Welcome to E-XFL.COM

Understanding Embedded - Microprocessors

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

Applications of Embedded - Microprocessors

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

Details	
Product Status	Active
Core Processor	68020
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	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	114-BPGA
Supplier Device Package	114-PGA (34.55x34.55)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mc68020rc33e

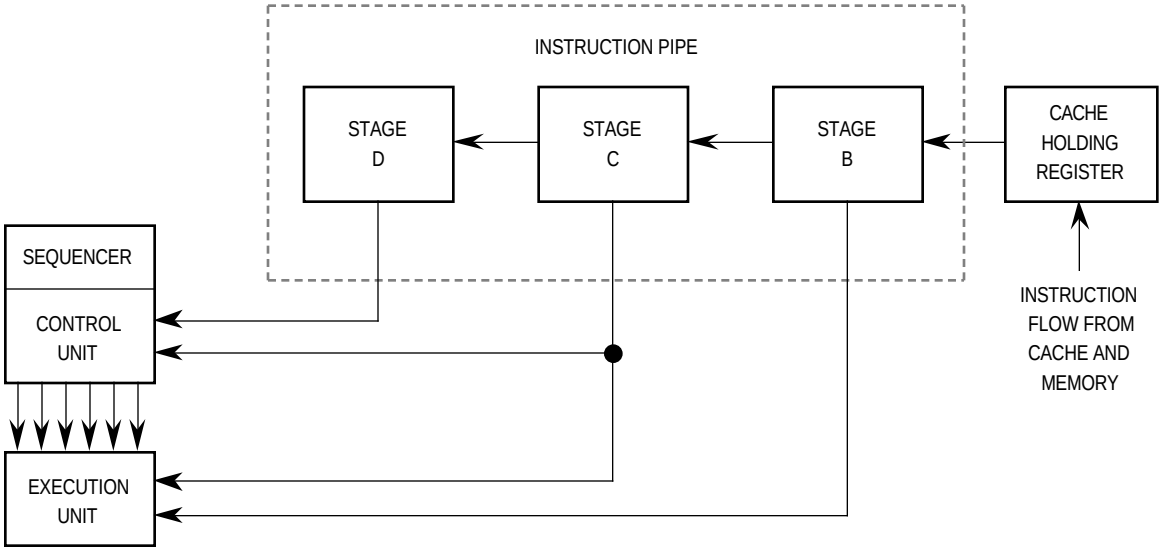


Figure 1-5. Instruction Pipe

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

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

1.7 CACHE MEMORY

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

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

2.3.2 Exception Stack Frame

Exception processing saves the most volatile portion of the current processor context on the top of the supervisor stack. This context is organized in a format called the exception stack frame. This information always includes a copy of the SR, the PC, the vector offset of the vector, and the frame format field. The frame format field identifies the type of stack frame. The RTE instruction uses the value in the format field to properly restore the information stored in the stack frame and to deallocate the stack space. The general form of the exception stack frame is illustrated in Figure 2-1. Refer to **Section 6 Exception Processing** for a complete list of exception stack frames.

