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	166MHz
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/pro/item?MUrl=&PartUrl=mc68020fe16e

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.

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

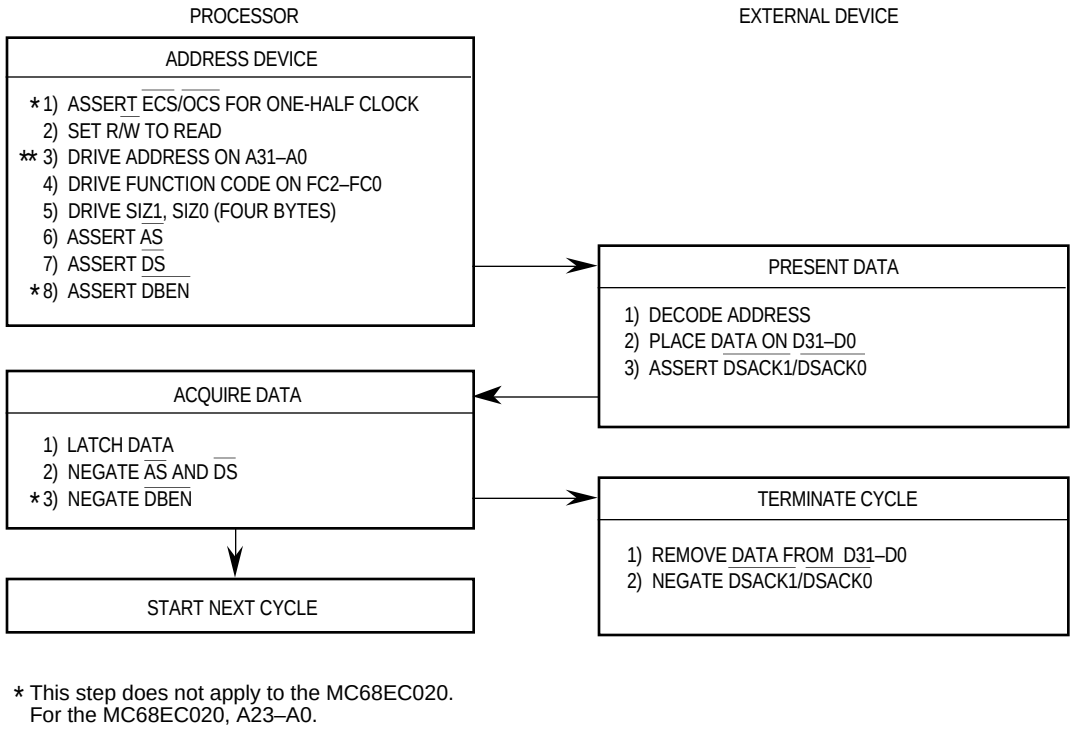
The bus transfers information between the MC68020/EC020 and an external memory, coprocessor, or peripheral device. External devices can accept or provide 8 bits, 16 bits, or 32 bits in parallel and must follow the handshake protocol described in this section. The maximum number of bits accepted or provided 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

5.3.1 Read Cycle

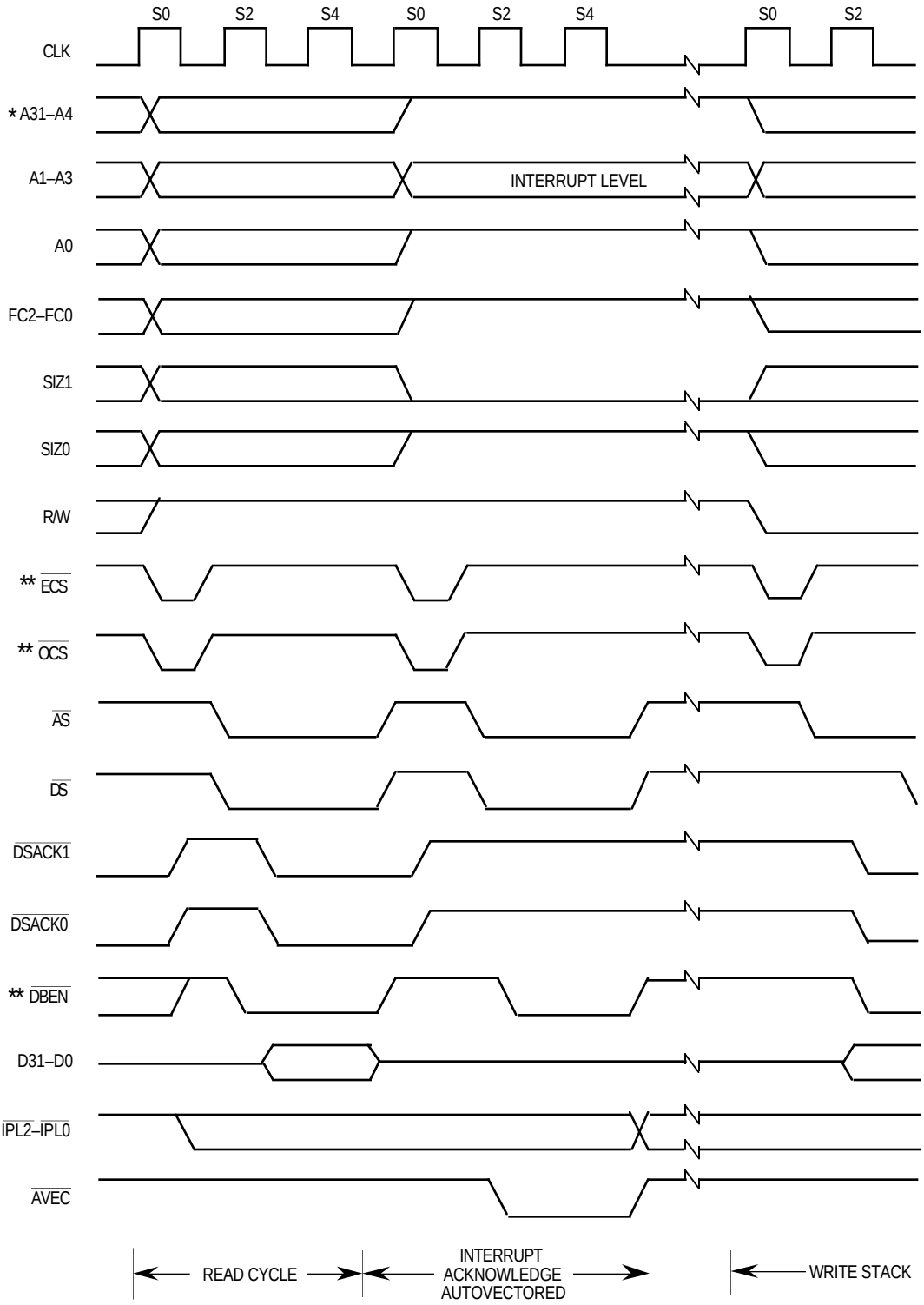
During a read cycle, the processor receives data from a memory, coprocessor, or peripheral device. If the instruction specifies a long-word operation, the MC68020/EC020 attempts to read four bytes at once. For a word operation, it attempts to read two bytes at once and for a byte operation, one byte. For some operations, the processor requests a three-byte transfer. The processor properly positions each byte internally. The section of the data bus from which each byte is read depends on the operand size, A1–A0, and the port size. Refer to **5.2.1 Dynamic Bus Sizing** and **5.2.2 Misaligned Operands** for more information on dynamic bus sizing and misaligned operands.

