Welcome to E-XFL.COM

Understanding Embedded - Microprocessors

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

Applications of Embedded - Microprocessors

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

Details	
Product Status	Obsolete
Core Processor	68020
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	25MHz
Co-Processors/DSP	-
RAM Controllers	-
Graphics Acceleration	No
Display & Interface Controllers	-
Ethernet	-
SATA	-
USB	-
Voltage - I/O	5.0V
Operating Temperature	0°C ~ 70°C (TA)
Security Features	-
Package / Case	100-BQFP
Supplier Device Package	100-QFP (14x20)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mc68ec020aa25

TABLE OF CONTENTS

TABLE OF CONTENTS (Concluded)

LIST OF TABLES (Continued)

3.1 SIGNAL INDEX

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

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

3.2 FUNCTION CODE SIGNALS (FC2–FC0)

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

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

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

3.4 DATA BUS (D31–D0)

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

3.5 TRANSFER SIZE SIGNALS (SIZ1, SIZ0)

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

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

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/$\overline{W}$ signal.

5.2.5 Cache Interactions

The organization and requirements of the on-chip instruction cache affect the interpretation of $\overline{DSACK1}$ and $\overline{DSACK0}$. 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 $\overline{AS}$ 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 $\overline{AS}$ is not asserted.

For the MC68020, if the bus is not occupied with another read or write cycle, the bus controller asserts the $\overline{ECS}$ signal (and the $\overline{OCS}$ signal, if appropriate). It is possible to have $\overline{ECS}$ asserted on multiple consecutive clock cycles. Note that there is a minimum time specified from the negation of $\overline{ECS}$ to the next assertion of $\overline{ECS}$ (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 $\overline{ECS}$ 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.

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.

