MOTOROLA

MC68881

FLOATING-POINT COPROCESSOR USER'S MANUAL

First Edition

All rights reserved. No part of this document may be reproduced in any form or by any means without permission in writing from Motorola Inc.

This document contains information on a new product. Specifications and information herein are subject to change without notice. Motorola reserves the right to make changes to any products herein to improve functioning or design. Although the information in this document has been carefully reviewed and is believed to be reliable, Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others.

Motorola, Inc. general policy does not recommend the use of its components in life support applications where in a failure or malfunction of the component may directly threaten life or injury. Per Motorola Terms and Conditions of Sale, the user of Motorola components in life support applications assumes all risk of such use and indemnifies Motorola against all damages.

© 1985 by Motorola Inc.

PREFACE

This manual is divided into two major parts, the first (sections 2 through 6) dealing with the programmer's model of the MC68881 and the floating point instruction set that it implements. This portion of the manual is written with the assumption that the MC68881 is connected as a coprocessor to the MC68020 microprocessor; and thus, the instruction formats and syntaxes used are for such a system. A prior knowledge of M68000 assembly language conventions is quite helpful, although not required. If the MC68881 is used in a system with a main processor other than the MC68020, it is expected that the host processor will simulate the M68000 Family Coprocessor Interface requirements of the MC68881 in such a way that the programmer's model discussion in this manual will be relevant. Included in this part of the manual is a detailed description of the function of each instruction, and a section on instruction timings that can be used for program optimization, and to predict floating point arithmetic performance.

The second part of this manual (sections 7 through 11) pertains to the hardware interface of the MC68881 to the host system, and is most pertinent to system hardware designers. Like the first portion of this manual, the hardware description, for the most part, assumes that the host processor is the MC68020. Thus, bus cycle timing diagrams, interface register addressing, etc., are discussed from the viewpoint of the MC68020 hardware conventions. A prior knowledge of the MC68020 bus interface, particularly as it pertains to the M68000 Family Coprocessor Interface, is quite helpful in understanding the operation of the MC68881 bus interface. Included at the end of the manual is a section containing the AC Electrical Specifications for the bus interface, with all pertinent timing diagrams and characteristic tables.

At the end of the manual is a glossary of terms, and a list of abbreviation and acronyms used throughout this manual.

Throughout this manual, "M68000" or "M68000 Family" is used to refer to the family of devices that support the Motorola 68000 Microprocessor Family architecture. When "MC" precedes a 68xxx number, this number refers to a specific part (eg., MC68020, MC68881, etc.). The MC68881 is designed to perform as a coprocessor to any of the M68000 Family processors, although the MC68020 is currently the only device that implements the M68000 Family Coprocessor Interface in hardware.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Concluded)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS
(Continued)

LIST OF ILLUSTRATIONS
(Concluded)

LIST OF TABLES

LIST OF TABLES
(Continued)

SECTION 1
GENERAL DESCRIPTION

The MC68881 floating-point coprocessor is a full implementation of the *IEEE Standard for Binary Floating-Point Arithmetic* (proposed by Task P754) for use with the Motorola M68000 Family of microprocessors. It is implemented using VLSI technology to give systems designers the highest possible functionality in a physically small device.

Intended primarily for use as a coprocessor to the MC68020 32-bit microprocessor unit (MPU), the MC68881 provides a logical extension to the main MPU integer data processing capabilities. It does this by providing a very high performance floating-point arithmetic unit and a set of floating-point data registers that are utilized in a manner that is analogous to the use of the integer data registers. The MC68881 instruction set is a natural extension of all earlier members of the M68000 Family, and supports all of the addressing modes of the host MPU. Due to the flexible bus interface of the M68000 Family, the MC68881 can be used with any of the MPU devices of the M68000 Family, and it may also be used as a peripheral to non-M68000 processors.

The major features of the MC68881 are:

- Eight general purpose floating-point data registers, each supporting a full 80-bit extended precision real data format (a 64-bit mantissa plus a sign bit, and a 15-bit signed exponent).
- A 67-bit arithmetic unit to allow very fast calculations, with intermediate precision greater than the extended precision format.
- A 67-bit barrel shifter for high-speed shifting operations (for normalizing etc.).
- Forty-six instructions, including 35 arithmetic operations.
- Full conformation to the IEEE P754 standard, including all requirements and suggestions.
- Support of functions not defined by the IEEE standard, including a full set of trigonometric and trancendental functions.
- Seven data types: byte, word and long integers; single, double, and extended precision real numbers; and packed binary coded decimal string real numbers.
- Twenty-two constants available in the on-chip ROM, including π, e, and powers of 10.
- Virtual memory/machine operations.
- Efficient mechanisms for procedure calls, context switches, and interrupt handling.
- Fully concurrent instruction execution with the main processor.
- Use with any host processor, on an 8-, 16-, or 32-bit data bus.

1.1 THE COPROCESSOR CONCEPT

The MC68881 functions as a coprocessor in systems where the MC68020 is the main processor via the M68000 coprocessor interface. It functions as a peripheral processor in systems where the main processor is the MC68000, MC68008, MC68010, or MC68012.

The MC68881 utilizes the M68000 Family coprocessor interface to provide a logical extension of the MC68020 registers and instruction set in a manner which is transparent to the programmer. The programmer perceives the MC68020/MC68881 execution model as if both devices were implemented on one chip.

A fundamental goal of the M68000 Family coprocessor interface is to provide the programmer with an execution model based upon sequential instruction execution by the MC68020 and the MC68881. For optimum performance, however, the coprocessor interface allows concurrent operations in the MC68881 with respect to the MC68020 whenever possible. In order to simplify the programmer's model, the coprocessor interface is designed to emulate, as closely as possible, non-concurrent operation between the MC68020 and the MC68881.

The MC68881 is a non-DMA type coprocessor which uses a subset of the general purpose coprocessor interface supported by the MC68020. Features of the interface implemented in the MC68881 are as follows:

- The main processor(s) and MC68881 communicate via standard M68000 bus cycles.
- The main processor(s) and MC68881 communications are not dependent upon the instruction sets or internal details of the individual devices (e.g., instruction pipes or caches, addressing modes).
- The main processor(s) and MC68881 may operate at different clock speeds.
- MC68881 instructions utilize all addressing modes provided by the main processor; all effective addresses are calculated by the main processor at the request of the coprocessor.
- All data transfers are performed by the main processor at the request of the MC68881; thus memory management, bus errors, address errors, and bus arbitration function as if the MC68881 instructions are executed by the main processor.
- Overlapped (concurrent) instruction execution enhances throughput while maintaining the programmer's model of sequential instruction execution.
- Coprocessor detection of exceptions which require a trap to be taken are serviced by the main processor at the request of the MC68881; thus exception processing functions as if the MC68881 instructions were executed by the main processor.
- Support of virtual memory/virtual machine systems is provided via the FSAVE and FRESTORE instructions.
- Up to eight coprocessors may reside in a system simultaneously; multiple coprocessors of the same type are also allowed.
- Systems may use software emulation of the MC68881 without reassembling or relinking user software.

1.2 HARDWARE OVERVIEW

The MC68881 is a high performance floating-point device designed to interface with the MC68020 as a coprocessor. This device fully supports the MC68020 virtual machine architecture, and is implemented in HCMOS, Motorola's low power, small geometry process. This process allows CMOS and HMOS (high density NMOS) gates to be combined on the same device. CMOS structures are used where speed and low power is required, and HMOS structures are used where minimum silicon area is desired. Using this technology enables the MC68881 to be very fast while consuming little power, and still have a reasonably small die size.

With some performance degradation, the MC68881 can also be used as a peripheral processor in systems where the MC68020 is not the main processor (e.g., MC68000, MC68008, MC68010, MC68012). The configuration of the MC68881 as a peripheral processor or coprocessor may be completely transparent to user software (i.e., the same object code may be executed in either configuration).

The architecture of the MC68881 appears to the user as a logical extension of the M68000 Family architecture. Because of the coupling of the coprocessor interface, the MC68020 programmer can view the MC68881 registers as though the registers are resident in the MC68020. Thus, a MC68020/MC68881 device pair appears to be one processor that supports seven floating-point and integer data types, and has eight integer data registers, eight address registers, and eight floating-point data registers.

The MC68881 programming model is shown in Figures 1-1 through 1-6, and consists of the following:

- Eight 80-bit floating-point data registers (FP0-FP7). These registers are analogous to the integer data registers (D0-D7) and are completely general purpose (i.e., any instruction may use any register).
- A 32-bit control register that contains enable bits for each class of exception trap, and mode bits to set the user-selectable rounding and precision modes.
- A 32-bit status register that contains floating-point condition codes, quotient bits, and exception status information.
- A 32-bit instruction address register that contains the main processor memory address of the last floating-point instruction that was executed. This address is used in exception handling to locate the instruction that caused the exception.

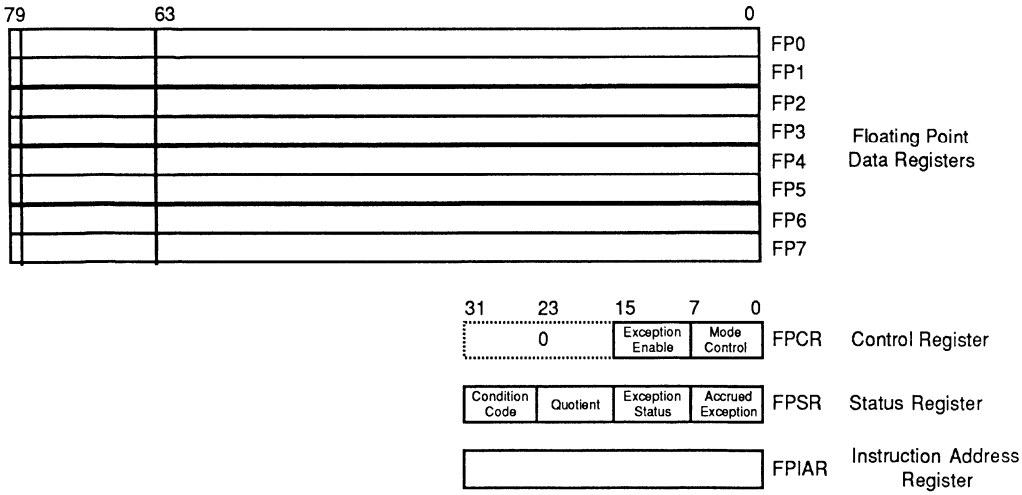


Figure 1-1. MC68881 Programming Model

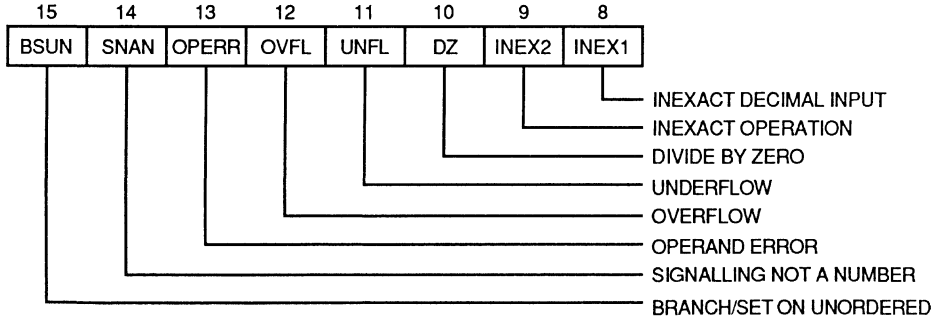


Figure 1-2. Exception Status/Enable Byte

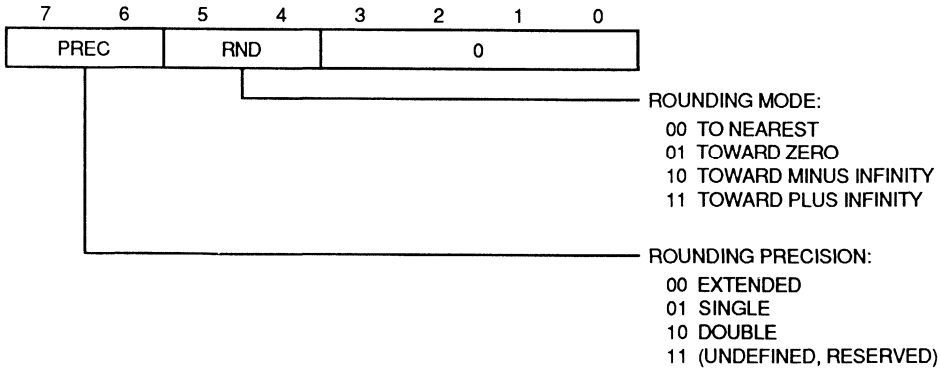


Figure 1-3. Mode Control Byte

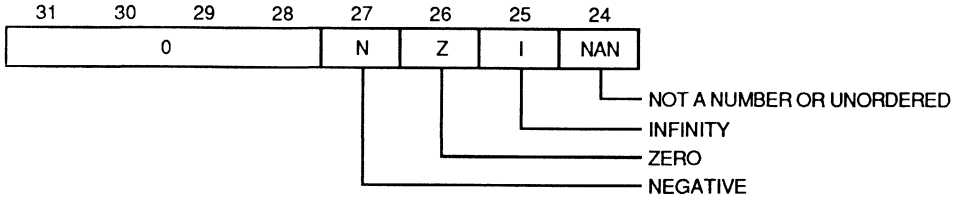


Figure 1-4. Condition Code Byte

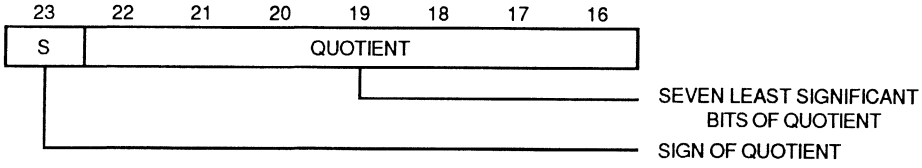


Figure 1-5. Quotient Byte

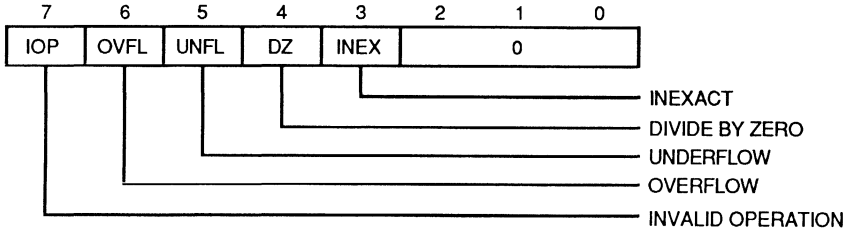


Figure 1-6. Accrued Exception Byte

The connection between the MC68020 and the MC68881 is a simple extension of the M68000 bus interface. The MC68881 is connected as a coprocessor to the MC68020, and the selection of the MC68881 is based upon a chip select (CS), which is decoded from the MC68020 function codes and address bus. Figure 1-7 illustrates the MC68881/MC68020 configuration.

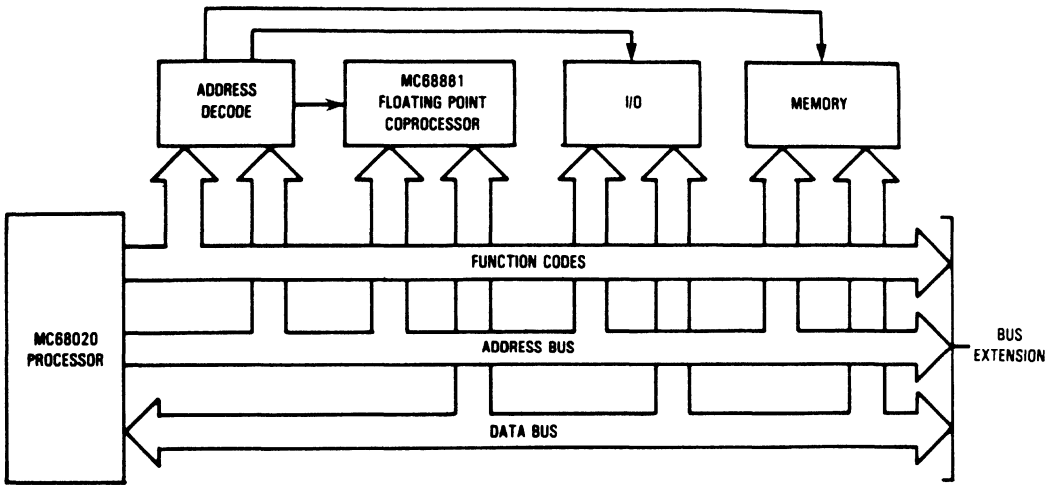


Figure 1-7. Typical Coprocessor Configuration

As shown in Figure 1-8, the MC68881 is internally divided into three processing elements; the bus interface unit (BIU), the execution control unit (ECU), and the microcode control unit (MCU). The BIU communicates with the MC68020, and the ECU and MCU execute all MC68881 instructions.

The BIU contains the coprocessor interface registers, and the 32-bit control, status, and instruction address registers. In addition to these registers, the register select and DSACK timing control logic is contained in the BIU. Finally, the status flags used to monitor the status of communications with the main processor are contained in the BIU.

The eight 80-bit floating-point data registers (FP0-FP7) are located in the ECU. In addition to these registers, the ECU contains a high-speed 67-bit arithmetic unit used for both mantissa and exponent calculations, a barrel shifter that can shift from 1 bit to 67 bits in one machine cycle, and ROM constants (for use by the internal algorithms or user programs).

The MCU contains the clock generator, a two-level microcoded sequencer that controls the ECU, the microcode ROM, and self-test circuitry. The built-in self-test capabilities of the MC68881 enhance reliability and ease manufacturing requirements; however, these diagnostic functions are not available to the user.

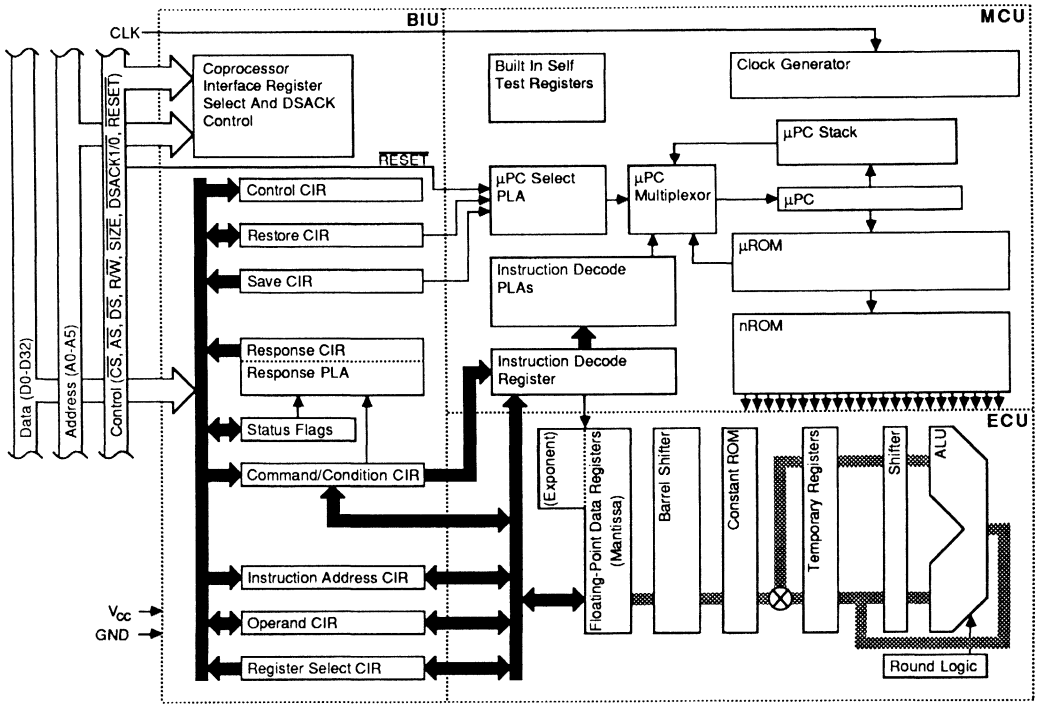


Figure 1-8. MC68881 Simplified Block Diagram

1.2.1 Bus Interface Unit

All communications between the MC68020 and the MC68881 occur via standard M68000 Family bus transfers. The MC68881 is designed to operate on 8-, 16-, or 32-bit data buses.

The MC68881 contains a number of coprocessor interface registers (CIRs) which are addressed in the same manner as memory by the main processor. The M68000 Family coprocessor interface is implemented via a protocol of reading and writing to these registers by the main processor. The MC68020 implements this general purpose coprocessor interface protocol in hardware and microcode.

When the MC68020 detects a typical MC68881 instruction, the MC68020 writes the instruction to the memory-mapped command CIR, and reads the response CIR. In this response, the BIU encodes requests for any additional action required of the MC68020 on behalf of the MC68881. For example, the response may request that the MC68020 fetch an operand from the evaluated effective address and transfer the operand to the operand CIR. Once the MC68020 fulfills the coprocessor request(s), the MC68020 is free to fetch and execute subsequent instructions.

A key concern in a coprocessor interface that allows concurrent instruction execution is synchronization during main processor and coprocessor communication. If a subsequent instruction is written to the MC68881 before the ECU has completed execution of the previous instruction, the response instructs the MC68020 to wait. Thus, the choice of concurrent or nonconcurrent instruction execution is determined on an instruction-by-instruction basis by the coprocessor.

The only difference between a coprocessor bus transfer and any other bus transfer is that the MC68020 issues a function code to indicate the CPU address space during the cycle (the function codes are generated by the M68000 Family processors to identify eight separate address spaces). Thus, the memory-mapped coprocessor interface registers do not infringe upon instruction or data address spaces. The MC68020 places a coprocessor ID field from the coprocessor instruction onto three of the upper address lines during coprocessor accesses. This ID, along with the CPU address space function code, is decoded to select one of eight coprocessors in the system.

Since the coprocessor interface protocol is based solely on bus transfers, the protocol is easily emulated by software when the MC68881 is used as a peripheral with any processor capable of memory-mapped I/O over an M68000 style bus. When used as a peripheral processor with the 8-bit MC68008 or the 16-bit MC68000, MC68010, or MC68012, all MC68881 instructions are trapped by the main processor to an exception handler at execution time. Thus, the software emulation of the coprocessor interface protocol can be totally transparent to the user. The system can be quickly upgraded by replacing the main processor with an MC68020 without changes to the user software.

Since the bus is asynchronous, the MC68881 need not run at the same clock speed as the main processor. Total system performance may therefore be customized. For example, a system requiring very fast floating-point arithmetic with relatively slow integer arithmetic can be designed with an inexpensive main processor and a fast MC68881.

1.2.2 Coprocessor Interface

The M68000 Family coprocessor interface is an integral part of the MC68881 and MC68020 design, with the interface tasks shared between the two. The interface is fully compatible with all present and future M68000 Family products. Tasks are partitioned such that the MC68020 does not have to decode coprocessor instructions, and the MC68881 does not have to duplicate main processor functions such as effective address evaluation.

This partitioning provides an orthogonal extension of the instruction set by permitting MC68881 instructions to utilize all MC68020 addressing modes and to generate execution time exception traps. Thus, from the programmer's view, the CPU and coprocessor appear to be integrated onto a single chip. While the execution of the great majority of MC68881 instructions may be overlapped with the execution of MC68020 instructions, concurrency is completely transparent to the programmer. The MC68020 single-step and program flow (trace) modes are fully supported by the MC68881 and the M68000 Family coprocessor interface.

While the M68000 Family coprocessor interface permits coprocessors to be bus masters, the MC68881 is never a bus master. The MC68881 requests that the MC68020 fetch all operands and store all results. In this manner, the MC68020 32-bit data bus provides high speed transfer of floating-point operands and results while simplifying the design of the MC68881.

Since the coprocessor interface is based solely upon bus cycles and the MC68881 is never a bus master, the MC68881 can be placed on either the logical or physical side of the system memory management unit. This provides a great deal of flexibility in the system design.

The virtual machine architecture of the MC68020 is supported by the coprocessor interface and the MC68881 through the FSAVE and FRESTORE instructions. If the MC68020 detects a page fault and/or a task time out, the MC68020 can force the MC68881 to stop whatever operation is in process at any time (even in the middle of the execution of an instruction) and save the MC68881 internal state in memory.

The size of the saved internal state of the MC68881 is dependent upon what the ECU is doing at the time that the FSAVE is executed. If the MC68881 is in the reset state when the FSAVE instruction is received, only one word of state is transferred to memory, which may be examined by the operating system to determine that the coprocessor programmer's model is empty. If the coprocessor is idle when the save instruction is received, only a few words of internal state are transferred to memory. If the MC68881 is in the middle of executing an instruction, it may be necessary to save the entire internal state of the machine. Instructions that can complete execution in less time than it would take to save the larger state in mid-instruction are allowed to complete execution and then save the idle state. Thus the size of the saved internal state is kept to a minimum. The ability to utilize several internal state sizes greatly reduces the average context switching time.

The FRESTORE instruction permits reloading of an internal state that was saved earlier, and continues any operation that was previously suspended. Restoring of the reset internal state functions just like a hardware reset to the MC68881 in that defaults are re-established.

1.3 OPERAND DATA FORMATS

The MC68881 supports the following data formats:

- Byte Integer (B)
- Word Integer (W)
- Long Word Integer (L)
- Single Precision Real (S)
- Double Precision Real (D)
- Extended Precision Real (X)
- Packed Decimal String Real (P)

The capital letters contained in parentheses denote suffixes added to instructions in the assembly language source to specify the data format to be used.

1.3.1 Integer Data Formats

The three integer data formats (byte, word, and long word) are the standard data formats supported in the M68000 Family architecture. Whenever an integer is used in a floating-point operation, the integer is automatically converted by the MC68881 to an extended precision floating-point number before being used. For example, to add an integer constant of five to the number contained in floating-point data register 3 (FP3), the following instruction can be used:

```
FADD.W  #5,FP3
```

(The Motorola assembler syntax "#" is used to denote immediate addressing.)

The ability to effectively use integers in floating-point operations saves user memory since an integer representation of a number, if representable, is usually smaller than the equivalent floating-point representation.

1.3.2 Floating-Point Data Formats

The floating-point data formats, single precision (32-bits) and double precision (64-bits) are as defined by the IEEE standard. These are the main floating-point formats and should be used for most calculations involving real numbers. Table 1-1 lists the exponent and mantissa size for single, double, and extended precision. The exponent is biased, and the mantissa is in sign and magnitude form. Since single and double precision require normalized numbers, the most significant bit of the mantissa is implied as a one and is not included, thus giving one extra bit of precision.

Table 1-1. Exponent and Mantissa Sizes

Data Format	Exponent Bits	Mantissa Bits
Single	8	23(+1)
Double	11	52(+1)
Extended	15	64

The extended precision data format is also in conformance with the IEEE standard, but the standard does not specify this format to the bit level as it does for single and double precision. The memory format on the MC68881 consists of 96 bits (three long words). Only 80 bits are actually used, the other 16 bits are for future expandability and for long-word alignment of floating-point data structures. Extended format has a 15-bit exponent, a 64-bit mantissa, and a 1-bit mantissa sign.

Extended precision numbers are intended for use as temporary variables, intermediate values, or in places where extra precision is needed. For example, a compiler might select extended precision arithmetic for evaluation of the right side of an equation with mixed sized data and then convert the answer to the data type on the left side of the equation. It is anticipated that extended precision data will not be stored in large arrays, due to the amount of memory required by each number.

1.3.3 Packed Decimal String Real Data Format

The packed decimal data format allows packed BCD strings to be input to and output from the MC68881. The strings consist of a 3-digit base 10 exponent and a 17-digit base 10 mantissa. Both the exponent and mantissa have a separate sign bit. All digits are packed BCD, such that an entire string fits in 96 bits (three long words). As is the case with all data formats, when packed BCD strings are input to the MC68881, the strings are automatically converted to extended precision real values. This allows packed BCD numbers to be used as inputs to any operation. For example:

```
FADD.P   #-6.023E+24,FP5
```

BCD numbers can be output from the MC68881 in a format readily used for printing by a program generated by a high-level language compiler. For example:

```
FMOVE.P  FP3,BUFFER{#-5}
```

instructs the MC68881 to convert the floating-point data register 3 (FP3) contents into a packed BCD string with five digits to the right of the decimal point (FORTRAN F format).

1.3.4 Data Format Summary

All data formats described above are supported orthogonally by all arithmetic and transcendental operations, and by all appropriate MC68020 addressing modes. For example, all of the following are legal instructions:

```
FADD.B    #0,FP0
FADD.W    D2,FP3
FADD.L    BIGINT,FP7
FADD.S    #3.14159,FP5
FADD.D    (SP)+,FP6
FADD.X    [(TEMP_PTR,A7)],FP3
FADD.P    #1.23E25,FP0
```

Most on-chip calculations are performed to extended precision format, and the eight floating-point data registers always contain extended precision values. All data used in an operation is converted to extended precision by the MC68881 before the specific operation is performed, and all results are in extended precision. This ensures maximum accuracy without sacrificing performance.

Refer to Figure 1-9 for a summary of the memory formats for the seven data formats supported by the MC68881.

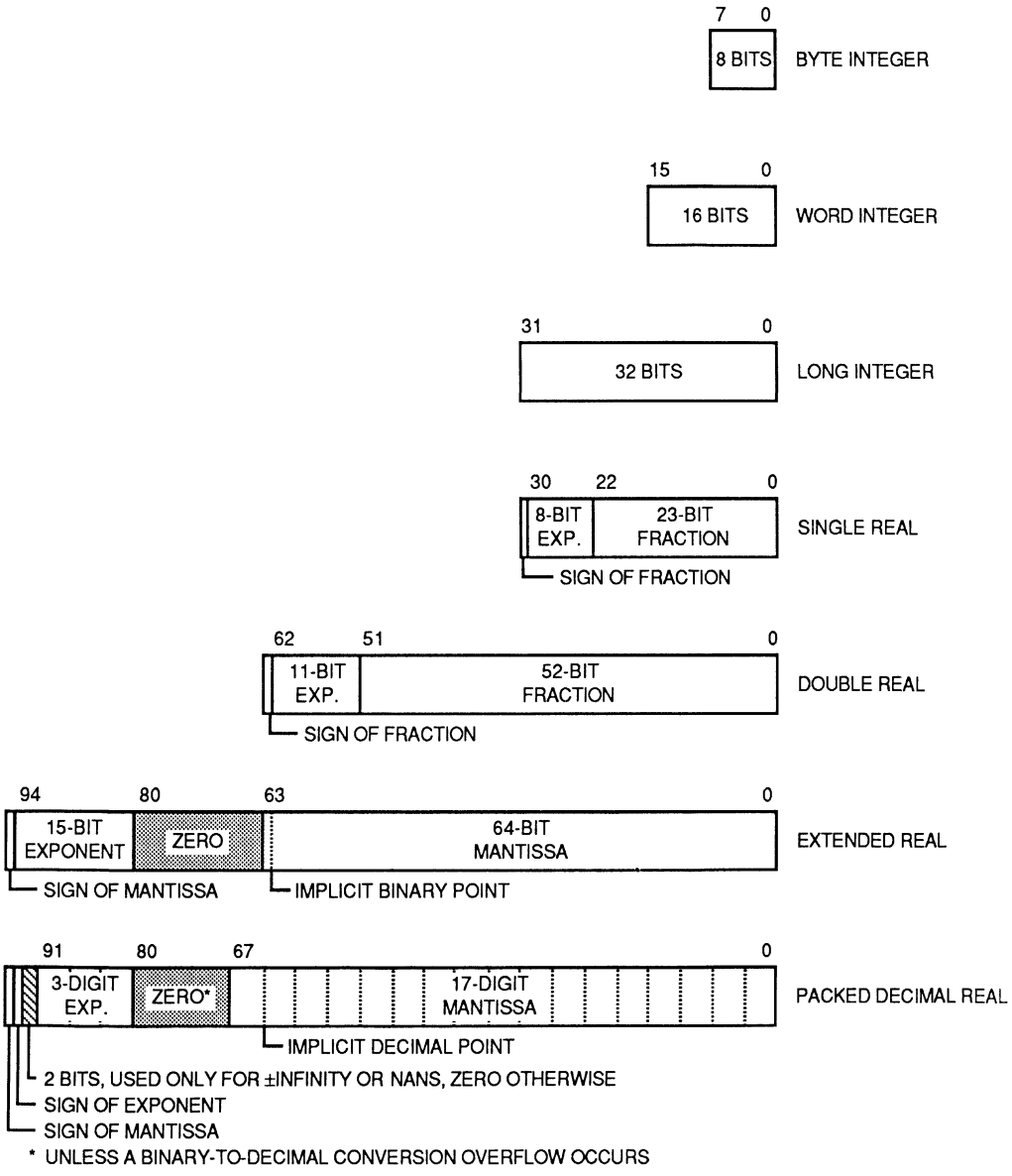


Figure 1-9. MC68881 Data Format Summary

1.4 INSTRUCTION SET

The MC68881 instruction set is organized into six major classes:

1. Moves between the MC68881 and memory or the MC68020 (in and out),
2. Move multiple registers (in and out),
3. Monadic operations,
4. Dyadic operations,
5. Branch, set, or trap conditionally, and
6. Miscellaneous.

1.4.1 Moves

On all moves from memory (or from an MC68020 data register) to the MC68881, data is converted from the source data format to the internal extended precision format.

On all moves from the MC68881 to memory (or to an MC68020 data register), data is converted from the internal extended precision format to the destination data format.

Note that data movement instructions perform arithmetic operations, since the result is always rounded to the precision selected in the FPCR mode control byte. The result is rounded using the selected rounding mode, and is checked for overflow and underflow.

