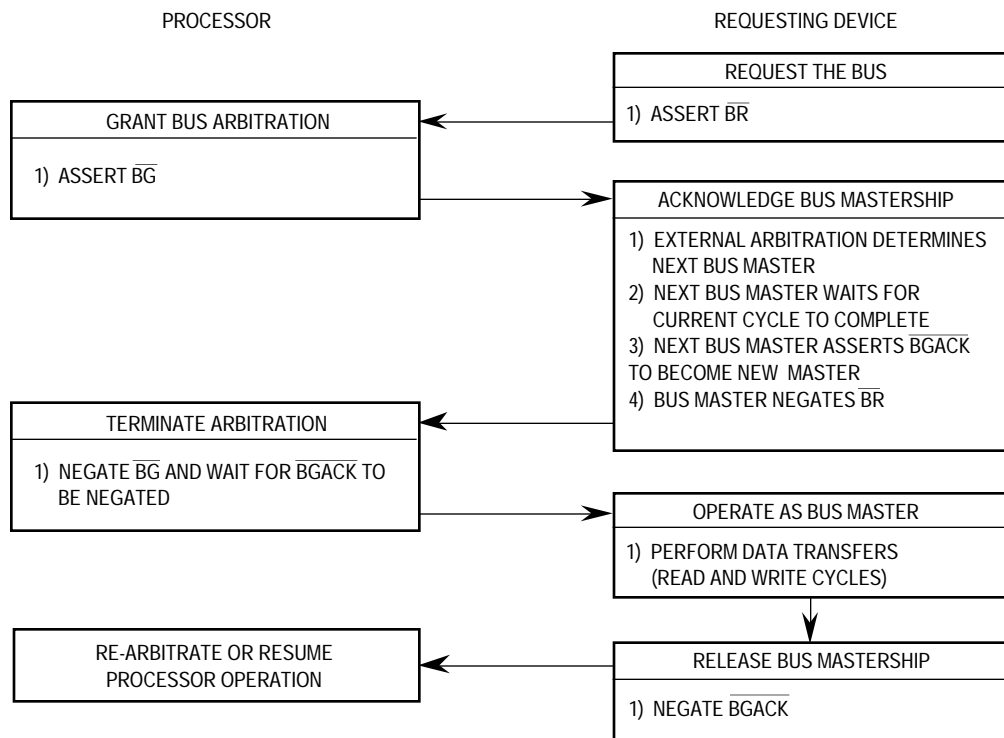


Figure 5-42. MC68020 Bus Arbitration Flowchart for Single Request

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.

