Welcome to E-XFL.COM

Understanding Embedded - Microprocessors

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

Applications of Embedded - Microprocessors

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

Details	
Product Status	Obsolete
Core Processor	68020
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	16MHz
Co-Processors/DSP	-
RAM Controllers	-
Graphics Acceleration	No
Display & Interface Controllers	-
Ethernet	-
SATA	-
USB	-
Voltage - I/O	5.0V
Operating Temperature	0°C ~ 70°C (TA)
Security Features	-
Package / Case	100-BQFP
Supplier Device Package	100-QFP (14x20)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mc68ec020aa16

TABLE OF CONTENTS (Continued)

Table 3-1. Signal Index

Signal Name	Mnemonic	Function
Function Codes	FC2–FC0	3-bit function code used to identify the address space of each bus cycle.
Address Bus MC68020 MC68EC020	 A31–A0 A23–A0	 32-bit address bus 24-bit address bus
Data Bus	D31–D0	32-bit data bus used to transfer 8, 16, 24, or 32 bits of data per bus cycle.
Size	SIZ1, SIZ0	Indicates the number of bytes remaining to be transferred for this cycle. These signals, together with A1 and A0, define the active sections of the data bus.
*External Cycle Start	ECS	Provides an indication that a bus cycle is beginning.
*Operand Cycle Start	OCS	Identical operation to that of ECS except that OCS is asserted only during the first bus cycle of an operand transfer.
Read/Write	R/W	Defines the bus transfer as a processor read or write.
Read-Modify-Write Cycle	RMC	Provides an indicator that the current bus cycle is part of an indivisible read-modify-write operation.
Address Strobe	AS	Indicates that a valid address is on the bus.
Data Strobe	DS	Indicates that valid data is to be placed on the data bus by an external device or has been placed on the data bus by the MC68020/EC020.
*Data Buffer Enable	DBEN	Provides an enable signal for external data buffers.
Data Transfer and Size Acknowledge	DSACK1, DSACK0	Bus response signals that indicate the requested data transfer operation has completed. In addition, these two lines indicate the size of the external bus port on a cycle-by-cycle basis and are used for asynchronous transfers.
Interrupt Priority Level	IPL2–IPL0	Provides an encoded interrupt level to the processor.
*Interrupt Pending	IPEND	Indicates that an interrupt is pending.
Autovector	AVEC	Requests an autovector during an interrupt acknowledge cycle.
Bus Request	BR	Indicates that an external device requires bus mastership.
Bus Grant	BG	Indicates that an external device may assume bus mastership.
*Bus Grant Acknowledge	BGACK	Indicates that an external device has assumed bus mastership.
Reset	RESET	System reset.
Halt	HALT	Indicates that the processor should suspend bus activity or that the processor has halted due to a double bus fault.
Bus Error	BERR	Indicates that an erroneous bus operation is being attempted.
Cache Disable	CDIS	Statically disables the on-chip cache to assist emulator support.
Clock	CLK	Clock input to the processor.
Power Supply	V_{CC}	Power supply.
Ground	GND	Ground connection.

*This signal is implemented in the MC68020 and not implemented in the MC68EC020.

3.13 SIGNAL SUMMARY

Table 3-2 provides a summary of the characteristics of the signals discussed in this section. Signal names preceded by an asterisk (*) are implemented in the MC68020 and not implemented in the MC68EC020.

Table 3-2. Signal Summary

Signal Function	Signal Name	Input/Output	Active State	Three-State
Function Codes	FC2–FC0	Output	High	Yes
Address Bus MC68020 MC68EC020	 A31–A0 A23–A0	Output	High	Yes
Data Bus	D31–D0	Input/Output	High	Yes
Transfer Size	SIZ1, SIZ0	Output	High	Yes
*Operand Cycle Start	OCS	Output	Low	No
*External Cycle Start	ECS	Output	Low	No
Read/Write	R/W	Output	High/Low	Yes
Read-Modify-Write Cycle	RMC	Output	Low	Yes
Address Strobe	AS	Output	Low	Yes
Data Strobe	DS	Output	Low	Yes
*Data Buffer Enable	DBEN	Output	Low	Yes
Data Transfer and Size Acknowledge	DSACK1, DSACK0	Input	Low	—
Interrupt Priority Level	IPL2–IPL0	Input	Low	—
*Interrupt Pending	IPEND	Output	Low	No
Autovector	AVEC	Input	Low	—
Bus Request	BR	Input	Low	—
Bus Grant	BG	Output	Low	No
*Bus Grant Acknowledge	BGACK	Input	Low	—
Reset	RESET	Input/Output	Low	No**
Halt	HALT	Input/Output	Low	No**
Bus Error	BERR	Input	Low	—
Cache Disable	CDIS	Input	Low	—
Clock	CLK	Input	—	—
Power Supply	V_{CC}	Input	—	—
Ground	GND	Input	—	—

*This signal is implemented in the MC68020 and not implemented in the MC68EC020.

