Welcome to E-XFL.COM

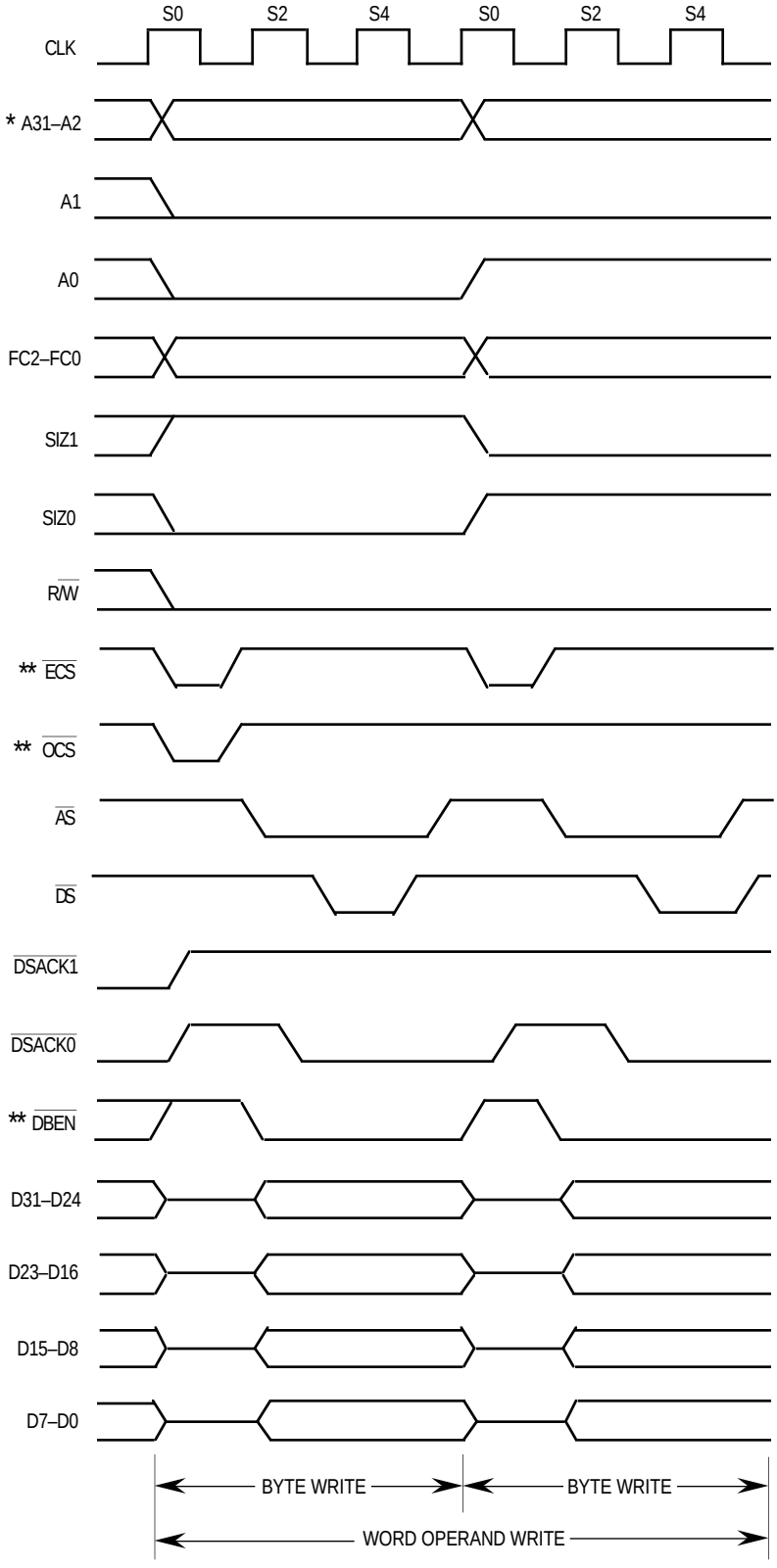
Understanding Embedded - Microprocessors

Embedded microprocessors are specialized computing chips designed to perform specific tasks within an embedded system. Unlike general-purpose microprocessors found in personal computers, embedded microprocessors are tailored for dedicated functions within larger systems, offering optimized performance, efficiency, and reliability. These microprocessors are integral to the operation of countless electronic devices, providing the computational power necessary for controlling processes, handling data, and managing communications.

Applications of Embedded - Microprocessors

Embedded microprocessors are utilized across a broad spectrum of applications, making them indispensable in

Details	
Product Status	Obsolete
Core Processor	68020
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	16MHz
Co-Processors/DSP	-
RAM Controllers	-
Graphics Acceleration	No
Display & Interface Controllers	-
Ethernet	-
SATA	-
USB	-
Voltage - I/O	5.0V
Operating Temperature	0°C ~ 70°C (TA)
Security Features	-
Package / Case	114-BPGA
Supplier Device Package	114-PGA (34.55x34.55)
Purchase URL	https://www.e-xfl.com/product-detail/nxp-semiconductors/mc68020rc16e



* 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.