The syntax for the move is:

FMOVE.<fmt>	<ea>,FPn	Move to MC68881
FMOVE.<fmt>	FPm,<ea>	Move from MC68881
FMOVE.X	FPm,FPn	Move within MC68881

where:

<ea> is an MC68020 effective address operand and .<fmt> is the data format size. FPm and FPn are floating-point data registers.

1.4.2 Move Multiples

The floating-point move multiple instructions on the MC68881 are much like the integer counterparts on the M68000 Family processors. Any set of the floating-point registers FP0 through FP7 can be moved to or from memory with one instruction. These registers are always moved as 96-bit extended data with no conversion (hence no possibility of conversion errors). Some move examples are as follows:

```
FMOVEM      <ea>,FP0-FP3/FP7
FMOVEM      FP2/FP4/FP6,<ea>
```

Move multiples are useful during context switches and interrupts to save or restore the state of a program. These moves are also useful at the start and end of a procedure to save and restore the calling routine's register set . In order to reduce procedure call overhead, the list of registers to be saved or restored can be contained in a data register. This allows run-time optimization by allowing a called routine to save as few registers as possible. Note that no rounding or overflow/underflow checking is performed by these operations.

1.4.3 Monadic Operations

1

Monadic operations have one operand. This operand may be in a floating-point data register, memory, or in an MC68020 data register. The result is always stored in a floating-point data register. For example, the syntax for square root is:

```
FSQRT.<fmt>    <ea>,FPn    or,
FSQRT.X        FPm,FPn     or,
FSQRT.X        FPn
```

The MC68881 monadic operations available are as follows:

FABS	Absolute Value	FLOG2	Log Base 2
FACOS	Arc Cosine	FLOGN	Log Base e
FASIN	Arc Sine	FLOGNP1	Log Base e of (x+1)
FATAN	Arc Tangent	FNEG	Negate
FATANH	Hyperbolic Arc Tangent	FSIN	Sine
FCOS	Cosine	FSINCOS	Simultaneous Sine and Cosine
FCOSH	Hyperbolic Cosine	FSINH	Hyperbolic Sine
FETOX	e to the x Power	FSQRT	Square Root
FETOXM1	e to the x Power – 1	FTAN	Tangent
FGETEXP	Get Exponent	FTANH	Hyperbolic Tangent
FGETMAN	Get Mantissa	FTENTOX	10 to the x Power
FINT	Integer Part	FTST	Test
FINTRZ	Integer Part (Truncated)	FTWOTOX	2 to the x Power
FLOG10	Log Base 10		

1.4.4 Dyadic Operations

Dyadic operations have two input operands. The first input operand comes from a floating-point data register, memory, or an MC68020 data register. The second input operand comes from a floating-point data register. The destination is the same floating-point data register used for the second input. For example, the syntax for add is:

```
FADD.<fmt>     <ea>,FPn    or,
FADD.X         FPm,FPn
```

The MC68881 dyadic operations available are as follows:

FADD	Add	FREM	IEEE Remainder
FCMP	Compare	FSCALE	Scale Exponent
FDIV	Divide	FSGLDIV	Single Precision Divide
FMOD	Modulo Remainder	FSGLMUL	Single Precision Multiply
FMUL	Multiply	FSUB	Subtract

1.4.5 Branch, Set, and Trap-On Condition

The floating-point branch, set, and trap on condition instructions implemented by the MC68881 are similar to the equivalent integer instructions of the M68000 Family processors, except that more conditions exist due to the special values in IEEE floating-point arithmetic. When a conditional instruction is executed, the MC68881 performs the necessary condition checking and tells the MC68020 whether the condition is true or false; the MC68020 then takes the appropriate action. Since the MC68881 and MC68020 are closely coupled, the floating-point branch operations executed by the pair are very fast.

The MC68881 conditional operations are:

FBcc	Branch
FDBcc	Decrement and Branch
FScc	Set According to Condition
FTRAPcc	Trap-on Condition (with an Optional Parameter)

where:

cc is one of the 32 floating-point conditional test specifiers as given in **3.3 Conditional Test Definitions**.

1.4.6 Miscellaneous Instructions

Miscellaneous instructions include moves to and from the status, control, and instruction address registers. Also included are the virtual memory/machine FSAVE and FRESTORE instructions that save and restore the internal state of the MC68881.

FMOVE	<ea>,FPcr	Move to Control Register(s)
FMOVE	FPcr,<ea>	Move from Control Register(s)
FSAVE	<ea>	Virtual Machine State Save
FRESTORE	<ea>	Virtual Machine State Restore

1.5 ADDRESSING MODES

The MC68881 does not perform address calculations. This satisfies the criterion that an M68000 Family coprocessor must not depend on certain features or capabilities that may or may not be implemented by a given main processor. Thus, if the MC68881 instructs the MC68020 to transfer an operand via the coprocessor interface, the MC68020 will perform the addressing mode calculations requested in the instruction. In this case, the instruction is encoded specifically for the MC68020, and the execution of the MC68881 is not dependent on that encoding, but only on the value of the command word written to the MC68881 by the main processor.

This interface is quite flexible and allows any addressing mode to be used with floating-point instructions. For the M68000 Family, these addressing modes include immediate, postincrement, predecrement, data or address register direct, and the indexed/indirect addressing modes of the MC68020. Some addressing modes are restricted for some instructions in keeping with the M68000 Family architectural definitions (e.g., PC relative addressing is not allowed for a destination operand).

The orthogonal instruction set of the MC68881, along with the flexible branches and addressing modes, allows a programmer writing assembly language code, or a compiler writer generating object or source code for the MC68020/MC68881 device pair, to think of the MC68881 as though the MC68881 is part of the MC68020. There are no special restrictions imposed by the coprocessor interface, and floating-point arithmetic is coded exactly like integer arithmetic.

SECTION 2
PROGRAMMING MODEL

This section describes the registers contained in the MC68881 programming model and the seven data formats supported by the instruction set. The notation used to refer to the registers conforms to the Motorola assembler syntax. Furthermore, the memory data format descriptions assume that the main processor is a member of the M68000 Family.

2.1 PROGRAMMIMG MODEL

Figure 2-1 shows a pictorial representation of the registers in the MC68881 programming model. The following paragraphs describe each group of registers.

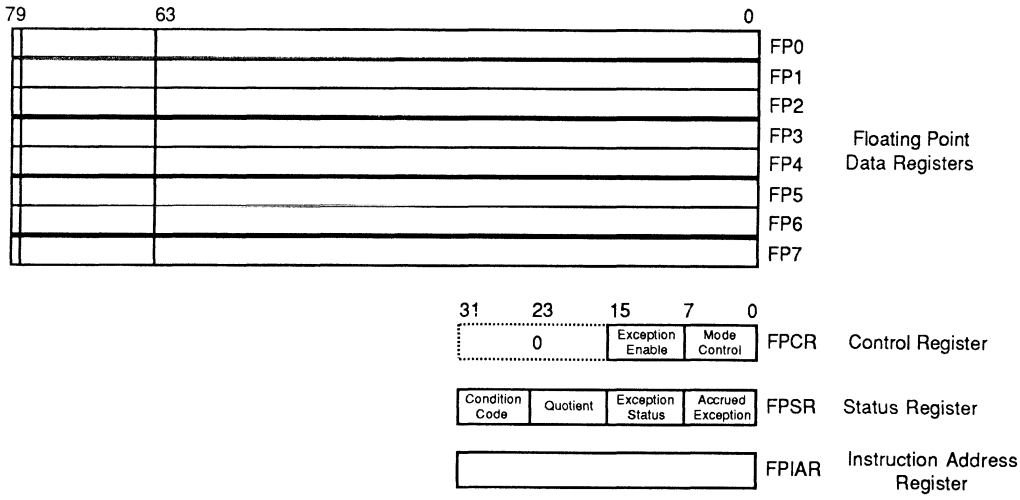


Figure 2-1. MC68881 Programming Model

2.1.1 Floating-Point Data Registers

The eight 80-bit floating-point data registers (FP0-FP7) are analogous to the integer data registers (D0-D7) of all M68000 Family processors. Floating-point data registers always contain extended precision numbers. The data format used is identical to the extended precision data format described in Table 2-3, except that the reserved (unused) 16 bits are deleted. All external operands, regardless of the data format, are converted to extended precision values before any calculation or storage into a floating-point data register is performed.

A reset function or a null state size restore operation sets FP0-FP7 to positive non-signaling not-a-numbers (NANs).

2.1.2 Floating-Point Control Register

The 32-bit floating-point control register (FPCR) contains an exception enable byte, which enables/disables traps for each class of floating-point exceptions, and a mode byte which sets the user selectable modes.

The control register can be read or written to by the user. Bits 16 through 31 are reserved for future definition by Motorola, will always read as zero, and are ignored during write operations (but should be zero for future compatibility). This register is cleared by the reset function or a null state size restore operation When cleared, this register provides the IEEE standard defaults.

2.1.2.1 FPCR EXCEPTION ENABLE BYTE. The exception enable byte (see Figure 2-2) contains one bit for each floating-point exception class. The user may separately enable traps for each class of floating-point exceptions.

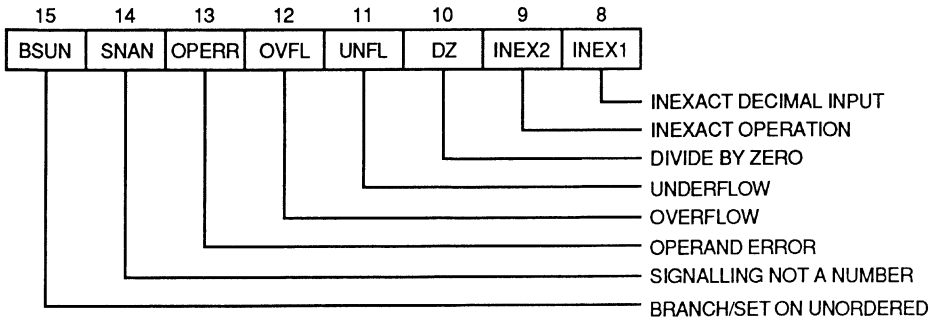


Figure 2-2. MC68881 FPCR Exception Enable Byte

If a bit in the status register exception byte is set by the MC68881 and the corresponding bit in the control register ENABLE byte is also set, an exception will be taken to a specific vector address corresponding to the exception. A user write of the control register ENABLE byte which enables a class of floating-point exceptions will not cause a trap to be taken due to previously generated floating-point exceptions, regardless of the value in the status register exception byte.

The eight floating-point exception classes shown in Figure 2-2 are described in greater detail in **4.1.1 Instruction Exceptions.** Note that the bits in the FPSR exception byte and the FPCR enable byte are in the same positions within each byte.

Dual and triple exceptions can be generated by a single instruction execution in a few cases. When multiple exceptions occur with traps enabled for more than one exception class, the highest priority exception will be taken; the lower priority exceptions will never be reported or taken. It is the responsibility of the exception handler routine to check for multiple exceptions. The bits of the ENABLE byte are organized in decreasing priority, left to right, i.e., BSUN is the highest priority and INEX1 is the lowest priority. The only multiple exception possibilities are:

SNAN and INEX1
OPERR and INEX2
OPERR and INEX1
OVFL and INEX2 and/or INEX1
UNFL and INEX2 and/or INEX1

2.1.2.2 FPCR MODE CONTROL BYTE. The MODE byte (see Figure 2-3) controls user selectable rounding modes and rounding precisions. A zero in this byte selects the IEEE defaults.

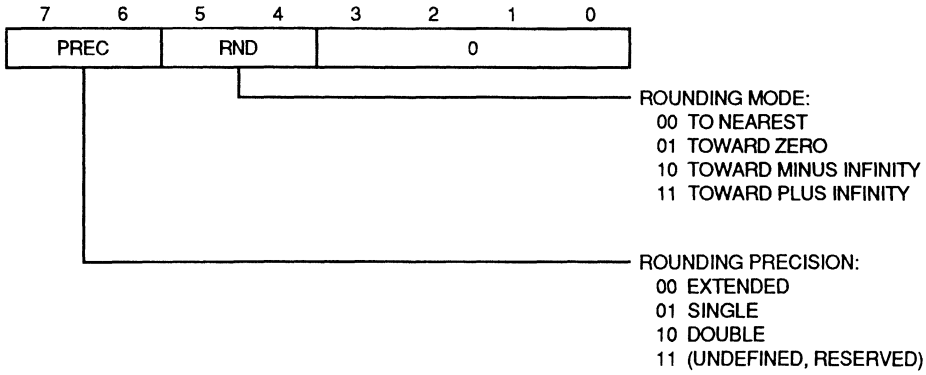


Figure 2-3. MC68881 FPCR Mode Control Byte

The rounding mode is used to determine how inexact results should be rounded. Round to the nearest specifies that the nearest number to the infinitely precise result should be selected as the rounded value. In case of a tie, the even result is selected. Round toward zero chops the result. Round toward plus infinity always rounds numbers towards plus infinity. Round toward minus infinity always rounds numbers towards minus infinity. See **4.1.2.7 INEXACT RESULT** for a detailed description of the rounding algorithm that is used.

The rounding precision selects where the rounding will occur in the mantissa. For extended precision, the result is rounded to a 64-bit boundary. A single precision result is rounded to a 24-bit boundary, and a double precision result is rounded to a 53-bit boundary.

Note that the rounding precisions of single and double are provided for emulation of machines that only support those precisions. When the MC68881 performs any operation, the calculation is carried out using extended precision inputs and an intermediate result calculated as if to produce infinite precision. After the calculation is complete, this intermediate result is rounded to the selected precision and stored in the destination.

If the destination is a floating-point data register, the stored value will be in the extended precision format rounded to the precision specified by the PREC bits This means that all mantissa bits beyond the selected precision are zero after the rounding operation. Also, the exponent value will be in the correct range for the single or double precision format, although it is stored in extended precision format.

If the destination is a memory location, the PREC bits are ignored. In this case, a number in the extended precision format is taken from the source floating-point data register, rounded to the destination format precision, and written to memory.

Since the single and double precision rounding modes are considered to be emulation modes, the execution speed of all instructions is degraded significantly when these modes are used. However, by using these modes, the result obtained by the MC68881 will be the same as any other machine that conforms to the IEEE standard but does not support extended precision calculations. Note that the result obtained by performing a series of operations with the rounding mode set to single or double precision may not be the same as the result of performing the same operations in the extended precision mode and then storing the final result in the single or double precision format.

2.1.3 Floating-Point Status Register

The floating-point status register (FPSR) contains a floating-point condition code byte, a floating-point exception status byte, quotient bits, and a floating-point accrued exception byte. All bits in the FPSR can be read or written to by the user. Execution of most floating-point instructions will modify parts of this register.

This register is cleared by the reset function or a null state size restore operation.

2.1.3.1 FPSR FLOATING-POINT CONDITION CODE BYTE. The floating-point condition code (FPCC) byte (see Figure 2-4) contains four condition code bits which are set at the end of all arithmetic instructions involving the floating-point data registers, except for the FMOVE FPm, <ea>, move multiple floating-point data register and move system control register instructions

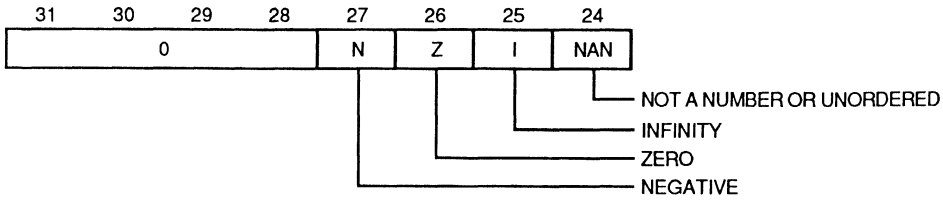


Figure 2-4. MC68881 FPSR Condition Code Byte

The operation result data type determines how the four condition code bits are set. Table 2-1 lists the condition code bit settings for each result data type. Note that of the 16 possible combinations of the condition code bits, the MC68881 generates only eight combinations. This is due to the mutually exclusive nature of the data types described by the condition code bits. Loading the FPCC byte with one of the other condition code bit combinations and performing a conditional instruction may produce an unexpected branch condition.

Table 2-1. Condition Code vs Result Data Type

N	Z	I	NAN	Result Data Type
0	0	0	0	+ Normalized or Denormalized
1	0	0	0	– Normalized or Denormalized
0	1	0	0	+ 0
1	1	0	0	– 0
0	0	1	0	+ Infinity
1	0	1	0	– Infinity
0	0	0	1	+ NAN
1	0	0	1	– NAN

The IEEE standard defines the following four conditions, and only requires the generation of the condition codes as a result of a floating-point compare operation. In addition to this requirement, the MC68881 can test these conditions at the end of any operation that affects the condition codes.

EQ Equal To
GT Greater Than
LT Less Than
UN Unordered

An unordered condition occurs when one or both of the operands in a floating-point compare operation is a NAN. For purposes of the floating-point conditional branch, set byte on condition, decrement and branch on condition, and trap on condition instructions, the MC68881 logically combines the four condition codes to form the IEEE conditions based on the following equations:

EQ = Z

GT = N ∧ NAN ∧ Z

LT = N ∧ NAN ∧ Z

UN = NAN

where:

"∧" = logical AND

Note that the setting of the MC68881 condition codes is independent of the operation executed; the condition codes simply indicate the data type of the result generated. The IEEE defined conditions can always be derived from the data type of the result. This is slightly different from other M68000 data types, where the setting of the integer condition codes is dependent upon the operation executed as well as the result.

The MC68881 implements in hardware the four floating-point condition code bits described above instead of the four IEEE defined conditions, to aid programmers of floating-point subroutine libraries (the IEEE conditions are derived by an instruction when needed). For example, the programmer of a complex arithmetic multiply subroutine will prefer to handle "special" data types, such as zeros, infinities, or NANs, separately from "normal" data types. The MC68881 condition codes allow such users to efficiently detect and handle these "special" values.

2.1.3.2 FPSR QUOTIENT BYTE. The quotient byte (see Figure 2-5) is set at the completion of the modulo (FMOD) or IEEE remainder (FREM) instructions. This byte contains the seven least significant bits of the quotient (unsigned) and the sign of the entire quotient.

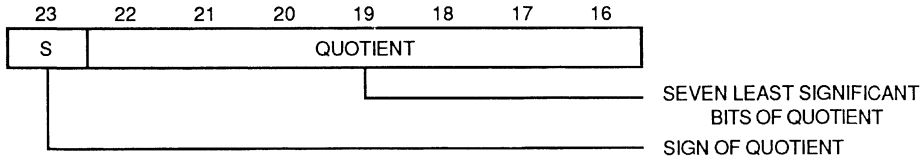


Figure 2-5. MC68881 FPSR Quotient Byte

The quotient bits can be used in argument reduction for transcendentals and other functions. For example, seven bits are more than enough to determine in which quadrant of a circle an operand resides. The quotient bits remain set until they are cleared by the user, or until another FMOD or FREM instruction is executed.

2.1.3.3 FPSR EXCEPTION STATUS BYTE. The exception status (EXC) byte (see Figure 2-6) contains a bit for each floating-point exception which may have occurred during the last arithmetic instruction or move operation.

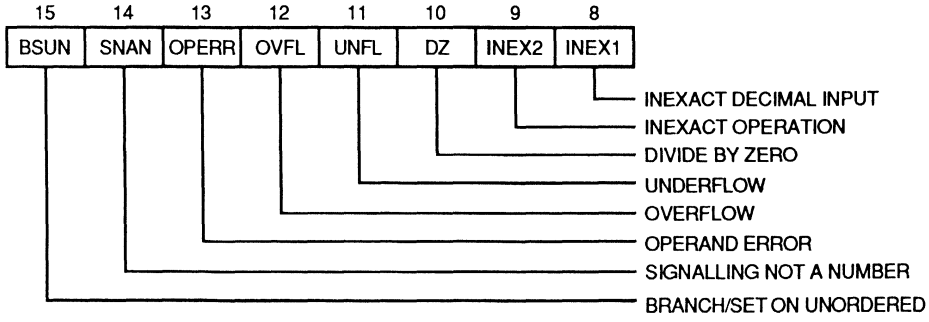


Figure 2-6. MC68881 FPSR Exception Status Byte

This byte is cleared by the MC68881 at the start of most operations; operations which cannot generate any floating-point exceptions (the FMOVEM and FMOVE control register instructions) do not clear this byte. This byte can be used by an exception handler to determine which floating-point exception(s) caused a trap.

If a bit is set by the MC68881 in the EXC byte and the corresponding bit in the ENABLE byte is also set, an exception will be signaled to the main processor. When a floating-point exception is detected by the MC68881, the corresponding bit in the EXC byte will be set, even if the trap for that exception class is disabled. (A user write operation to the status register, which sets a bit in the EXC byte, will not cause a trap to be taken regardless of the value in the ENABLE byte.)

Note that the bits in the status EXC byte and control ENABLE byte are in the same bit positions within each byte. The eight floating-point exception classes are described in greater detail in **4.1.1 Exception Vectors.**

2.1.3.4 FPSR ACCRUED EXCEPTION BYTE. The accrued exception (AEXC) byte (see Figure 2-7) contains the five exception bits required by the IEEE standard for trap disabled operation. These exceptions are logical combinations of the bits in the EXC byte. The AEXC byte contains the history of all floating-point exceptions which have occurred since the user last cleared the AEXC byte. In normal operations, only the user will clear this byte by writing to the status register. The AEXC byte is cleared by the MC68881 only by a reset or a null state size restore operation.

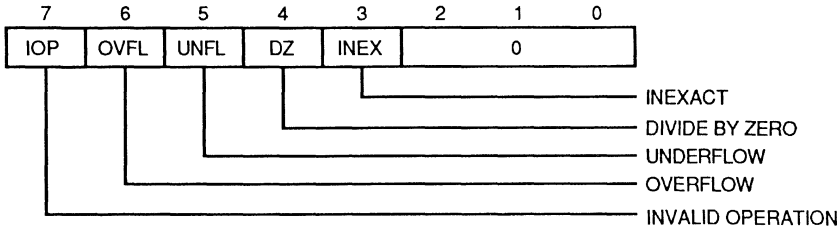


Figure 2-7. MC68881 FPSR Accrued Exception Byte

Many users will elect to disable traps for all or part of the floating-point exception classes. To allow these users to avoid polling the EXC byte after each floating-point instruction, the AEXC byte is provided. At the end of most operations (all but the FMOVEM and FMOVE control register instructions), the bits in the EXC byte are logically combined to form an AEXC value which is logically ORed into the existing AEXC byte. This creates "sticky" floating-point exception bits in the AEXC byte which the user need poll only once (at the end of a series of floating-point operations, for example).

The setting or clearing of bits in the AEXC byte has no effect on whether or not the MC68881 will take an exception. The relationship between the bits in the EXC byte and the bits in the AEXC byte is given below. At the end of each operation that can affect the AEXC byte, the following equations are used to generate the new AEXC bits.

AEXC(IOP) = AEXC(IOP) v EXC(BSUN v SNAN v OPERR)
AEXC(OVFL) = AEXC(OVFL) v EXC(OVFL)
AEXC(UNFL) = AEXC(UNFL) v EXC(UNFL ^ INEX2)
AEXC(DZ) = AEXC(DZ) v EXC(DZ)
AEXC(INEX) = AEXC(INEX) v EXC(INEX1 v INEX2 v OVFL)

where:
"v" = Logical OR
"^" = Logical AND

2.1.4 Floating-Point Instruction Address Register

A majority of the MC68881 instructions operate concurrently with the MC68020, such that the MC68020 can be executing instructions while the MC68881 is executing a floating-point instruction. As a result of this non-sequential instruction execution, the program counter value stacked by the MC68020 in response to an enabled floating-point exception trap may not point to the offending instruction.

For the subset of the MC68881 instructions which can generate floating-point exception traps, the 32-bit floating-point instruction address (FPIAR) register is loaded with the logical address of an instruction before the instruction is executed (unless all arithmetic exceptions are disabled). This address can then be used by a floating-point exception handler to locate a floating-point instruction that causes an exception. Since the MC68881 FMOVE to/from the

FPCR, FPSR, or FPIAR and FMOVEM instructions cannot generate floating-point exceptions, they do not modify the FPIAR; thus, these instructions can be used to read the FPIAR in the trap handler without changing the previous value.

This register is cleared by the reset function or a null state size restore operation..

2.2 OPERAND DATA FORMATS AND TYPES

The following paragraphs describe the MC68881 operand data formats. Seven data formats are supported: three signed binary integer formats, three binary floating-point formats, and one packed binary coded decimal (BCD) floating-point format. All data formats are supported uniformly by all arithmetic and transcendental instructions. These formats are as follows:

Byte Integer (B)
Word Integer (W)
Long Word Integer (L)
Single Precision Real (S)
Double Precision Real (D)
Extended Precision Real (X)
Packed Decimal Real (P)

The capital letters contained in parentheses denote the suffixes added to an instruction in the assembly language syntax to specify the data format of operands external to the MC68881. All data formats are organized in memory consistent with the M68000 Family data organization, i.e., the most significant byte is located at the lowest address (nearest $00000000), with each successively less significant byte located at the next address (N + 1, N + 2, etc.). The least significant byte is located at the highest address (nearest $FFFFFFFF).

Within the floating-point data formats, there are five types of numbers that can be represented: normalized numbers, denormalized numbers, zeros, infinities, and not-a-numbers (NANs). These data types are represented through special encodings of each data format.

2.3 INTEGER DATA FORMATS

The three signed (twos complement) integer data formats supported by the MC68881 are identical to those supported by the M68000 Family architecture (see Figure 2-8).

Since all MC68881 internal operations are performed in full extended precision, signed integer operands are converted to extended precision values before the specified operation is performed. Thus, mixed mode arithmetic is implicitly supported.

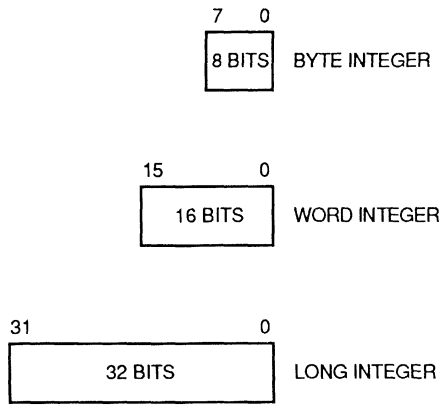


Figure 2-8. Signed Integer Data Formats

2.4 FLOATING-POINT DATA FORMATS

Floating-point numbers may be encoded in three different data formats: single precision (32 bits), double precision (64 bits), and double extended precision (96 bits, 80 of which are used). All three of these formats fully comply with the *IEEE Standard for Binary Floating-Point Arithmetic.*

NOTE

The single extended precision data format defined in the IEEE standard is redundant when the double extended precision format is included; thus, all references in this manual which refer to extended precision imply double extended precision as defined by the IEEE standard.

Since all MC68881 internal operations are performed in extended precision, single and double precision operands are converted to extended precision values before the specified operation is performed. Thus, mixed mode arithmetic is implicitly supported. The memory formats for the real data formats are shown in Figure 2-9.

The exponent in all three binary formats is an unsigned binary integer with an implied bias added to it. The bias values for single, double, and extended precision are 127, 1023, and 16383, respectively. When the bias is subtracted from the value of the exponent, the result represents a signed, twos complement power of two which, when multiplied by the mantissa, yields the magnitude of a normalized floating-point number. Note that the use of biased exponents allows floating-point numbers in memory to be compared using the M68000 Family integer compare instruction (CMP), regardless of the absolute magnitude of the exponents.

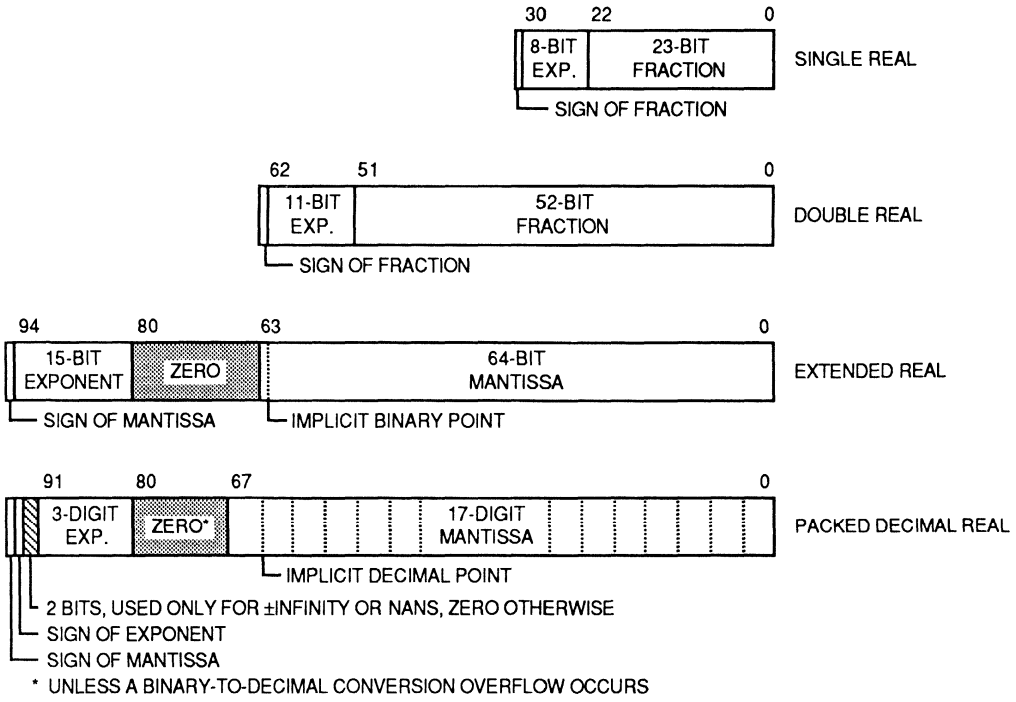


Figure 2-9. Memory Formats for Real Data Types

The exponent in all three binary formats is an unsigned binary integer with an implied bias added to it. The bias values for single, double, and extended precision are 127, 1023, and 16383, respectively. When the bias is subtracted from the value of the exponent, the result represents a signed, twos complement power of two which, when multiplied by the mantissa, yields the magnitude of a normalized floating-point number. Note that the use of biased exponents allows floating-point numbers in memory to be compared using the M68000 Family integer compare instruction (CMP), regardless of the absolute magnitude of the exponents.

Data formats for single and double precision numbers differ slightly from the data format for extended precision numbers in the representation of the mantissa. A normalized mantissa, for all three precisions, is always in the range [1.0...2.0). The extended precision data format explicitly represents the entire mantissa, including the explicit integer part bit. However, for single and double precision data formats, only the fractional portion of the mantissa is explicitly represented and the integer part is always one. Thus, the integer part bit is implicit for single and double precision formats.

The IEEE standard has created the term "significand" to bridge this difference and to avoid the historical implications of the term mantissa. The IEEE standard defines a significand as that component of a binary floating-point number which consists of an explicit or implicit leading bit to the left of the implied binary point and a fraction field to the right of the implied binary point.

This manual uses the terms mantissa and significand interchangeably, given the relationships as shown below.

Single Precision Mantissa	=	Single Precision Significand
	=	1.<23-Bit Fraction Field>
Double Precision Mantissa	=	Double Precision Significand
	=	1.<52-Bit Fraction Field>
Extended Precision Mantissa	=	Extended Precision Significand
	=	1.Fraction
	=	<64-Bit Mantissa Field>

NOTE

Throughout this manual, ranges are specified using traditional set notation, with the format "bound...bound" specifing the boundaries of the range. The type of brackets enclosing the range defines whether the endpoint is inclusive or exclusive. A square bracket indicates inclusive while a parenthesis indicates exclusive. For example, the range specification "[1.0...2.0]" defines the range of numbers greater than or equal to 1.0 and less than or equal to 2.0. The range specification "(0.0...+inf]" defines the range of numbers greater than 0.0 and less than or equal to positive infinity.

Each of the three floating-point data formats can represent five unique floating-point data types:

- Normalized Numbers
- Denormalized Numbers
- Zeros
- Infinities
- Not-A-Numbers (NANs)

The normalized data type never uses the maximum or minimum exponent value for a given format (except for the extended precision format,as noted below). These exponent values in each precision are reserved for representing the special data types: zeros, infinities, denormalized numbers, and NANs. Details of the formats for each type of number for each precision is given in **2.8 DATA FORMAT DETAILS**. The following paragraphs provide a summary of each type.

NOTE

There is a subtle difference between the definition of an extended precision number with an exponent equal to zero and a single or double precision number with an exponent equal to zero. If the exponent of a single or double precision number is zero, then the number is defined to be denormalized, and the implied integer bit is also a zero. However, an extended precision number

with an exponent of zero may have an explicit integer bit equal to one, which results in a normalized number (even though the exponent is equal to the minimum value).

For simplicity, the following discussion treats all three real formats in the same manner, where an exponent value of zero identifies a denormalized number. However, it should be kept in mind that the extended precision format may deviate from this rule.

2.4.1 Normalized Numbers

Normalized numbers encompass all representable real values between the overflow and underflow thresholds, i.e., those numbers whose exponents lie between the maximum and minimum values. Normalized numbers may be positive or negative. For normalized numbers, the implied integer part bit in single and double precision is a one (1). In extended precision, the integer bit is explicitly a one (1). See Figure 2-10.