Figure 5-19 is a flowchart of a long-word read cycle. Figure 5-20 is a flowchart of a byte read cycle. Figures 5-21–5-23 are read cycle timing diagrams in terms of clock periods. Figure 5-21 corresponds to byte and word read cycles from a 32-bit port. Figure 5-22 corresponds to a long-word read cycle from an 8-bit port. Figure 5-23 also applies to a long-word read cycle, but from 16- and 32-bit ports.



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

Figure 5-19. Long-Word Read Cycle Flowchart



* For the MC68EC020, A23–A4.
** This signal does not apply to the MC68EC020.

Figure 5-34. Autovector Operation Timing

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.

5.7.2.3 BUS ARBITRATION CONTROL (MC68EC020). The bus arbitration control unit in the MC68EC020 is implemented with a finite state machine. As discussed previously, all asynchronous inputs to the MC68EC020 are internally synchronized in a maximum of two cycles of the processor clock.

As shown in Figure 5-48, the input signal labeled R is an internally synchronized version of the $\overline{BR}$ signal. The $\overline{BG}$ output is labeled G, and the internal high-impedance control signal is labeled T. If T is true, the address, data, and control buses are placed in the high-impedance state after the next rising edge following the negation of $\overline{AS}$ and $\overline{RMC}$. All signals are shown in positive logic (active high), regardless of their true active voltage level.

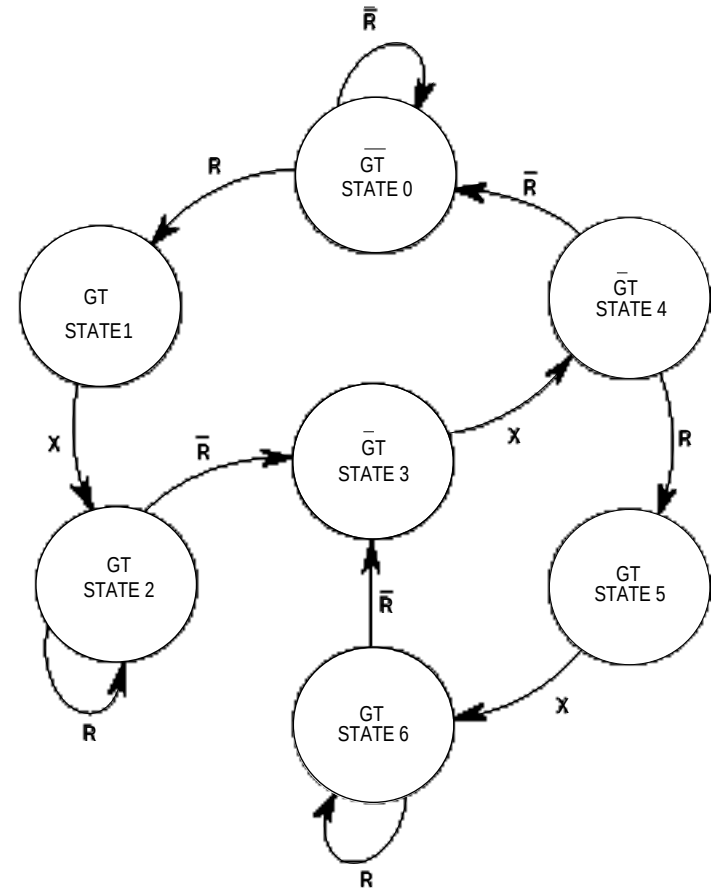


Figure 5-48. MC68EC020 Bus Arbitration State Diagram

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.