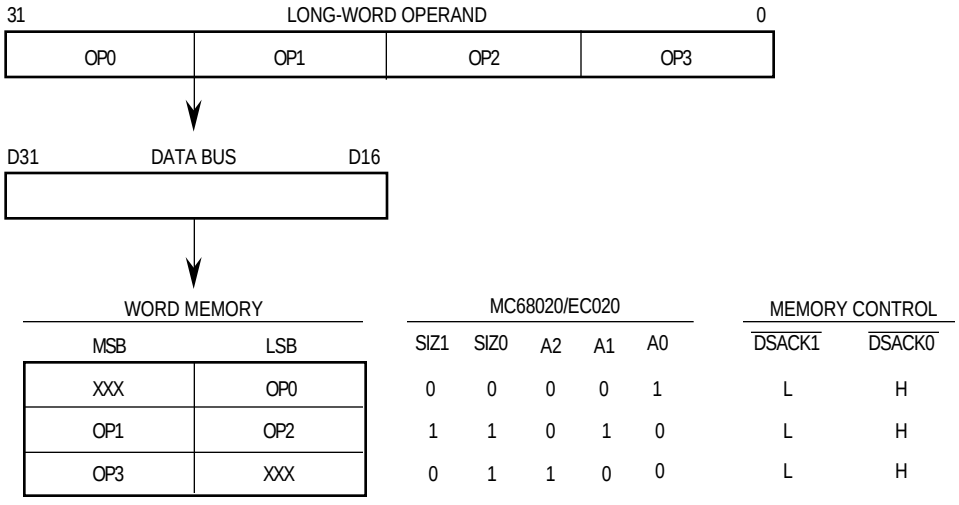


Figure 5-9. Misaligned Long-Word Operand Write to Word Port Example

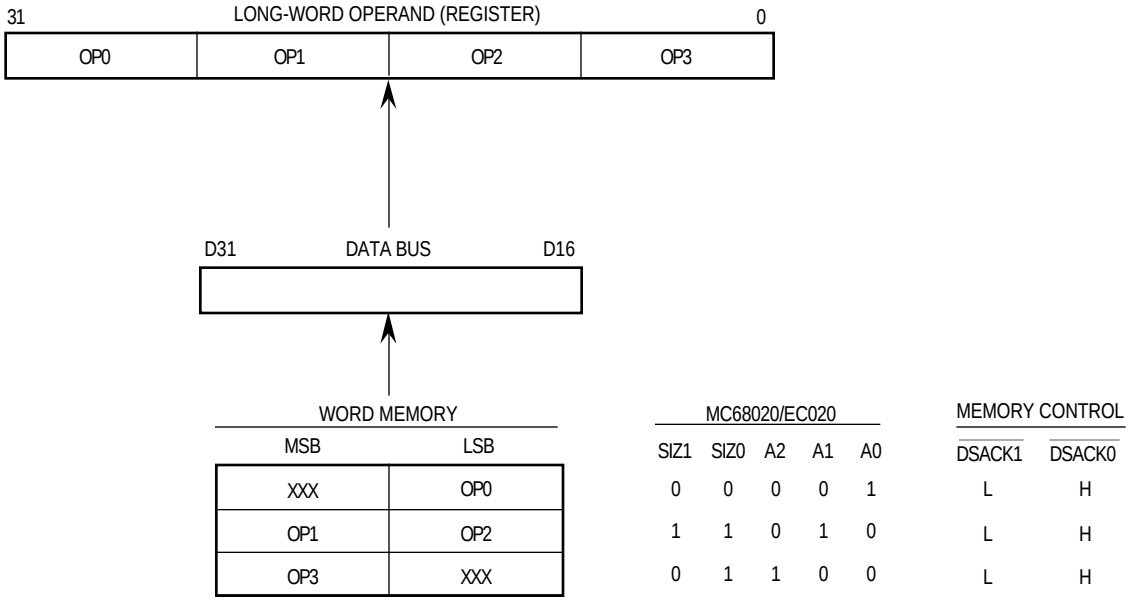


Figure 5-11. Misaligned Long-Word Operand Read from Word Port Example

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.