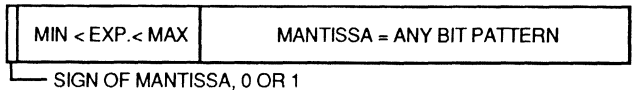


Figure 2-10. Format of Normalized Numbers

2.4.2 Denormalized Numbers

Denormalized numbers represent real values near the underflow threshold (underflow is detected for a given data format and operation when the result exponent is less than or equal to the minimum exponent value). Denormalized numbers may be positive or negative. For denormalized numbers, the implied integer part bit in single and double precision is a zero (0). In extended precision, the integer bit is explicity a zero (0). See Figure 2-11.

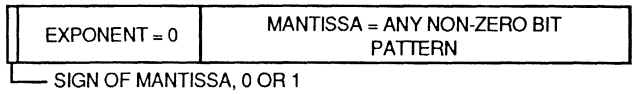


Figure 2-11. Format of Denormalized Numbers

Traditionally, floating-point number systems perform a "flush-to-zero" when underflow is detected. This leaves a large gap in the number line between the smallest magnitude normalized number and zero. The IEEE standard implements gradual underflow, where the

result mantissa is shifted right (denormalized) while incrementing the result exponent until the result exponent reaches the minimum value. If all mantissa bits of the result are shifted off to the right during this denormalization, then the result becomes zero. In many instances, gradual underflow reduces the potential underflow damage to no more than a round-off error (this underflow and denormalization description ignores the effects of rounding and the user selectable rounding modes). Thus, the large gap in the number line created by "flush-to-zero" floating-point number systems is filled with representable (denormalized) numbers in the IEEE "gradual underflow" floating-point number system.

NOTE

Since the extended precision data format has an explicit integer part bit, a number can be formatted with a non-zero exponent (that is not equal to the maximum value), and a zero integer bit, which is not defined by the IEEE standard. Such a number is called an unnormalized number. The MC68881 never generates an unnormalized number as the result of any operation, and unnormalized inputs are always converted to normalized or denormalized numbers or zero before being used. Thus, as required by the IEEE standard, the MC68881 does not distinguish between redundant encodings of extended precision values.

2.4.3 Zeros

Zeros are signed (positive or negative) and represent the real values +0.0 and –0.0. See Figure 2-12.

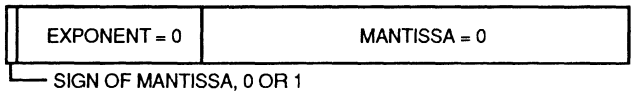


Figure 2-12. Format of Zero

2.4.4 Infinities

Infinities are signed (positive or negative) and represent real values which exceed the overflow threshold. Overflow is detected for a given data format and operation when the result exponent is greater than or equal to the maximum exponent value (this overflow description ignores the effects of rounding and the user selectable rounding modes). See Figure 2-13. For extended precision infinites, the most significant bit of the mantissa (the integer bit) is a don't care.

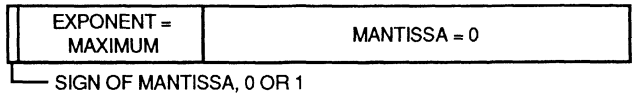


Figure 2-13. Format of Infinity

2.4.5 Not-A-Number

When created by the MC68881, not-a-numbers (NANs), represent the results of operations which have no mathematical interpretation, such as infinity divided by infinity. All operations involving a NAN operand as an input will return a NAN result. When created by the user, NANs can protect against un-initialized variables and arrays, or represent user-defined special number types. See Figure 2-14. For extended precision NANs, the most significant bit of the mantissa (the integer bit) is a don't care.

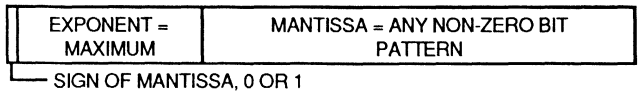


Figure 2-14. Format of Not-A-Numbers

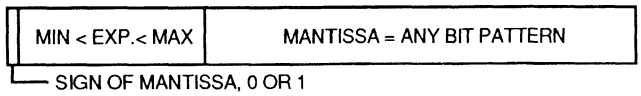
Two different types of NANs, differentiated by the most significant bit (MSB) of the fraction (the MSB of the mantissa for single and double precision, the MSB minus one of the mantissa for extended precision) are implemented. NANs with a leading fraction bit equal to one are non-signaling NANs; NANs with a leading fraction bit equal to zero are signaling NANs (SNANs). SNANs can be used as escape mechanisms for user defined non-IEEE data types. The MC68881 never creates a SNAN as a result of an operation.

The IEEE specification defines the manner in which NANs are handled when used as inputs to an operation. Particularly, if a SNAN is used as an input and the SNAN trap is not enabled, it is required that a non-signaling NAN be returned as the result. The MC68881 does this by using the source SNAN, setting the most significant bit of the fraction, and storing the resultant non-signaling NAN in the destination. Due to the IEEE formats for NANs, the result of setting the most significant fraction bit of a SNAN will always produce a non-signaling NAN.

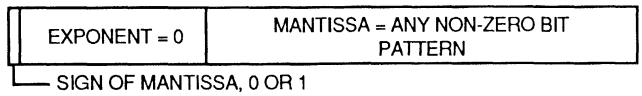
When NANs are created by the MC68881, the NANs always contain the same bit pattern in the mantissa; for any precision, all bits of the mantissa are ones. When created by the user, any non-zero bit pattern can be stored in the mantissa.

2.4.6 Data Type Summary

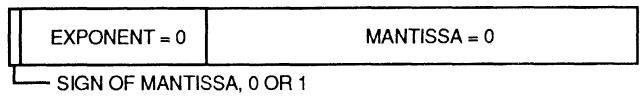
Figure 2-15 presents a summary, for quick reference, of the five floating-point data types for the single, double and extended precision formats.



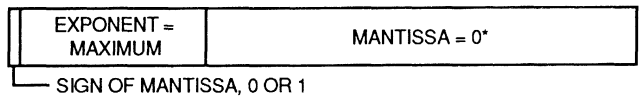
FORMAT OF NORMALIZED NUMBERS



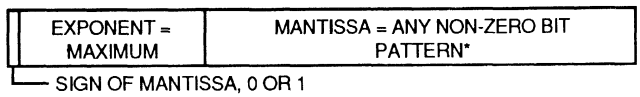
FORMAT OF DENORMALIZED NUMBERS



FORMAT OF ZERO



FORMAT OF INFINITY



FORMAT OF NOT-A-NUMBERS

*For the extended precision format, the most significant bit of the mantissa (the integer bit) is a don't care.

Figure 2-15. Floating-Point Data Type Summary

2.5 PACKED DECIMAL DATA FORMAT

The packed decimal floating-point data format consists of a twenty-four digit packed decimal string as shown in Figure 2-17. Decimal floating-point source operands are converted to extended precision values before the specified operation is performed. Thus, mixed mode arithmetic is implicitly supported.

Paragraph **2.8 DATA FORMAT DETAILS** shows the packed decimal representation for the special data types of zero, infinity, and NAN and also defines all possible data patterns in the packed decimal data format.

2.6 INTERNAL DATA FORMATS

All MC68881 internal operations are performed in extended precision. All external operands, regardless of the data format, are converted to extended precision values before the specified operation is performed.

The format used in the eight floating-point data registers is identical to the extended precision data format described previously and in **2.8 DATA FORMAT DETAILS** (with the deletion of the 16 unused bits). The extended precision data format has a 15-bit biased integer exponent and a 64-bit mantissa.

The format of an intermediate result is shown in Figure 2-16. The intermediate result exponent for some dyadic operations (multiply and divide) can easily overflow or underflow the 15-bit exponent. In order to simplify overflow and underflow detection, intermediate results in the MC68881 maintain a 17-bit twos complement integer exponent. Subsequent detection of an overflow or underflow intermediate result always converts the intermediate 17-bit exponent back into a 15-bit biased exponent before storing into a floating-point data register. Additionally, mantissas are maintained internally as 67 bits for rounding purposes, but are always rounded to 64 bits (or less, depending on the selected rounding precision) before storing into a floating-point data register.

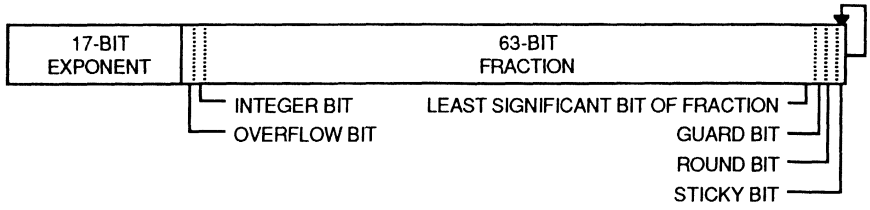


Figure 2-16. Intermediate Result Format

2.7 FORMAT CONVERSIONS

Conversions between two data formats are accomplished in two steps:

1. Convert an operand in any memory data format to the extended precision data format and store it into a floating-point data register, or use it as the source operand for an arithmetic operation.
2. Convert the extended precision value in a floating-point data register to any data format and store it in a memory destination.

2.7.1 Conversion to Extended Precision Data Format

Since the internal data format used by the MC68881 is always extended precision, all external operands, regardless of the data format, are converted to extended precision values before the specified operation is performed. If the external operand, regardless of the data format, is a denormalized number, the number will be normalized before the specified operation is performed. Conversion and normalization applies not only to loading a floating-point data register, but also to external operands involved in arithmetic operations.

Since floating-point data registers always contain extended precision data format values, an external extended precision denormalized number moved into a floating-point data register is stored as an extended precision denormalized number. In this case, the number will first be normalized, and then denormalized before being stored into the designated floating-point data register. This method simplifies the handling of all other data formats and types.

If an external operand is an extended precision unnormalized number, the number will be normalized before it is used in an arithmetic operation. If the external operand is an extended precision unnormalized zero (i.e., with a mantissa of all zeros), the number will be converted to an extended precision true zero before the specified operation is performed. This normalization and conversion applies not only to external unnormalized operands involved in arithmetic operations, but also to loading a floating-point data register. Note that the regular use of unnormalized inputs defeats the purpose of the IEEE standard, and may produce gross inaccuracy in the results.

2.7.2 Conversions to Other Data Formats

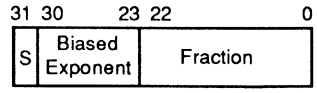
Conversion from the extended precision data format to any of the other six data formats occurs when the contents of an MC68881 floating-point data register are stored to memory or an MC68020 data register. Since no operation performed by the MC68881 can create a unnormalized result, the result of moving a floating-point data register to an extended precision external destination can never be an unnormalized number.

2.8 DATA FORMAT DETAILS

This section provides the format specification details for the single (S), double (D), and extended (X) precision binary real, and packed decimal (P) real string data formats. Refer to Tables 2-2 through 2-5 and Figure 2-17.

Table 2-2. Single Precision Binary Real Format

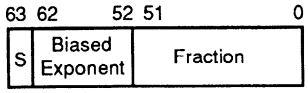
Memory Format:



Field Size (in Bits):	
s = Sign	1
e = Biased	8
f = Fraction	23
Total	32
Interpretation of Sign:	
Positive Mantissa, s =	0
Negative Mantissa, s =	1
Normalized Numbers:	
Bias of e	+127 ($7F)
Range of e	0 < e < 255 ($FF)
Range of f	Zero or Non-Zero
Mantissa = Significand =	1.f
Relation to Representation of Real Numbers	$(-1)^s \times 2^{(e-127)} \times 1.f$
Denormalized Numbers:	
e = Format Minimum =	0 ($00)
Bias of e	+126 ($7E)
Range of f	Non-Zero
Mantissa = Significand =	0.f
Relation to Representation of Real Numbers	$(-1)^s \times 2^{(-126)} \times 0.f$
Signed Zeros:	
e = Format Minimum =	0 ($00)
f = Mantissa = Significand =	0.f = 0.0
Signed Infinities:	
e = Format Maximum =	255 ($FF)
f = Mantissa = Significand =	0.f = 0.0
NANs (Not-A-Number):	
s =	Don't Care
e = Format Maximum =	255 ($FF)
f =	Non-Zero
Representation of f	.1xxxx...xxxx, Non-Signaling
	.0xxxx...xxxx, Signaling
xxxx...xxxx	Non-Zero Bit Pattern
f When Created by the MC68881	.11111...1111
Ranges (Approximate):	
Maximum Positive Normalized	3.4×10^{38}
Minimum Positive Normalized	1.2×10^{-38}
Minimum Positive Denormalized	1.4×10^{-45}

Table 2-3. Double Precision Binary Real Format

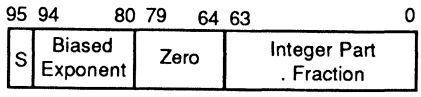
Storage Format:



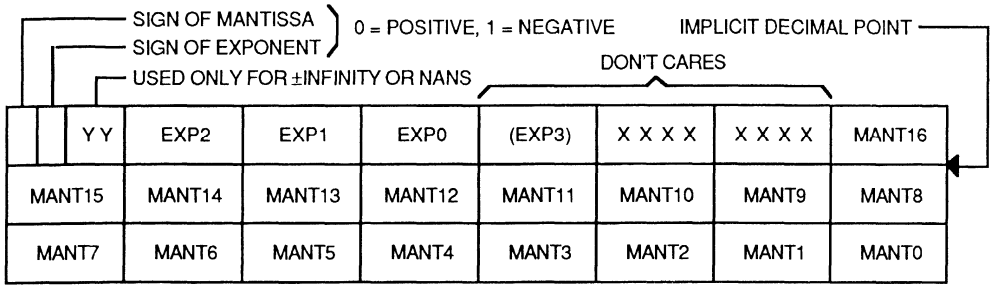
Field Size (in Bits):	
s = Sign	1
e = Biased	11
f = Fraction	52
Total	64
Interpretation of Sign:	
Positive Mantissa, s =	0
Negative Mantissa, s =	1
Normalized Numbers:	
Bias of e	+1023 ($3FF)
Range of e	0 < e < 2047 ($7FF)
Range of f	Zero or Non-Zero
Mantissa = Significand =	1.f
Relation to Representation of Real Numbers	$(-1)^s \times 2^{(e-1023)} \times 1.f$
Denormalized Numbers:	
e = Format Minimum =	0 ($000)
Bias of e	+1022 ($3FE)
Range of f	Non-Zero
Mantissa = Significand =	0.f
Relation to Representation of Real Numbers	$(-1)^s \times 2^{(-1022)} \times 0.f$
Signed Zeros:	
e = Format Minimum =	0 ($00)
f = Mantissa = Significand =	0.f = 0.0
Signed Infinities:	
e = Format Maximum =	2047 ($7FF)
f = Mantissa = Significand =	0.f = 0.0
NANs (Not-A-Number):	
s =	Don't Care
e = Format Maximum =	2047 ($7FF)
f =	Non-Zero
Representation of f	.1xxxx...xxxx, Non-Signaling
	.0xxxx...xxxx, Signaling
xxxx...xxxx	Non-Zero Bit Pattern
f When Created by the MC68881	.11111...1111
Ranges (Approximate):	
Maximum Positive Normalized	18×10^{307}
Minimum Positive Normalized	2.2×10^{-308}
Minimum Positive Denormalized	4.9×10^{-324}

Table 2-4. Extended Precision Binary Real Format

Memory Format:



Field Size (in Bits):	
s = Sign	1
e = Biased Exponent	15
u = Zero, Reserved	16
j = Integer Part	1
f = Fraction	63
Total	96
Interpretation of Unused Bits:	
Input	Don't Care
Output	All Zeros
Interpretation of Sign:	
Positive Mantissa, s =	0
Negative Mantissa, x =	1
Normalized Numbers:	
Bias of e	+16383 ($3FFF)
Range of e	$0 \le e < 32767$ ($7FFF)
j =	1
Range of f	Zero or Non-Zero
j.f = Mantissa = Significand =	1.f
Relation to Representation of Real Numbers	$(-1)^s \times 2^{(e-16383)} \times j.f$
Denormalized Numbers:	
e = Format Minimum	0 ($0000)
Bias of e	+16383 ($3FFF)
j =	0
Range of f	Non-Zero
j.f = Maintissa = Significand =	0.f
Relation to Representation of Real Numbers	$(-1)^s \times 2^{(-16383)} \times 0.f$
Signed Zeros:	
e = Format Minimum =	0 ($0000)
j.f = Mantissa = Significand =	0.0
Signed Infinities:	
e = Format Maximum =	32767 ($7FFF)
j =	Don't Care
j.f = Maintissa = Significand	x.000...0000
NANs (Not-A-Number):	
s =	Don't Care
j =	Don't Care
e = Format Maximum =	32767 ($7FFF)
f =	Non-Zero
Representation of f	x.1xxx...xxxx, Non-Signaling
	x.0xxx...xxxx, Signaling
xxx...xxxx	Non-Zero Bit Pattern
f When Created by the MC68881	1.11111...1111
Ranges (Approximate):	
Maximum Positive Normalized	6×10^{4931}
Minimum Positive Normalized	8×10^{-4933}
Minimum Positive Denormalized	9×10^{-4952}



MANTn Is the nth digit of the mantissa.
EXPn Is the nth digit of the exponent. EXP3 is generated only during a move out operation if the source operand exponent exceeds the magnitude of a three digit exponent; otherwise, it is a don't care. Only EXP0-EXP2 are used for input.
XXXX Are don't care bits, which are zero on output and ignored on input.

Figure 2-17. Packed Decimal Floating-Point Data Format

Table 2-5. Decimal String Type Definitions

Operand Type	Word 5					Word 4	Words 3-0
	15	14	13	12	11...0	15...0	
	SM	SE	y	y	3-Digit Exponent	1-Digit Integer	16-Digit Fraction
±INFINITY	0/1	1	1	1	$FFF	$xxxx	$00...00
±NAN	0/1	1	1	1	$FFF	$xxxx	Non-Zero, see Note 1
±SNAN	0/1	1	1	1	$FFF	$xxxx	Non-Zero, see Note 1
+ZERO	0	0/1	x	x	$000-$999	$xxx0	$00...00
-ZERO	1	0/1	x	x	$000-$999	$xxx0	$00...00
+In-Range	0	0/1	x	x	$000-$999	$xxx0-$xxx9	$00...01-$99...$99
-In-Range	1	0/1	x	x	$000-$999	$xxx0-$xxx9	$00...01-$99...$99

NOTES:
1. A decimal string with the SE and y bits set, an exponent of $FFF, and a non-zero 16-digit decimal fraction is a NAN. On input, the fraction part of the NAN is moved bit-for-bit into the extended precision mantissa of a floating-point register. The exponent of the register is set to signify a NAN, but no decimal-to-binary conversion nor any other conversion is performed. Therefore, the most significant bit of the most significant digit in the decimal fraction (most significant bit of MANT15) is a don't care (as in extended NANs) and the most significant bit minus one of MANT15 is the signaling NAN (SNAN) bit. If it is a zero, then the NAN is a SNAN.
2. If a non-decimal digit [$A...$F] appears in the exponent of a zero, the number will be converted to a true zero. Hardware will not detect if non-decimal digits [$A...$F] appear in the exponent, integer, or fraction digits of an in-range decimal string. These non-decimal digits are converted to binary in the same manner as decimal digits. This will produce a result that will probably be useless, although it is repeatable.
3. Since in-range numbers can not overflow or underflow on conversion to extended precision, normalized extended precision numbers will always be produced by conversion from decimal.

SECTION 3
INSTRUCTION SET

This section details the MC68881 instruction set using the Motorola assembly language syntax and notation. First, a summary of the instruction set is given as an introduction (and can be used for quick reference), followed by a detailed description of each instruction. Also included at the end of this section is a listing of the binary patterns of all of the instructions, and an opcode map summary for use by assembler and compiler writers

3.1 INSTRUCTION SET SUMMARY

The following paragraphs give a brief description of each instruction group, along with tables showing the Motorola syntax for each instructions. The MC68881 instructions can be separated into the following groups:

Data Movement
Dyadic Operations
Monadic Operations
Program Control
System Control

The instruction set is discussed in the following summary and detail sections using this functional grouping and the following notation:

B, W, L	The same size as all M68000 Family processors; specifies a signed integer data type (twos complement) of byte (8 bits), word (16 bits), or long word (32 bits).
S	Single precision floating-point data format (32 bits)
D	Double precision floating-point data format (64 bits)
X	Extended precision floating-point data format (96 bits, 16 bits unused)
P	Packed BCD floating-point data format (96 bits, 12 bytes)
FPm, FPn	One of the eight floating-point data registers
FPcr	One of the three floating-point system control registers (FPCR, FPSR, or FPIAR)
<ea>	Any valid MC68020 addressing mode
k	An twos complement signed integer (-64 to +17) that specifies the format of a number to be stored in the packed decimal format
ccc	An index into the MC68881 constant ROM
<list>	A list of floating-point data registers or control registers
<label>	A relative label used by an assembler to calculate a displacement

3.1.1 Data Movement Operations

This group of instructions provides the means to load or store the user visible configuration of the MC68881 and to move operands into, between, or out of the floating-point data registers. Data format conversion functions are also implicitly supported, since all external data formats are converted to extended precision for internal storage, and vice versa. Operations to move the system control registers into and out of the MC68881 are also provided. The move constant ROM (FMOVECR) instruction allows floating-point data registers to be loaded quickly with commonly used constants such as π, e, 0.0, 1.0, etc. Table 3-1 gives a summary of the data movement instructions that are available and the operand data formats supported.

Table 3-1. Data Movement Operations

Instruction	Operand Syntax	Operand Format	Operation
FMOVE	FPm,FPn	X	source → destination
	<ea>,FPn	B,W,L,S,D,X,P	
	FPm,<ea>	B,W,L,S,D,X	
	FPm,<ea>{#k}	P	
	FPm,<ea>{Dn}	P	
	<ea>,FPcr	L	
	FPcr,<ea>	L	
FMOVECR	#ccc,FPn	X	ROM constant → FPn
FMOVEM	<ea>,<list>[1]	L,X	listed registers → destination
	<ea>,Dn	X	
	<list>[1],<ea>	L,X	source → listed registers
	Dn,<ea>	X	

NOTE:
The register list may include any combination of the eight floating-point registers, or it may contain any combination of the three control registers FPCR, FPSR, and FPIAR. If the register list mask resides in a main processor data register, only floating-point data registers may be specified.

3.1.2 Dyadic Operations

The dyadic floating-point instructions provide several arithmetic functions that require two input operands such as add, subtract, multiply, and divide. For these operations, the first operand may be located in memory, an integer data register, or a floating-point data register, and the second operand is always contained in a floating-point data register. The results of the operation are stored in this register. All operations support any data format and are performed to extended precision, with the exception of the single precision multiply and divide (FSGLMUL and FSGLDIV), which support any precision inputs, but return results accurate only to single precision. These two instructions provide very high speed operations

by sacrificing accuracy. The general format of the dyadic instructions is given in Table 3-2, with a list of the available operations in Table 3-3.

Table 3-2. Dyadic Operation Format

Instruction	Operand Syntax	Operand Format	Operation
F<dop>	<ea>,FPn FPm,FPn	B,W,L,S,D,X,P X	FPn <function> source → FPn

NOTE: <dop> is any one of the dyadic operation specifiers.

Table 3-3. Dyadic Operations

Instruction	Function
FADD	add
FCMP	compare
FDIV	divide
FMOD	modulo remainder
FMUL	multiply
FREM	IEEE remainder
FSCALE	scale exponent
FSGLDIV	single precision divide
FSGLMUL	single precision multiply
FSUB	subtract

3.1.3 Monadic Operations

The monadic floating-point instructions provide several arithmetic functions that require only one input operand. Unlike the integer counterparts to these functions (e.g., NEG <ea>), a source *and* a destination may be specified. The operation is performed on the source operand, and the result is stored in the destination, which is always a floating-point data register. When the source is not a floating-point data register, all data formats are supported; the data format is always extended precision for register-to-register operations. The general format of these instructions is shown in Table 3-4, with a list of the available operations shown in Table 3-5. The form of the simultaneous sine and cosine instruction is given in Table 3-6.

Table 3-4. Monadic Operation Format

Instruction	Operand Syntax	Operand Format	Operation
F<mop>	<ea>,FPn FPm,FPn FPn	B,W,L,S,D,X,P X X	source → function → FPn FPn → function → FPn

NOTE: <mop> is any one of the monadic operations specifiers.

Table 3-5. Monadic Operations

Instruction	Function
FABS	absolute value
FACOS	arc cosine
FASIN	arc sine
FATAN	arc tangent
FATANH	hyperbolic arc tangent
FCOS	cosine
FCOSH	hyperbolic cosine
FETOX	e^x
FETOXM1	e^x-1
FGETEXP	extract exponent
FGETMAN	extract mantissa
FINT	extract integer part
FINTRZ	extract integer part, rounded-to-zero
FLOGN	ln(x)
FLOGNP1	ln(x+1)
FLOG10	$\log_{10}(x)$
FLOG2	$\log_2(x)$
FNEG	negate
FSIN	sine
FSINH	hyperbolic sine
FSQRT	square root
FTAN	tangent
FTANH	hyperbolic tangent
FTENTOX	10^x
FTWOTOX	2^x

Table 3-6. Dual Monadic Operation Format

Instruction	Operand Syntax	Operand Format	Operation
FSINCOS	<ea>,FPc:FPs FPm,FPc:FPs	B,W,L,S,D,X,P X	SIN(source) → FPs; COS(source) → FPc

3.1.4 Program Control Operations

The program control instructions provide a means of affecting program flow based on conditions present in the floating-point status register after any operation that sets the condition codes. In addition to allowing direct control of program flow with the branch conditionally (FBcc) and the decrement and branch conditionally (FDBcc) instructions, the set conditionally (FScc) instruction allows the user to set a Boolean variable based on the floating-point condition codes as an intermediate result in the evaluation of a complex Boolean equation. Also included is a test operand instructon (FTST) that sets the floating-point condition codes for use by the other program and system control instructions, and a no operation instruction (FNOP) that may be used to force synchronization of the MC68881 with the main processor. Table 3-7 gives a summary of the program control instructions that are available.

The MC68881 supports 32 conditional tests that are separated into two groups — 16 that will cause an exception if an unordered condition is present when the conditional test is attempted, and 16 that will not cause a exception if an unordered condition is present (an unordered condition occurs when an input to an arithmetic operation is a NAN). Table 3-8 lists the 32 condition code mnemonics along with the conditional test function. Refer to paragraph **3.3 CONDITION CODE COMPUTATION** for a detailed definition of the conditional equation used by each test.

Table 3-7. Program Control Operations

Instruction	Operand Syntax	Operand Size or Format	Operation
FBcc	<label>	W,L	if condition true, then PC + d → PC
FDBcc	Dn,<label>	W	if condition true, then no operation; else Dn − 1 → Dn; if Dn ≠ −1 then PC + d → PC
FNOP	none	none	no operation
FScc	<ea>	B	if condition true, then 1's → destination else 0's → destination
FTST	<ea> FPn	B,W,L,S,D,X,P X	set FPSR condition codes

Table 3-8. Conditional Test Mnemonics

Exception on Unordered		No Exception on Unordered	
GE	greater than or equal	OGE	ordered greater than or equal
GL	greater than or less than	OGL	ordered greater than or less than
GLE	greater than or less than or equal	OR	ordered
GT	greater than	OGT	ordered greater than
LE	less than or equal	OLE	ordered less than or equal
LT	less than	OLT	ordered less than
NGE	not (greater than or equal)	UGE	unordered or greater than or equal
NGL	not (greater than or less than)	UEQ	unordered or equal
NGLE	not (greater than or less than or equal)	UN	unordered
NGT	not greater than	UGT	unordered or greater than
NLE	not (less than or equal)	ULE	unordered or less than or equal
NLT	not less than	ULT	unordered or less than
SEQ	signalling equal	EQ	equal
SNE	signalling not equal	NE	not equal
SF	signalling always false	F	always false
ST	signalling always true	T	always true

3.1.5 System Control Operations

The system control instructions are utilized for communications with the operating system via a conditional trap instruction (FTRAPcc), and for saving or restoring (FSAVE or FRESTORE) the non-user visible portion of the MC68881 during context switches in a virtual memory or other type of multitasking system. The conditional trap instruction uses the same conditional tests as the program control instructions and allows an optional 16- or 32-bit immediate operand to be included as part of the instruction, for passing parameters to the operating system. Table 3-9 gives a summary of the system control instructions that are available.

Table 3-9. System Control Operations

Instruction	Operand Syntax	Operand Size	Operation
FRESTORE	<ea>	none	state frame → internal registers
FSAVE	<ea>	none	internal registers → state frame
FTRAPcc	none #xxx	none W,L	if condition true, then take exception

3.2 COMPUTATIONAL ACCURACY

Whenever an attempt is made to represent a real number in a binary format of finite precision, there is a possibility that the number can not be represented exactly; this is commonly referred to as round-off error. Furthermore, when two inexact numbers are used in

a calculation, the error present in each number is reflected, and possibly aggravated, in the result.

One of the major reasons the *IEEE Standard for Binary Floating-Point Arithmetic* (P754) was developed is to define the error bounds for calculation of binary floating-point values so that all machines that conform to the standard produce the same results for an operation (when using a specific rounding mode and set of input values, and producing a result of a particular precision). This is because the IEEE standard specifies not only the format of data items, but also defines 1) the maximum allowable error that may be introduced during a calculation and 2) the manner in which rounding of the result is performed. However, the IEEE specification specifies only the operation of some of the instructions supported by the MC68881; those not specified by the IEEE standard are detailed in the following paragraphs. The following paragraphs discuss the accuracy of the calculations performed by the MC68881 by separating them into three groups: 1) the IEEE specified operations and non-transcendental functions, 2) the transcendental functions, and 3) the IEEE specified conversions between binary and decimal real formats.

3.2.1 Arithmetic Instructions

The *IEEE Specification for Binary Floating-Point Arithmetic* specifies that the following operations must be supported for each data format: add, subtract, multiply, divide, remainder, square root, integer part, and compare. Conversions between the various data formats are also required. In addition to these arithmetic functions, the MC68881 also supports the non-transcendental operations of: absolute value, get exponent, get mantissa, negate, modulo remainder, scale exponent, and test. Since the IEEE specification defines the error bounds to which all calculations are performed, the result obtained by any conforming machine can be predicted exactly for a partiuclar precision and rounding mode. The error bound that is defined by the IEEE specification is one-half unit in the last place of the destination data format in the round-to-nearest mode, and one unit in the last place in the other rounding modes.

The MC68881 performs all calculations using a 67-bit mantissa for the intermediate results. The three bits beyond the precision of the extended format allow the MC68881 to perform all calculations as if to infinite precision, and then round the result to the desired precision before storing it in the destination. By performing calculations in this manner, the final result is always correct for the specified destination data format before rounding is performed (unless an overflow or underflow error occurs). The specified rounding operation then produces a number that is as close as possible to the infinitely precise intermediate value and still is representable in the selected precision. An example of how the 67-bit mantissa allows the MC68881 to meet the error bound of the IEEE specification is as follows:

	Mantissa	l	g	r	s	
Intermediate Result:	x.x...	...x00	1	0	0	(Tie Case)
Round-to-Nearest Result:	x.x...	...x00				

In this case, the least significant bit (l) of the rounded result is not incremented, even though the guard bit (g) is set in the intermediate result, because of the way that the IEEE standard specifies how tie cases are to be handled. Assuming that the destination data format is

extended, if the difference between the infinitely precise intermediate result and the round-to-nearest result is calculated, the relative difference is 2^{-64} (the value of the guard bit). This error is equal to one-half of the value of the least significant bit, and is the worst-case error that can be introduced when using the round-to-nearest mode; thus the term one-half unit in the last place correctly identifies the error bound for this operation. This error specification is the relative error present in the result; the absolute error bound is equal to $2^{exponent} \times 2^{-64}$. An example of the error bound for the other rounding modes is as follows:

	Mantissa	l	g	r	s
Intermediate Result:	x.x...	...x00	1	1	1
Round-to-Zero Result:	x.x...	...x00			

In this case, the difference between the infinitely precise result and the rounded result is $2^{-64} + 2^{-65} + 2^{-66}$, which is slightly less than 2^{-63} (the value of the least significant bit); thus the error bound for this operation is not more than one unit in the last place. For all of the arithmetic operations, these error bounds are met by the MC68881, thus providing accurate and repeatable results.

3.2.2 Transcendental Instructions

The IEEE specification does not define the error bound to which transcendental functions are to be performed (except square root). In this context, the transcendental functions are all of those operations not mentioned in the previous paragraphs (i.e., the trigonometric, hyperbolic, logarithmic, and exponential instructions). Due to the highly recursive nature of the alogrithms used to calculate these functions, the round-off error in the input operands to a function, combined wth the limited precision of the MC68881 ALU, do not allow the calculation of a result with the same error limit as the standard arithmetic functions. However, these operations are quite accurate, given the constraint of using an ALU with a finite precision of 67 bits. In general, the worst-case accuracy of any transcendental function is one unit in the last place of double precision (which is equal to 4096 units in the last place of extended precision). It is possible that the accuracy of the MC68881 algorithms is actually much better than this, and at the time of this printing, an exhaustive analysis of the error bounds for these functions is being performed. It is believed that the error bound for these instructions is approximately 64 units in the last place of extended precision. The following example illustrates what this error bounds specification means:

	Mantissa	
Correct Result:	x.x...	...x00000000
MC68881 Calculated Result:	x.x...	...x01000000

In this case, the relative difference between the correct result and the result calculated by the MC68881 is 2^{-57} (assuming an extended precision result), which is 2^6 times the value of the least significant bit. This corresponds to an error of 64 units in the last place.