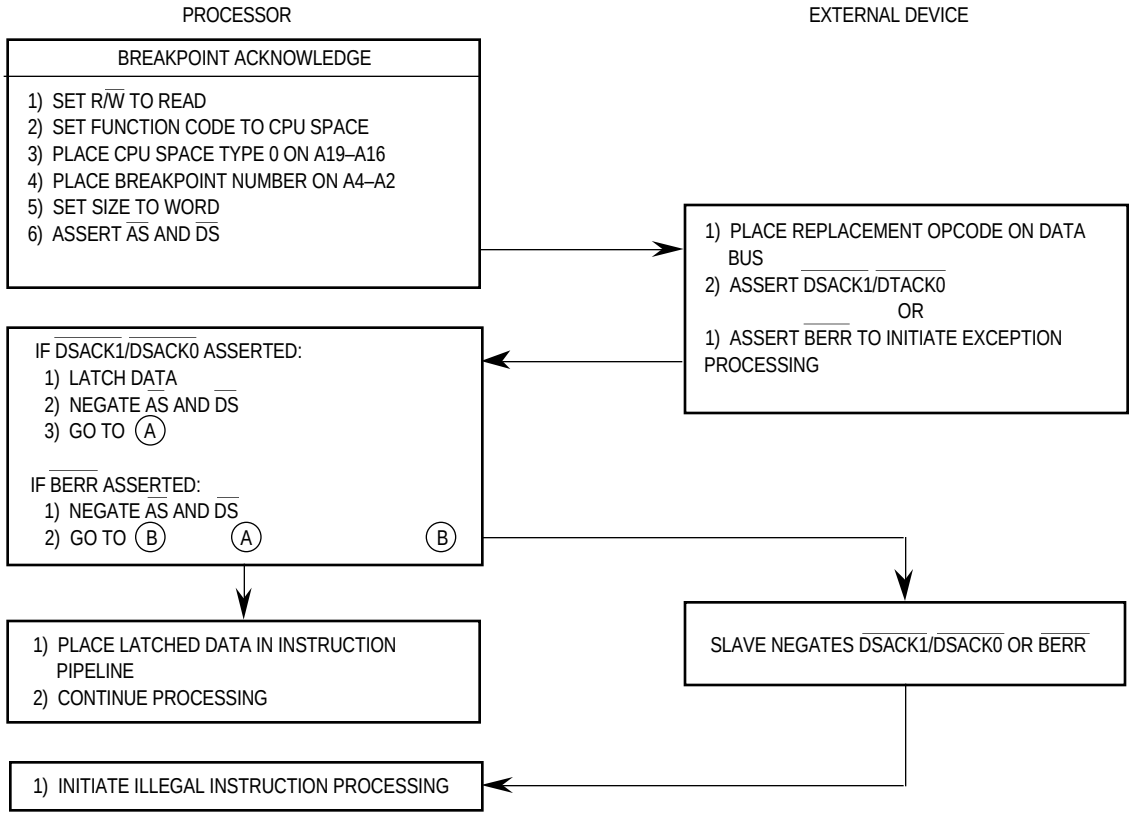

Figure 5-35. Breakpoint Acknowledge Cycle Flowchart

5.5.3 Halt Operation

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

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

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

5.5.4 Double Bus Fault

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

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

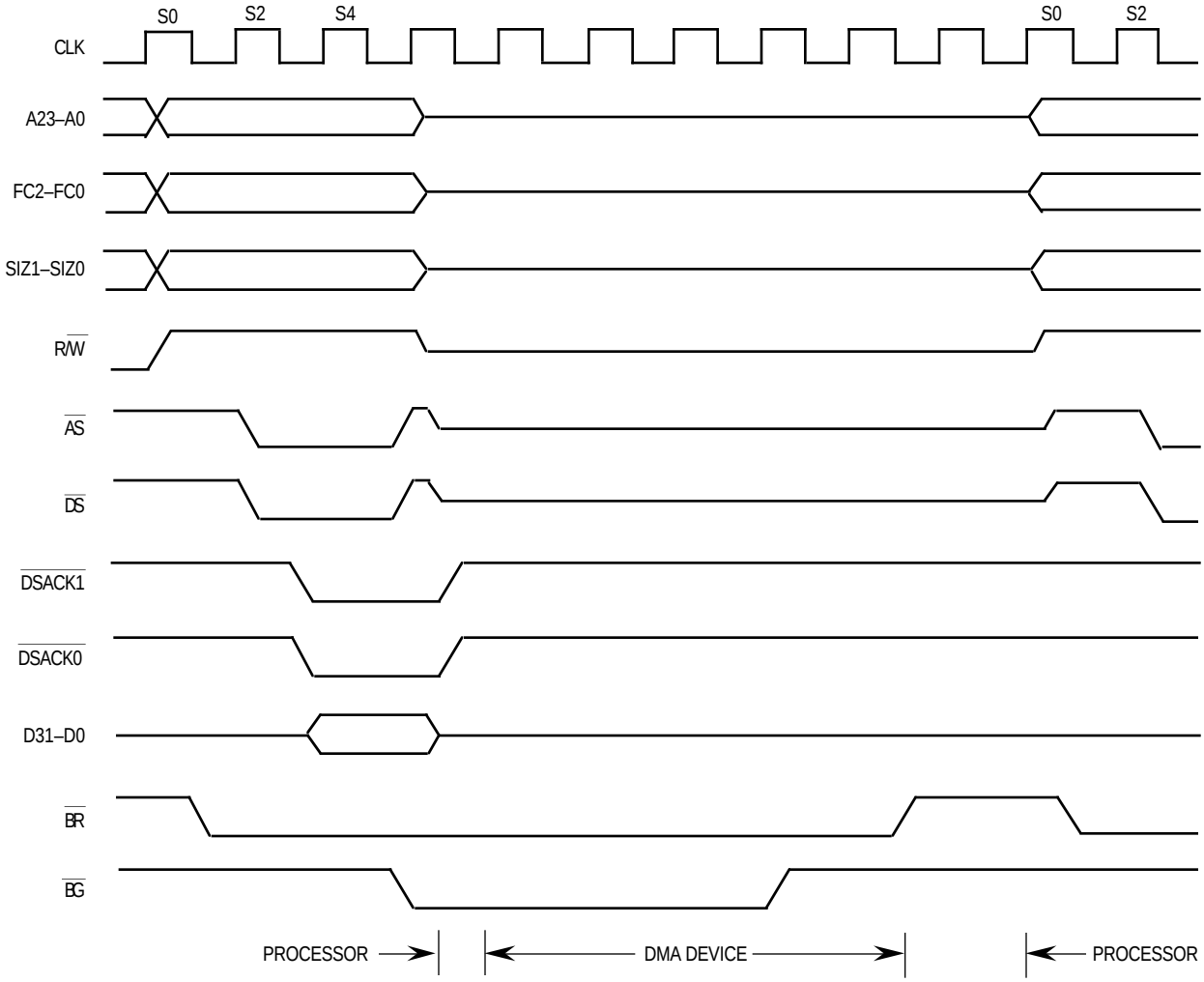

Figure 5-47. MC68EC020 Bus Arbitration Operation Timing for Single Request

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.