** Open-drain

F—Freeze Cache

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

E—Enable Cache

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

4.3.2 Cache Address Register (CAAR)

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

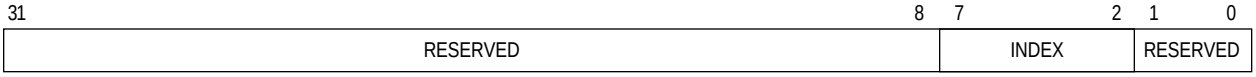


Figure 4-3. Cache Address Register

Bits 31–8, 1, and 0—Reserved

These bits are reserved for use by Motorola.

Index Field

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

When initiating a bus cycle, the MC68020 asserts $\overline{ECS}$ in addition to A1–A0, SIZ1, SIZ0, FC2–FC0, and R/$\overline{W}$. $\overline{ECS}$ can be used to initiate various timing sequences that are eventually qualified with $\overline{AS}$. Qualification with $\overline{AS}$ may be required since, in the case of an internal cache hit, a bus cycle may be aborted after $\overline{ECS}$ has been asserted. During the first MC68020 external bus cycle of an operand transfer, $\overline{OCS}$ is asserted with $\overline{ECS}$. When several bus cycles are required to transfer the entire operand, $\overline{OCS}$ is asserted only at the beginning of the first external bus cycle. With respect to $\overline{OCS}$, an "operand" is any entity required by the execution unit, whether a program or data item. Note that $\overline{ECS}$ and $\overline{OCS}$ are not implemented in the MC68EC020.

The FC2–FC0 signals select one of eight address spaces (see Table 2-1) to which the address applies. Five address spaces are presently defined. Of the remaining three, one is reserved for user definition, and two are reserved by Motorola for future use. FC2–FC0 are valid while $\overline{AS}$ is asserted.

The SIZ1 and SIZ0 signals indicate the number of bytes remaining to be transferred during an operand cycle (consisting of one or more bus cycles) or during a cache fill operation from a device with a port size that is less than 32 bits. Table 5-2 lists the encoding of SIZ1 and SIZ0. SIZ1 and SIZ0 are valid while $\overline{AS}$ is asserted.

The R/$\overline{W}$ signal determines the direction of the transfer during a bus cycle. When required, this signal changes state at the beginning of a bus cycle and is valid while $\overline{AS}$ is asserted. R/$\overline{W}$ only transitions when a write cycle is preceded by a read cycle or vice versa. This signal may remain low for two consecutive write cycles.

The $\overline{RMC}$ signal is asserted at the beginning of the first bus cycle of a read-modify-write operation and remains asserted until completion of the final bus cycle of the operation. The $\overline{RMC}$ signal is guaranteed to be negated before the end of state 0 for a bus cycle following a read-modify-write operation.

5.1.2 Address Bus

A31–A0 (for the MC68020) or A23–A0 (for the MC68EC020) define the address of the byte (or the most significant byte) to be transferred during a bus cycle. The processor places the address on the bus at the beginning of a bus cycle. The address is valid while $\overline{AS}$ is asserted. In the MC68EC020, A31–A24 are used internally, but not externally.

5.1.3 Address Strobe

$\overline{AS}$ is a timing signal that indicates the validity of an address on the address bus and of many control signals. It is asserted one-half clock after the beginning of a bus cycle.