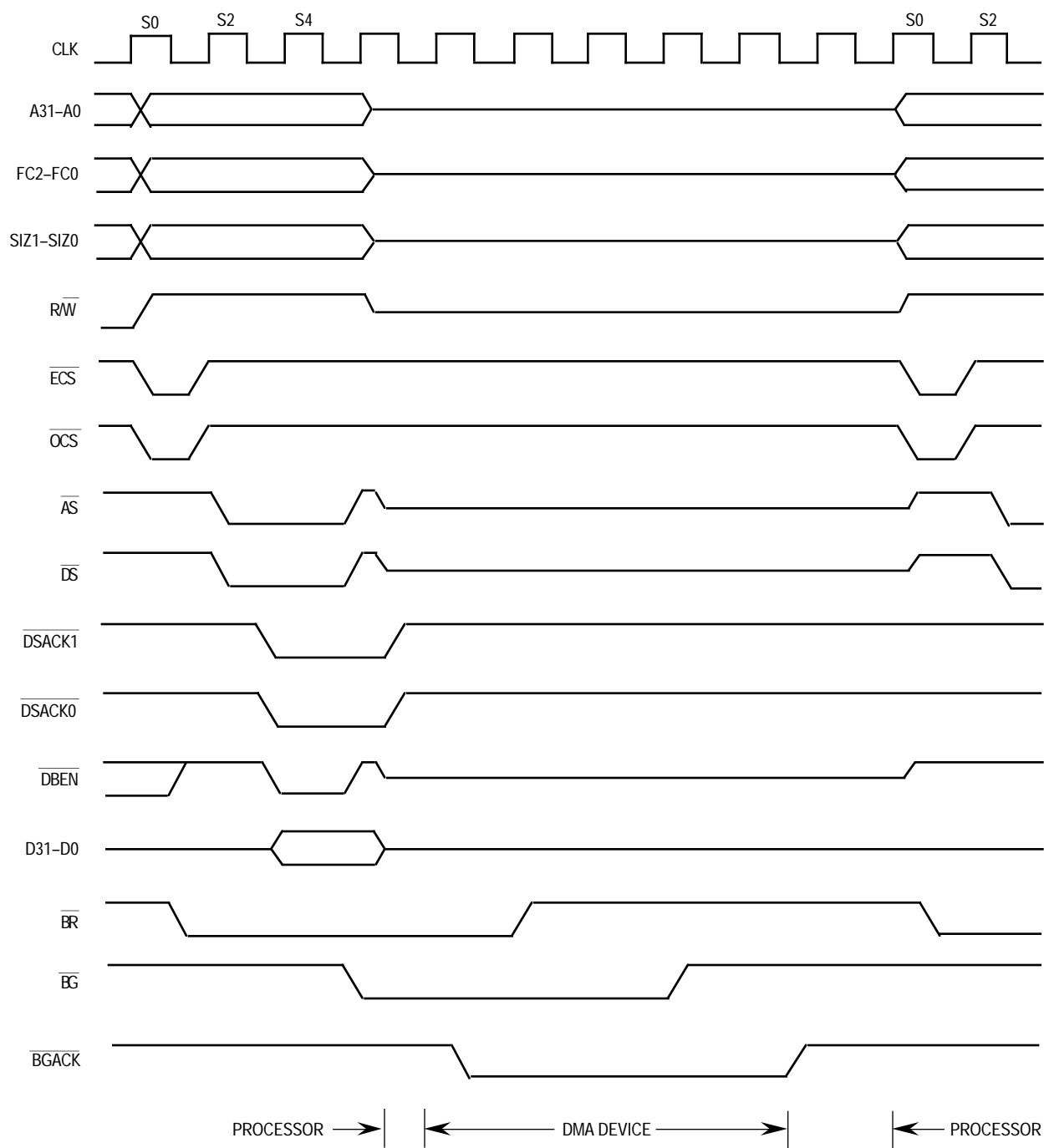


Figure 5-43. MC68020 Bus Arbitration Operation Timing for Single Request

5.7.1.1 BUS REQUEST (MC68020). External devices capable of becoming bus masters request the bus by asserting \overline{BR} . \overline{BR} can be a wire-ORed signal (although it need not be constructed from open-collector devices) that indicates to the processor that some external device requires control of the bus. The processor is at a lower bus priority level than the external device and relinquishes the bus after it has completed the current bus cycle (if one has started).

If no \overline{BGACK} is received while \overline{BR} is asserted, the processor remains bus master once \overline{BR} is negated. This prevents unnecessary interference with ordinary processing if the arbitration circuitry inadvertently responds to noise or if an external device determines that it no longer requires use of the bus before it has been granted mastership.

5.7.1.2 BUS GRANT (MC68020). The processor asserts \overline{BG} as soon as possible after receipt of the bus request. \overline{BG} assertion immediately follows internal synchronization except during a read-modify-write cycle or follows an internal decision to execute a bus cycle. During a read-modify-write cycle, the processor does not assert \overline{BG} until the entire operation has completed. \overline{RMC} is asserted to indicate that the bus is locked. In the case of an internal decision to execute another bus cycle, \overline{BG} is deferred until the bus cycle has begun.

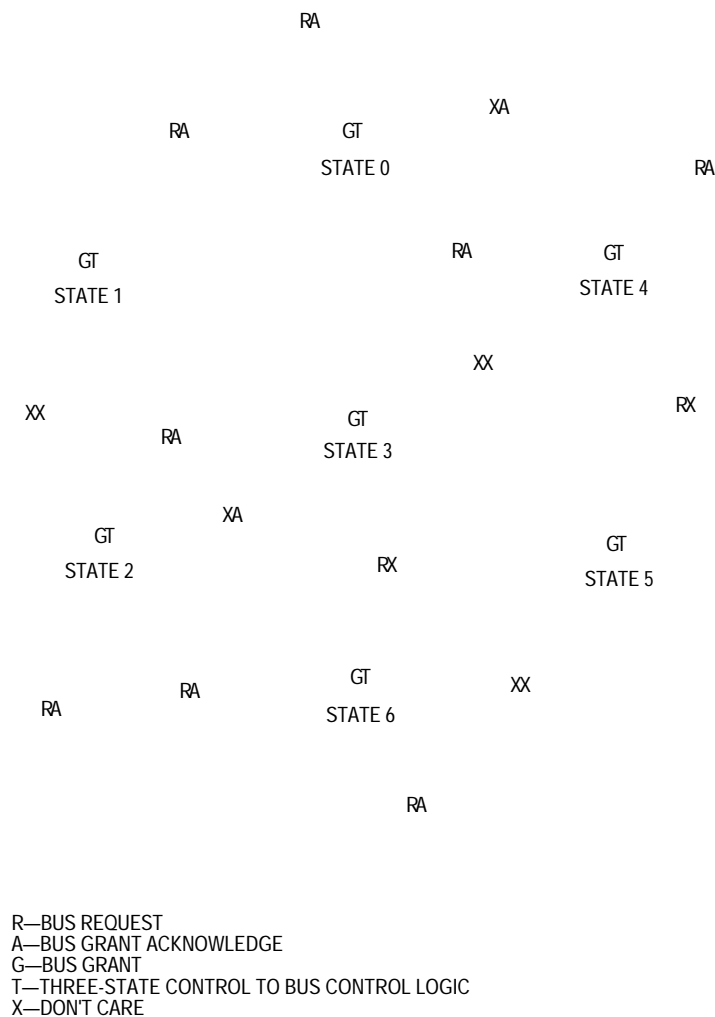
\overline{BG} may be routed through a daisy-chained network or through a specific priority-encoded network. The processor allows any type of external arbitration that follows the protocol.

5.7.1.3 BUS GRANT ACKNOWLEDGE (MC68020). Upon receiving \overline{BG} , the requesting device waits until \overline{AS} , $\overline{DSACK1/DSACK0}$, and \overline{BGACK} are negated before asserting its own \overline{BGACK} . The negation of \overline{AS} indicates that the previous master releases the bus after specification #7 (refer to **Section 10 Electrical Characteristics**). The negation of $\overline{DSACK1/DSACK0}$ indicates that the previous slave has completed its cycle with the previous master. Note that in some applications, $\overline{DSACK1/DSACK0}$ might not be used in this way.

