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	166MHz
Co-Processors/DSP	-
RAM Controllers	-
Graphics Acceleration	No
Display & Interface Controllers	-
Ethernet	-
SATA	-
USB	-
Voltage - I/O	5.0V
Operating Temperature	-40°C ~ 85°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=mc68020crc16e

LIST OF TABLES

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.

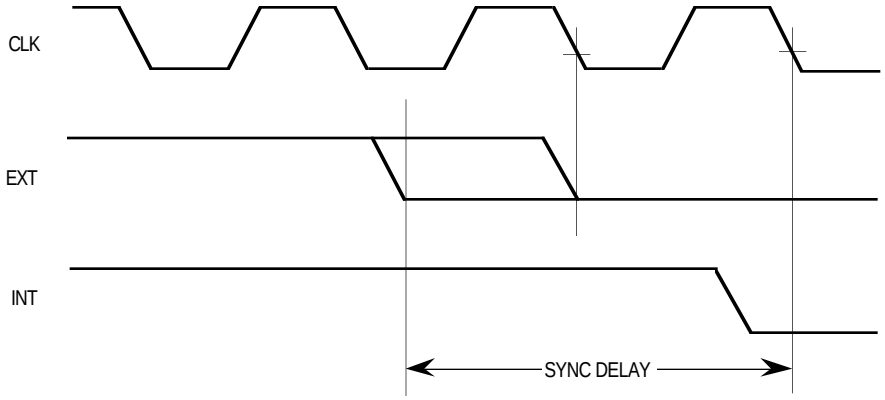


Figure 5-1. Relationship between External and Internal Signals

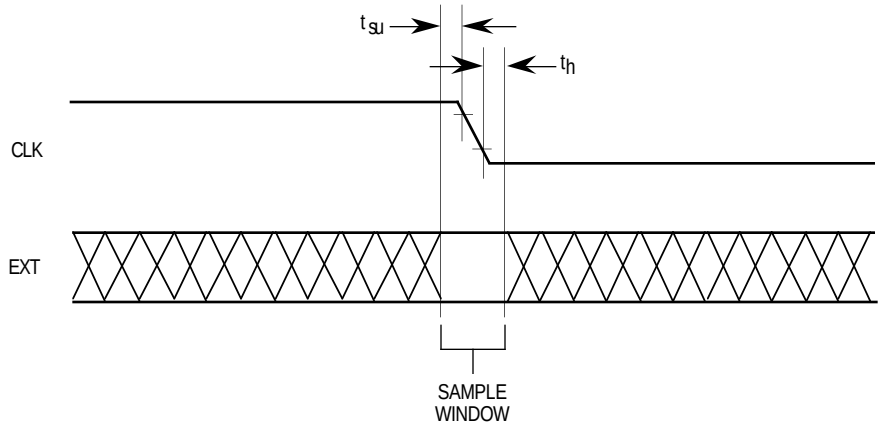
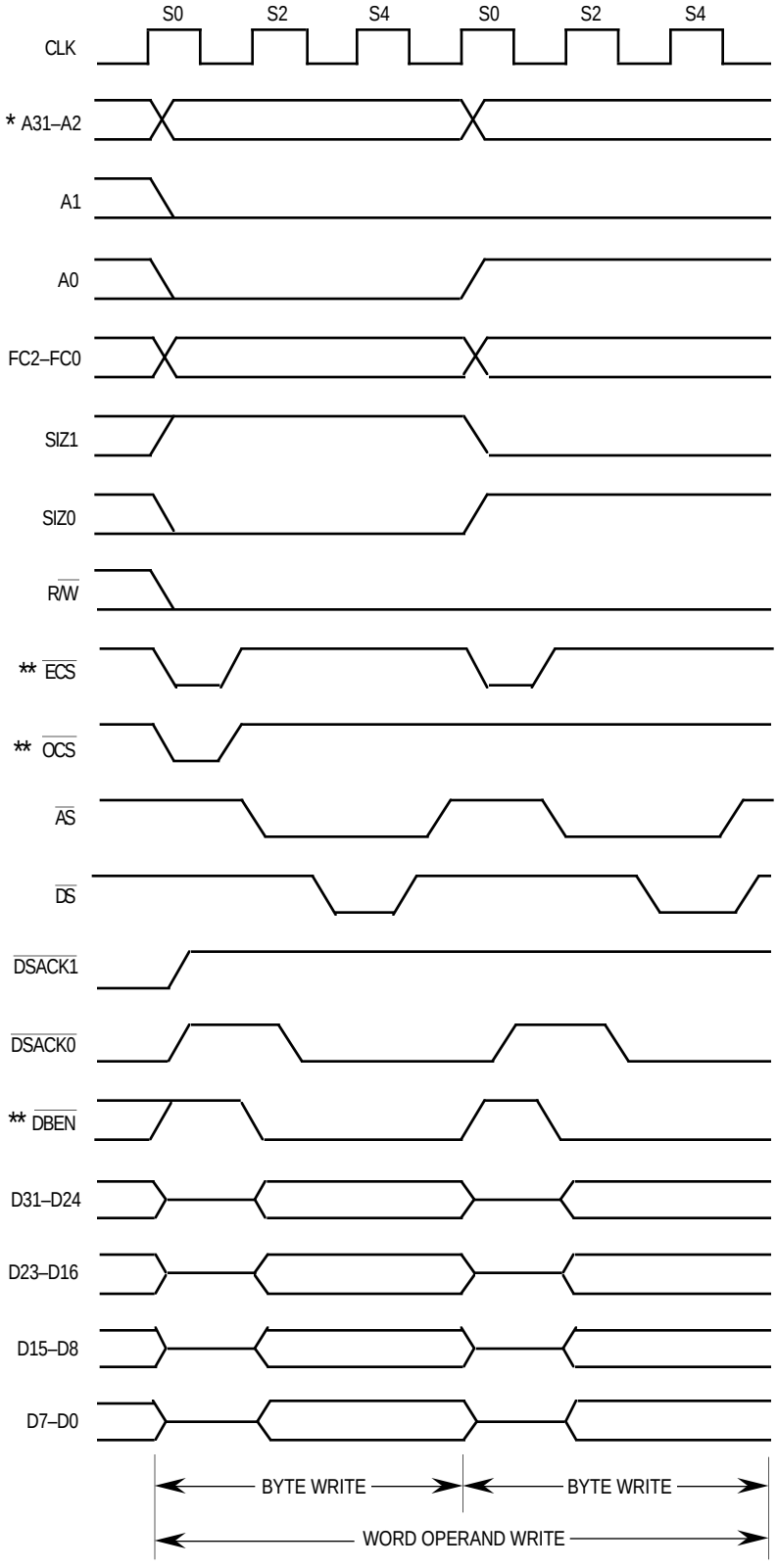


Figure 5-2. Input Sample Window

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 the MC68EC020, A23–A2.
** This signal does not apply to the MC68EC020.