5.1.4 Data Bus

D31–D0 comprise a bidirectional, nonmultiplexed parallel bus that contains the data being transferred to or from the processor. A read or write operation may transfer 8, 16, 24, or 32 bits of data (one, two, three, or four bytes) in one bus cycle. During a read cycle, the data is latched by the processor on the last falling edge of the clock for that bus cycle. For

Table 5-7. Data Bus Byte Enable Signals for Byte, Word, and Long-Word Ports

Transfer Size	SIZ1	SIZ0	A1	A0	Data Bus Active Sections Byte (B), Word (W) , Long-Word (L) Ports			
					D31–D24	D23–D16	D15–D8	D7–D0
Byte	0	1	0	0	B W L	—	—	—
	0	1	0	1	B	W L	—	—
	0	1	1	0	B W	—	L	—
	0	1	1	1	B	W	—	L
Word	1	0	0	0	B W L	W L	—	—
	1	0	0	1	B	W L	L	—
	1	0	1	0	B W	W	L	L
	1	0	1	1	B	W	—	L
3 Bytes	1	1	0	0	B W L	W L	L	—
	1	1	0	1	B	W L	L	L
	1	1	1	0	B W	W	L	L
	1	1	1	1	B	W	—	L
Long Word	0	0	0	0	B W L	W L	L	L
	0	0	0	1	B	W L	L	L
	0	0	1	0	B W	W	L	L
	0	0	1	1	B	W	—	L

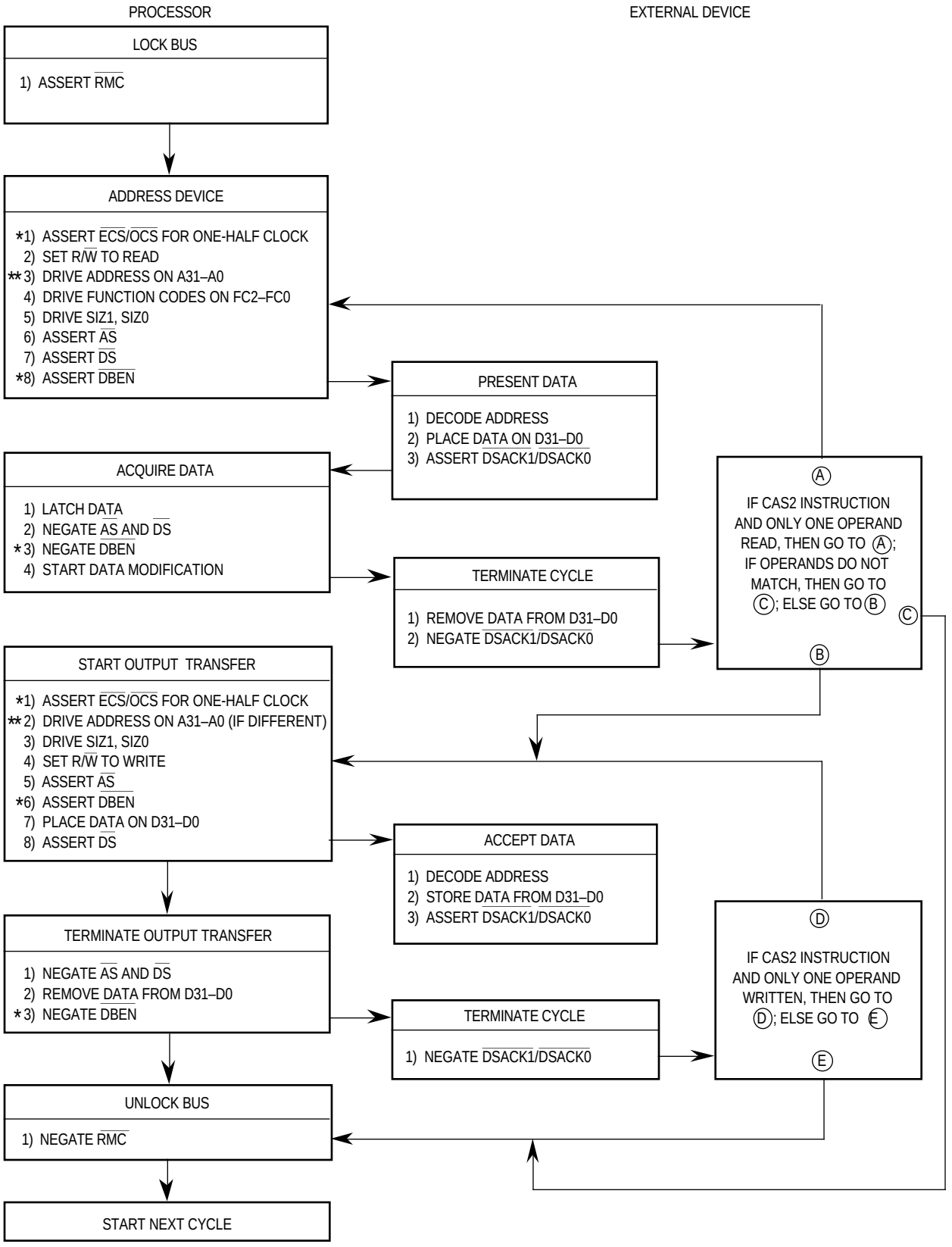
Figure 5-18 shows a logic diagram of one method for generating byte enable signals for 16- and 32-bit ports from the SIZ1, SIZ0, A1, and A0 encodings and the R/$\overline{W}$ signal.

