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	132-BQFP Bumpered
Supplier Device Package	132-PQFP (46x46)
Purchase URL	https://www.e-xfl.com/product-detail/nxp-semiconductors/mc68020eh25e

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
3.12	Power Supply Connections	3-7
3.13	Signal Summary	3-8

Section 4
On-Chip Cache Memory

Paragraph Number	Title	Page Number
4.1	On-Chip Cache Organization and Operation	4-1
4.2	Cache Reset	4-3
4.3	Cache Control	4-3
4.3.1	Cache Control Register (CACR)	4-3
4.3.2	Cache Address Register (CAAR)	4-4

Section 5
Bus Operation

Paragraph Number	Title	Page Number
5.1	Bus Transfer Signals	5-1
5.1.1	Bus Control Signals	5-2
5.1.2	Address Bus	5-3
5.1.3	Address Strobe	5-3
5.1.4	Data Bus	5-3
5.1.5	Data Strobe	5-4
5.1.6	Data Buffer Enable	5-4
5.1.7	Bus Cycle Termination Signals	5-4
5.2	Data Transfer Mechanism	5-5
5.2.1	Dynamic Bus Sizing	5-5
5.2.2	Misaligned Operands	5-14
5.2.3	Effects of Dynamic Bus Sizing and Operand Misalignment	5-20
5.2.4	Address, Size, and Data Bus Relationships	5-21
5.2.5	Cache Interactions	5-22
5.2.6	Bus Operation	5-24
5.2.7	Synchronous Operation with DSACK1/DSACK0	5-24
5.3	Data Transfer Cycles	5-25
5.3.1	Read Cycle	5-26
5.3.2	Write Cycle	5-33
5.3.3	Read-Modify-Write Cycle	5-39
5.4	CPU Space Cycles	5-44
5.4.1	Interrupt Acknowledge Bus Cycles	5-45
5.4.1.1	Interrupt Acknowledge Cycle—Terminated Normally	5-45
5.4.1.2	Autovector Interrupt Acknowledge Cycle	5-48
5.4.1.3	Spurious Interrupt Cycle	5-48
5.4.2	Breakpoint Acknowledge Cycle	5-50
5.4.3	Coprocessor Communication Cycles	5-53
5.5	Bus Exception Control Cycles	5-53
5.5.1	Bus Errors	5-55

TABLE OF CONTENTS (Concluded)

Paragraph Number	Title	Page Number
9.4	Clock Driver	9-10
9.5	Memory Interface	9-11
9.6	Access Time Calculations	9-12
9.7	Module Support	9-14
9.7.1	Module Descriptor	9-14
9.7.2	Module Stack Frame	9-16
9.8	Access Levels	9-17
9.8.1	Module Call	9-18
9.8.2	Module Return	9-19

Section 10
Electrical Characteristics

10.1	Maximum Ratings	10-1
10.2	Thermal Considerations	10-1
10.2.1	MC68020 Thermal Characteristics and DC Electrical Characteristics	10-2
10.2.2	MC68EC020 Thermal Characteristics and DC Electrical Characteristics	10-4
10.3	AC Electrical Characteristics	10-5

Section 11
Ordering Information and Mechanical Data

11.1	Standard Ordering Information	11-1
11.1.1	Standard MC68020 Ordering Information	11-1
11.1.2	Standard MC68EC020 Ordering Information	11-1
11.2	Pin Assignments and Package Dimensions	11-2
11.2.1	MC68020 RC and RP Suffix—Pin Assignment	11-2
11.2.2	MC68020 RC Suffix—Package Dimensions	11-3
11.2.3	MC68020 RP Suffix—Package Dimensions	11-4
11.2.4	MC68020 FC and FE Suffix—Pin Assignment	11-5
11.2.5	MC68020 FC Suffix—Package Dimensions	11-6
11.2.6	MC68020 FE Suffix—Package Dimensions	11-7
11.2.7	MC68EC020 RP Suffix—Pin Assignment	11-8
11.2.8	MC68EC020 RP Suffix—Package Dimensions	11-9
11.2.9	MC68EC020 FG Suffix—Pin Assignment	11-10
11.2.10	MC68EC020 FG Suffix—Package Dimensions	11-11

Appendix A
Interfacing an MC68EC020 to a DMA Device That Supports a Three-Wire Bus Arbitration Protocol

SECTION 4
ON-CHIP CACHE MEMORY

The MC68020/EC020 incorporates an on-chip cache memory as a means of improving performance. The cache is implemented as a CPU instruction cache and is used to store the instruction stream prefetch accesses from the main memory.

An increase in instruction throughput results when instruction words required by a program are available in the on-chip cache and the time required to access them on the external bus is eliminated. In systems with more than one bus master (e.g., a processor and a DMA device), reduced external bus activity increases overall performance by increasing the availability of the bus for use by external devices without degrading the performance of the MC68020/EC020.

4.1 ON-CHIP CACHE ORGANIZATION AND OPERATION

The MC68020/EC020 on-chip instruction cache is a direct-mapped cache of 64 long-word entries. Each cache entry consists of a tag field (A31–A8 and FC2), one valid bit, and 32 bits (two words) of instruction data. Figure 4-1 shows a block diagram of the on-chip cache organization.

Externally, the MC68EC020 does not use the upper eight bits of the address (A31–A24), and addresses $FF000000 and $00000000 from the MC68EC020 appear the same. However, the MC68EC020 does use A31–A24 internally in the instruction cache address tag, and addresses $FF000000 and $00000000 appear different in the MC68EC020 instruction cache. The MC68020, MC68030/EC030, and MC68040/EC040 use all 32 bits of the address externally. To maintain object-code upgrade compatibility when designing with the MC68EC020, the upper eight bits should be considered part of the address when assigning address spaces in hardware.