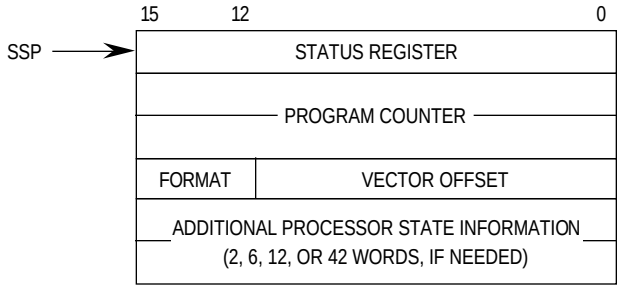


Figure 2-1. General Exception Stack Frame

SECTION 5
BUS OPERATION

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

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

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

5.1 BUS TRANSFER SIGNALS

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

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

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

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

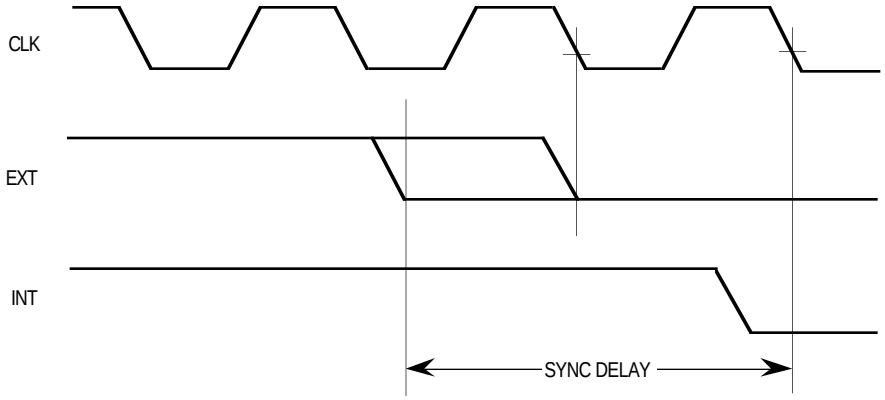


Figure 5-1. Relationship between External and Internal Signals

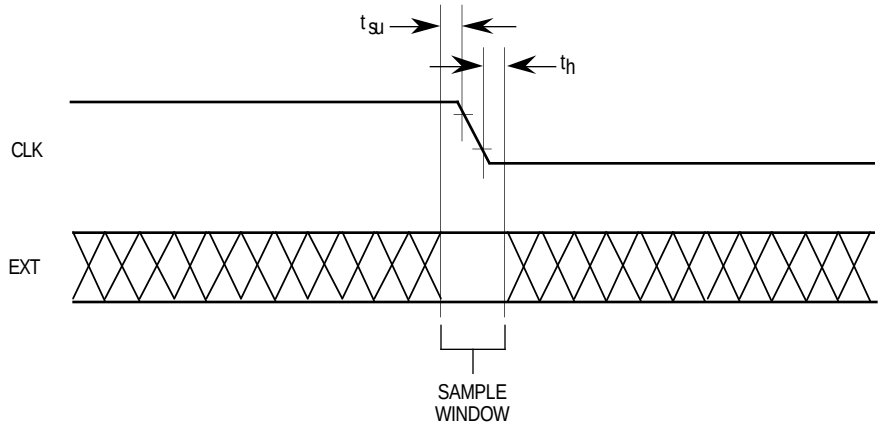


Figure 5-2. Input Sample Window

5.1.1 Bus Control Signals

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

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.

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.

5.7.2 MC68EC020 Bus Arbitration

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

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

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

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

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

Bus arbitration requests are recognized during normal processing, $\overline{RESET}$ assertion, $\overline{HALT}$ assertion, and when the processor has halted due to a double bus fault.

After writing the format word to the restore CIR, the main processor continues cpRESTORE dialog by reading that same register. If the coprocessor returns a valid format word, the main processor transfers the number of bytes specified by the format word at the effective address to the operand CIR.

If the format word written to the restore CIR does not represent a valid coprocessor state frame, the coprocessor places an invalid format word in the restore CIR and terminates any current operations. The main processor receives the invalid format code, writes an abort mask (refer to **7.2.3.2.3 Invalid Format Word**) to the control CIR, and initiates format error exception processing (refer to **7.5.1.5 Format Errors**).

The cpRESTORE instruction is a privileged instruction. When the MC68020/EC020 accesses a cpRESTORE instruction, it checks the S-bit in the SR. If the MC68020/EC020 attempts to execute a cpRESTORE instruction while at the user privilege level (S-bit in the SR is clear), it initiates privilege violation exception processing without accessing any of the CIRs (refer to **7.5.2.3 Privilege Violations**).

7.3 COPROCESSOR INTERFACE REGISTER SET

The instructions of the M68000 coprocessor interface use registers of the CIR set to communicate with the coprocessor. These CIRs are not directly related to the coprocessor programming model.

Figure 7-4 is a memory map of the CIR set. The response, control, save, restore, command, condition, and operand registers must be included in a coprocessor interface that implements all four coprocessor instruction categories. The complete register model must be implemented if the system uses all coprocessor response primitives defined for the M68000 coprocessor interface.

The following paragraphs contain detailed descriptions of the registers.

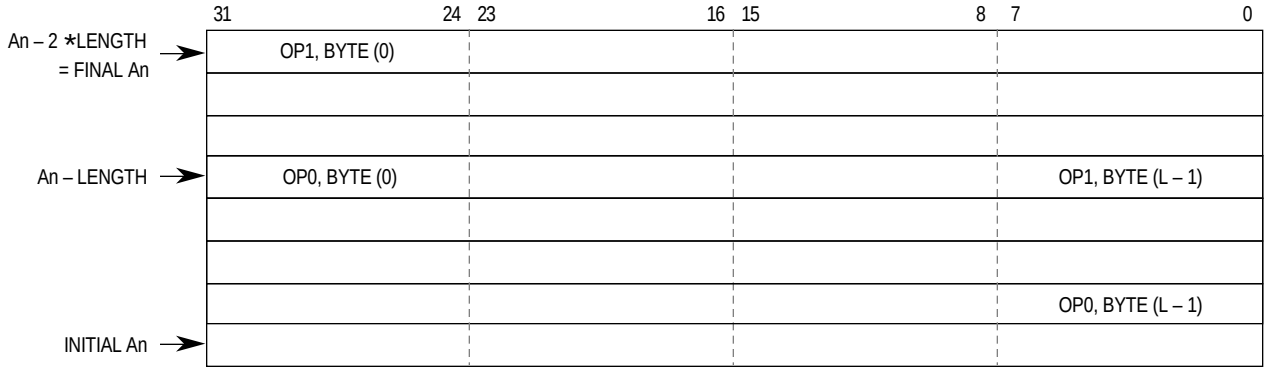
7.3.1 Response CIR

The coprocessor uses the 16-bit response CIR to communicate all service requests (coprocessor response primitives) to the main processor. The main processor reads the response CIR to receive the coprocessor response primitives during the execution of instructions in the general and conditional instruction categories. The offset from the base address of the CIR set for the response CIR is $00. Refer to **7.4 Coprocessor Response Primitives** for additional information.

7.3.2 Control CIR

The main processor writes to the 2-bit control CIR to acknowledge coprocessor-requested exception processing or to abort the execution of a coprocessor instruction. The offset from the base address of the CIR set for the control CIR is $02. The control CIR occupies the two least significant bits of the word at that offset. The 14 most significant bits of the word are undefined and reserved by Motorola. Figure 7-19 shows the format of this register.