5.2.5 Cache Interactions

The organization and requirements of the on-chip instruction cache affect the interpretation of $\overline{DSACK1}$ and $\overline{DSACK0}$. Since the MC68020/EC020 attempts to load all instructions into the on-chip cache, the bus may operate differently when caching is enabled. Specifically, on read cycles that terminate normally, the A1, A0, SIZ1, and SIZ0 signals do not apply.

The cache can also affect the assertion of $\overline{AS}$ and the operation of a read cycle. The search of the cache by the processor begins when the sequencer requires an instruction. At this time, the bus controller may also initiate an external bus cycle in case the requested item is not resident in the instruction cache. If an internal cache hit occurs, the external cycle aborts, and $\overline{AS}$ is not asserted.

For the MC68020, if the bus is not occupied with another read or write cycle, the bus controller asserts the $\overline{ECS}$ signal (and the $\overline{OCS}$ signal, if appropriate). It is possible to have $\overline{ECS}$ asserted on multiple consecutive clock cycles. Note that there is a minimum time specified from the negation of $\overline{ECS}$ to the next assertion of $\overline{ECS}$ (refer to **Section 10 Electrical Characteristics**). Instruction prefetches can occur every other clock so that if, after an aborted cycle due to an instruction cache hit, the bus controller asserts $\overline{ECS}$ on the next clock, this second cycle is for a data fetch. Note that, if the bus controller is executing other cycles, these aborted cycles due to cache hits may not be seen externally.



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

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

5.6 BUS SYNCHRONIZATION

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

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

5.7 BUS ARBITRATION

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

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

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

Table 6-1. Exception Vector Assignments

Vector Number	Vector Offset		Assignment
	Hex	Space	
0	000	SP	Reset Initial Interrupt Stack Pointer
1	004	SP	Reset Initial Program Counter
2	008	SD	Bus Error
3	00C	SD	Address Error
4	010	SD	Illegal Instruction
5	014	SD	Zero Divide
6	018	SD	CHK, CHK2 Instruction
7	01C	SD	cpTRAPcc, TRAPcc, TRAPV Instructions
8	020	SD	Privilege Violation
9	024	SD	Trace
10	028	SD	Line 1010 Emulator
11	02C	SD	Line 1111 Emulator
12	030	SD	(Unassigned, Reserved)
13	034	SD	Coprocessor Protocol Violation
14	038	SD	Format Error
15	03C	SD	Uninitialized Interrupt
16–23	040 05C	SD SD	Unassigned, Reserved
24	060	SD	Spurious Interrupt
25	064	SD	Level 1 Interrupt Autovector
26	068	SD	Level 2 Interrupt Autovector
27	06C	SD	Level 3 Interrupt Autovector
28	070	SD	Level 4 Interrupt Autovector
29	074	SD	Level 5 Interrupt Autovector
30	078	SD	Level 6 Interrupt Autovector
31	07C	SD	Level 7 Interrupt Autovector
32–47	080 0BC	SD SD	TRAP #0–15 Instruction Vectors
48	0C0	SD	FPCP Branch or Set on Unordered Condition
49	0C4	SD	FPCP Inexact Result
50	0C8	SD	FPCP Divide by Zero
51	0CC	SD	FPCP Underflow
52	0D0	SD	FPCP Operand Error
53	0D4	SD	FPCP Overflow
54	0D8	SD	FPCP Signaling NAN
55	0DC	SD	Unassigned, Reserved
56	0E0	SD	PMMU Configuration
57	0E4	SD	PMMU Illegal Operation
58	0E8	SD	PMMU Access Level Violation
59–63	0EC 0FC	SD SD	Unassigned, Reserved
64–255	100 3FC	SD SD	User-Defined Vectors (192)