When enabled, the MC68020/EC020 instruction cache is used to store instruction prefetches (instruction words and extension words) as they are requested by the CPU. Instruction prefetches are normally requested from sequential memory addresses except when a change of program flow occurs (e.g., a branch taken) or when an instruction is executed that can modify the SR. In these cases, the instruction pipe is automatically flushed and refilled.

F—Freeze Cache

The F-bit is set to freeze the instruction cache. When the F-bit is set and a cache miss occurs, the entry (or line) is not replaced. When the F-bit is clear, a cache miss causes the entry (or line) to be filled. A reset operation clears the F-bit.

E—Enable Cache

The E-bit is set to enable the instruction cache. When it is clear, the instruction cache is disabled. A reset operation clears the E-bit. The supervisor normally enables the instruction cache, but it can clear the E-bit for system debugging or emulation, as required. Disabling the instruction cache does not flush the entries. If the cache is reenabled, the previously valid entries remain valid and may be used.

4.3.2 Cache Address Register (CAAR)

The format of the 32-bit CAAR is shown in Figure 4-3.

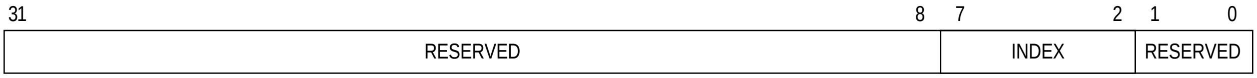


Figure 4-3. Cache Address Register

Bits 31–8, 1, and 0—Reserved

These bits are reserved for use by Motorola.

Index Field

The index field contains the address for the “clear cache entry” operations. The bits of this field, which correspond to A7–A2, specify the index and a long word of a cache line.

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

Table 5-4. Data Bus Requirements for Read Cycles

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

State 0

MC68020—The write cycle starts in S0. The processor negates ECS, indicating the beginning of an external cycle. If the cycle is the first external cycle of a write operation, OCS is asserted simultaneously. During S0, the processor places a valid address on A31–A0 and valid function codes on FC2–FC0. The function codes select the address space for the cycle. The processor drives R/W low for a write cycle. SIZ1–SIZ0 become valid, indicating the number of bytes to be transferred.

MC68EC020—The write cycle starts in S0. During S0, the processor places a valid address on A23–A0 and valid function codes on FC2–FC0. The function codes select the address space for the cycle. The processor drives R/W low for a write cycle. SIZ1, SIZ0 become valid, indicating the number of bytes to be transferred.

State 1

MC68020—One-half clock later in S1, the processor asserts AS, indicating that the address on the address bus is valid. The processor also asserts DBEN during S1, which can enable external data buffers. In addition, the ECS (and OCS, if asserted) signal is negated during S1.

MC68EC020—One-half clock later in S1, the processor asserts AS, indicating that the address on the address bus is valid.

