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.

In contrast, standard peripheral hardware is generally accessed through interface registers mapped into the memory space of the main processor. To use the services provided by the peripheral, the programmer accesses the peripheral registers with standard processor instructions. While a peripheral could conceivably provide capabilities equivalent to a coprocessor for many applications, the programmer must implement the communication protocol between the main processor and the peripheral necessary to use the peripheral hardware.

The communication protocol defined for the M68000 coprocessor interface is described in **7.2 Coprocessor Instruction Types**. The algorithms that implement the M68000 coprocessor interface are provided in the microcode of the MC68020/EC020 and are completely transparent to the MC68020/EC020 programming model. For example, floating-point operations are not implemented in the MC68020/EC020 hardware. In a system utilizing both the MC68020/EC020 and the MC68881 or MC68882 floating-point coprocessor, a programmer can use any of the instructions defined for the coprocessor without knowing that the actual computation is performed by the MC68881 or MC68882 hardware.

7.1.1 Interface Features

The M68000 coprocessor interface design incorporates a number of flexible capabilities. The physical coprocessor interface uses the main processor external bus, which simplifies the interface since no special-purpose signals are involved. With the MC68020/EC020, a coprocessor uses the asynchronous bus transfer protocol. Since standard bus cycles transfer information between the main processor and the coprocessor, the coprocessor can be implemented in whatever technology is available to the coprocessor designer. A coprocessor can be implemented as a VLSI device, as a separate system board, or even as a separate computer system.

Since the main processor and a M68000 coprocessor can communicate using the asynchronous bus, they can operate at different clock frequencies. The system designer can choose the speeds of a main processor and coprocessor that provide the optimum performance for a given system. Both the MC68881 and MC68882 floating-point coprocessors use the asynchronous bus handshake protocol.

The M68000 coprocessor interface also facilitates the design of coprocessors. The coprocessor designer must only conform to the coprocessor interface and does not need an extensive knowledge of the architecture of the main processor. Also, the main processor can operate with a coprocessor without having explicit provisions made in the main processor for the capabilities of that coprocessor. This type of interface provides a great deal of freedom in the implementation of a given coprocessor.

7.1.2 Concurrent Operation Support

The programming model for the M68000 family of microprocessors is based on sequential, nonconcurrent instruction execution, which implies that the instructions in a given sequence must appear to be executed in the order in which they occur. To maintain a uniform programming model, any coprocessor extensions should also maintain the

model of sequential, nonconcurrent instruction execution at the user level. Consequently, the programmer can assume that the images of registers and memory affected by a given instruction have been updated when the next instruction in the sequence accessing these registers or memory locations is executed.

The M68000 coprocessor interface provides full support of all operations necessary for nonconcurrent operation of the main processor and its associated coprocessors. Although the M68000 coprocessor interface allows concurrency in coprocessor execution, the coprocessor designer is responsible for implementing this concurrency while maintaining a programming model based on sequential nonconcurrent instruction execution.

For example, if the coprocessor determines that instruction B does not use or alter resources to be altered or used by instruction A, instruction B can be executed concurrently (if the execution hardware is also available). Thus, the required instruction interdependencies and sequences of the program are always respected. The MC68882 coprocessor offers concurrent instruction execution; whereas, the MC68881 coprocessor does not. However, the MC68020/EC020 can execute instructions concurrently with coprocessor instruction execution in the MC68881.

7.1.3 Coprocessor Instruction Format

The instruction set for a given coprocessor is defined by the design of that coprocessor. When a coprocessor instruction is encountered in the main processor instruction stream, the MC68020/EC020 hardware initiates communication with the coprocessor and coordinates any interaction necessary to execute the instruction with the coprocessor. A programmer needs to know only the instruction set and register set defined by the coprocessor to use the functions provided by the coprocessor hardware.

The instruction set of an M68000 coprocessor uses a subset of the F-line operation words in the M68000 instruction set. The operation word is the first word of any M68000 family instruction. The F-line operation word contains ones in bits 15–12 (refer to Figure 7-1); the remaining bits are coprocessor and instruction dependent. The F-line operation word may be followed by as many extension words as are required to provide additional information necessary for the execution of the coprocessor instruction.

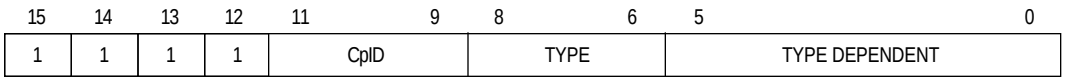


Figure 7-1. F-Line Coprocessor Instruction Operation Word

As shown in Figure 7-1, bits 11–9 of the F-line operation word encode the coprocessor identification (CpID) field. The MC68020/EC020 uses the CpID field to indicate the coprocessor to which the instruction applies. F-line operation words, in which the CpID is zero, are not coprocessor instructions for the MC68020/EC020. Instructions with a CpID of zero and a nonzero type field are unimplemented instructions that cause the

MC68020/EC020 to begin exception processing. The MC68020/EC020 never generates coprocessor interface bus cycles with the CpID equal to zero (except via the MOVES instruction).

CpID codes of 000–101 are reserved for current and future Motorola coprocessors, and CpID codes of 110–111 are reserved for user-defined coprocessors. The Motorola CpID code of 001 designates the MC68881 or MC68882 floating-point coprocessor. By default, Motorola assemblers will use a CpID code of 001 when generating the instruction operation codes for the MC68881 or MC68882.

The encoding of bits 8–0 of the coprocessor instruction operation word is dependent on the particular instruction being implemented (refer to **7.2 Coprocessor Instruction Types**).

7.1.4 Coprocessor System Interface

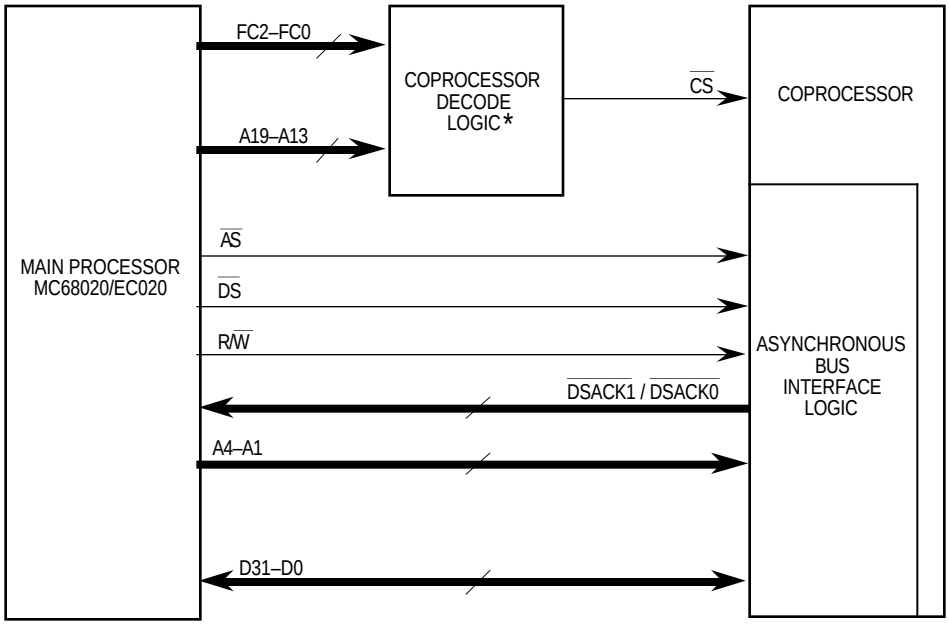
The communication protocol between the main processor and coprocessor necessary to execute a coprocessor instruction uses a group of interface registers, CIRs, resident within the coprocessor. By accessing one of the CIRs, the MC68020/EC020 hardware initiates coprocessor instructions. The coprocessor uses a set of response primitive codes and format codes defined for the M68000 coprocessor interface to communicate status and service requests to the main processor through these registers. The CIRs are also used to pass operands between the main processor and the coprocessor. The CIR set, response primitives, and format codes are discussed in **7.3 Coprocessor Interface Register Set** and **7.4 Coprocessor Response Primitives**.

7.1.4.1 COPROCESSOR CLASSIFICATION. M68000 coprocessors can be classified into two categories depending on their bus interface capabilities. The first category, non-DMA coprocessors, consists of coprocessors that always operate as bus slaves. The second category, DMA coprocessors, consists of coprocessors that operate as bus slaves while communicating with the main processor across the coprocessor interface. These coprocessors also have the ability to operate as bus masters, directly controlling the system bus.

If the operation of a coprocessor does not require a large portion of the available bus bandwidth or has special requirements not directly satisfied by the main processor, that coprocessor can be efficiently implemented as a non-DMA coprocessor. Since non-DMA coprocessors always operate as bus slaves, all external bus-related functions that the coprocessor requires are performed by the main processor. The main processor transfers operands from the coprocessor by reading the operand from the appropriate CIR and then writing the operand to a specified effective address with the appropriate address space specified on the FC2–FC0. Likewise, the main processor transfers operands to the coprocessor by reading the operand from a specified effective address (and address space) and then writing that operand to the appropriate CIR using the coprocessor interface. The bus interface circuitry of a coprocessor operating as a bus slave is not as complex as that of a device operating as a bus master.

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

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



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

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

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

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

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.

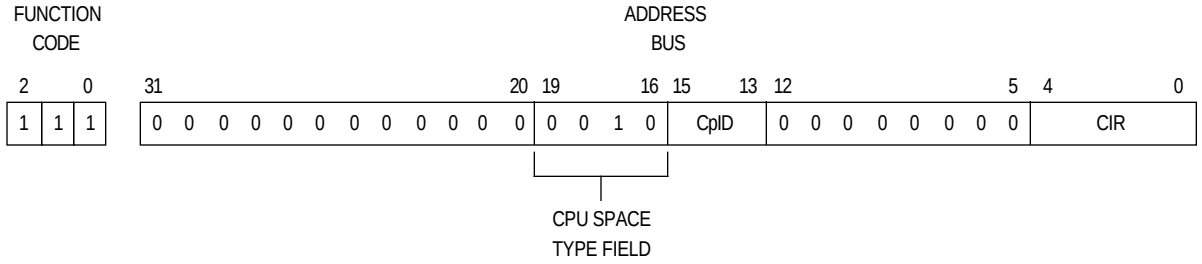


Figure 7-3. MC68020/EC020 CPU Space Address Encodings

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**.

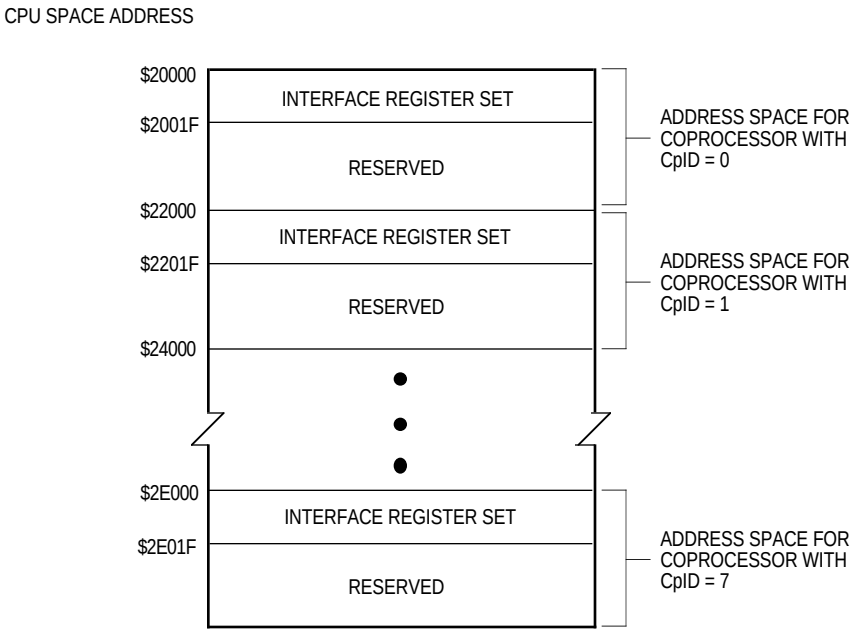


Figure 7-4. Coprocessor Address Map in MC68020/EC020 CPU Space

	31 — 16	15 — 0
$00	RESPONSE	CONTROL
$04	SAVE	RESTORE
$08	OPERATION WORD	COMMAND
$0C	(RESERVED)	CONDITION
$10	OPERAND	
$14	REGISTER SELECT	(RESERVED)
$18	INSTRUCTION ADDRESS	
$1C	OPERAND ADDRESS	

Figure 7-5. Coprocessor Interface Register Set Map

7.2 COPROCESSOR INSTRUCTION TYPES

The M68000 coprocessor interface supports four categories of coprocessor instructions: general, conditional, context save, and context restore. The category name indicates the type of operations provided by the coprocessor instructions in the category. The instruction category also determines the CIR accessed by the MC68020/EC020 to initiate instruction and communication protocols between the main processor and the coprocessor necessary for instruction execution.