General-purpose devices are connected to be dependent only on \overline{AS} . When \overline{BGACK} is asserted, the device is the bus master until it negates \overline{BGACK} . \overline{BGACK} should not be negated until all bus cycles required by the alternate bus master have been completed. Bus mastership terminates at the negation of \overline{BGACK} . The \overline{BR} from the granted device should be negated after \overline{BGACK} is asserted. If another \overline{BR} is still pending after the assertion of \overline{BGACK} , another \overline{BG} is asserted within a few clocks of the negation of the first \overline{BG} , as described in **5.7.1.4 Bus Arbitration Control (MC68020)**. Note that the processor does not perform any external bus cycles before it reasserts \overline{BG} in this case.

5.7.1.4 BUS ARBITRATION CONTROL (MC68020). The bus arbitration control unit in the MC68020 is implemented with a finite state machine. As discussed previously, all asynchronous inputs to the MC68020 are internally synchronized in a maximum of two cycles of the processor clock.

As shown in Figure 5-44, input signals labeled R and A are internally synchronized versions of the \overline{BR} and \overline{BGACK} signals, respectively. The \overline{BG} output is labeled G, and the internal high-impedance control signal is labeled T. If T is true, the address, data, and control buses are placed in the high-impedance state after the next rising edge following the negation of \overline{AS} and \overline{RMC} . All signals are shown in positive logic (active high), regardless of their true active voltage level.



NOTE: The BG output will not be asserted while RMC is asserted.

Figure 5-44. MC68020 Bus Arbitration State Diagram

State changes occur on the next rising edge of the clock after the internal signal is recognized as valid. The \overline{BG} signal transitions on the falling edge of the clock after a state is reached during which G changes. The bus control signals (controlled by T) are driven by the processor immediately following a state change when bus mastership is returned to the MC68020.

State 0, at the top center of the diagram, in which both G and T are negated, is the state of the bus arbiter while the processor is bus master. Request R and acknowledge A keep the arbiter in state 0 as long as they are both negated. When a request R is received, both grant G and signal T are asserted (in state 1 at the top left). The next clock causes a change to state 2, at the lower left, in which G and T are held. The bus arbiter remains in that state until acknowledge A is asserted or request R is negated. Once either occurs, the arbiter changes to the center state, state 3, and negates grant G. The next clock takes the arbiter to state 4, at the upper right, in which grant G remains negated and signal T remains asserted. With acknowledge A asserted, the arbiter remains in state 4 until A is negated or request R is again asserted. When A is negated, the arbiter returns to the original state, state 0, and negates signal T. This sequence of states follows the normal sequence of signals for relinquishing the bus to an external bus master. Other states apply to other possible sequences of combinations of R and A.

The MC68020 does not allow arbitration of the external bus during the read-modify-write sequence. For the duration of this sequence, the MC68020 ignores the \overline{BR} input. If mastership of the MC68020 bus is required during a read-modify-write operation, \overline{BERR} must be used to abort the read-modify-write sequence. The bus arbitration sequence while the bus is inactive (i.e., executing internal operations such as a multiply instruction) is shown in Figure 5-45.