SP—Supervisor Program Space

SD—Supervisor Data Space

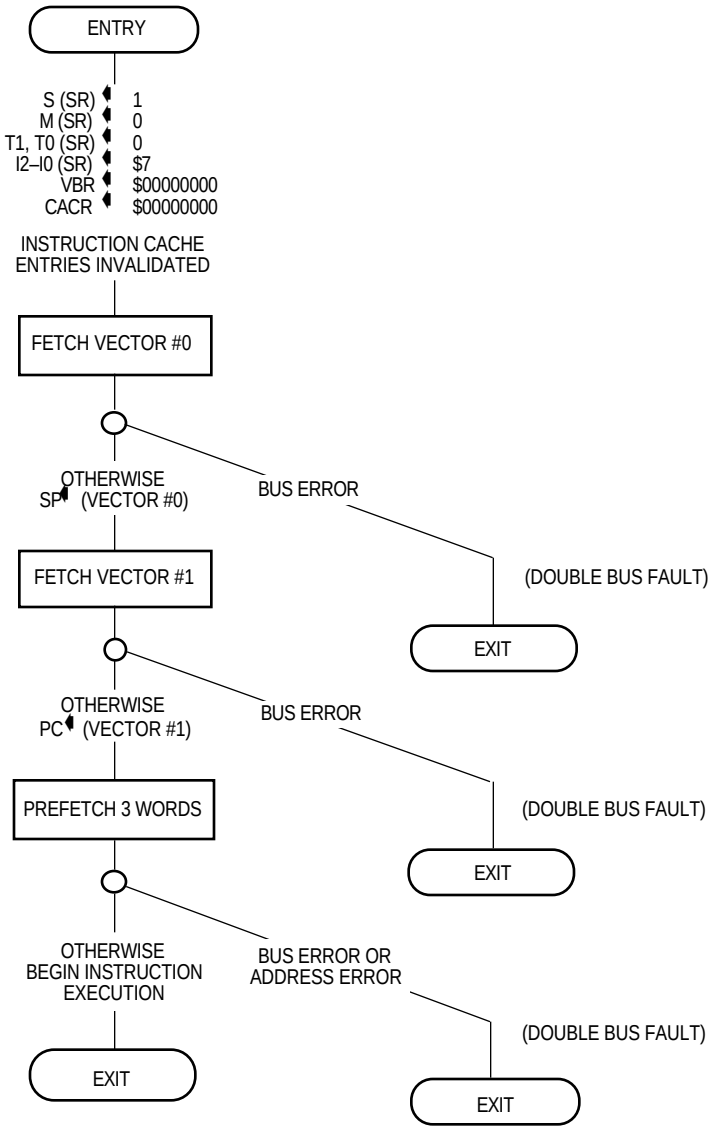


Figure 6-1. Reset Operation Flowchart

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

execution of instructions. The processor also saves the contents of some of its internal registers. The information saved on the stack is sufficient to identify the cause of the bus fault and recover from the error.

For efficiency, the MC68020/EC020 uses two different bus error stack frame formats. When the bus error exception is taken at an instruction boundary, less information is required to recover from the error, and the processor builds the short bus fault stack frame as shown in Table 6-5. When the exception is taken during the execution of an instruction, the processor must save its entire state for recovery and uses the long bus fault stack frame shown in Table 6-5. The format code in the stack frame distinguishes the two stack frame formats. Stack frame formats are described in detail in **6.4 Exception Stack Frame Formats**.

If a bus error occurs during the exception processing for a bus error, address error, or reset or while the processor is loading internal state information from the stack during the execution of an RTE instruction, a double bus fault occurs and the processor enters the halted state. In this case, the processor does not attempt to alter the current state of memory. Only an external RESET can restart a processor halted by a double bus fault.