During the execution of instructions in the general or conditional categories, the coprocessor uses the set of coprocessor response primitive codes defined for the M68000 coprocessor interface to request services from and indicate status to the main processor. During the execution of the instructions in the context save and context

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.

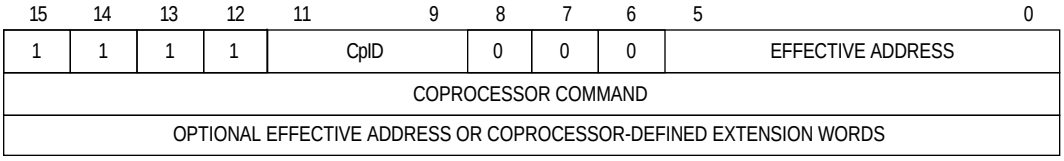


Figure 7-6. Coprocessor General Instruction Format (cpGEN)

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.

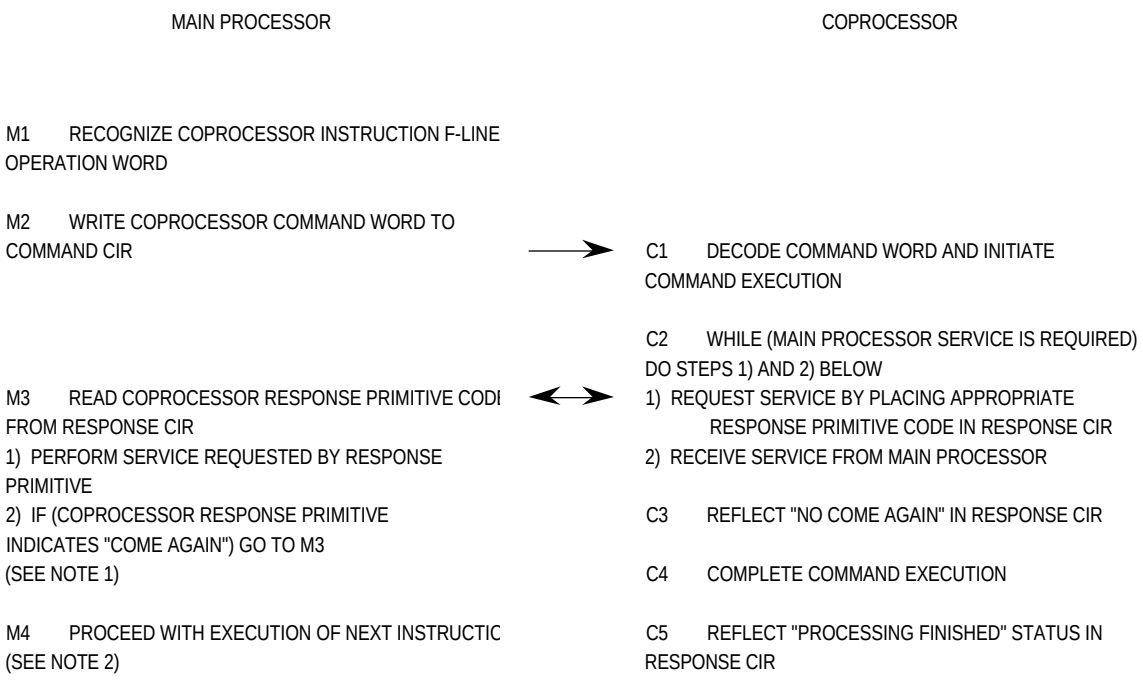
An instruction in the coprocessor general instruction category optionally includes a number of extension words following the coprocessor command word. These words can provide additional information required for the coprocessor instruction. For example, if

the coprocessor requests that the MC68020/EC020 calculate an effective address during coprocessor instruction execution, information required for the calculation must be included in the instruction format as effective address extension words.

7.2.1.2 PROTOCOL. The execution of a cpGEN instruction follows the protocol shown in Figure 7-7. The main processor initiates communication with the coprocessor by writing the instruction command word to the command CIR. The coprocessor decodes the command word to begin processing the cpGEN instruction. Coprocessor design determines the interpretation of the coprocessor command word; the MC68020/EC020 does not attempt to decode it.

While the coprocessor is executing an instruction, it requests any required services from and communicates status to the main processor by placing coprocessor response primitive codes in the response CIR. After writing to the command CIR, the main processor reads the response CIR and responds appropriately. When the coprocessor has completed the execution of an instruction or no longer needs the services of the main processor to execute the instruction, it provides a response to release the main processor. The main processor can then execute the next instruction in the instruction stream. However, if a trace exception is pending, the MC68020/EC020 does not terminate communication with the coprocessor until the coprocessor indicates that it has completed all processing associated with the cpGEN instruction (refer to **7.5.2.5 Trace Exceptions**).

The coprocessor interface protocol shown in Figure 7-7 allows the coprocessor to define the operation of each coprocessor general type instruction. That is, the main processor initiates the instruction execution by writing the instruction command word to the command CIR and by reading the response CIR to determine its next action. The execution of the coprocessor instruction is then defined by the internal operation of the coprocessor and by its use of response primitives to request services from the main processor. This instruction protocol allows a wide range of operations to be implemented in the general instruction category.



NOTES: 1. "Come Again" indicates that further service of the main processor is being requested by the coprocessor.
2. The next instruction should be the operation word pointed to by the ScanPC at this point. The operation of the MC68020/EC020 ScanPC is discussed in **7.4.1 ScanPC**.

Figure 7-7. Coprocessor Interface Protocol for General Category Instructions

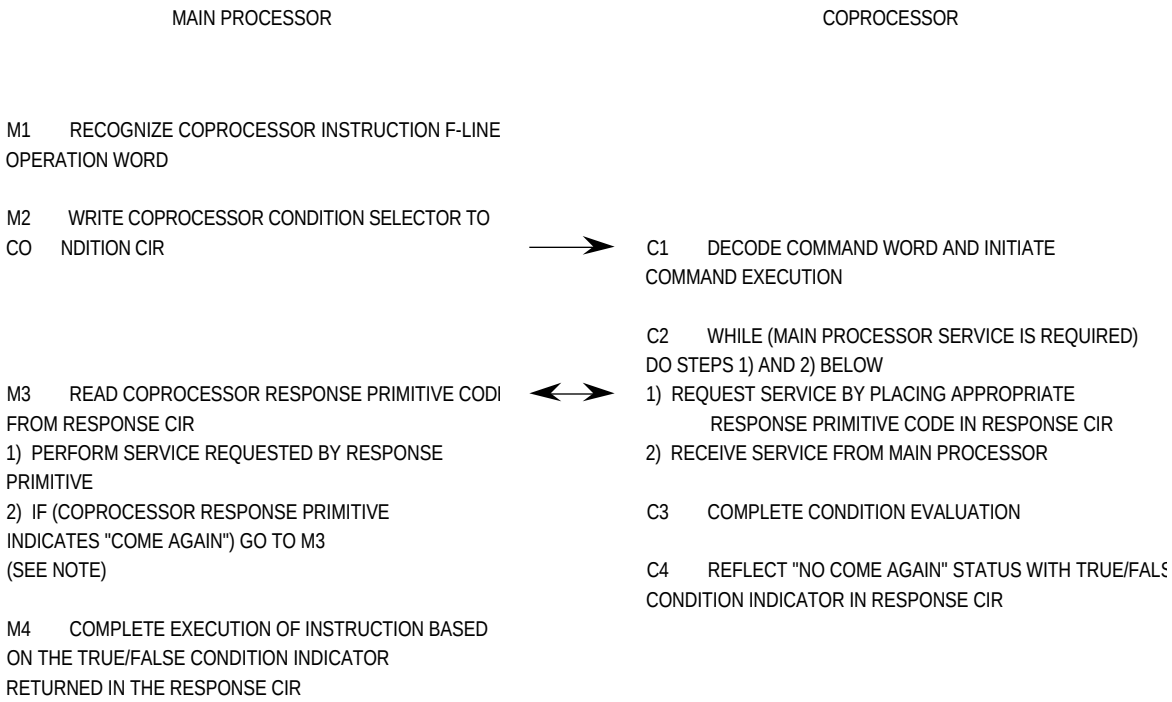
7.2.2 Coprocessor Conditional Instructions

The conditional instruction category provides program control based on the operations of the coprocessor. The coprocessor evaluates a condition and returns a true/false indicator to the main processor. The main processor completes the execution of the instruction based on this true/false condition indicator.

The implementation of instructions in the conditional category promotes efficient use of both the main processor and the coprocessor hardware. The condition specified for the instruction is related to the coprocessor operation and is therefore evaluated by the coprocessor. However, the instruction completion following the condition evaluation is directly related to the operation of the main processor. The main processor performs the change of flow, the setting of a byte, or the TRAP operation, since its architecture explicitly implements these operations for its instruction set.

Figure 7-8 shows the protocol for a conditional category coprocessor instruction. The main processor initiates execution of an instruction in this category by writing a condition selector to the condition CIR. The coprocessor decodes the condition selector to determine the condition to evaluate. The coprocessor can use response primitives to request that the main processor provide services required for the condition evaluation.

After evaluating the condition, the coprocessor returns a true/false indicator to the main processor by placing a null primitive (refer to **7.4.4 Null Primitive**) in the response CIR. The main processor completes the coprocessor instruction execution when it receives the condition indicator from the coprocessor.



NOTE: All coprocessor response primitives, except the Null primitive, that allow the "Come Again" primitive attribute must indicate "Come Again" when used during the execution of a conditional category instruction. If a "Come Again" attribute is not indicated in one of these primitives, the main processor will initiate protocol violation exception processing (see 7.5.2.1 Protocol Violations).

Figure 7-8. Coprocessor Interface Protocol for Conditional Category Instructions

7.2.2.1 BRANCH ON COPROCESSOR CONDITION INSTRUCTION. The conditional instruction category includes two formats of the M68000 family branch instruction. These instructions branch on conditions related to the coprocessor operation. They execute similarly to the conditional branch instructions provided in the M68000 family instruction set.

7.2.2.1.1 Format. Figure 7-9 shows the format of the branch on coprocessor condition instruction that provides a word-length displacement. Figure 7-10 shows the format of this instruction that includes a long-word displacement.

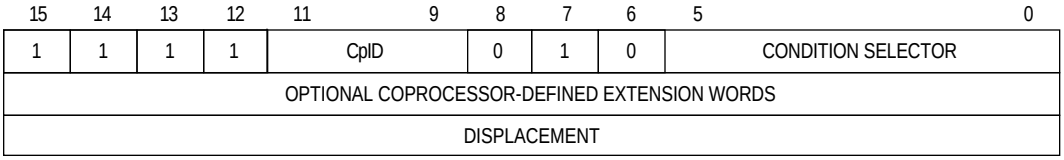


Figure 7-9. Branch on Coprocessor Condition Instruction Format (cpBcc.W)

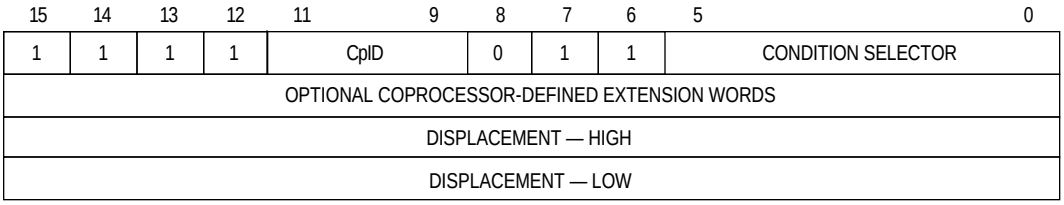


Figure 7-10. Branch on Coprocessor Condition Instruction Format (cpBcc.L)

The first word of the branch on coprocessor condition instruction is the F-line operation word. Bits 15–12 = 1111 and bits 11–9 contain the CpID code of the coprocessor that is to evaluate the condition. The value in bits 8–6 identifies either the word or the long-word displacement format of the branch instruction, which is specified by the cpBcc.W or cpBcc.L mnemonic, respectively. Bits 5–0 of the F-line operation word contain the coprocessor condition selector field. The MC68020/EC020 writes the entire operation word to the condition CIR to initiate execution of the branch instruction by the coprocessor. The coprocessor uses bits 5–0 to determine which condition to evaluate.

If the coprocessor requires additional information to evaluate the condition, the branch instruction format can include this information in extension words. Following the F-line operation word, the number of extension words is determined by the coprocessor design. The final word(s) of the cpBcc instruction format contains the displacement used by the main processor to calculate the destination address when the branch is taken.