Note that the transcendental functions perform limited checking for special case input values such as boundary conditions. For example, the exponential functions check for a zero input value, but do not check for exact integer values. Thus, raising a number to an exact integer

value may not produce an exact result (e.g., the instruction FTENTOX #1,FP0 will not produce an extended precision value of exactly 10.0), and the INEX2 bit in the FPSR may be set even if an exact result is produced

3.2.3 Decimal Conversions

The IEEE standard does not specify the format of the decimal real representation that is to be used by any conforming machine, but it does define the error bounds for conversions between decimal and the single and double precision binary formats. Thus, the result of such conversions will always produce consistently rounded results, and those results are predictable and repeatable on any conforming system. However, it is not always possible to perform an exact conversion between these data formats, due to the limited precision of the numbers and the different radices of the values. The error bounds for these conversions is 0.97 unit in the last digit of the destination precision for the round-to-nearest mode; and 1.47 units in the last digit of the destination precision for the other rounding modes. When an input conversion cannot produce an exact result, the MC68881 sets the INEX1 bit in the FPSR exception byte; thus allowing for special handling of these conversion errors that is separate from the handling of other types of inaccurate results. When an output conversion cannot produce an exact result, the INEX2 bit is set.

The packed decimal data format supported by the MC68881 allows the representation of double precision binary numbers in a decimal form, in accordance with the IEEE specification. When a packed decimal number is converted to extended precision, the result is always in range, although the conversion may be inexact. This is because the magnitudes of the exponent and mantissa of a packed decimal number are less than the largest values representable in the extended precision format. Refer to **4.1.2.8 INEXACT RESULT ON DECIMAL INPUT** for a description of how inaccurate decimal to binary conversions are handled.

When an extended precision number is converted to packed decimal, there is not only the possibility that the number cannot be represented exactly, but also that it is too large to be represented with a three-digit exponent. When this type of conversion is performed, the k factor that is specified is used to locate the decimal rounding boundary. If the magnitude of the rounded decimal result exponent exceeds 999, the MC68881 will signal an operand error and calculate a fourth exponent digit, which is included in the destination operand (see Figure 2-11 for the position of the fourth digit). Refer to **4.1.2.7 INEXACT RESULT** for a description of how inaccurate binary to decimal conversions are handled.

Note that the error bounds specified by the IEEE standard apply only to conversions of values in the range of the double precision format. The error bounds for conversion of extended precision values which cannot be represented in double precision will be significantly larger. Conversion of such extended precision values to decimal must be performed by a software envelope to generate decimal results with error bounds analogous to those specified in the IEEE standard for double precision values. Such a software envelope must utilize a "super" extended precision to acheive such error bounds.

Note that the binary to/from decimal conversions performed by the MC68881 utilize the on-chip ROM values of powers of 10 for speed and accuracy, thus allowing exact conversions in many cases (particularly for values that are exact powers of ten).

3.3 CONDITIONAL TEST DEFINITIONS

The MC68881 provides a very simple mechanism for performing conditional tests of the result of any arithmetic floating-point operation. First, the condition code bits in the FPSR are set or cleared at the end of any arithmetic operation, or a move to a single floating-point data register. The condition code bits are always set consistently, based on the result of the operation. Secondly, the MC68881 provides 32 conditional tests that are supported in hardware by the M68000 Family coprocessor interface. This allows conditional instructions to test floating-point conditions to be coded in exactly the same way as the integer conditional instructions, and the evaluation of the conditional test by the MC68881 is performed automatically. The combination of the consistent setting of the condition code bits and the simple programming of conditional instructions gives the MC68020/MC68881 combination a very flexible, high performance method of altering program flow based on floating-point results.

One important consideration of which programmers must be cognizant is that the inclusion of the NAN data type in the IEEE floating-point number system means that each conditional test must include the NAN condition code bit in the Boolean equation for that test. Because a comparison of a NAN with anything is unordered (i.e., it is impossible to determine if a NAN is bigger or smaller than an in-range number), the compare instruction sets the NAN condition code bit when an unordered compare is attempted. All arithmetic instructions also set the NAN bit if the result of an operation is a NAN. The conditional instructions interpret the NAN condition code being set as the unordered condition

The inclusion of the unordered condition in floating-point branches destroys the familiar trichotomy relationship (greater than, equal, less than) that exists for integers. For example, the opposite of floating-point branch greater than (FBGT) is not floating-point branch less than or equal (FBLE).

Rather, it is floating-point branch not greater than (FBNGT). If the result of the previous instruction was unordered, FBNGT is true, whereas both FBGT and FBLE would be false, since unordered fails both of these tests (and sets BSUN). Compiler programmers should be particularly careful of the lack of trichotomy in the floating-point branches since it is common for compilers to invert the sense of conditions.

In the following paragraphs, the conditional tests are broken into three main categories: 1) IEEE non-aware tests, 2) IEEE aware tests, and 3) miscellaneous.

The IEEE non-aware test set is best used when porting a program from a system that does not support the IEEE standard to one that does, or when generating high-level language code that does not support IEEE floating-point concepts (i.e., the unordered condition). When using the IEEE non-aware test set, the user receives a BSUN exception whenever a branch is attempted and the NAN condition code bit is set, unless the branch is an FBEQ or

an FBNE. If the BSUN trap is enabled in the FPCR register, the exception causes a trap. Therefore, the IEEE non-aware program is interrupted if something unexpected occurs.

The IEEE aware branch set should be used by compilers and programmers who are knowledgeable about the IEEE standard and wish to deal with ordered and unordered conditions. Since the ordered or unordered attribute is explicitly included in the conditional test, the BSUN bit is not set in the status register EXC byte when the unordered condition occurs.

3.3.1 IEEE Non-Aware Tests

All of the conditional tests below, except EQ and NE, set the BSUN bit in the status register exception byte if the NAN condition code bit is set when a conditional instruction is executed.

Mnemonic	Definition	Equation	Predicate
EQ	Equal	$\overline{Z}$	000001
NE	Not Equal	Z	001110
GT	Greater Than	$\overline{NAN \vee Z \vee N}$	010010
NGT	Not Greater Than	$NAN \vee Z \vee N$	011101
GE	Greater Than or Equal	$Z \vee \overline{(NAN \vee N)}$	010011
NGE	Not (Greater Than or Equal)	$NAN \vee \overline{(N \wedge Z)}$	011100
LT	Less Than	$N \wedge \overline{(NAN \vee Z)}$	010100
NLT	Not Less Than	$\overline{NAN \vee Z \vee N}$	011011
LE	Less Than or Equal	$Z \vee (N \wedge \overline{NAN})$	010101
NLE	Not (Less Than or Equal)	$\overline{NAN \vee (N \vee Z)}$	011010
GL	Greater or Less Than	$\overline{NAN \vee Z}$	010110
NGL	Not (Greater or Less Than)	$\overline{NAN} \vee Z$	011001
GLE	Greater, Less or Equal	$\overline{NAN}$	010111
NGLE	Not (Greater, Less or Equal)	NAN	011000

3.3.2 IEEE Aware Tests

The following conditional tests do not set the BSUN bit in the status register exception byte under any circumstances.

Mnemonic	Definition	Equation	Predicate
EQ	Equal	Z	000001
NE	Not Equal	$\overline{Z}$	001110
OGT	Ordered Greater Than	$\overline{NAN \vee Z \vee N}$	000010
ULE	Unordered or Less or Equal	$NAN \vee Z \vee N$	001101
OGE	Ordered Greater Than or Equal	$Z \vee (\overline{NAN \vee N})$	000011
ULT	Unordered or Less Than	$NAN \vee (N \wedge \overline{Z})$	001100
OLT	Ordered Less Than	$N \wedge (\overline{NAN \vee Z})$	000100
UGE	Unordered or Greater or Equal	$NAN \vee Z \vee \overline{N}$	001011
OLE	Ordered Less Than or Equal	$Z \vee (N \wedge \overline{NAN})$	000101
UGT	Unordered or Greater Than	$NAN \vee (\overline{N \vee Z})$	001010
OGL	Ordered Greater or Less Than	$\overline{NAN \vee Z}$	000110
UEQ	Unordered or Equal	$NAN \vee Z$	001001
OR	Ordered	$\overline{NAN}$	000111
UN	Unordered	NAN	001000

3.3.3 Miscellaneous Tests

The following tests are not generally used, but are implemented for completeness of the set. If the NAN condition code bit is set, T and F do not set the BSUN bit while SF, ST, SEQ, and SNE will set the BSUN bit.

Mnemonic	Definition	Equation	Predicate
F	False	False	000000
T	True	True	001111
SF	Signalling False	False	010000
ST	Signalling True	True	011111
SEQ	Signalling Equal	Z	010001
SNE	Signalling Not Equal	$\overline{Z}$	011110

3.4 DETAILED INSTRUCTION DESCRIPTIONS

The following paragraphs contain detailed information about each instruction in the MC68881 instruction set. Instructions are arranged in alphabetical order by assembler mnemonic. The following paragraphs provide background information that will aid in reading the detailed instruction information presented.

3.4.1 MC68020/MC68881 Addressing Modes

Due to the nature of the MC68020/MC68881 coprocessor interface, the MC68881 supports all MC68020 addressing modes. The MC68020 effective address modes are categorized by the manner in which the modes are used. The following classifications are used in the instruction details.

Data If an effective address is used to refer to data operands, it is considered a data addressing mode.

Memory If an effective address is used to refer to memory operands, it is considered a memory addressing mode.

Alterable If an effective address is used to refer to alterable (writable) operands, it is considered an alterable addressing mode.

Control If an effective address is used to refer to memory operands that do not have an associated size, it is considered a control addressing mode.

Table 3-10 shows the various addressing categories of each effective address mode. These catgories may be combined so that additional, more restrictive, classifications may be defined. For example, the instruction descriptions use such classifications as memory alterable or data alterable. The former refers to those addressing modes which are both memory and alterable addresses (i.e., the intersection of the two sets of modes), and the latter refers to addressing modes which are both data and alterable.

Table 3-10. Effective Addressing Mode Categories

Address Modes	Mode	Register	Data	Memory	Control	Alterable	Assembler Syntax
Data Register Direct	000	reg. no.	X	–	–	X	Dn
Address Register Direct	001	reg. no.	–	–	–	X	An
Address Register Indirect	010	reg. no.	X	X	X	X	(An)
Address Register Indirect with Postincrement	011	reg. no.	X	X	–	X	(An)+
Address Register Indirect with Predecrement	100	reg. no.	X	X	–	X	–(An)
Address Register Indirect with Displacement	101	reg. no.	X	X	X	X	(d16,An)
Address Register Indirect with Index (8-Bit Displacement)	110	reg. no.	X	X	X	X	(d8,An,Xn)
Address Register Indirect with Index (Base Displacement)	110	reg. no.	X	X	X	X	(bd,An,Xn)
Memory Indirect Post-Indexed	110	reg. no.	X	X	X	X	([bd,An],Xn,od)
Memory Indirect Pre-Indexed	110	reg. no.	X	X	X	X	([bd,An,Xn],od)
Absolute Short	111	000	X	X	X	X	(xxx).W
Absolute Long	111	001	X	X	X	X	(xxx).L
Program Counter Indirect with Displacement	111	010	X	X	X	–	(d16,PC)
Program Counter Indirect with Index (8-Bit Displacement)	111	011	X	X	X	–	(d8,PC,Xn)
Program Counter Indirect with Index (Base Displacement)	111	011	X	X	X	–	(bd,PC,Xn)
PC Memory Indirect Post-Indexed	111	011	X	X	X	–	([bd,PC],Xn,od)
PC Memory Indirect Pre-Indexed	111	011	X	X	X	–	([bd,PC,Xn],od)
Immediate	111	100	X	X	–	–	#<data>

3.4.2 Instruction Description Format

The details of each instruction are given in **3.4.3 Individual Instruction Descriptions**. Figure 3-1 illustrates what information is given in these instructions descriptions.

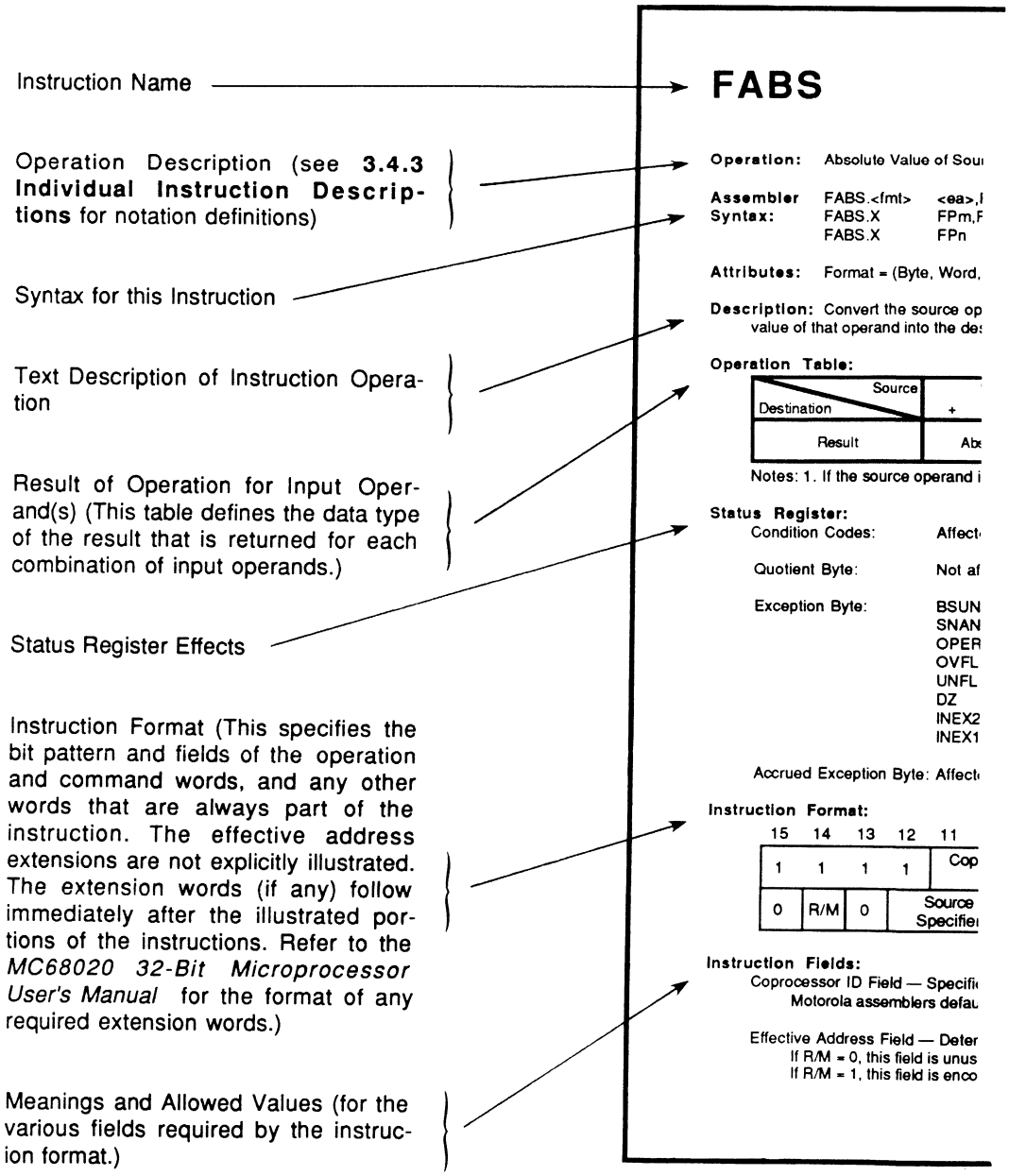


Figure 3-1. Instruction Description Format

3.4.2.1 OPERATION TABLES. An operation table is included for most instructions. This table gives the result data type for the instruction based on types of input operand(s). For example, Figure 3-2 illustrates the table for the FADD instruction.

Destination \ Source	In Range +	In Range -	Zero +	Zero -	Infinity +	Infinity -
In Range +	Add		Add		+inf	- inf
In Range -						
Zero +	Add		+0.0	0.0[1]	+ inf	- inf
Zero -			0.0[1]	-0.0		
Infinity +	+ inf		+ inf		+ inf	NAN[2]
Infinity -	- inf		- inf		NAN[2]	- inf

NOTES:
1. Returns +0.0 in rounding modes RN, RZ and RP; returns -0.0 in RM.
2. Sets the OPERR bit in the FPSR exception status byte.
3. If either operand is a NAN, refer to **3.4.2.2 NANS** for more information.

Figure 3-2. Operation Table Example (FADD Instruction)

In this table, the type of the source operand is shown along the top, and the type of the destination operand is shown along the side. In-range numbers are normalized, denormalized, or un-normalized real numbers, integers, or packed decimal numbers that are converted to normalized or denormalized extended precision numbers upon entering the MC68881.

From this table, it can be seen that if both the source and destination operand are positive zero, the result is also a positive zero. For another example, if the source operand is a positive zero and the destination operand is an in-range number, then the ADD algorithm will be executed to obtain the result. If a label such as ADD appears in the table, it indicates that the MC68881 will perform the indicated operation and return the correct result.

A third example of using the tables is when a source operand is plus infinity and the destination operand is minus infinity. Since the result of such an operation is undefined, a not-a-number (NAN) will be returned as the result and the OPERR bit will be set in the FPSR exception byte.

3.4.2.2 NANS. In addition to the data types covered in the operation tables for each instruction, NANs can also be used as inputs to an arithmetic operation. The operation tables do not contain a row and column for NANs, because NANs are always handled the same way in all operations.

3.4.2.2.1 Non-Signaling NANs. If either operand, but not both operands, to an operation is a NAN, and it is a non-signaling NAN, then that NAN is returned as the result. If both operands are non-signaling NANs, then the destination operand non-signalling NAN is returned as the result.

3.4.2.2.2 Signaling NANs. If either operand to an operation is a signaling NAN (SNAN), then the SNAN bit will be set in FPSR EXC byte. If the SNAN trap enable bit is set in FPCR ENABLE byte, then the trap is taken and the destination is not modified. If the SNAN trap enable bit is not set, then the SNAN is converted to a non-signaling NAN (by setting the SNAN bit in the operand to a one) and the operation continues as described above for non-signaling NANs.

3.4.2.3 OPERATION POST PROCESSING. Most floating-point operations end with an identical post processing step. While reading the summary for each instruction, it should be assumed that an instruction performs post processing unless it is specifically stated that it does not do so. The following paragraphs detail post processing.

3.4.2.3.1 Setting Floating-Point Condition Codes. Unlike the integer arithmetic condition codes found in the MC68020, which are set uniquely for each instruction, the floating-point condition codes are either not changed at all by an instruction, or are always set in the same way by any instruction. Therefore, it is not necessary to include details of condition code settings for each MC68881 instruction in the detailed instruction summary to follow. The following paragraphs explain how condition codes are set for all instructions that modify any condition codes.

Refer to **2.1.3.1 FPSR FLOATING-POINT CONDITON CODE BYTE** for a description of the FPSR condition code byte.The four condition code bits are:

N	Sign of Mantissa
Z	Zero
I	Infinity
NAN	Not-A-Number

These condition code bits differ slightly from integer condition codes in that they are not dependent on the type of operation being performed, but rather, can be set at the end of the operation by examining the result. (The M68000 integer condition codes bits N and Z have this characteristic, but the V and C bits are set differently for different instructions.) At the end of any floating-point operation, the result is inspected and the condition code bits are set or cleared accordingly. For example, if the result of an operation is a positive normalized number, then all of the condition code bits are set to zero. If the result is a minus infinity, then the N and I bits are set and the Z and NAN bits are cleared.

Refer to **2.1.3.1 FPSR FLOATING-POINT CONDITION CODE BYTE** for a description of how these bits are used to generate the four conditions required by the IEEE floating-point standard. Refer to **3.3 CONDITIONAL TEST DEFINITIONS** for a description of how the four condition code bits are used to generate the 32 floating-point conditional tests.

3.4.2.3.2 Underflow, Round, Overflow. During calculation of an arithmetic result, the ALU of the MC68881 has more precision and range than the 80-bit extended precision format. However, the final result of these operations is an extended precision floating-point value. In some cases, an internal result becomes either smaller or larger than can be represented in extended precision. Also the operation may have generated more bits of precision than can be represented in the chosen rounding precision. For these reasons, every arithmetic instruction ends by rounding the result, and checking for overflow and underflow.

At the completion of an arithmetic operation, the internal result is checked to see if it is too small to be represented as a normalized number in the selected precision. If so, the underflow (UNFL) bit is set in the FPSR EXC byte. Unless the number is so grossly underflowed that denormalization will produce a zero, it is also denormalized. Denormalizing a number causes a loss of accuracy, but a zero is not returned unless absolutely necessary. If a number is grossly underflowed, the MC68881 will return a zero. For more details on underflow, refer to **4.1.2.5 UNDERFLOW**.

If no underflow occurs, the internal result is rounded according to the user-selected rounding precision and rounding mode. Refer to Figure 4.2 for a detailed description of how rounding is performed. After rounding, the inexact bit (INEX2) is set appropriately. Lastly, the magnitude of the result is checked to see if it is too large to be represented in the current rounding precision. If so, the overflow (OVFL) bit is set and a correctly signed infinity or correctly signed largest number is returned, depending on the rounding mode in effect. For details on overflow refer to **4.1.2.4 OVERFLOW**.

3.4.3 Individual Instruction Descriptions

The following notation is used in the detailed instruction definitions that follow:

(operand)	Contents of the referenced location or register.
<fmt>	Operand data format: Byte, word, long, single, double, extended, or packed (denoted in the assembler syntax as an extension to the instruction mnemonic of .B, .W, .L, .S, .D, .X or P, respectively).
<ea>	Any valid MC68020 addressing mode
<label>	A relative label used by an assembler to calculate a displacement
<list>	A list of the floating-point data registers or control registers
→	The left operand is moved to the location specified by the right operand.
FPcr	One of the three floating-point system control registers (FPCR, FPSR, or FPIAR)

FPn	One of eight floating-point data registers (always specifies the destination register)
FPm	One of eight floating-point data registers (always specifies the source register)
FPc:FPs	Two of eight floating-point data registers. This notation is used only with the FSINCOS instruction and specifies the register pair where the cosine and sine values are stored.
+inf	Positive infinity
-inf	Negative infinity
NAN	Not-A-Number
d	Displacement
k	An integer (-64 to +17) that specifies the format of a number to be stored in the packed BCD format
ccc	An index into the MC68881 constant ROM

Operation: Absolute Value of Source → FPn

Assembler Syntax:

FABS.<fmt>	<ea>,FPn
FABS.X	FPm,FPn
FABS.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary) and store the absolute value of that operand into the destination floating-point data register.

Operation Table:

Destination \ Source	In Range +	In Range -	Zero +	Zero -	Infinity +	Infinity -
Result	Absolute Value		Absolute Value		Absolute Value	

Notes: 1. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Cleared
OVFL	Cleared
UNFL	If source is an extended precision denormalized number, refer to **4.1.2.5**; cleared otherwise.
DZ	Cleared
INEX2	Cleared
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register			0	0	1	1	0	0	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: Arc Cosine of Source → FPn

Assembler Syntax:
FACOS.<fmt> <ea>,FPn
FACOS.X FPm,FPn
FACOS.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary) and calculate the arc cosine of that value. Return the result to the destination floating-point data register. The function is not defined for source operands outside of the range [-1...+1]; if the source is not in the correct range, a NAN is returned as the result and the OPERR bit is set in the FPSR. If the source is in the correct range, the result will have a value in the range of [0...π].

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Arc Cosine		+π/2		NAN[1]	

Notes: 1. Sets the OPERR bit in the FPSR exception byte.
2. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if the source is infinity, > +1 or < -1; cleared otherwise.
OVFL	Cleared
UNFL	Cleared
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register			0	0	1	1	1	0	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: Source + FPn → FPn

Assembler Syntax: FADD.<fmt> <ea>,FPn
FADD.X FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Convert the source operand to extended precision (if necessary) and add that number to the number contained in the destination floating-point data register. The result is stored in the destination floating-point data register

Operation Table:

Destination \ Source		In Range		Zero		Infinity	
		+	-	+	-	+	-
In Range	+	Add		Add		+inf	- inf
	-						
Zero	+	Add		+0.0	0.0[1]	+ inf	- inf
	-			0.0[1]	-0.0		
Infinity	+	+ inf		+ inf		+ inf	NAN[2]
	-	- inf		- inf		NAN[2]	- inf

Notes: 1. Returns +0.0 in rounding modes RN, RZ and RP; returns -0.0 in RM.
2. Sets the OPERR bit in the FPSR exception byte.
3. If either operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if the source and the destination are opposite-signed infinities, cleared otherwise.
OVFL	Refer to **4.1.2.4**.
UNFL	Refer to **4.1.2.5**.
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register			0	1	0	0	0	1	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn.

Operation: Arc Sine of the Source → FPn

Assembler Syntax:

FASIN.<fmt>	<ea>,FPn
FASIN.X	FPm,FPn
FASIN.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary) and calculate the arc sine of the number. The result is stored in the destination floating-point data register. The function is not defined for source operands outside of the range [-1...+1]; if the source in not in the correct range, a NAN is returned as the result and the OPERR bit is set in the FPSR. If the source is in the correct range, the result will have a value in the range of [-π/2...+π/2].

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Arc Sine		+0.0	-0.0	NAN[1]	

Notes: 1. Sets the OPERR bit in the FPSR exception byte.
2. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if the source is infinity, > +1 or < -1; cleared otherwise.
OVFL	Cleared
UNFL	Cleared
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	0	0	1	1	0	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: Arc Tangent of Source → FPn

Assembler Syntax:
FATAN.<fmt> <ea>,FPn
FATAN.X FPm,FPn
FATAN.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary) and calculate the arc tangent of that value. Return the result to the destination floating-point data register. The result will have a value in the range of [-π/2...+π/2].

Operation Table:

Destination \ Source	In Range +	In Range -	Zero +	Zero -	Infinity +	Infinity -
Result	Arc Tangent		+0.0	-0.0	+π/2	-π/2

Notes: 1. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Cleared
OVFL	Cleared
UNFL	Refer to **4.1.2.5**.
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register			0	0	0	1	0	1	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

FATANH Hyperbolic Arc Tangent FATANH

Operation: Hyperbolic Arc Tangent of Source → FPn

Assembler Syntax:

FATANH.<fmt>	<ea>,FPn
FATANH.X	FPm,FPn
FATANH.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary) and calculate the hyperbolic arc tangent of that value. Return the result to the destination floating-point data register. The function is not defined for source operands outside of the range (-1...+1), with the result equal to -infinity or + infinity if the source is equal to +1 or -1, respectively. If the source is outside of the range [-1...+1], a NAN is returned as the result and the OPERR bit is set in the FPSR.

Operation Table:

Source / Destination	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Hyperbolic Arc Tangent		+0.0	-0.0	NAN[1]	

Notes: 1. Sets the OPERR bit in the FPSR exception byte.
2. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if the source is > +1 or < -1; cleared otherwise.
OVFL	Cleared
UNFL	Refer to **4.1.2.5**.
DZ	Set if the source is equal to +1 or -1; cleared otherwise.
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**; cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register			0	0	0	1	1	0	1

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: If condition true, then PC + d →PC

Assembler Syntax: FBcc.<size> <label>

Attributes: Size = (Word, Long)

Description: If the specified floating-point condition is met, program execution continues at the location (PC) + displacement. The displacement is a twos complement integer which counts the relative distance in bytes. The value of the PC used to calculate the destination address is the address of the branch instruction plus two. If the displacement size is word, then a 16-bit displacement is stored in the word immediately following the instruction operation code. If the displacement size is long word, then a 32-bit displacement is stored in the two words immediately following the instruction operation code.

The conditional specifier "cc" may specify any one of the 32 floating-point conditional tests as described in **3.3 Conditional Test Definitions**.

Status Register:

Condition Codes: Not affected.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Set if the NAN condition code is set and the condition selected is an IEEE non-aware test.
SNAN	Not Affected
OPERR	Not Affected
OVFL	Not Affected
UNFL	Not Affected
DZ	Not Affected
INEX2	Not Affected
INEX1	Not Affected

Accrued Exception Byte: If the BSUN bit is set in the FPSR exception byte, then IOP is set in the accumulated exception byte. All other bits are not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	1	Size	Conditional Predicate					
16-bit Displacement, or Most Significant Word of 32-bit Displacement															
Least Significant Word of 32-bit Displacement (if needed)															

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Size Field — Specifies the size of the signed displacement:
If Format = 0, then the displacement is 16-bits and will be sign extended before use.
If Format = 1, then the displacement is 32-bits.

Conditional Predicate Field — Specifies one of 32 conditional tests as defined in section **3.3**.

Note: When a BSUN exception occurs, it causes a pre-instruction exeption to be taken by the main processor. If the exception handler returns without modifying the image of the PC on the stack frame (to point to the instruction following the FBcc), then it must clear the cause of the exception (by clearing the NAN bit or disabling the BSUN trap) or the exception will occur again immediately upon return to the routine that caused the exception.

Operation: FPn - Source

Assembler Syntax:
FCMP.<fmt> <ea>,FPn
FCMP.X FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Convert the source operand to extended precision (if necessary) and subtract that number from the destination floating-point data register. The result of the subtraction is not retained, but it is used to set the floating-point condition codes as described in section **3.4.2.3.1**.

Operation Table: The entries in this operation table differ from those of the tables describing most of the MC68881 instructions. For each combination of input operand types, the condition code bits that may be set are indicated. If the name of a condition code bit is given and is not enclosed in brackets, then it is always set. If the name of a condition code bit is enclosed in brackets, then that bit is either set or cleared, as appropriate. If the name of a condition code bit is not given, then that bit is always cleared by the operation. The infinity bit is always cleared by the FCMP instruction, since it is not used by any of the conditional predicate equations. Note that the NAN bit is not shown, since NANs are always handled in the same manner (as described in section **3.4.2.2**).

Destination \ Source		In Range		Zero		Infinity	
		+	-	+	-	+	-
In Range	+	{NZ}	none	none	none	N	none
	-	N	{NZ}	N	N	N	none
Zero	+	N	none	Z	Z	N	none
	-	N	none	NZ	NZ	N	none
Infinity	+	none	none	none	none	Z	none
	-	N	N	N	N	N	NZ

Notes: 1. If either operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in the operation table above.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Cleared
OVFL	Cleared
UNFL	Cleared
DZ	Cleared
INEX2	Cleared
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	1	1	1	0	0	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn.

Operation: Cosine of Source → FPn

Assembler Syntax:

FCOS.<fmt>	<ea>,FPn
FCOS.X	FPm,FPn
FCOS.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary) and calculate the cosine of that value. Return the result to the destination floating-point data register. The function is not defined for source operands of ±infinity. If the source operand is not in the range of [-2π...+2π], then the argument will be reduced to within that range before the cosine is calculated. However, large arguments may lose accuracy during reduction, and very large arguments (greater than approximately 10^{20}) will lose all accuracy. The result will be in the range of [-1...+1].

Operation Table:

Source / Destination	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Cosine		+1.0		NAN[1]	

Notes: 1. Sets the OPERR bit in the FPSR exception byte.
2. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if the source operand is ±infinity, cleared otherwise.
OVFL	Cleared
UNFL	Cleared
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	0	1	1	1	0	1

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: Hyperbolic Cosine of Source → FPn

Assembler Syntax:
FCOSH.<fmt> <ea>,FPn
FCOSH.X FPm,FPn
FCOSH.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Convert the source operand to extended precision (if necessary) and calculate the hyperbolic cosine of that value. Return the result to the destination floating-point data register.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Hyperbolic Cosine		+1.0		+inf	

Notes: 1. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Cleared
OVFL	Refer to **4.1.2.4**.
UNFL	Cleared
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register			0	0	1	1	0	0	1

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: If condition true then no operation;
else Dn - 1 → Dn;
if Dn ≠ -1
then PC + d → PC

Assembler Syntax: FDBcc Dn,<label>

Attributes: Unsized

Description: This instruction is a looping primitive of three parameters: a floating-point condition, a counter (an MC68020 data register) and a 16-bit displacement. The instruction first tests the condition to determine if the termination condition for the loop has been met, and if so, the main processor proceeds to execute the next instruction in the instruction stream. If the termination condition is not true, the low order 16-bits of the counter register are decremented by one. If the result is -1, the counter is exhausted and execution continues with the next instruction. If the result is not equal to -1, execution continues at the location specified by the current value of the PC plus the sign-extended 16-bit displacement. The value of the PC used in the branch address calculation is the address of the FDBcc instruction plus two.

The conditional specifier "cc" may specify any one of the 32 floating-point conditional tests as described in **3.3 Conditional Test Definitions**.

Status Register:

Condition Codes: Not affected.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Set if the NAN condition code is set and the condition selected is an IEEE non-aware test.
SNAN	Not Affected
OPERR	Not Affected
OVFL	Not Affected
UNFL	Not Affected
DZ	Not Affected
INEX2	Not Affected
INEX1	Not Affected

Accrued Exception Byte: If the BSUN bit is set in the exception byte, then IOP is set in the accumulated exception byte. All other bits are not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	1	0	0	1	Count Register		
0	0	0	0	0	0	0	0	0	0	Conditional Predicate					
16-bit Displacement															

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Count Register Field — Specifies main processor data register that is used as the counter.

Conditional Predicate Field — Specifies one of the 32 floating-point conditional tests as described in **3.3**.

Displacement Field — Specifies the branch distance (from the address of the instruction plus 2) to the destination in bytes.

Notes: 1. The terminating condition is like that defined by the UNTIL loop constructs of high-level languages. For example: FDBOLT can be stated as "decrement and branch until ordered less than".