6.1.3 Address Error Exception

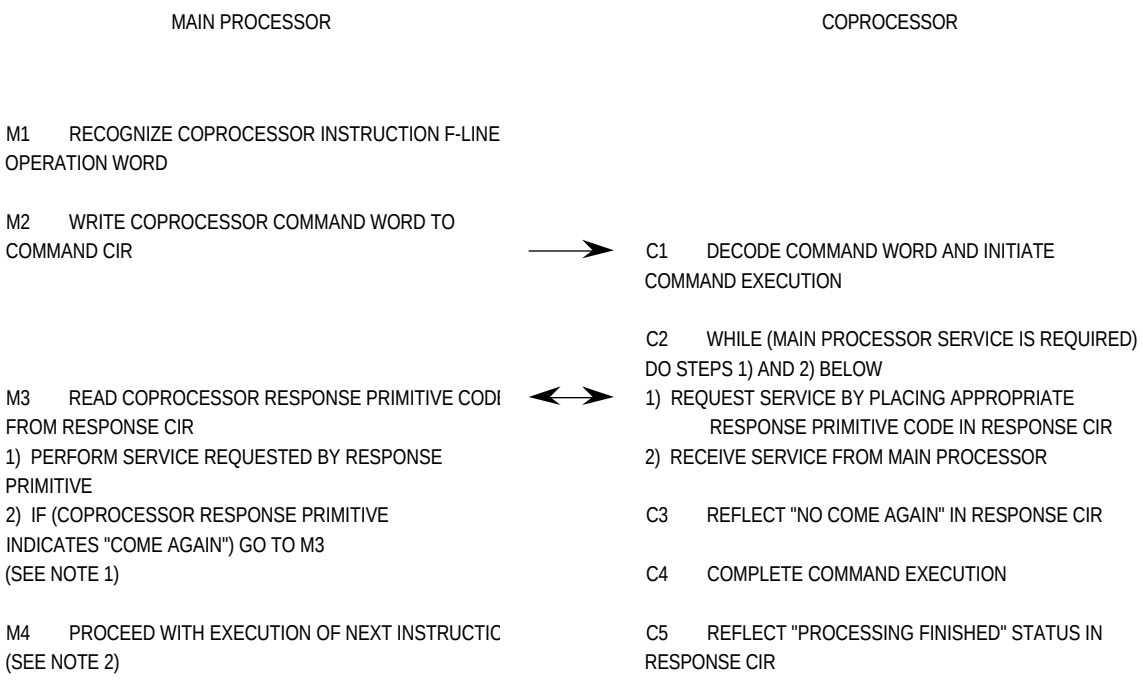
An address error exception occurs when the processor attempts to prefetch an instruction from an odd address. This exception is similar to a bus error exception but is internally initiated. A bus cycle is not executed, and the processor begins exception processing immediately. After exception processing commences, the sequence is the same as that for bus error exceptions described in the preceding paragraphs, except that the vector number is 3 and the vector offset in the stack frame refers to the address error vector. Either a short or long bus fault stack frame may be generated. If an address error occurs during the exception processing for a bus error, address error, or reset, a double bus fault occurs.

6.1.4 Instruction Trap Exception

Certain instructions are used to explicitly cause trap exceptions. The TRAP instruction always forces an exception and is useful for implementing system calls in user programs. The TRAPcc, TRAPV, cpTRAPcc, CHK, and CHK2 instructions force exceptions if the user program detects an error, which may be an arithmetic overflow or a subscript value that is out of bounds.

The DIVS and DIVU instructions force exceptions if a division operation is attempted with a divisor of zero.

When a trap exception occurs, the processor copies the SR internally, enters the supervisor privilege level (by setting the S-bit in the SR), and clears the T1 and T0 bits in the SR. If tracing is enabled for the instruction that caused the trap, a trace exception is taken after the RTE instruction from the trap handler is executed, and the trace corresponds to the trap instruction; the trap handler routine is not traced. The processor generates a vector number according to the instruction being executed; for the TRAP



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.

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.

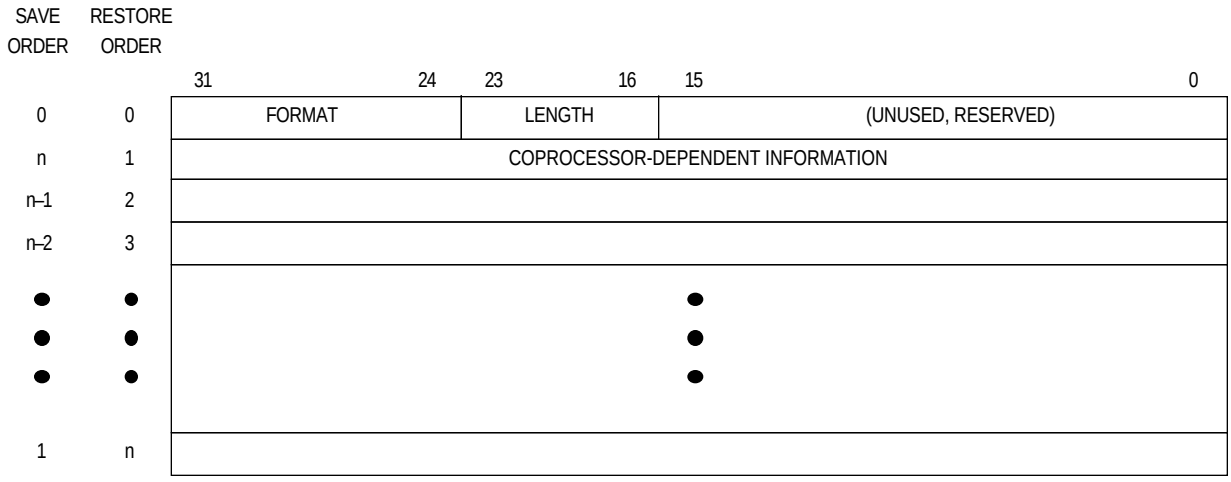


Figure 7-14. Coprocessor State Frame Format in Memory

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.