7.2.2.1.2 Protocol. Figure 7-8 shows the protocol for the cpBcc.L and cpBcc.W instructions. The main processor initiates the instruction by writing the F-line operation word to the condition CIR to transfer the condition selector to the coprocessor. 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. If the coprocessor returns the false condition indicator, the main processor executes the next instruction in the instruction stream. If the coprocessor returns the true condition indicator, the main processor adds the displacement to the MC68020/EC020 scanPC (refer to **7.4.1 ScanPC**) to determine the address of the next instruction for the main processor to execute. The scanPC must be pointing to the location of the first word of the displacement in the instruction stream when the address is calculated. The displacement is a twos-complement integer that can be either a 16-bit word or a 32-bit long word. The main processor sign-extends the 16-bit displacement to a long-word value for the destination address calculation.

7.2.2.2 SET ON COPROCESSOR CONDITION INSTRUCTION. The set on coprocessor condition instruction sets or resets a flag (a data alterable byte) according to a condition evaluated by the coprocessor. The operation of this instruction type is similar to the operation of the Scc instruction in the M68000 family instruction set. Although the Scc instruction and the cpScc instruction do not explicitly cause a change of program flow, they are often used to set flags that control program flow.

7.2.2.2.1 Format. Figure 7-11 shows the format of the set on coprocessor condition instruction, denoted by the cpScc mnemonic.

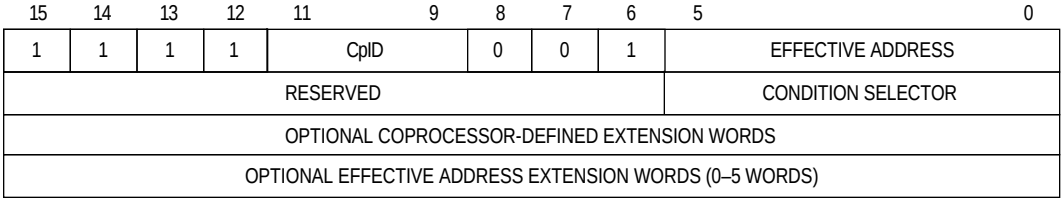


Figure 7-11. Set on Coprocessor Condition Instruction Format (cpScc)

The first word of the cpScc instruction, the F-line operation word, contains the CpID field in bits 11–9 and 001 in bits 8–6 to identify the cpScc instruction. Bits 5–0 of the F-line operation word are used to encode an M68000 family effective addressing mode (refer to M68000PM/AD, *M68000 Family Programmer's Reference Manual*).

The second word of the cpScc instruction format contains the coprocessor condition selector field in bits 5–0. Bits 15–6 of this word are reserved by Motorola and should be zero to ensure compatibility with future M68000 products. This word is written to the condition CIR to initiate execution of the cpScc instruction.

If the coprocessor requires additional information to evaluate the condition, the instruction can include extension words to provide this information. The number of these extension words, which follow the word containing the coprocessor condition selector field, is determined by the coprocessor design.

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.

15	14	13	12	11 9	8	7	6	5	4	3	2 0
1	1	1	1	CpID	0	0	1	0	0	1	REGISTER
(RESERVED)								CONDITION SELECTOR			
OPTIONAL COPROCESSOR-DEFINED EXTENSION WORDS											
DISPLACEMENT											

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.

If the coprocessor requires additional information to evaluate the condition, the cpDBcc instruction can include this information in extension words. These extension words follow the word containing the coprocessor condition selector field in the cpDBcc instruction format.

The last word of the instruction contains the displacement for the cpDBcc instruction. This displacement is a twos-complement 16-bit value that is sign-extended to long-word size when it is used in a destination address calculation.

7.2.2.3.2 Protocol. Figure 7-8 shows the protocol for the cpDBcc instructions. The MC68020/EC020 transfers the condition selector to the coprocessor by writing the word following the operation word to the condition CIR. The main processor then reads the response CIR to determine its next action. The coprocessor can use a response primitive to request any services necessary to evaluate the condition. If the coprocessor returns the true condition indicator, the main processor executes the next instruction in the instruction stream. If the coprocessor returns the false condition indicator, the main processor decrements the low-order word of the register specified by bits 2–0 of the F-line operation word. If this register contains minus one (–1) after being decremented, the main processor executes the next instruction in the instruction stream. If the register does not contain minus one (–1) after being decremented, the main processor branches to the destination address to continue instruction execution.

The MC68020/EC020 adds the displacement to the scanPC (refer to **7.4.1 ScanPC**) to determine the address of the next instruction. The scanPC must point to the 16-bit displacement in the instruction stream when the destination address is calculated.

7.2.2.4 TRAP ON COPROCESSOR CONDITION INSTRUCTION. The trap on coprocessor condition instruction allows the programmer to initiate exception processing based on conditions related to the coprocessor operation.

7.2.2.4.1 Format. Figure 7-13 shows the format of the trap on coprocessor condition instruction, denoted by the cpTRAPcc mnemonic.

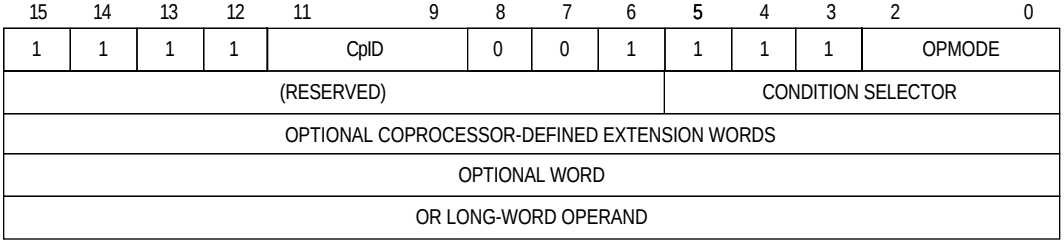


Figure 7-13. Trap on Coprocessor Condition Instruction Format (cpTRAPcc)

The first word of the cpTRAPcc instruction, the F-line operation word contains the CpID field in bits 11–9 and 001111 in bits 8–3 to identify the cpTRAPcc instruction. Bits 2–0 of the cpTRAPcc F-line operation word specify the opmode, which selects the instruction format. The instruction format can include zero, one, or two operand words.

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

If the coprocessor requires additional information to evaluate a condition, the instruction can include this information in extension words. These extension words follow the word containing the coprocessor condition selector field in the cpTRAPcc instruction format.

The operand words of the cpTRAPcc F-line operation word follow the coprocessor-defined extension words. These operand words are not explicitly used by the MC68020/EC020, but can be used to contain information referenced by the cpTRAPcc exception handling routines. The valid encodings for bits 2–0 of the F-line operation word and the corresponding numbers of operand words are listed in Table 7-1. Other encodings of these bits are invalid for the cpTRAPcc instruction.

Table 7-1. cpTRAPcc Opmode Encodings

Opmode	Operand Words in Instruction Format
010	One
011	Two
100	Zero

7.2.2.4.2 Protocol. Figure 7-8 shows the protocol for the cpTRAPcc instructions. The MC68020/EC020 transfers the condition selector to the coprocessor by writing the word following the 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 any services necessary to evaluate the condition. If the coprocessor returns the true condition indicator, the main processor initiates exception processing for the cpTRAPcc exception (refer to **7.5.2.4 cpTRAPcc Instruction Traps**). If the coprocessor returns the false condition indicator, the main processor executes the next instruction in the instruction stream.

7.2.3 Coprocessor Context Save and Restore Instructions

The coprocessor context save and context restore instruction categories in the M68000 coprocessor interface support multitasking programming environments. In a multitasking environment, the context of a coprocessor may need to be changed asynchronously with respect to the operation of that coprocessor. That is, the coprocessor may be interrupted at any point in the execution of an instruction in the general or conditional category to begin context change operations.

In contrast to the general and conditional instruction categories, the context save and context restore instruction categories do not use the coprocessor response primitives. A set of format codes defined by the M68000 coprocessor interface communicates status

information to the main processor during the execution of these instructions. These coprocessor format codes are discussed in detail in **7.2.3.2 Coprocessor Format Words**.

7.2.3.1 COPROCESSOR INTERNAL STATE FRAMES. The context save (cpSAVE) and context restore (cpRESTORE) instructions transfer an internal coprocessor state frame between memory and a coprocessor. This internal coprocessor state frame represents the state of coprocessor operations. Using the cpSAVE and cpRESTORE instructions, it is possible to interrupt coprocessor operation, save the context associated with the current operation, and initiate coprocessor operations with a new context.

A cpSAVE instruction stores a coprocessor internal state frame as a sequence of long-word entries in memory. Figure 7-14 shows the format of a coprocessor state frame. The format and length fields of the coprocessor state frame format comprise the format word. During execution of the cpSAVE instruction, the MC68020/EC020 calculates the state frame effective address from information in the operation word of the instruction and stores a format word at this effective address. The processor writes the long words that form the coprocessor state frame to descending memory addresses, beginning with the address specified by the sum of the effective address and the length field multiplied by four. During execution of the cpRESTORE instruction, the MC68020/EC020 reads the state frame from ascending addresses beginning with the effective address specified in the instruction operation word.

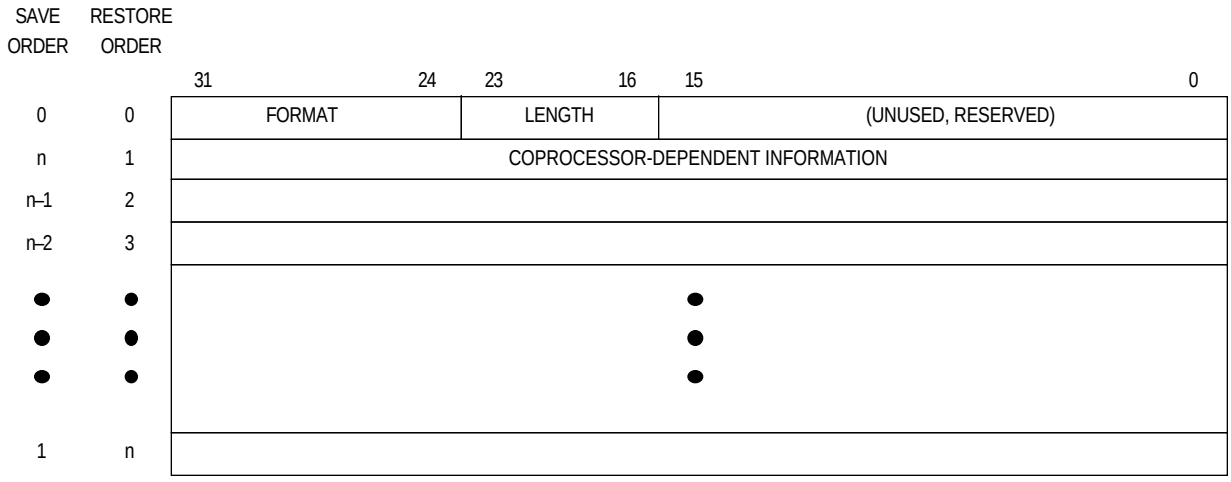


Figure 7-14. Coprocessor State Frame Format in Memory

The processor stores the coprocessor format word at the lowest address of the state frame in memory, and this word is the first word transferred for both the cpSAVE and cpRESTORE instructions. The word following the format word does not contain information relevant to the coprocessor state frame, but serves to keep the information in the state frame a multiple of four bytes in size. The number of entries following the format word (at higher addresses) is determined by the format word length for a given coprocessor state.

The information in a coprocessor state frame describes a context of operation for that coprocessor. This description of a coprocessor context includes the program invisible state information and, optionally, the program visible state information. The program invisible state information consists of any internal registers or status information that cannot be accessed by the program but is necessary for the coprocessor to continue its operation at the point of suspension. Program visible state information includes the contents of all registers that appear in the coprocessor programming model and that can be directly accessed using the coprocessor instruction set. The information saved by the cpSAVE instruction must include the program invisible state information. If cpGEN instructions are provided to save the program visible state of the coprocessor, the cpSAVE and cpRESTORE instructions should only transfer the program invisible state information to minimize interrupt latency during a save or restore operation.

7.2.3.2 COPROCESSOR FORMAT WORDS. The coprocessor communicates status information to the main processor during the execution of cpSAVE and cpRESTORE instructions using coprocessor format words. The format words defined for the M68000 coprocessor interface are listed in Table 7-2.

Table 7-2. Coprocessor Format Word Encodings

Format Code	Length	Meaning
$00	$xx	Empty/Reset
$01	$xx	Not Ready, Come Again
$02	$xx	Invalid Format
$03–$OF	$xx	Undefined, Reserved
$10–$FF	Length	Valid Format, Coprocessor Defined

xx—Don’t care

The upper byte of the coprocessor format word contains the code used to communicate coprocessor status information to the main processor. The MC68020/EC020 recognizes four types of format words: empty/reset, not ready, invalid format, and valid format. The MC68020/EC020 interprets the reserved format codes ($03–$0F) as invalid format words. The lower byte of the coprocessor format word specifies the size in bytes (which must be a multiple of four) of the coprocessor state frame. This value is only relevant when the code byte contains the valid format code (refer to **7.2.3.2.4 Valid Format Word**).

