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	33MHz
Co-Processors/DSP	-
RAM Controllers	-
Graphics Acceleration	No
Display & Interface Controllers	-
Ethernet	-
SATA	-
USB	-
Voltage - I/O	5.0V
Operating Temperature	0°C ~ 70°C (TA)
Security Features	-
Package / Case	132-BCQFP
Supplier Device Package	132-CQFP (24x24)
Purchase URL	https://www.e-xfl.com/product-detail/nxp-semiconductors/mc68020fe33e

PREFACE

The *M68020 User's Manual* describes the capabilities, operation, and programming of the MC68020 32-bit, second-generation, enhanced microprocessor and the MC68EC020 32-bit, second-generation, enhanced embedded microprocessor.

Throughout this manual, "MC68020/EC020" is used when information applies to both the MC68020 and the MC68EC020. "MC68020" and "MC68EC020" are used when information applies only to the MC68020 or MC68EC020, respectively.

For detailed information on the MC68020 and MC68EC020 instruction set, refer to M68000PM/AD, *M68000 Family Programmer's Reference Manual*.

This manual consists of the following sections:

Section 1	Introduction
Section 2	Processing States
Section 3	Signal Description
Section 4	On-Chip Cache Memory
Section 5	Bus Operation
Section 6	Exception Processing
Section 7	Coprocessor Interface Description
Section 8	Instruction Execution Timing
Section 9	Applications Information
Section 10	Electrical Characteristics
Section 11	Ordering Information and Mechanical Data
Appendix A	Interfacing an MC68EC020 to a DMA Device That Supports a Three-Wire Bus Arbitration Protocol

NOTE

In this manual, *assert* and *negate* are used to specify forcing a signal to a particular state. In particular, *assertion* and *assert* refer to a signal that is active or true; *negation* and *negate* indicate a signal that is inactive or false. These terms are used independently of the voltage level (high or low) that they represent.