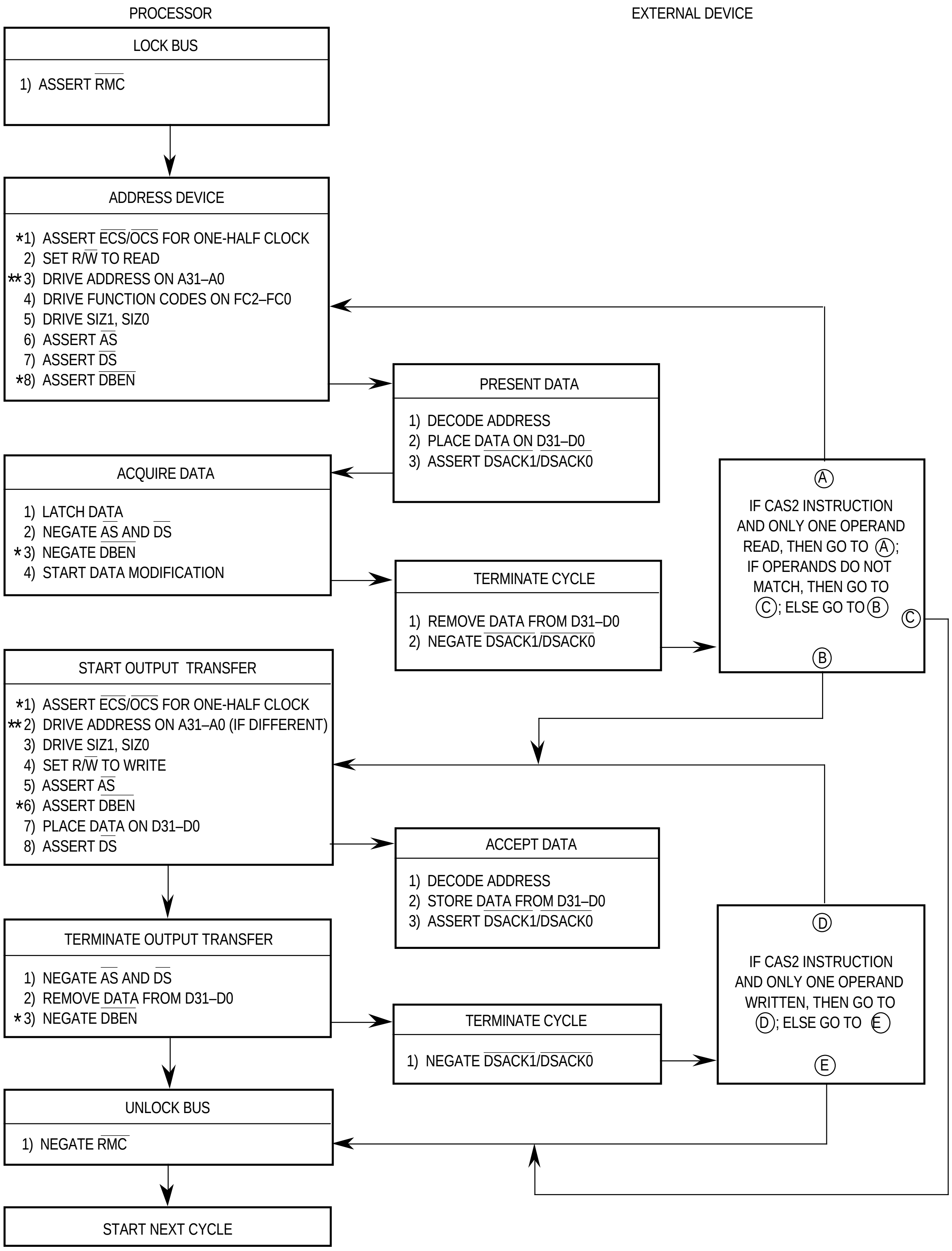
State 2

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

State 3

MC68020/EC020—The processor asserts DS during S3, indicating that the data on the data bus is stable. As long as at least one of the DSACK1/DSACK0 signals is recognized by the end of S2 (meeting the asynchronous input setup time requirement), the cycle terminates one clock later. If DSACK1/DSACK0 is not recognized by the start of S3, the processor inserts wait states instead of proceeding to S4 and S5. To ensure that wait states are inserted, both DSACK1 and DSACK0 must remain negated throughout the asynchronous input setup and hold times around the end of S2. If wait states are added, the processor continues to sample the DSACK1/DSACK0 signals on the falling edges of the clock until one is recognized.

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



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

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

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.

level 6 interrupt is not processed. However, if the MC68020/EC020 is handling a level 7 interrupt (I2–I0 in the SR set to 111) and the external request is lowered to level 3 and then raised back to level 7, a second level 7 interrupt is processed. The second level 7 interrupt is processed because the level 7 interrupt is transition sensitive. A level 7 interrupt is also generated by a level comparison if the request level and mask level are at 7 and the priority mask is then set to a lower level (with the MOVE to SR or RTE instruction, for example). As shown in Figure 6-3 for level 6 interrupt request level and mask level, this is the case for all interrupt levels.

Note that a mask value of 6 and a mask value of 7 both inhibit request levels of 1–6 from being recognized. In addition, neither masks a transition to an interrupt request level of 7. The only difference between mask values of 6 and 7 occurs when the interrupt request level is 7 and the mask value is 7. If the mask value is lowered to 6, a second level 7 interrupt is recognized.