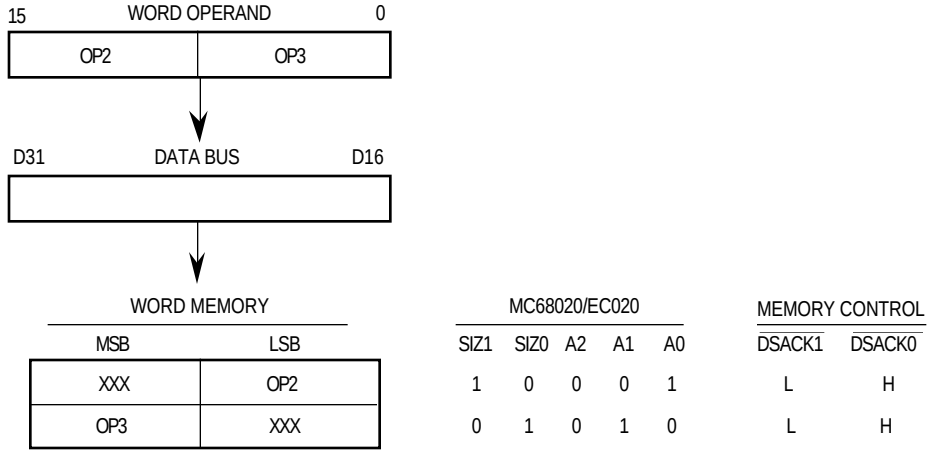


Figure 5-12. Misaligned Word Operand Write to Word Port Example

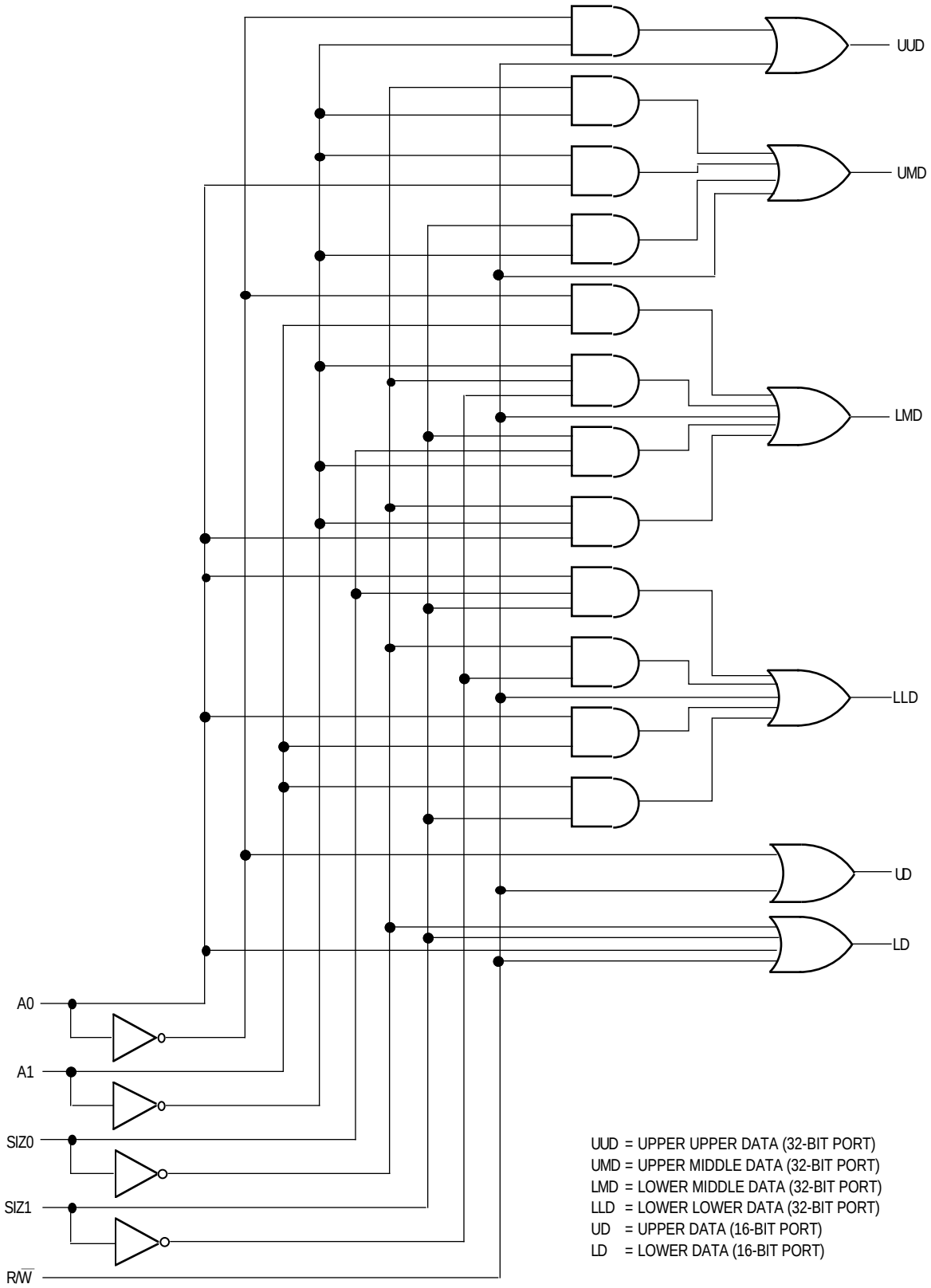


Figure 5-18. Byte Enable Signal Generation for 16- and 32-Bit Ports

for asynchronous operation can be ignored. All timing parameters referred to are described in **Section 10 Electrical Characteristics**. If a system asserts DSACK1/DSACK0 for the required window around the falling edge of state 2 and obeys the proper bus protocol by maintaining DSACK1/DSACK0 (and/or BERR/HALT) until and throughout the clock edge that negates 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 DSACK1/DSACK0.

To ensure proper operation in a synchronous system when BERR or BERR/HALT is asserted after DSACK1/DSACK0, BERR (and HALT) must meet the appropriate setup time (parameter #27A) prior to the falling clock edge one clock cycle after 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 DS signal.

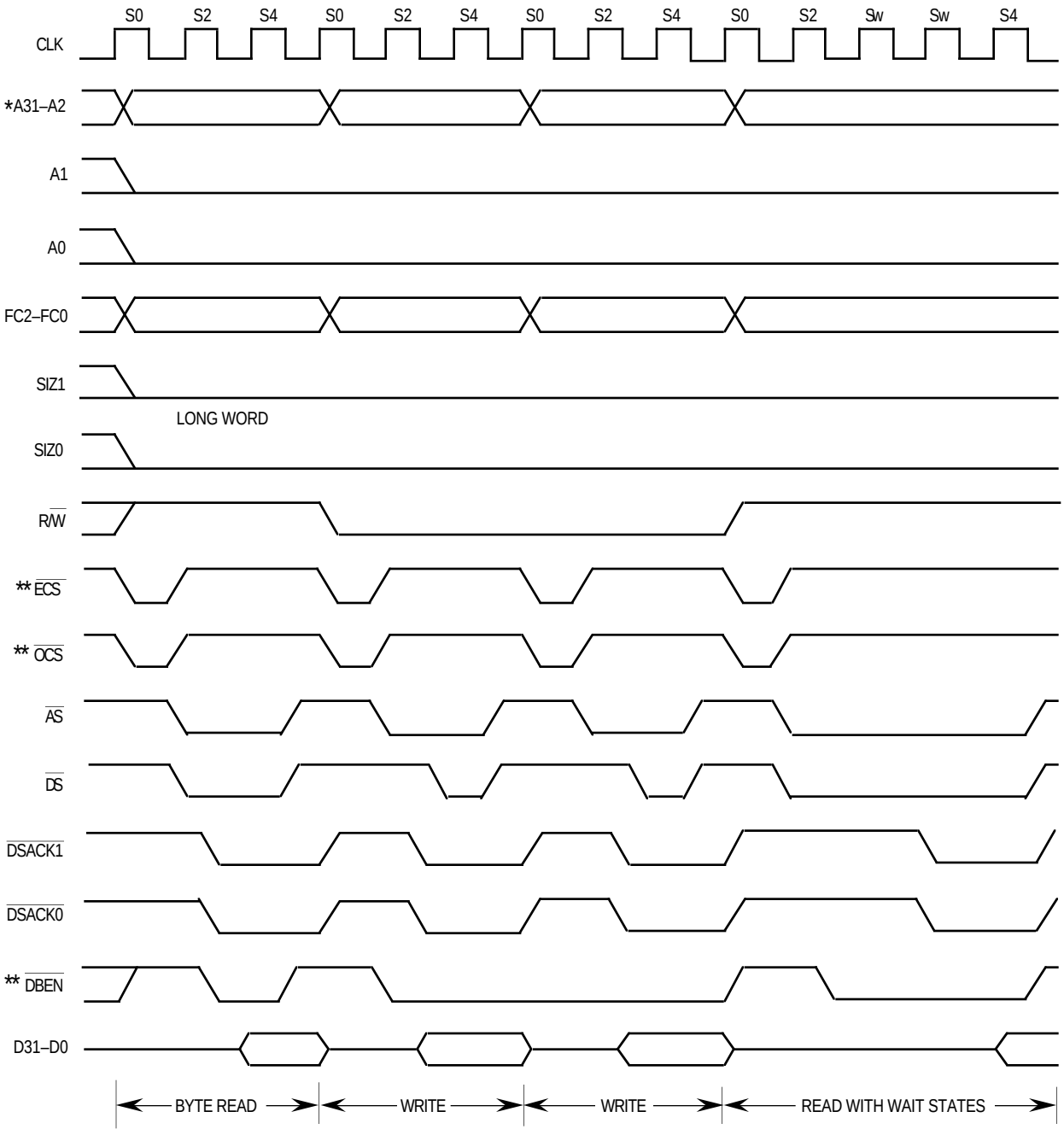
5.3 DATA TRANSFER CYCLES

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 DSACK1/DSACK0, D31–D0, BERR, HALT, and, for the MC68020, 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.