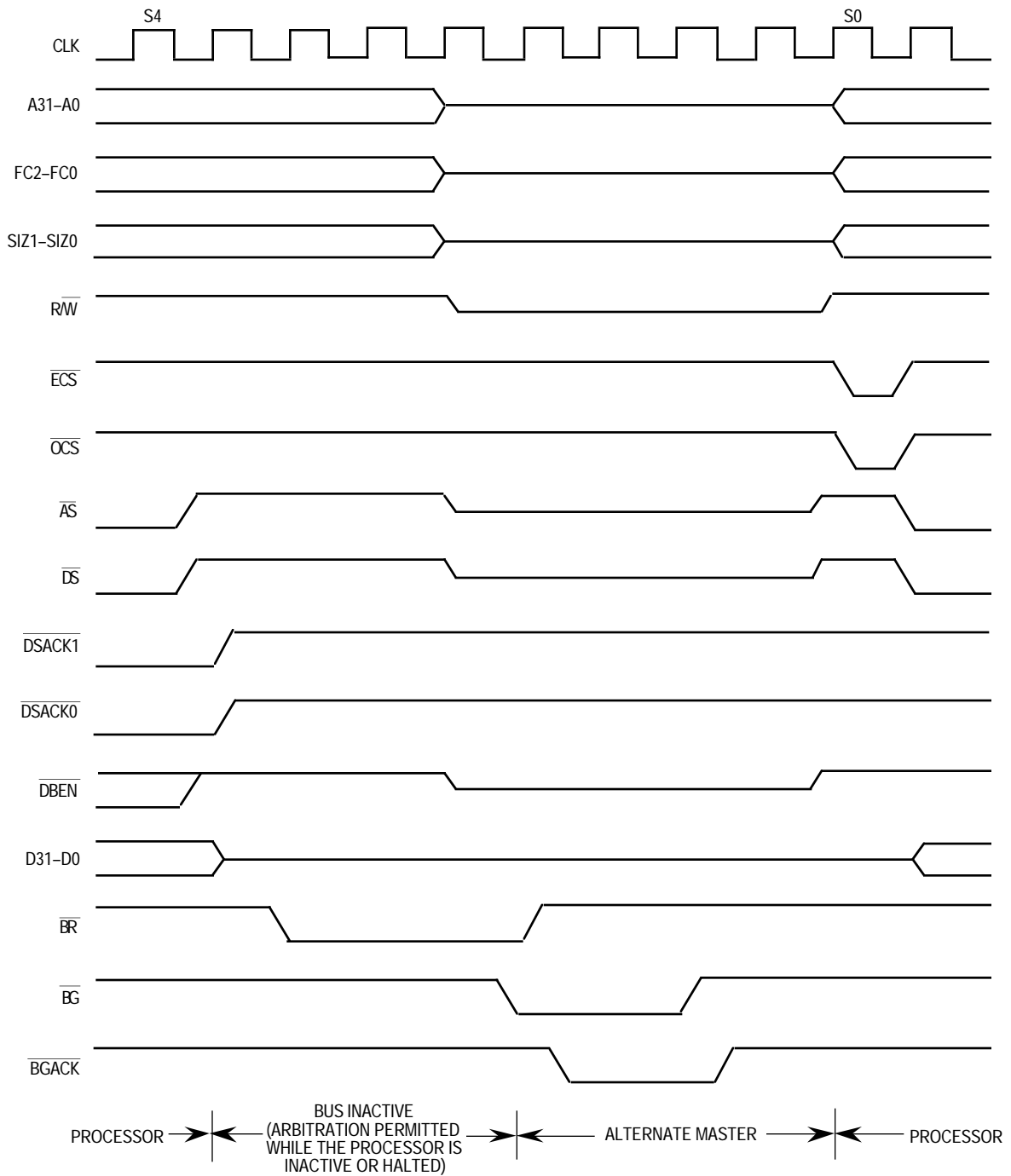


Figure 5-45. MC68020 Bus Arbitration Operation Timing—Bus Inactive

5.7.2 MC68EC020 Bus Arbitration

The sequence of the MC68EC020 bus arbitration protocol is as follows:

1. An external device asserts the \overline{BR} signal.
2. The processor asserts the \overline{BG} signal to indicate that the bus will become available at the end of the current bus cycle.
3. The external device asserts the \overline{BR} signal throughout its bus mastership.

\overline{BR} may be issued any time during a bus cycle or between cycles. \overline{BG} is asserted in response to \overline{BR} ; it is usually asserted as soon as \overline{BR} has been synchronized and recognized, except when the MC68020 has made an internal decision to execute a bus cycle. Then, the assertion of \overline{BG} is deferred until the bus cycle has begun. Additionally, \overline{BG} is not asserted until the end of a read-modify-write operation (when \overline{RMC} is negated) in response to a \overline{BR} signal. When the requesting device receives \overline{BG} and more than one external device can be bus master, the requesting device should begin whatever arbitration is required. The external device continues to assert \overline{BR} when it assumes bus mastership, and maintains \overline{BR} during the entire bus cycle (or cycles) for which it is bus master. The following conditions must be met for an external device to assume mastership of the bus through the normal bus arbitration procedure:

- The external device must have received \overline{BG} through the arbitration process.
- \overline{AS} must be negated, indicating that no bus cycle is in progress, and the external device must ensure that all appropriate processor signals have been placed in the high-impedance state (by observing specification #7 in **Section 10 Electrical Specifications**).
- The termination signal ($\overline{DSACK1}/\overline{DSACK0}$) for the most recent cycle must have been negated, indicating that external devices are off the bus.
- No other bus master has claimed ownership of the bus.

Figure 5-46 is a flowchart of MC68EC020 bus arbitration for a single device. Figure 5-47 is a timing diagram for the same operation. This technique allows processing of bus requests during data transfer cycles.

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.

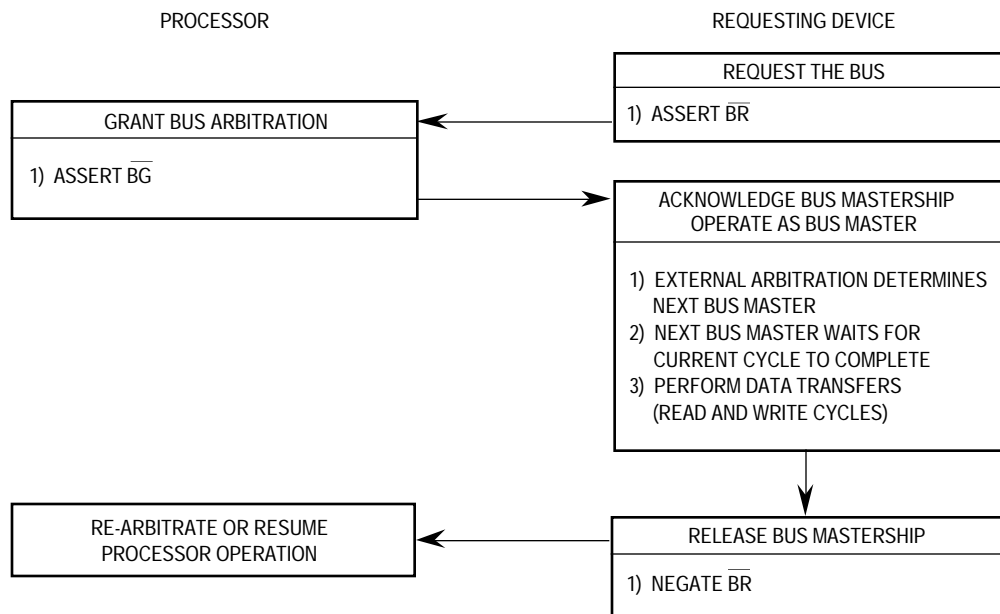


Figure 5-46. MC68EC020 Bus Arbitration Flowchart for Single Request

5.7.2.1 BUS REQUEST (MC68EC020). External devices capable of becoming bus masters request the bus by asserting \overline{BR} . \overline{BR} can be a wire-ORed signal (although it need not be constructed from open-collector devices) that indicates to the processor that some external device requires control of the bus. The processor is at a lower bus priority level than the external device and relinquishes the bus after it has completed the current bus cycle (if one has started). \overline{BR} remains asserted throughout the external device's bus mastership.