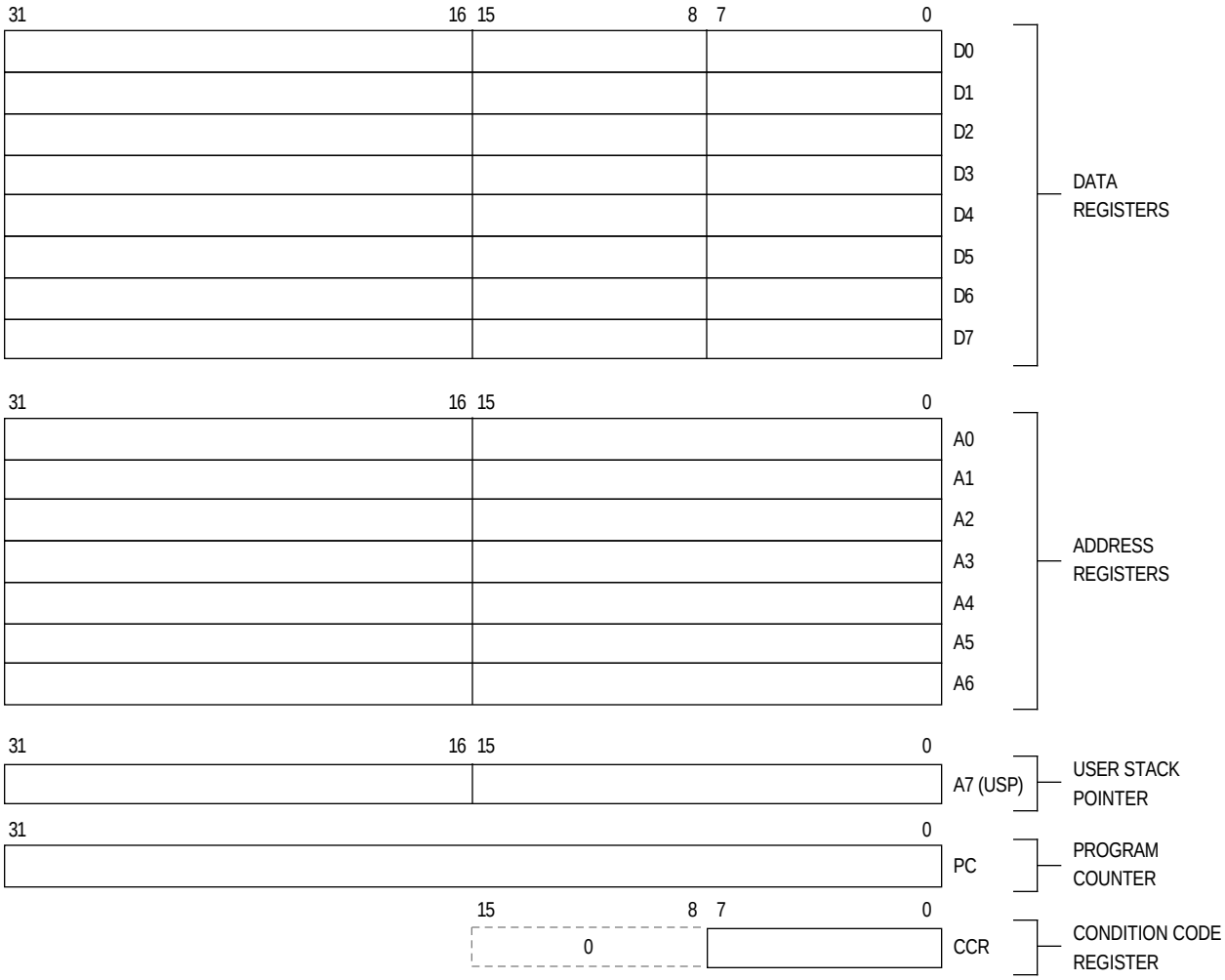


Figure 1-2. User Programming Model

Table 1-1. Addressing Modes

Addressing Modes	Syntax
Register Direct	
Data	Dn
Address	An
Register Indirect	
Address	(An)
Address with Postincrement	(An)+
Address with Predecrement	–(An)
Address with Displacement	(d_{16}, An)
Address Register Indirect with Index	
8-Bit Displacement	(d_8, An, Xn)
Base Displacement	(bd, An, Xn)
Memory Indirect	
Postindexed	([bd, An], Xn, od)
Preindexed	([bd, An, Xn], od)
PC Indirect with Displacement	(d_{16}, PC)
PC Indirect with Index	
8-Bit Displacement	(d_8, PC, Xn)
Base Displacement	(bd, PC, Xn)
PC Indirect	
Postindexed	([bd, PC], Xn, od)
Preindexed	([bd, PC, Xn], od)
Absolute Data Addressing	
Short	(xxx).W
Long	(xxx).L
Immediate	#<data>

NOTE:

Dn = Data Register, D7–D0

An = Address Register, A7–A0

d_8, d_{16} = A twos complement or sign-extended displacement added as part of the effective address calculation; size is 8 (d_8) or 16 (d_{16}) bits; when omitted, assemblers use a value of zero.

Xn = Address or data register used as an index register; form is Xn.SIZE∗SCALE, where SIZE is .W or .L (indicates index register size) and SCALE is 1, 2, 4, or 8 (index register is multiplied by SCALE); use of SIZE and/or SCALE is optional.

bd = A twos-complement base displacement; when present, size can be 16 or 32 bits.

od = Outer displacement added as part of effective address calculation after any memory indirection; use is optional with a size of 16 or 32 bits.

PC = Program Counter

<data> = Immediate value of 8, 16, or 32 bits

( ) = Effective Address

[ ] = Use as indirect access to long-word address.

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

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

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

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

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

5.1.2 Address Bus

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

5.1.3 Address Strobe

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

5.1.4 Data Bus

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

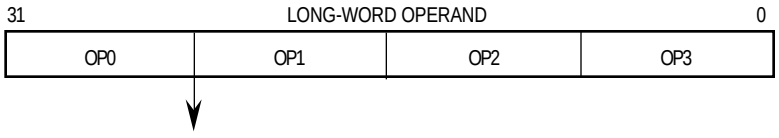
Table 5-5 lists the combinations of SIZ1, SIZ0, A1, and A0 and the corresponding pattern of the data transfer for write cycles from the internal multiplexer of the MC68020/EC020 to the external data bus.

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.



WORD MEMORY		MC68020/EC020				MEMORY CONTROL	
MSB	LSB	SIZ1	SIZ0	A1	A0	$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$
OP0	OP1	0	0	0	0	L	H
OP2	OP3	1	0	1	0	L	H

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

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

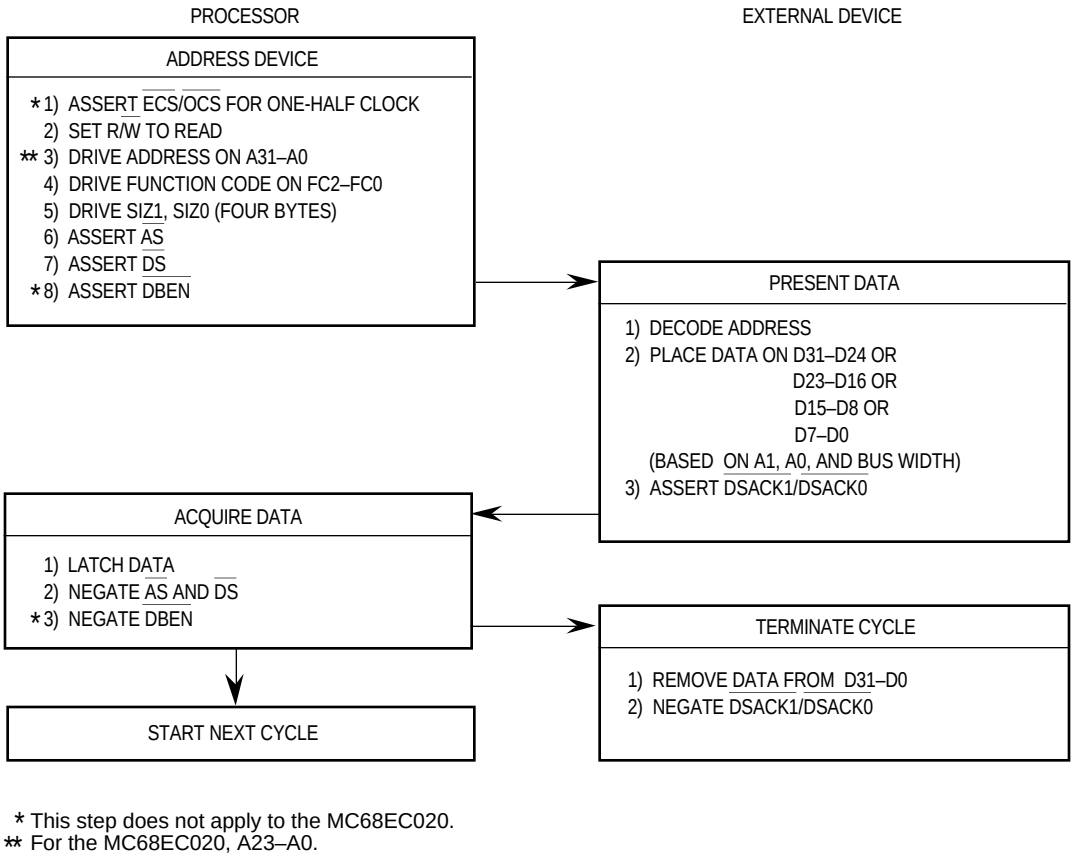
Figure 5-18 shows a logic diagram of one method for generating byte enable signals for 16- and 32-bit ports from the SIZ1, SIZ0, A1, and A0 encodings and the R/$\overline{W}$ signal.