For the predecrement addressing mode, the operands are written to memory with descending addresses, but the bytes within each operand are written to memory with ascending addresses. As an example, Figure 7-38 shows the format in long-word-oriented memory for two 12-byte operands transferred from the coprocessor to the effective address using the –(An) addressing mode. The processor decrements the address register by the size of an operand before the operand is transferred. It writes the bytes of the operand to ascending memory addresses. When the transfer is complete, the address register has been decremented by the total number of bytes transferred. The MC68020/EC020 transfers the data using long-word transfers whenever possible.



NOTE: OP0, Byte (0) is the first byte written to memory
OP0, Byte (L–1) is the last byte of the first operand written to memory
OP1, Byte (0) is the first byte of the second operand written to memory
OP1, Byte (L–1) is the last byte written to memory

Figure 7-38. Operand Format in Memory for Transfer to –(An)

7.4.17 Transfer Status Register and ScanPC Primitive

The transfer status register and the scanPC primitive transfers values between the coprocessor and the MC68020/EC020 SR. On an optional basis, the scanPC also makes transfers. This primitive applies to general category instructions. If the coprocessor issues this primitive during the execution of a conditional category instruction, the main processor initiates protocol violation exception processing. Figure 7-39 shows the format of the transfer status register and scanPC primitive.

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

Figure 7-39. Transfer Status Register and ScanPC Primitive Format

The transfer status register and scanPC primitive uses the CA, PC, and DR bits as described in **7.4.2 Coprocessor Response Primitive General Format**.

The SP bit selects the scanPC option. If SP = 1, the primitive transfers both the scanPC and SR. If SP = 0, only the SR is transferred.

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 four-word stack frame format shown in Figure 7-41.

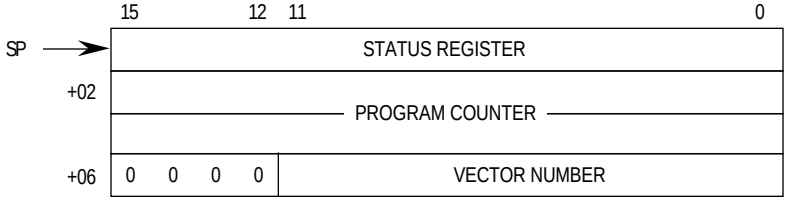


Figure 7-41. MC68020/EC020 Preinstruction Stack Frame

The value of the PC saved in this stack frame is the F-line operation word address of the coprocessor instruction during which the primitive was received. Thus, if the exception handler routine does not modify the stack frame, an RTE instruction causes the MC68020/EC020 to return and reinitiate execution of the coprocessor instruction.

The take preinstruction exception primitive can be used when the coprocessor does not recognize a value written to either its command CIR or condition CIR to initiate a coprocessor instruction. This primitive can also be used if an exception occurs in the coprocessor instruction before any program-visible resources are modified by the instruction operation. This primitive should not be used during a coprocessor instruction if program-visible resources have been modified by that instruction. Otherwise, since the MC68020/EC020 reinitiates the instruction when it returns from exception processing, the restarted instruction receives the previously modified resources in an inconsistent state.

One of the most important uses of the take preinstruction exception primitive is to signal an exception condition in a cpGEN instruction that was executing concurrently with the main processor's instruction execution. If the coprocessor no longer requires the services of the main processor to complete a cpGEN instruction and if the concurrent instruction completion is transparent to the programming model, the coprocessor can release the main processor by issuing a primitive with CA = 0. The main processor usually executes the next instruction in the instruction stream, and the coprocessor completes its operations concurrently with the main processor operation. If an exception occurs while the coprocessor is executing an instruction concurrently, the exception is not processed until the main processor attempts to initiate the next general or conditional instruction. After the main processor writes to the command or condition CIR to initiate a general or conditional instruction, it then reads the response CIR. At this time, the coprocessor can return the take preinstruction exception primitive. This protocol allows the main processor to proceed with exception processing related to the previous concurrently executing coprocessor instruction and then return and reinitiate the coprocessor instruction during which the exception was signaled. The coprocessor should record the addresses of all general category instructions that can be executed concurrently with the main processor and that support exception recovery. Since the exception is not reported until the next coprocessor instruction is initiated, the processor usually requires the instruction address to determine

7.5.1.2 COPROCESSOR-DETECTED ILLEGAL COMMAND OR CONDITION WORDS. Illegal coprocessor command or condition words are values written to the command CIR or condition CIR that the coprocessor does not recognize. If a value written to either of these registers is not valid, the coprocessor should return the take preinstruction exception primitive in the response CIR. When it receives this primitive, the main processor takes a preinstruction exception as described in **7.4.18 Take Preinstruction Exception Primitive**. If the exception handler does not modify the main processor stack frame, an RTE instruction causes the MC68020/EC020 to reinitiate the instruction that took the exception. The coprocessor designer should ensure that the state of the coprocessor is not irrecoverably altered by an illegal command or condition exception if the system supports emulation of the unrecognized command or condition word.

All M68000 coprocessors signal illegal command and condition words by returning the take preinstruction exception primitive with the F-line emulator exception vector number 11.

7.5.1.3 COPROCESSOR DATA-PROCESSING-RELATED EXCEPTIONS. Exceptions related to the internal operation of a coprocessor are classified as data-processing-related exceptions. These exceptions are analogous to the divide-by-zero exception defined by M68000 microprocessors and should be signaled to the main processor using one of the three take exception primitives containing an appropriate exception vector number. Which of these three primitives is used to signal the exception is usually determined by the point in the instruction operation where the main processor should continue the program flow after exception processing. Refer to **7.4.18 Take Preinstruction Exception Primitives**, **7.4.19 Take Midinstruction Exception Primitive**, and **7.4.20 Take Postinstruction Exception Primitive**.