5.7.2.2 BUS GRANT (MC68EC020). The processor asserts \overline{BG} as soon as possible after receipt of the bus request. \overline{BG} assertion immediately follows internal synchronization except during a read-modify-write cycle or follows an internal decision to execute a bus cycle. During a read-modify-write cycle, the processor does not assert \overline{BG} until the entire operation has completed. \overline{RMC} is asserted to indicate that the bus is locked. In the case of an internal decision to execute another bus cycle, \overline{BG} is deferred until the bus cycle has begun.

\overline{BG} may be routed through a daisy-chained network or through a specific priority-encoded network. The processor allows any type of external arbitration that follows the protocol.

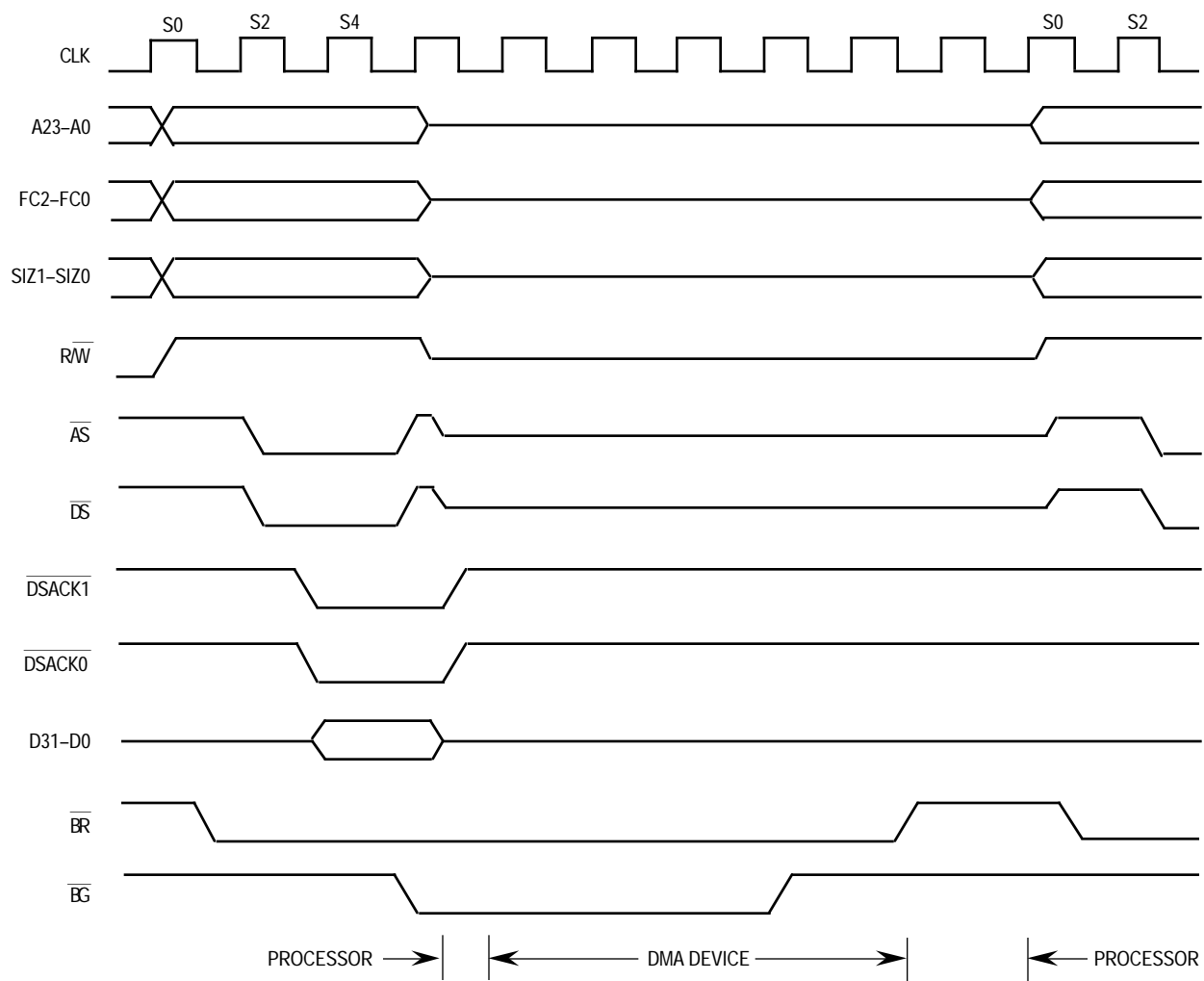


Figure 5-47. MC68EC020 Bus Arbitration Operation Timing for Single Request

5.7.2.3 BUS ARBITRATION CONTROL (MC68EC020). The bus arbitration control unit in the MC68EC020 is implemented with a finite state machine. As discussed previously, all asynchronous inputs to the MC68EC020 are internally synchronized in a maximum of two cycles of the processor clock.

As shown in Figure 5-48, the input signal labeled R is an internally synchronized version of the \overline{BR} signal. The \overline{BG} output is labeled G, and the internal high-impedance control signal is labeled T. If T is true, the address, data, and control buses are placed in the high-impedance state after the next rising edge following the negation of \overline{AS} and \overline{RMC} . All signals are shown in positive logic (active high), regardless of their true active voltage level.