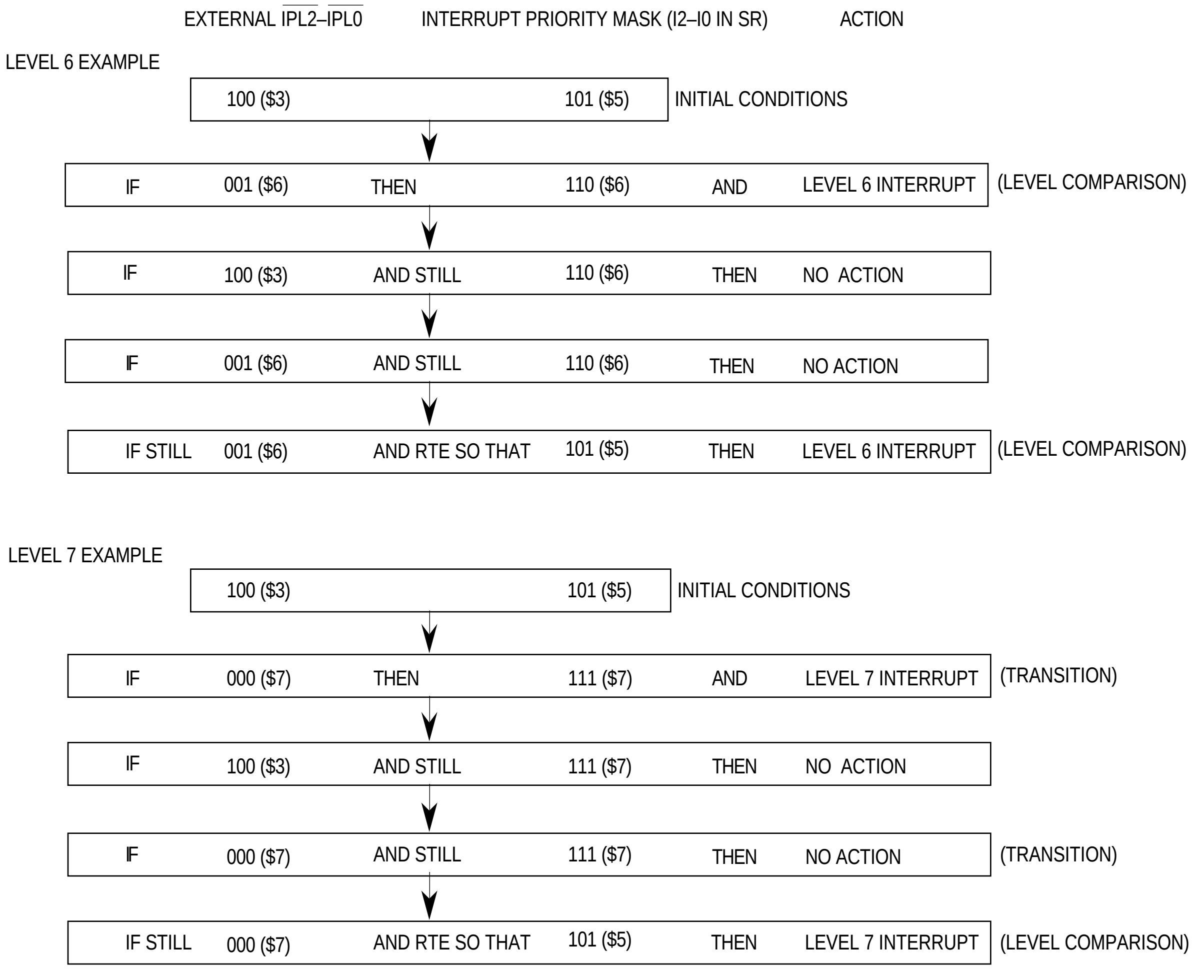

EXTERNAL $\overline{IPL2}$–$\overline{IPL0}$
INTERRUPT PRIORITY MASK (I2–I0 IN SR)
ACTION
LEVEL 6 EXAMPLE
100 ($3) 101 ($5) INITIAL CONDITIONS
IF 001 ($6) THEN 110 ($6) AND LEVEL 6 INTERRUPT (LEVEL COMPARISON)
IF 100 ($3) AND STILL 110 ($6) THEN NO ACTION
IF 001 ($6) AND STILL 110 ($6) THEN NO ACTION
IF STILL 001 ($6) AND RTE SO THAT 101 ($5) THEN LEVEL 6 INTERRUPT (LEVEL COMPARISON)
LEVEL 7 EXAMPLE
100 ($3) 101 ($5) INITIAL CONDITIONS
IF 000 ($7) THEN 111 ($7) AND LEVEL 7 INTERRUPT (TRANSITION)
IF 100 ($3) AND STILL 111 ($7) THEN NO ACTION
IF 000 ($7) AND STILL 111 ($7) THEN NO ACTION (TRANSITION)
IF STILL 000 ($7) AND RTE SO THAT 101 ($5) THEN LEVEL 7 INTERRUPT (LEVEL COMPARISON)

Figure 6-3. Interrupt Recognition Examples

SECTION 7
COPROCESSOR INTERFACE DESCRIPTION

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

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

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

7.1 INTRODUCTION

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

restore categories, the coprocessor uses the set of coprocessor format codes defined for the M68000 coprocessor interface to indicate its status to the main processor.

7.2.1 Coprocessor General Instructions

The coprocessor general instruction category contains data processing instructions and other general-purpose instructions for a given coprocessor.

7.2.1.1 FORMAT. Figure 7-6 shows the format of a coprocessor general instruction.

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11 … 9</th><th>8</th><th>7</th><th>6</th><th>5 … 0</th></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>CpID</td><td>0</td><td>0</td><td>0</td><td>EFFECTIVE ADDRESS</td></tr>
<tr><td colspan="9">COPROCESSOR COMMAND</td></tr>
<tr><td colspan="9">OPTIONAL EFFECTIVE ADDRESS OR COPROCESSOR-DEFINED EXTENSION WORDS</td></tr>
</table>

Figure 7-6. Coprocessor General Instruction Format (cpGEN)

The mnemonic cpGEN is a generic mnemonic used in this discussion for all general instructions. The mnemonic of a specific general instruction usually suggests the type of operation it performs and the coprocessor to which it applies. The actual mnemonic and syntax used to represent a coprocessor instruction is determined by the syntax of the assembler or compiler that generates the object code.

A coprocessor general instruction consists of at least two words. The first word of the instruction is an F-line operation code (bits 15–12 = 1111). The CpID field of the F-line operation code is used during the coprocessor access to indicate which coprocessor in the system executes the instruction. During accesses to the CIRs (refer to **7.1.4.2 Processor-Coprocessor Interface**), the processor places the CpID on address lines A15–A13.

Bits 8–6 = 000 of the first word of an instruction indicate that the instruction is in the general instruction category. Bits 5–0 of the F-line operation code sometimes encode a standard M68000 effective address specifier (refer to M68000PM/AD, *M68000 Family Programmer's Reference Manual*). During the execution of a cpGEN instruction, the coprocessor can use a coprocessor response primitive to request that the MC68020/EC020 perform an effective address calculation necessary for that instruction. Using the effective address specifier field of the F-line operation code, the processor then determines the effective addressing mode. If a coprocessor never requests effective address calculation, bits 5–0 can have any value (don't cares).

The second word of the general type instruction is the coprocessor command word. The main processor writes this command word to the command CIR to initiate execution of the instruction by the coprocessor.

An instruction in the coprocessor general instruction category optionally includes a number of extension words following the coprocessor command word. These words can provide additional information required for the coprocessor instruction. For example, if

The write to previously evaluated effective address primitive uses the CA and PC bits as described in **7.4.2 Coprocessor Response Primitive General Format**.

The length field of the primitive format specifies the length of the operand in bytes. The MC68020/EC020 transfers operands of 0–255 bytes in length.

When the main processor receives this primitive during the execution of a general category instruction, it transfers an operand from the operand CIR to an effective address specified by a temporary register within the MC68020/EC020. When a previous primitive for the current instruction has evaluated the effective address, this temporary register contains the evaluated effective address. Primitives that store an evaluated effective address in a temporary register of the main processor are the evaluate and transfer effective address, evaluate effective address and transfer data, and transfer multiple coprocessor registers primitive. If this primitive is used during an instruction in which the effective address specified in the instruction operation word has not been calculated, the effective address used for the write is undefined. Also, if the previously evaluated effective address was register direct, the address written to in response to this primitive is undefined.

The function code value during the write operation indicates either supervisor or user data space, depending on the value of the S-bit in the MC68020/EC020 SR when the processor reads this primitive. While a coprocessor should request writes to only alterable effective addressing modes, the MC68020/EC020 does not check the type of effective address used with this primitive. For example, if the previously evaluated effective address was PC relative and the MC68020/EC020 is at the user privilege level (S = 0 in SR), the MC68020/EC020 writes to user data space at the previously calculated program relative address (the 32-bit value in the temporary internal register of the processor).

Operands longer than four bytes are transferred in increments of four bytes (operand parts) when possible. The main processor reads a long-word operand part from the operand CIR and transfers this part to the current effective address. The transfers continue in this manner using ascending memory locations until all of the long-word operand parts are transferred, and any remaining operand part is then transferred using a one-, two-, or three-byte transfer as required. The operand parts are stored in memory using ascending addresses beginning with the address in the MC68020/EC020 temporary register, which is internal to the processor and not for user use.