Figure 5-8. Word Operand Write to Byte Port Timing

5.6 BUS SYNCHRONIZATION

The MC68020/EC020 overlaps instruction execution—that is, during bus activity for one instruction, instructions that do not use the external bus can be executed. Due to the independent operation of the on-chip cache relative to the operation of the bus controller, many subsequent instructions can be executed, resulting in seemingly nonsequential instruction execution. When this is not desired and the system depends on sequential execution following bus activity, the NOP instruction can be used. The NOP instruction forces instruction and bus synchronization by freezing instruction execution until all pending bus cycles have completed.

An example of the use of the NOP instruction for this purpose is the case of a write operation of control information to an external register in which the external hardware attempts to control program execution based on the data that is written with the conditional assertion of $\overline{\text{BERR}}$. Since the MC68020/EC020 cannot process the bus error until the end of the bus cycle, the external hardware has not successfully interrupted program execution. To prevent a subsequent instruction from executing until the external cycle completes, the NOP instruction can be inserted after the instruction causing the write. In this case, bus error exception processing proceeds immediately after the write and before subsequent instructions are executed. This is an irregular situation, and the use of the NOP instruction for this purpose is not required by most systems.

5.7 BUS ARBITRATION

The bus design of the MC68020/EC020 provides for a single bus master at any one time: either the processor or an external device. One or more of the external devices on the bus can have the capability of becoming bus master. Bus arbitration is the protocol by which an external device becomes bus master; the bus controller in the MC68020/EC020 manages the bus arbitration signals so that the processor has the lowest priority.

Bus arbitration differs in the MC68020 and MC68EC020 due to the absence of $\overline{\text{BGACK}}$ in the MC68EC020. Because of this difference, bus arbitration of the MC68020 and MC68EC020 is discussed separately.

External devices that need to obtain the bus must assert the bus arbitration signals in the sequences described in **5.7.1 MC68020 Bus Arbitration** or **5.7.2 MC68EC020 Bus Arbitration**. Systems having several devices that can become bus master require external circuitry to assign priorities to the devices, so that when two or more external devices attempt to become bus master at the same time, the one having the highest priority becomes bus master first.

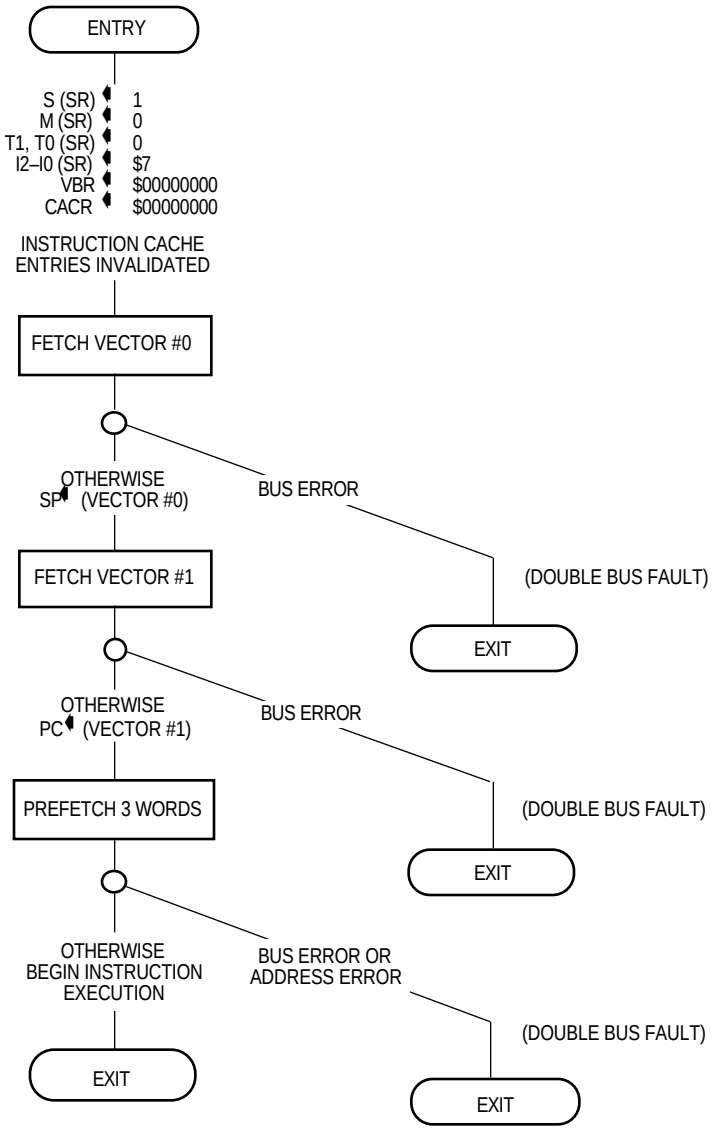


Figure 6-1. Reset Operation Flowchart

The processor begins exception processing for a bus error by making an internal copy of the current SR. The processor then enters the supervisor privilege level (by setting the S-bit in the SR) and clears the T1 and T0 bits in the SR. The processor generates exception vector number 2 for the bus error vector. It saves the vector offset, PC, and the internal copy of the SR on the active supervisor stack. The saved PC value is the logical address of the instruction that was executing at the time the fault was detected. This is not necessarily the instruction that initiated the bus cycle since the processor overlaps