7.5.1.4 COPROCESSOR SYSTEM-RELATED EXCEPTIONS. System-related exceptions detected by a DMA coprocessor include those associated with bus activity and any other exceptions (interrupts, for example) occurring external to the coprocessor. The actions taken by the coprocessor and the main processor depend on the type of exception that occurs.

When an address or bus error is detected by a DMA coprocessor, the coprocessor should store any information necessary for the main processor exception handling routines in system-accessible registers. The coprocessor should place one of the three take exception primitives encoded with an appropriate exception vector number in the response CIR. Which of the three primitives is used depends upon the point in the coprocessor instruction at which the exception was detected and the point in the instruction execution at which the main processor should continue after exception processing. Refer to **7.4.18 Take Preinstruction Exception Primitives**, **7.4.19 Take Midinstruction Exception Primitive**, and **7.4.20 Take Postinstruction Exception Primitive**.

Table 7-6. Exceptions Related to Primitive Processing

Primitive	Protocol	F-Line	Other
Busy			
Null			
Supervisory Check* Other: Privilege Violation if S-Bit in the SR = 0			X
Transfer Operation Word*			
Transfer from Instruction Stream* Protocol: If Length Field Is Odd (Zero Length Legal)	X		
Evaluate and Transfer Effective Address Protocol: If Used with Conditional Instruction F-Line: If EA in Opword Is NOT Control Alterable	X	X	
Evaluate Effective Address and Transfer Data Protocol: 1. If Used with Conditional Instructions 2. Length Is Not 1, 2, or 4 and EA = Register Direct 3. If EA = Immediate and Length Odd and Greater Than 1 4. Attempt to Write to Unalterable Address Even if Address Declared Legal in Primitive F-Line: Valid EA Field Does Not Match EA in Opword	X	X	
Write to Previously Evaluated Effective Address Protocol: If Used with Conditional Instruction	X		
Take Address and Transfer Data*			
Transfer to/from Top of Stack* Protocol: Length Field Other Than 1, 2, or 4	X		
Transfer Single Main Processor Register*			
Transfer Main Processor Control Register Protocol: Invalid Control Register Select Code	X		
Transfer Multiple Main Processor Registers*			
Transfer Multiple Coprocessor Registers Protocol: 1. If Used with Conditional Instructions 2. Odd Length Value F-Line: 1. EA Not Control Alterable or (An)+ for CP to Memory Transfer 2. EA Not Control Alterable or –(An) for Memory to CP Transfer	X	X	
Transfer Status and ScanPC Protocol: If Used with Conditional Instruction Other: 1. Trace—Trace Made Pending if MC68020/EC020 in “Trace on Change of Flow” Mode and DR = 1 2. Address Error—If Odd Value Written to ScanPC	X		X
Take Preinstruction, Midinstruction, or Postinstruction Exception Exception Depends on Vector Supplies in Primitive	X	X	X

*Use of this primitive with CA = 0 will cause protocol violation on conditional instructions.

Abbreviations:

EA—Effective Address
CP—Coprocessor

WORST 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	**9**(0/1/1)	**12**(0/2/1)	**10**(0/1/1)	**14**(0/2/1)	**19**(0/2/1)	**14**(1/1/1)	**17**(1/2/1)	**20**(1/2/1)
#<data>.B,W	**9**(0/1/1)	**12**(0/2/1)	**10**(0/1/1)	**14**(0/2/1)	**19**(0/2/1)	**14**(1/1/1)	**17**(1/2/1)	**20**(1/2/1)
#<data>.L	**11**(0/1/1)	**14**(0/2/1)	**12**(0/1/1)	**16**(0/2/1)	**21**(0/2/1)	**16**(1/1/1)	**19**(1/2/1)	**22**(1/2/1)
(An)	**11**(1/1/1)	**14**(1/2/1)	**12**(1/1/1)	**16**(1/2/1)	**21**(1/2/1)	**12**(2/1/1)	**19**(2/2/1)	**22**(2/2/1)
(An)+	**11**(1/1/1)	**14**(1/2/1)	**12**(1/1/1)	**16**(1/2/1)	**21**(1/2/1)	**12**(2/1/1)	**19**(2/2/1)	**22**(2/2/1)
–(An)	**12**(1/1/1)	**15**(1/2/1)	**13**(1/1/1)	**17**(1/2/1)	**22**(1/2/1)	**13**(2/1/1)	**20**(2/2/1)	**23**(2/2/1)
(d_{16},An) or (d_{16},PC)	**13**(1/2/1)	**16**(1/3/1)	**14**(1/2/1)	**18**(1/3/1)	**23**(1/3/1)	**14**(2/2/1)	**21**(2/3/1)	**24**(2/3/1)
(xxx).W	**12**(1/2/1)	**15**(1/3/1)	**13**(1/2/1)	**17**(1/3/1)	**22**(1/3/1)	**13**(2/2/1)	**20**(2/3/1)	**23**(2/3/1)
(xxx).L	**14**(1/2/1)	**17**(1/3/1)	**15**(1/2/1)	**19**(1/3/1)	**24**(1/3/1)	**15**(2/2/1)	**22**(2/3/1)	**25**(2/3/1)
(d_8,An,Xn) or (d_8,PC,Xn)	**15**(1/2/1)	**18**(1/3/1)	**16**(1/2/1)	**20**(1/3/1)	**25**(1/3/1)	**16**(2/2/1)	**23**(2/3/1)	**26**(2/3/1)
(d_{16},An,Xn) or (d_{16},PC,Xn)	**16**(1/2/1)	**19**(1/3/1)	**17**(1/2/1)	**21**(1/3/1)	**26**(1/3/1)	**17**(2/2/1)	**24**(2/3/1)	**27**(2/3/1)
(B)	**16**(1/2/1)	**19**(1/3/1)	**17**(1/2/1)	**21**(1/3/1)	**26**(1/3/1)	**17**(2/2/1)	**24**(2/3/1)	**27**(2/3/1)
(d_{16},B)	**19**(1/2/1)	**22**(1/3/1)	**20**(1/2/1)	**24**(1/3/1)	**29**(1/3/1)	**20**(2/2/1)	**27**(2/3/1)	**30**(2/3/1)
(d_{32},B)	**23**(1/3/1)	**26**(1/4/1)	**24**(1/3/1)	**28**(1/4/1)	**33**(1/4/1)	**24**(2/3/1)	**31**(2/4/1)	**34**(2/4/1)
([B],I)	**20**(2/2/1)	**23**(2/3/1)	**21**(2/2/1)	**25**(2/3/1)	**30**(2/3/1)	**21**(3/2/1)	**28**(3/3/1)	**31**(3/3/1)
([B],I,d_{16})	**23**(2/2/1)	**26**(2/3/1)	**24**(2/2/1)	**28**(2/3/1)	**33**(2/3/1)	**24**(3/2/1)	**31**(3/3/1)	**34**(3/3/1)
([B],I,d_{32})	**24**(2/3/1)	**27**(2/4/1)	**25**(2/3/1)	**29**(2/4/1)	**34**(2/4/1)	**25**(3/3/1)	**32**(3/4/1)	**35**(3/4/1)
([d_{16},B],I)	**23**(2/2/1)	**26**(2/3/1)	**24**(2/2/1)	**28**(2/3/1)	**33**(2/3/1)	**24**(3/2/1)	**31**(3/3/1)	**34**(3/3/1)
([d_{16},B],I,d_{16})	**26**(2/3/1)	**29**(2/4/1)	**27**(2/3/1)	**31**(2/4/1)	**36**(2/4/1)	**27**(3/3/1)	**34**(3/4/1)	**37**(3/4/1)
([d_{16},B],I,d_{32})	**27**(2/3/1)	**30**(2/4/1)	**28**(2/3/1)	**32**(2/4/1)	**37**(2/4/1)	**28**(3/3/1)	**35**(3/4/1)	**38**(3/4/1)
([d_{32},B],I)	**27**(2/3/1)	**30**(2/4/1)	**28**(2/3/1)	**32**(2/4/1)	**37**(2/4/1)	**28**(3/3/1)	**35**(3/4/1)	**38**(3/4/1)
([d_{32},B],I,d_{16})	**29**(2/3/1)	**32**(2/4/1)	**30**(2/3/1)	**34**(2/4/1)	**39**(2/4/1)	**30**(3/3/1)	**37**(3/4/1)	**40**(3/4/1)
([d_{32},B],I,d_{32})	**31**(2/4/1)	**34**(2/5/1)	**32**(2/4/1)	**36**(2/5/1)	**41**(2/5/1)	**32**(3/4/1)	**39**(3/5/1)	**42**(3/5/1)

8.2.8 Arithmetic/Logical Instructions

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

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

*Add Fetch Effective Address Time

**Add Fetch Immediate Address Time

```
PAL16L8
FPCP CS GENERATION CIRCUITRY FOR 25 MHz OPERATION
MOTOROLA INC., AUSTIN, TEXAS
INPUTS:     CLK   ~AS   FC2   FC1   FC0   A19   A18   A17   A16   A15   A14   A13
OUTPUTS:    ~CS   CLKD

!~CS        = FC2          *FC1          *FC0          *!A16     ;cpu space = $7
            *!A19          *!A18         *A17          *!A16     ;coprocessor access = $2
            *!A15          *!A14         *A13                    ;coprocessor id = $1
            *!CLK                                                ;qualified by MPU clock low
            +FC2           *FC1          *FC0                    ;cpu space = $7
            *!A19          *!A18         *A17          *!A16     ;coprocessor access = $2
            *!A15          *!A14         *A13                    ;coprocessor id = $1
            *!~AS                                                ;qualified by address strobe low
            +FC2           *FC1          *FC0                    ;cpu space = $7
            *!A19          *!A18         *A17          *!A16     ;coprocessor access = $2
            *!A15          *!A14         *A13                    ;coprocessor id = $1
            *CLKD                                                ;qualified by CLKD (delayed CLK)

CLKD = CLK
```

Description: There are three terms to the CS generation. The first term denotes the earliest time CS can be asserted. The second term is used to assert CS until the end of the FPCP access. The third term is to ensure that no race condition occurs in case of a late AS.