2. There are two basic ways of entering a loop: at the beginning, or by branching to the trailing FDBcc instruction. If a loop structure terminated with FDBcc is entered at the beginning, the control counter must be one less than the number of loop executions desired. This count is useful for indexed addressing modes and dynamically specified bit operations. However, when entering a loop by branching directly to the trailing FDBcc instruction, the count should equal the loop execution count. In this case, if the counter is zero when the loop is entered, the FDBcc instruction will not branch, causing a complete bypass of the main loop.

3. When a BSUN exception occurs, it causes a pre-instruction exeption to be taken by the main processor. If the exception handler returns without modifying the image of the PC on the stack frame (to point to the instruction following the FDBcc), then it must clear the cause of the exception (by clearing the NAN bit or disabling the BSUN trap) or the exception will occur again immediately upon return to the routine that caused the exception.

Operation: FPn + Source → FPn

Assembler Syntax:
FDIV.<fmt> <ea>,FPn
FDIV.X FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Convert the source operand to extended precision (if necessary) and divide that number into the number contained in the destination floating-point data register. The result is stored in the destination floating-point data register

Operation Table:

Destination \ Source		In Range +	In Range -	Zero +	Zero -	Infinity +	Infinity -
In Range	+	Divide		+inf[1]	-inf[1]	+0.0	-0.0
	-			-inf[1]	+inf[1]	-0.0	+0.0
Zero	+	+0.0	+0.0	NAN[2]		+0.0	-0.0
	-	-0.0	-0.0			-0.0	+0.0
Infinity	+	+inf	-inf	+inf	-inf	NAN[2]	
	-	-inf	+inf	-inf	+inf		

Notes: 1. Sets the DZ bit in the FPSR exception byte.
2. Sets the OPERR bit in the FPSR exception byte.
3. If either operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set for 0/0 or infinity/infinity, cleared otherwise.
OVFL	Refer to **4.1.2.4**.
UNFL	Refer to **4.1.2.5**.
DZ	Set if the source is zero and the destination is in range, cleared otherwise.
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register			0	1	0	0	0	0	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn.

Operation: $e^{(Source)} \rightarrow FPn$

Assembler Syntax:

	FETOX.<fmt>	<ea>,FPn
	FETOX.X	FPm,FPn
	FETOX.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary) and calculate e to the power of that number. The result is stored in the destination floating-point data register

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	e^X		+1.0		+inf	+0.0

Notes: If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Cleared
OVFL	Refer to **4.1.2.4**.
UNFL	Refer to **4.1.2.5**.
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register			0	0	0	1	0	0	0	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: $e^{(Source)} - 1 \rightarrow FPn$

Assembler Syntax:

FETOXM1.<fmt>	<ea>,FPn
FETOXM1.X	FPm,FPn
FETOXM1.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary) and calculate e to the power of that number, then subtract one from that value. The result is stored in the destination floating-point data register

Operation Table:

Destination \ Source	In Range +	In Range -	Zero +	Zero -	Infinity +	Infinity -
Result	$e^X - 1$		+0.0	-0.0	+inf	-1.0

Notes: 1. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Cleared
OVFL	Refer to **4.1.2.4**.
UNFL	Refer to **4.1.2.5**.
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **3.?**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register			0	0	0	1	0	0	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: Exponent of Source → FPn

Assembler Syntax:

FGETEXP.<fmt>	<ea>,FPn
FGETEXP.X	FPm,FPn
FGETEXP.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary) and extract the binary exponent. Convert the extracted exponent to an extended precision floating-point number, remove the exponent bias and store the result in the destination floating-point data register.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Exponent		+0.0	-0.0	NAN[1]	

Notes: 1. Sets the OPERR bit in the FPSR exception byte.
2. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if the source is ±infinity, cleared otherwise.
OVFL	Cleared
UNFL	Cleared
DZ	Cleared
INEX2	Cleared
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	0	1	1	1	1	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

FGETMAN Get Mantissa FGETMAN

Operation: Mantissa of Source → FPn

Assembler Syntax:

FGETMAN.<fmt>	<ea>,FPn
FGETMAN.X	FPm,FPn
FGETMAN.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary) and extract the mantissa. Convert the mantissa to an extended precision value and store the result in the destination floating-point data register. The result will be in the range [1.0...2.0), with the sign of the source mantissa, zero, or a NAN.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Mantissa		+0.0	-0.0	NAN[1]	

Notes: 1. Sets the OPERR bit in the FPSR exception byte.
2. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if the source is ±infinity, cleared otherwise.
OVFL	Cleared
UNFL	Cleared
DZ	Cleared
INEX2	Cleared
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	0	1	1	1	1	1

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: Integer Part of Source → FPn

Assembler Syntax:

FINT.<fmt>	<ea>,FPn
FINT.X	FPm,FPn
FINT.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary), extract the integer part, and convert it to an extended precision floating-point number. Store the result in the destination floating-point data register. The integer part is extracted by rounding the extended precision number to an integer using the current rounding mode selected in the FPCR Control byte. Thus, the integer part returned is the number that is to the left of the radix point when the exponent is zero, after rounding. For example, the integer part of 137.57 is 137.0 for the round-to-zero and round-to-minus infinity modes, and 138.0 for the round-to-nearest and round-to-plus infinity modes. Note that the result of this operation is a floating-point number.

Operation Table:

Source / Destination	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Integer		+0.0	-0.0	+inf	-inf

Notes: 1. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Cleared
OVFL	Cleared
UNFL	Cleared
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	0	0	0	0	0	1

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: Integer Part of Source → FPn

Assembler Syntax:

FINTRZ.<fmt>	<ea>,FPn
FINTRZ.X	FPm,FPn
FINTRZ.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Convert the source operand to extended precision (if necessary), extract the integer part, and convert it to an extended precision floating-point number. Store the result in the destination floating-point data register. The integer part returned is the number that is to the left of the radix point when the exponent is zero. The integer part is extracted by rounding the extended precision number to an integer using the round-to-zero mode, regardless of the current rounding mode selected in the FPCR Control byte (making it useful for FORTRAN assignments). For example, the integer part of 137.57 is 137.0; the integer part of 0.1245×10^2 is 12.0. Note that the result of this operation is a floating-point number.

Operation Table:

Source / Destination	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Integer, forced Round-to-Zero		+0.0	-0.0	+inf	-inf

Notes: 1. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Cleared
OVFL	Cleared
UNFL	Cleared
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	0	0	0	1	1

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: Log_{10} of Source -> FPn

Assembler Syntax:

	FLOG10.<fmt>	<ea>,FPn
	FLOG10.X	FPm,FPn
	FLOG10.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary) and calculate its logarithm using base 10 arithmetic. Store the result in the destination floating-point data register. This function is not defined for input values less than zero.

Operation Table:

Source / Destination	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Log_{10}	NAN[1]	-inf[2]		+inf	NAN[1]

Notes: 1. Sets the OPERR bit in the FPSR exception byte.
2. Sets the DZ bit in the FPSR exception byte.
3. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if the source operand is < 0, cleared otherwise.
OVFL	Cleared
UNFL	Cleared
DZ	Set if the source is ±0, cleared otherwise.
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	0	1	0	1	0	1

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: Log_2 of Source → FPn

Assembler Syntax:

FLOG2.<fmt>	<ea>,FPn
FLOG2.X	FPm,FPn
FLOG2.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary) and calculate its logarithm using base 2 arithmetic. Store the result in the destination floating-point data register. This function is not defined for input values less than zero.

Operation Table:

Destination \ Source	In Range +	In Range -	Zero +	Zero -	Infinity +	Infinity -
Result	Log_2	NAN[1]	-inf[2]		+inf	NAN[1]

Notes: 1. Sets the OPERR bit in the FPSR exception byte.
2. Sets the DZ bit in the FPSR exception byte.
3. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if the source is < 0, cleared otherwise.
OVFL	Cleared
UNFL	Cleared
DZ	Set if the source is ±0, cleared otherwise.
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	0	1	0	1	1	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: Log_e of Source → FPn

Assembler Syntax:

FLOGN.<fmt>	<ea>,FPn
FLOGN.X	FPm,FPn
FLOGN.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary) and calculate the natural logarithm of that number. Store the result in the destination floating-point data register. This function is not defined for input values less than zero.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	ln(x)	NAN[1]	-inf[2]		+inf	NAN[1]

Notes: 1. Sets the OPERR bit in the FPSR exception byte.
2. Sets the DZ bit in the FPSR exception byte.
3. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if the source operand is < 0, cleared otherwise.
OVFL	Cleared
UNFL	Cleared
DZ	Set if the source is ±0, cleared otherwise.
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	0	1	0	1	0	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

FLOGNP1 $Log_e(x+1)$ FLOGNP1

Operation: Log_e of (Source + 1) → FPn

Assembler Syntax:

FLOGNP1.<fmt>	<ea>,FPn
FLOGNP1.X	FPm,FPn
FLOGNP1.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary), add 1 to that value, and calculate the natural logarithm of the intermediate result. Store the final result in the destination floating-point data register. This function is not defined for input values less than -1.

Operation Table:

Destination \ Source	In Range +	In Range -	Zero +	Zero -	Infinity +	Infinity -
Result	ln(x+1)	ln(x+1)[1]	+0.0	-0.0	+inf	NAN[2]

Notes: 1. If the source is -1, sets the DZ bit in the FPSR exception byte and returns a NAN. If the source is < -1, sets the OPERR bit in the FPSR exception byte and returns a NAN.
2. Sets the OPERR bit in the FPSR exception byte.
3. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if the source operand is < -1, cleared otherwise.
OVFL	Cleared
UNFL	Refer to **4.1.2.5**.
DZ	Set if the source operand is -1, cleared otherwise.
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	0	0	0	1	1	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: Modulo remainder of (FPn ÷ Source) → FPn

Assembler Syntax:
FMOD.<fmt> <ea>,FPn
FMOD.X FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary) and calculate the modulo remainder of the destination floating-point data register, using the source value as the modulus. The result is stored in the destination floating-point data register, and the seven least significant quotient bits and the sign of the quotient are stored in the FPSR quotient byte (where the quotient is the result of FPn ÷ Source). The modulo remainder function is defined as:

$$FPn - (Source \times N)$$

where N = INT(FPn ÷ Source) in the round-to-zero mode

The FMOD function is not defined for a source operand equal to zero or for a destination operand equal to infinity. Note that this function is not the same as the FREM instruction, which uses the round-to-nearest mode and thus returns the remainder that is required by the *IEEE Specification for Binary Floating-Point Arithmetic.*

Operation Table:

Destination \ Source	In Range +	In Range -	Zero +	Zero -	Infinity +	Infinity -
In Range +	Modulo Remainder		NAN [1]		FPn [2]	
In Range -						
Zero +	+0.0		NAN [1]		+0.0	
Zero -	-0.0				-0.0	
Infinity +	NAN [1]		NAN [1]		NAN [1]	
Infinity -						

Notes: 1. Sets the OPERR bit in the FPSR exception byte.
2. Returns the value of FPn before the operation. However, the result will be processed by the normal instruction termination procedure to round it as required. Thus, an underflow and/or inexact result may occur if the rounding precision has been changed to a smaller size since the FPn value was last loaded.
3. If either operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Loaded with the sign and least significant seven bits of the quotient (FPn ÷ Source). The sign of the quotient is the exclusive OR of the sign bits of the source and destination operands.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if the souce is zero, or the destination is infinity; cleared otherwise.
OVFL	Cleared
UNFL	Refer to **4.1.2.5**.
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier				Destination Register			0	1	0	0	0	0	1

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn.

Operation: Source → Destination

Assembler Syntax:

```
FMOVE.<fmt>   <ea>,FPn
FMOVE.<fmt>   FPm,<ea>
FMOVE.P       FPm,<ea>{Dn}
FMOVE.P       FPm,<ea>{#k}
```

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Move the contents of the source operand to the destination operand. Although the primary function of this intruction is data movement, it is also considered an arithmetic insruction, since conversions from the source operand format to the destination operand format are performed implicitly during the move operation. Also, the source operand will be rounded according to the selected rounding precision and mode.

Unlike the M68000 Family integer data movement instruction, the floating-point move instruction does not support a memory-to-memory format (for such transfers, it is much faster to utilize the M68000 Family integer MOVE instruction to transfer the floating-point data than to use the FMOVE instruction). The FMOVE instruction only supports memory-to-register, register-to-register, and register-to-memory operations (in this context, "memory" may refer to an MC68020 data register if the data format is byte, word, long or single). In fact, these two operations use distinctly different command word encodings, and are described separately below.

Memory-to-Register Operation:

The source operand is converted to an extended precision floating-point number (if necessary) and stored in the destination floating-point data register. Depending on the source data format and the rounding precision, some operations may produce an inexact result. In the following table, combi - nations that can produce an inexact result are marked with a dot (•), while all other combinations will produce an exact result.

	Source Format:	B	W	L	S	D	X	P
Rounding	Single			•		•	•	•
Precision:	Double						•	•
	Extended							•

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Cleared
OVFL	Cleared
UNFL	Refer to **4.1.2.5** if the source is an extended precision denormalized number, cleared otherwise.
DZ	Cleared
INEX2	Refer to **4.1.2.7** if <fmt> is L, D or X, cleared otherwise.
INEX1	Refer to **4.1.2.8** if <fmt> is P; cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	0	0	0	0	0	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn.

Register-to-Memory Operation:

The source operand is rounded, if necessary, to the specified size and stored at the destination effective address. If the format of the destination is packed decimal, then a third operand is required to specify the format of the resultant string. This operand, called the k-factor, is a 7-bit signed integer (twos complement) and may be specified as an immediate value or in a main processor data register. If a data register contains the k-factor, only the least significant 7-bits are used, and the rest of the register is ignored.

Status Register:

Condition Codes: Not affected.

Quotient Byte: Not affected.

Exception Byte:		
<fmt> is B, W or L	BSUN	Cleared
	SNAN	Refer to **3.4.2.2**.
	OPERR	Set if the source operand is infinity, or if the destination size is exceeded after conversion and rounding. Cleared otherwise.
	OVFL	Cleared
	UNFL	Cleared
	DZ	Cleared
	INEX2	Refer to **4.1.2.7**.
	INEX1	Cleared
<fmt> is S, D or X	BSUN	Cleared
	SNAN	Refer to **3.4.2.2**.
	OPERR	Cleared
	OVFL	Refer to **4.1.2.4**.
	UNFL	Refer to **4.1.2.5**.
	DZ	Cleared
	INEX2	Refer to **4.1.2.7**.
	INEX1	Cleared
<fmt> is P	BSUN	Cleared
	SNAN	Refer to **3.4.2.2**.
	OPERR	Set if the k-factor > +17, or the magnitude of the decimal exponent exceeds 3 digits. Cleared otherwise.
	OVFL	Cleared
	UNFL	Cleared
	DZ	Cleared
	INEX2	Refer to **4.1.2.7**.
	INEX1	Cleared

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	1	1	Destination Format			Source Register			k-factor (if required)						

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Encoded with the M68000 addressing mode for the destination operand as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

* Only if <fmt> is Byte, Word, Long or Single.

Destination Format Field — Specifies the data format of the destination operand:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P {#k}	Packed Decimal Real with static k-factor
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer
111	P {Dn}	Packed Decimal Real with dynamic k-factor

Source Register Field — Specifies the source floating-point data register, FPm.

k-factor Field — Only used if the destination format is Packed Decimal, to specify the format of the decimal string. For any other destination format, this field should be set to all zeroes. For a static k-factor, this field is encoded with a twos complement integer where the value defines the format as follows:

-64 to 0 — Indicates the number of significant digits to the right of the decimal point (Fortran "F" format).
+1 to +17 — Indicates the number of significant digits in the mantissa (Fortran "E" format).
+18 to +63 — Sets the OPERR bit in the FPSR exception byte, treated as +17.

The format of this field for a dynamic k-factor is:

r r r 0 0 0 0

Where "rrr" is the number of the main processor data register that contains the k-factor value.

The following table gives several examples of how the k-factor value affects the format of the decimal string that is produced by the MC68881. The format of the string that is generated is independent of the source of the k-factor (static or dynamic).

k-factor	Source Operand Value	Destination String
-5	+12345.678765	+1.234567877 E+4
-3	+12345.678765	+1.2345679 E+4
-1	+12345.678765	+1.23457 E+4
0	+12345.678765	+1.2346 E+4
+1	+12345.678765	+1. E+4
+3	+12345.678765	+1.23 E+4
+5	+12345.678765	+1.2346 E+4

Operation: Source → Destination

Assembler Syntax:
FMOVE.L <ea>,FPcr
FMOVE.L FPcr,<ea>

Attributes: Size = (Long)

3

Description: Move the contents of a floating-point system control register into or out of the MC68881 (the control registers are the FPCR, FPSR and FPIAR). The external register image may be located in memory or an MC68020 register. A 32-bit transfer is always performed, even though the system control register may not have 32 implemented bits. Unimplemented bits of a control register are read as zeros and are ignored during writes (but must be zero for compatability with future devices).

This instruction will not cause a pending exception to be reported to the main processor. Furthermore, a write to the FPCR exception enable byte or the FPSR exception status byte will not generate a new exception, regardless of the value written.

Status Register: Will be changed only if the destination is the FPSR; in which case all bits will be modified to reflect the value of the source operand.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
1	0	dr	Register Select				0	0	0	0	0	0	0	0	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for the operation:

Memory-to-Register —

Addr. Mode	Mode	Register
Dn	000	reg. number: Dn
An*	001	reg. number: An
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if the source register is the FPIAR.

Register-to-Memory —

Addr. Mode	Mode	Register
Dn	000	reg. number: Dn
An*	001	reg. number: An
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

* Only if the destination register is the FPIAR.

dr Field — Specifies the direction of the data transfer.
0 — Move an external operand to the specified system control register.
1 — Move the specified system control register to an external location.

Register Select Field — Specifies the system control register to be moved:

100	FPCR	Floating-point Control Register
010	FPSR	Floating-point Status Register
001	FPIAR	Floating-point Instruction Address Register

Operation: ROM Constant → FPn

Assembler Syntax: FMOVECR.X #ccc,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Move an extended precision constant from the MC68881 on-chip ROM, round it to the precision specified by the FPCR, and store it in the destination floating-point data register. A constant value is specified by a predefined offset into the constant ROM. The constants contained in the ROM are shown in the table below.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Cleared
OPERR	Cleared
OVFL	Cleared
UNFL	Cleared
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	Cleared

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	0	0	0	0	0	0
0	1	0	1	1	1	Destination Register			ROM offset						

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Destination Register Field — Specifies the destination floating-point data register, FPn.

ROM offset Field — Specifies the offset into the MC68881 on-chip constant ROM where the desired constant is located. The offsets for the available constants are:

Offset	Constant
$00	π
$0B	$Log_{10}(2)$
$0C	e
$0D	$Log_2(e)$
$0E	$Log_{10}(e)$
$0F	0.0
$30	ln(2)
$31	ln(10)
$32	10^0
$33	10^1
$34	10^2
$35	10^4
$36	10^8
$37	10^{16}
$38	10^{32}
$39	10^{64}
$3A	10^{128}
$3B	10^{256}
$3C	10^{512}
$3D	10^{1024}
$3E	10^{2048}
$3F	10^{4096}

Other constants are contained in the on-chip ROM, but are useful only to the on-chip microcode routines. The values contained at offsets other than those defined above are reserved for the use of Motorola, and may be different on various mask sets of the MC68881.

Operation: Register List → Destination
Source → Register List

Assembler Syntax:

```
FMOVEM.X    <list>,<ea>
FMOVEM.X    Dn,<ea>
FMOVEM.X    <ea>,<list>
FMOVEM.X    <ea>,Dn
```


<list> A list of any combination of the eight floating-point data registers, with individual register names separated by a slash, "/"; and/or contiguous blocks of registers specified by the first and last register names separated by a dash, "-".

Attributes: Format = (Extended)

Description: Move one or more extended precision values to or from a list of floating-point data registers. No conversion or rounding is performed during this operation, and the FPSR is not affected by the instruction. This instruction will not cause a pending exception to be reported to the main processor.

Any combination of the eight floating-point data registers may be transferred, with the selected registers specified by a user-supplied mask. This mask is an 8-bit number, where each bit corresponds to one register; if a bit is set in the mask, that register will be moved. The register select mask may be specified as a static value contained in the instruction, or a dynamic value in the least significant 8-bits of an MC68020 data register (the upper 24-bits of the register are ignored).

FMOVEM allows three types of addressing modes: the control modes, the predecrement mode, or the postincrement mode. If the effective address is one of the control addressing modes, the registers are transferred between the MC68881 and memory starting at the specified address and up through higher addresses. The order of the transfer is from FP0 through FP7.

If the effective address is the predecrement mode, only a register to memory operation is allowed. The registers are stored starting at the address contained in the address register and down through lower addresses. Before each register is stored, the address register is decremented by 12 (the size of an extended precision number in memory) and the floating-point data register is then stored at the resultant address. When the operation is complete, the address register points to the image of the last floating-point data register stored. Each register is stored in the format described in section **2.2 Operand Data Types and Formats**, such that the most significant byte of the register image is stored at the lowest address, and the least significant byte at the highest address. The order of the transfer is from FP7 through FP0.

If the effective address is the postincrement mode, only a memory to register operation is allowed. The registers are loaded starting at the specified address and up through higher addresses. After each register is stored, the address register is incremented by 12 (the size of an extended precision number in memory). When the operation is complete, the address register points to the byte immediately following the image of the last floating-point data register loaded. The order of the transfer is the same as for the control addressing modes, FP0 through FP7.

Status Register:
Not Affected. Note that the FMOVEM instruction provides the only mechanism for moving a floating-pointdata item between the MC68881 and memory without performing any data conversions or affecting the condition code and exception status bits.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
1	1	dr		Mode			0	0	0	Register List					

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for the operation:

Memory-to-Register —

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	—	—
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Register-to-Memory —

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number: An
(An)+	—	—
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

dr Field — Specifies the direction of the transfer.

0 — Move the listed registers from memory to the MC68881.

1 — Move the listed registers from the MC68881 to memory.

Mode Field — Specifies the type of the register list and addressing mode.
- 00 — Static register list, predecrement addressing mode.
- 01 — Dynamic register list, predecrement addressing mode.
- 10 — Static register list, postincrement or control addressing mode.
- 11 — Dynamic register list, postincrement or control addressing mode.

Register List Field:

Static list — contains the register select mask; if a register is to be moved, the corresponding bit in the mask is set, otherwise it is clear.

Dynamic list — contains the main processor data register number, rrr, as shown below.

Register List Format

Static, -(An)	—	FP7	FP6	FP5	FP4	FP3	FP2	FP1	FP0
Static, (An)+ or Control	—	FP0	FP1	FP2	FP3	FP4	FP5	FP6	FP7
Dynamic	—	0	r	r	r	0	0	0	0

The format of the dynamic list mask is the same as for the static list, and is contained in the least significant 8-bits of the specified MC68020 data register.

Programming Note: This instruction provides a very useful feature, dynamic register list specification, that can significantly enhance system performance. If the calling conventions used for procedure calls utilize the dynamic register list feature, the number of floating-point data registers saved and restored can be reduced. Since a save or restore of a floating-point data register requires at least 6 bus cycles (more if the memory address is not long word aligned), then if a register does not need to be saved and restored, a minimum of 36 clock cycles will be eliminated from the procedure call and return overhead for each register not saved unnecessarily.

In order to utilize the dynamic register selection feature of the FMOVEM instruction, both the calling and the called procedures must be written to communicate information about register usage. When one procedure calls another procedure, a register mask should be passed to the called procedure that indicates which registers must not be altered upon return to the calling procedure. The called procedure can then save only those registers that will be modified *and* are already in use. There are several techniques that can be used to utilize this mechanism, and an example is given below.

In this example, a convention is defined where each called procedure is passed a word mask in D7 which identifies all floating-point registers in use by the calling procedure. Bits 15 though 8 identify the registers in the order FP0 through FP7, while bits 7 through 0 identify the registers in the order FP7 through FP0 (the two masks are required due to the different transfer order used by the predecrement and postincrement addressing modes). The code used by the calling procedure consists of simply moving the mask (which is generated at compile time) for the floating-point data registers currently in use into D7:

```
Calling procedure...
    MOVE.W    #ACTIVE_NOW,D7    Load the list of FP registers that are in use
    BSR       PROC_2
```

The entry code for all other procedures computes two masks. The first mask identifies the registers in use by the calling procedure that will be used by the called procedure (and therefore saved and restored by the called procedure). The second mask identifies the registers in use by the calling procedure that will not be used by the called procedure (and therefore not saved on entry). The appropriate registers are then stored along with the two masks:

```
Called procedure...
        MOVE.W    D7,D6                 Copy the list of active registers
        AND.W     #WILL_USE,D7          Generate the list of doubly-used registers
        FMOVEM    D7,-(A7)              Save those registers
        MOVE.W    D7,-(A7)              Save the register list
        EOR.W     D7,D6                 Generate the list of not saved active registers
        MOVE.W    D6,NOT_SAVED(A7)      Save it for later use
```


If the second procedure must call a third procedure, a register mask must be passed to the third procedure that will indicate which registers must not be altered by the third procedure. This mask must identify any registers in the list from the first procedure that were not saved by the second procedure, plus any registers used by the second procedure that must not be altered by the third procedure. An example of the calculation of this mask is:

```
Nested calling sequence...
        MOVE.W    NOT_SAVED(A7),D7      Load the list of active registers not saved at entry
        OR.W      #ACTIVE_NOW,D7        Combine with those active at this time
        BSR       PROC_3
```

Upon return from a procedure, the restoration of the necessary registers follows the same convention, and the register mask generated during the save operation on entry can be used to restore the required floating-point data registers:

```
Return to caller...
        MOVE.B    (A7)+,D7              Get the register list (pop a word, use high byte)
        FMOVEM    (A7)+,D7              Restore the registers
          •
          •
          •
        RTS                             Return to the calling routine
```

Operation: Register List → Destination
Source → Register List

Assembler Syntax:
FMOVEM.L <list>,<ea>
FMOVEM.L <ea>,<list>

<list> A list of any combination of the three floating-point system control registers (FPCR, FPSR and FPIAR), with individual register names separated by a slash, "/".

3

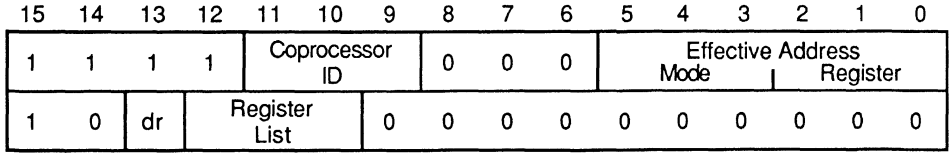
Attributes: Size = (Long)

Description: Move one or more 32-bit values into or out of the specified system control registers. Any combination of the three system control registers may be selected. The registers are always moved in the same order, regardless of the addressing mode used; with the FPCR moved first, followed by the FPSR, and the FPIAR moved last (if a register is not selected for the transfer, the relative order of the transfer of the other registers is the same). The first register is transferred between the MC68881 and the specified address, with successive registers located up through higher addresses.

When more than one register is moved, the memory or memory alterable addressing modes are allowed as shown below. If the addressing mode is predecrement, the address register is first decremented by the total size of the register images to be moved (ie, 4 times the number of registers) and then the registers are transferred starting at the resultant address. For the postincrement addressing mode, the selected registers are transferred to or from the specified address, and then the address register is incremented by the total size of the register images that were transferred. If a single system control register is selected, the data register direct addressing mode may be used; or, if the selected register is the FPIAR, then the address register direct addressing mode may be used. Note that if a single register is selected, the opcode generated is the same as for the FMOVE single system control register instruction.

Status Register: Will be changed only if the destination list includes the FPSR; in which case all bits will be modified to reflect the value of the source register image.

Instruction Format:



Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for the operation:

Memory-to-Register —

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An**	001	reg. number: An
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if a single FPcr is selected.
** Only if the FPIAR is the single register selected.

Register-to-Memory —

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An**	001	reg. number: An
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

* Only if a single FPcr is selected.
** Only if the FPIAR is the single register selected.

dr Field — Specifies the direction of the transfer.
0 — Move the listed registers from memory to the MC68881.
1 — Move the listed registers from the MC68881 to memory.

Register List Field: — Contains the register select mask; if a register is to be moved, the corresponding bit in the list is set, otherwise it is clear.

Bit Number — 12 11 10
Register — FPCR FPSR FPIAR

Operation: Source x FPn → FPn

Assembler Syntax:
FMUL.<fmt> <ea>,FPn
FMUL.X FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Convert the source operand to extended precision (if necessary) and multiply that number by the value in the destination floating-point data register. Store the result in the destination floating-point data register.

Operation Table:

Destination \ Source		In Range +	In Range -	Zero +	Zero -	Infinity +	Infinity -
In Range	+	Multiply		+0.0	-0.0	+inf	-inf
	-			-0.0	+0.0	-inf	+inf
Zero	+	+0.0	-0.0	+0.0	-0.0	NAN[1]	
	-	-0.0	+0.0	-0.0	+0.0		
Infinity	+	+inf	-inf	NAN[1]		+inf	-inf
	-	-inf	+inf			-inf	+inf

Notes: 1. Sets the OPERR bit in the FPSR exception byte.
2. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set for 0 x infinty, cleared otherwise.
OVFL	Refer to **4.1.2.4**.
UNFL	Refer to **4.1.2.5**.
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register			0	1	0	0	0	1	1

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn.

Operation: -(Source) → FPn

Assembler Syntax:

FNEG.<fmt>	<ea>,FPn
FNEG.X	FPm,FPn
FNEG.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary), invert the sign of the mantissa, and store the result in the destination floating-point data register.

Operation Table:

Destination \ Source	In Range +	In Range -	Zero +	Zero -	Infinity +	Infinity -
Result	Negate		-0.0	+0.0	-inf	+inf

Notes: 1. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Cleared
OVFL	Cleared
UNFL	If source is an extended precision denormalized number, refer to **4.1.2.5**; cleared otherwise.
DZ	Cleared
INEX2	Cleared
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	0	1	1	0	1	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: None

Assembler Syntax: FNOP

Attributes: Unsized

3

Description: This instruction does not perform any explicit operation. It is useful, however, to force synchronization of the MC68881 with a main processor, or to force processing of pending exceptions. The syncronization function is inherent in the way that the MC68881 uses the M68000 Family Coprocessor Interface. For most MC68881 instructions, the main processor is allowed to continue with the execution of the next instruction once the MC68881 has any operands needed for an operation; thus supporting concurrent execution of floating-point and integer instructions. However, if the main processor attempts to initiate the execution of a new instruction in the MC68881 before the previous one is completed, then the main processor will be forced to wait until that instruction execution is done before proceeding with the new instruction. FNOP is treated in the same way as other instructions, and thus cannot be executed until the previous floating-point instruction is completed and the main processor is "synchronized" with the MC68881.

The FNOP can also be used to force the processing of pending exceptions from the execution of previous instructions. This is also inherent in the way that the MC68881 utilizes the M68000 Family Coprocessor Interface. Once the MC68881 has received an input operand for an arithmetic instruction, it will **always** release the main procesor to execute the next instruction (regardless of whether or not concurrent execution is prevented for the instruction due to tracing) without reporting the exception during the execution of that instruction. Then, when the main processor attempts to initiate the execution of the next MC68881 instruction, a pre-instruction exception will be reported that starts exception processing for the exception that occurred during the previous instruction. By using the FNOP instruction, the user can force any pending exceptions to be processed without performing any other operations.

Status Register: Not Affected.

Instruction Format:

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

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Note: FNOP uses the same opcode as the "FBcc.W <label>" instruction, with cc = F (non-trapping false) and <label> = *+2 (which results in a displacement of 0).

Operation: IEEE Remainder of (FPn ÷ Source) → FPn

Assembler Syntax:
FREM.<fmt> <ea>,FPn
FREM.X FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Convert the source operand to extended precision (if necessary) and calculate the IEEE remainder of the destination floating-point data register using the source value as the divisor. The result is stored in the destination floating-point data register, and the seven least significant quotient bits and the sign of the quotient are stored in the FPSR quotient byte (where the quotient is the result of FPn ÷ Source). The IEEE remainder function is defined as:

$$FPn - (Source \times N)$$

where N = INT(FPn ÷ Source) in the round-to-nearest mode

The FREM function is not defined for a source operand equal to zero or for a destination operand equal to infinity. Note that this function is not the same as the FMOD instruction, which uses the round-to-zero mode and thus returns a remainder that is different from the remainder required by the *IEEE Specification for Binary Floating-Point Arithmetic.*

Operation Table:

Destination \ Source	In Range +	In Range -	Zero +	Zero -	Infinity +	Infinity -
In Range +	IEEE Remainder		NAN[1]		FPn[2]	
In Range -						
Zero +	+0.0		NAN[1]		+0.0	
Zero -	-0.0				-0.0	
Infinity +	NAN[1]		NAN[1]		NAN[1]	
Infinity -						

Notes: 1. Sets the OPERR bit in the FPSR exception byte.
2. Returns the value of FPn before the operation. However, the result will be processed by the normal instruction termination procedure to round it as required. Thus, an underflow and/or inexact result may occur if the rounding precision has been changed to a smaller size since the FPn value was last loaded.
3. If either operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Loaded with the sign and least significant seven bits of the quotient (FPn ÷ Source). The sign of the quotient is the exclusive OR of the sign bits of the source and destination operands.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if the souce is zero, or the destination is infinity; cleared otherwise.
OVFL	Cleared
UNFL	Refer to **4.1.2.5**.
DZ	Cleared
INEX2	Cleared
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register			0	1	0	0	1	0	1

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn.

FRESTORE Restore Internal State (Privileged Instruction) FRESTORE

Operation: If in supervisor state
then MC68881 State Frame → Internal State
else trap

Assembler Syntax: FRESTORE <ea>

3

Attributes: Unsized, privileged.