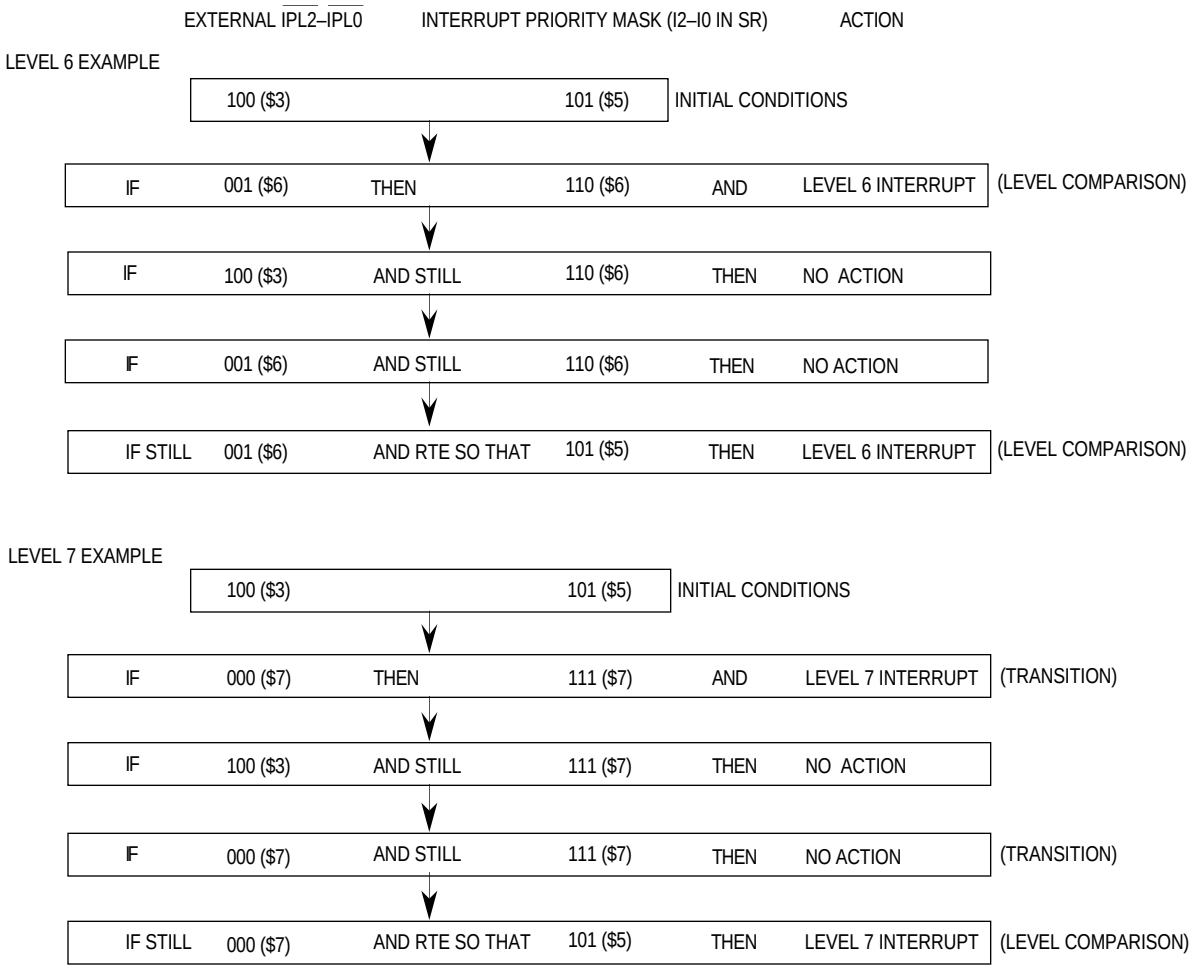


Figure 6-3. Interrupt Recognition Examples

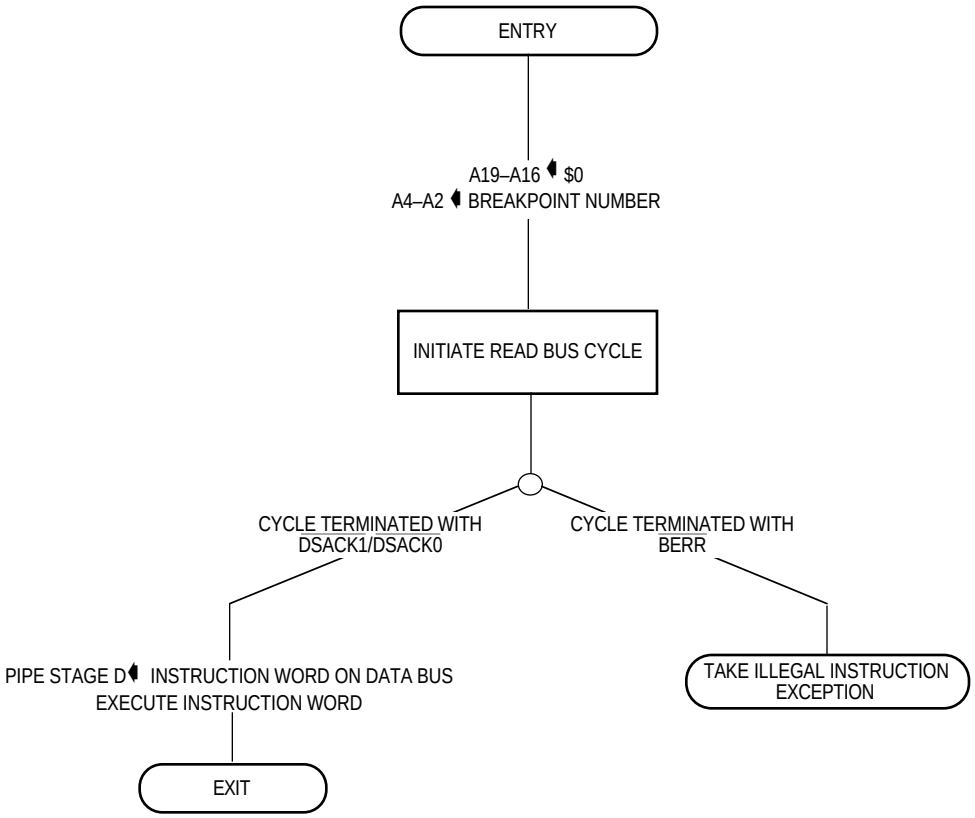


Figure 6-6. Breakpoint Instruction Flowchart

Table 6-4. Exception Priority Groups

Group/ Priority	Exception and Relative Priority	Characteristic
0	0.0—Reset	Aborts all processing (instruction or exception) and does not save old context.
1	1.0—Address Error 1.1—Bus Error	Suspends processing (instruction or exception) and saves internal context.
2	2.0—BKPT, CALLM, CHK, CHK2, cp Midinstruction, cp Protocol Violation, cpTRAPcc, Divide by Zero, RTE, RTM, TRAP, TRAPcc, TRAPV	Exception processing is part of instruction execution.
3	3.0—Illegal Instruction, Line A, Unimplemented Line F, Privilege Violation, cp Preinstruction	Exception processing begins before instruction is executed.
4	4.0—cp Postinstruction 4.1—Trace 4.2—Interrupt	Exception processing begins when current instruction or previous exception processing has completed.

NOTE: 0.0 is the highest priority; 4.2 is the lowest.

The final portion of the cpScc instruction format contains zero to five effective address extension words. These words contain any additional information required to calculate the effective address specified by bits 5–0 of the F-line operation word.

7.2.2.2.2 Protocol. Figure 7-8 shows the protocol for the cpScc instruction. The MC68020/EC020 transfers the condition selector to the coprocessor by writing the word following the F-line operation word to the condition CIR. The main processor then reads the response CIR to determine its next action. The coprocessor can return a response primitive to request services necessary to evaluate the condition. The operation of the cpScc instruction depends on the condition evaluation indicator returned to the main processor by the coprocessor. When the coprocessor returns the false condition indicator, the main processor evaluates the effective address specified by bits 5–0 of the F-line operation word and sets the byte at that effective address to FALSE (all bits cleared). When the coprocessor returns the true condition indicator, the main processor sets the byte at the effective address to TRUE (all bits set to one).

7.2.2.3 TEST COPROCESSOR CONDITION, DECREMENT, AND BRANCH INSTRUCTION. The operation of the test coprocessor condition, decrement, and branch instruction is similar to that of the DBcc instruction provided in the M68000 family instruction set. This operation uses a coprocessor-evaluated condition and a loop counter in the main processor. It is useful for implementing DO UNTIL constructs used in many high-level languages.