Figure 9-3. Chip Select PAL Equations

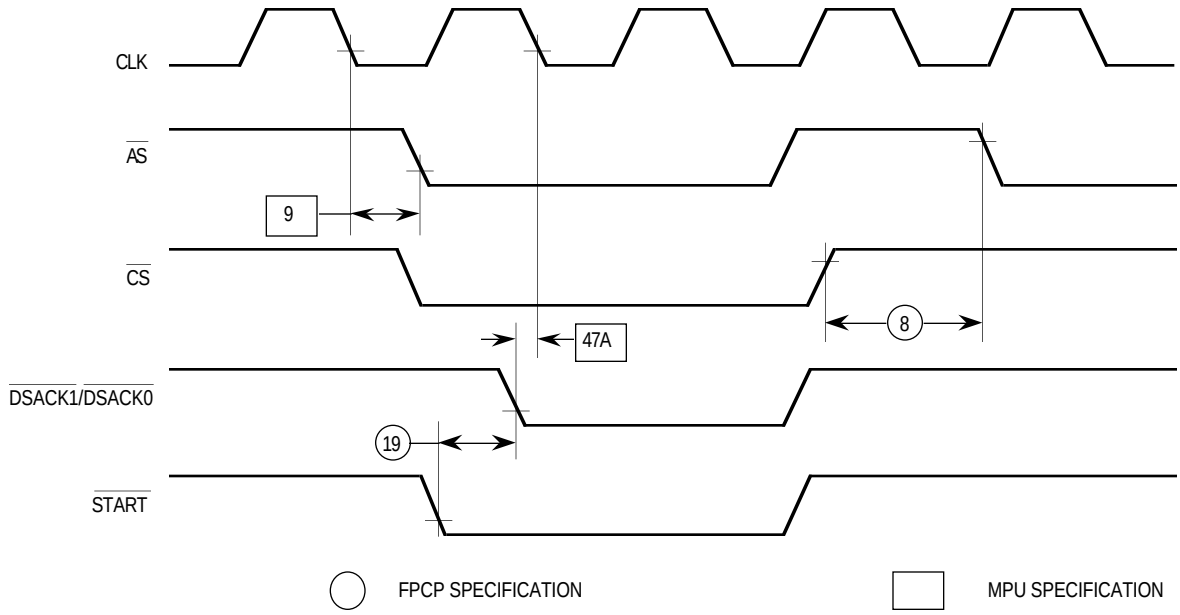
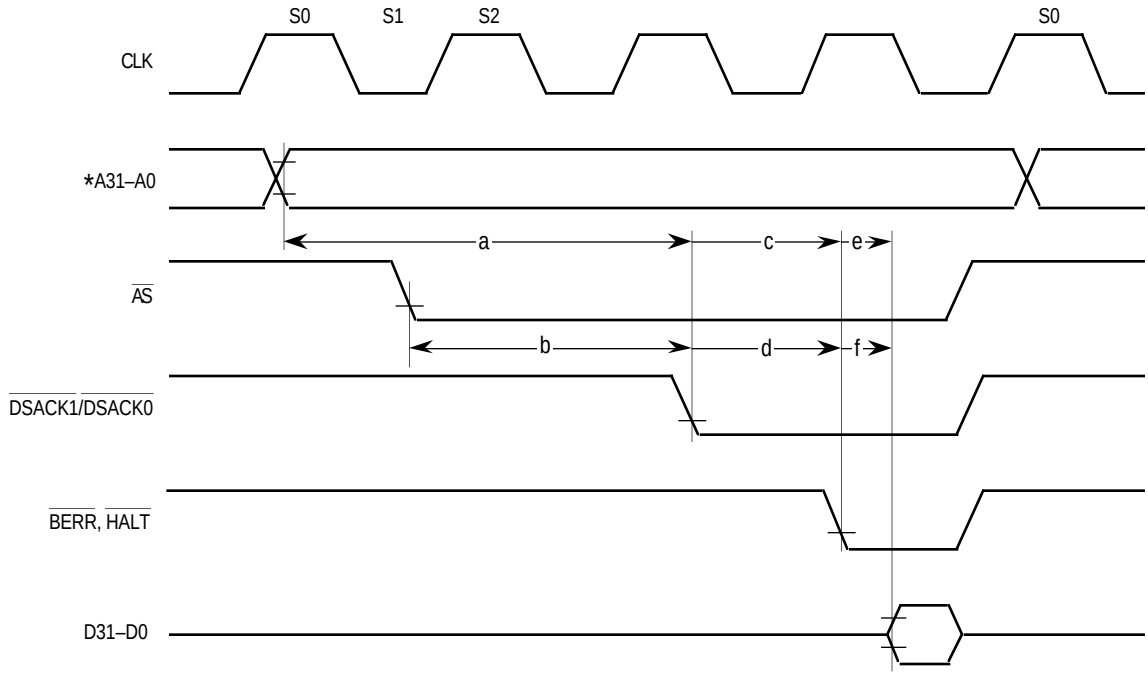


Figure 9-4. Bus Cycle Timing Diagram

9.6 ACCESS TIME CALCULATIONS

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

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



*For the MC68EC020, A23–A0.