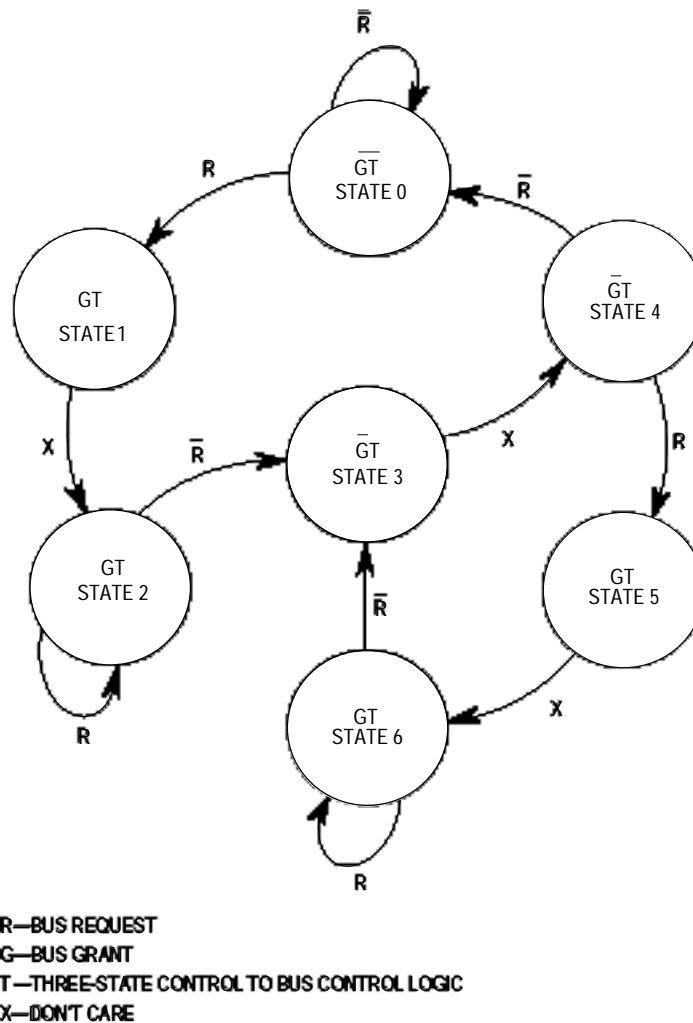


Figure 5-48. MC68EC020 Bus Arbitration State Diagram

State changes occur on the next rising edge of the clock after the internal signal is recognized as valid. The \overline{BG} signal transitions on the falling edge of the clock after a state is reached during which G changes. The bus control signals (controlled by T) are driven by the processor immediately following a state change when bus mastership is returned to the MC68EC020.

State 0, at the top center of the diagram, in which both G and T are negated, is the state of the bus arbiter while the processor is bus master. Request R keeps the arbiter in state 0 as long as it is negated. When a request R is received, both grant G and signal T are asserted (in state 1 at the top left). The next clock causes a change to state 2, at the lower left, in which G and T are held. The bus arbiter remains in that state until request R is negated. Then the arbiter changes to the center state, state 3, and negates grant G. The next clock takes the arbiter to state 4, at the upper right, in which grant G remains negated and signal T remains asserted. The arbiter returns to the original state, state 0, and negates signal T. This sequence of states follows the normal sequence of signals for relinquishing the bus to an external bus master. Other states apply to other possible sequences of R.

The MC68EC020 does not allow arbitration of the external bus during the read-modify-write sequence. For the duration of this sequence, the MC68EC020 ignores the \overline{BR} input. If mastership of the MC68EC020 bus is required during a read-modify-write operation, \overline{BERR} must be used to abort the read-modify-write sequence. The bus arbitration sequence while the bus is inactive (i.e., executing internal operations such as a multiply instruction) is shown in Figure 5-49.