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.

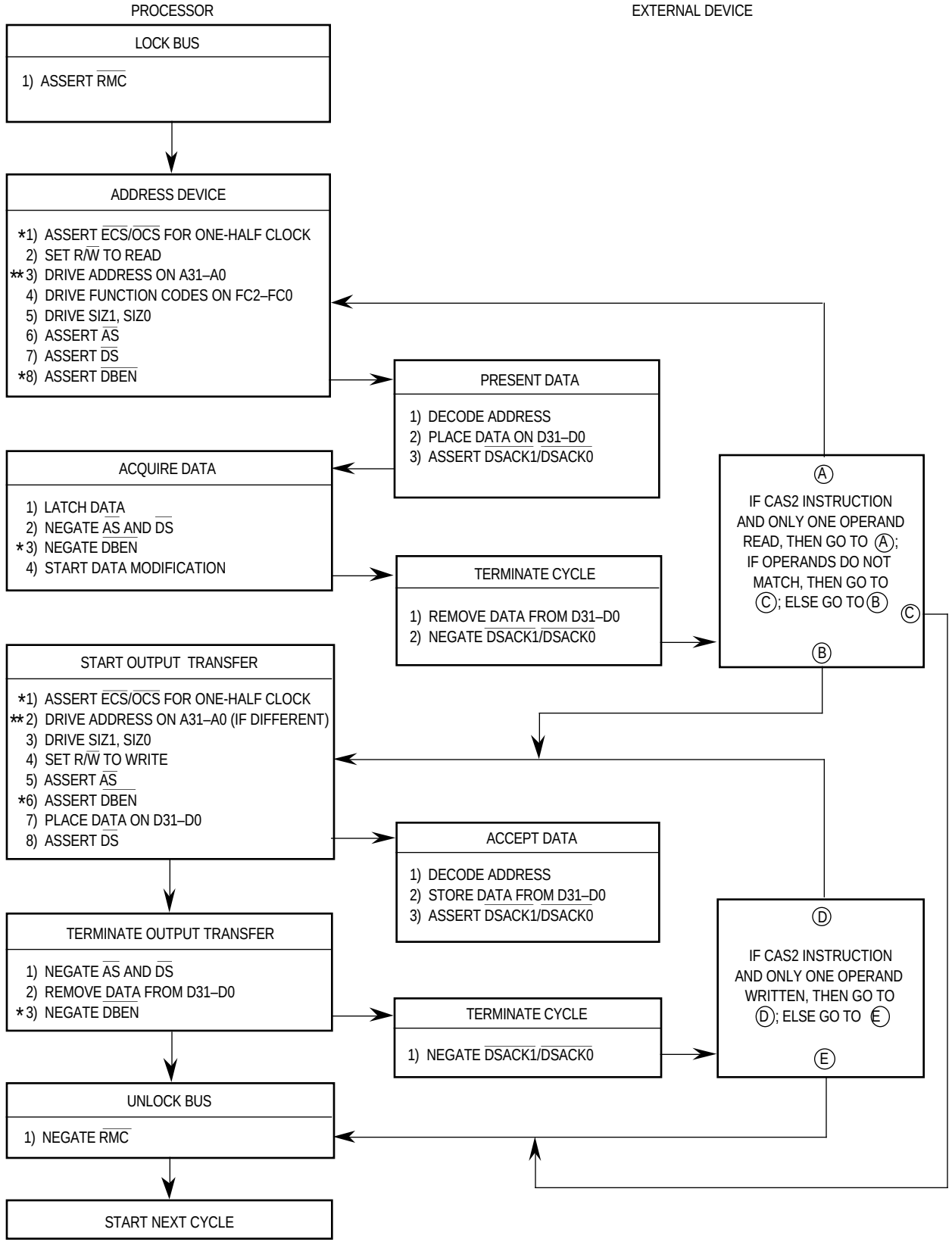
The cache can also affect the assertion of $\overline{AS}$ and the operation of a read cycle. The search of the cache by the processor begins when the sequencer requires an instruction. At this time, the bus controller may also initiate an external bus cycle in case the requested item is not resident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