The execution of this primitive does not modify any of the registers in the MC68020/EC020 programming model, even if the previously evaluated effective address mode is the predecrement or postincrement mode. If the previously evaluated effective addressing mode used any of the MC68020/EC020 internal address or data registers, the effective address value used is the final value from the preceding primitive. That is, this primitive uses the value from an evaluate and transfer effective address, evaluate effective address and transfer data, or transfer multiple coprocessor registers primitive without modification.

7.5.1.1 COPROCESSOR-DETECTED PROTOCOL VIOLATIONS. Protocol violation exceptions are communication failures between the main processor and coprocessor across the M68000 coprocessor interface. Coprocessor-detected protocol violations occur when the main processor accesses entries in the CIR set in an unexpected sequence. The sequence of operations that the main processor performs for a given coprocessor instruction or coprocessor response primitive has been described previously in this section.

A coprocessor can detect protocol violations in various ways. According to the M68000 coprocessor interface protocol, the main processor always accesses the operation word, operand, register select, instruction address, or operand address CIRs synchronously with respect to the operation of the coprocessor. That is, the main processor accesses these five registers in a certain sequence, and the coprocessor expects them to be accessed in that sequence. As a minimum, all M68000 coprocessors should detect a protocol violation if the main processor accesses any of these five registers when the coprocessor is expecting an access to either the command or condition CIR. Likewise, if the coprocessor is expecting an access to the command or condition CIR and the main processor accesses one of these five registers, the coprocessor should detect and signal a protocol violation.

According to the M68000 coprocessor interface protocol, the main processor can perform a read of either the save CIR or response CIR or a write of either the restore CIR or control CIR asynchronously with respect to the operation of the coprocessor. That is, an access to one of these registers without the coprocessor explicitly expecting that access at that point can be a valid access. Although the coprocessor can anticipate certain accesses to the restore, response, and control CIRs, these registers can be accessed at other times also.

The coprocessor cannot signal a protocol violation to the main processor during execution of a cpSAVE or cpRESTORE instruction. If a coprocessor detects a protocol violation during execution of the cpSAVE or cpRESTORE instruction, it should signal the exception to the main processor when the next coprocessor instruction is initiated.

The main philosophy of the coprocessor-detected protocol violation is that the coprocessor should always acknowledge an access to one of its interface registers. If the coprocessor determines that the access is not valid, it should assert DSACK1/DSACK0 to the main processor and signal a protocol violation when the main processor next reads the response CIR. If the coprocessor fails to assert DSACK1/DSACK0, the main processor waits for the assertion of that signal (or some other bus termination signal) indefinitely. The protocol previously described ensures that the coprocessor cannot halt the main processor.

The coprocessor can signal a protocol violation to the main processor with the take midinstruction exception primitive. To maintain consistency, the vector number should be 13, as it is for a protocol violation detected by the main processor. When the main processor reads this primitive, it proceeds as described in **7.4.19 Take Midinstruction Exception Primitive**. If the exception handler does not modify the stack frame, the MC68020/EC020 returns from the exception handler and reads the response CIR.

8.2.3 Calculate Effective Address

The calculate immediate effective address table indicates the number of clock periods needed for the processor to calculate the specified effective address. Fetch time is only included for the first level of indirection on memory indirect addressing modes. 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.

Address Mode	Best Case	Cache Case	Worst Case
Dn	**0**(0/0/0)	**0**(0/0/0)	**0**(0/0/0)
An	**0**(0/0/0)	**0**(0/0/0)	**0**(0/0/0)
(An)	**2**(0/0/0)	**2**(0/0/0)	**2**(0/0/0)
(An)+	**2**(0/0/0)	**2**(0/0/0)	**2**(0/0/0)
–(An)	**2**(0/0/0)	**2**(0/0/0)	**2**(0/0/0)
(d$_{16}$,An) or (d$_{16}$,PC)	**2**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
<data>.W	**2**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
<data>.L	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
(d$_8$,An,Xn) or (d$_8$,PC,Xn)	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
(d$_{16}$,An,Xn) or (d$_{16}$,PC,Xn)	**3**(0/0/0)	**6**(0/0/0)	**7**(0/1/0)
(B)	**3**(0/0/0)	**6**(0/0/0)	**7**(0/1/0)
(d$_{16}$,B)	**5**(0/0/0)	**8**(0/0/0)	**10**(0/1/0)
(d$_{32}$,B)	**9**(0/0/0)	**12**(0/0/0)	**15**(0/2/0)
([B],I)	**8**(1/0/0)	**11**(1/0/0)	**12**(1/1/0)
([B],I,d$_{16}$)	**10**(1/0/0)	**13**(1/0/0)	**15**(1/1/0)
([B],I,d$_{32}$)	**10**(1/0/0)	**13**(1/0/0)	**16**(1/2/0)
([d$_{16}$,B],I)	**10**(1/0/0)	**13**(1/0/0)	**15**(1/1/0)
([d$_{16}$,B],I,d$_{16}$)	**12**(1/0/0)	**15**(1/0/0)	**18**(1/2/0)
([d$_{16}$,B],I,d$_{32}$)	**12**(1/0/0)	**15**(1/0/0)	**19**(1/2/0)
([d$_{32}$,B],I)	**14**(1/0/0)	**17**(1/0/0)	**19**(1/2/0)
([d$_{32}$,B],I,d$_{16}$)	**16**(1/0/0)	**19**(1/0/0)	**21**(1/2/0)
([d$_{32}$,B],I,d$_{32}$)	**16**(1/0/0)	**19**(1/0/0)	**24**(1/3/0)

B = Base address; 0, An, PC, Xn, An + Xn. Form does not affect timing.

I = Index; 0, Xn

NOTE: Xn cannot be in B and I at the same time. Scaling and size of Xn do not affect timing.