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**.

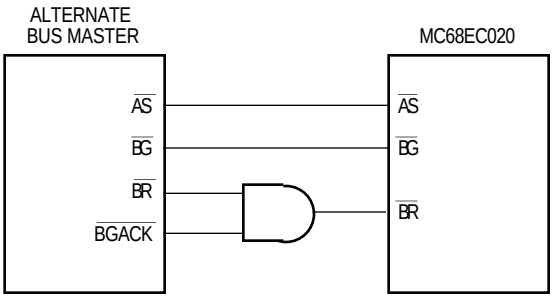

Figure 5-50. Interface for Three-Wire to Two-Wire Bus Arbitration

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.

6.1.1 Reset Exception

Assertion of the RESET signal by external hardware causes a reset exception. For details on the requirements for the assertion of RESET, refer to **Section 5 Bus Operation**.

The reset exception has the highest priority of any exception; it provides for system initialization and recovery from catastrophic failure. When a reset exception is recognized, it aborts any processing in progress and that processing cannot be recovered. Figure 6-1 is a flowchart of the reset exception, which performs the following operations:

1. Clears the T1 and T0 bits in the SR to disable tracing.
2. Places the processor in the interrupt mode of the supervisor privilege level by setting the S-bit and clearing the M-bit in the SR.
3. Sets the I2–I0 bits in the SR to the highest priority level (level 7).
4. Initializes the VBR to zero ($00000000).
5. Clears the E and F bits in the CACR.
6. Invalidates all entries in the instruction cache.
7. Generates a vector number to reference the reset exception vector (two long words) at offset zero in the supervisor program address space.
8. Loads the first long word of the reset exception vector into the interrupt stack pointer.
9. Loads the second long word of the reset exception vector into the PC.

After the initial instruction prefetches, program execution begins at the address in the PC. The reset exception does not save the value of either the PC or the SR.

As described in **Section 5 Bus Operation**, if a bus error or address error occurs during the exception processing sequence for a reset, a double bus fault occurs. The processor halts and asserts the HALT signal to indicate the halted condition.

Execution of the RESET instruction does not cause a reset exception, nor does it affect any internal registers, but it does cause the MC68020/EC020 to assert the RESET signal, resetting all external devices.

6.1.2 Bus Error Exception

A bus error exception occurs when external logic aborts a bus cycle by asserting the BERR signal. If the aborted bus cycle is a data access, the processor immediately begins exception processing. If the aborted bus cycle is an instruction prefetch, the processor may delay taking the exception until it attempts to use the prefetched information.

7.4.3 Busy Primitive

The busy response primitive causes the main processor to reinitiate a coprocessor instruction. This primitive applies to instructions in the general and conditional categories. Figure 7-23 shows the format of the busy primitive.

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

Figure 7-23. Busy Primitive Format

The busy primitive uses the PC bit as described in **7.4.2 Coprocessor Response Primitive General Format**.

Coprocessors that can operate concurrently with the main processor but cannot buffer write operations to their command or condition CIR use the busy primitive. A coprocessor may execute a cpGEN instruction concurrently with an instruction in the main processor. If the main processor attempts to initiate an instruction in the general or conditional instruction category while the coprocessor is executing a cpGEN instruction, the coprocessor can place the busy primitive in the response CIR. When the main processor reads this primitive, it services pending interrupts using a preinstruction exception stack frame (refer to Figure 7-41). The processor then restarts the general or conditional coprocessor instruction that it had attempted to initiate earlier.

The busy primitive should only be used in response to a write to the command or condition CIR. It should be the first primitive returned after the main processor attempts to initiate a general or conditional category instruction. In particular, the busy primitive should not be issued after program-visible resources have been altered by the instruction. (Program-visible resources include coprocessor and main processor program-visible registers and operands in memory, but not the scanPC.) The restart of an instruction after it has altered program-visible resources causes those resources to have inconsistent values when the processor reinitiates the instruction.