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

Figure 5-20. Byte Read Cycle Flowchart



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

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

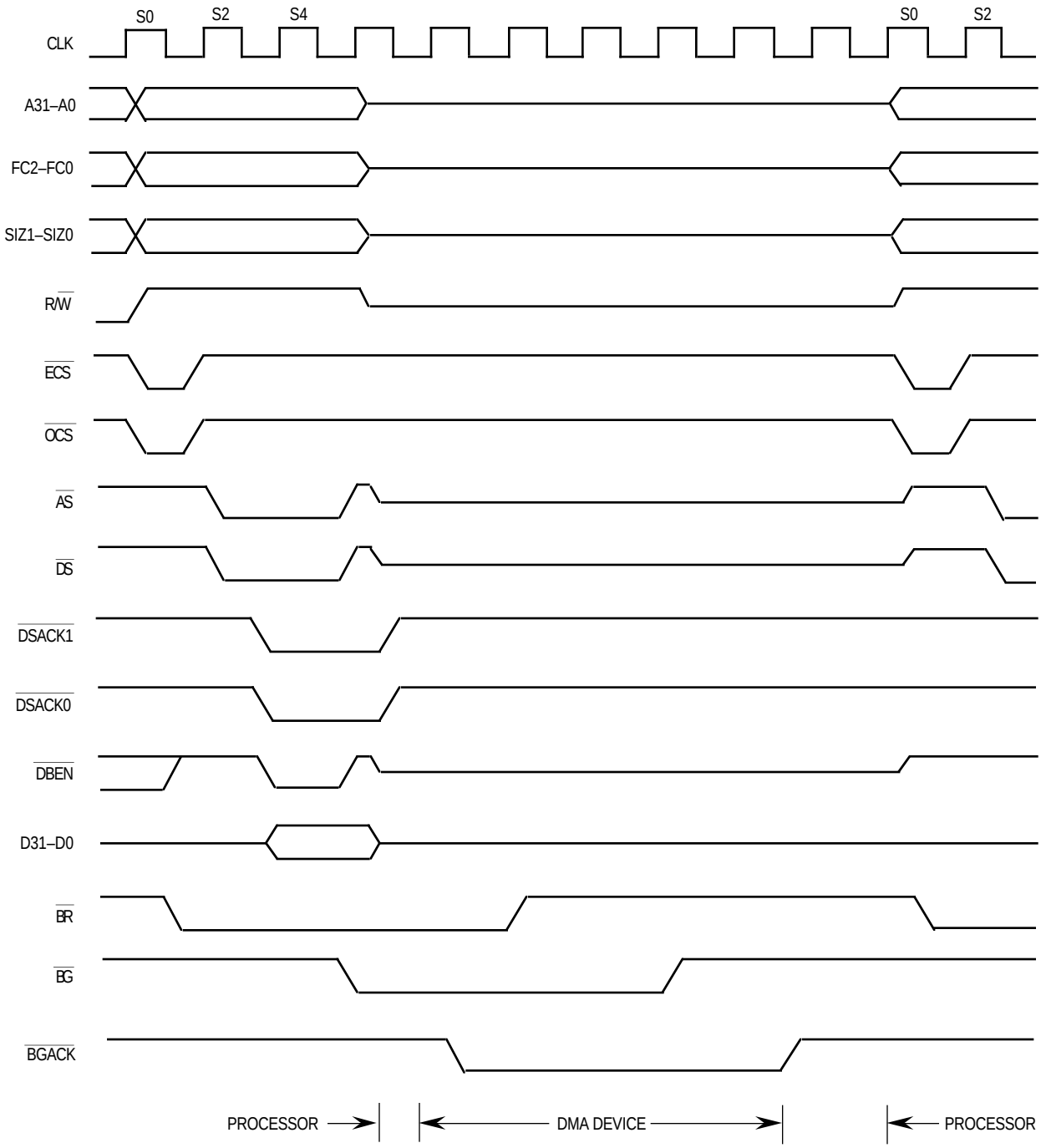


Figure 5-43. MC68020 Bus Arbitration Operation Timing for Single Request

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.

execution of instructions. The processor also saves the contents of some of its internal registers. The information saved on the stack is sufficient to identify the cause of the bus fault and recover from the error.

For efficiency, the MC68020/EC020 uses two different bus error stack frame formats. When the bus error exception is taken at an instruction boundary, less information is required to recover from the error, and the processor builds the short bus fault stack frame as shown in Table 6-5. When the exception is taken during the execution of an instruction, the processor must save its entire state for recovery and uses the long bus fault stack frame shown in Table 6-5. The format code in the stack frame distinguishes the two stack frame formats. Stack frame formats are described in detail in **6.4 Exception Stack Frame Formats**.

If a bus error occurs during the exception processing for a bus error, address error, or reset or while the processor is loading internal state information from the stack during the execution of an RTE instruction, a double bus fault occurs and the processor enters the halted state. In this case, the processor does not attempt to alter the current state of memory. Only an external RESET can restart a processor halted by a double bus fault.

6.1.3 Address Error Exception

An address error exception occurs when the processor attempts to prefetch an instruction from an odd address. This exception is similar to a bus error exception but is internally initiated. A bus cycle is not executed, and the processor begins exception processing immediately. After exception processing commences, the sequence is the same as that for bus error exceptions described in the preceding paragraphs, except that the vector number is 3 and the vector offset in the stack frame refers to the address error vector. Either a short or long bus fault stack frame may be generated. If an address error occurs during the exception processing for a bus error, address error, or reset, a double bus fault occurs.

6.1.4 Instruction Trap Exception

Certain instructions are used to explicitly cause trap exceptions. The TRAP instruction always forces an exception and is useful for implementing system calls in user programs. The TRAPcc, TRAPV, cpTRAPcc, CHK, and CHK2 instructions force exceptions if the user program detects an error, which may be an arithmetic overflow or a subscript value that is out of bounds.

The DIVS and DIVU instructions force exceptions if a division operation is attempted with a divisor of zero.

When a trap exception occurs, the processor copies the SR internally, enters the supervisor privilege level (by setting the S-bit in the SR), and clears the T1 and T0 bits in the SR. If tracing is enabled for the instruction that caused the trap, a trace exception is taken after the RTE instruction from the trap handler is executed, and the trace corresponds to the trap instruction; the trap handler routine is not traced. The processor generates a vector number according to the instruction being executed; for the TRAP

6.1.7 Trace Exception

To aid in program development, the M68000 processors include an instruction-by-instruction tracing capability. The MC68020/EC020 can be programmed to trace all instructions or only instructions that change program flow. In the trace mode, an instruction generates a trace exception after it completes execution, allowing a debugger program to monitor execution of a program.