7.2.2.3.1 Format. Figure 7-12 shows the format of the test coprocessor condition, decrement, and branch instruction, denoted by the cpDBcc mnemonic.

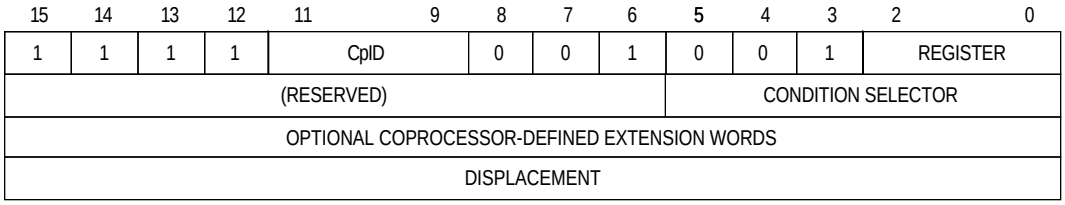


Figure 7-12. Test Coprocessor Condition, Decrement, and Branch Instruction Format (cpDBcc)

The first word of the cpDBcc instruction, F-line operation word, contains the CpID field in bits 11–9 and 001001 in bits 8–3 to identify the cpDBcc instruction. Bits 2–0 of this operation word specify the main processor data register used as the loop counter during the execution of the instruction.

The second word of the cpDBcc instruction format contains the coprocessor condition selector field in bits 5–0 and should contain zeros in bits 15–6 (reserved by Motorola) to maintain compatibility with future M68000 products. This word is written to the condition CIR to initiate execution of the cpDBcc instruction.

When the main processor reads the empty/reset format word from memory during the execution of the cpRESTORE instruction, it writes the format word to the restore CIR. The main processor then reads the restore CIR and, if the coprocessor returns the empty/reset format word, executes the next instruction. The main processor can then initialize the coprocessor by writing the empty/reset format code to restore the CIR. When the coprocessor receives the empty/reset format code, it terminates any current operations and waits for the main processor to initiate the next coprocessor instruction. In particular, after the cpRESTORE of the empty/reset format word, the execution of a cpSAVE should cause the empty/reset format word to be returned when a cpSAVE instruction is executed before any other coprocessor instructions. Thus, an empty/reset state frame consists only of the format word and the following reserved word in memory (refer to Figure 7-14).

7.2.3.2.2 Not-Ready Format Word. When the main processor initiates a cpSAVE instruction by reading the save CIR, the coprocessor can delay the save operation by returning a not-ready format word. The main processor then services any pending interrupts and reads the save CIR again. The not-ready format word delays the save operation until the coprocessor is ready to save its internal state. The cpSAVE instruction can suspend execution of a general or conditional coprocessor instruction; the coprocessor can resume execution of the suspended instruction when the appropriate state is restored with a cpRESTORE. If no further main processor services are required to complete coprocessor instruction execution, it may be more efficient to complete the instruction and thus reduce the size of the saved state. The coprocessor designer should consider the efficiency of completing the instruction or of suspending and later resuming the instruction when the main processor executes a cpSAVE instruction.

When the main processor initiates a cpRESTORE instruction by writing a format word to the restore CIR, the coprocessor should usually terminate any current operations and restore the state frame supplied by the main processor. Thus, the not-ready format word should usually not be returned by the coprocessor during the execution of a cpRESTORE instruction. If the coprocessor must delay the cpRESTORE operation for any reason, it can return the not-ready format word when the main processor reads the restore CIR. If the main processor reads the not-ready format word from the restore CIR during the cpRESTORE instruction, it reads the restore CIR again without servicing any pending interrupts.

7.2.3.2.3 Invalid Format Word. When the format word placed in the restore CIR to initiate a cpRESTORE instruction does not describe a valid coprocessor state frame, the coprocessor returns the invalid format word in the restore CIR. When the main processor reads this format word during the cpRESTORE instruction, it sets the abort bit in the control CIR and initiates format error exception processing.

A coprocessor usually should not place an invalid format word in the save CIR when the main processor initiates a cpSAVE instruction. A coprocessor, however, may not be able to support the initiation of a cpSAVE instruction while it is executing a previously initiated cpSAVE or cpRESTORE instruction. In this situation, the coprocessor can return the invalid format word when the main processor reads the save CIR to initiate the cpSAVE instruction while either another cpSAVE or cpRESTORE instruction is executing. If the

7.3.7 Condition CIR

The main processor initiates a conditional category instruction by writing the condition selector to bits 5–0 of the 16-bit condition CIR. Bits 15–6 are undefined and reserved by Motorola. The offset from the base address of the CIR set for the condition CIR is $0E. Figure 7-20 shows the format of the condition CIR.

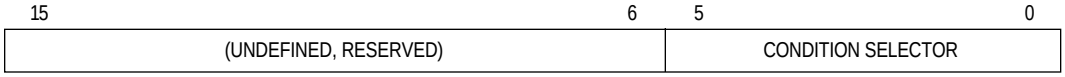


Figure 7-20. Condition CIR Format

7.3.8 Operand CIR

When the coprocessor requests the transfer of an operand, the main processor performs the transfer by reading from or writing to the 32-bit operand CIR. The offset from the base address of the CIR set for the operand CIR is $10.

The MC68020/EC020 aligns all operands transferred to and from the operand CIR to the most significant byte of this CIR. The processor performs a sequence of long-word transfers to read or write any operand larger than four bytes. If the operand size is not a multiple of four bytes, the portion remaining after the initial long-word transfer is aligned to the most significant byte of the operand CIR. Figure 7-21 shows the operand alignment used by the MC68020/EC020 when accessing the operand CIR.