7.2.3.2.1 Empty/Reset Format Word. The coprocessor returns the empty/reset format code during a cpSAVE instruction to indicate that the coprocessor contains no user-specific information. That is, no coprocessor instructions have been executed since either a previous cpRESTORE of an empty/reset format code or the previous hardware reset. If the main processor reads the empty/reset format word from the save CIR during the initiation of a cpSAVE instruction, it stores the format word at the effective address specified in the cpSAVE instruction and executes the next 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

main processor reads an invalid format word from the save CIR, it writes the abort mask to the control CIR and initiates format error exception processing (refer to **7.5.1.5 Format Errors**).

7.2.3.2.4 Valid Format Word. When the main processor reads a valid format word from the save CIR during the cpSAVE instruction, it uses the length field to determine the size of the coprocessor state frame to save. The length field in the lower eight bits of a format word is relevant only in a valid format word. During the cpRESTORE instruction, the main processor uses the length field in the format word read from the effective address in the instruction to determine the size of the coprocessor state frame to restore.

The length field of a valid format word, representing the size of the coprocessor state frame, must contain a multiple of four. If the main processor detects a value that is not a multiple of four in a length field during the execution of a cpSAVE or cpRESTORE instruction, the main processor writes the abort mask (refer to **7.2.3.2.3 Invalid Format Word**) to the control CIR and initiates format error exception processing.

7.2.3.3 COPROCESSOR CONTEXT SAVE INSTRUCTION. The M68000 coprocessor context save instruction category consists of one instruction. The coprocessor context save instruction, denoted by the cpSAVE mnemonic, saves the context of a coprocessor dynamically without relation to the execution of coprocessor instructions in the general or conditional instruction categories. During the execution of a cpSAVE instruction, the coprocessor communicates status information to the main processor by using the coprocessor format codes.

7.2.3.3.1 Format. Figure 7-15 shows the format of the cpSAVE instruction. The first word of the instruction, the F-line operation word, contains the CpID code in bits 11–9 and an M68000 effective address code in bits 5–0. The effective address encoded in the cpSAVE instruction is the address at which the state frame associated with the current context of the coprocessor is saved in memory.

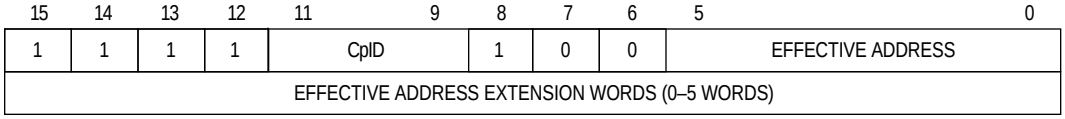


Figure 7-15. Coprocessor Context Save Instruction Format (cpSAVE)

The control alterable and predecrement addressing modes are valid for the cpSAVE instruction. Other addressing modes cause the MC68020/EC020 to initiate F-line emulator exception processing as described in **7.5.2.2 F-Line Emulator Exceptions**.

The instruction can include as many as five effective address extension words following the F-line operation word. These words contain any additional information required to calculate the effective address specified by bits 5–0 of the F-line operation word.

7.2.3.3.2 Protocol. Figure 7-16 shows the protocol for the coprocessor context save instruction. The main processor initiates execution of the cpSAVE instruction by reading the save CIR. Thus, the cpSAVE instruction is the only coprocessor instruction that begins by reading from a CIR. All other coprocessor instructions write to a CIR to initiate execution of the instruction by the coprocessor. The coprocessor communicates status information associated with the context save operation to the main processor by placing coprocessor format codes in the save CIR.

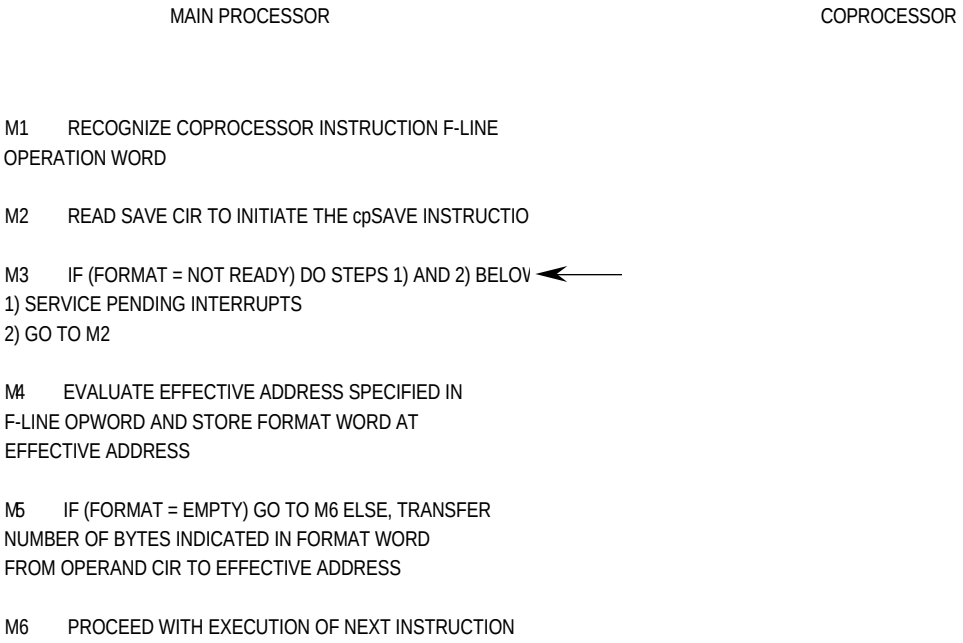


Figure 7-16. Coprocessor Context Save Instruction Protocol

If the coprocessor is not ready to suspend its current operation when the main processor reads the save CIR, it returns a not-ready format code. The main processor services any pending interrupts and then reads the save CIR again. After placing the not-ready format code in the save CIR, the coprocessor should either suspend or complete the instruction it is currently executing.

Once the coprocessor has suspended or completed the instruction it is executing, it places a format code representing the internal coprocessor state in the save CIR. When the main processor reads the save CIR, it transfers the format word to the effective address specified in the cpSAVE instruction. The lower byte of the coprocessor format word specifies the number of bytes of state information, not including the format word and associated null word, to be transferred from the coprocessor to the effective address specified. If the state information is not a multiple of four bytes in size, the MC68020/EC020 initiates format error exception processing (refer to **7.5.1.5 Format Errors**). The coprocessor and main processor coordinate the transfer of the internal state of the coprocessor using the operand CIR. The MC68020/EC020 completes the coprocessor context save by repeatedly reading the operand CIR and writing the

information obtained into memory until all the bytes specified in the coprocessor format word have been transferred. Following a cpSAVE instruction, the coprocessor should be in an idle state—that is, not executing any coprocessor instructions.

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

The MC68020/EC020 initiates format error exception processing if it reads an invalid format word (or a valid format word whose length field is not a multiple of four bytes) from the save CIR during the execution of a cpSAVE instruction (refer to **7.2.3.2.3 Invalid Format Word**). The MC68020/EC020 writes an abort mask (refer to **7.2.3.2.3 Invalid Format Word**) to the control CIR to abort the coprocessor instruction prior to beginning exception processing. Figure 7-16 does not include this case since a coprocessor usually returns either a not-ready or a valid format code in the context of the cpSAVE instruction. The coprocessor can return the invalid format word, however, if a cpSAVE is initiated while the coprocessor is executing a cpSAVE or cpRESTORE instruction and the coprocessor is unable to support the suspension of these two instructions.

7.2.3.4 COPROCESSOR CONTEXT RESTORE INSTRUCTION. The M68000 coprocessor context restore instruction category includes one instruction. The coprocessor context restore instruction, denoted by the cpRESTORE mnemonic, forces a coprocessor to terminate any current operations and to restore a former state. During execution of a cpRESTORE instruction, the coprocessor can communicate status information to the main processor by placing format codes in the restore CIR.

7.2.3.4.1 Format. Figure 7-17 shows the format of the cpRESTORE instruction.

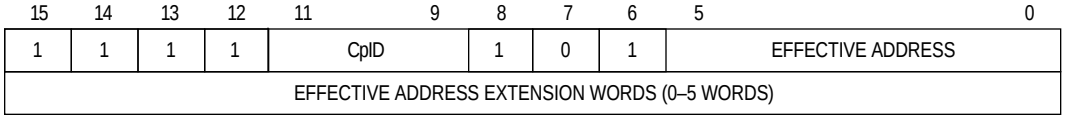


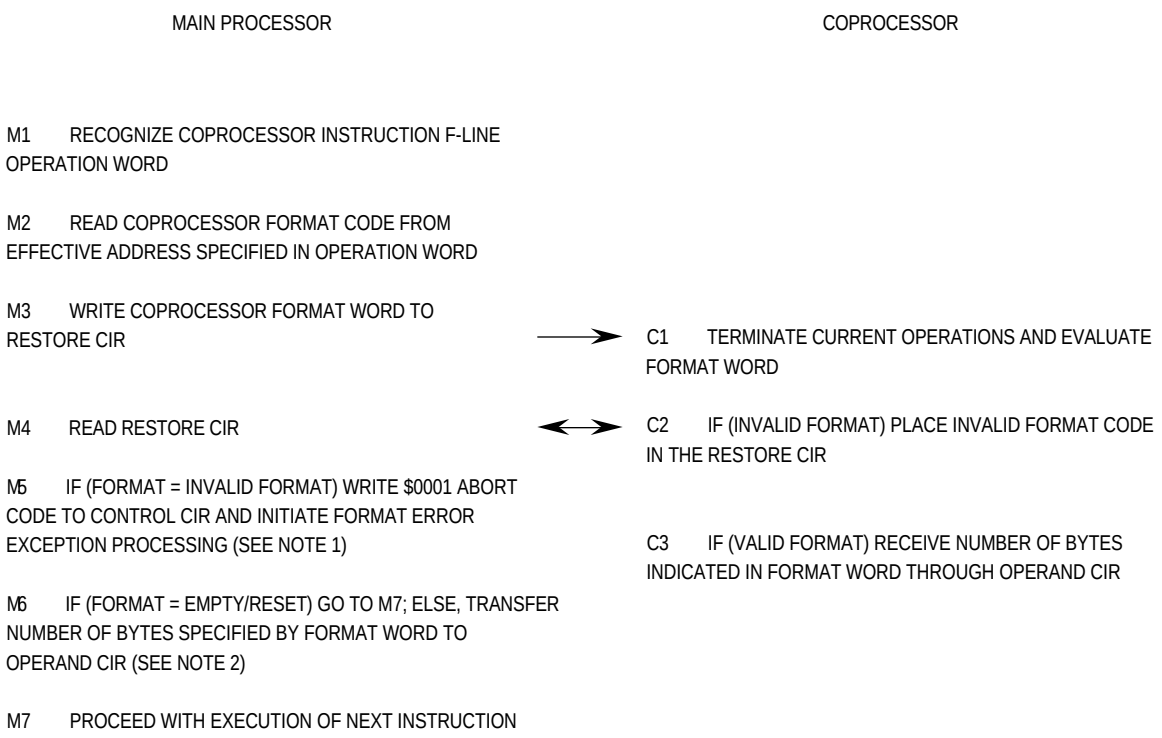
Figure 7-17. Coprocessor Context Restore Instruction Format (cpRESTORE)

The first word of the instruction, the F-line operation word, contains the CpID code in bits 11–9 and an M68000 effective addressing code in bits 5–0. The effective address encoded in the cpRESTORE instruction is the starting address in memory where the coprocessor context is stored. The effective address is that of the coprocessor format word that applies to the context to be restored to the coprocessor.

The instruction can include as many as five effective address extension words following the F-line operation word in the cpRESTORE instruction format. These words contain any additional information required to calculate the effective address specified by bits 5–0 of the F-line operation word.

All memory addressing modes except the predecrement addressing mode are valid. Invalid effective address encodings cause the MC68020/EC020 to initiate F-line emulator exception processing (refer to **7.5.2.2 F-Line Emulator Exceptions**).

7.2.3.4.2 Protocol. Figure 7-18 shows the protocol for the coprocessor context restore instruction. When the main processor executes a cpRESTORE instruction, it first reads the coprocessor format word from the effective address in the instruction. This format word contains a format code and a length field. During cpRESTORE operation, the main processor retains a copy of the length field to determine the number of bytes to be transferred to the coprocessor during the cpRESTORE operation and writes the format word to the restore CIR to initiate the coprocessor context restore.



NOTES: 1. See 7.6.1.5 Format Error.
2. The MC68020/EC020 uses the length field in the format word read during M2 to determine the number of bytes to read from memory and write to the operand CIR.

Figure 7-18. Coprocessor Context Restore Instruction Protocol

When the coprocessor receives the format word in the restore CIR, it must terminate any current operations and evaluate the format word. If the format word represents a valid coprocessor context as determined by the coprocessor design, the coprocessor returns the format word to the main processor through the restore CIR and prepares to receive the number of bytes specified in the format word through its operand CIR.