The T1 and T0 bits in the supervisor portion of the SR control tracing. The state of these bits when an instruction begins execution determines whether the instruction generates a trace exception after the instruction completes. Clearing both the T1 and T0 bits disables tracing, and instruction execution proceeds normally. Clearing the T1 bit and setting the T0 bit causes an instruction that forces a change of flow to take a trace exception. Instructions that increment the PC normally do not take the trace exception. Instructions that are traced in this mode include all branches, jumps, instruction traps, returns, and coprocessor instructions that modify the PC flow. This mode also includes SR manipulations because the processor must re-prefetch instruction words to fill the pipe again any time an instruction that can modify the SR is executed. The execution of the BKPT instruction causes a change of flow if the opcode replacing the BKPT is an instruction that causes a change of flow (i.e., a jump, branch, etc.). Setting the T1 bit and clearing the T0 bit causes the execution of all instructions to force trace exceptions. Table 6-2 shows the trace mode selected by each combination of T1 and T0.

Table 6-2. Tracing Control

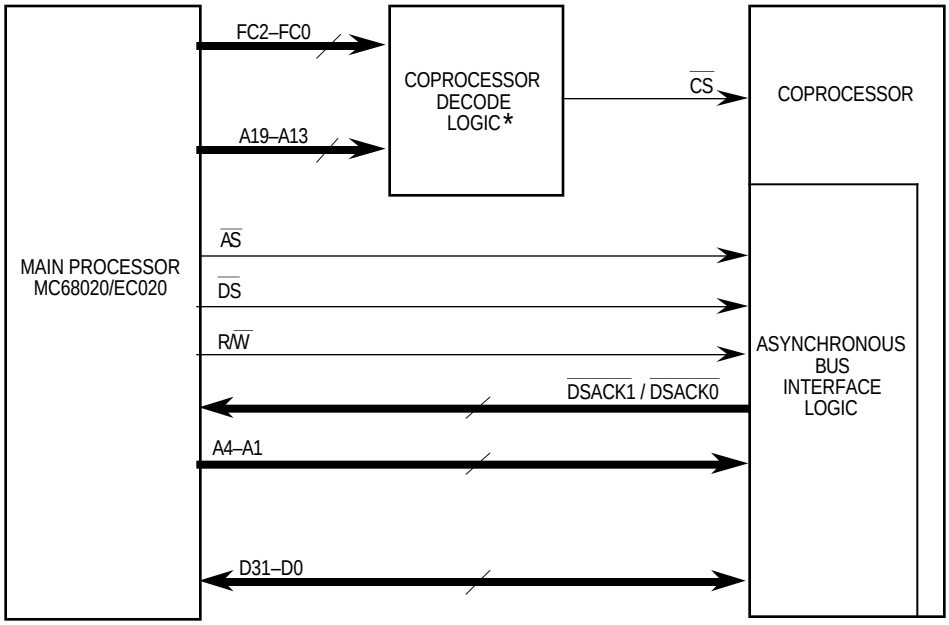
TI	T0	Tracing Function
0	0	No Tracing
0	1	Trace on Change of Flow (BRA, JMP, etc.)
1	0	Trace on Instruction Execution (Any Instruction)
1	1	Undefined, Reserved

In general terms, a trace exception is an extension to the function of any traced instruction—i.e., the execution of a traced instruction is not complete until completion of trace exception processing. If an instruction does not complete due to a bus error or address error exception, trace exception processing is deferred until after the execution of the suspended instruction is resumed, and the instruction execution completes normally. If an interrupt is pending at the completion of an instruction, the trace exception processing occurs before the interrupt exception processing starts. If an instruction forces an exception as part of its normal execution, the forced exception processing occurs before the trace exception is processed. See **6.1.11 Multiple Exceptions** for a more complete discussion of exception priorities.

When tracing is enabled and the processor attempts to execute an illegal or unimplemented instruction, that instruction does not cause a trace exception since it is not executed. This is of particular importance to an instruction emulation routine that performs the instruction function, adjusts the stacked PC to skip the unimplemented instruction, and returns. Before returning, the T1 and T0 bits of the SR on the stack should be checked. If

To improve the efficiency of operand transfers between memory and the coprocessor, a coprocessor that requires a relatively high amount of bus bandwidth or has special bus requirements can be implemented as a DMA coprocessor. The DMA coprocessor provides all control, address, and data signals necessary to request and obtain the bus and then performs DMA transfers using the bus. DMA coprocessors, however, must still act as bus slaves when they require information or services of the main processor using the M68000 coprocessor interface protocol.

7.1.4.2 PROCESSOR-COPROCESSOR INTERFACE. Figure 7-2 is a block diagram of the signals involved in an asynchronous non-DMA M68000 coprocessor interface. Since the CpID on signals A15–A13 of the address bus is used with other address signals to select the coprocessor, the system designer can use several coprocessors of the same type and assign a unique CpID to each one.



FC2–FC0 = 111 ➧ CPU SPACE CYCLE
A19–A16 = 0010 ➧ COPROCESSOR ACCESS IN CPU SPACE
A15–A13 = xxx ➧ COPROCESSOR IDENTIFICATION
A4–A1 = rrrr ➧ COPROCESSOR INFERFACE REGISTER SELECTOR

*Chip select logic may be integrated into the coprocessor.
Address lines not specified above are "0" during coprocessor access.