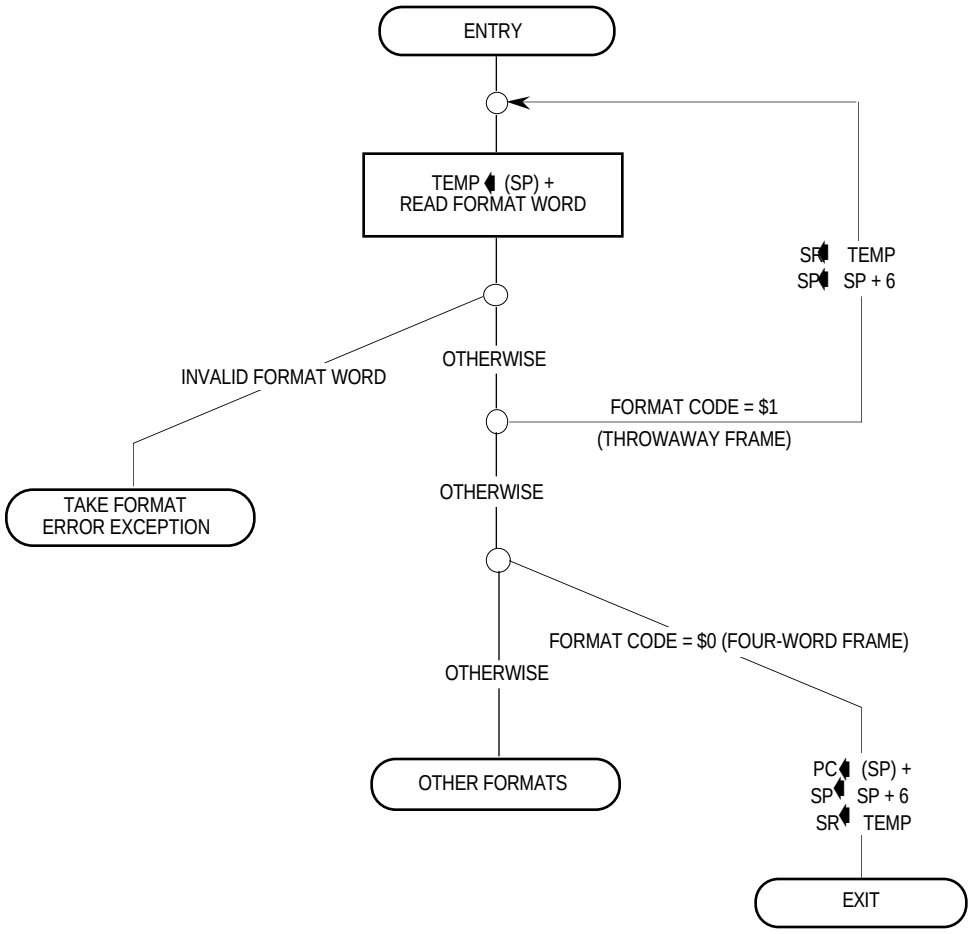
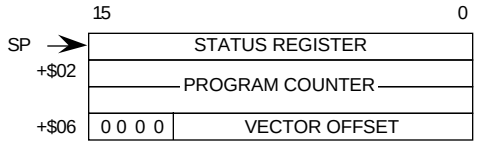
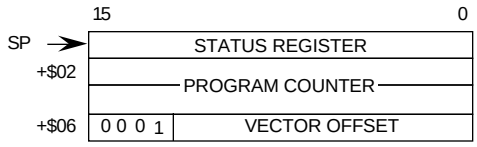
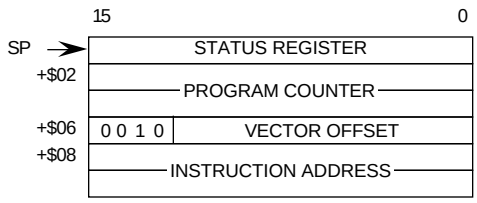
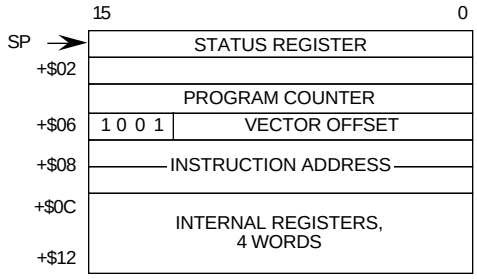


Figure 6-7. RTE Instruction for Throwaway Four-Word Frame

For the coprocessor midinstruction stack frame, the processor reads the SR, PC, instruction address, internal register values, and the evaluated effective address from the stack, restores these values to the corresponding internal registers, and increments the stack pointer by 20. The processor then reads from the response register of the coprocessor that initiated the exception to determine the next operation to be performed. Refer to **Section 7 Coprocessor Interface Description** for details of coprocessor-related exceptions.

For both the short and long bus fault stack frames, the processor first checks the format value on the stack for validity. In addition, for the long stack frame, the processor compares the version number in the stack with its own version number. The version number is located in the most significant nibble (bits 15–12) of the word at location SP + $36 in the long stack frame. This validity check is required in a multiprocessor system to ensure that the data is properly interpreted by the RTE instruction. The RTE instruction also reads from both ends of the stack frame to make sure it is accessible. If the frame is invalid or inaccessible, the processor takes a format error or a bus error exception, respectively. Otherwise, the processor reads the entire frame into the proper internal registers, deallocates the stack, and resumes normal processing. Once the processor begins to load the frame to restore its internal state, the assertion of the BERR signal