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

Table 9-4. Memory Access Time Equations at 16.67 and 25 MHz

Equation	16.667 MHz	N = 3	N = 4	N = 5	N = 6	N = 7	Unit
9-3	t_{AVDL} = (N – 1) • t1 – t2 – t6 – t47A	61	121	181	241	301	ns
9-4	t_{SADL} = (N – 1) • t1 – t9 – t60	25	85	145	205	265	ns
9-5	t_{AVBHL} = N • t1 – t2 – t6 – t27A	22	46	70	94	118	ns
9-6	t_{SABHL} = (N – 1) • t1 – t9 – t27A	40	70	100	130	160	ns
9-7	t_{AVDV} = N • t1 – t2 – t6 – t27	121	181	241	301	361	ns
9-8	t_{SADV} = (N – 1) • t1 – t9 – t27	85	145	205	265	325	ns

Equation	25 MHz	N = 3	N = 4	N = 5	N = 6	N = 7	Unit
9-3	t_{AVDL} = (N – 1) • t1 – t2 – t6 – t47A	31	71	111	151	191	ns
9-4	t_{SADL} = (N – 1) • t1 – t9 – t60	17	57	97	137	177	ns
9-5	t_{AVBHL} = N • t1 – t2 – t6 – t27A	22	41	60	79	98	ns
9-6	t_{SABHL} = (N – 1) • t1 – t9 – t27A	26	44	62	80	98	ns
9-7	t_{AVDV} = N • t1 – t2 – t6 – t27	71	111	151	191	231	ns
9-8	t_{SADV} = (N – 1) • t1 – t9 – t27	57	97	137	177	217	ns

Where:

tX = Refers to AC Electrical Specification X
t1 = The Clock Period
t2 = The Clock Low Time
t3 = The Clock High Time
t6 = The Clock High to Address Valid Time
t9 = The Clock Low to AS Low Delay
t27 = The Data-In to Clock Low Setup Time
t27A = The BERR/HALT to Clock Low Setup Time
t47A = The Asynchronous Input Setup Time
N = The Total Number of Clock Periods in the Bus Cycle (N ≥ 3 Cycles)

During asynchronous bus cycles, DSACK1/DSACK0 are used to terminate the current bus cycle. In true asynchronous operations, such as accesses to peripherals operating at a different clock frequency, either or both signals may be asserted without regard to the clock, and then data must be valid a certain amount of time later as defined by specification 31. With a 25-MHz controller, this time is 32 ns after DSACK1/DSACK0 asserts; with a 16.67-MHz controller, this time is 50 ns after DSACK1/DSACK0 asserts (both numbers vary with the actual clock frequency).

However, many local memory systems do not operate in a truly asynchronous manner because either the memory control logic can be related to the MC68020/EC020 clock or worst-case propagation delays are known; thus, asynchronous setup times for the DSACK1/DSACK0 signals can be guaranteed. The timing requirements for this pseudo-synchronous DSACK1/DSACK0 generation is governed by the equation for t_{AVDL}.

MC68EC020 DC Electrical Characteristics

(V_{CC} = 5.0 V_{dc} ± 5%; GND = 0 V_{dc}; Temperature within defined ranges)

Characteristics		Symbol	Min	Max	Unit
Input High Voltage		V_{IH}	2.0	V_{CC}	V
Input Low Voltage		V_{IL}	GND –0.5	0.8	V
Input Leakage Current GND ≤ V_{in} ≤ V_{CC}	BERR, BR, CLK, IPL2–IPL0, AVEC,DSACK1, DSACK0, CDIS, HALT, RESET	I_{in}	 –1.0 –20	 1.0 20	µA
Hi-Z (Off-State) Leakage Current @ 2.4 V/0.5 V	A23–A0, AS, DS, D31–D0, FC2–FC0, R/W, RMC, SIZ1–SIZ0	I_{TSI}	 –20	 20	µA
Output High Voltage I_{OH} = 400 µA	A23–A0, AS, BG, D31–D0, DS, R/W, RMC, SIZ1–SIZ0, FC2–FC0	V_{OH}	 2.4	 —	V
Output Low Voltage I_{OL} = 3.2 mA I_{OL} = 5.3 mA I_{OL} = 10.7 mA	 A23–A0, FC2–FC0, SIZ1–SIZ0, BG, D31–D0 AS, DS, R/W, RMC, HALT, RESET	V_{OL}	 — — —	 0.5 0.5 0.5	V
Power Dissipation (T_A = 0°C)	f = 25 MHz f = 16 MHz	P_{INT}	— —	1.5 1.2	W
Capacitance (see Note) V_{in} = 0 V, T_A = 25°C, f = 1 MHz		C_{in}	—	20	pF
Load Capacitance		C_L	—	130	pF

NOTE: Capacitance is periodically sampled rather than 100% tested.

10.3 AC ELECTRICAL CHARACTERISTICS

The AC specifications presented consist of output delays, input setup and hold times, and signal skew times. All signals are specified relative to an appropriate edge of the clock and possibly to one or more other signals.

The measurement of the AC specifications is defined by the waveforms shown in Figure 10-1. To test the parameters guaranteed by Motorola, inputs must be driven to the voltage levels specified in Figure 10-1. Outputs are specified with minimum and/or maximum limits, as appropriate, and are measured as shown in Figure 10-1. Inputs are specified with minimum setup and hold times, and are measured as shown. Finally, the measurement for signal-to-signal specifications is also shown.

Note that the testing levels used to verify conformance to the AC specifications do not affect the guaranteed DC operation of the device as specified in the DC electrical specifications. The 20 MHz and 33.33 MHz specifications do not apply to the MC68EC020.