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

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

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

7.3 COPROCESSOR INTERFACE REGISTER SET

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

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

The following paragraphs contain detailed descriptions of the registers.

7.3.1 Response CIR

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

7.3.2 Control CIR

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

Figure 7-19. Control CIR Format

When the MC68020/EC020 receives one of the three take exception coprocessor response primitives, it acknowledges the primitive by setting the exception acknowledge bit (XA) in the control CIR. The MC68020/EC020 sets the abort bit (AB) in the control CIR to abort any coprocessor instruction in progress. (The 14 most significant bits of both masks are undefined.) The MC68020/EC020 aborts a coprocessor instruction when it detects one of the following exception conditions:

- An F-line emulator exception condition after reading a response primitive
- A privilege violation exception as it performs a supervisor check in response to a supervisor check primitive
- A format error exception when it receives an invalid format word or a valid format word that contains an invalid length

7.3.3 Save CIR

The coprocessor uses the 16-bit save CIR to communicate status and state frame format information to the main processor while executing a cpSAVE instruction. The main processor reads the save CIR to initiate execution of the cpSAVE instruction by the coprocessor. The offset from the base address of the CIR set for the save CIR is $04. Refer to **7.2.3.2 Coprocessor Format Words** for more information on the save CIR.

7.3.4 Restore CIR

The main processor initiates the cpRESTORE instruction by writing a coprocessor format word to the 16-bit restore register. During the execution of the cpRESTORE instruction, the coprocessor communicates status and state frame format information to the main processor through the restore CIR. The offset from the base address of the CIR set for the restore CIR is $06. Refer to **7.2.3.2 Coprocessor Format Words** for more information on the restore CIR.

7.3.5 Operation Word CIR

The main processor writes the F-line operation word of the instruction in progress to the 16-bit operation word CIR in response to a transfer operation word coprocessor response primitive (refer to **7.4.6 Transfer Operation Word Primitive**). The offset from the base address of the CIR set for the operation word CIR is $08.

7.3.6 Command CIR

The main processor initiates a coprocessor general category instruction by writing the instruction command word, which follows the instruction F-line operation word in the instruction stream, to the 16-bit command CIR. The offset from the base address of the CIR set for the command CIR is $0A.

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.

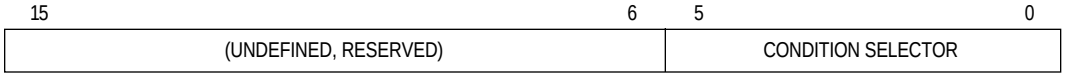


Figure 7-20. Condition CIR Format

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.

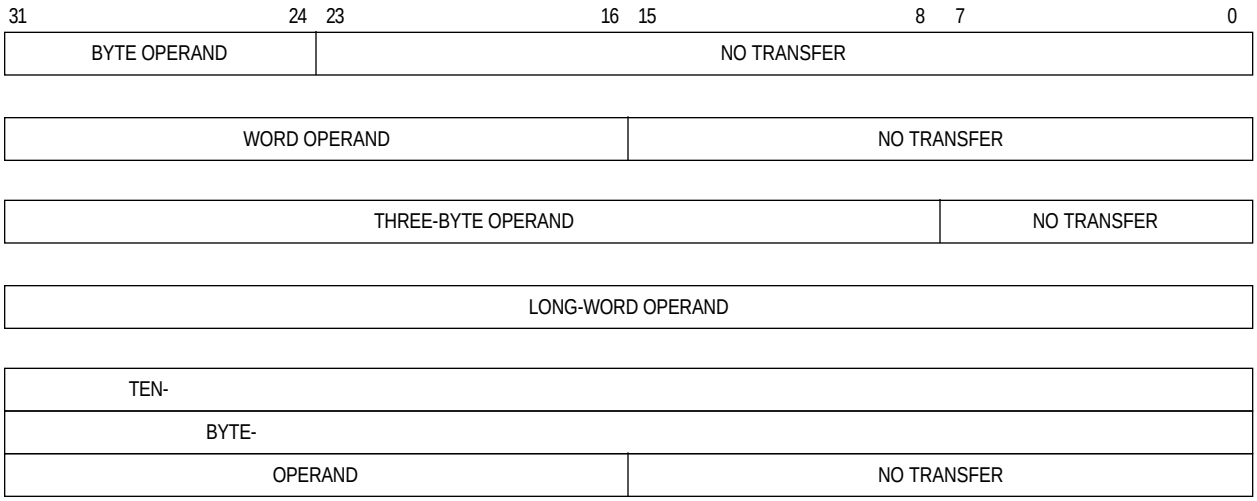


Figure 7-21. Operand Alignment for Operand CIR Accesses

7.3.9 Register Select CIR

When the coprocessor requests the transfer of one or more main processor registers or a group of coprocessor registers, the main processor reads the 16-bit register select CIR to identify the number or type of registers to be transferred. The offset from the base address of the CIR set for the register select CIR is $14. The format of this register depends on the primitive that is currently using it (refer to **7.4 Coprocessor Response Primitives**).

7.3.10 Instruction Address CIR

When the coprocessor requests the address of the instruction it is currently executing, the main processor transfers this address to the 32-bit instruction address CIR. Any transfer of the scanPC is also performed through the instruction address CIR (refer to **7.4.17 Transfer Status Register and ScanPC Primitive**). The offset from the base address of the CIR set for the instruction address CIR is $18.

7.3.11 Operand Address CIR

When a coprocessor requests an operand address transfer between the main processor and the coprocessor, the address is transferred through the 32-bit operand address CIR. The offset from the base address of the CIR set for the operand address CIR is $1C.

7.4 COPROCESSOR RESPONSE PRIMITIVES

The response primitives are primitive instructions that the coprocessor issues to the main processor during the execution of a coprocessor instruction. The coprocessor uses response primitives to communicate status information and service requests to the main processor. In response to an instruction command word written to the command CIR or a condition selector in the condition CIR, the coprocessor returns a response primitive in the response CIR. Within the general and conditional instruction categories, individual instructions are distinguished by the operation of the coprocessor hardware and by services specified by coprocessor response primitives and provided by the main processor.

Subsequent paragraphs, beginning with **7.4.2 Coprocessor Response Primitive General Format**, consist of detailed descriptions of the M68000 coprocessor response primitives supported by the MC68020/EC020. Any response primitive that the MC68020/EC020 does not recognize causes it to initiate protocol violation exception processing (refer to **7.5.2.1 Protocol Violations**). This processing of undefined primitives supports emulation of extensions to the M68000 coprocessor response primitive set by the protocol violation exception handler. Exception processing related to the coprocessor interface is discussed in **7.5 Exceptions**.

7.4.1 ScanPC

Several of the response primitives involve the scanPC, and many of them require the main processor to use it while performing services requested. These paragraphs describe the scanPC and its operation.

During the execution of a coprocessor instruction, the PC in the MC68020/EC020 contains the address of the F-line operation word of that instruction. A second register, called the scanPC, sequentially addresses the remaining words of the instruction.

If the main processor requires extension words to calculate an effective address or destination address of a branch operation, it uses the scanPC to address these extension words in the instruction stream. Also, if a coprocessor requests the transfer of extension words, the scanPC addresses the extension words during the transfer. As the processor references each word, it increments the scanPC to point to the next word in the instruction stream. When an instruction has completed, the processor transfers the value in the scanPC to the PC to address the operation word of the next instruction.

The value in the scanPC when the main processor reads the first response primitive after beginning to execute an instruction depends on the instruction being executed. For a cpGEN instruction, the scanPC points to the word following the coprocessor command word. For the cpBcc instructions, the scanPC points to the word following the instruction F-line operation word. For the cpScc, cpTRAPcc, and cpDBcc instructions, the scanPC points to the word following the coprocessor condition specifier word.

If a coprocessor implementation uses optional instruction extension words with a general or conditional instruction, the coprocessor must use these words consistently so that the scanPC is updated accordingly during the instruction execution. Specifically, during the execution of general category instructions, when the coprocessor terminates the instruction protocol, the MC68020/EC020 assumes that the scanPC is pointing to the operation word of the next instruction to be executed. During the execution of conditional category instructions, when the coprocessor terminates the instruction protocol, the MC68020/EC020 assumes that the scanPC is pointing to the word following the last of any coprocessor-defined extension words in the instruction format.

7.4.2 Coprocessor Response Primitive General Format

The M68000 coprocessor response primitives are encoded in a 16-bit word that is transferred to the main processor through the response CIR. Figure 7-22 shows the format of the coprocessor response primitives.

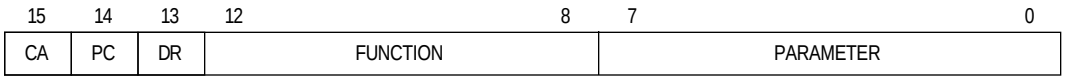


Figure 7-22. Coprocessor Response Primitive Format

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.

7.4.3 Busy Primitive

The busy response primitive causes the main processor to reinitiate a coprocessor instruction. This primitive applies to instructions in the general and conditional categories. Figure 7-23 shows the format of the busy primitive.

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

Figure 7-23. Busy Primitive Format

The busy primitive uses the PC bit as described in **7.4.2 Coprocessor Response Primitive General Format**.

Coprocessors that can operate concurrently with the main processor but cannot buffer write operations to their command or condition CIR use the busy primitive. A coprocessor may execute a cpGEN instruction concurrently with an instruction in the main processor. If the main processor attempts to initiate an instruction in the general or conditional instruction category while the coprocessor is executing a cpGEN instruction, the coprocessor can place the busy primitive in the response CIR. When the main processor reads this primitive, it services pending interrupts using a preinstruction exception stack frame (refer to Figure 7-41). The processor then restarts the general or conditional coprocessor instruction that it had attempted to initiate earlier.

The busy primitive should only be used in response to a write to the command or condition CIR. It should be the first primitive returned after the main processor attempts to initiate a general or conditional category instruction. In particular, the busy primitive should not be issued after program-visible resources have been altered by the instruction. (Program-visible resources include coprocessor and main processor program-visible registers and operands in memory, but not the scanPC.) The restart of an instruction after it has altered program-visible resources causes those resources to have inconsistent values when the processor reinitiates the instruction.

The MC68020/EC020 responds to the busy primitive differently in a special case that can occur during a breakpoint operation (refer to **Section 6 Exception Processing**). This special case occurs when a breakpoint acknowledge cycle initiates a coprocessor F-line instruction, the coprocessor returns the busy primitive in response to the instruction initiation, and an interrupt is pending. When these three conditions are met, the processor reexecutes the breakpoint acknowledge cycle after completion of interrupt exception processing. A design that uses a breakpoint to monitor the number of passes through a loop by incrementing or decrementing a counter may not work correctly under these conditions. This special case may cause several breakpoint acknowledge cycles to be executed during a single pass through a loop.

7.4.4 Null Primitive

The null coprocessor response primitive communicates coprocessor status information to the main processor. This primitive applies to instructions in the general and conditional categories. Figure 7-24 shows the format of the null primitive.

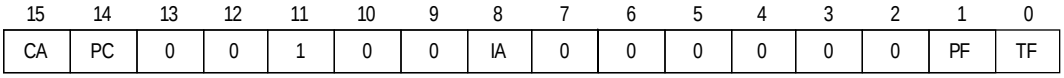


Figure 7-24. Null Primitive Format

The null primitive uses the CA and PC bits as described in **7.4.2 Coprocessor Response Primitive General Format**.

The IA bit specifies the interrupts allowed optional operation. This bit determines whether the MC68020/EC020 services pending interrupts prior to rereading the response CIR after receiving a null primitive. Interrupts are allowed when the IA bit is set.

The PF bit shows the processing-finished status of the coprocessor. That is, PF = 1 indicates that the coprocessor has completed all processing associated with an instruction.

The TF bit indicates the true/false condition during execution of a conditional category instruction. TF = 1 is the true condition specifier; TF = 0 is the false condition specifier. The TF bit is only relevant for null primitives with CA = 0 that are used by the coprocessor during the execution of a conditional instruction.

The MC68020/EC020 processes a null primitive with CA = 1 in the same manner whether executing a general or conditional category coprocessor instruction. If the coprocessor sets CA and IA in the null primitive, the main processor services pending interrupts using a midinstruction stack frame (refer to Figure 7-43) and reads the response CIR again. If the coprocessor sets CA and clears IA in the null primitive, the main processor reads the response CIR again without servicing any pending interrupts.

A null primitive with CA = 0 provides a condition evaluation indicator to the main processor during the execution of a conditional instruction and ends the dialogue between the main processor and coprocessor for that instruction. The main processor completes the execution of a conditional category coprocessor instruction when it receives the primitive. The PF bit is not relevant during conditional instruction execution since the primitive itself implies completion of processing.