The MC68020/EC020 responds to the busy primitive differently in a special case that can occur during a breakpoint operation (refer to **Section 6 Exception Processing**). This special case occurs when a breakpoint acknowledge cycle initiates a coprocessor F-line instruction, the coprocessor returns the busy primitive in response to the instruction initiation, and an interrupt is pending. When these three conditions are met, the processor reexecutes the breakpoint acknowledge cycle after completion of interrupt exception processing. A design that uses a breakpoint to monitor the number of passes through a loop by incrementing or decrementing a counter may not work correctly under these conditions. This special case may cause several breakpoint acknowledge cycles to be executed during a single pass through a loop.

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

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

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

7.5 EXCEPTIONS

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

7.5.1 Coprocessor-Detected Exceptions

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

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

8.1.4 Instruction Execution Overlap

Overlap is the time, measured in clocks, when two instructions execute concurrently. In Figure 8-1, instructions A and B execute concurrently, and the overlapped portion of instruction B is absorbed in the instruction execution time of A (the previous instruction). The overlap time is deducted from the execution time of instruction B. Similarly, there is an overlap period between instruction B and instruction C, which reduces the attributed execution time for C.

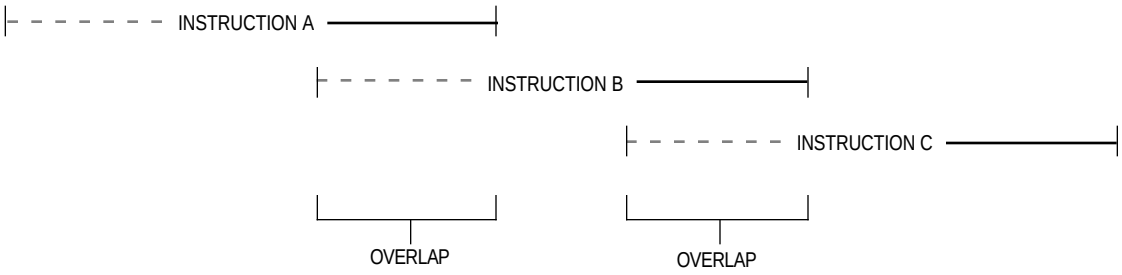

Figure 8-1. Concurrent Instruction Execution

The execution time attributed to instructions A, B, and C (after considering the overlap) is depicted in Figure 8-2.

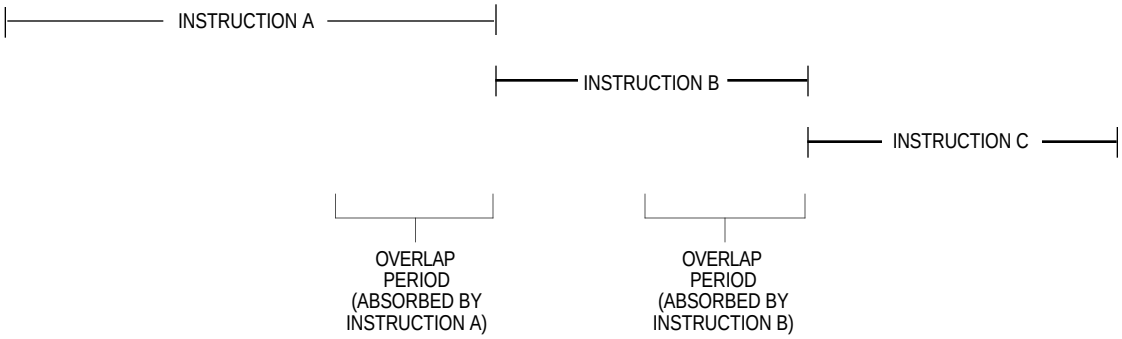

Figure 8-2. Instruction Execution for Instruction Timing Purposes

It is possible that the execution time of an instruction will be absorbed by the overlap with a previous instruction for a net execution time of zero clocks.

Because of this overlap, a NOP is required between a write to a peripheral to clear an interrupt request and a subsequent MOVE to SR instruction to lower the interrupt mask level. Otherwise, the MOVE to SR instruction may complete before the write is accomplished, and a new interrupt exception will be generated for an old interrupt request.

NOTE

This CC time is a maximum since the times given for the MULU.L and DIVS.L are maximums.

The MOVE instruction timing tables include all necessary timing for extension word fetch, address calculation, and operand fetch.

The instruction timing tables are used to calculate a best-case and worst-case bounds for some target instruction stream. Calculating exact timing from the timing tables is impossible because the tables cannot anticipate how the combination of factors will influence every particular sequence of instructions. This is illustrated by comparing the observed instruction timing from the prior four examples with instruction timing derived from the instruction timing tables.