Figure 7-2. Asynchronous Non-DMA M68000 Coprocessor Interface Signal Usage

The MC68020/EC020 accesses the registers in the CIR set using standard asynchronous bus cycles. Thus, the bus interface implemented by a coprocessor for its interface register set must satisfy the MC68020/EC020 address, data, and control signal timing. The MC68020/EC020 bus operation is described in detail in **Section 5 Bus Operation**.

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.

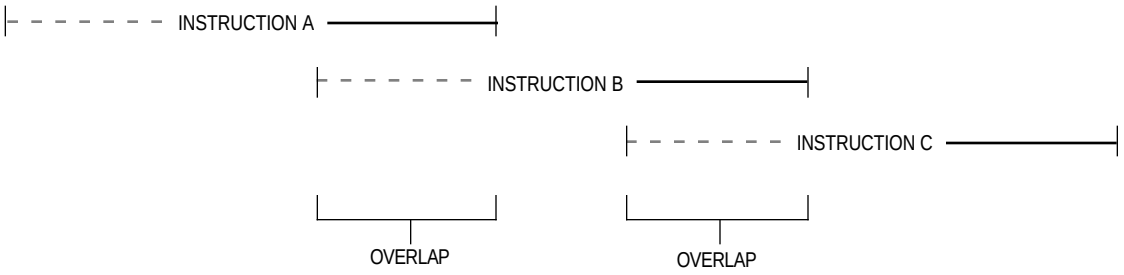


Figure 8-1. Concurrent Instruction Execution

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

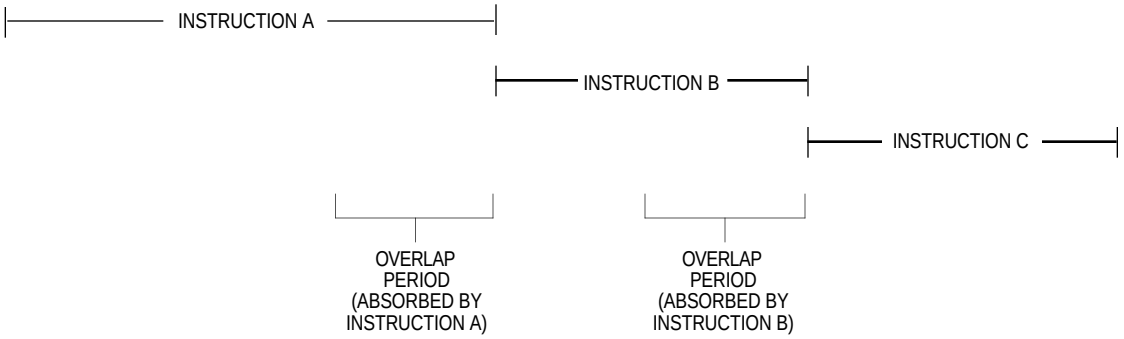


Figure 8-2. Instruction Execution for Instruction Timing Purposes

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.

CACHE CASE (Concluded)

Source Address Mode	Destination					
	([d_{16},B],I)	([d_{16},B],I,d_{16})	([d_{16},B],I,d_{32})	([d_{32},B],I)	([d_{32},B],I,d_{16})	([d_{32},B],I,d_{32})
Rn	**14**(1/0/1)	**16**(1/0/1)	**17**(1/0/1)	**18**(1/0/1)	**20**(1/0/1)	**21**(1/0/1)
#<data>.B,W	**14**(1/0/1)	**16**(1/0/1)	**17**(1/0/1)	**18**(1/0/1)	**20**(1/0/1)	**21**(1/0/1)
#<data>.L	**16**(1/0/1)	**18**(1/0/1)	**19**(1/0/1)	**20**(1/0/1)	**22**(1/0/1)	**23**(1/0/1)
(An)	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)
(An)+	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)
–(An)	**17**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)	**21**(2/0/1)	**23**(2/0/1)	**24**(2/0/1)
(d_{16},An) or (d_{16},PC)	**17**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)	**21**(2/0/1)	**23**(2/0/1)	**24**(2/0/1)
(xxx).W	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)
(xxx).L	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)
(d_8,An,Xn) or (d_8,PC,Xn)	**19**(2/0/1)	**21**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)	**25**(2/0/1)	**26**(2/0/1)
(d_{16},An,Xn) or (d_{16},PC,Xn)	**19**(2/0/1)	**21**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)	**25**(2/0/1)	**26**(2/0/1)
(B)	**19**(2/0/1)	**21**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)	**25**(2/0/1)	**26**(2/0/1)
(d_{16},B)	**21**(2/0/1)	**23**(2/0/1)	**24**(2/0/1)	**25**(2/0/1)	**27**(2/0/1)	**28**(2/0/1)
(d_{32},B)	**25**(2/0/1)	**27**(2/0/1)	**28**(2/0/1)	**29**(2/0/1)	**31**(2/0/1)	**32**(2/0/1)
([B],I)	**24**(3/0/1)	**26**(3/0/1)	**27**(3/0/1)	**28**(3/0/1)	**30**(3/0/1)	**31**(3/0/1)
([B],I,d_{16})	**26**(3/0/1)	**28**(3/0/1)	**29**(3/0/1)	**30**(3/0/1)	**32**(3/0/1)	**33**(3/0/1)
([B],I,d_{32})	**26**(3/0/1)	**28**(3/0/1)	**29**(3/0/1)	**30**(3/0/1)	**32**(3/0/1)	**33**(3/0/1)
([d_{16},B],I)	**26**(3/0/1)	**28**(3/0/1)	**29**(3/0/1)	**30**(3/0/1)	**32**(3/0/1)	**33**(3/0/1)
([d_{16},B],I,d_{16})	**28**(3/0/1)	**30**(3/0/1)	**31**(3/0/1)	**32**(3/0/1)	**34**(3/0/1)	**35**(3/0/1)
([d_{16},B],I,d_{32})	**28**(3/0/1)	**30**(3/0/1)	**31**(3/0/1)	**32**(3/0/1)	**34**(3/0/1)	**35**(3/0/1)
([d_{32},B],I)	**30**(3/0/1)	**32**(3/0/1)	**33**(3/0/1)	**34**(3/0/1)	**36**(3/0/1)	**37**(3/0/1)
([d_{32},B],I,d_{16})	**32**(3/0/1)	**34**(3/0/1)	**35**(3/0/1)	**36**(3/0/1)	**38**(3/0/1)	**39**(3/0/1)
([d_{32},B],I,d_{32})	**32**(3/0/1)	**34**(3/0/1)	**35**(3/0/1)	**36**(3/0/1)	**38**(3/0/1)	**39**(3/0/1)