Usually, when the main processor reads any primitive that does not have CA = 1 while executing a general category instruction, it terminates the dialogue between the main processor and coprocessor. If a trace exception is pending, however, the main processor does not terminate the instruction dialogue until it reads a null primitive with CA = 0 and PF = 1 from the response CIR (refer to **7.5.2.5 Trace Exceptions**). Thus, the main processor continues to read the response CIR until it receives a null primitive with CA = 0

and PF = 1, and then performs trace exception processing. When IA = 1, the main processor services pending interrupts before reading the response CIR again.

A coprocessor can be designed to execute a cpGEN instruction concurrently with the execution of main processor instructions and, also, buffer one write operation to either its command or condition CIR. This type of coprocessor issues a null primitive with CA = 1 when it is concurrently executing a cpGEN instruction, and the main processor initiates another general or conditional coprocessor instruction. This primitive indicates that the coprocessor is busy and the main processor should read the response CIR again without reinitiating the instruction. The IA bit of this null primitive usually should be set to minimize interrupt latency while the main processor is waiting for the coprocessor to complete the general category instruction.

Table 7-3 summarizes the encodings of the null primitive.

Table 7-3. Null Coprocessor Response Primitive Encodings

CA	PC	IA	PF	TF	General Instructions	Conditional Instructions
x	1	x	x	x	Pass Program Counter to Instruction Address CIR, Clear PC Bit, and Proceed with Operation Specified by CA, IA, PF, and TF Bits	Same as General Category
1	0	0	x	x	Reread Response CIR, Do Not Service Pending Interrupts	Same as General Category
1	0	1	x	x	Service Pending Interrupts and Reread the Response CIR	Same as General Category
0	0	0	0	c	If (Trace Pending) Reread Response CIR; Else, Execute Next Instruction	Main Processor Completes Instruction Execution Based on TF = c
0	0	1	0	c	If (Trace Pending) Service Pending Interrupts and Reread Response CIR; Else, Execute Next Instruction	Main Processor Completes Instruction Execution Based on TF = c
0	0	x	1	c	Coprocessor Instruction Completed; Service Pending Exceptions or Execute Next Instruction	Main Processor Completes Instruction Execution Based on TF = c.

x = Don't Care

c = 1 or 0 Depending on Coprocessor Condition Evaluation

7.4.5 Supervisor Check Primitive

The supervisor check primitive verifies that the main processor is operating in the supervisor privilege level while executing a coprocessor instruction. This primitive applies to instructions in the general and conditional coprocessor instruction categories. Figure 7-25 shows the format of the supervisor check primitive.

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

Figure 7-25. Supervisor Check Primitive Format

The supervisor check primitive uses the PC bit as described in **7.4.2 Coprocessor Response Primitive General Format**. Bit 15 is shown as one, but during execution of a general category instruction, this primitive performs the same operations, regardless of the value of bit 15. However, if this primitive is issued with bit 15 = 0 during a conditional category instruction, the main processor initiates protocol violation exception processing.

When the MC68020/EC020 reads the supervisor check primitive from the response CIR, it checks the value of the S-bit in the SR. If S = 0 (main processor operating at user privilege level), the main processor aborts the coprocessor instruction by writing an abort mask to the control CIR (refer to **7.3.2 Control CIR**). The main processor then initiates privilege violation exception processing (refer to **7.5.2.3 Privilege Violations**). If the main processor is at the supervisor privilege level when it receives this primitive, it reads the response CIR again.

The supervisor check primitive allows privileged instructions to be defined in the coprocessor general and conditional instruction categories. This primitive should be the first one issued by the coprocessor during the dialog for an instruction that is implemented as privileged.

7.4.6 Transfer Operation Word Primitive

The transfer operation word primitive requests a copy of the coprocessor instruction operation word for the coprocessor. This primitive applies to general and conditional category instructions. Figure 7-26 shows the format of the transfer operation word primitive.

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

Figure 7-26. Transfer Operation Word Primitive Format

The transfer operation word primitive uses the CA and PC bits as described in **7.4.2 Coprocessor Response Primitive General Format**. If this primitive is issued with CA = 0 during a conditional category instruction, the main processor initiates protocol violation exception processing.

When the main processor reads this primitive from the response CIR, it transfers the F-line operation word of the currently executing coprocessor instruction to the operation word CIR. The value of the scanPC is not affected by this primitive.

7.4.7 Transfer from Instruction Stream Primitive

The transfer from instruction stream primitive initiates transfers of operands from the instruction stream to the coprocessor. This primitive applies to general and conditional category instructions. Figure 7-27 shows the format of the transfer from instruction stream primitive.

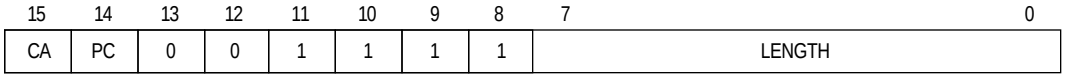


Figure 7-27. Transfer from Instruction Stream Primitive Format

The transfer from instruction stream primitive uses the CA and PC bits as described in **7.4.2 Coprocessor Response Primitive General Format**. If this primitive is issued with CA = 0 during a conditional category instruction, the main processor initiates protocol violation exception processing.

The length field of this primitive specifies the length, in bytes, of the operand to be transferred from the instruction stream to the coprocessor. The length must be an even number of bytes. If an odd length is specified, the main processor initiates protocol violation exception processing (refer to **7.5.2.1 Protocol Violations**).

This primitive transfers coprocessor-defined extension words to the coprocessor. When the main processor reads this primitive from the response CIR, it copies the number of bytes indicated by the length field from the instruction stream to the operand CIR. The first word or long word transferred is at the location pointed to by the scanPC when the primitive is read by the main processor. The scanPC is incremented after each word or long word is transferred. When execution of the primitive has completed, the scanPC has been incremented by the total number of bytes transferred and points to the word following the last word transferred. The main processor transfers the operands from the instruction stream, using a sequence of long-word writes, to the operand CIR. If the length field is not an even multiple of four bytes, the last two bytes from the instruction stream are transferred using a word write to the operand CIR.

7.4.8 Evaluate and Transfer Effective Address Primitive

The evaluate and transfer effective address primitive evaluates the effective address specified in the coprocessor instruction operation word and transfers the result to the coprocessor. This primitive applies to general category instructions. If this primitive is issued by the coprocessor during the execution of a conditional category instruction, the main processor initiates protocol violation exception processing. Figure 7-28 shows the format of the evaluate and transfer effective address primitive.

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

Figure 7-28. Evaluate and Transfer Effective Address Primitive Format

The evaluate and transfer effective address primitive uses the CA and PC bits as described in **7.4.2 Coprocessor Response Primitive General Format**.

When the main processor reads this primitive while executing a general category instruction, it evaluates the effective address specified in the instruction. At this point, the scanPC contains the address of the first of any required effective address extension words. The main processor increments the scanPC by two after it references each of these extension words. After the effective address is calculated, the resulting 32-bit value is written to the operand address CIR.

The MC68020/EC020 only calculates effective addresses for control alterable addressing modes in response to this primitive. If the addressing mode in the operation word is not a control alterable mode, the main processor aborts the instruction by writing a $0001 to the control CIR and initiates F-line emulation exception processing (refer to **7.5.2.2 F-Line Emulator Exceptions**).

7.4.9 Evaluate Effective Address and Transfer Data Primitive

The evaluate effective address and transfer data primitive transfers an operand between the coprocessor and the effective address specified in the coprocessor instruction operation word. This primitive applies to general category instructions. If the coprocessor issues this primitive during the execution of a conditional category instruction, the main processor initiates protocol violation exception processing. Figure 7-29 shows the format of the evaluate effective address and transfer data primitive.

15	14	13	12	11	10	9	8	7	0
CA	PC	DR	1	0	VALID EA			LENGTH	

Figure 7-29. Evaluate Effective Address and Transfer Data Primitive Format

This primitive uses the CA, PC, and DR bits as described in **7.4.2 Coprocessor Response Primitive General Format**.

The valid EA field of the primitive format specifies the valid effective address categories for this primitive. If the effective address specified in the instruction operation word is not a member of the class specified by the valid EA field, the main processor aborts the coprocessor instruction by writing an abort mask to the control CIR (refer to **7.3.2 Control CIR**) and by initiating F-line emulation exception processing. Table 7-4 lists the valid effective address field encodings.

Table 7-4. Valid Effective Address Field Codes

Field	Category
000	Control Alterable
001	Data Alterable
010	Memory Alterable
011	Alterable
100	Control
101	Data
110	Memory
111	Any Effective Address (No Restriction)

Even when the valid EA fields specified in the primitive and in the instruction operation word match, the MC68020/EC020 initiates protocol violation exception processing if the primitive requests a write to an unalterable effective address.

The length in bytes of the operand to be transferred is specified by the length field of the primitive format. Several restrictions apply to operand lengths for certain effective addressing modes. If the effective address is a main processor register (register direct mode), only operand lengths of one, two, or four bytes are valid; all other lengths cause the main processor to initiate protocol violation exception processing. Operand lengths of 0–255 bytes are valid for the memory addressing modes.

The length of 0–255 bytes does not apply to an immediate operand. The length of an immediate operand must be one byte or an even number of bytes (less than 256), and the direction of transfer must be to the coprocessor; otherwise, the main processor initiates protocol violation exception processing.

When the main processor receives the evaluate effective address and transfer data primitive during the execution of a general category instruction, it verifies that the effective address encoded in the instruction operation word is in the category specified by the primitive. If so, the processor calculates the effective address using the appropriate effective address extension words at the current scanPC address and increments the scanPC by two for each word referenced. Using long-word transfers whenever possible, the main processor then transfers the number of bytes specified in the primitive between the operand CIR and the effective address. Refer to **7.3.8 Operand CIR** for information concerning operand alignment for transfers involving the operand CIR.

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.

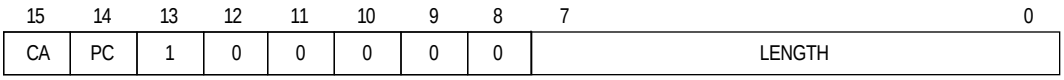


Figure 7-30. Write to Previously Evaluated Effective Address Primitive Format

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.

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.

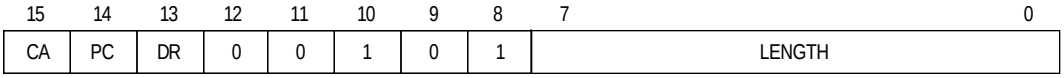


Figure 7-31. Take Address and Transfer Data Primitive Format

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.

7.4.12 Transfer to/from Top of Stack Primitive

The transfer to/from top of stack primitive transfers an operand between the coprocessor and the top of the active system stack of the main processor. This primitive applies to general and conditional category instructions. Figure 7-32 shows the format of the transfer to/from top of stack primitive.

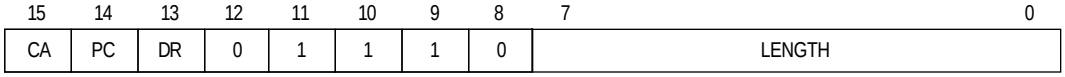


Figure 7-32. Transfer to/from Top of Stack Primitive Format

The transfer to/from top of stack 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 length in bytes of the operand to be transferred. The operand may be one, two, or four bytes in length; other length values cause the main processor to initiate protocol violation exception processing.

If DR = 0, the main processor transfers the operand from the active system stack to the operand CIR. The implied effective address mode used for the transfer is the (A7)+ addressing mode. A one-byte operand causes the stack pointer to be incremented by two after the transfer to maintain word alignment of the stack.

If DR = 1, the main processor transfers the operand from the operand CIR to the active system stack. The implied effective address mode used for the transfer is the –(A7) addressing mode. A one-byte operand causes the stack pointer to be decremented by two before the transfer to maintain word alignment of the stack.

7.4.13 Transfer Single Main Processor Register Primitive

The transfer single main processor register primitive transfers an operand between one of the main processor's data or address registers and the coprocessor. This primitive applies to general and conditional category instructions. Figure 7-33 shows the format of the transfer single main processor register primitive.

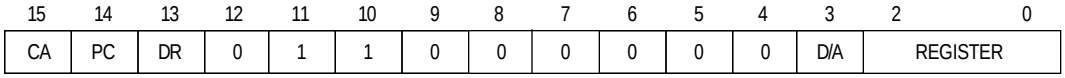


Figure 7-33. Transfer Single Main Processor Register Primitive Format

The transfer single main processor register 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 D/A bit specifies whether the primitive transfers an address or data register. D/A = 0 indicates a data register, and D/A = 1 indicates an address register. The register field contains the register number.

If DR = 0, the main processor writes the long-word operand in the specified register to the operand CIR. If DR = 1, the main processor reads a long-word operand from the operand CIR and transfers it to the specified data or address register.