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

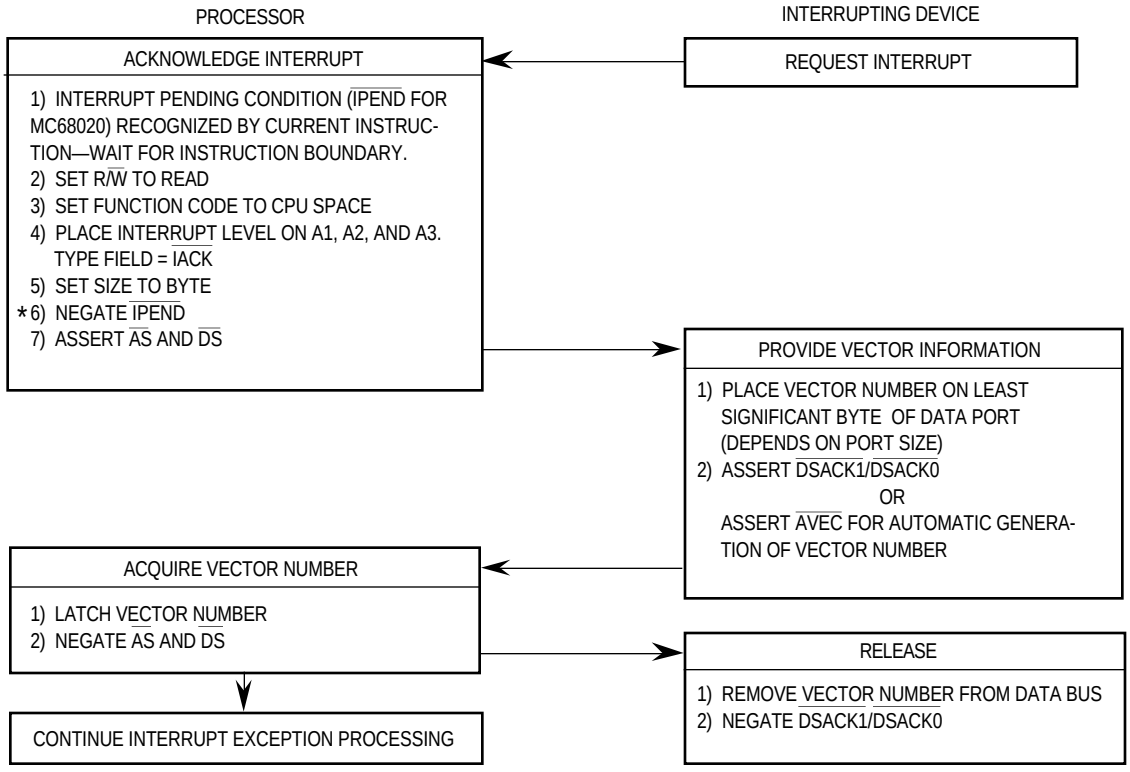
Figure 5-25. Read-Write-Read Cycles—32-Bit Port

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 $\overline{IPL2}$, $\overline{IPL1}$, and $\overline{IPL0}$, 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 $\overline{DSACK1}/\overline{DSACK0}$. Figure 5-32 is the flowchart of the interrupt acknowledge cycle.

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



* This step does not apply to the MC68EC020.

Figure 5-32. Interrupt Acknowledge Cycle Flowchart

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 $\overline{\text{DSACK1}}$/$\overline{\text{DSACK0}}$ from an external device to signal that a bus cycle is complete. $\overline{\text{DSACK1}}$/$\overline{\text{DSACK0}}$ 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 $\overline{\text{DSACK1}}$/$\overline{\text{DSACK0}}$ 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, $\overline{\text{DSACK1}}$/$\overline{\text{DSACK0}}$, $\overline{\text{BERR}}$, 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 $\overline{DSACK1}/\overline{DSACK0}$ assertion as follows (case numbers refer to Table 5-8):

Normal Termination:

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

Halt Termination:

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

Bus Error Termination:

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

Retry Termination:

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

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

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

5.6 BUS SYNCHRONIZATION

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

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

5.7 BUS ARBITRATION

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

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

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

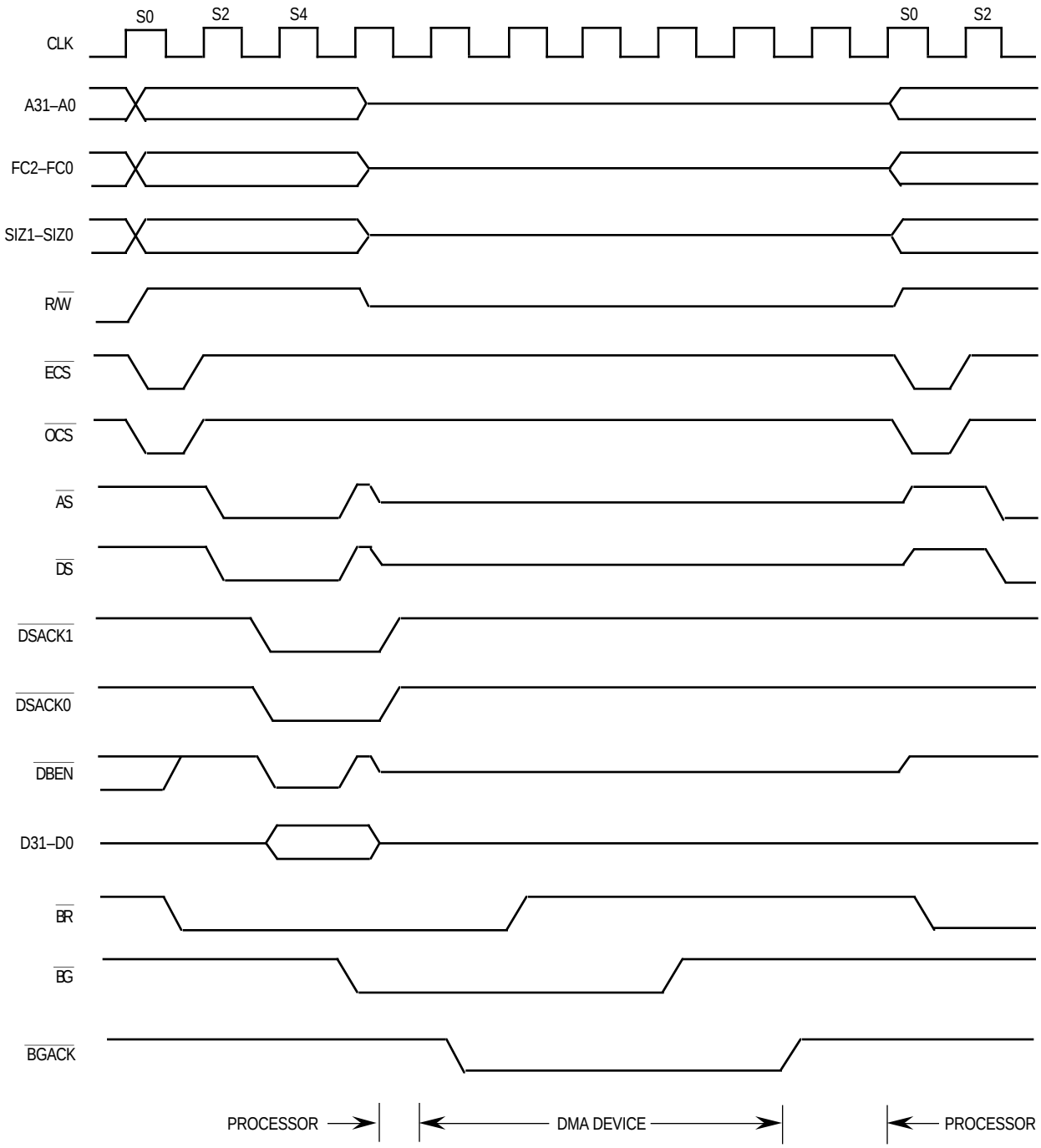


Figure 5-43. MC68020 Bus Arbitration Operation Timing for Single Request

Table 6-1. Exception Vector Assignments

Vector Number	Vector Offset		Assignment
	Hex	Space	
0	000	SP	Reset Initial Interrupt Stack Pointer
1	004	SP	Reset Initial Program Counter
2	008	SD	Bus Error
3	00C	SD	Address Error
4	010	SD	Illegal Instruction
5	014	SD	Zero Divide
6	018	SD	CHK, CHK2 Instruction
7	01C	SD	cpTRAPcc, TRAPcc, TRAPV Instructions
8	020	SD	Privilege Violation
9	024	SD	Trace
10	028	SD	Line 1010 Emulator
11	02C	SD	Line 1111 Emulator
12	030	SD	(Unassigned, Reserved)
13	034	SD	Coprocessor Protocol Violation
14	038	SD	Format Error
15	03C	SD	Uninitialized Interrupt
16–23	040 05C	SD SD	Unassigned, Reserved
24	060	SD	Spurious Interrupt
25	064	SD	Level 1 Interrupt Autovector
26	068	SD	Level 2 Interrupt Autovector
27	06C	SD	Level 3 Interrupt Autovector
28	070	SD	Level 4 Interrupt Autovector
29	074	SD	Level 5 Interrupt Autovector
30	078	SD	Level 6 Interrupt Autovector
31	07C	SD	Level 7 Interrupt Autovector
32–47	080 0BC	SD SD	TRAP #0–15 Instruction Vectors
48	0C0	SD	FPCP Branch or Set on Unordered Condition
49	0C4	SD	FPCP Inexact Result
50	0C8	SD	FPCP Divide by Zero
51	0CC	SD	FPCP Underflow
52	0D0	SD	FPCP Operand Error
53	0D4	SD	FPCP Overflow
54	0D8	SD	FPCP Signaling NAN
55	0DC	SD	Unassigned, Reserved
56	0E0	SD	PMMU Configuration
57	0E4	SD	PMMU Illegal Operation
58	0E8	SD	PMMU Access Level Violation
59–63	0EC 0FC	SD SD	Unassigned, Reserved
64–255	100 3FC	SD SD	User-Defined Vectors (192)