8.2.9 Immediate Arithmetic/Logical Instructions

The immediate arithmetic/logical instructions table indicates the number of clock periods needed for the processor to fetch the source immediate data value and perform the specified arithmetic/logical operation using the specified destination addressing mode. Footnotes indicate when to add appropriate fetch effective or fetch immediate effective address time. This computation will give the total execution time needed to perform the appropriate immediate arithmetic/logical operation. 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
	MOVEQ	#<data>,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
	ADDQ	#<data>,Rn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	ADDQ	#<data>,Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
	SUBQ	#<data>,Rn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	SUBQ	#<data>,Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
**	ADDI	#<data>,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
**	ADDI	#<data>,Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
**	ANDI	#<data>,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
**	ANDI	#<data>,Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
**	EORI	#<data>,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
**	EORI	#<data>,Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
**	ORI	#<data>,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
**	ORI	#<data>,Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
**	SUBI	#<data>,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
**	SUBI	#<data>,Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
**	CMPI	#<data>,EA	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)

*Add Fetch Effective Address Time

**Add Fetch Immediate Address Time

AC ELECTRICAL CHARACTERISTICS—READ AND WRITE CYCLES

(Continued)

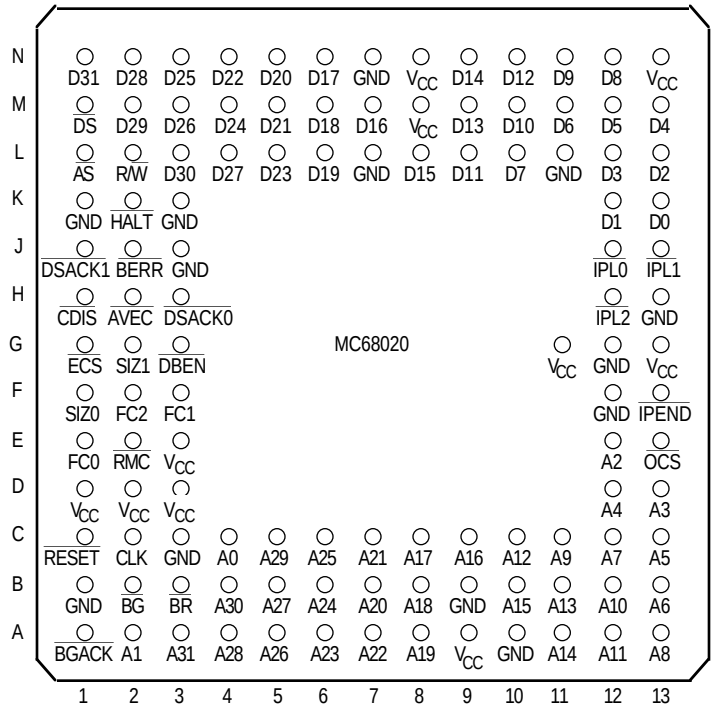
Num.	Characteristics	16.67 MHz		20 MHz		25 MHz**		33.33 MHz		Unit
		Min	Max	Min	Max	Min	Max	Min	Max	
29	AS, DS Negated to Data-In Invalid (Data-In Hold Time)	0	—	0	—	0	—	0	—	ns
29A	AS, DS Negated to Data-In (High Impedance)	—	60	—	50	—	40	—	30	ns
30	Clock Low to Data-In Invalid (Data-In Hold Time)	15	—	15	—	10	—	10	—	ns
31[2]	DSACK≈ Asserted to Data-In Valid	—	50	—	43	—	32	—	17	ns
31A[3]	DSACK≈ Asserted to DSACK≈ Valid (DSACK≈ Asserted Skew)	—	15	—	10	—	10	—	10	ns
32	RESET Input Transition Time	—	1.5	—	1.5	—	1.5	—	1.5	Clks
33	Clock Low to BG Asserted	0	30	0	25	0	20	0	20	ns
34	Clock Low to BG Negated	0	30	0	25	0	20	0	20	ns
35	BR Asserted to BG Asserted (RMC Not Asserted)	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	Clks
*37	BGACK Asserted to BG Negated	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	Clks
*37A[6]	BGACK Asserted to BR Negated	0	1.5	0	1.5	0	1.5	0	1.5	Clks
39	BG Width Negated	90	—	75	—	60	—	50	—	ns
39A	BG Width Asserted	90	—	75	—	60	—	50	—	ns
*40	Clock High to DBEN Asserted (Read)	0	30	0	25	0	20	0	15	ns
*41	Clock Low to DBEN Negated (Read)	0	30	0	25	0	20	0	15	ns
*42	Clock Low to DBEN Asserted (Write)	0	30	0	25	0	20	0	15	ns
*43	Clock High to DBEN Negated (Write)	0	30	0	25	0	20	0	15	ns
*44	R/W Low to DBEN Asserted (Write)	15	—	10	—	10	—	5	—	ns
*45[5]	DBEN Width Asserted Read Write	60 120	— —	50 100	— —	40 80	— —	30 60	— —	ns
46	R/W Width Valid (Write or Read)	150	—	125	—	100	—	75	—	ns
47A	Asynchronous Input Setup Time	5	—	5	—	5	—	5	—	ns
47B	Asynchronous Input Hold Time	15	—	15	—	10	—	10	—	ns
48[4]	DSACK≈ Asserted to BERR, HALT Asserted	—	30	—	20	—	18	—	15	ns
53	Data-Out Hold from Clock High	0	—	0	—	0	—	0	—	ns
55	R/W Valid to Data Bus Impedance Change	30	—	25	—	20	—	20	—	ns
56	RESET Pulse Width (Reset Instruction)	512	—	512	—	512	—	512	—	Clks
57	BERR Negated to HALT Negated (Rerun)	0	—	0	—	0	—	0	—	ns
*58[10]	BGACK Negated to Bus Driven	1	—	1	—	1	—	1	—	Clks
59[10]	BG Negated to Bus Driven	1	—	1	—	1	—	1	—	Clks