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.

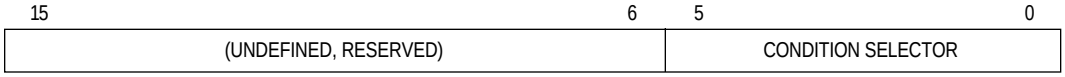


Figure 7-20. Condition CIR Format

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.

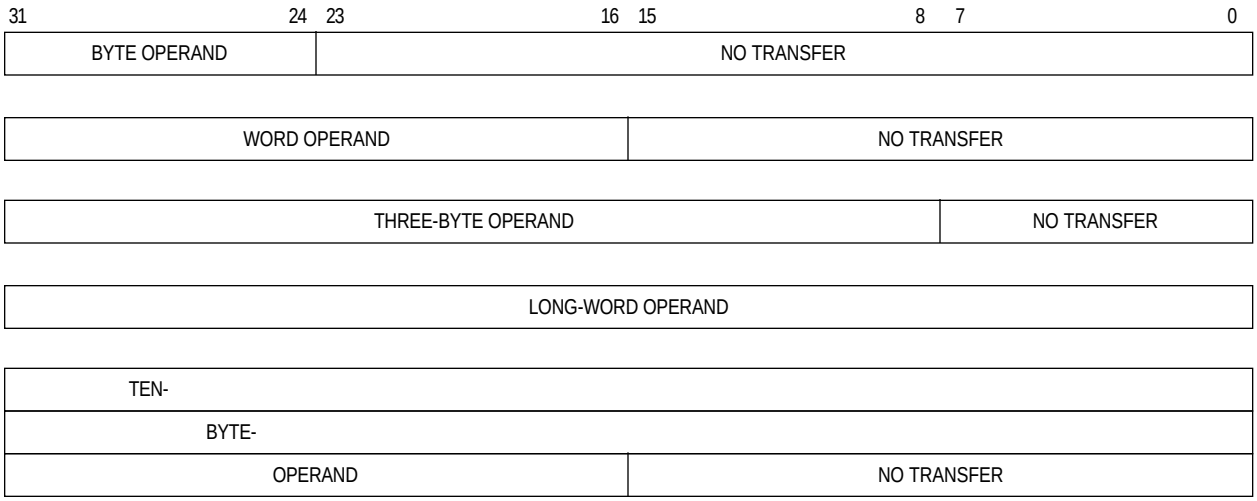


Figure 7-21. Operand Alignment for Operand CIR Accesses

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.

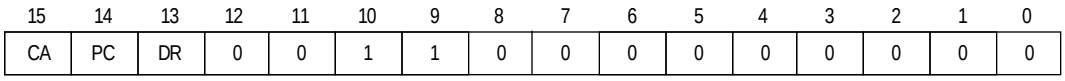


Figure 7-35. Transfer Multiple Main Processor Registers Primitive Format

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.

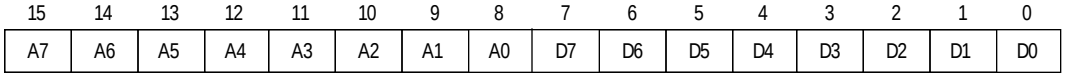


Figure 7-36. Register Select Mask Format

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.

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.

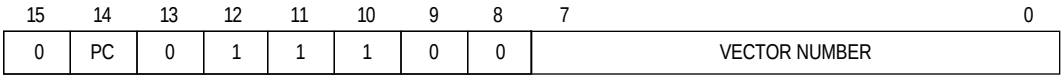


Figure 7-40. Take Preinstruction Exception Primitive Format

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

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.