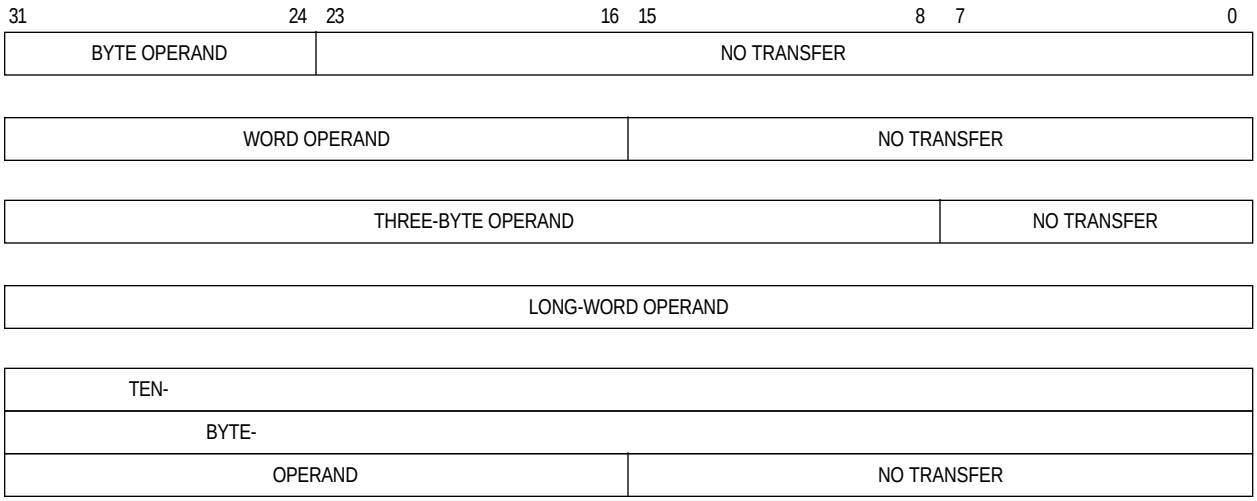


Figure 7-21. Operand Alignment for Operand CIR Accesses

The take address and transfer data primitive described in **7.4.11 Take Address and Transfer Data Primitive** does not replace the effective address value that has been calculated by the MC68020/EC020. The address that the main processor obtains in response to the take address and transfer data primitive is not available to the write to previously evaluated effective address primitive.

A coprocessor can issue an evaluate effective address and transfer data primitive followed by this primitive to perform a read-modify-write operation that is not indivisible. The bus cycles for this operation are normal bus cycles that can be interrupted, and the bus can be arbitrated between the cycles.

7.4.11 Take Address and Transfer Data Primitive

The take address and transfer data primitive transfers an operand between the coprocessor and an address supplied by the coprocessor. This primitive applies to general and conditional category instructions. Figure 7-31 shows the format of the take address and transfer data primitive.

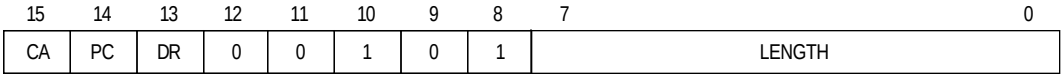


Figure 7-31. Take Address and Transfer Data Primitive Format

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

The length field of the primitive format specifies the operand length, which can be from 0–255 bytes.

The main processor reads a 32-bit address from the operand address CIR. Using a series of long-word transfers, the processor transfers the operand between this address and the operand CIR. The DR bit determines the direction of the transfer. The processor reads or writes the operand parts to ascending addresses, starting at the address from the operand address CIR. If the operand length is not a multiple of four bytes, the final operand part is transferred using a one-, two-, or three-byte transfer as required.

The function code used with the address read from the operand address CIR indicates either supervisor or user data space according to the value of the S-bit in the MC68020/EC020 SR.

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**.

The MC68020/EC020 also services interrupts if it reads the not-ready format word from the save CIR during a cpSAVE instruction. The MC68020/EC020 uses the normal four-word preinstruction stack frame (see Figure 7-41) when it services interrupts after reading the not-ready format word. Thus, the processor can service any pending interrupts and execute an RTE to return and reinitiate the cpSAVE instruction by reading the save CIR.

7.5.2.7 FORMAT ERRORS. The MC68020/EC020 can detect a format error while executing a cpSAVE or cpRESTORE instruction if the length field of a valid format word is not a multiple of four bytes. If the MC68020/EC020 reads a format word with an invalid length field from the save CIR during the cpSAVE instruction, it aborts the coprocessor instruction by writing an abort mask to the control CIR (refer to **7.3.2 Control CIR**) and initiates format error exception processing. If the MC68020/EC020 reads a format word with an invalid length field from the effective address specified in the cpRESTORE instruction, the MC68020/EC020 writes that format word to the restore CIR and then reads the coprocessor response from the restore CIR. The MC68020/EC020 then aborts the cpRESTORE instruction by writing an abort mask to the control CIR (refer to **7.3.2 Control CIR**) and initiates format error exception processing.

The MC68020/EC020 uses the four-word preinstruction stack frame (see Figure 7-41) and the format error vector number 14 when it initiates format error exception processing. Thus, if the exception handler does not modify the stack frame, the main processor, after it executes an RTE to return from the handler, attempts to restart the instruction during which the exception occurred.

7.5.2.8 ADDRESS AND BUS ERRORS. Coprocessor-instruction-related bus faults can occur during main processor bus cycles to CPU space to communicate with a coprocessor or during memory cycles run as part of the coprocessor instruction execution. If a bus error occurs during the CIR access that is used to initiate a coprocessor instruction, the main processor assumes that the coprocessor is not present and takes an F-line emulator exception as described in **7.5.2.2 F-Line Emulator Exceptions**. That is, the processor takes an F-line emulator exception when a bus error occurs during the initial access to a CIR by a coprocessor instruction. If a bus error occurs on any other coprocessor access or on a memory access made during the execution of a coprocessor instruction, the main processor performs bus error exception processing as described in **Section 6 Exception Processing**. After the exception handler has corrected the cause of the bus error, the main processor can return to the point in the coprocessor instruction at which the fault occurred.

An address error occurs if the MC68020/EC020 attempts to prefetch an instruction from an odd address. This can occur if the calculated destination address of a cpBcc or cpDBcc instruction is odd or if an odd value is transferred to the scanPC with the transfer status register and the scanPC response primitive. If an address error occurs, the MC68020/EC020 performs exception processing for a bus fault as described in **Section 6 Exception Processing**.

Comparing Tables 8-2 and 8-3 demonstrates that calculation of instruction timing cannot be a simple lookup of only BC or only WC timings. Even when the assumptions are known and fixed, as in the four examples summarized in Table 8-3, the microprocessor can sometimes achieve best-case timings under worst-case assumptions.

Looking across the four examples in Table 8-3 for an individual instruction, it is difficult to predict which timing table entry is used, since the influence of instruction overlap may or may not improve the BC, WC, or CC timings. When looking at the observed instruction timings for one example, it is also difficult to determine which combination of BC/CC/WC timing is required. Just how the instruction stream will fit and run with the cache enabled, how instructions are positioned in memory, and the degree of instruction overlap are factors that are impossible to account for in all combinations of the timing tables.