*This specification does not apply to the MC68EC020.

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

11.2 PIN ASSIGNMENTS AND PACKAGE DIMENSIONS

11.2.1 MC68020 RC and RP Suffix—Pin Assignment

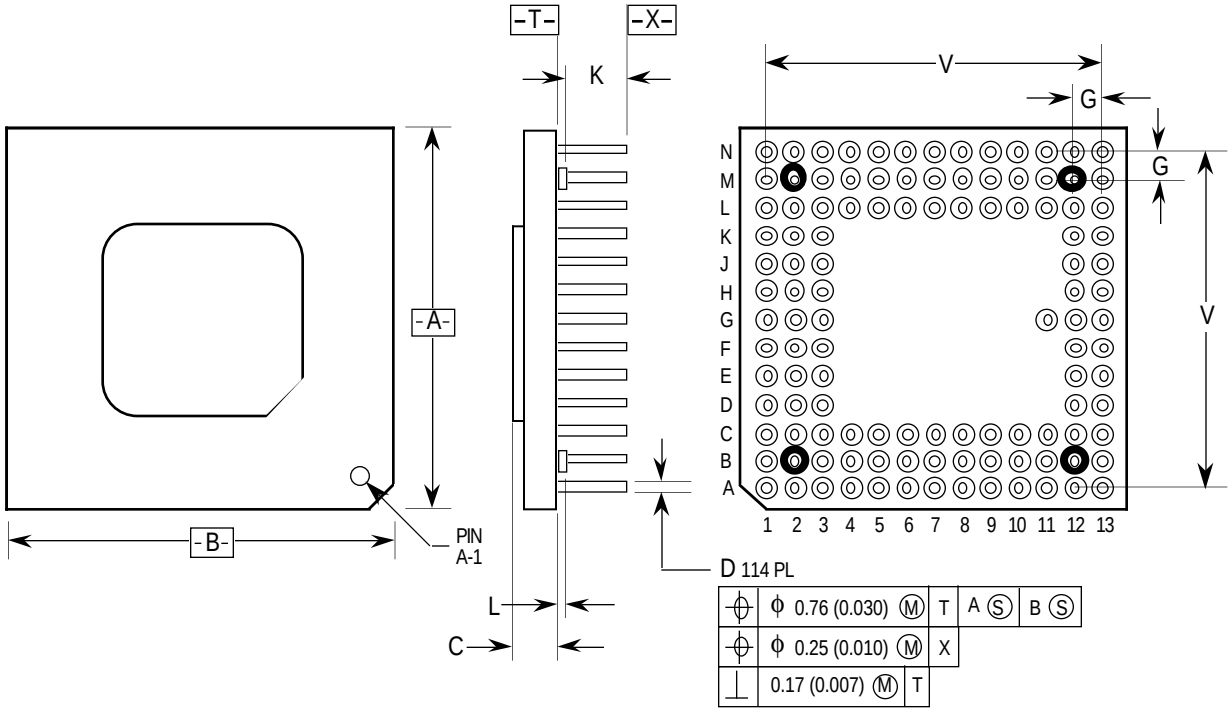


The V_{CC} and GND pins are separated into four groups to provide individual power supply connections for the address bus buffers, data bus buffers, and all other output buffers and internal logic. It is recommended that all pins be connected to power and ground as indicated.

Group	V_{CC}	GND
Address Bus	A9, D3	A10, B9, C3, F12
Data Bus	M8, N8, N13	L7, L11, N7, K3
Logic	D1, D2, E3, G11, G13	G12, H13, J3, K1
Clock	—	B1

11.2.3 MC68020 RP Suffix—Package Dimensions

RP SUFFIX
CASE 789E-02
MC68020



DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	34.04	35.05	1.340	1.380
B	34.04	35.05	1.340	1.380
C	2.92	3.18	0.115	0.135
D	0.44	0.55	0.017	0.022
G	2.54 BSC		0.100 BSC	
K	2.79	3.81	0.110	0.150
L	1.02	1.52	0.040	0.060
V	30.48 BSC		1.200 BSC	

NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: INCH.
3. DIMENSION D INCLUDES LEAD FINISH.
4. 789E-01 OBSOLETE. NEW STANDARD 789E-02.