Table 6-5. Exception Stack Frames

Stack Frames	Exception Types (Stacked PC Points to)
 FOUR-WORD STACK FRAME — FORMAT $0	● Interrupt [Next instruction] ● Format Error [RTE or cpRESTORE instruction ● TRAP #N [NEXT instruction] ● Illegal Instruction [Illegal instruction] ● A-Line Instruction [A-line instruction] ● F-Line Instruction [F-line instruction] ● Privilege Violation [First word of instruction causing Privilege Violation] ● Coprocessor Preinstruction [Opword of instruction that returned the 'take preinstruction' primitive]
 THROWAWAY FOUR-WORD STACK FRAME — FORMAT $1	● Created on Interrupt Stack during interrupt exception processing when transition from master state to interrupt state occurs [Next instruction — same as on master stack]
 SIX-WORD STACK FRAME — FORMAT $2	● CHK ● CHK2 ● cpTRAPcc ● TRAPcc ● TRAPPV ● Trace ● Zero Divide ● MMU Configuration ● Coprocessor Postinstruction [Next instruction for all these exceptions] INSTRUCTION ADDRESS is the address of the instruction that caused the exception
 COPROCESSOR MIDINSTRUCTION STACK FRAME (10 WORDS) — FORMAT $9	● Coprocessor Midinstruction ● Main-Detected Protocol Violation ● Interrupt Detected During Coprocessor Instruction (supported with 'null come again with interrupts allowed' primitive) [Next word to be fetched from instruction stream for all these exceptions] INSTRUCTION ADDRESS is the address of the instruction that caused the exception

During coprocessor instruction execution, the MC68020/EC020 executes CPU space bus cycles to access the CIR set. The MC68020/EC020 asserts FC2–FC0, identifying a CPU space bus cycle. The CIR set is mapped into CPU space in the same manner that a peripheral interface register set is generally mapped into data space. The information encoded on FC2–FC0 and the address bus of the MC68020/EC020 during a coprocessor access is used to generate the chip select signal for the coprocessor being accessed. Other address lines select a register within the interface set. The information encoded on the function code and address lines of the MC68020/EC020 during a coprocessor access is illustrated in Figure 7-3.

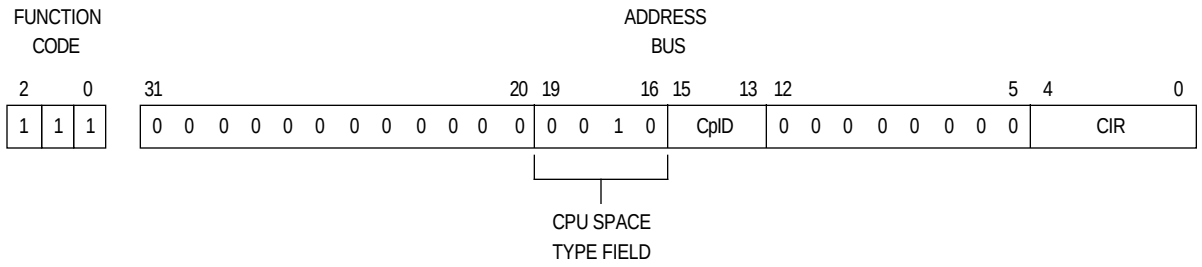


Figure 7-3. MC68020/EC020 CPU Space Address Encodings

Signals A19–A16 of the MC68020/EC020 address bus specify the CPU space cycle type for a CPU space bus cycle. The types of CPU space cycles currently defined for the MC68020/EC020 are interrupt acknowledge, breakpoint acknowledge, module support operations, and coprocessor access cycles. CPU space type $2 (A19–A16 = 0010) specifies a coprocessor access cycle.

A15–A13 specify the CpID code for the coprocessor being accessed. This code is transferred from bits 11–9 of the coprocessor instruction operation word (refer to Figure 7-1) to the address bus during each coprocessor access. Thus, decoding the MC68020/EC020 FC2–FC0 and A19–A13 signals provides a unique chip select signal for a given coprocessor. The FC2–FC0 and A19–A16 signals indicate a coprocessor access; A15–A13 indicate which of the possible eight coprocessors (000–111) is being accessed. Bits A31–A20 and A12–A5 of the MC68020 address bus and bits A23–A20 and A12–A5 of the MC68EC020 address bus are always zero during a coprocessor access.

7.1.4.3 COPROCESSOR INTERFACE REGISTER SELECTION. Figure 7-4 shows that the value on the MC68020/EC020 address bus during a coprocessor access addresses a unique region of the main processor's CPU address space. Signals A4–A0 of the MC68020/EC020 address bus select the CIR being accessed. The register map for the M68000 coprocessor interface is shown in Figure 7-5. The individual registers are described in detail in **7.3 Coprocessor Interface Register Set**.

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.

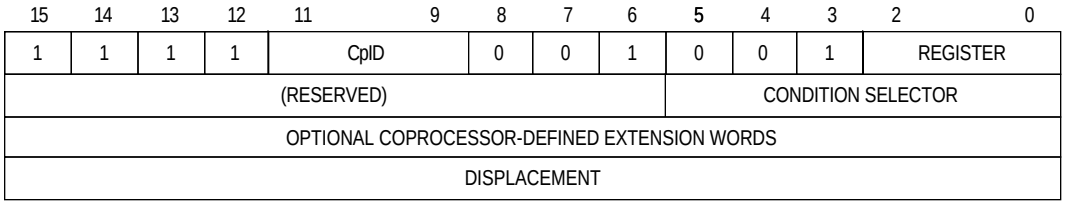


Figure 7-12. Test Coprocessor Condition, Decrement, and Branch Instruction Format (cpDBcc)

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.

The encoding of bits 12–0 of a coprocessor response primitive depends on the individual primitive. Bits 15–13, however, specify optional additional operations that apply to most of the primitives defined for the M68000 coprocessor interface.

The CA bit specifies the come-again operation of the main processor. When the main processor reads a response primitive from the response CIR with the CA bit set, it performs the service indicated by the primitive and then reads the response CIR again. Using the CA bit, a coprocessor can transfer several response primitives to the main processor during the execution of a single coprocessor instruction.

The PC bit specifies the pass program counter operation. When the main processor reads a primitive with the PC bit set from the response CIR, the main processor immediately passes the current value in its program counter to the instruction address CIR as the first operation in servicing the primitive request. The value in the program counter is the address of the F-line operation word of the coprocessor instruction currently executing. The PC bit is implemented in all coprocessor response primitives currently defined for the M68000 coprocessor interface.