SP—Supervisor Program Space

SD—Supervisor Data Space

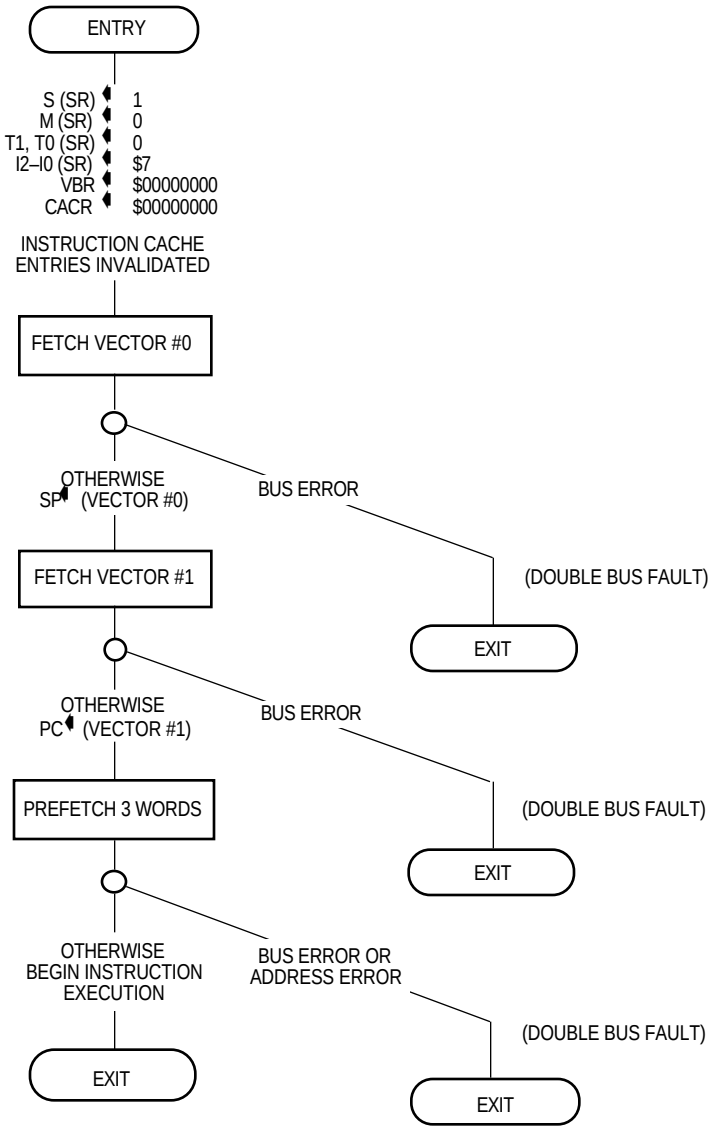
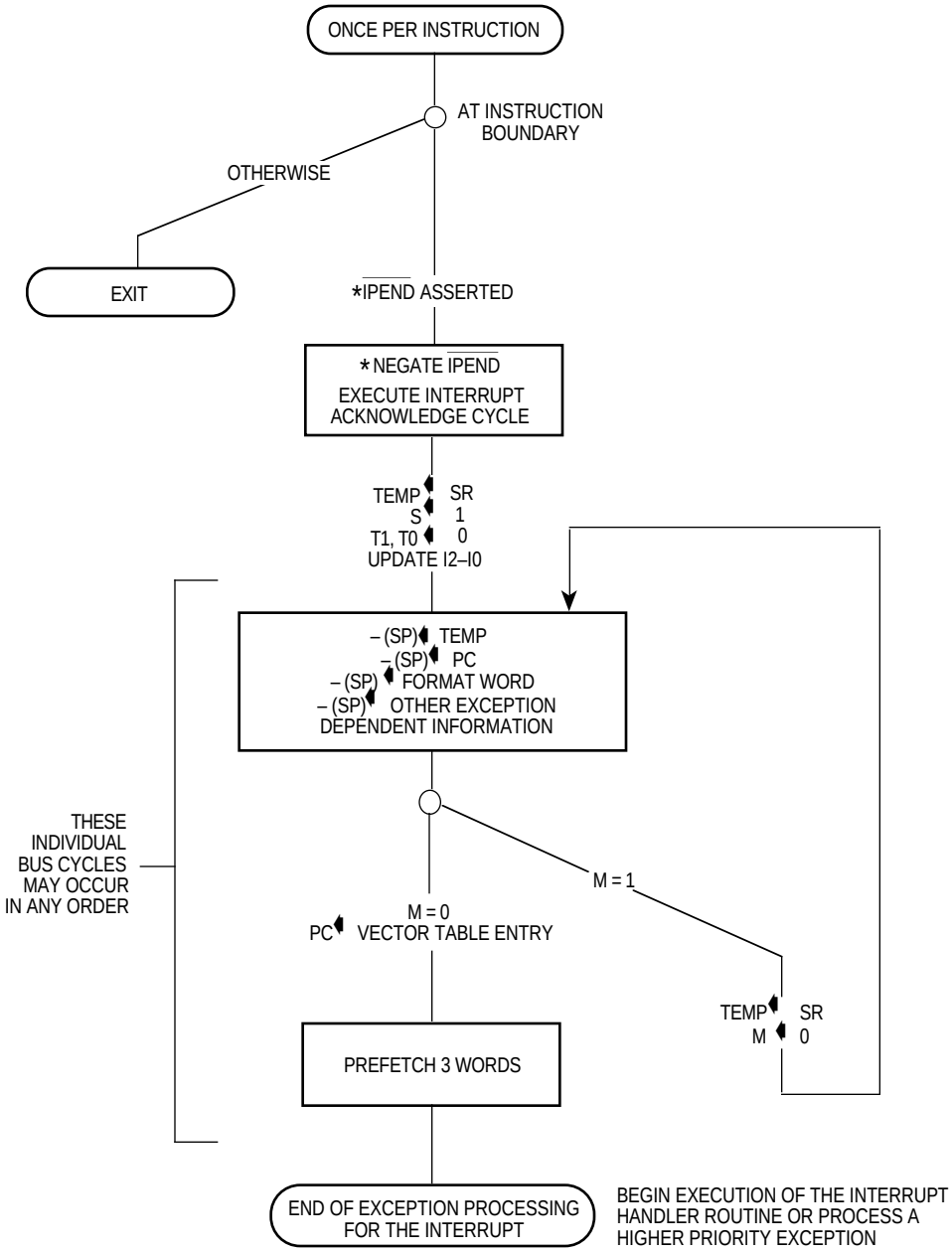


Figure 6-1. Reset Operation Flowchart

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



*Does not apply to the MC68EC020.

Figure 6-5. Interrupt Exception Processing Flowchart

SECTION 7
COPROCESSOR INTERFACE DESCRIPTION

The M68000 family of general-purpose microprocessors provides a level of performance that satisfies a wide range of computer applications. Special-purpose hardware, however, can often provide a higher level of performance for a specific application. The coprocessor concept allows the capabilities and performance of a general-purpose processor to be enhanced for a particular application without encumbering the main processor architecture. A coprocessor can efficiently meet specific capability requirements that must typically be implemented in software by a general-purpose processor. With a general-purpose main processor and the appropriate coprocessor(s), the processing capabilities of a system can be tailored to a specific application.

The MC68020/EC020 supports the M68000 coprocessor interface described in this section. This section is intended for designers who are implementing coprocessors to interface with the MC68020/EC020.

The designer of a system that uses one or more Motorola coprocessors (the MC68881 or MC68882 floating-point coprocessor, for example) does not require a detailed knowledge of the M68000 coprocessor interface. Motorola coprocessors conform to the interface described in this section. Typically, they implement a subset of the interface, and that subset is described in the coprocessor user's manual. These coprocessors execute Motorola-defined instructions that are described in the user's manual for each coprocessor.

7.1 INTRODUCTION

The distinction between standard peripheral hardware and an M68000 coprocessor is important from a programming model perspective. The programming model of the main processor consists of the instruction set, register set, and memory map. An M68000 coprocessor is a device or set of devices that communicates with the main processor through the protocol defined as the M68000 coprocessor interface. The programming model for a coprocessor is different than that for a peripheral device. A coprocessor adds additional instructions and generally additional registers and data types to the programming model that are not directly supported by the main processor architecture. The additional instructions are dedicated coprocessor instructions that utilize the coprocessor capabilities. The necessary interactions between the main processor and the coprocessor that provide a given service are transparent to the programmer. That is, the programmer does not need to know the specific communication protocol between the main processor and the coprocessor because this protocol is implemented in hardware. Thus, the coprocessor can provide capabilities to the user without appearing separate from the main processor.

The transfer operation word primitive uses the CA and PC bits as described in **7.4.2 Coprocessor Response Primitive General Format**. If this primitive is issued with CA = 0 during a conditional category instruction, the main processor initiates protocol violation exception processing.

When the main processor reads this primitive from the response CIR, it transfers the F-line operation word of the currently executing coprocessor instruction to the operation word CIR. The value of the scanPC is not affected by this primitive.

7.4.7 Transfer from Instruction Stream Primitive

The transfer from instruction stream primitive initiates transfers of operands from the instruction stream to the coprocessor. This primitive applies to general and conditional category instructions. Figure 7-27 shows the format of the transfer from instruction stream primitive.

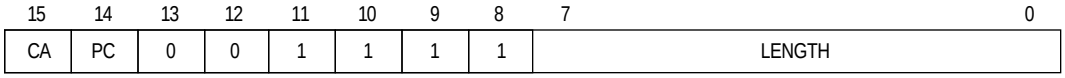


Figure 7-27. Transfer from Instruction Stream Primitive Format

The transfer from instruction stream primitive uses the CA and PC bits as described in **7.4.2 Coprocessor Response Primitive General Format**. If this primitive is issued with CA = 0 during a conditional category instruction, the main processor initiates protocol violation exception processing.