7.4.14 Transfer Main Processor Control Register Primitive

The transfer main processor control register primitive transfers a long-word operand between one of its control registers and the coprocessor. This primitive applies to general and conditional category instructions. Figure 7-34 shows the format of the transfer main processor control register primitive.

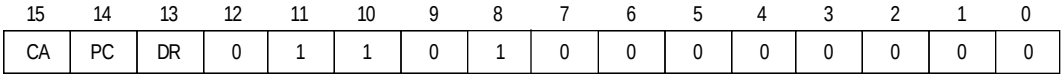


Figure 7-34. Transfer Main Processor Control Register Primitive Format

The transfer main processor control register 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.

When the main processor receives this primitive, it reads a control register select code from the register select CIR. This code determines which main processor control register is transferred. Table 7-5 lists the valid control register select codes. If the control register select code is not valid, the MC68020/EC020 initiates protocol violation exception processing (refer to **7.5.2.1 Protocol Violations**).

Table 7-5. Main Processor Control Register Select Codes

Select Code	Control Register
$x000	SFC
$x001	DFC
$x002	CACR
$x800	USP
$x801	VBR
$x802	CAAR
$x803	MSP
$x804	ISP

All other codes cause a protocol violation exception.

After reading a valid code from the register select CIR, if DR = 0, the main processor writes the long-word operand from the specified control register to the operand CIR. If DR = 1, the main processor reads a long-word operand from the operand CIR and places it in the specified control register.

7.4.15 Transfer Multiple Main Processor Registers Primitive

The transfer multiple main processor registers primitive transfers long-word operands between one or more of its data or address registers and the coprocessor. This primitive applies to general and conditional category instructions. Figure 7-35 shows the format of the transfer multiple main processor registers primitive.

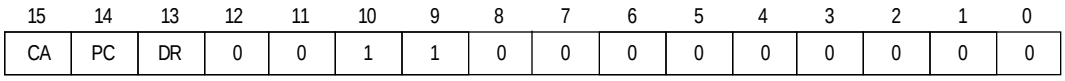


Figure 7-35. Transfer Multiple Main Processor Registers Primitive Format

The transfer multiple main processor registers 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.

When the main processor receives this primitive, it reads a 16-bit register select mask from the register select CIR. The format of the register select mask is shown in Figure 7-36. A register is transferred if the bit corresponding to the register in the register select mask is set. The selected registers are transferred in the order D7–D0 and then A7–A0.

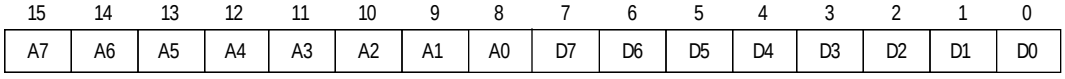


Figure 7-36. Register Select Mask Format

If DR = 0, the main processor writes the contents of each register indicated in the register select mask to the operand CIR using a sequence of long-word transfers. If DR = 1, the main processor reads a long-word operand from the operand CIR into each register indicated in the register select mask. The registers are transferred in the same order, regardless of the direction of transfer indicated by the DR bit.

7.4.16 Transfer Multiple Coprocessor Registers Primitive

The transfer multiple coprocessor registers primitive transfers from 0–16 operands between the effective address specified in the coprocessor instruction and the coprocessor. This primitive applies to general category instructions. If the coprocessor issues this primitive during the execution of a conditional category instruction, the main processor initiates protocol violation exception processing. Figure 7-37 shows the format of the transfer multiple coprocessor registers primitive.

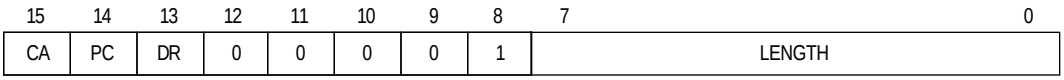


Figure 7-37. Transfer Multiple Coprocessor Registers Primitive Format

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**).

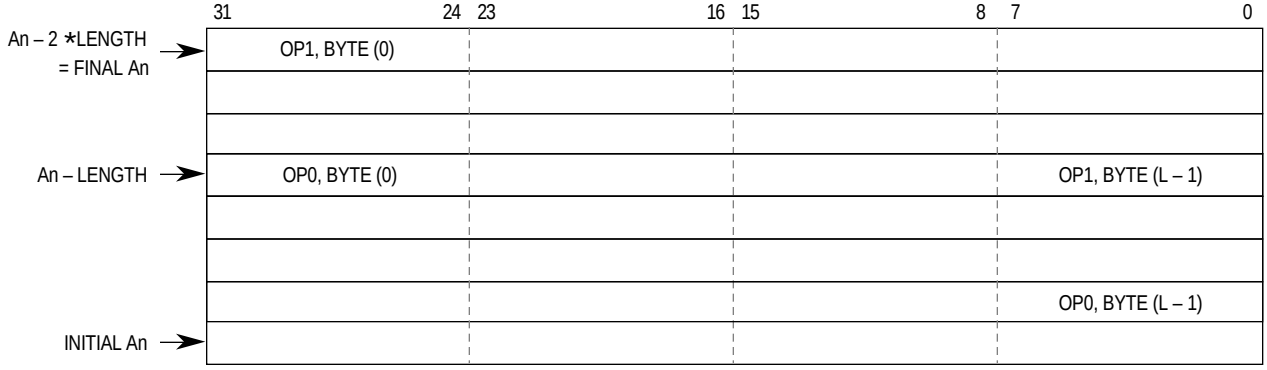
When the main processor reads this primitive, it calculates the effective address specified in the coprocessor instruction. The scanPC should be pointing to the first of any necessary effective address extension words when this primitive is read from the response CIR; the scanPC is incremented by two 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.

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



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

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

7.4.17 Transfer Status Register and ScanPC Primitive

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

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

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

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

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

If SP = 0 and DR = 0, the main processor writes the 16-bit SR value to the operand CIR. If SP = 0 and DR = 1, the main processor reads a 16-bit value from the operand CIR into the main processor SR.

If SP = 1 and DR = 0, the main processor writes the long-word value in the scanPC to the instruction address CIR and then writes the SR value to the operand CIR. If SP = 1 and DR = 1, the main processor reads a 16-bit value from the operand CIR into the SR and then reads a long-word value from the instruction address CIR into the scanPC.

With this primitive, a general category instruction can change the main processor program flow by placing a new value in the SR, in the scanPC, or new values in both the SR and the scanPC. By accessing the SR, the coprocessor can determine and manipulate the main processor condition codes, supervisor status, trace modes, selection of the active stack, and interrupt mask level.

The MC68020/EC020 discards any instruction words that have been prefetched beyond the current scanPC location when this primitive is issued with DR = 1 (transfer to main processor). The MC68020/EC020 then refills the instruction pipe from the scanPC address in the address space indicated by the S-bit of the SR.

If the MC68020/EC020 is operating in the trace on change of flow mode (T1, T0 in the SR = 01) when the coprocessor instruction begins to execute and if this primitive is issued with DR = 1 (from coprocessor to main processor), the MC68020/EC020 prepares to take a trace exception. The trace exception occurs when the coprocessor signals that it has completed all processing associated with the instruction. Changes in the trace modes due to the transfer of the SR to the main processor take effect on execution of the next instruction.

7.4.18 Take Preinstruction Exception Primitive

The take preinstruction exception primitive initiates exception processing using a coprocessor-supplied exception vector number and the preinstruction exception stack frame format. This primitive applies to general and conditional category instructions. Figure 7-40 shows the format of the take preinstruction exception primitive.

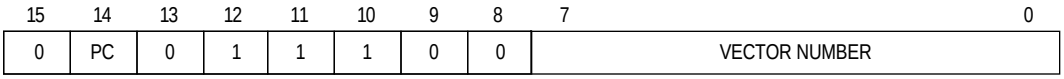


Figure 7-40. Take Preinstruction Exception Primitive Format

The take preinstruction exception primitive uses the PC bit as described in **7.4.2 Coprocessor Response Primitive General Format**. The vector number field contains the exception vector number used by the main processor to initiate exception processing.

When the main processor receives this primitive, it acknowledges the coprocessor exception request by writing an exception acknowledge mask to the control CIR (refer to **7.3.2 Control CIR**). The MC68020/EC020 then proceeds with exception processing as

described in **Section 6 Exception Processing**. The vector number for the exception is taken from the vector number field of the primitive, and the MC68020/EC020 uses the four-word stack frame format shown in Figure 7-41.

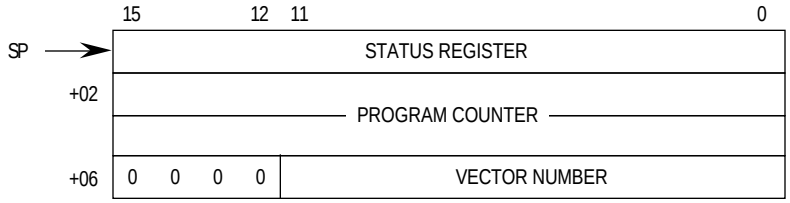


Figure 7-41. MC68020/EC020 Preinstruction Stack Frame

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

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

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

which instruction the coprocessor was executing when the exception occurred. A coprocessor can record the instruction address by setting PC = 1 in one of the primitives it uses before releasing the main processor.

7.4.19 Take Midinstruction Exception Primitive

The take midinstruction exception primitive initiates exception processing using a coprocessor-supplied exception vector number and the midinstruction exception stack frame format. This primitive applies to general and conditional category instructions. Figure 7-42 shows the format of the take midinstruction exception primitive.

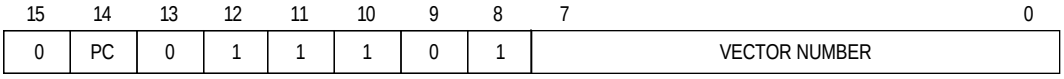


Figure 7-42. Take Midinstruction Exception Primitive Format

The take midinstruction exception primitive uses the PC bit as described in **7.4.2 Coprocessor Response Primitive General Format**. The vector number field contains the exception vector number used by the main processor to initiate exception processing.

When the main processor receives this primitive, it acknowledges the coprocessor exception request by writing an exception acknowledge mask (refer to **7.3.2 Control CIR**) to the control CIR. The MC68020/EC020 then performs exception processing as described in **Section 6 Exception Processing**. The vector number for the exception is taken from the vector number field of the primitive, and the MC68020/EC020 uses the 10-word stack frame format shown in Figure 7-43.

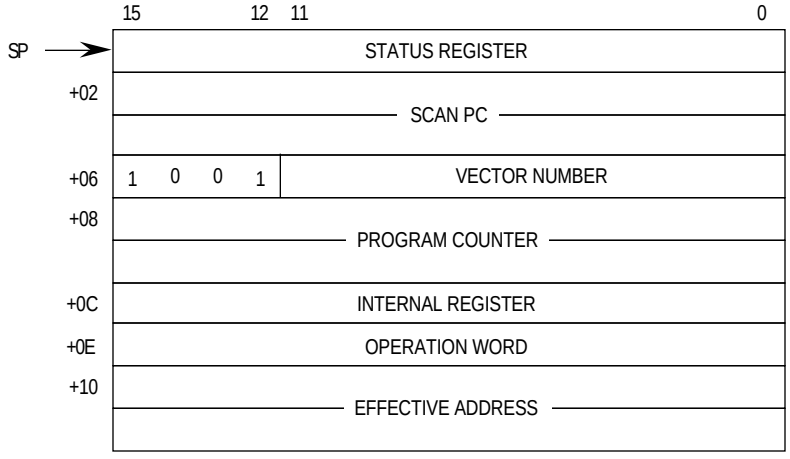


Figure 7-43. MC68020/EC020 Midinstruction Stack Frame

The PC value saved in this stack frame is the operation word address of the coprocessor instruction during which the primitive is received. The scanPC field contains the value of the MC68020/EC020 scanPC when the primitive is received. If the current instruction does not evaluate an effective address prior to the exception request primitive, the value of the effective address field in the stack frame is undefined.

The coprocessor uses this primitive to request exception processing for an exception during the instruction dialog with the main processor. If the exception handler does not modify the stack frame, the MC68020/EC020 returns from the exception handler and reads the response CIR. Thus, the main processor attempts to continue executing the suspended instruction by reading the response CIR and processing the primitive it receives.

7.4.20 Take Postinstruction Exception Primitive

The take postinstruction exception primitive initiates exception processing using a coprocessor-supplied exception vector number and the postinstruction exception stack frame format. This primitive applies to general and conditional category instructions. Figure 7-44 shows the format of the take postinstruction exception primitive.

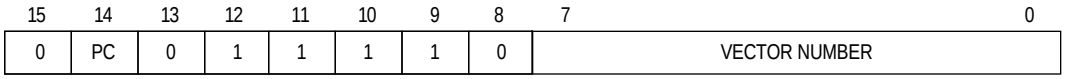


Figure 7-44. Take Postinstruction Exception Primitive Format

The take postinstruction exception primitive uses the PC bit as described in **7.4.2 Coprocessor Response Primitive General Format**. The vector number field contains the exception vector number used by the main processor to initiate exception processing.