WORST CASE (Concluded)

Source Address Mode	Destination					
	([d_{16},B],I)	**([d_{16},B],I,d16)**	**([d_{16},B],I,d_{32})**	**([d_{32},B],I)**	**([d_{32},B],I,d_{16})**	**([d_{32},B],I,d_{32})**
Rn	**17**(1/2/1)	**20**(1/2/1)	**23**(1/3/1)	**22**(1/2/1)	**25**(1/3/1)	**27**(1/3/1)
#<data>.B,W	**17**(1/2/1)	**20**(1/2/1)	**23**(1/3/1)	**22**(1/2/1)	**25**(1/3/1)	**27**(1/3/1)
#<data>.L	**19**(1/2/1)	**22**(1/2/1)	**25**(1/3/1)	**24**(1/2/1)	**27**(1/3/1)	**29**(1/3/1)
(An)	**19**(2/2/1)	**22**(2/2/1)	**25**(2/3/1)	**24**(2/2/1)	**27**(2/3/1)	**29**(2/3/1)
(An)+	**19**(2/2/1)	**22**(2/2/1)	**25**(2/3/1)	**24**(2/2/1)	**27**(2/3/1)	**29**(2/3/1)
–(An)	**20**(2/2/1)	**23**(2/2/1)	**26**(2/3/1)	**25**(2/2/1)	**28**(2/3/1)	**30**(2/3/1)
(d_{16},An) or (d_{16},PC)	**21**(2/3/1)	**24**(2/3/1)	**27**(2/4/1)	**26**(2/3/1)	**29**(2/4/1)	**31**(2/4/1)
(xxx).W	**20**(2/3/1)	**23**(2/3/1)	**26**(2/4/1)	**27**(2/3/1)	**28**(2/4/1)	**30**(2/4/1)
(xxx).L	**22**(2/3/1)	**25**(2/3/1)	**28**(2/4/1)	**29**(2/3/1)	**30**(2/4/1)	**32**(2/4/1)
(d_8,An,Xn) or (d_8,PC,Xn)	**23**(2/3/1)	**26**(2/3/1)	**29**(2/4/1)	**30**(2/3/1)	**31**(2/4/1)	**33**(2/4/1)
(d_{16},An,Xn) or (d_{16},PC,Xn)	**24**(2/3/1)	**27**(2/3/1)	**30**(2/4/1)	**31**(2/3/1)	**32**(2/4/1)	**34**(2/4/1)
(B)	**24**(2/3/1)	**27**(2/3/1)	**30**(2/4/1)	**31**(2/3/1)	**32**(2/4/1)	**34**(2/4/1)
(d_{16},B)	**27**(2/3/1)	**30**(2/3/1)	**33**(2/4/1)	**34**(2/3/1)	**35**(2/4/1)	**37**(2/4/1)
(d_{32},B)	**31**(2/4/1)	**34**(2/4/1)	**37**(2/5/1)	**38**(2/4/1)	**39**(2/5/1)	**41**(2/5/1)
([B],I)	**28**(3/3/1)	**31**(3/3/1)	**34**(3/4/1)	**35**(3/3/1)	**36**(3/4/1)	**38**(3/4/1)
([B],I,d_{16})	**31**(3/3/1)	**34**(3/3/1)	**37**(3/4/1)	**38**(3/3/1)	**39**(3/4/1)	**41**(3/4/1)
([B],I,d_{32})	**32**(3/4/1)	**35**(3/4/1)	**38**(3/5/1)	**39**(3/4/1)	**40**(3/5/1)	**42**(3/5/1)
([d_{16},B],I)	**31**(3/3/1)	**34**(3/3/1)	**37**(3/4/1)	**38**(3/3/1)	**39**(3/4/1)	**41**(3/4/1)
([d_{16},B],I,d_{16})	**34**(3/4/1)	**37**(3/4/1)	**40**(3/5/1)	**41**(3/4/1)	**42**(3/5/1)	**44**(3/5/1)
([d_{16},B],I,d_{32})	**35**(3/4/1)	**38**(3/4/1)	**41**(3/5/1)	**42**(3/4/1)	**43**(3/5/1)	**45**(3/5/1)
([d_{32},B],I)	**35**(3/4/1)	**38**(3/4/1)	**41**(3/5/1)	**42**(3/4/1)	**43**(3/5/1)	**45**(3/5/1)
([d_{32},B],I,d_{16})	**37**(3/4/1)	**40**(3/4/1)	**43**(3/5/1)	**44**(3/4/1)	**45**(3/5/1)	**47**(3/5/1)
([d_{32},B],I,d_{32})	**39**(3/5/1)	**42**(3/5/1)	**45**(3/6/1)	**46**(3/5/1)	**47**(3/6/1)	**49**(3/6/1)