When an undefined primitive or a primitive that requests an illegal operation is passed to the main processor, the main processor initiates exception processing for either an F-line emulator or a protocol violation exception (refer to **7.5.2 Main-Processor-Detected Exceptions**). If the PC bit is set in one of these response primitives, however, the main processor passes the program counter to the instruction address CIR before it initiates exception processing.

When the main processor initiates a cpGEN instruction that can be executed concurrently with main processor instructions, the PC bit is usually set in the first primitive returned by the coprocessor. Since the main processor proceeds with instruction stream execution once the coprocessor releases it, the coprocessor must record the instruction address to support any possible exception processing related to the instruction. Exception processing related to concurrent coprocessor instruction execution is discussed in **7.5.1 Coprocessor-Detected Exceptions**.

The DR bit is the direction bit. It applies to operand transfers between the main processor and the coprocessor. If the DR bit is clear, the direction of transfer is from the main processor to the coprocessor (main processor write). If the DR bit is set, the direction of transfer is from the coprocessor to the main processor (main processor read). If the operation indicated by a given response primitive does not involve an explicit operand transfer, the value of this bit depends on the particular primitive encoding.

The DR bit specifies the direction of the operand transfer. DR = 0 requests a transfer from the main processor to the coprocessor, and DR = 1 specifies a transfer from the coprocessor to the main processor.

If the effective addressing mode specifies the predecrement mode, the address register used is decremented by the size of the operand before the transfer. The bytes within the operand are then transferred to or from ascending addresses beginning with the location specified by the decremented address register. In this mode, if A7 is used as the address register and the operand length is one byte, A7 is decremented by two to maintain a word-aligned stack.

For the postincrement effective addressing mode, the address register used is incremented by the size of the operand after the transfer. The bytes within the operand are transferred to or from ascending addresses beginning with the location specified by the address register. In this mode, if A7 is used as the address register and the operand length is one byte, A7 is incremented by two after the transfer to maintain a word-aligned stack. Transferring odd length operands longer than one byte using the –(A7) or (A7)+ addressing modes can result in a stack pointer that is not word aligned.

The processor repeats the effective address calculation each time this primitive is issued during the execution of a given instruction. The calculation uses the current contents of any required address and data registers. The instruction must include a set of effective address extension words for each repetition of a calculation that requires them. The processor locates these words at the current scanPC location and increments the scanPC by two for each word referenced in the instruction stream.

The MC68020/EC020 sign-extends a byte or word-sized operand to a long-word value when it is transferred to an address register (A7–A0) using this primitive with the register direct effective addressing mode. A byte or word-sized operand transferred to a data register (D7–D0) only overwrites the lower byte or word of the data register.

7.4.10 Write to Previously Evaluated Effective Address Primitive

The write to previously evaluated effective address primitive transfers an operand from the coprocessor to a previously evaluated effective address. This primitive applies to general category instructions. If the coprocessor uses this primitive during the execution of a conditional category instruction, the main processor initiates protocol violation exception processing. Figure 7-30 shows the format of the write to previously evaluated effective address primitive.

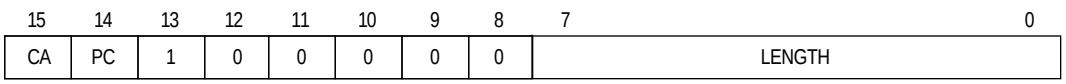


Figure 7-30. Write to Previously Evaluated Effective Address Primitive Format

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.

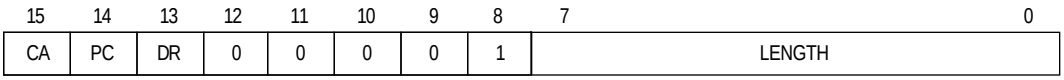


Figure 7-37. Transfer Multiple Coprocessor Registers Primitive Format

The transfer multiple coprocessor registers primitive uses the CA, PC, and DR bits as described in **7.4.2 Coprocessor Response Primitive General Format**.

The length field of the primitive format indicates the length in bytes of each operand transferred. The operand length must be an even number of bytes; odd length operands cause the MC68020/EC020 to initiate protocol violation exception processing (refer to **7.5.2.1 Protocol Violations**).

When the main processor reads this primitive, it calculates the effective address specified in the coprocessor instruction. The scanPC should be pointing to the first of any necessary effective address extension words when this primitive is read from the response CIR; the scanPC is incremented by two for each extension word referenced during the effective address calculation. For transfers from the effective address to the coprocessor (DR = 0), the control addressing modes and the postincrement addressing mode are valid. For transfers from the coprocessor to the effective address (DR = 1), the control alterable and predecrement addressing modes are valid. Invalid addressing modes cause the MC68020/EC020 to abort the instruction by writing an abort mask to the control CIR (refer to **7.3.2 Control CIR**) and to initiate F-line emulator exception processing (refer to **7.5.2.2 F-Line Emulator Exceptions**).

After performing the effective address calculation, the MC68020/EC020 reads a 16-bit register select mask from the register select CIR. The coprocessor uses the register select mask to specify the number of operands to transfer; the MC68020/EC020 counts the number of ones in the register select mask to determine the number of operands. The order of the ones in the register select mask is not relevant to the operation of the main processor. As many as 16 operands can be transferred by the main processor in response to this primitive. The total number of bytes transferred is the product of the number of operands transferred and the length of each operand specified in the length field of the primitive format.

If DR = 1, the main processor reads the number of operands specified in the register select mask from the operand CIR and writes these operands to the effective address specified in the instruction using long-word transfers whenever possible. If DR = 0, the main processor reads the number of operands specified in the register select mask from the effective address and writes them to the operand CIR.

For the control addressing modes, the operands are transferred to or from memory using ascending addresses. For the postincrement addressing mode, the operands are read from memory with ascending addresses also, and the address register used is incremented by the size of an operand after each operand is transferred. The address register used with the (An)+ addressing mode is incremented by the total number of bytes transferred during the primitive execution.