When the main processor receives this primitive, it acknowledges the coprocessor exception request by writing an exception acknowledge mask to the control CIR (refer to **7.3.2 Control CIR**). The MC68020/EC020 then performs exception processing as described in **Section 6 Exception Processing**. The vector number for the exception is taken from the vector number field of the primitive, and the MC68020/EC020 uses the six-word stack frame format shown in Figure 7-45.

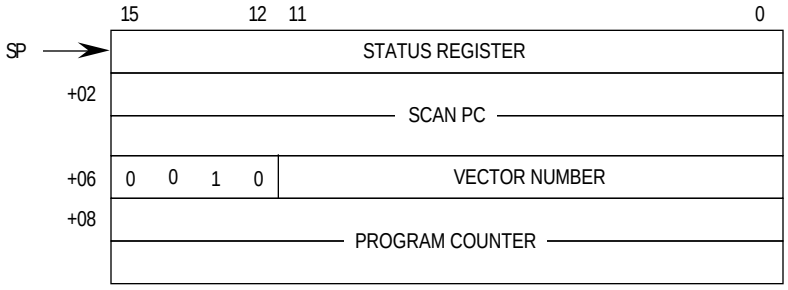


Figure 7-45. MC68020/EC020 Postinstruction Stack Frame

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**.

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.

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**.

7.5.1.5 FORMAT ERRORS. Format errors are the only coprocessor-detected exceptions that are not signaled to the main processor with a response primitive. When the main processor writes a format word to the restore CIR during the execution of a cpRESTORE instruction, the coprocessor decodes this word to determine if it is valid (refer to **7.2.3.3 Coprocessor Context Save Instruction**). If the format word is not valid, the coprocessor places the invalid format code in the restore CIR. When the main processor reads the invalid format code, it aborts the coprocessor instruction by writing an abort mask to the control CIR (refer to **7.3.2 Control CIR**). The main processor then performs exception processing using a four-word preinstruction stack frame and the format error exception vector number 14. Thus, if the exception handler does not modify the stack frame, the MC68020/EC020 restarts the cpRESTORE instruction when the RTE instruction in the handler is executed. If the coprocessor returns the invalid format code when the main processor reads the save CIR to initiate a cpSAVE instruction, the main processor performs format error exception processing as outlined for the cpRESTORE instruction.

7.5.2 Main-Processor-Detected Exceptions

A number of exceptions related to coprocessor instruction execution are detected and serviced by the main processor instead of the coprocessor. These exceptions can be related to the execution of coprocessor response primitives, communication across the M68000 coprocessor interface, or completion of conditional coprocessor instructions by the main processor.

7.5.2.1 PROTOCOL VIOLATIONS. The main processor detects a protocol violation when it reads a primitive from the response CIR that is not a valid primitive. The protocol violations that can occur in response to the primitives defined for the M68000 coprocessor interface are summarized in Table 7-6.

Table 7-6. Exceptions Related to Primitive Processing

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

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

Abbreviations:

EA—Effective Address
CP—Coprocessor

When the MC68020/EC020 detects a protocol violation, it does not automatically notify the coprocessor of the resulting exception by writing to the control CIR. However, the exception handling routine may use the MOVES instruction to read the response CIR and thus determine the primitive that caused the MC68020/EC020 to initiate protocol violation exception processing. The main processor initiates exception processing using the midinstruction stack frame (refer to Figure 7-43) and the coprocessor protocol violation exception vector number 13. If the exception handler does not modify the stack frame, the main processor reads the response CIR again following the execution of an RTE instruction to return from the exception handler. This protocol allows extensions to the M68000 coprocessor interface to be emulated in software by a main processor that does not provide hardware support for these extensions. Thus, the protocol violation is transparent to the coprocessor if the primitive execution can be emulated in software by the main processor.

7.5.2.2 F-LINE EMULATOR EXCEPTIONS. The F-line emulator exceptions detected by the MC68020/EC020 are either explicitly or implicitly related to the encodings of F-line operation words in the instruction stream. If the main processor determines that an F-line operation word is not valid, it initiates F-line emulator exception processing. Any F-line operation word with bits 8–6 = 110 or 111 causes the MC68020/EC020 to initiate exception processing without initiating any communication with the coprocessor for that instruction. Also, an operation word with bits 8–6 = 000–101 that does not map to one of the valid coprocessor instructions in the instruction set causes the MC68020/EC020 to initiate F-line emulator exception processing. If the F-line emulator exception is either of these two situations, the main processor does not write to the control CIR prior to initiating exception processing.

F-line exceptions can also occur if the operations requested by a coprocessor response primitive are not compatible with the effective address type in bits 5–0 of the coprocessor instruction operation word. The F-line emulator exceptions that can result from the use of the M68000 coprocessor response primitives are summarized in Table 7-6. If the exception is caused by receiving an invalid primitive, the main processor aborts the coprocessor instruction in progress by writing an abort mask (refer to **7.3.2 Control CIR**) to the control CIR prior to F-line emulator exception processing.

Another type of F-line emulator exception occurs when a bus error occurs during the CIR access that initiates a coprocessor instruction. The main processor assumes that the coprocessor is not present and takes the exception.

When the main processor initiates F-line emulator exception processing, it uses the four-word preinstruction exception stack frame (refer to Figure 7-41) and the F-line emulator exception vector number 11. Thus, if the exception handler does not modify the stack frame, the main processor attempts to restart the instruction that caused the exception after it executes an RTE instruction to return from the exception handler.

If the cause of the F-line exception can be emulated in software, the handler stores the results of the emulation in the appropriate registers of the programming model and in the status register field of the saved stack frame. The exception handler adjusts the program

counter field of the saved stack frame to point to the next instruction operation word and executes the RTE instruction. The MC68020/EC020 then executes the instruction following the instruction that was emulated.

The exception handler should also check the copy of the SR on the stack to determine whether tracing is enabled. If tracing is enabled, the trace exception processing should also be emulated. Refer to **Section 6 Exception Processing** for additional information.

7.5.2.3 PRIVILEGE VIOLATIONS. Privilege violations can result from the cpSAVE and cpRESTORE instructions and from the supervisor check coprocessor response primitive. The MC68020/EC020 initiates privilege violation exception processing if it attempts to execute either the cpSAVE or cpRESTORE instruction when it is in the user state (S = 0 in the SR). The main processor initiates this exception processing prior to any communication with the coprocessor associated with the cpSAVE or cpRESTORE instructions.

If the main processor is executing a coprocessor instruction in the user state when it reads the supervisor check primitive, it aborts the coprocessor instruction in progress by writing an abort mask to the control CIR (refer to **7.3.2 Control CIR**). The main processor then performs privilege violation exception processing.

If a privilege violation occurs, the main processor initiates exception processing using the four-word preinstruction stack frame (refer to Figure 7-41) and the privilege violation exception vector number 8. Thus, if the exception handler does not modify the stack frame, the main processor attempts to restart the instruction during which the exception occurred after it executes an RTE to return from the handler.

7.5.2.4 cpTRAPcc INSTRUCTION TRAPS. If, during the execution of a cpTRAPcc instruction, the coprocessor returns the TRUE condition indicator to the main processor with a null primitive, the main processor initiates trap exception processing. The main processor uses the six-word postinstruction exception stack frame (refer to Figure 7-45) and the trap exception vector number 7. The scanPC field of this stack frame contains the address of the instruction following the cpTRAPcc instruction. The processing associated with the cpTRAPcc instruction can then proceed, and the exception handler can locate any immediate operand words encoded in the cpTRAPcc instruction using the information contained in the six-word stack frame. If the exception handler does not modify the stack frame, the main processor executes the instruction following the cpTRAPcc instruction after it executes an RTE instruction to exit from the handler.

7.5.2.5 TRACE EXCEPTIONS. The MC68020/EC020 supports two modes of instruction tracing, as discussed in **Section 6 Exception Processing**. In the trace on instruction execution mode, the MC68020/EC020 takes a trace exception after completing each instruction. In the trace on change of flow mode, the MC68020/EC020 takes a trace exception after each instruction that alters the SR or places an address other than the address of the next instruction in the PC.

The protocol used to execute coprocessor cpSAVE, cpRESTORE, or conditional category instructions does not change when a trace exception is pending in the main processor. The main processor performs a pending trace on instruction execution exception after completing the execution of that instruction. If the main processor is in the trace on change of flow mode and an instruction places an address other than that of the next instruction in the PC, the processor takes a trace exception after it executes the instruction.

If a trace exception is not pending during a general category instruction, the main processor terminates communication with the coprocessor after reading any primitive with CA = 0. Thus, the coprocessor can complete a cpGEN instruction concurrently with the execution of instructions by the main processor. When a trace exception is pending, however, the main processor must ensure that all processing associated with a cpGEN instruction has been completed before it takes the trace exception. In this case, the main processor continues to read the response CIR and to service the primitives until it receives either a null primitive with CA = 0 and PF = 1 or until exception processing caused by a take postinstruction exception primitive has completed. The coprocessor should return the null primitive with CA = 0 and PF = 0 while it is completing the execution of the cpGEN instruction. The main processor may service pending interrupts between reads of the response CIR if IA = 1 in these primitives (refer to Table 7-3). This protocol ensures that a trace exception is not taken until all processing associated with a cpGEN instruction has completed.

If T1, T0 = 01 in the MC68020/EC020 SR (trace on change of flow mode) when a general category instruction is initiated, a trace exception is taken for the instruction only when the coprocessor issues a transfer status register and scanPC primitive with DR = 1 during the execution of that instruction. In this case, it is possible that the coprocessor is still executing the cpGEN instruction concurrently when the main processor begins execution of the trace exception handler. A cpSAVE instruction executed during the trace on change of flow exception handler could thus suspend the execution of a concurrently operating cpGEN instruction.

7.5.2.6 INTERRUPTS. Interrupt processing, discussed in **Section 6 Exception Processing**, can occur at any instruction boundary. Interrupts are also serviced during the execution of a general or conditional category instruction under either of two conditions. If the main processor reads a null primitive with CA = 1 and IA = 1, it services any pending interrupts prior to reading the response CIR. Similarly, if a trace exception is pending during cpGEN instruction execution and the main processor reads a null primitive with CA = 0, IA = 1, and PF = 0 (refer to **7.5.2.5 Trace Exceptions**), the main processor services pending interrupts prior to reading the response CIR again.

The MC68020/EC020 uses the 10-word midinstruction stack frame (see Figure 7-43) when it services interrupts during the execution of a general or conditional category coprocessor instruction. Since it uses this stack frame, the main processor can perform all necessary processing and then return to read the response CIR. Thus, it can continue execution of the coprocessor instruction during which the interrupt exception occurred.

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**.

7.5.3 Coprocessor Reset

Either an external reset signal or a RESET instruction can reset the external devices of a system. The system designer can design a coprocessor to be reset and initialized by both reset types or by external reset signals only. To be consistent with the MC68020/EC020 design, the coprocessor should be affected by external reset signals only and not by RESET instructions, because the coprocessor is an extension to the main processor programming model and to the internal state of the MC68020/EC020.

7.6 COPROCESSOR SUMMARY

Coprocessor instruction formats are included with the instruction formats in the M68000PM/AD, *M68000 Family Programmer's Reference Manual*.

The M68000 coprocessor response primitive formats are shown in this section. Any response primitive with bits 13–8 = $00 or $3F causes a protocol violation exception. Response primitives with bits 13–8 = $0B, $18–$1B, $1F, $28–$2B, and $38–3B currently cause protocol violation exceptions; they are undefined and reserved for future use by Motorola.

Busy

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

Transfer Multiple Coprocessor Registers

15	14	13	12	11	10	9	8	7	0
CA	PC	DR	0	0	0	0	1	LENGTH	

Transfer Status Register and ScanPC

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

Supervisor Check

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

Take Address and Transfer Data

15	14	13	12	11	10	9	8	7	0
CA	PC	DR	0	0	1	0	1	LENGTH	

Transfer Multiple Main Processor Registers

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

Transfer Operation Word

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

Null

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	1	0	0	IA	0	0	0	0	0	0	PF	TF

Evaluate and Transfer Effective Address

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

Transfer Single Main Processor Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	1	1	0	0	0	0	0	0	D/A	REGISTER		

Transfer Main Processor Control Register

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

Transfer to/from Top of Stack

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

Transfer from Instruction Stream

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

Evaluate Effective Address and Transfer Data

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

Take Preinstruction Exception

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PC	0	1	1	1	0	0	VECTOR NUMBER							

Take Midinstruction Exception

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PC	0	1	1	1	0	1	VECTOR NUMBER							

Take Postinstruction Exception

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PC	0	1	1	1	1	0	VECTOR NUMBER							

Write to Previously Evaluated Effective Address

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