Although the timing tables cannot accurately predict the instruction timing that would be observed when executing an instruction stream on the MC68020/EC020, the tables can be used to calculate best-case and worst-case bounds for instruction timing. Absolute instruction timing must be measured by using the microprocessor itself to execute the target instruction stream.

Address Mode	Best Case	Cache Case	Worst Case
#<data>.W,([B],I,d_{32})	**11**(2/0/0)	**16**(2/0/0)	**20**(2/2/0)
#<data>.L,([d_{16},B],I,d_{32})	**12**(2/0/0)	**18**(2/0/0)	**22**(2/3/0)
#<data>.W,([d_{16},B],I)	**11**(2/0/0)	**16**(2/0/0)	**19**(2/2/0)
#<data>.L,([d_{16},B],I)	**12**(2/0/0)	**18**(2/0/0)	**21**(2/2/0)
#<data>.W,([d_{16},B],I,d_{16})	**13**(2/0/0)	**18**(2/0/0)	**22**(2/2/0)
#<data>.L,([d_{16},B],I,d_{16})	**14**(2/0/0)	**20**(2/0/0)	**24**(2/3/0)
#<data>.W,([d_{32},B],I)	**15**(2/0/0)	**20**(2/0/0)	**23**(2/3/0)
#<data>.L,([d_{32},B],I)	**16**(2/0/0)	**22**(2/0/0)	**25**(2/3/0)
#<data>.W,([d_{32},B],I,d_{16})	**17**(2/0/0)	**22**(2/0/0)	**25**(2/3/0)
#<data>.L,([d_{32},B],I,d_{16})	**18**(2/0/0)	**24**(2/0/0)	**27**(2/3/0)
#<data>.W,([d_{32},b],I,d_{32})	**17**(2/0/0)	**22**(2/0/0)	**27**(2/3/0)
#<data>.L,([d_{32},b],I,d_{32})	**18**(2/0/0)	**24**(2/0/0)	**29**(2/4/0)

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.

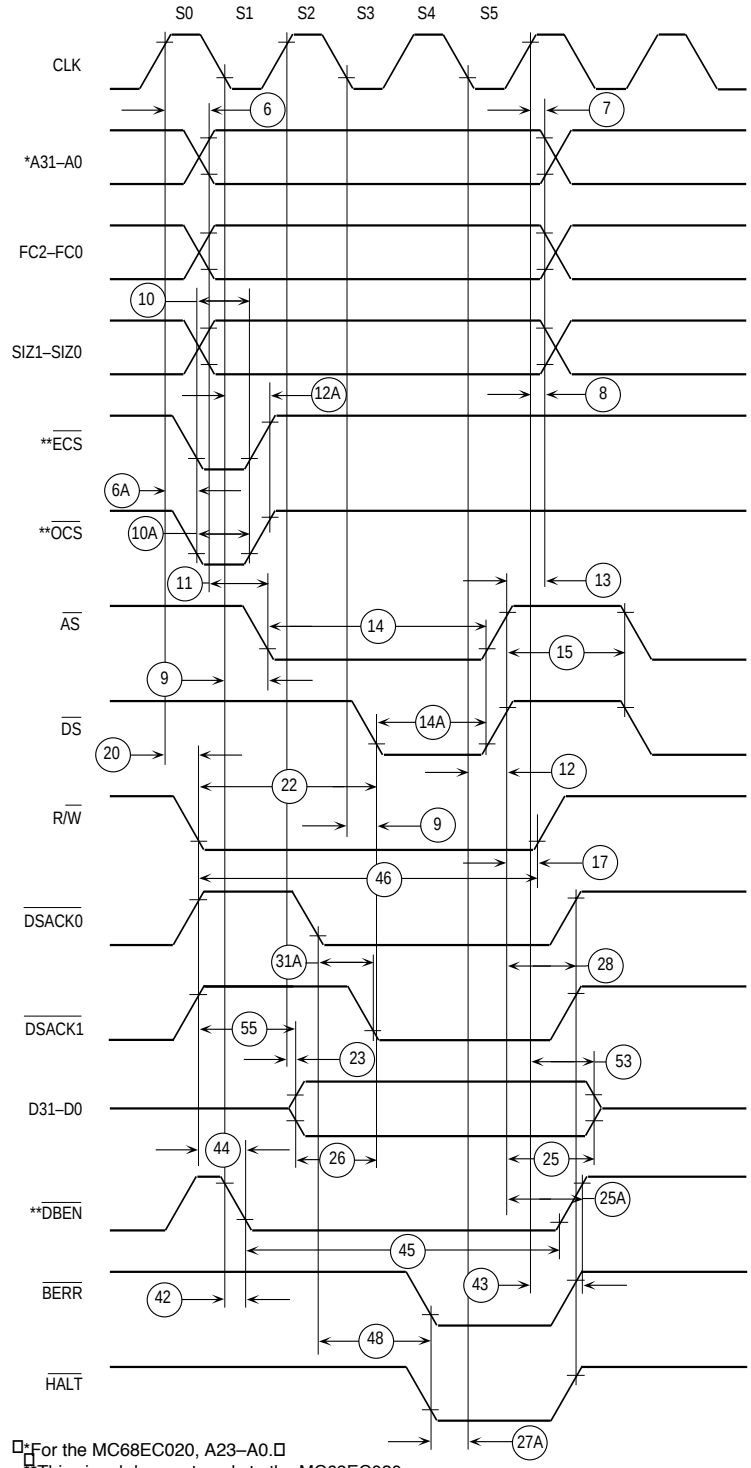
8.2.14 Bit Field Manipulation Instructions