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

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

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

7.5 EXCEPTIONS

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

7.5.1 Coprocessor-Detected Exceptions

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

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

CACHE 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	**7**(0/0/1)	**9**(0/0/1)	**8**(0/0/1)	**10**(0/0/1)	**14**(0/0/1)	**12**(1/0/1)	**14**(1/0/1)	**15**(1/0/1)
#<data>.B,W	**7**(0/0/1)	**9**(0/0/1)	**8**(0/0/1)	**10**(0/0/1)	**14**(0/0/1)	**12**(1/0/1)	**14**(1/0/1)	**15**(1/0/1)
#<data>.L	**9**(0/0/1)	**11**(0/0/1)	**10**(0/0/1)	**12**(0/0/1)	**16**(0/0/1)	**14**(1/0/1)	**16**(1/0/1)	**17**(1/0/1)
(An)	**9**(1/0/1)	**11**(1/0/1)	**10**(1/0/1)	**12**(1/0/1)	**16**(1/0/1)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/0/1)
(An)+	**9**(1/0/1)	**11**(1/0/1)	**10**(1/0/1)	**12**(1/0/1)	**16**(1/0/1)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/0/1)
–(An)	**10**(1/0/1)	**12**(1/0/1)	**11**(1/0/1)	**13**(1/0/1)	**17**(1/0/1)	**15**(2/0/1)	**17**(2/0/1)	**18**(2/0/1)
(d$_{16}$,An) or (d$_{16}$,PC)	**10**(1/0/1)	**12**(2/0/1)	**11**(1/0/1)	**13**(1/0/1)	**17**(1/0/1)	**15**(2/0/1)	**17**(2/0/1)	**18**(2/0/1)
(xxx).W	**9**(1/0/1)	**11**(1/0/1)	**10**(1/0/1)	**12**(1/0/1)	**16**(1/0/1)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/0/1)
(xxx).L	**9**(1/0/1)	**11**(1/0/1)	**10**(1/0/1)	**12**(1/0/1)	**16**(1/0/1)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/0/1)
(d$_8$,An,Xn) or (d$_8$,PC,Xn)	**12**(1/0/1)	**14**(1/0/1)	**13**(1/0/1)	**15**(1/0/1)	**19**(1/0/1)	**17**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)
(d$_{16}$,An,Xn) or (d$_{16}$,PC,Xn)	**12**(1/0/1)	**14**(1/0/1)	**13**(1/0/1)	**15**(1/0/1)	**19**(1/0/1)	**17**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)
(B)	**12**(1/0/1)	**14**(1/0/1)	**13**(1/0/1)	**15**(1/0/1)	**19**(1/0/1)	**17**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)
(d$_{16}$,B)	**14**(1/0/1)	**16**(1/0/1)	**15**(1/0/1)	**17**(1/0/1)	**21**(1/0/1)	**19**(2/0/1)	**21**(2/0/1)	**22**(2/0/1)
(d$_{32}$,B)	**18**(1/0/1)	**20**(1/0/1)	**19**(1/0/1)	**21**(1/0/1)	**25**(1/0/1)	**23**(2/0/1)	**25**(2/0/1)	**26**(2/0/1)
([B],I)	**17**(2/0/1)	**19**(2/0/1)	**18**(2/0/1)	**20**(2/0/1)	**24**(2/0/1)	**22**(3/0/1)	**24**(3/0/1)	**25**(3/0/1)
([B],I,d$_{16}$)	**19**(2/0/1)	**21**(2/0/1)	**20**(2/0/1)	**22**(2/0/1)	**26**(2/0/1)	**24**(3/0/1)	**26**(3/0/1)	**27**(3/0/1)
([B],I,d$_{32}$)	**19**(2/0/1)	**21**(2/0/1)	**20**(2/0/1)	**22**(2/0/1)	**26**(2/0/1)	**24**(3/0/1)	**26**(3/0/1)	**27**(3/0/1)
([d$_{16}$,B],I)	**19**(2/0/1)	**21**(2/0/1)	**20**(2/0/1)	**22**(2/0/1)	**26**(2/0/1)	**24**(3/0/1)	**26**(3/0/1)	**27**(3/0/1)
([d$_{16}$,B],I,d$_{16}$)	**21**(2/0/1)	**23**(2/0/1)	**22**(2/0/1)	**24**(2/0/1)	**28**(2/0/1)	**26**(3/0/1)	**28**(3/0/1)	**29**(3/0/1)
([d$_{16}$,B],I,d$_{32}$)	**21**(2/0/1)	**23**(2/0/1)	**22**(2/0/1)	**24**(2/0/1)	**28**(2/0/1)	**26**(3/0/1)	**28**(3/0/1)	**29**(3/0/1)
([d$_{32}$,B],I)	**23**(2/0/1)	**25**(2/0/1)	**24**(2/0/1)	**26**(2/0/1)	**30**(2/0/1)	**28**(3/0/1)	**30**(3/0/1)	**31**(3/0/1)
([d$_{32}$,B],I,d$_{16}$)	**25**(2/0/1)	**27**(2/0/1)	**26**(2/0/1)	**28**(2/0/1)	**32**(2/0/1)	**30**(3/0/1)	**32**(3/0/1)	**33**(3/0/1)
([d$_{32}$,B],I,d$_{32}$)	**25**(2/0/1)	**27**(2/0/1)	**26**(2/0/1)	**28**(2/0/1)	**32**(2/0/1)	**30**(3/0/1)	**32**(3/0/1)	**33**(3/0/1)

9.3 POWER AND GROUND CONSIDERATIONS

The MC68020/EC020 is fabricated in Motorola's advanced HCMOS process and is capable of operating at clock frequencies of up to 25 MHz. While the use of CMOS for a device containing such a large number of transistors allows significantly reduced power consumption compared to an equivalent NMOS circuit, the high clock speed makes the characteristics of power supplied to the device very important. The power supply must be able to furnish large amounts of instantaneous current when the MC68020/EC020 performs certain operations, and it must remain within the rated specification at all times. To meet these requirements, more detailed attention must be given to the power supply connection to the MC68020/EC020 than is required for NMOS devices operating at slower clock rates.