AC ELECTRICAL CHARACTERISTICS—READ AND WRITE CYCLES

(V_{CC} = 5.0 V_{dc} ± 5%; GND = 0 V_{dc}; Temperature within defined ranges; see Figures 10-3–10-5)

Num.	Characteristics	16.67 MHz		20 MHz		25 MHz**		33.33 MHz		Unit
		Min	Max	Min	Max	Min	Max	Min	Max	
6	Clock High to FC, Size, RMC, Address Valid	0	30	0	25	0	25	0	21	ns
*6A	Clock High to ECS, OCS Asserted	0	20	0	15	0	12	0	10	ns
7	Clock High to Address, Data, FC, Size, RMC High Impedance	0	60	0	50	0	40	0	30	ns
8	Clock High to Address, FC, Size, RMC Invalid	0	—	0	—	0	—	0	—	ns
9	Clock Low to AS, DS Asserted	3	30	3	25	3	18	3	15	ns
9A[1]	AS to DS Assertion Skew (Read)	–15	15	–10	10	–10	10	–10	10	ns
9B[11]	AS Asserted to DS Asserted (Write)	37	—	32	—	27	—	22	—	ns
*10	ECS Width Asserted	20	—	15	—	15	—	10	—	ns
*10A	OCS Width Asserted	20	—	15	—	15	—	10	—	ns
*10B[7]	ECS, OCS Width Negated	15	—	10	—	5	—	5	—	ns
11	Address, FC, Size, RMC Valid to AS (and DS Asserted, Read)	15	—	10	—	6	—	5	—	ns
12	Clock Low to AS, DS Negated	0	30	0	25	0	15	0	15	ns
*12A	Clock Low to ECS, OCS Negated	0	30	0	25	0	15	0	15	ns
13	AS, DS Negated to Address, FC, Size, RMC Invalid	15	—	10	—	10	—	5	—	ns
14	AS (and DS Read) Width Asserted	100	—	85	—	70	—	50	—	ns
14A	DS Width Asserted (Write)	40	—	38	—	30	—	25	—	ns
15	AS, DS Width Negated	40	—	38	—	30	—	23	—	ns
15A[8]	DS Negated to AS Asserted	35	—	30	—	25	—	18	—	ns
16	Clock High to AS, DS, R/W, DBEN High Impedance	—	60	—	50	—	40	—	30	ns
17	AS, DS Negated to R/W Invalid	15	—	10	—	10	—	5	—	ns
18	Clock High to R/W High	0	30	0	25	0	20	0	15	ns
20	Clock High to R/W Low	0	30	0	25	0	20	0	15	ns
21	R/W High to AS Asserted (Read)	15	—	10	—	5	—	5	—	ns
22	R/W Low to DS Asserted (Write)	75	—	60	—	50	—	35	—	ns
23	Clock High to Data-Out Valid	—	30	—	25	—	25	—	18	ns
25	AS, DS Negated to Data-Out Invalid	15	—	10	—	5	—	5	—	ns
*25A[9]	DS Negated to DBEN Negated (Write)	15	—	10	—	5	—	5	—	ns
26	Data-Out Valid to DS Asserted (Write)	15	—	10	—	5	—	5	—	ns
27	Data-In Valid to Clock Low (Setup) (Read)	5	—	5	—	5	—	5	—	ns
27A	Late BERR/HALT Asserted to Clock Low (Setup)	20	—	15	—	10	—	5	—	ns
28	AS, DS Negated to DSACK≈, BERR, HALT, AVEC Negated	0	80	0	65	0	50	0	40	ns

AC ELECTRICAL CHARACTERISTICS—READ AND WRITE CYCLES
(Concluded)

NOTES:

1. This number can be reduced to 5 ns if strobes have equal loads.
2. If the asynchronous setup time (#47A) requirements are satisfied, the DSACK≈ low to data setup time (#31) and DSACK≈ low to BERR low setup time (#48) can be ignored. The data must only satisfy the data-in clock low setup time (#27) for the following clock cycle, and BERR must only satisfy the late BERR low to clock low setup time (#27A) for the following clock cycle.
3. This parameter specifies the maximum allowable skew between DSACK0 to DSACK1 asserted or DSACK1 to DSACK0 asserted; specification #47A must be met by DSACK0 or DSACK1.
4. This specification applies to the first (DSACK0 or DSACK1) DSACK≈ signal asserted. In the absence of DSACK≈, BERR is an asynchronous input using the asynchronous input setup time (#47A).
5. DBEN may stay asserted on consecutive write cycles.
6. The minimum values must be met to guarantee proper operation. If this maximum value is exceeded, BG may be reasserted.
7. This specification indicates the minimum high time for ECS and OCS in the event of an internal cache hit followed immediately by a cache miss or operand cycle.
8. This specification guarantees operation with the MC68881/MC68882, which specifies a minimum time for DS negated to AS asserted (specification #13A in MC68881UM/AD, MC68881/MC68882 Floating-Point Coprocessor User's Manual). Without this specification, incorrect interpretation of specifications #9A and #15 would indicate that the MC68020/EC020 does not meet the MC68881/MC68882 requirements.
9. This specification allows a system designer to guarantee data hold times on the output side of data buffers that have output enable signals generated with DBEN.
10. These specifications allow system designers to guarantee that an alternate bus master has stopped driving the bus when the MC68020/EC020 regains control of the bus after an arbitration sequence.
11. This specification allows system designers to qualify the CS signal of an MC68881/MC68882 with AS (allowing 7 ns for a gate delay) and still meet the CS to DS setup time requirement (specification 8B of MC68881UM/AD, MC68881/MC68882 Floating-Point Coprocessor User's Manual).