Description: The MC68881 aborts any execution of any operation that it was performing, and a new internal state is loaded from the state frame located at the effective address. The first word at the specified address is the format word of the state frame, which specifies the size of the frame and the revision number of the MC68881 that created it. The MC68020 will write the first word to the MC68881 Restore CIR to initiate the restore operation, and then read the Response CIR to verify that the MC68881 recognizes the format word as valid. If the format word is invalid for this MC68881 (either because the size of the frame is not recognized, or the revision number does not match the revision of this processor), then the MC68020 is instructed to take a format exception and the MC68881 enters the IDLE state. If the format word is valid, the appropriate state frame is loaded, starting at the specified location and up through higher addresses.

The FRESTORE does not normally affect the programmer's model registers of the MC68881 (except for the NULL state size, as described below); but, rather, is used only to restore the non-user visible portion of the machine. The FRESTORE instruction may be used with the FMOVEM instruction to perform a full context restoration of the MC68881, including the floating-point data registers and system control registers. In order to accomplish such a restoration, the FMOVEM instructions are first executed to load the programmer's model, followed by the FRESTORE instruction to load the internal state and continue any previously suspended operation. Refer to **4.3 Context Switching** for more information.

The current implementation of the MC68881 supports three state sizes. Refer to **4.3.2 State Frames** for more information on the exact format of these state sizes.

NULL: This state frame is four bytes long, with a format word of $0000. An FRESTORE with this size state frame is identical to a hardware reset of the MC68881. The programmer's model is set to the reset state, with non-signalling NANs in the floating-point data registers and zero in the FPCR, FPSR and FPIAR (thus, the programmer's model does not need to be loaded after this operation).

IDLE: This state frame is 28 ($1C) bytes long. An FRESTORE with this size state frame causes the MC68881 to restore itself to an idling condition, waiting for the initiation of the next instruction. Any exceptions that were pending at the time of the previous FSAVE will be pending after the FRESTORE. The programmer's model is not affected by the loading of this type of a state frame (although the completion of the suspended instruction after the restore is executed may modify the programmer's model).

BUSY: This state frame is 184 ($B8) bytes long. An FRESTORE with this size state frame causes the MC68881 to restore itself to the busy state, executing the instruction that was previously suspended by an FSAVE. The programmer's model is not affected by the loading of this type of a state frame.

FRESTORE Restore Internal State (Privileged Instruction) FRESTORE

Status Register: Cleared if the state size is NULL, otherwise not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			1	0	1	Effective Address Mode			Register		

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for the state frame. Only postincrement or control addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	—	—
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Operation: If in supervisor state
then MC68881 Internal State → State Frame
else trap

Assembler Syntax: FSAVE <ea>

Attributes: Unsized, privileged.

Description: The MC68881 suspends the execution of any operation that it was performing, and saves its internal state in a state frame located at the effective address. After the save operation, the MC68881 is in the idle state, waiting for the execution of the next instruction. The first word written to the state frame is the format word, which specifies the size of the frame and the revision number of this MC68881. The MC68020 initiates the FSAVE instruction by reading the MC68881 Save CIR, which will be encoded with a format word that indicates the appropriate action to be taken by the main processor. The current implementation of the MC68881 will always return one of five responses in the Save CIR:

Value	Definition
$0018	Save NULL state frame
$0118	Not Ready, come again
$0218	Illegal, take Format exception
$1F18	Save IDLE state frame
$1FB4	Save BUSY state frame

The Not Ready format word indicates that the MC68881 is not prepared to perform a state save and that the MC68020 should process interrupts, if necessary, and then re-read the Save CIR. The MC68881 uses this format word to cause the main processor to wait while an internal operation is completed, if possible, in order to allow an IDLE frame to be saved rather than a BUSY frame. The Illegal format word is used to abort an FSAVE operation that is attempted while the MC68881 was previously executing an FSAVE operation. All other format words cause the MC68020 to save the indicated state frame at the specified address. These state frames are defined as follows; for more information, refer to **4.3.2 State Frames**.

NULL: This state frame is four bytes long. An FSAVE of this size state frame indicates that the MC68881 state has not been modified since the last FRESTORE with a NULL state frame, or hardware reset. This indicates that programmer's model is in the reset state, with non-signalling NANs in the floating-point data registers and zero in the FPCR and FPSR (thus, it is not necessary to perform a save of the programmer's model).

IDLE: This state frame is 28 ($1C) bytes long. An FSAVE of this size state frame indicates that the MC68881 was in an idle condition, waiting for the initiation of the next instruction. Any exceptions that were pending are saved in the frame, and are then cleared internally. Thus, the pending exceptions will not be reported until after a subsequent FRESTORE of the state frame. In addition to being used for context switching, this frame may be used by exception handler routines, since it contains the value of the operand that caused the last floating-point exception to be taken.

BUSY: This state frame is 184 ($B8) bytes long. An FSAVE of this size state frame indicates that the MC68881 was at a point within an instruction where it was necessary to save the entire

internal state of the processor. This frame size is only used when absolutely necessary, because of the large size of the frame and the amount of time required to transfer it. The action of the MC68881 when this state frame is saved is same as for the IDLE state frame.

The FSAVE does not save the programmer's model registers of the MC68881; but, rather, is used only to save the non-user visible portion of the machine. The FSAVE instruction may be used with the FMOVEM instruction to perform a full context save of the MC68881, including the floating-point data registers and system control registers. In order to accomplish such a save, the FSAVE instruction is first executed to suspend the current operation and save the internal state, followed by the FMOVEM instructions to store the programmer's model. Refer to **4.3 Context Switching** for more information.

Status Register: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			1	0	0	Effective Address Mode			Register		

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for the state frame. Only pre-decrement or control alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number: An
(An)+	—	—
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Operation: FPn **x** INT(2^{Source}) → FPn

Assembler Syntax: FSCALE.<fmt> <ea>,FPn
FSCALE.X FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to an integer (if necessary) and add it to the destination exponent. Save the result in the destination floating-point data register. This function has the effect of multiplying the destination by 2^{Source}, but is much faster than multiplying the destination by 2^{Source} when the source is an integer value.

The MC68881 assumes that the scale factor is already an integer value before the operation. If not, the factor will be chopped (ie, rounded using the round-to-zero mode) to an integer before being added to the exponent. When the absolute value of the source operand is $\geq 2^{16}$, an overflow or underflow will always result.

Operation Table:

Destination \ Source		In Range	Zero	Infinity
		+ -	+ -	+ -
In Range	+ -	Scale Exponent	FPn[1]	NAN[2]
Zero	+	+0.0	+0.0	NAN[2]
	-	-0.0	-0.0	
Infinity	+	+inf	+inf	NAN[2]
	-	-inf	-inf	

Notes: 1. Returns the FPn as the result, however, the normal FSCALE algorithm is performed, including the rounding of the result to the defined precision. Therefore, it is possible for an underflow and/or inexact error to occur.
2. Sets the OPERR bit in the FPSR exception byte.
3. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if the source operand is ±infinity, cleared otherwise.
OVFL	Refer to **4.1.2.4**.
UNFL	Refer to **4.1.2.5**.
DZ	Cleared
INEX2	Cleared
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

FSCALE Scale Exponent FSCALE

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register			0	1	0	0	1	1	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn.

Operation: If (condition true)
then 1s → Destination
else 0s → Destination

Assembler Syntax: FBcc.<size> <ea>

3

Attributes: Size= (Byte)

Description: If the specified floating-point condition is true, set the byte integer operand at the destination to TRUE (all ones), otherwise set the byte to FALSE (all zeroes). The condition code may be any of the 32 floating-point conditional tests as described in **3.3 Conditional Test Definitions.**

Status Register:

Condition Codes: Not affected.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Set if the NAN condition code is set and the condition selected is an IEEE non-aware test.
SNAN	Not Affected
OPERR	Not Affected
OVFL	Not Affected
UNFL	Not Affected
DZ	Not Affected
INEX2	Not Affected
INEX1	Not Affected

Accrued Exception Byte: If the BSUN bit is set in the exception byte, then IOP is set in the accumulated exception byte. All other bits are not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	1	Effective Address Mode			Register		
0	0	0	0	0	0	0	0	0	0	Conditional Predicate					

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Specifies the addressing mode for the byte integer operand:

Addr. Mode	Mode	Register
Dn	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Conditional Predicate Field — Specifies one of 32 conditional tests as defined in section **3.3**.

Note: When a BSUN exception occurs, it causes a pre-instruction exeption to be taken by the main processor. If the exception handler returns without modifying the image of the PC on the stack frame (to point to the instruction following the FScc), then it must clear the cause of the exception (by clearing the NAN bit or disabling the BSUN trap) or the exception will occur again immediately upon return to the routine that caused the exception.

FSGLDIV Single Precision Divide FSGLDIV

Operation: FPn ÷ Source → FPn

Assembler Syntax: FSGLDIV.<fmt> <ea>,FPn
FSGLDIV.X FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary) and divide the destination floating-point data register by that value. Store the result in the destination floating-point data register. The result will be rounded to single precision.

Both the source and the destination operands are assumed to be representable as single precision values. If either operand requires more than 24 bits of mantissa to be accurately represented, the accuracy of the result is not guaranteed. This function is undefined for 0/0 and infinity/infinity.

Operation Table:

Destination \ Source		In Range		Zero		Infinity	
		+	-	+	-	+	-
In Range	+	Divide (single precision)		+inf[1]	-inf[1]	+0.0	-0.0
	-			-inf[1]	+inf[1]	-0.0	+0.0
Zero	+	+0.0	-0.0	NAN[2]		+0.0	-0.0
	-	-0.0	+0.0			-0.0	+0.0
Infinity	+	+inf	-inf	+inf	-inf	NAN[2]	
	-	-inf	+inf	-inf	+inf		

Notes: 1. Sets the DZ bit in the FPSR exception byte.
2. Sets the OPERR bit in the FPSR exception byte.
3. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set for 0/0 or infinity/infinity
OVFL	Refer to **4.1.2.4**.
UNFL	Refer to **4.1.2.5**.
DZ	Set if the source is zero and the destination is in range, cleared otherwise.
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

FSGLDIV Single Precision Divide FSGLDIV

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register			0	1	0	0	1	0	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn.

FSGLMUL Single Precision Multiply FSGLMUL

Operation: Source **x** FPn → FPn

Assembler Syntax:
FSGLMUL.<fmt> <ea>,FPn
FSGLMUL.X FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary) and multiply the destination floating-point data register by that value. Store the result in the destination floating-point data register. The result will be rounded to single precision.

Both the source and the destination operands are assumed to be representable as single precision values. Both operands are chopped to 24 bits of mantissa before the multiplication is performed.

Operation Table:

Destination \ Source		In Range		Zero		Infinity	
		+	-	+	-	+	-
In Range	+	Multiply (single precision)		+0.0	-0.0	+inf	-inf
	-			-0.0	+0.0	-inf	+inf
Zero	+	+0.0	-0.0	+0.0	-0.0	NAN [1]	
	-	-0.0	+0.0	-0.0	+0.0		
Infinity	+	+inf	-inf	NAN [1]		+inf	-inf
	-	-inf	+inf			-inf	+inf

Notes: 1. Sets the OPERR bit in the FPSR exception byte.
2. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if one operand is zero and the other is infinity, cleared otherwise.
OVFL	Refer to **4.1.2.4**.
UNFL	Refer to **4.1.2.5**.
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

FSGLMUL Single Precision Multiply FSGLMUL

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register			0	1	0	0	1	1	1

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn.

Operation: Sine of Source → FPn

Assembler Syntax:

	FSIN.<fmt>	<ea>,FPn
	FSIN.X	FPm,FPn
	FSIN.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary) and calculate the sine of that value. Return the result to the destination floating-point data register. The function is not defined for source operands of ±infinity. If the source operand is not in the range of [-2π...+2π], then the argument will be reduced to within that range before the sine is calculated. However, large arguments may lose accuracy during reduction, and very large arguments (greater than approximately 10^{20}) will lose all accuracy. The result will be in the range of [-1...+1].

Operation Table:

Source / Destination	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Sine		+0.0	-0.0	NAN [1]	

Notes: 1. Sets the OPERR bit in the FPSR exception byte.
2. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if the source is ±infinity, cleared otherwise.
OVFL	Cleared
UNFL	Set if a sine underflow occurs, in which case the cosine result is 1. Cosine can not underflow. Refer to **4.1.2.5**.
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	0	0	1	1	1	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: Sine of Source → FPs
Cosine of Source → FPc

Assembler Syntax:
FSINCOS.<fmt> <ea>,FPc:FPs
FSINCOS.X FPm,FPc:FPs

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

3

Description: Convert the source operand to extended precision (if necessary) and calculate both the sine and the cosine of that value. Both functions are calculated simultaneously, thus, this instruction is significantly faster than performing a separate FSIN and FCOS instruction. The sine result is loaded into the destination floating-point data register FPs; and the cosine result is loaded into the destination floating-point data register FPc. The condition code bits are set according to the sine result. If FPs and FPc are specified to be the same register, the cosine result is first loaded into the register, and then is over written with the sine result. The function is not defined for source operands of ±infinity.

If the source operand is not in the range of [-2π...+2π], then the argument will be reduced to within that range before the sine and cosine are calculated. However, large arguments may lose accuracy during reduction, and very large arguments (greater than approximately 10^{20}) will lose all accuracy. The results will be in the range of [–1...+1].

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
FPs	Sine		+0.0	-0.0	NAN [1]	
FPc	Cosine		+1.0	+1.0	NAN [1]	

Notes: 1. Sets the OPERR bit in the FPSR exception byte.
2. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1** (for the sine result).

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if the source is ±infinity, cleared otherwise.
OVFL	Cleared
UNFL	Refer to **4.1.2.5**.
DZ	Cleared
INEX2	Refer to **4.1.2.7.**
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

FSINCOS Simultaneous Sine and Cosine FSINCOS

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register, FPs				0	1	1	0	Destination Register, FPc	

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register, FPc Field — Specifies the destination floating-point data register, FPc. The cosine result will be stored in this register.

Destination Register, FPs Field — Specifies the destination floating-point data register, FPc. The sine result will be stored in this register. If FPc and FPs specify the same floating point data register, then the sine result will be left in that register, and the cosine result will be discarded.

If R/M=0 and the source register field is equal to either of the destination register fields, then the input operand is taken from the specified floating-point data register, and the appropriate result is then written into the same register.

Operation: Hyperbolic Sine of Source → FPn

Assembler Syntax:

FSINH.<fmt>	<ea>,FPn
FSINH.X	FPm,FPn
FSINH.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Convert the source operand to extended precision (if necessary) and calculate the hyperbolic sine of that value. Return the result to the destination floating-point data register.

Operation Table:

Destination \ Source	In Range	Zero		Infinity	
	+	+	-	+	-
Result	Hyperbolic Sine	+0.0	-0.0	+inf	-inf

Notes: 1. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Cleared
OVFL	Refer to **4.1.2.4**.
UNFL	Refer to **4.1.2.5**.
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	0	0	0	0	1	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: Square Root of Source → FPn

Assembler Syntax:
FSQRT.<fmt> <ea>,FPn
FSQRT.X FPm,FPn
FSQRT.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Convert the source operand to extended precision (if necessary) and calculate the square root of that number. Store the result in the destination floating-point data register. This function is not defined for negative source operands.

Operation Table:

Destination \ Source	In Range +	In Range -	Zero +	Zero -	Infinity +	Infinity -
Result	√x	NAN[1]	+0.0	-0.0	+inf	NAN[1]

Notes: 1. Sets the OPERR bit in the FPSR exception byte.
2. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if the source operand is not zero and is negative, cleared otherwise.
OVFL	Cleared
UNFL	Cleared
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	0	0	0	1	0	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: FPn – Source → FPn

Assembler Syntax:
FSUB.<fmt> <ea>,FPn
FSUB.X FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Convert the source operand to extended precision (if necessary) and subtract that number from the number in the destination floating-point data register. The result is stored in the destination floating-point data register.

Operation Table:

Destination \ Source		In Range		Zero		Infinity	
		+	-	+	-	+	-
In Range	+	Subtract		Subtract		-inf	+inf
	-						
Zero	+	Subtract		+0.0[1]	-0.0	-inf	+inf
	-			-0.0	+0.0[1]		
Infinity	+	+inf		+inf		NAN[2]	-inf
	-	-inf		-inf		-inf	NAN[2]

Notes: 1. Returns +0.0 in rounding modes RN, RZ and RP; returns -0.0 in RM.
2. Sets the OPERR bit in the FPSR exception byte.
3. If either operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if both the source and destination are like-signed infinities, cleared otherwise.
OVFL	Refer to **4.1.2.4**.
UNFL	Refer to **4.1.2.5**.
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier				Destination Register			0	1	0	1	0	0	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn.

Operation: Tangent of Source → FPn

Assembler Syntax:

FTAN.<fmt>	<ea>,FPn
FTAN.X	FPm,FPn
FTAN.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Convert the source operand to extended precision (if necessary) and calculate the tangent of that number. Store the result in the destination floating-point data register. The function is not defined for source operands of ±infinity. If the source operand is not in the range of $[-\pi/2...+\pi/2]$, then the argument will be reduced to within that range before the tangent is calculated. However, large arguments may lose accuracy during reduction, and very large arguments (greater than approximately 10^{20}) will lose all accuracy.

Operation Table:

Source / Destination	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Tangent		+0.0	-0.0	NAN [1]	

Notes: 1. Sets the OPERR bit in the FPSR exception byte.
2. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Set if the source is ±infinity, cleared otherwise.
OVFL	Refer to **4.1.2.4**.
UNFL	Refer to **4.1.2.5**.
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	0	0	1	1	1	1

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: Hyperbolic Tangent of Source → FPn

Assembler Syntax:
```
FTANH.<fmt>  <ea>,FPn
FTANH.X      FPm,FPn
FTANH.X      FPn
```

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Convert the source operand to extended precision (if necessary) and calculate the hyperbolic tangent of that number. Store the result in the destination floating-point data register.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Hyperbolic Tangent		+0.0	-0.0	+1.0	-1.0

Notes: 1. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Cleared
OVFL	Cleared
UNFL	Refer to **4.1.2.5**.
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	0	0	1	0	0	1

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

FTENTOX 10^X FTENTOX

Operation: $10^{Source} \rightarrow FPn$

Assembler Syntax:

FTENTOX.<fmt>	<ea>,FPn
FTENTOX.X	FPm,FPn
FTENTOX.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Convert the source operand to extended precision (if necessary) and calculate 10 to the power of that number. Store the result in the destination floating-point data register.

Operation Table:

Destination \ Source	In Range +	In Range -	Zero +	Zero -	Infinity +	Infinity -
Result	10^X		+1.0		+inf	+0.0

Notes: 1. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Cleared
OVFL	Refer to **4.1.2.4**.
UNFL	Refer to **4.1.2.5**.
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier				Destination Register			0	0	1	0	0	1	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

Operation: If condition true, then TRAP

Assembler Syntax:
FTRAPcc
FTRAPcc.W #<data>
FTRAPcc.L #<data>

Attributes: Size = (Word, Long)

Description: If the selected condition is true, the main processor initiates exception processing. The vector number is generated to reference the TRAPcc exception vector. The stacked program counter points to the next instruction. If the selected condition is not true, no operation is performed, and execution continues with the next instruction in sequence. The immediate data operand is placed in the next word(s) following the conditional predicate word and is available for user definition for use within the trap handler. The conditional test may be any one of the 32 conditional tests defined in **3.3 Conditional Test Definitions**.

Status Register:

Condition Codes: Not affected.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Set if the NAN condition code is set and the condition selected is an IEEE non-aware.
SNAN	Not Affected
OPERR	Not Affected
OVFL	Not Affected
UNFL	Not Affected
DZ	Not Affected
INEX2	Not Affected
INEX1	Not Affected

Accrued Exception Byte: If the BSUN bit is set in the exception byte, then IOP is set in the accumulated exception byte. All other bits are not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	1	1	1	1	Mode		
0	0	0	0	0	0	0	0	0	0	Conditional Predicate					
16-bit Operand or Most Significant Word of 32-bit Operand (if needed)															
Least Significant Word of 32-bit Operand (if needed)															

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Mode Field — Specifies the form of the instruction.
010 — The instruction is followed by a word operand.
011 — The instruction is followed by a long word operand.
100 — The instruction has no operand following it.

Conditional Predicate Field — Specifies one of 32 conditional tests as described in **3.3**.

Operand Field — Contains an optional word or long word operand that is user defined.

Note: When a BSUN exception occurs, it causes a pre-instruction exeption to be taken by the main processor. If the exception handler returns without modifying the image of the PC on the stack frame (to point to the instruction following the FTRAPcc), then it must clear the cause of the exception (by clearing the NAN bit or disabling the BSUN trap) or the exception will occur again immediately upon return to the routine that caused the exception.

Operation: Test Source Operand and Set the Floating-Point Condition Codes

Assembler Syntax:
FTST.<fmt> <ea>
FTST.X FPm

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Convert the source operand to extended precision (if necessary) and set the condition codes according to that number.

Operation Table: The contents of this table differ from the other operation tables. A letter in an entry of this table indicates that the designated condition code bit is always set by the FTST operation. All unspecified condition code bits are cleared during the operation.

Source / Destination	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	none	N	Z	NZ	I	NI

Notes: 1. If the source operand is a NAN, set the NAN condition code bit.
2. If the source operand is a SNAN, set the SNAN bit in the FPSR exception byte.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Cleared
OVFL	Cleared
UNFL	Cleared
DZ	Cleared
INEX2	Cleared
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier				Destination Register			0	1	1	1	0	1	0

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Since the MC68881 uses a common command word format for all of the arithmetic instructions (which FTST is considered one of) this field is treated in the same manner for FTST as for the other arithmetic instructions, even though the destination register is not modified. This field should be set to zero in order to maintain compatibility with future devices, although the MC68881 will not take an illegal instruction trap if it is not zero.

FTWOTOX 2^x FTWOTOX

Operation: $2^{Source} \rightarrow FPn$

Assembler Syntax:

	FTWOTOX.<fmt>	<ea>,FPn
	FTWOTOX.X	FPm,FPn
	FTWOTOX.X	FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Convert the source operand to extended precision (if necessary) and calculate 2 to the power of that number. Store the result in the destination floating-point data register.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	2^x		+1.0		+inf	+0.0

Notes: 1. If the source operand is a NAN, refer to **3.4.2.2** for more information.

Status Register:

Condition Codes: Affected as described in **3.4.2.3.1**.

Quotient Byte: Not affected.

Exception Byte:

BSUN	Cleared
SNAN	Refer to **3.4.2.2**.
OPERR	Cleared
OVFL	Refer to **4.1.2.4**.
UNFL	Refer to **4.1.2.5**.
DZ	Cleared
INEX2	Refer to **4.1.2.7**.
INEX1	If <fmt> is Packed, refer to **4.1.2.8**, cleared otherwise.

Accrued Exception Byte: Affected as described in **4.1.2.10**.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier				Destination Register			0	0	1	0	0	0	1

Instruction Fields:

Coprocessor ID Field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the MC68881.

Effective Address Field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeroes.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number: Dn
An	—	—
(An)	010	reg. number: An
(An)+	011	reg. number: An
-(An)	100	reg. number: An
(d_{16},An)	101	reg. number: An
(d_8,An,Xn)	110	reg. number: An
(bd,An,Xn)	110	reg. number: An
([bd,An,Xn],od)	110	reg. number: An
([bd,An],Xn,od)	110	reg. number: An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* Only if <fmt> is Byte, Word, Long or Single.

R/M Field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPm.
If R/M = 1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register Field — Specifies the destination floating-point data register, FPn. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers will set the source and destination fields to the same value.

3.5 INSTRUCTION ENCODING DETAILS

The following paragraphs provide the details for the object code formats for the general, branch, set on condition, save, and restore type coprocessor instructions.

3.5.1 Object Code Format

All MC68881 instructions are from two to eight words in length as shown below (the longest case is for an immediate operand of six words — the X or P format).

Operation Word
MC68881 Command Word, or Conditional Predicate
Effective Address Extension Words, Displacement, or Immediate Operand (If Any, 1-6 Words)

All MC68881 instructions contain an operation word, formatted as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor Identification			Type			Type Dependent					

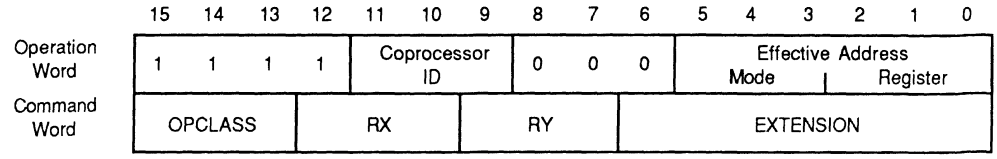
Coprocessor ID — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the MC68881.

Type – Specifies the type of coprocessor instruction:
000 — General Instruction (Arithmetics, FMOVE, FMOVEM)
001 — FDBcc, FScc, FTRAPcc
010 — FBcc.W
011 — FBcc.L
100 — FSAVE
101 — FRESTORE
110 — (Undefined, Reserved)
111 — (Undefined, Reserved)

Type Dependent — Normally specifies the effective address or conditional predicate, but usage depends on the Type field.

3.5.2 General Type Coprocessor Instruction Format

The general type coprocessor instruction format (shown below) is used for all MC68881 arithmetic, move, move multiple, move constant, and transcendental instructions.



The interpretation of the command word fields, OPCLASS, RX, RY, and instruction extension field varies with the instruction type and is summarized in Table 3-11.

Table 3-11. General Type Instruction Command Word Fields

OPCLASS	RX	RY	Instruction Class
000	Source, FPm	Destination, FPn	FPm to FPn. The extension field specifies the operation (move, add, etc.).
001	—	—	Undefined, reserved.
010	000 - 110 Source Data Format	Destination, FPn	Memory to FPn. The extension field specifies the operation (move, add, etc.).
	111	Destination, FPn	Move constant to FPn. The extension field contains the offset of the ROM constant.
011	Destination Data Format	Source, FPm	Move FPm to an external destination. If the destination format is packed decimal, the extension field specifies the k-factor (#k or Dn); otherwise it should be zero.
100	FPcr Select	000	Move single or multiple to the system control registers. The extension field should be zero.
101	FPcr Select	000	Move single or multiple system control registers to memory. The extension field should be zero.
110	Register list and addressing mode select.	00m	Move multiple to the floating-point data registers. The least significant bit of the RY field and the extension field contains the register list, or the number of the main processor data register that contains the list.
111	Register list and addressing mode select.	00m	Move multiple from the floating-point data registers. The least significant bit of the RY field and the extension field contains the register list, or the number of the main processor data register that contains the list.

The MC68881 general type instructions are classified into groups based upon instruction function and argument location (external or internal to the MC68881) as follows:

1. Floating-Point Register to Register
2. External Operand to Floating-Point Data Register
3. Move Constant to Floating-Point Data Register
4. Move Floating-Point Data Register to External Destination
5. Move System Control Register
6. Move Multiple Floating-Point Data Registers

Subdivision of the instruction set on this basis simplifies the specification of the MC68020 services required by each MC68881 instruction. The MC68881 requests services from the MC68020 via the coprocessor interface primitives described in **5.3 INSTRUCTION DIALOGS**.

If the command word indicates that an operand external to the MC68881 needs to be fetched or stored, the effective address field of the operation word is an MC68020 effective address descriptor. When the MC68881 requests an external data access, the MC68020 evaluates the source/destination effective address based upon this effective address descriptor and transfers operand(s) to/from the MC68881.

If all operands are contained in MC68881 floating-point data registers, the effective address field should be all zeros. No F-line emulator exception trap is taken if the effective address field is not all zeros; instruction execution proceeds normally. However, to ensure compatibility with future devices, assembler and compiler programmers should fill this field with zeros if it is not used.

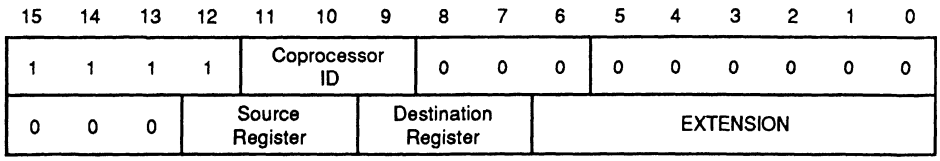
3.5.2.1 REGISTER-TO-REGISTER INSTRUCTIONS. This class of instructions includes floating-point data register to floating-point data register moves and the monadic and dyadic arithmetic and transcendental instructions. For dyadic arithmetic instructions, the destination operand is replaced by the result.

$$FPm \langle op \rangle FPn \rightarrow FPn$$

For monadic arithmetic instructions, the operand is the source FPm and the result is placed into the destination FPn. The source FPm and destination FPn may be the same floating-point data register.

$$FPm \langle op \rangle \rightarrow FPn$$

The instruction format for this instruction class is:



Source or Destination Register Field Encoding

000 - FP0	100 - FP4
001 - FP1	101 - FP5
010 - FP2	110 - FP6
011 - FP3	111 - FP7

The extension field indicates the operation to be performed. Table 3-12 lists the extension field encodings and functions. Also shown are the services requested of the MC68020 by the MC68881.

Table 3-12. Extension Field Encoding, Arithmetic Operations

Extension Field	Instruction Type	MC68020 Services
$00	FMOVE to FPn	Note 1
$01	FINT	Note 1
$02	FSINH	Note 1
$03	FINTRZ	Note 1
$04	FSQRT	Note 1
$06	FLOGNP1	Note 1
$08	FETOXM1	Note 1
$09	FTANH	Note 1
$0A	FATAN	Note 1
$0C	FASIN	Note 1
$0D	FATANH	Note 1
$0E	FSIN	Note 1
$0F	FTAN	Note 1
$10	FETOX	Note 1
$11	FTWOTOX	Note 1
$12	FTENTOX	Note 1
$14	FLOGN	Note 1
$15	FLOG10	Note 1
$16	FLOG2	Note 1
$18	FABS	Note 1
$19	FCOSH	Note 1
$1A	FNEG	Note 1

Table 3-12. Extension Field Encoding, Arithmetic Operations (Continued)

Extension Field	Instruction Type	MC68020 Services
$1C	FACOS	Note 1
$1D	FCOS	Note 1
$1E	FGETEXP	Note 1
$1F	FGETMAN	Note 1
$20	FDIV	Note 1
$21	FMOD	Note 1
$22	FADD	Note 1
$23	FMUL	Note 1
$24	FSGLDIV	Note 1
$25	FREM	Note 1
$26	FSCALE	Note 1
$27	FSGLMUL	Note 1
$28	FSUB	Note 1
$30-$37	FSINCOS	Note 1
$38	FCMP	Note 1
$3A	FTST	Note 1
$40-$7F	Unused, Reserved	Note 2

NOTES:

1. Two primitives may be issued for these operations. If the operation is register-to-register, the first primitive issued is null, with PC=1 to request that the MC68020 pass the current program counter if there are enabled exceptions. If the operation is external operand-to-register, then the first primitive is evaluate effective address and transfer data (with CA = 1 and PC=1 if exceptions are enabled). The second primitive is null (CA = 0) to terminate the instruction dialog.
2. The MC68881 will issue the take pre-instruction exception primitive with a vector number of 11 to instruct the MC68020 to take an F-line emulator trap.
3 Some extension field encodings are unspecified and are redundant with valid instructions implemented by the MC68881 and will not cause an F-line exception if executed. However, these encodings are reserved for future definition by Motorola, and thus should not be generated by assemblers or compilers. The redundant encodings are: $05, $07, $0B, $13, $17, $1B, $29–$2F, and $3B–$3F.

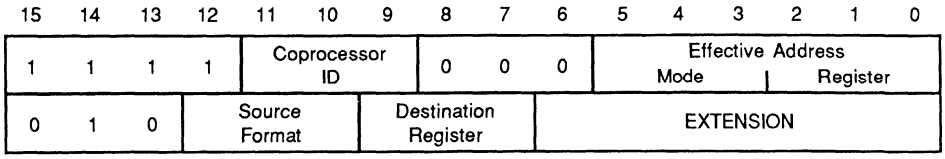
3.5.2.2 EXTERNAL OPERAND-TO-REGISTER INSTRUCTIONS. This class of instructions includes external operand to floating-point data register moves and arithmetic instructions. External operands may be located in memory or an MC68020 data register (for B, W, L, or S data types). Data format conversion from one of the seven memory data formats to the extended data form is implicit in these instructions. For dyadic arithmetic instructions, the number in FPn is replaced by the result.

External Operand <op> FPn → FPn

For monadic arithmetic instructions, the external operand is the source and the result is placed into the destination FPn.

External Operand <op> → FPn

The instruction format for this instruction class is shown below.



Destination Register Field Encoding

000 - FP0	100 - FP4
001 - FP1	101 - FP5
010 - FP2	110 - FP6
011 - FP3	111 - FP7

The source format field specifies the data format of the external operand. From the external operand are derived the length (in bytes) of the operand and the allowed effective addressing modes. The MC68881 decodes the source format field as listed in Table 3-13. The extension field indicates the operation to be performed. Table 3-12 lists the extension field encodings and functions. Also listed are services requested of the MC68020 by the MC68881.