8.2.13 Bit Manipulation Instructions

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

Instruction			Best Case	Cache Case	Worst Case
	BTST	#<data>,Dn	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
	BTST	Dn,Dn	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
**	BTST	#<data>,Mem	**4**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
*	BTST	Dn,Mem	**4**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
	BCHG	#<data>,Dn	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
	BCHG	Dn,Dn	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
**	BCHG	#<data>,Mem	**4**(0/0/1)	**4**(0/0/1)	**5**(0/1/1)
*	BCHG	Dn,Mem	**4**(0/0/1)	**4**(0/0/1)	**5**(0/1/1)
	BCLR	#<data>,Dn	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
	BCLR	Dn,Dn	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
**	BCLR	#<data>,Mem	**4**(0/0/1)	**4**(0/0/1)	**5**(0/1/1)
*	BCLR	Dn,Mem	**4**(0/0/1)	**4**(0/0/1)	**5**(0/1/1)
	BSET	#<data>,Dn	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
	BSET	Dn,Dn	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
**	BSET	#<data>,Mem	**4**(0/0/1)	**4**(0/0/1)	**5**(0/1/1)
*	BSET	Dn,Mem	**4**(0/0/1)	**4**(0/0/1)	**5**(0/1/1)

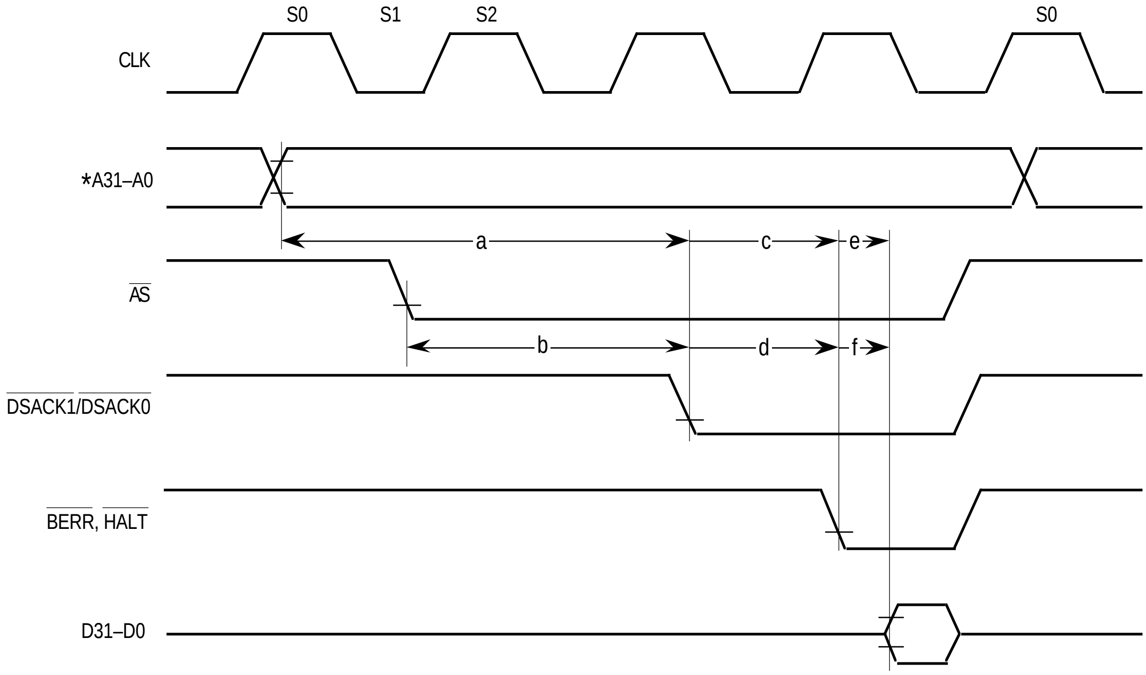
*Add Fetch Effective Address Time

**Add Fetch Immediate Address Time

9.6 ACCESS TIME CALCULATIONS

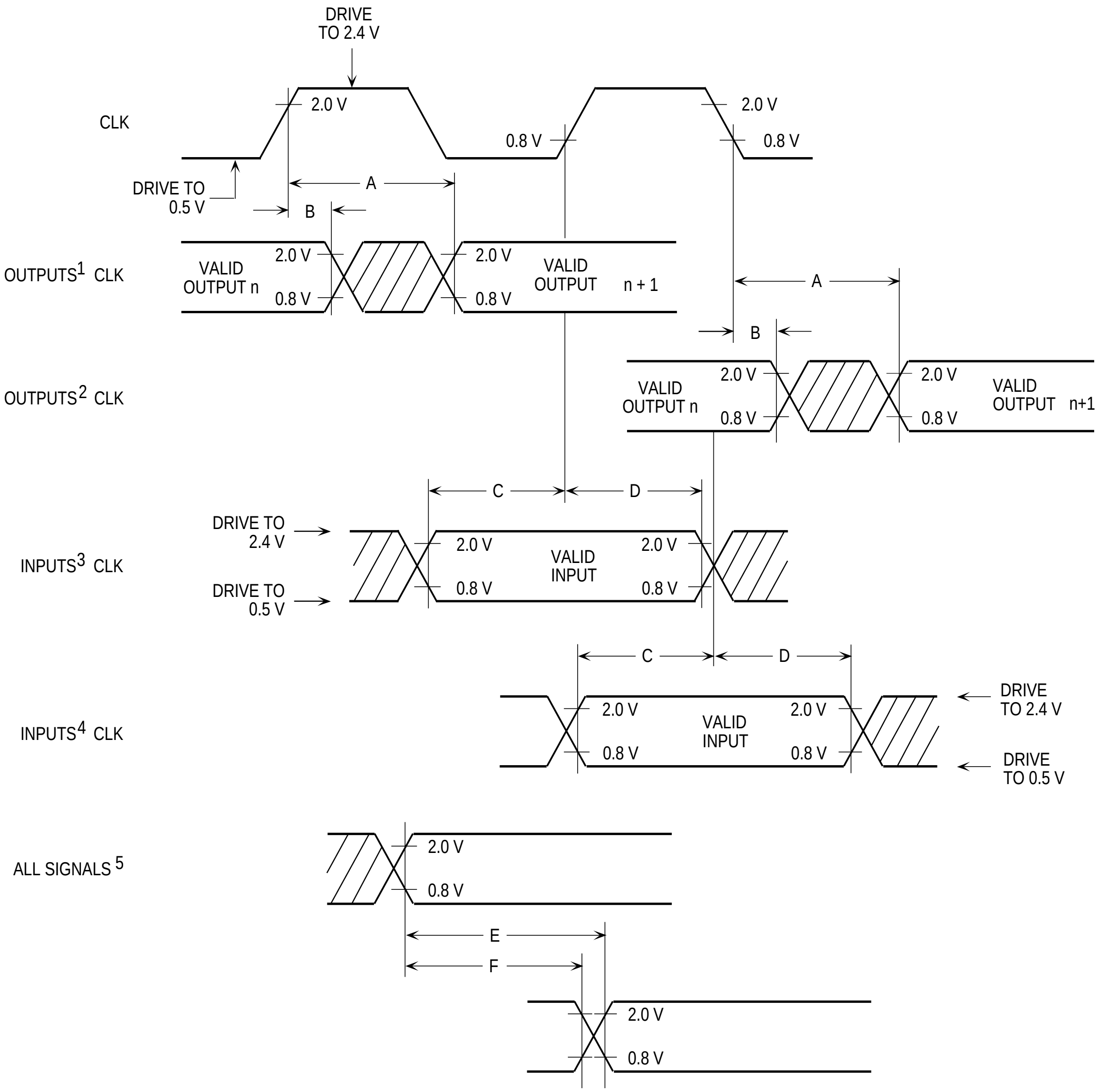
The timing paths that are critical in any memory interface are illustrated and defined in Figure 9-9.