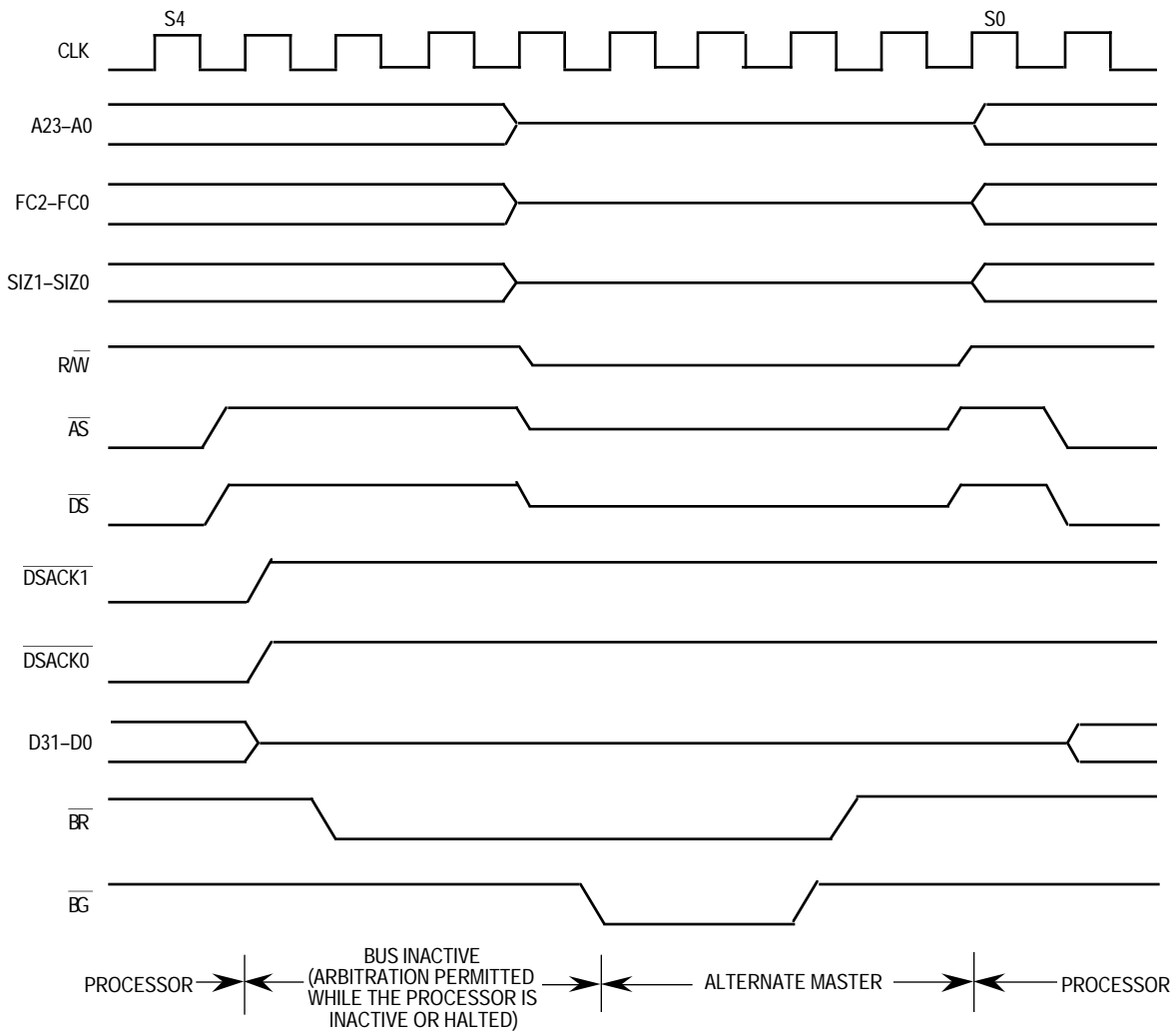


Figure 5-49. MC68EC020 Bus Arbitration Operation Timing—Bus Inactive

The existing three-wire arbitration design (\overline{BR} , \overline{BG} , and \overline{BGACK}) of some peripherals can be converted to the MC68EC020 two-wire arbitration with the addition of an AND gate. Figure 5-50 shows the combination of \overline{BR} and \overline{BGACK} for a three-wire arbitration system to \overline{BR} of the MC68EC020 or \overline{BR} and \overline{BG} from an MC68EC020 to \overline{BG} for a three-wire arbitration system. The speed of the AND gate must be faster than the time between the assertion of \overline{BGACK} and the negation of \overline{BR} by the alternate bus master. Figure 5-50 assumes the alternate bus master does not assume bus mastership until the MC68EC020 \overline{AS} is negated and MC68EC020 \overline{BG} is asserted.

An example of MC68EC020 bus arbitration to a DMA device that supports three-wire bus arbitration is described in **Appendix A Interfacing an MC68EC020 to a DMA Device That Supports a Three-Wire Bus Arbitration Protocol**.

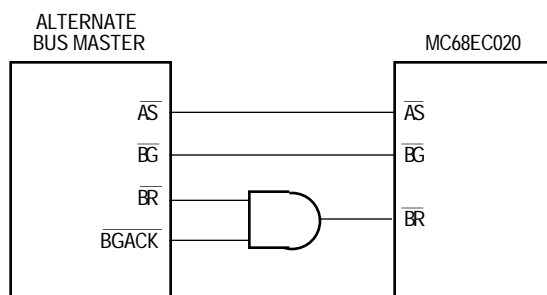


Figure 5-50. Interface for Three-Wire to Two-Wire Bus Arbitration

5.8 RESET OPERATION

$\overline{\text{RESET}}$ is a bidirectional signal with which an external device resets the system or the processor resets external devices. When power is applied to the system, external circuitry should assert $\overline{\text{RESET}}$ for a minimum of 520 clocks after V_{CC} and clock timing have stabilized and are within specification limits. Figure 5-51 is a timing diagram of the power-up reset operation, showing the relationships between $\overline{\text{RESET}}$, V_{CC} , and bus signals. The clock signal is required to be stable by the time V_{CC} reaches the minimum operating specification. During the reset period, the entire bus three-states (except for non-three-statable signals, which are driven to their inactive state). Once $\overline{\text{RESET}}$ negates, all control signals are negated, the data bus is in read mode, and the address bus is driven. After this, the first bus cycle for reset exception processing begins.

The external $\overline{\text{RESET}}$ signal resets the processor and the entire system. Except for the initial reset, $\overline{\text{RESET}}$ should be asserted for at least 520 clock periods to ensure that the processor resets. Asserting $\overline{\text{RESET}}$ for 10 clock periods is sufficient for resetting the processor logic; the additional clock periods prevent a RESET instruction from overlapping the external $\overline{\text{RESET}}$ signal.