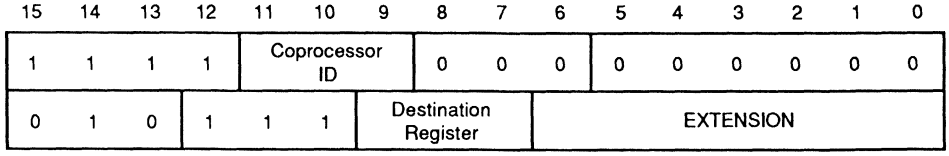
Table 3-13. Length and Allowed <ea> for External-to-Register Arithmetic Instructions

Source Format Encoding	External Operand DataFormat	Length in Bytes	Allowed <ea>
000	Long Word Integer	4	Note 1
001	Single Precision Real	4	Note 1
010	Extended Precision Real	12	Note 2
011	Packed Decimal Real	12	Note 2
100	Word Integer	2	Note 1
101	Double Precision Real	8	Note 2
110	Byte Integer	1	Note 1

NOTES:

1. Only data effective addressing modes are allowed.
2. Only memory effective addressing modes are allowed.

3.5.2.3 MOVE CONSTANT TO FLOATING-POINT DATA REGISTER INSTRUCTIONS. The MC68881 constant ROM contains frequently used constants such as 0.0 and π. These instructions permit the loading of a correctly rounded constant into a floating-point data register without an external data access. The instruction format for this instruction class is shown below.

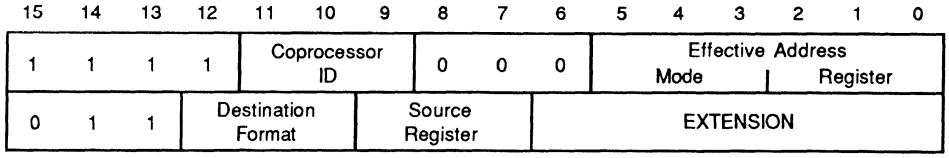


Destination Register Field Encoding

000 - FP0 100 - FP4
001 - FP1 101 - FP5
010 - FP2 110 - FP6
011 - FP3 111 - FP7

The extension field is used as an address into the MC68881 constant ROM. The FMOVECR instruction definition in 3.4.3 Individual Instruction Descriptions provides the valid extension field values for the FMOVECR instruction. The only service required by the MC68881 from the MC68020 is the passing of the MC68020 PC to FPIAR if exceptions are enabled (other than BSUN), requested with the null (CA = 1, PC = 1) primitive.

3.5.2.4 MOVE TO EXTERNAL DESTINATION INSTRUCTIONS. External destinations may be memory or an MC68020 data register. Data format conversion from the extended data format to one of the seven memory data formats is implicit for these instructions. The instruction format for this instruction class is shown below.



Source Register Field Encoding

000 - FP0 100 - FP4
001 - FP1 101 - FP5
010 - FP2 110 - FP6
011 - FP3 111 - FP7

The destination format field indicates the data format of the external destination. The MC68020 performs all transfers to an external destination at the request of the MC68881. When the MC68881 makes a request for a transfer to an external destination, the length (in bytes) of the operand, and the allowed effective addressing modes are specified in the primitive.

The MC68881 decodes the destination format field to determine the length of the operand to be stored and the allowed effective addressing modes as listed in Table 3–14.

Table 3-14. Length and Allowed <ea> for Register-to-External Instructions

Destination Format Encoding	External Operand DataFormat	Length in Bytes	Allowed <ea>
000	Long Word Integer	4	Note 1
001	Single Precision Real	4	Note 1
010	Extended Precision Real	12	Note 2
011	Packed Decimal Real with static k-factor	12	Note 2
100	Word Integer	2	Note 1
101	Double Precision Real	8	Note 2
110	Byte Integer	1	Note 1
111	Packed Decimal Real with dynamic k-factor	12	Note 2

NOTES:
1. Only data alterable effective addressing modes are allowed.
2. Only memory alterable effective addressing modes are allowed.

The extension field affects instruction execution only when the destination data format is packed decimal. A destination format encoding of 011 specifies a packed decimal string destination with the formatting parameter, k, in the extension field (encoded as a twos complement value).
A destination format encoding of 111 indicates a packed decimal string destination with the formatting parameter, k, in an MC68020 data register. The extension field contains the number of the MC68020 data register that contains the k-factor. The MC68020 data register number is encoded in bits 4 though 6 of the extension field; bits 0 through 3 should be zero. The seven least significant bits of the MC68020 data register contain a twos complement k-factor. The 25 most significant bits of the MC68020 data register are ignored. Table 3–15 lists the destination format field encodings, related extension field encodings, instruction operation, and the services requested of the MC68020 by the MC68881.

Table 3-15. Extension Field Encodings for Register-to-Memory Move Instructions

Destination Format Encoding	Extension Encoding	External Operand Data Format	MC68020 Services
0 0 0	0 0 0 0 0 0 0	Long Word Integer	Notes 1, 2
0 0 1	0 0 0 0 0 0 0	Single Precision Real	Notes 1, 2
0 1 0	0 0 0 0 0 0 0	Extended Precision Real	Notes 1, 2
0 1 1	k k k k k k k	Packed Decimal Real with a Static k Factor	Note 1
1 0 0	0 0 0 0 0 0 0	Word Integer	Notes 1, 2
1 0 1	0 0 0 0 0 0 0	Double Precison Real	Notes 1, 2
1 1 0	0 0 0 0 0 0 0	Byte Integer	Notes 1, 2
1 1 1	r r r 0 0 0 0	Packed Decimal with a Dynamic k Factor	Note 3

NOTES:

1. Four service requests may be issued for this instruction type:
   a. Null (CA = 1, PC = x) may be first used to request the transfer of the PC to the FPIAR if exceptions are enabled.
   b. Null (CA = 1, IA = 1) is used to force the MC68020 to wait while the conversion takes place.
   c. Evaluate effective address and transfer data (CA = 1) is issued to request the transfer of the converted operand.
   d. Null (CA = 0) is used to terminate the dialog if no exception occurred. If an exception occurred, the take mid-instruciton exception primitive is used to terminate the dialog.
2. The extension field should be all zeros; although no F-line emulator trap is taken if it is not. Assemblers and compilers should fill the extension field with zeros to ensure compatibility with future devices.
3. Bits 0 through 3 of the extension field should be zero, although no F-line emulator trap is taken if not. Assemblers and compilers should set these bits to zero to assure compatibility with future devices. Four service requests are issued for this instruction:
   a. Transfer single main processor register (CA = 1, PC = x) is first used to request the transfer of the PC to the FPIAR (if exceptions are enabled) and to transfer the MC68020 data register containing the k factor.
   b. Null (CA = 1, IA = 1) is used to force the MC68020 to wait while the conversion takes place.
   c. Evaluate effective address and transfer data (CA = 1) is issued to request the transfer of the converted operand.
   d. Null (CA = 0) is used to terminate the dialog if no exceptions occurred. If an exception occurred, the take mid-instruction exception primitive is used to terminate the dialog.

3.5.2.5 MOVE SYSTEM CONTROL REGISTER INSTRUCTIONS. This class of instructions includes the move single system control register instruction and the move multiple system control registers instruction. For the move single system control register instruction, external 32-bit operands may be immediate, in memory or an MC68020 register. For the move multiple system control register instructions, external operands may only be immediate or in memory (immediate addressing is only allowed if dr = 0). The instruction format for this class of instructions is shown below.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Effective Address Register		
1	0	dr		Register List			0	0	0	0	0	0	0	0	0

The dr bit indicates a read of the MC68881 (1) or a write to the MC68881 (0). The register select field specifies the system control register or registers to be moved during the operation. Table 3-16 lists the dr and register list field encodings, instruction operation, operand size, allowed effective addressing modes, and services required of the MC68020 by the MC68881 for this instruction type.

Bits 0 to 9 of the command word should be zero, although no F-line trap will be taken if they are not. Assemblers and compilers should set these bits to zeros to ensure compatibility with future devices.

3.5.2.6 MOVE MULTIPLE FLOATING-POINT DATA REGISTERS INSTRUCTIONS. This class of instructions provides move multiple floating-point data register operations analogous to the M68000 move multiple address and data registers instructions. Unlike the integer counterpart, the floating-point register list can be specified either statically in the instruction or dynamically in an MC68020 data register.

The addressing modes for the move multiple from memory to floating-point data registers instruction are restricted to the control and address register indirect with postincrement effective addressing modes.

Table 3-16. Encoding for Move FPcr Operations

dr Bit	Register List	Instruction Operation	Transfer Size (in Bytes)	Allowed <ea>	MC68020 Services
0	000	(Undefined, Reserved)	—	—	Notes 1,2
0	001	Move to FPIAR	4	Any	Note 1
0	010	Move to FPSR	4	Data	Note 1
0	011	Move to FPSR and FPIAR	8	Memory	Note 1
0	100	Move to FPCR	4	Data	Note 1
0	101	Move to FPCR and FPIAR	8	Memory	Note 1
0	110	Move to FPCR and FPSR	8	Memory	Note 1
0	111	Move to FPCR, FPSR, and FPIAR	12	Memory	Note 1
1	000	(Undefined, Reserved)	—	—	Notes 1,2
1	001	Move from FPIAR	4	Alterable	Note 1
1	010	Move from FPSR	4	Data Alterable	Note 1
1	011	Move from FPSR and FPIAR	8	Memory Alterable	Note 1
1	100	Move from FPCR	4	Data Alterable	Note 1
1	101	Move from FPCR and FPIAR	8	Memory Alterable	Note 1
1	110	Move from FPCR and FPSR	8	Memory Alterable	Note 1
1	111	Move to FPCR, FPSR, and FPIAR	12	Memory Alterable	Note 1

NOTES:

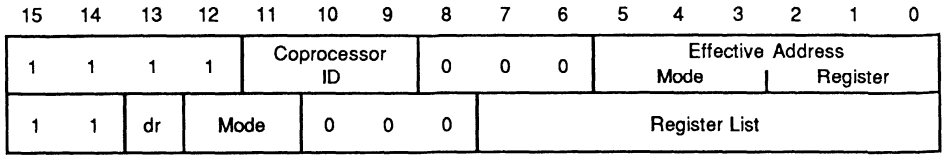
1. This operation requires two primitives to be issued to the MC68020. Thefirst primitive is evaluate effective address and transfer data (CA = 1), indicating the appropriate transfer size and allowed effective addressing mode. The second primitive is null (CA = 0) to terminate the instruction dialog.
2. For the current implementation of the MC68881, this encoding is redundant with the 001 encoding of the register select field (i.e., it selects the FPIAR as the only register to be moved); however, this encoding is reserved for future use by Motorola.

The addressing modes for the move multiple from floating-point data registers to memory instruction are restricted to the control alterable and address register indirect with predecrement effective addressing modes.

NOTE

The effective addressing mode restrictions for this instruction are enforced by the MC68020 when the transfer multiple coprocessor registers response primitive is received (not by the MC68881 when it receives the command word). If the encoding of the effective address field in the operation word is inconsistent with the encoding of the dr and mode fields in the command word, unexpected results will occur. In some cases, the instruction will be executed, but the order of the register transfer will be the reverse of the appropriate order for the addressing mode. However, system integrity is preserved for all cases.

The instruction format for this class of instructions is shown below.



The dr bit indicates a read of the MC68881 (1) or a write to the MC68881 (0). The mode field specifies the order of the register transfer and the location of the register list. The definitions of the mode field bits are:

0x Transfer FP7 through FP0.
1x Transfer FP0 through FP7.
x0 Register List is Static.
x1 Register List is Dynamic.

The order of the register transfer that is selected affects the interpretation of the register list, because the list is always scanned starting with the most significant bit. Thus, for the 0x encoding of the mode field, the most significant bit of the register list corresponds to FP7, and the least significant bit corresponds to FP0. For the 1x encoding, this relationship is reversed.

The type of the register list also affects the interpretation of the register list field. If a static list is selected, then the register list field of the command word contains the register list. If a dynamic register list is selected, then the register list field of the command word contains the number of the MC68020 data register that contains the register list. The format of the register list field in the command word for the various mode field encodings is shown below. If a bit in the register list is set, then the corresponding register is moved, otherwise the list is scanned for the next bit that is set. For the dynamic register list format, "rrr" specifies the MC68020 data register that contains the register list. The format of a dynamic register list is the same as the format of the appropriate static list, and it is contained in the least significant eight bits of the MC68020 data register.

Mode Field Encoding		**Register List Field Format**							
00	—	FP7	FP6	FP5	FP4	FP3	FP2	FP1	FP0
10	—	FP0	FP1	FP2	FP3	FP4	FP5	FP6	FP7
x1	—	0	r	r	r	0	0	0	0

Table 3-17 lists the dr and mode field encodings, instruction operation, allowed effective addressing modes, and services required of the MC68020 by the MC68881 for this instruction type.

Table 3-17. Encodings for Move Multiple FPn Operations

dr Bit	Mode Field	Instruction Operation	Allowed <ea> Modes	MC68020 Services
0	00	(Invalid operation)	—	Note 1
0	01	(Invalid operation)	—	Note 1
0	10	Move to registers, static register list	(An)+ or Control	Note 2
0	11	Move to registers, dynamic register list	(An)+ or Control	Note 3
1	00	Move from registers, static register list	–(An)	Note 2
1	01	Move from registers, dynamic register list	–(An)	Note 3
1	10	Move from registers, static register list	Control Alterable	Note 2
1	11	Move from registers, dynamic register list	Control Alterable	Note 3

NOTES:

1) These encodings cause the MC68881 to perform an operation that is inconsistent with the M68000 Family move multiple operations. For these cases, the selected registers are transferred in the order that is appropriate for the pre-decrement addressing mode (ie., FP7 though FP0), using a static or dynamic register list, respectively. However, the MC68020 does not allow the pre-decrement addressing mode for a move from memory to multiple coprocessor registers operation. Thus, assemblers and compilers should not generate these encodings, or unexpected results may occur.
2) This instruction requires two primitives; the first is the transfer multiple coprocessor registers (CA = 1) primitive to request that the MC68020 evaluate the effective address, read the register select CIR, and transfer the number of registers indicated by the mask (with an operand size of 12 bytes for each register). The second primitive is null(CA = 0), which is used to terminate the dialog.
3) This instruction requires three primitives; the first is the transfer single main processor register (CA = 1) primitive to request the transfer of the MC68020 data register that contains the dynamic register list. The second is the transfer multiple coprocessor registers (CA = 1) primitive to request that the MC68020 evaluate the effective address, read the register select CIR, and transfer the number of registers indicated by the mask (with an operand size of 12 bytes for each register). The third primitive is null (CA = 0) to terminate the dialog.

3.5.2.7 UNDEFINED, RESERVED COMMAND WORDS. The command word encoding shown below is undefined and reserved for future Motorola use. All undefined, reserved command word encodings generate an F-line emulator exception. Additional command word encodings which generate an F-line emulator exception are specified in Table 3-17.

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

3.5.3 FDBcc, FScc, and FTRAPcc Instruction Formats

These instructions all use the same operation word type field encoding and command word format as shown below. The instruction specific field of the operation word determines the instruction variation and is defined in Table 3-18 for each instruction type.

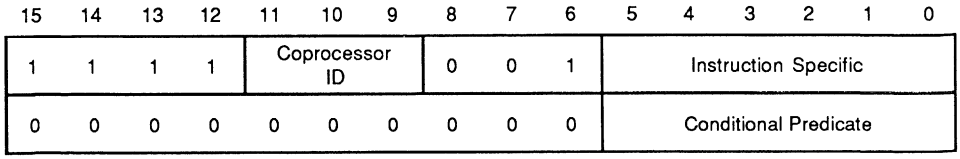


Table 3-18 Encodings for the FDBcc, FScc, and FTRAPcc Instructions

Instruction Specific Field	Instruction Operation		Selected <ea>	MC68020 Services
000 xxx	FScc	<ea>	Dn	(Note 1)
001 xxx	FDBcc	Dn,<label>	—	(Note 2)
010 xxx	FScc	<ea>	(An)	(Note 1)
011 xxx	FScc	<ea>	(An)+	(Note 1)
100 xxx	FScc	<ea>	-(An)	(Note 1)
101 xxx	FScc	<ea>	d_{16}(An)	(Note 1)
110 xxx	FScc	<ea>	indexed/indirect	(Note 1)
111 000	(Undefined, reserved)		—	(Note 3)
111 001	(Undefined, reserved)		—	(Note 3)
111 010	FTRAPcc.W	#<data>	—	(Note 4)
111 011	FTRAPcc.L	#<data>	—	(Note 4)
111 100	FTRAPcc with no parameter		—	(Note 4)
111 101	(Undefined, reserved)		—	(Note 3)
111 110	(Undefined, reserved)		—	(Note 3)
111 111	(Undefined, reserved)		—	(Note 3)

NOTES:

1) The MC68020 evaluates the <ea> and writes the conditional predicate to the MC68881 for evaluation. The null (CA = 0) primitive is used to return the true/false evaluation, and the appropriate value is then written to the the <ea> by the MC68020. The value of xxx specifies the MC68020 data or address register (Dn or An) used in the <ea> evaluation.

2) The MC68020 writes the conditional precicate to the MC68881 for evaluation. The null (CA = 0) primitive is used to return the true/false evaluation. If the condition is true, then the MC68020 proceeds to the next instruction, otherwise, the counter register Dn.W (specified by the value of xxx) is decremented, and the new value is compared with –1. If it is equal to –1, then the MC68020 proceeds to the next instruction; otherwise, the 16–bit displacement is sign extended and added to the PC.

3) The MC68020 takes an F-line emulation trap.

4) The MC68020 writes the conditional precicate to the MC68881 for evaluation. The Null (CA = 0) primitive is used to return the true/false evaluation. If the condition is true, then the cpTRAPcc exception is taken. Otherwise, the MC68020 will proceed to the next instruction, discarding the optional immediate operand if necessary.

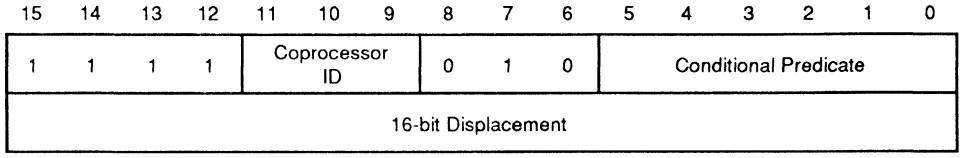
For the FDBcc, FTRAPcc.W, and FTRAPcc.L forms of this instruction class, the displacement or operand word(s) follows immediately after the conditional predicate word.

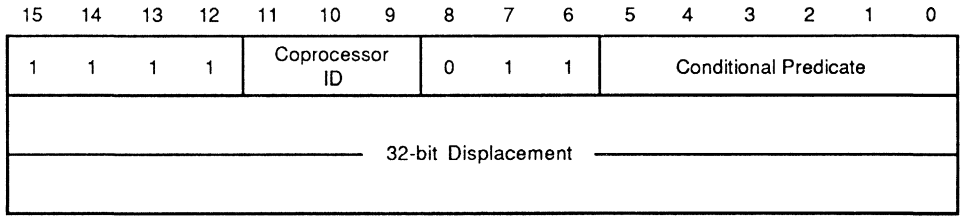
NOTE

From the perspective of the MC68881, these instructions are identical to the branch type coprocessor instructions. The various operations are handled by the MC68020 in a manner that is transparent to the MC68881.

3.5.4 Conditional Branch Instruction Format

For this instruction type, the MC68020 writes a conditional predicate to the MC68881 condition CIR for evaluation. The MC68881 determines whether the conditional predicate is true or false based on the floating-point condition codes as described in **3.3 Conditional Test Definitions**. The true or false result is returned to the main processor with the null primitive. The formats for this instruction type are shown below.





The conditional predicate field specifies the conditional test to be performed. Table 3–19 lists the conditional predicate encodings and the MC68881 responses. For details on how the true or false response is calculated, refer to **3.3 Conditional Test Definitions**.

The displacement is a twos complement integer which indicates the relative distance in bytes from the displacement word(s) (i.e., the PC value used in the branch destination calculation is the address of the displacement word(s)). A 16-bit displacement is sign extended before it is used in the branch destination calculation.

NOTE

From the perspective of the MC68881, the two forms of this instruction are identical. The size of the displacement is determined by the MC68020 and is

transparent to the MC68881. Also, the FNOP instruction syntax that is recognized by Motorola assemblers generates an FBcc.W instruction with cc = F (false) and a displacement value of zero.

Table 3-19. Conditional Predicate Evaluation Responses

Conditional Predicate	Mnemonic	Definition	MC68881 Response
000000	F	False	Note 1
000001	EQ	Equal	Note 1
000010	OGT	Ordered Greater Than	Note 1
000011	OGE	Ordered Greater Than or Equal	Note 1
000100	OLT	Ordered Less Than	Note 1
000101	OLE	Ordered Less Than or Equal	Note 1
000110	OGL	Ordered Greater or Less Than	Note 1
000111	OR	Ordered	Note 1
001000	UN	Unordered	Note 1
001001	UEQ	Unordered or Equal	Note 1
001010	UGT	Unordered or Greater Than	Note 1
001011	UGE	Unordered or Greater or Equal	Note 1
001100	ULT	Unordered or Less Than	Note 1
001101	ULE	Unordered or Less or Equal	Note 1
001110	NE	Not Equal	Note 1
001111	T	True	Note 1
010000	SF	Signaling False	Note 2
010001	SEQ	Signaling Equal	Note 2
010010	GT	Greater Than	Note 2
010011	GE	Greater Than or Equal	Note 2
010100	LT	Less Than	Note 2
010101	LE	Less Than or Equal	Note 2
010110	GL	Greater or Less Than	Note 2
010111	GLE	Greater Less or Equal	Note 2
011000	NGLE	Not (Greater, Less or Equal)	Note 2
011001	NGL	Not (Greater or Less)	Note 2
011010	NLE	Not (Less or Equal)	Note 2
011011	NLT	Not (Less Than)	Note 2
011100	NGE	Not (Greater or Equal)	Note 2
011101	NGT	Not (Greater Than)	Note 2
011110	SNE	Signaling Not Equal	Note 2
011111	ST	Signaling True	Note 2
1xxxxx	—	—(Undefined, Reserved)—	Note 3

NOTES:

1. Indicate the condition true or false result by using the null (CA = 0) primitive.
2. If the NAN condition code bit is set, then set the BSUN bit in the FPSR. If the BSUN trap is enabled, then return the take pre-instructionexception primitive with the BSUN vector number; otherwise, indicate the condition true/false result by using the null (CA = 0) primitive.
3. Not used, redundant encodings with 0xxxxx. No F-line trap is taken if these bit patterns are used. To ensure compatibility with future devices, assemblers and compilers should use the 0xxxxx encodings.

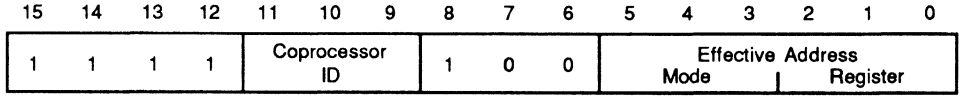
For all instructions, the conditional predicate field specifies the conditional test to be evaluated. Table 3-19 lists the conditional predicate encodings and the MC68881 responses. For details on how the true or false response is calculated, refer to **3.3 Conditional Test Definitions.**

NOTE

Bits 6 through 15 of the command word are shown to be filled with zeros; however no F-line emulator trap is taken if they are not. To ensure compatibility with future devices, assemblers and compilers should fill this field with zeros.

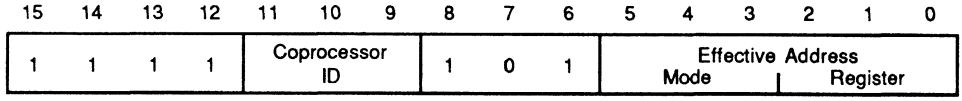
3.5.5 Save Instruction Format

The FSAVE instruction indicates that the MC68881 must immediately suspend any current operation and saving the internal state in memory. Effective addressing modes are restricted to control alterable and address register indirect with predecrement modes.



3.5.6 Restore Instruction Format

The MC68881 restore instruction indicates that regardless of the current state of operation, a new internal state is to be loaded immediately. Effective addressing modes are restricted to control and address register indirect with postincrement modes.



3.6 INSTRUCTION FORMAT SUMMARY

The following paragraphs present a summary of the binary encodings for the MC68881 instruction set. The unique encoding for each instruction is shown explicitly, with the encoded fields common to all of the instructions detailed in a single table at the beginning of this section.

3.6.1 Coprocessor ID Field

This field of each instruction specifies which one of eight possible coprocessors in a system is to perform the operation. There are no restrictions placed on the value of the ID field by the main processor in the system; however, certain conventions should be followed. Motorola assemblers default to coprocessor ID = 1 for the MC68881, although directives are available to change this default. Furthermore, due to the hardware implementation of the MC68851 paged memory management unit, that device **must** be assigned to coprocessor ID = 0 if it is used in a system. Thus the MC68881 should not be assigned to coprocessor ID = 0 if it is anticipated that an MC68851 may be used in a system.

3.6.2 Effective Address Field

This field specifies the M68000 Family addressing mode that is to be used to locate operands external to the MC68881 (if required by the instruction). For some operations, restrictions are placed on which of the available addressing modes are allowed to be used. These restrictions are enforced by hardware in the MC68020 and MC68881, and Motorola assemblers will not generate operation words with disallowed effective addressing mode field encodings. The encoding for this fields are shown in Table 3-20.

3.6.3 Register/Memory Field

This field is common to all of the arithmetic instructions and the FMOVE to FPn instruction. A zero in this field indicates that the operation is register-to-register, while a one in this field indicates that the source operand is external to the MC68881.

Table 3-20. Effective Address Field Encoding Summary

Address Modes	Mode	Register	Data	Memory	Control	Alterable	Assembler Syntax
Data Register Direct	000	reg. no.	X	–	–	X	Dn
Address Register Direct	001	reg. no.	–	–	–	X	An
Address Register Indirect	010	reg. no.	X	X	X	X	(An)
Address Register Indirect with Postincrement	011	reg. no.	X	X	–	X	(An)+
Address Register Indirect with Predecrement	100	reg. no.	X	X	–	X	–(An)
Address Register Indirect with Displacement	101	reg. no.	X	X	X	X	(d16,An)
Address Register Indirect with Index (8-Bit Displacement)	110	reg. no.	X	X	X	X	(d8,An,Xn)
Address Register Indirect with Index (Base Displacement)	110	reg. no.	X	X	X	X	(bd,An,Xn)
Memory Indirect Post-Indexed	110	reg. no.	X	X	X	X	([bd,An],Xn,od)
Memory Indirect Pre-Indexed	110	reg. no.	X	X	X	X	([bd,An,Xn],od)
Absolute Short	111	000	X	X	X	X	(xxx).W
Absolute Long	111	001	X	X	X	X	(xxx).L
Program Counter Indirect with Displacement	111	010	X	X	X	–	(d16,PC)
Program Counter Indirect with Index (8-Bit Displacement)	111	011	X	X	X	–	(d8,PC,Xn)
Program Counter Indirect with Index (Base Displacement)	111	011	X	X	X	–	(bd,PC,Xn)
PC Memory Indirect Post-Indexed	111	011	X	X	X	–	([bd,PC],Xn,od)
PC Memory Indirect Pre-Indexed	111	011	X	X	X	–	([bd,PC,Xn],od)
Immediate	111	100	X	X	–	–	#<data>

3.6.4 Source Specifier Field

This field is common to all of the arithmetic instructions and the FMOVE floating-point data register instruction. The definition of this field is affected by the value of the R/M field as shown below:

If R/M = 0, it specifies the source floating-point data register, FPm.

If R/M = 1, it specifies the source operand data format.

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

3.6.5 Destination Register Field

This field is common to all of the arithmetic instructions and the FMOVE to FPn instruction. This field specifies the floating-point data register that is to be used as the destination. The result of an operation is always stored in this register, and one of the source operands will come from this register for dyadic instructions.

3.6.6 Conditional Predicate Field

This field is common to all of the conditional instructions and specifies the MC68881 conditional test that is to be evaluated for the main processor. Table 3-21 shows the conditional predicate binary encodings for the 32 conditional tests supported by the MC68881.

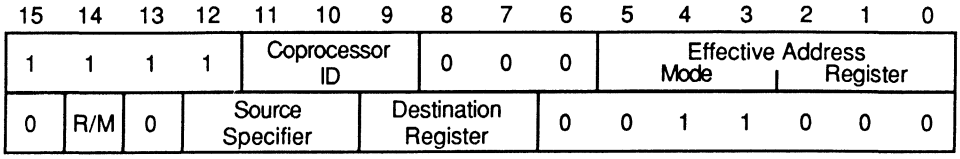
Table 3-21. Conditional Predicate Field Encoding Summary

Conditional Predicate	Mnemonic	Definition
000000	F	False
000001	EQ	Equal
000010	OGT	Ordered Greater Than
000011	OGE	Ordered Greater Than or Equal
000100	OLT	Ordered Less Than
000101	OLE	Ordered Less Than or Equal
000110	OGL	Ordered Greater or Less Than
000111	OR	Ordered
001000	UN	Unordered
001001	UEQ	Unordered or Equal
001010	UGT	Unordered or Greater Than
001011	UGE	Unordered or Greater or Equal
001100	ULT	Unordered or Less Than
001101	ULE	Unordered or Less or Equal
001110	NE	Not Equal
001111	T	True
010000	SF	Signaling False
010001	SEQ	Signaling Equal
010010	GT	Greater Than
010011	GE	Greater Than or Equal
010100	LT	Less Than
010101	LE	Less Than or Equal
010110	GL	Greater or Less Than
010111	GLE	Greater Less or Equal
011000	NGLE	Not (Greater, Less or Equal)
011001	NGL	Not (Greater or Less)
011010	NLE	Not (Less or Equal)
011011	NLT	Not (Less Than)
011100	NGE	Not (Greater or Equal)
011101	NGT	Not (Greater Than)
011110	SNE	Signaling Not Equal
011111	ST	Signaling True

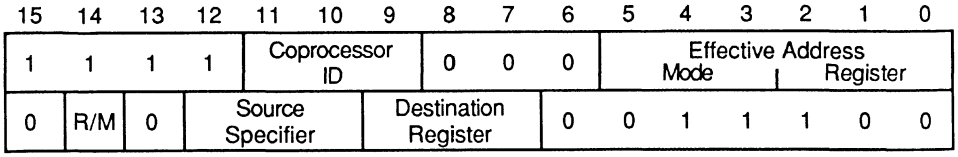
3.6.7 Instruction Format Diagrams

The instruction formats are summarized below.

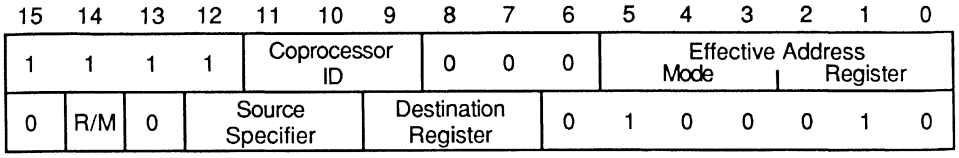
FABS



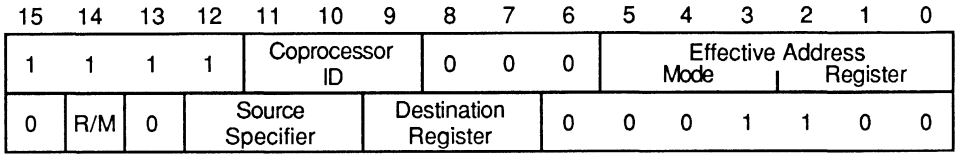
FACOS



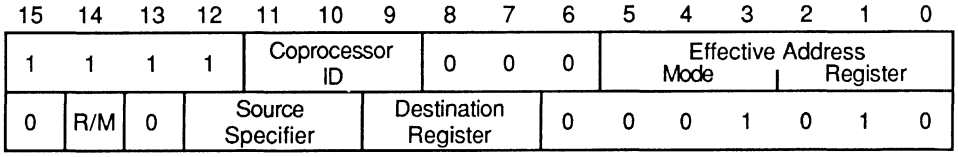
FADD



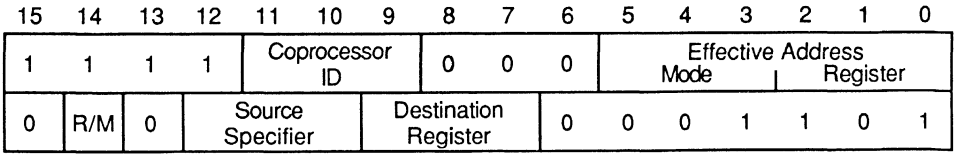
FASIN



FATAN



FATANH



FBcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	1	Size	Conditional Predicate					
16-bit Displacement, or Most Significant Word of 32-bit Displacement															
Least Significant Word of 32-bit Displacement (if needed)															

Size Field — Specifies the size of the twos complement displacement:
Size = 0 — Displacement is 16-bits and will be sign extended.
Size = 1 — Displacement is 32-bits.

3

FCMP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	1	1	1	0	0	0

FCOS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	0	1	1	1	0	1

FCOSH

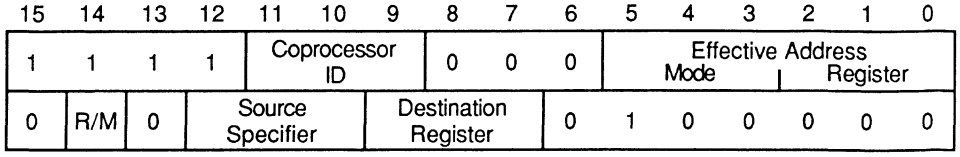
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register				0	0	1	1	0	0	1

FDBcc

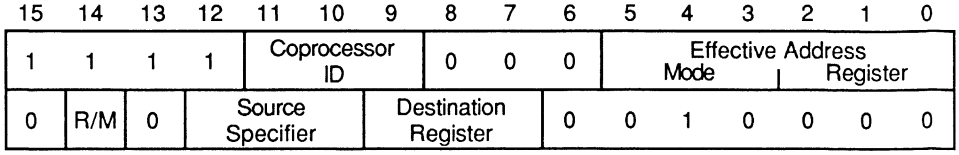
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	1	0	0	1	Count Register		
0	0	0	0	0	0	0	0	0	0	Conditional Predicate					
16-bit Displacement															

Count Register Field — Specifies the main processor data register to be decremented.