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



NOTES:

1. This output timing is applicable to all parameters specified relative to the rising edge of the clock.
2. This output timing is applicable to all parameters specified relative to the falling edge of the clock.
3. This input timing is applicable to all parameters specified relative to the rising edge of the clock.
4. This input timing is applicable to all parameters specified relative to the falling edge of the clock.
5. This timing is applicable to all parameters specified relative to the assertion/negation of another signal.

LEGEND:

A. Maximum output delay specification.
B. Minimum output hold time.
C. Minimum input setup time specification.
D. Minimum input hold time specification.
E. Signal valid to signal valid specification (maximum or minimum).
F. Signal valid to signal invalid specification (maximum or minimum).

FIGURE 10-1
MC68020UM

Figure 10-1. Drive Levels and Test Points for AC Specifications

SECTION 11
ORDERING INFORMATION AND MECHANICAL DATA

This section contains the pin assignments and package dimensions of the MC68020 and the MC68EC020. In addition, detailed information is provided to be used as a guide when ordering.

11.1 STANDARD ORDERING INFORMATION

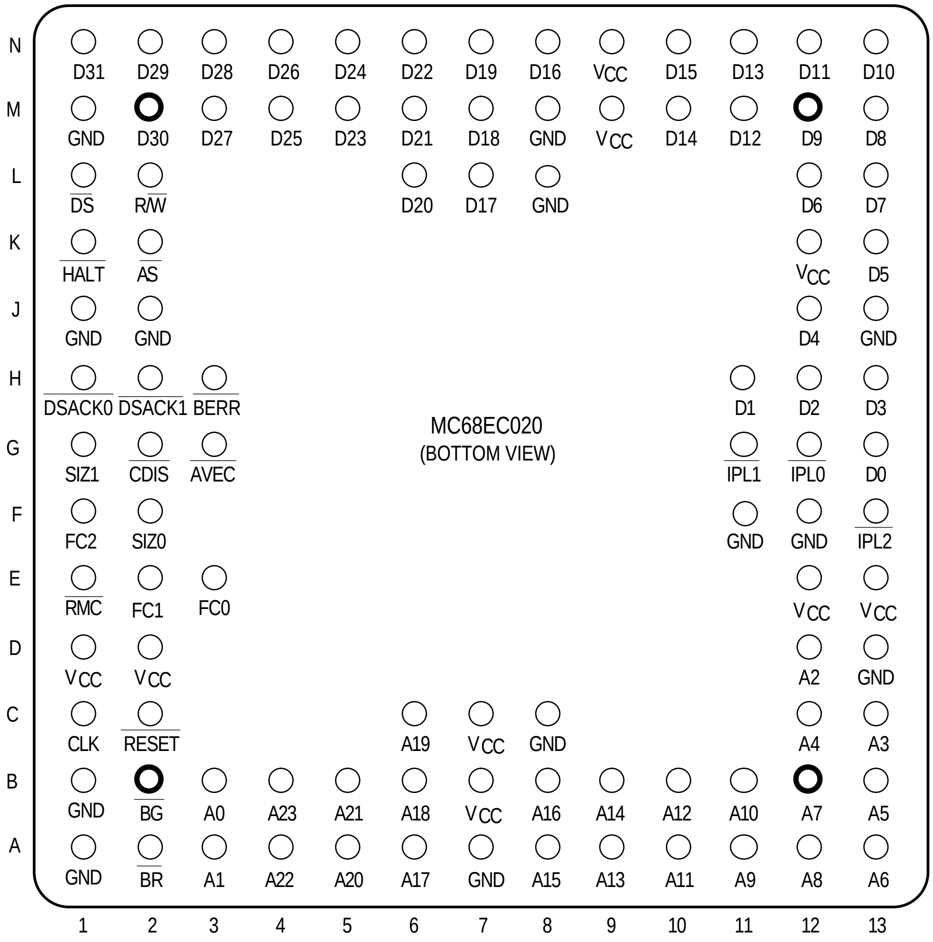
11.1.1 Standard MC68020 Ordering Information