The length field of this primitive specifies the length, in bytes, of the operand to be transferred from the instruction stream to the coprocessor. The length must be an even number of bytes. If an odd length is specified, the main processor initiates protocol violation exception processing (refer to **7.5.2.1 Protocol Violations**).

This primitive transfers coprocessor-defined extension words to the coprocessor. When the main processor reads this primitive from the response CIR, it copies the number of bytes indicated by the length field from the instruction stream to the operand CIR. The first word or long word transferred is at the location pointed to by the scanPC when the primitive is read by the main processor. The scanPC is incremented after each word or long word is transferred. When execution of the primitive has completed, the scanPC has been incremented by the total number of bytes transferred and points to the word following the last word transferred. The main processor transfers the operands from the instruction stream, using a sequence of long-word writes, to the operand CIR. If the length field is not an even multiple of four bytes, the last two bytes from the instruction stream are transferred using a word write to the operand CIR.

The D/A bit specifies whether the primitive transfers an address or data register. D/A = 0 indicates a data register, and D/A = 1 indicates an address register. The register field contains the register number.

If DR = 0, the main processor writes the long-word operand in the specified register to the operand CIR. If DR = 1, the main processor reads a long-word operand from the operand CIR and transfers it to the specified data or address register.

7.4.14 Transfer Main Processor Control Register Primitive

The transfer main processor control register primitive transfers a long-word operand between one of its control registers and the coprocessor. This primitive applies to general and conditional category instructions. Figure 7-34 shows the format of the transfer main processor control register primitive.

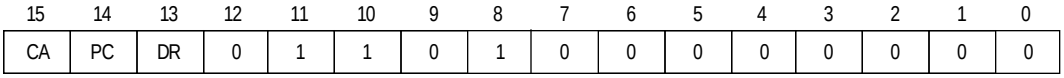


Figure 7-34. Transfer Main Processor Control Register Primitive Format

The transfer main processor control register primitive uses the CA, PC, and DR bits as described in **7.4.2 Coprocessor Response Primitive General Format**. If the coprocessor issues this primitive with CA = 0 during a conditional category instruction, the main processor initiates protocol violation exception processing.

When the main processor receives this primitive, it reads a control register select code from the register select CIR. This code determines which main processor control register is transferred. Table 7-5 lists the valid control register select codes. If the control register select code is not valid, the MC68020/EC020 initiates protocol violation exception processing (refer to **7.5.2.1 Protocol Violations**).

Table 7-5. Main Processor Control Register Select Codes

Select Code	Control Register
$x000	SFC
$x001	DFC
$x002	CACR
$x800	USP
$x801	VBR
$x802	CAAR
$x803	MSP
$x804	ISP

All other codes cause a protocol violation exception.

The PC value saved in this stack frame is the operation word address of the coprocessor instruction during which the primitive is received. The scanPC field contains the value of the MC68020/EC020 scanPC when the primitive is received. If the current instruction does not evaluate an effective address prior to the exception request primitive, the value of the effective address field in the stack frame is undefined.

The coprocessor uses this primitive to request exception processing for an exception during the instruction dialog with the main processor. If the exception handler does not modify the stack frame, the MC68020/EC020 returns from the exception handler and reads the response CIR. Thus, the main processor attempts to continue executing the suspended instruction by reading the response CIR and processing the primitive it receives.

7.4.20 Take Postinstruction Exception Primitive

The take postinstruction exception primitive initiates exception processing using a coprocessor-supplied exception vector number and the postinstruction exception stack frame format. This primitive applies to general and conditional category instructions. Figure 7-44 shows the format of the take postinstruction exception primitive.

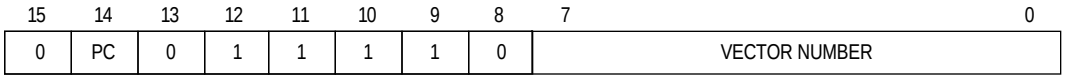


Figure 7-44. Take Postinstruction Exception Primitive Format

The take postinstruction exception primitive uses the PC bit as described in **7.4.2 Coprocessor Response Primitive General Format**. The vector number field contains the exception vector number used by the main processor to initiate exception processing.

When the main processor receives this primitive, it acknowledges the coprocessor exception request by writing an exception acknowledge mask to the control CIR (refer to **7.3.2 Control CIR**). The MC68020/EC020 then performs exception processing as described in **Section 6 Exception Processing**. The vector number for the exception is taken from the vector number field of the primitive, and the MC68020/EC020 uses the six-word stack frame format shown in Figure 7-45.

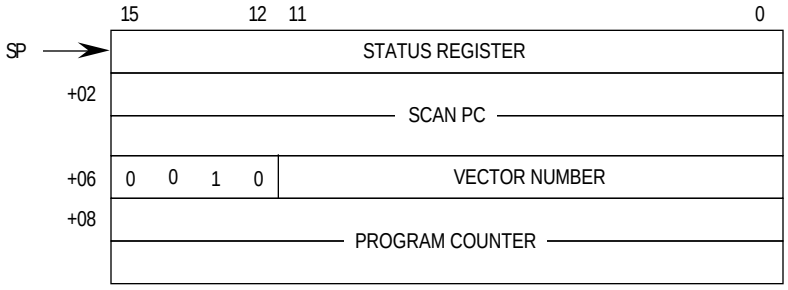


Figure 7-45. MC68020/EC020 Postinstruction Stack Frame

BEST CASE (Continued)