Table 8-2 lists the original instruction stream and the corresponding clock timing from the appropriate timing tables for the best case, cache-only case, and worst case.

Table 8-2. Instruction Timings from Timing Tables

Instruction		Best Case	Cache Case	Worst Case
#1) MOVE.L	D4,(A1)+	4	4	6
#2) ADD.L	D4,D5	0	2	3
#3) MOVE.L	(A1),–(A2)	6	7	9
#4) ADD.L	D5,D6	0	2	3
Total		10	15	21

Table 8-3 summarizes the observed instruction timings for the same instruction stream as executed according to the assumptions of the four examples. For each example, Table 8-3 shows which entry (BC/CC/WC) from the timing tables corresponds to the observed timing for each of the four instructions. Some of the observed instruction timings cannot be found in the timing tables and appear in Table 8-3 within parentheses in the most appropriate column. These timings occur when instruction execution overlap dynamically alters what would otherwise be a BC, CC, or WC timing.

Table 8-3. Observed Instruction Timings

Instruction		Example 1			Example 2			Example 3			Example 4		
		BC	CC	WC	BC	CC	WC	BC	CC	WC	BC	CC	WC
#1) MOVE.L	D4,(A1)+			6	4				4			(5)	
#2) ADD.L	D4,D5	0					3	0			0		
#3) MOVE.L	(A1),–(A2)			9	6				7			(8)	
#4) ADD.L	D5,D6	(1)					3	(1)			0		
Total			(16)			(16)			(12)			(13)	

WORST CASE (Concluded)

Source Address Mode	Destination					
	([d_{16},B],I)	**([d_{16},B],I,d16)**	**([d_{16},B],I,d_{32})**	**([d_{32},B],I)**	**([d_{32},B],I,d_{16})**	**([d_{32},B],I,d_{32})**
Rn	**17**(1/2/1)	**20**(1/2/1)	**23**(1/3/1)	**22**(1/2/1)	**25**(1/3/1)	**27**(1/3/1)
#<data>.B,W	**17**(1/2/1)	**20**(1/2/1)	**23**(1/3/1)	**22**(1/2/1)	**25**(1/3/1)	**27**(1/3/1)
#<data>.L	**19**(1/2/1)	**22**(1/2/1)	**25**(1/3/1)	**24**(1/2/1)	**27**(1/3/1)	**29**(1/3/1)
(An)	**19**(2/2/1)	**22**(2/2/1)	**25**(2/3/1)	**24**(2/2/1)	**27**(2/3/1)	**29**(2/3/1)
(An)+	**19**(2/2/1)	**22**(2/2/1)	**25**(2/3/1)	**24**(2/2/1)	**27**(2/3/1)	**29**(2/3/1)
–(An)	**20**(2/2/1)	**23**(2/2/1)	**26**(2/3/1)	**25**(2/2/1)	**28**(2/3/1)	**30**(2/3/1)
(d_{16},An) or (d_{16},PC)	**21**(2/3/1)	**24**(2/3/1)	**27**(2/4/1)	**26**(2/3/1)	**29**(2/4/1)	**31**(2/4/1)
(xxx).W	**20**(2/3/1)	**23**(2/3/1)	**26**(2/4/1)	**27**(2/3/1)	**28**(2/4/1)	**30**(2/4/1)
(xxx).L	**22**(2/3/1)	**25**(2/3/1)	**28**(2/4/1)	**29**(2/3/1)	**30**(2/4/1)	**32**(2/4/1)
(d_8,An,Xn) or (d_8,PC,Xn)	**23**(2/3/1)	**26**(2/3/1)	**29**(2/4/1)	**30**(2/3/1)	**31**(2/4/1)	**33**(2/4/1)
(d_{16},An,Xn) or (d_{16},PC,Xn)	**24**(2/3/1)	**27**(2/3/1)	**30**(2/4/1)	**31**(2/3/1)	**32**(2/4/1)	**34**(2/4/1)
(B)	**24**(2/3/1)	**27**(2/3/1)	**30**(2/4/1)	**31**(2/3/1)	**32**(2/4/1)	**34**(2/4/1)
(d_{16},B)	**27**(2/3/1)	**30**(2/3/1)	**33**(2/4/1)	**34**(2/3/1)	**35**(2/4/1)	**37**(2/4/1)
(d_{32},B)	**31**(2/4/1)	**34**(2/4/1)	**37**(2/5/1)	**38**(2/4/1)	**39**(2/5/1)	**41**(2/5/1)
([B],I)	**28**(3/3/1)	**31**(3/3/1)	**34**(3/4/1)	**35**(3/3/1)	**36**(3/4/1)	**38**(3/4/1)
([B],I,d_{16})	**31**(3/3/1)	**34**(3/3/1)	**37**(3/4/1)	**38**(3/3/1)	**39**(3/4/1)	**41**(3/4/1)
([B],I,d_{32})	**32**(3/4/1)	**35**(3/4/1)	**38**(3/5/1)	**39**(3/4/1)	**40**(3/5/1)	**42**(3/5/1)
([d_{16},B],I)	**31**(3/3/1)	**34**(3/3/1)	**37**(3/4/1)	**38**(3/3/1)	**39**(3/4/1)	**41**(3/4/1)
([d_{16},B],I,d_{16})	**34**(3/4/1)	**37**(3/4/1)	**40**(3/5/1)	**41**(3/4/1)	**42**(3/5/1)	**44**(3/5/1)
([d_{16},B],I,d_{32})	**35**(3/4/1)	**38**(3/4/1)	**41**(3/5/1)	**42**(3/4/1)	**43**(3/5/1)	**45**(3/5/1)
([d_{32},B],I)	**35**(3/4/1)	**38**(3/4/1)	**41**(3/5/1)	**42**(3/4/1)	**43**(3/5/1)	**45**(3/5/1)
([d_{32},B],I,d_{16})	**37**(3/4/1)	**40**(3/4/1)	**43**(3/5/1)	**44**(3/4/1)	**45**(3/5/1)	**47**(3/5/1)
([d_{32},B],I,d_{32})	**39**(3/5/1)	**42**(3/5/1)	**45**(3/6/1)	**46**(3/5/1)	**47**(3/6/1)	**49**(3/6/1)