To reduce the amount of noise in the power supply connected to the MC68020/EC020 and to provide for the instantaneous current requirements, common capacitive decoupling techniques should be observed. While there is no recommended layout for this capacitive decoupling, it is essential that the inductance and distance between these devices and the MC68020/EC020 be minimized to provide sufficiently fast response time to satisfy momentary current demands and to maintain a constant supply voltage. It is suggested that high-frequency, high-quality capacitors be placed as close to the MC68020/EC020 as possible. Table 9-2 lists the V_{CC} and GND pin assignments for the MC68EC020 PPGA (RP suffix) package. Table 9-3 lists the V_{CC} and GND pin assignments for the MC68EC020 PQFP (FG suffix) package. Refer to **Section 11 Ordering Information and Mechanical Data** for the V_{CC} and GND pin assignments for the MC68020 packages. When assigning capacitors to the V_{CC} and GND pins, the noisier pins (address and data buses) should be heavily decoupled from the internal logic pins. Typical decoupling practices include a high-frequency, high-quality capacitor to decouple every device on the printed circuit board; however, due to the power requirements and drive capability of the MC68020/EC020, each V_{CC} pin should be decoupled with an individual capacitor. Motorola recommends using a capacitor in the range of 0.01 µF to 0.1 µF on each V_{CC} pin on each device to provide filtering for most frequencies prevalent in a digital system. In addition to the individual decoupling, several bulk decoupling capacitors should be placed onto the printed circuit board with typical values in the range of 33 µF to 330 µF. When power and ground planes are used with an adequate number of high-frequency, high-quality capacitors, the system noise will be reduced to the required levels, and the MC68020/EC020 will function properly. Similar decoupling techniques should also be observed for other VLSI devices in the system.

In addition to the capacitive decoupling of the power supply, care must be taken to ensure a low-impedance connection between all MC68020/EC020 V_{CC} and GND pins and the system power supply. A solid power supply connection from the power and ground planes to the MC68020/EC020 V_{CC} and GND pins, respectively, will meet this requirement. Failure to provide connections of sufficient quality between the MC68020/EC020 power pins and the system power supplies will result in increased assertion delays for external signals, decreased voltage noise margins, increased system noise, and possible errors in MC68020/EC020 internal logic.

Figure 9-9. Access Time Computation Diagram

All module descriptor types $10–$1F are reserved for user definition and cause a format error exception. This provides the user with a means of disabling any module by setting a single bit in its descriptor without loss of any descriptor information.

If the called module does not wish the module data area pointer to be loaded into a register, the module entry word can select register A7, and the loaded value will be overwritten with the correction stack pointer value after the module stack frame is created and filled.

9.7.2 Module Stack Frame

Figure 9-12 illustrates the format of the module stack frame. This frame is constructed by the CALLM instruction and is removed by the RTM instruction. The first and second long words contain control information passed by the CALLM instruction to the RTM instruction. The module descriptor pointer contains the address of the descriptor used during the module call. All other locations contain information to be restored on return to the calling module.

The PC is the saved address of the instruction following the CALLM instruction. The opt and type fields, which specify the argument options and type of module stack frame, are copied to the frame from the module descriptor by the CALLM instruction; the RTM instruction will cause a format error if the opt and type fields do not have recognizable values. The access level is the saved access control information, which is saved from external hardware by the CALLM instruction and restored by the RTM instruction. The argument count field is set by the CALLM instruction and is used by the RTM instruction to remove arguments from the stack of the calling module. The contents of the CCR are saved by the CALLM instruction and restored by the RTM instruction. The saved stack pointer field contains the value of the stack pointer when the CALLM instruction started execution, and that value is restored by RTM. The saved module data area pointer field contains the saved value of the module data area pointer register from the calling module.

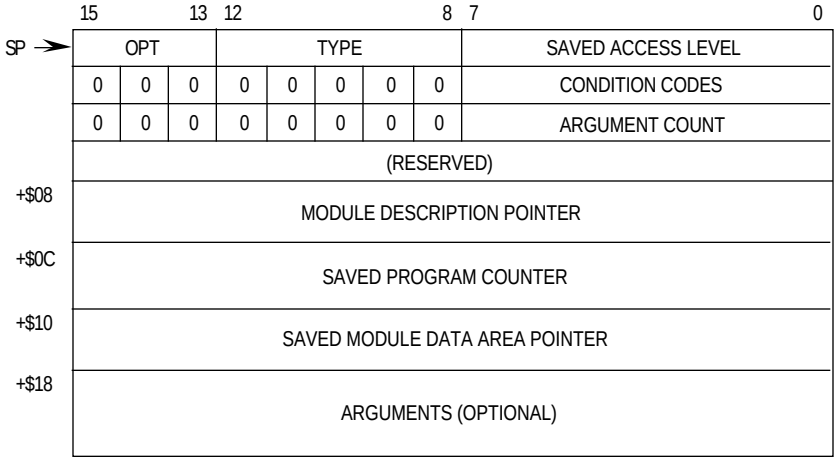
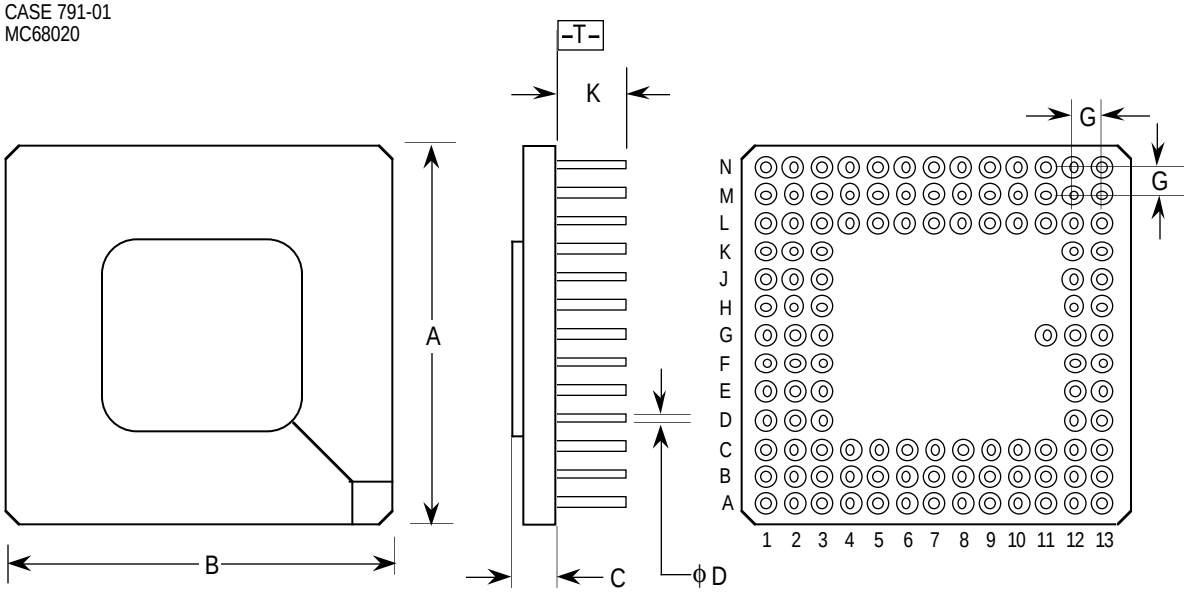