Package Type	Frequency (MHz)	Temperature (°C)	Order Number
Ceramic Pin Grid Array RC Suffix	16.67 20.0 25.0 33.33	0 to 70 0 to 70 0 to 70 0 to 70	MC68020RC16 MC68020RC20 MC68020RC25 MC68020RC33
Plastic Quad Flat Pack FC Suffix	16.67 20.0 25.0	0 to 70 0 to 70 0 to 70	MC68020FC16 MC68020FC20 MC68020FC25
Plastic Pin Grid Array RP Suffix	16.67 20.0 25.0	0 to 70 0 to 70 0 to 70	MC68020RP16 MC68020RP20 MC68020RP25
Ceramic Quad Flat Pack FE Suffix	16.67 20.0 25.0 33.33	0 to 70 0 to 70 0 to 70 0 to 70	MC68020FE16 MC68020FE20 MC68020FE25 MC68020FE33

11.1.2 Standard MC68EC020 Ordering Information

Package Type	Frequency (MHz)	Temperature (°C)	Order Number
Plastic Pin Grid Array RP Suffix	16.67 25.0	0 to 70 0 to 70	MC68EC020RP16 MC68EC020RP25
Plastic Quad Flat Pack FG Suffix	16.67 25.0	0 to 70 0 to 70	MC68EC020FG16 MC68EC020FG25

11.2.7 MC68EC020 RP Suffix—Pin Assignment


N
D31 D29 D28 D26 D24 D22 D19 D16 VCC D15 D13 D11 D10
M
GND D30 D27 D25 D23 D21 D18 GND VCC D14 D12 D9 D8
L
DS R/W D20 D17 GND D6 D7
K
HALT AS VCC D5
J
GND GND D4 GND
H
DSACK0 DSACK1 BERR D1 D2 D3
G
SIZ1 CDIS AVEC IPL1 IPL0 D0
F
FC2 SIZ0 GND GND IPL2
E
RMC FC1 FC0 VCC VCC
D
VCC VCC A2 GND
C
CLK RESET A19 VCC GND A4 A3
B
GND BG A0 A23 A21 A18 VCC A16 A14 A12 A10 A7 A5
A
GND BR A1 A22 A20 A17 GND A15 A13 A11 A9 A8 A6
1 2 3 4 5 6 7 8 9 10 11 12 13
MC68EC020
(BOTTOM VIEW)

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	B7, C7	A1, A7, C8, D13
Data Bus	K12, M9, N9	J13, L8, M1, M8
Logic	D1, D2, E12, E13	F11, F12, J1, J2
Clock	—	B1

INDEX

— A —
A1, A0 Signals, 5-2, 5-7, 5-9, 5-21, 9-5
A15–A13 Signals, 7-6
A19–A16 Signals, 7-6
A31–A24 Signals, 4-1, 5-3
AC Specifications, 10-5
Access Level, 9-17
Access Time Calculations, 9-12
Address Bus, 3-2, 5-3, 5-25
Address Error Exception, 5-14, 6-6
Address Registers, 1-4
Address Space, 2-4, 5-3
Addressing Modes, 1-8
Arithmetic/Logical Instruction, 8-30, 8-31
AS Signal, 3-4, 5-2, 5-3
Autovector, 5-48
Autovector Interrupt Acknowledge Cycle, 5-48
AVEC Signal, 3-5, 5-4, 5-48, 5-53

— B —
BERR Signal, 3-7, 5-4, 5-25, 5-53, 5-55, 6-4
BG Signal, 3-6, 5-63, 5-66, 5-70
BGACK Signal, 3-6, 5-62, 5-63, 5-66
Binary-Coded Decimal, 8-32
Bit Field Manipulation Instructions, 8-36
Bit Manipulation Instructions, 8-35
BKPT Instruction, 5-50
    Flowchart, 6-17
Block Diagram, 1-2
BR Signal, 3-6, 5-63, 5-66, 5-70, 5-71
Breakpoint Acknowledge Cycle, 5-50, 6-17
    Flowchart, 5-50
    Timing, 5-50
Breakpoint Instruction Exception, 6-17
Bus, 5-24
    Arbitration, 5-62
    Cycles, 5-1
    Master, 5-1
    Operation, 5-1, 5-24
Bus Arbitration (MC68020), 5-63
    Control Unit, 5-67
    Flowchart, 5-63
    Read-Modify-Write, 5-68
    Timing, 5-63
Bus Arbitration (MC68EC020), 5-70
    Control Unit, 5-73
    Flowchart, 5-70
    Timing, 5-70
    Two-Wire, 5-75, A-1
Bus Controller, 5-22, 8-2, 8-5
Bus Cycles, 5-1, 5-25
Bus Error Exception, 6-4, 6-21
Bus Fault, 6-21
Bus Master, 5-1, 5-25, 5-62
Bus Operation, 5-24
Byte Enable Signals, 5-21
Byte Select Control Signals, 9-5

— C —
Cache, 1-13, 4-1, 5-2, 5-22, 5-62, 8-1, 8-7, 9-11
    Control, 4-3
    Internal Cache Holding Register, 5-21
    Reset, 4-3
Cache Address Register (CAAR), 1-7, 4-3, 4-4
Cache Control Register (CACR), 1-7, 4-2, 4-3
CALLM Instruction, 9-14, 9-16, 9-18
CAS Instruction, 5-39
CAS2 Instruction, 5-39
CDIS Signal, 3-7, 4-3
CLK Signal, 3-7
Clock Drivers, 9-10
Condition Codes, 1-7
Conditional Branch Instructions, 8-37
Control Instructions, 8-38
Coprocessor, 6-25, 7-1
    Classification, 7-4
    Communication Protocol, 7-4
    Conditional Instruction Category, 7-10
    Coprocessor Context Restore Instruction Category, 7-22