Table 9-1. Data Bus Activity 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

During cachable read cycles, the addressed device must provide valid data over its full bus width as indicated by DSACK1/DSACK0. While instructions are always prefetched as long-word-aligned accesses, data fetches can occur with any alignment and size. Because the MC68020/EC020 assumes that the entire data bus port size contains valid data, cachable data read bus cycles must provide as much data as signaled by the port size during a bus cycle. To satisfy this requirement, the R/W signal must be included in the byte select logic for the MC68020/EC020.

Figure 9-5 shows a block diagram of an MC68020/EC020 system with a single memory bank. The PAL provides memory-mapped byte select signals for an asynchronous 32-bit port and unmapped byte select signals for other memory banks or ports. Figure 9-6 provides sample equations for the PAL.

The PAL equations and circuits presented here cannot be the optimal implementation for every system. Depending on the CPU clock frequency, memory access times, and system architecture, different circuits may be required.

Table 10-1. θ_{JA} vs. Airflow—MC68020 CQFP Package

θ_{JA}	Airflow in Linear Feet/Minute		
	0*	200	500
Maximum			
No Heatsink	46	28	24
With Heatsink	35	20	18
Typical			
No Heatsink	43	25	21
With Heatsink	32	17	15

*Natural convection

Table 10-2 shows the relationship between clock speed and power dissipation for any package type. The worst case operating conditions are used for thermal management design, while typical values are used for reliability analysis.

Table 10-2. Power vs. Rated Frequency (at T_J Maximum = 110°C)

Rated Frequency (MHz)	P_D Maximum (Watts)	P_D Typical (Watts)
33	1.4	0.84
25	1.2	0.72
20	1.0	0.60
16	0.9	0.54

Table 10-3 shows the relationship between board temperature rise and power dissipation in the test environment for the CQFP package. Derate θ_{JA} based on measurements made in the application by adding $(0.8/P_D) * [T_{ba(application)} - T_{ba(table)}]$ to the θ_{JA} values in the table. Board temperature was measured on the top surface of the board directly under the device.

Table 10-3. Temperature Rise of Board vs. P_D —MC68020 CQFP Package

Natural Convection	P_D		
	0.6W	1.0W	1.75W
T_{ba} (°C)—No Heatsink	18	27	53

Values for thermal resistance presented in this document were derived using the procedure described in Motorola Reliability Report 7843, “Thermal Resistance Measurement Method for MC68XX Microcomponent Devices,” and are provided for design purposes only. Thermal measurements are complex and dependent on procedure and setup. User-derived values for thermal resistance may differ.