Figure 9-12. Module Call Stack Frame

11.2.2 MC68020 RC Suffix—Package Dimensions

RC SUFFIX
CASE 791-01
MC68020



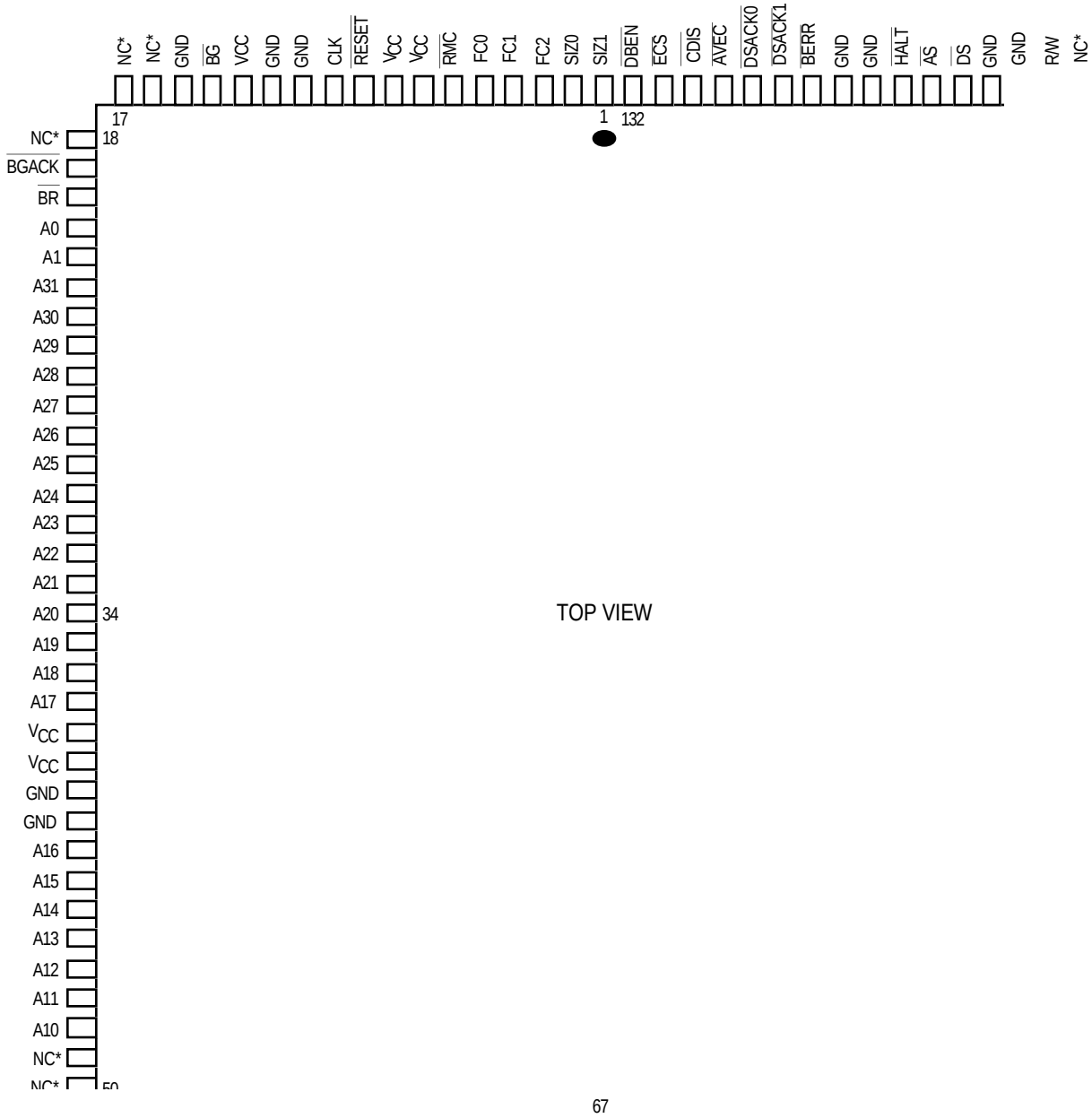
DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	34.04	35.05	1.340	1.380
B	34.04	35.05	1.340	1.380
C	2.54	3.81	0.100	0.150
D	0.43	0.55	0.017	0.022
G	2.54 BSC		0.100 BSC	
K	4.32	4.95	0.170	0.195

NOTES:
1. **A** AND **B** ARE DATUMS AND **T** IS A DATUM SURFACE.
2. POSITIONAL TOLERANCE FOR LEADS (114 PLACES).

⌖	0.13 (0.005) Ⓜ	T	A Ⓢ	B Ⓢ

3. DIMENSIONING AND TOLERANCING PER Y14.5M,1982.
4. CONTROLLING DIMENSION: INCH.

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