The bit field manipulation instructions table indicates the number of clock periods needed for the processor to perform the specified bit field operation using 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
	BFTST	Dn	**3**(0/0/0)	**6**(0/0/0)	**7**(0/1/0)
‡	BFTST	Mem (< 5 Bytes)	**11**(1/0/0)	**11**(1/0/0)	**12**(1/1/0)
‡	BFTST	Mem (5 Bytes)	**15**(2/0/0)	**15**(2/0/0)	**16**(2/1/0)
	BFCHG	Dn	**9**(0/0/0)	**12**(0/0/0)	**12**(0/1/0)
‡	BFCHG	Mem (< 5 Bytes)	**16**(1/0/1)	**16**(1/0/1)	**16**(1/1/1)
‡	BFCHG	Mem (5 Bytes)	**24**(2/0/2)	**24**(2/0/2)	**24**(2/1/2)
	BFCLR	Dn	**9**(0/0/0)	**12**(0/0/0)	**12**(0/1/0)
‡	BFCLR	Mem (< 5 Bytes)	**16**(1/0/1)	**16**(1/0/1)	**16**(1/1/1)
‡	BFCLR	Mem (5 Bytes)	**24**(2/0/2)	**24**(2/0/2)	**24**(2/1/2)
	BFSET	Dn	**9**(0/0/0)	**12**(0/0/0)	**12**(0/1/0)
‡	BFSET	Mem (< 5 Bytes)	**16**(1/0/1)	**16**(1/0/1)	**16**(1/1/1)
‡	BFSET	Mem (5 Bytes)	**24**(2/0/2)	**24**(2/0/2)	**24**(2/1/2)
	BFEXTS	Dn	**5**(0/0/0)	**8**(0/0/0)	**8**(0/1/0)
‡	BFEXTS	Mem (< 5 Bytes)	**13**(1/0/0)	**13**(1/0/0)	**13**(1/1/0)
‡	BFEXTS	Mem (5 Bytes)	**18**(2/0/0)	**18**(2/0/0)	**18**(2/1/0)
	BFEXTU	Dn	**5**(0/0/0)	**8**(0/0/0)	**8**(0/1/0)
‡	BFEXTU	Mem (< 5 Bytes)	**13**(1/0/0)	**13**(1/0/0)	**13**(1/1/0)
‡	BFEXTU	Mem (5 Bytes)	**18**(2/0/0)	**18**(2/0/0)	**18**(2/1/0)
	BFINS	Dn	**7**(0/0/0)	**10**(0/0/0)	**10**(0/1/0)
‡	BFINS	Mem (< 5 Bytes)	**14**(1/0/1)	**14**(1/0/1)	**15**(1/1/1)
‡	BFINS	Mem (5 Bytes)	**20**(2/0/2)	**20**(2/0/2)	**21**(2/1/2)
	BFFFO	Dn	**15**(0/0/0)	**18**(0/0/0)	**18**(0/1/0)
‡	BFFFO	Mem (< 5 Bytes)	**24**(1/0/0)	**24**(1/0/0)	**24**(1/1/0)
‡	BFFFO	Mem (5 Bytes)	**32**(2/0/0)	**32**(2/0/0)	**32**(2/1/0)

‡Add Calculate Immediate Address Time

NOTE: A bit field of 32 bits may span five bytes that require two operand cycles to access or may span four bytes that require only one operand cycle to access.



*For the MC68EC020, A23–A0.

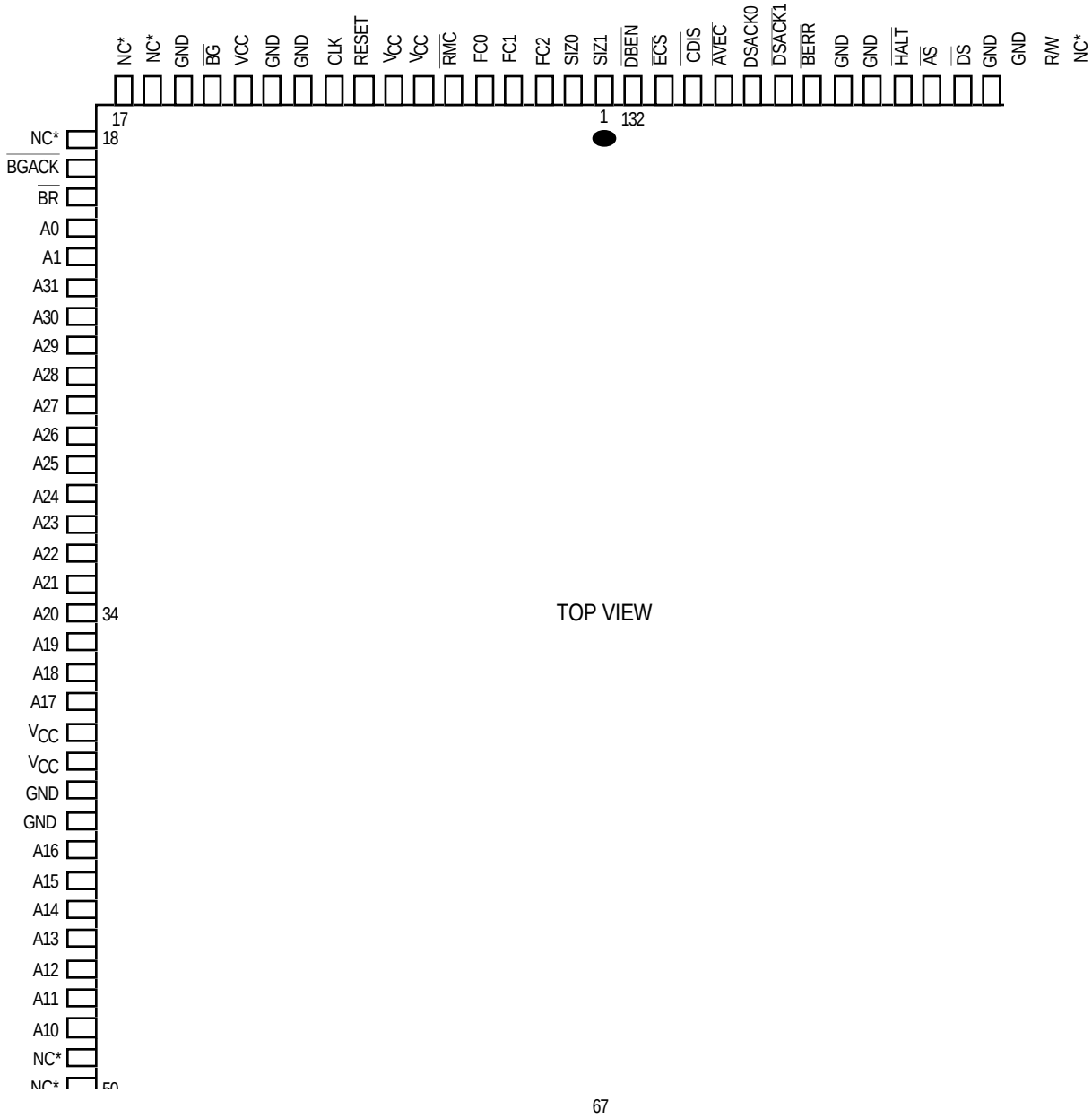
**This signal does not apply to the MC68EC020.

NOTE: Timing measurements are referenced to and from a low voltage of 0.8 V and a high voltage of 2.0 V, unless otherwise noted. The voltage swing through this range should start outside and pass through the range such that the rise or fall will be linear between 0.8 V and 2.0 V.