Source Address Mode	Destination							
	(d_8,An,Xn)	**(d_{16},An,Xn)**	**(B)**	**(d_{16},B)**	**(d_{32},B)**	**([B],I)**	**([B],I,d_{16})**	**([B],I,d_{32})**
Rn	**4**(0/0/1)	**6**(0/0/1)	**5**(0/0/1)	**7**(0/0/1)	**11**(0/0/1)	**9**(1/0/1)	**11**(1/0/1)	**12**(1/0/1)
#<data>.B,W	**4**(0/0/1)	**6**(0/0/1)	**5**(0/0/1)	**7**(0/0/1)	**11**(0/0/1)	**9**(1/0/1)	**11**(1/0/1)	**12**(1/0/1)
#<data>.L	**4**(0/0/1)	**6**(0/0/1)	**5**(0/0/1)	**7**(0/0/1)	**11**(0/0/1)	**9**(1/0/1)	**11**(1/0/1)	**12**(1/0/1)
(An)	**8**(1/0/1)	**10**(1/0/1)	**9**(1/0/1)	**11**(1/0/1)	**15**(1/0/1)	**13**(2/0/1)	**15**(2/0/1)	**16**(2/0/1)
(An)+	**9**(1/0/1)	**11**(1/0/1)	**10**(1/0/1)	**12**(1/0/1)	**16**(1/0/1)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/0/1)
–(An)	**8**(1/0/1)	**10**(1/0/1)	**9**(1/0/1)	**11**(1/0/1)	**15**(1/0/1)	**13**(2/0/1)	**15**(2/0/1)	**16**(2/0/1)
(d_{16},An) or (d_{16},PC)	**8**(1/0/1)	**10**(1/0/1)	**9**(1/0/1)	**11**(1/0/1)	**15**(1/0/1)	**13**(2/0/1)	**15**(2/0/1)	**16**(2/0/1)
(xxx).W	**8**(1/0/1)	**10**(1/0/1)	**9**(1/0/1)	**11**(1/0/1)	**15**(1/0/1)	**13**(2/0/1)	**15**(2/0/1)	**16**(2/0/1)
(xxx).L	**8**(1/0/1)	**10**(1/0/1)	**9**(1/0/1)	**11**(1/0/1)	**15**(1/0/1)	**13**(2/0/1)	**15**(2/0/1)	**16**(2/0/1)
(d_8,An,Xn) or (d_8,PC,Xn)	**9**(1/0/1)	**10**(1/0/1)	**10**(1/0/1)	**12**(1/0/1)	**16**(1/0/1)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/0/1)
(d_{16},An,Xn) or (d_{16},PC,Xn)	**9**(1/0/1)	**11**(1/0/1)	**10**(1/0/1)	**12**(1/0/1)	**16**(1/0/1)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/0/1)
(B)	**9**(1/0/1)	**11**(1/0/1)	**10**(1/0/1)	**12**(1/0/1)	**16**(1/0/1)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/0/1)
(d_{16},B)	**11**(1/0/1)	**13**(1/0/1)	**12**(1/0/1)	**14**(1/0/1)	**18**(1/0/1)	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)
(d_{32},B)	**15**(1/0/1)	**17**(1/0/1)	**18**(1/0/1)	**18**(1/0/1)	**22**(1/0/1)	**20**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)
([B],I)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/0/1)	**17**(2/0/1)	**21**(2/0/1)	**19**(3/0/1)	**21**(3/0/1)	**22**(3/0/1)
([B],I,d_{16})	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**19**(2/0/1)	**23**(2/0/1)	**21**(3/0/1)	**23**(3/0/1)	**24**(3/0/1)
([B],I,d_{32})	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**19**(2/0/1)	**23**(2/0/1)	**21**(3/0/1)	**23**(3/0/1)	**24**(3/0/1)
([d_{16},B],I)	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**19**(2/0/1)	**23**(2/0/1)	**21**(3/0/1)	**23**(3/0/1)	**24**(3/0/1)
([d_{16},B],I,d_{16})	**18**(2/0/1)	**20**(2/0/1)	**21**(2/0/1)	**21**(2/0/1)	**25**(2/0/1)	**23**(3/0/1)	**25**(3/0/1)	**26**(3/0/1)
([d_{16},B],I,d_{32})	**18**(2/0/1)	**20**(2/0/1)	**21**(2/0/1)	**21**(2/0/1)	**25**(2/0/1)	**23**(3/0/1)	**25**(3/0/1)	**26**(3/0/1)
([d_{32},B],I)	**20**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)	**23**(2/0/1)	**27**(2/0/1)	**25**(3/0/1)	**27**(3/0/1)	**28**(3/0/1)
([d_{32},B],I,d_{16})	**22**(2/0/1)	**24**(2/0/1)	**25**(2/0/1)	**25**(2/0/1)	**29**(2/0/1)	**27**(3/0/1)	**29**(3/0/1)	**30**(3/0/1)
([d_{32},B],I,d_{32})	**22**(2/0/1)	**24**(2/0/1)	**25**(2/0/1)	**25**(2/0/1)	**29**(2/0/1)	**27**(3/0/1)	**29**(3/0/1)	**30**(3/0/1)

8.2.17 Exception-Related Instructions

The exception-related instructions table indicates the number of clock periods needed for the processor to perform the specified exception-related action. Footnotes specify when it is necessary to add the entry from another table to calculate the total effective execution time for the given instruction. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Instruction	Best Case	Cache Case	Worst Case
BKPT	**9**(1/0/0)	**10**(1/0/0)	**10**(1/0/0)
Interrupt (I-Stack)	**26**(2/0/4)	**26**(2/0/4)	**33**(2/2/4)
Interrupt (M-Stack)	**41**(2/0/8)	**41**(2/0/8)	**48**(2/2/8)
RESET Instruction	**518**(0/0/0)	**518**(0/0/0)	**519**(0/1/0)
STOP	**8**(0/0/0)	**8**(0/0/0)	**8**(0/0/0)
Trace	**25**(1/0/5)	**25**(1/0/5)	**32**(1/2/5)
TRAP #n	**20**(1/0/4)	**20**(1/0/4)	**27**(1/2/4)
Illegal Instruction	**20**(1/0/4)	**20**(1/0/4)	**27**(1/2/4)
A-Line Trap	**20**(1/0/4)	**20**(1/0/4)	**27**(1/2/4)
F-Line Trap	**20**(1/0/4)	**20**(1/0/4)	**27**(1/2/4)
Privilege Violation	**20**(1/0/4)	**20**(1/0/4)	**27**(1/2/4)
TRAPcc (Trap)	**23**(1/0/5)	**25**(1/0/5)	**32**(1/2/5)
TRAPcc (No Trap)	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
TRAPcc.W (Trap)	**23**(1/0/5)	**25**(1/0/5)	**33**(1/3/5)
TRAPcc.W (No Trap)	**3**(0/0/0)	**6**(0/0/0)	**7**(0/1/0)
TRAPcc.L (Trap)	**23**(1/0/5)	**25**(1/0/5)	**33**(1/3/5)
TRAPcc.L (No Trap)	**5**(0/0/0)	**8**(0/0/0)	**10**(0/2/0)
TRAPV (Trap)	**23**(1/0/5)	**25**(1/0/5)	**32**(1/2/5)
TRAPV (No Trap)	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)