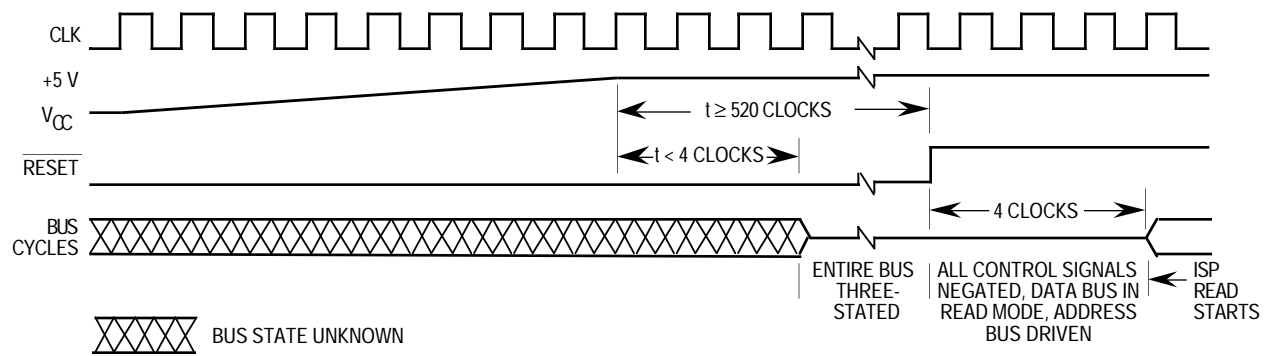
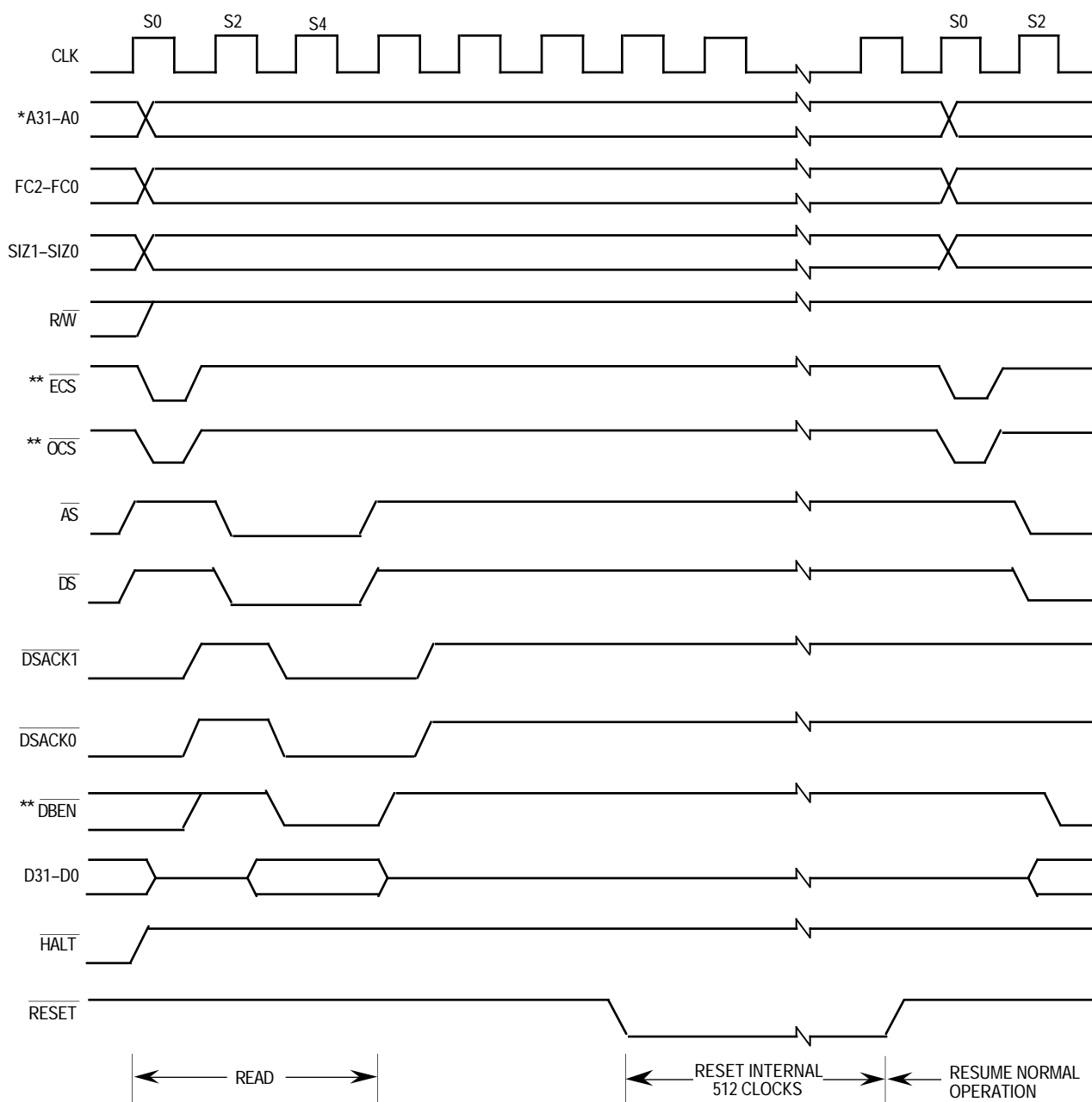


Figure 5-51. Initial Reset Operation Timing

Resetting the processor causes any bus cycle in progress to terminate as if $\overline{\text{DSACK1/DSACK0}}$ 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.



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

Figure 5-52. RESET Instruction Timing