FIGURE 10-4
MC68020UM

Figure 10-4. Write Cycle Timing Diagram

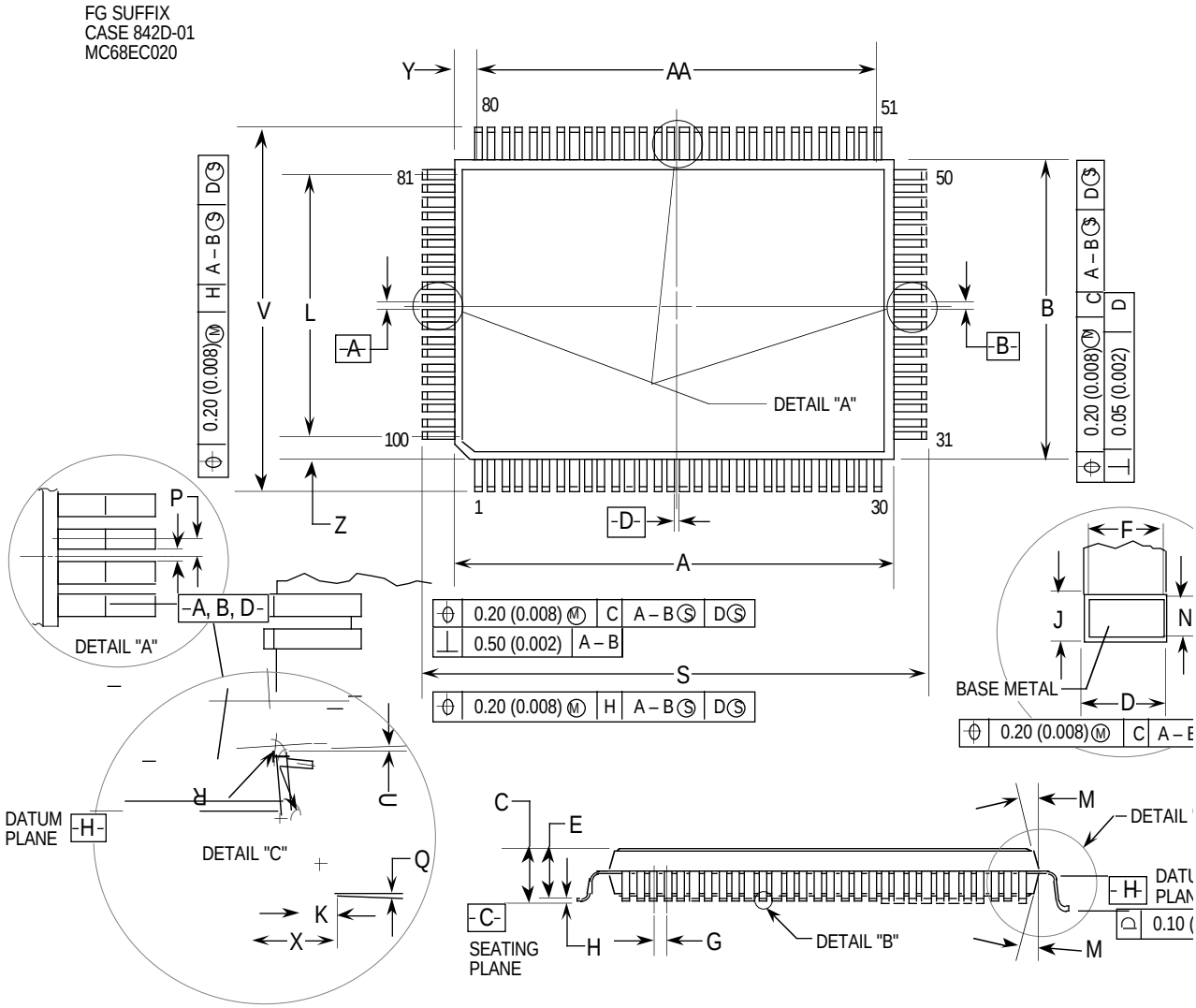
11.2.4 MC68020 FC and FE Suffix—Pin Assignment



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. NC pins are reserved by Motorola for future use and should have no external connection.

Group	V_{CC}	GND
Address Bus	13, 38, 39	15, 40, 41, 62
Data Bus	79, 80, 96, 97	77, 78, 98, 99, 119, 120
Logic	7, 8, 65, 66	67, 68, 124, 125
Clock	—	11, 12

11.2.10 MC68EC020 FG Suffix—Package Dimensions



DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	19.90	20.10	0.783	0.791
B	13.90	14.10	0.547	0.555
C	—	3.30	—	0.130
D	0.22	0.38	0.009	0.015
E	2.55	3.05	0.100	0.120
F	0.22	0.33	0.009	0.013
G	0.65 BSC		0.026 BSC	
H	0.10	0.36	0.004	0.014
J	0.13	0.23	0.005	0.009
K	0.65	0.95	0.026	0.037
L	12.35 REF		0.486 REF	
M	5°	16°	5°	16°
N	0.13	0.17	0.005	0.007
P	0.325 BSC		0.013 BSC	
Q	0°	7°	0°	7°
R	0.25	0.35	0.010	0.014
S	23.65	24.15	0.931	0.951
T	0.13	—	0.005	—
U	0°	—	0°	—

NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: MILLIMETER.
3. DATUM PLANE **-H-** IS LOCATED AT BOTTOM OF LEAD AND IS COINCIDENT WITH THE LEAD WHERE THE LEAD EXITS THE PLASTIC BODY AT THE BOTTOM OF THE PARTING LINE.
4. DATUMS **-A-**, **-B-**, AND **-D-** TO BE DETERMINED AT DATUM PLANE **-H-**.
5. DIMENSIONS S AND V TO BE DETERMINED AT SEATING PLANE **-C-**.
6. DIMENSIONS A AND B DO NOT INCLUDE MOLD PROTRUSION. ALLOW PROTRUSION IS 0.25 (0.010) PER SIDE. DIMENSIONS A AND B DO INCLU MOLD MISMATCH AND ARE DETERMINED AT DATUM PLANE **-H-**.
7. DIMENSION D DOES NOT INCLUDE DAMBAR PROTRUSION. ALLOWAE DAMBAR PROTRUSION SHALL BE 0.08 (0.003) TOTAL IN EXCESS OF THE DIMENSION AT MAXIMUM MATERIAL CONDITION. DAMBAR CANNOT BE LOCATED ON THE LOWER RADIUS OR THE FOOT.