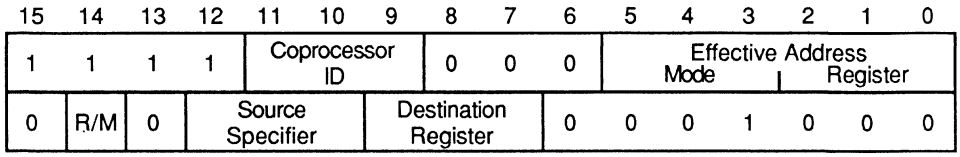
FDIV



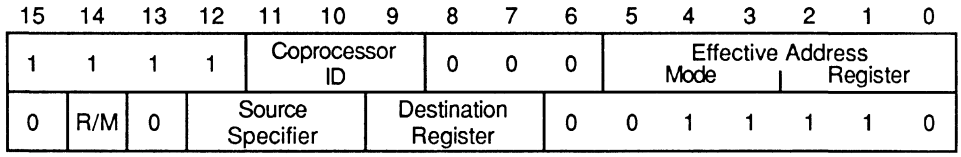
FETOX



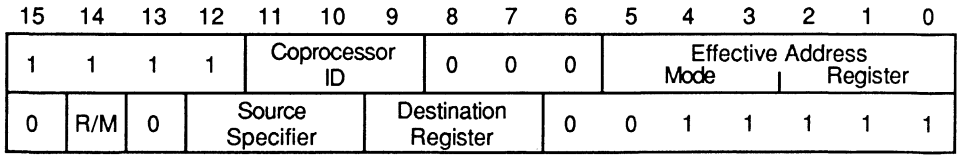
FETOXM1



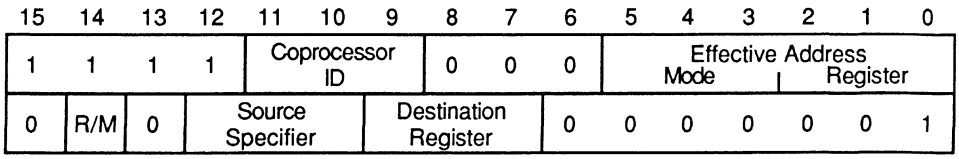
FGETEXP



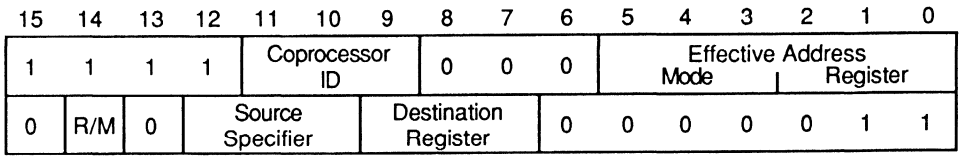
FGETMAN



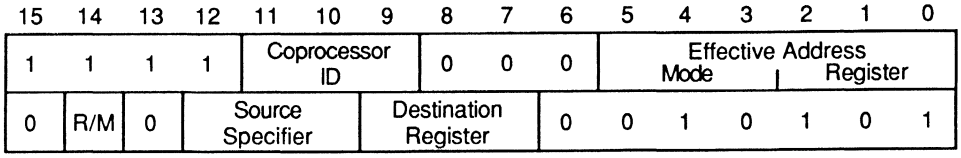
FINT



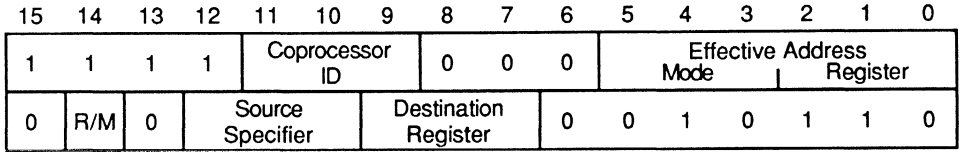
FINTRZ



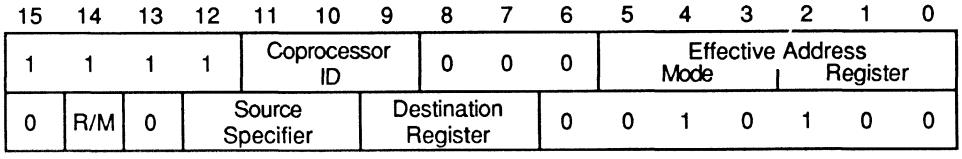
FLOG10



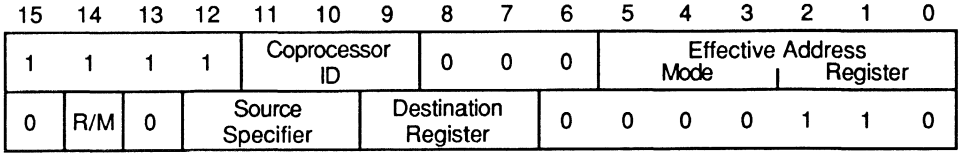
FLOG2



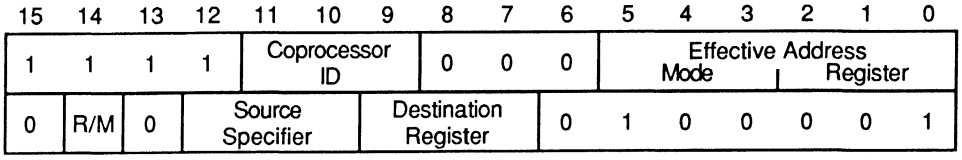
FLOGN



FLOGNP1



FMOD



FMOVE to FPn

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier				Destination Register			0	0	0	0	0	0	0

FMOVE from FPn

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	1	1	Destination Format			Source Register			k-factor (if required)						

Destination Format Field — Specifies the data format of the destination operand as follows:

000 — Long Word Integer
001 — Single Precision Real
010 — Extended Precision Real
011 — Packed Decimal Real, static k-factor
100 — Word Integer
101 — Double Precision Real
110 — Byte Integer
111 — Packed Decimal Real, dynamic k-factor

k-factor Field — Specifies the format of the packed decimal string to be generated (if the Destination Format field indicates packed decimal), or the number of the main processor data register that contains the format specification. The interpretation of the k-factor is:

-64 to 0 — Number of significant digits to the right of the decimal point.
+1 to +17 — Number of significant digits in the mantissa.
+18 to +63 — Sets the OPERR bit, treated as +17.

The format of this field for a dynamic k-factor is:

rrr0000

Where "rrr" is the number of the main processor data register that contains the k-factor.

FMOVE FPcr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
1	0	dr	Register Select			0	0	0	0	0	0	0	0	0	0

dr Field — Specifies the direction of the transfer:
0 — Move memory to system control register.
1 — Move system control register to memory.

Register Select Field — Specifies the system control register to be moved:
001 — FPIAR
010 — FPSR
100 — FPCR

FMOVECR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	0	0	0	0	0	0
0	1	0	1	1	1	Destination Register				ROM offset					

ROM offset Field — Specifies the offset in the the MC68881 Constant ROM where the desired constant is located.

FMOVEM FPn

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
1	1	dr	Mode		0	0	0	Register List							

dr Field — Specifies the direction of the transfer.
0 — Move the listed registers from memory to the MC68881.
1 — Move the listed registers from the MC68881 to memory.

Mode Field — Specifies the type of the register list and addressing mode.
00 — Static register list, predecrement addressing mode.
01 — Dynamic register list, predecrement addressing mode.
10 — Static register list, postincrement or control addressing mode.
11 — Dynamic register list, postincrement or control addressing mode.

Register List Field:

Static list — contains the select mask; if a register is to be moved, the corresponding bit in the list is set, otherwise it is clear.

Dynamic list — contains the main processor data register number, rrr, as shown below.

		Register List Format							
Static, -(An)	—	FP7	FP6	FP5	FP4	FP3	FP2	FP1	FP0
Static, (An)+ or Control	—	FP0	FP1	FP2	FP3	FP4	FP5	FP6	FP7
Dynamic	—	0	r	r	r	0	0	0	0

The format of the dynamic list mask is the same as for the static list, and is contained in the least significant 8-bits of the specified MC68020 data register.

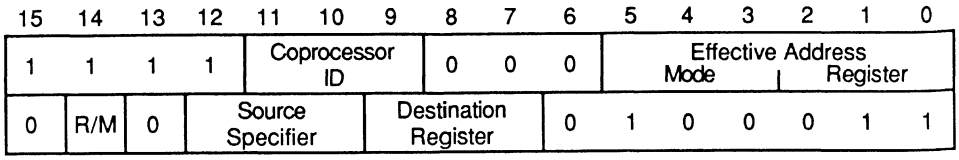
FMOVEM FPcr

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
1	0	dr	Register List			0	0	0	0	0	0	0	0	0	0

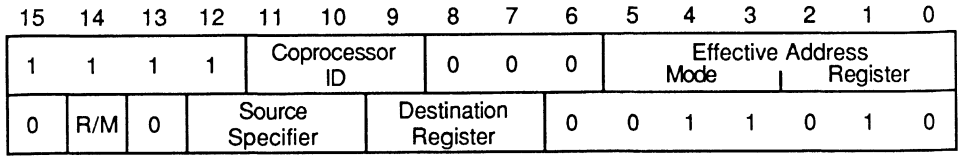
dr Field — Specifies the direction of the transfer:
0 — Move memory to system control registers.
1 — Move system control registers to memory.

Register List Field — Specifies the system control registers to be moved:
000 — (Undefined, reserved)
001 — FPIAR
010 — FPSR
011 — FPSR, then FPIAR
100 — FPCR
101 — FPCR, then FPIAR
110 — FPCR, then FPSR
111 — FPCR, then FPSR, then FPIAR

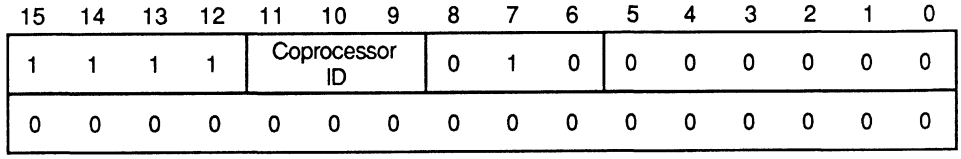
FMUL



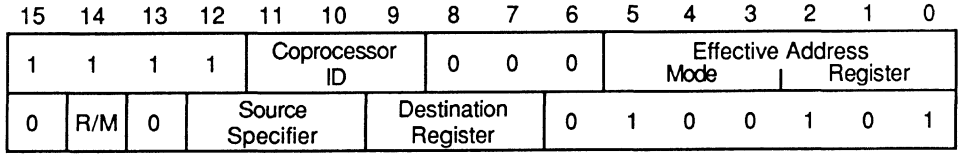
FNEG



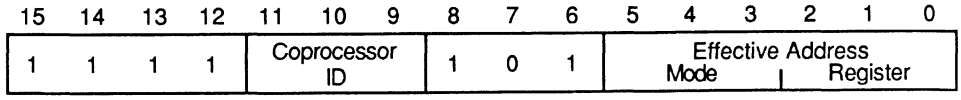
FNOP



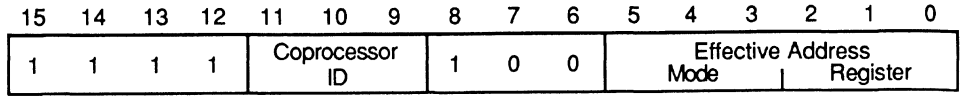
FREM



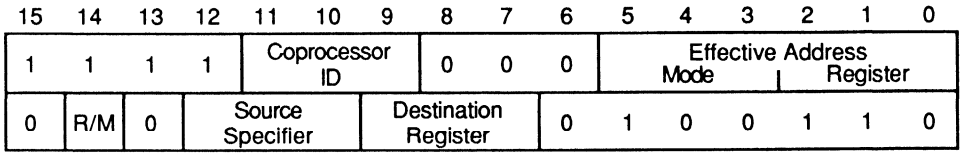
FRESTORE



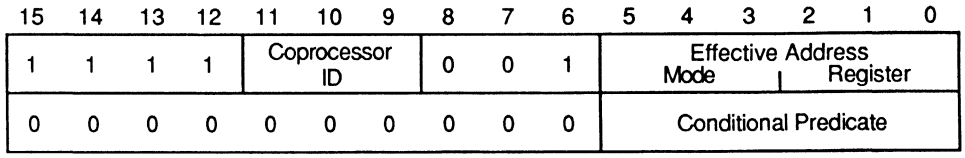
FSAVE



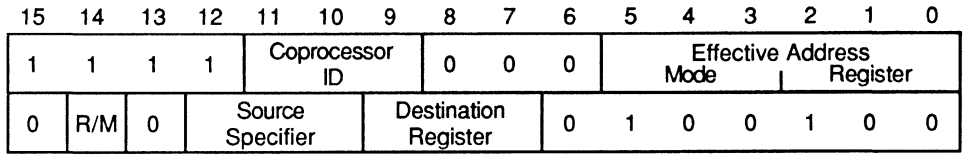
FSCALE



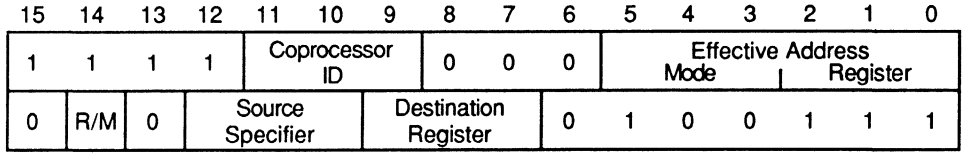
FScc



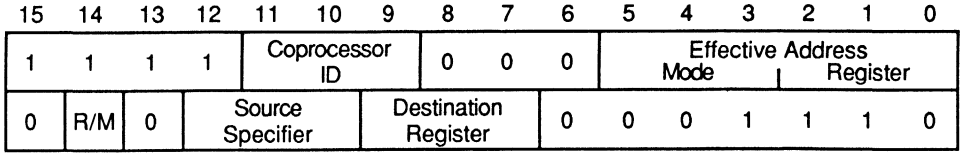
FSGLDIV



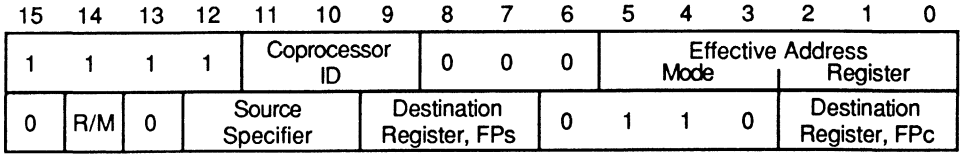
FSGLMUL



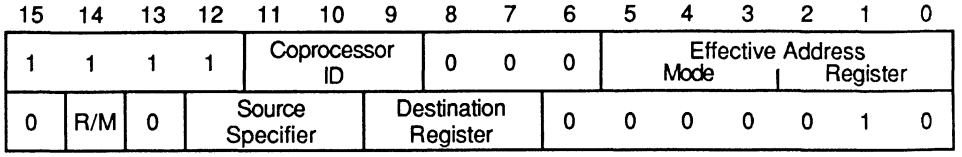
FSIN



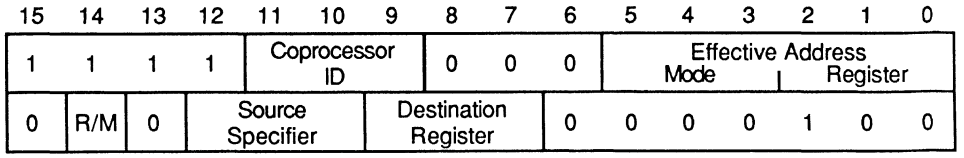
FSINCOS



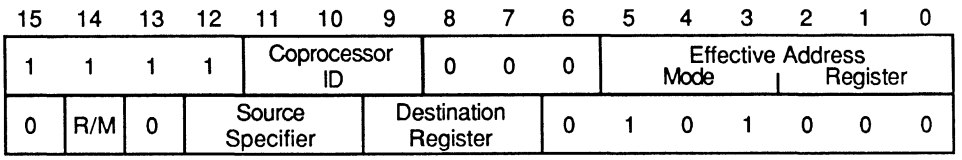
FSINH



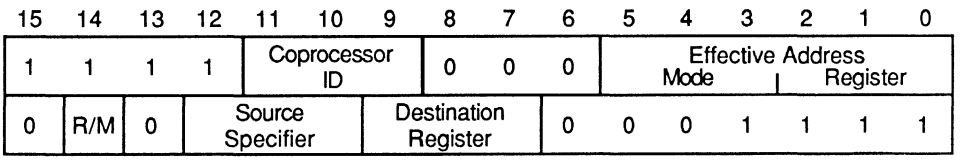
FSQRT



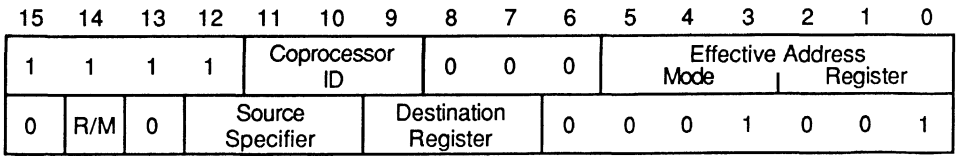
FSUB



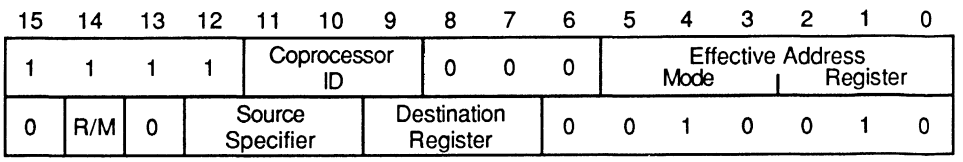
FTAN



FTANH



FTENTOX



FTRAPcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	1	1	1	1	Mode		
0	0	0	0	0	0	0	0	0	0	Conditional Predicate					
16-bit Operand or Most Significant Word of 32-bit Operand (if needed)															
Least Significant Word of 32-bit Operand (if needed)															

Mode Field — Secifies the form of the instruction:
010 — The instruction is followed by a 16-bit operand.
011 — The instruction is followed by a 32-bit operand.
100 — The instruction has no operand following it.

FTST

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register			0	1	1	1	0	1	0

FTWOTOX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Coprocessor ID			0	0	0	Effective Address Mode			Register		
0	R/M	0	Source Specifier			Destination Register			0	0	1	0	0	0	1

SECTION 4
EXCEPTION PROCESSING

This section describes how the MC68881 and the main processor handle exceptional conditions during the processing of floating-point instructions. These exceptional conditions may be detected internally by the MC68881 or the main processor or externally by the main processor.

The MC68020 handles exceptions by viewing any coprocessor in an M68000 system as an extension to the main processor; the fact that a coprocessor is separate from the main processor is transparent to the programmer. Thus, the exception processing for all coprocessors in a system is coordinated by the main processor in a manner that is consistent across all exception types, whether detected during the execution of an instruction native to the main processor or during a coprocessor instruction.

The processing of an exception detected during the execution of an MC68881 instruction involves the following basic steps:

1) Detect the exception.
2) Determine the exception vector number and report the exception to the main processor (if detected by the MC68881).
3) Change processing states if needed (user to supervisor).
4) Save the old context of the main processor (performed automatically by the MC68020).
5) Load a new context from the address contained in the exception vector table.
6) Execute the exception handler.
7) Return to the previous context.

The first two steps involve slightly different operations for exceptions detected by the main processor and those detected by the MC68881, but the manner in which these operations are performed is consistent with non-coprocessor related exceptions. The major difference in the processsing of exceptions detected by the MC68881 and the main processor is the point at which exception processing starts. For all main processor-detected, and some MC68881-detected exceptions, processing for the exception begins during the execution of the coprocessor instruction by the main processor. However, for many of the MC68881-detected exceptions, processing for the exception does not begin until after the main processor completes execution of the offending instruction and attempts execution of a new floating-point instruction. This is because the execution of most MC68881 instructions is concurrent with the execution of non-MC68881 instructions by the main processor. The manner of handling this type exception supports a sequential instruction programming model.

The action of the processor during step 7 above depends upon the type of exception that was previously taken. When the exception handler completes execution, a return from exception (RTE) instruction is executed, and the previously interrupted program resumes execution at:

1) The beginning of the instruction that was preempted by an exception detected by or reported to the MC68020 (pre-instruction exception),
2) The point where the exception occurred during the execution of an instruction (mid-instruction exception), or
3) The beginning of the instruction immediately following the instruction that caused or detected the exception (post-instruction exception).

The following paragraphs describe the causes of various MC68881-related exceptions and how they are handled by the MC68881 and the main processor. Throughout this discussion, the main processor is assumed to be an MC68020; although any other processor can be programmed to simulate the M68000 Family coprocessor interface that is implemented on the MC68020.

4.1 MC68881 DETECTED EXCEPTIONS

MC68881-detected exceptions fall into two categories: those related to communications with the main processor (F-line traps and protocol violations) and those related to the execution of floating-point instructions (computational errors such as divide by zero, or instructions designed to cause a trap such as the FTRAPcc instruction). The protocol for handling each of these exception types is described in detail below.

The main processor coordinates all exception processing. Therefore when the MC68881 detects an exception, it cannot always force exception processing immediately but must wait until the main processor is ready to start exception processing. The main processor is always prepared to process an exception when it attempts to initiate a new MC68881 instruction. Thus, if an MC68881-detected exception occurs during the calculation phase of an instruction, it is held pending within the MC68881 until the next write to the command or condition coprocessor interface register (CIR). Then, instead of returning the first primitive of the dialog for the new instruction, the MC68881 returns the take pre-instruction exception primitive to start exception processing for the previous instruction.

The MC68881 may also report an exception after the output of an operand to memory. In this case, a take mid-instruction exception primitive is issued after the operand is stored in memory (if a conversion error occured). The mid-instruction exception allows the exception handler to more easily determine the address of the exceptional operand, since the MC68020 includes the results of the effective address calculation for the destination operand in the mid-instruction stack frame (the long word at offset +$10).

The third point at which the MC68881 can indicate an exception to the main processor is in response to a protocol violation. If an unexpected access to a coprocessor interface register causes a protocol violation, the MC68881 will immediately encode the response CIR to the take mid-instruction exception primitive using the protocol violation vector number. This allows the protocol violation handler to determine the cause of the violation (either an illegal

primitve from the MC68881 or an illegal access by the MC68020) and perform necessary action. Since an MC68881 protocol violation is a catastrophic error, and the MC68881 cannnot return an illegal primitive, the only appropriate action is to abort the task that detected the protocol violation.

The basic protocol followed in response to an MC68881 detected exception is:

1) The MC68881 encodes the appropriate take exception primitive (pre- or mid-instruction), along with the vector number, in the response CIR.
2) The MC68020 reads the response CIR (usually in an attempt to initiate the next instruction) and receives the take exception request.
3) The MC68020 acknowledges the request by writing an exception acknowledge to the control CIR. The appropriate stack frame is then stored in memory, and control is transferred to the exception handler routine.
4) In response to the exception acknowledge, the MC68881 aborts any internal operation that may be active (this only applies to protocol violations), clears all pending exceptions, and enters the idle state.

The following paragraphs discuss the exception vector assignments used by the MC68881, and each of the exception types that can be detected by the MC68881.

4.1.1 Exception Vectors

The M68000 Family of processors uses a data structure called the exception vector table as a localized dispatching point for all exceptional conditions that may occur in a system. The exception vector table is a 1024-byte structure made up of 256 long word entries. Each entry in the table is a pointer to the routine that should be given control in response to a specific exceptional occurance. When an exception occurs, an index is generated during the automatic processing for that exception that selects one of the vector entries. This index is called the vector number, and is an 8-bit value that is multiplied by four to calculate an offset into the vector table. Of the 256 possible vector numbers, 64 are reserved by Motorola for definition by M68000 Family devices; the remaining 192 are for definition by system designers.

Of the 64 reserved vectors, the MC68020 defines all but 25. The MC68881 utilizes three of the same vector entries defined by the MC68020 and defines seven additional vectors for support of floating-point exceptions. The vectors defined by the MC68881 are shown in Table 4-1. The vector number given is the value (shown in decimal) that is encoded in the appropriate take exception response primitive (except for the FTRAPcc vector number which is generated internally by the MC68020). The vector offset is the location of the corresponding entry in the vector table. The MC68020 adds the vector offset to the value contained in the vector base register to calculate the absolute address of the vector. Refer to the *MC68020 32-Bit Microprocessor User's Manual* for further information on the exception processing operations performed by the MC68020 and the full definition of the exception vector table.

Table 4-1. MC68881 Exception Vector Assignments

Vector Number (Decimal)	Vector Offset (Hexadecimal)	Assignment
7	$01C	FTRAPcc Instruction
11	$02C	F-Line Emulator
13	$034	Coprocessor Protocol Violation
48	$0C0	Branch or Set on Unordered Condition
49	$0C4	Inexact Result
50	$0C8	Floating-Point Divide by Zero
51	$0CC	Underflow
52	$0D0	Operand Error
53	$0D4	Overflow
54	$0D8	Signaling NAN

4.1.2 Instruction Exceptions and Traps

The following paragraphs describe the causes for each exception, what information is available to the trap handler, and what results are generated if traps are enabled or disabled. MC68881 instruction exceptions arise from the detection of abnormal conditions during coprocessor instruction execution. All MC68881 detected instruction exceptions are enabled or disabled via the FPCR ENABLE byte.

There are eight exceptions that can be generated by the execution of a floating-point instruction. The location of the exception bits in the EXC and ENABLE bytes are shown in Figure 4-1. If more than one enabled exception occurs on the same instruction, then the highest priority instruction trap is taken (BSUN is the highest; INEX2/INEX1 is the lowest). When multiple exceptions occur, the MC68881 traps to the highest priority exception that is enabled, and the lower priority exception can not cause a second trap. It is the programmer's responsibility to determine if any of the exception bits that have lower priority than the exception taken are set.

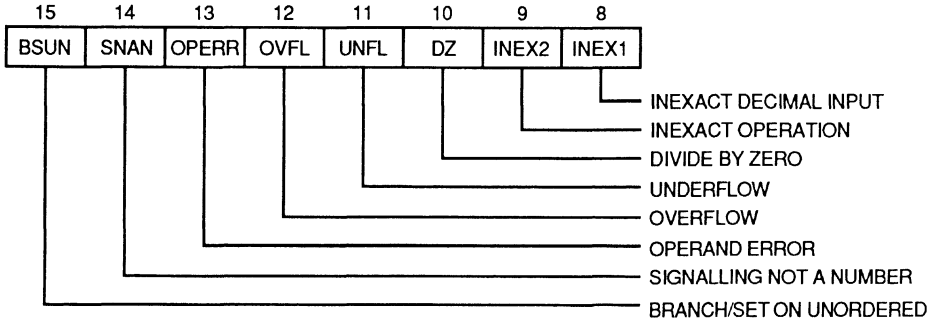


Figure 4-1. EXC and ENABLE Byte Bit Assignments

MC68881 instruction exceptions which arise from the move floating-point data register to external destination instructions are reported to the MC68020 as mid-instruction exceptions. All other MC68881 detected instruction exceptions are reported as pre-instruction exceptions. The MC68881 move multiple, move system control register, and FSAVE instructions can not generate coprocessor detected instruction exceptions. The FRESTORE instruction can generate coprocessor detected instruction exceptions only when the state frame format written to the coprocessor is unrecognized.

In the following exception descriptions, the term "intermediate result" is used frequently. During the execution of a floating-point algorithm, the MC68881 execution control unit (ECU) contains a 67-bit mantissa (for rounding purposes) and a 17-bit exponent (to ensure that overflow or underflow can never occur during the main algorithm). At the end of the algorithm, this intermediate result must then be stored into a floating-point data register, an MC68020 data register, or into memory. It is this intermediate result that is checked for underflow, rounded, and checked for overflow to obtain the final result.

4.1.2.1 BRANCH/SET ON UNORDERED (BSUN). The BSUN exception may occur only on MC68881 conditional instructions with the following IEEE non-aware branch condition predicates:

GT	Greater Than	GL	Greater or Less Than
NGT	Not Greater Than	NGL	Not Greater or Less Than
GE	Greater Than or Equal	GLE	Greater or Less or Equal
NGE	Not Greater Than or Equal	NGLE	Not Greater or Less or Equal
LT	Less Than	SF	Signaling False
NLT	Not Less Than	ST	Signaling True
LE	Less Than or Equal	SEQ	Signaling Equal
NLE	Not Less than or Equal	SNE	Signaling Not Equal

During the MC68881 conditional instructions, the MC68020 writes the conditional predicate to the condition CIR and reads the response CIR. If an instruction exception is pending from a previously executed MC68881 instruction, the exception is reported as a pre-instruction exception. Following exception processing of the pending instruction exception, the MC68020 restarts the MC68881 conditional instruction. If no exception is pending, the MC68881 will evaluate the conditional test and report the result to the MC68020.

The MC68881 detects a BSUN exception if the conditional predicate is one of the IEEE non-aware branches, and the NAN condition code bit is set. When this exception is detected, the BSUN bit in the FPSR exception status byte is set.

Trap Disabled Results: The MC68881 evaluates the condition and reports "true" or "false" to the MC68020 in the response CIR.

Trap Enabled Results: The MC68881 reports a pre-instruction exception with the BSUN vector number to the MC68020 in lieu of a "true" or "false" report.

The BSUN exception is unique in that the trap is taken before the requested conditional predicate is evaluated. Furthermore, the instruction that caused the BSUN exception is re-executed upon return from the BSUN trap handler. Therefore, it is the responsibility of the

trap handler to modify the floating-point condition codes so that when the trap handler returns, the conditional instruction does not continue to take the BSUN trap. The trap handler, by modifying the condition codes, determines whether the "true" or "false" report is indicated to the MC68020 when the conditional instruction is re-executed. This allows the trap handler to determine how the unordered condition is to be handled.

4.1.2.2 SIGNALING NOT-A-NUMBER. SNANs are used as escape mechanisms for user defined, non-IEEE data types. The MC68881 never creates a SNAN as a result of an operation; NANs created by operand error exceptions are always non-signaling NANs.

When a SNAN is an operand involved in an arithmetic instruction, the SNAN bit is set in the FPSR exception byte. Since the FMOVEM, FMOVE FPcr, and FSAVE instructions do not modify the status bits, they cannot generate exceptions. Therefore, these instructions are useful for manipulating SNANs.

Trap Disabled Results: If the destination data format is S, D, X, or P, then the SNAN bit in the NAN is set to one and the resulting non-signaling NAN is transferred to the destination. No bits other than the SNAN bit of the NAN are modified, although the input NAN is truncated if necessary. If the destination data format is B, W, or L, then the 8, 16, or 32 most significant bits of the SNAN significand, with the SNAN bit set, are written to the destination.

Trap Enabled Results: For memory or MC68020 data register destinations, the result is written in the same manner as if the traps were disabled, and then a mid-instruction except - ion is signaled. If desired, the trap handler can overwrite the result.

For floating-point data register destinations, instruction execution is terminated and the floating-point data registers are not modified. In this case, the SNAN trap handler should supply the result. To enable the trap handler to return a result, the MC68020 and the MC68881 supply:

1) The address of the instruction where the error occurred (in the FPIAR). By examining the instruction, the trap handler may determine the operation being performed, the value of the second operand (for dyadic instructions), and the destination location.
2) The address of the destination, if in memory, in the mid-instruction stack frame (at offset +$10). This allows the trap handler to overwrite the NAN, if necessary, without recalculating the effective address.
3) The FSAVE instruction that places the exceptional operand in a stack frame. For additional FSAVE stack frame information, refer to **4.3.2 State Frames**. The exceptional operand is the source input argument converted to extended precision.

Note that the trap handler should only use the FMOVEM instruction to read or write the floating-point data registers, since FMOVEM can not generate further exceptions. Also, the only way that a SNAN can be written into a floating-point data register is via the FMOVEM instruction.

4.1.2.3 OPERAND ERROR. Operand errors encompass problems arising in a variety of operations, and cover those errors not frequent or important enough to merit a specific exception condition. Basically, an operand error occurs when an operation has no mathematical interpretation for the given operands. The possible operand errors are listed in Table 4-2. When an operand error occurs, the OPERR bit is set in the FPSR exception status byte.

Trap Disabled Results: For a memory or MC68020 data register destination, several possible results can be generated, depending on the destination size and error type. (An operand error is never generated when the destination is an MC68020 data register or memory if the destination format is S, D, or X.)

Table 4-2. Possible Operand Errors

Instruction	Condition Causing Operand Error
FACOS	Source is ±infinity, > +1, or < −1
FADD	(+infinity) + (−infinity) or (−infinity) + (+infinity)
FASIN	Source is ±infinity, > +1, or < −1
FATANH	Source is ≥ +1, or ≤ −1
FCOS	Source is ±infinity
FDIV	0/0 or infinity/infinity
FGETEXP	Source is ±infinity
FGETMAN	Source is ±infinity
FLOG10	Source is < 0
FLOG2	Source is < 0
FLOGN	Source is < 0
FLOGNP1	Source is ≤ −1
FMOD	Floating-Point Data Register is ±infinity or Source is 0, other Operand is Not a NAN
FMOVE to B, W, or L	Integer Overflow/Underflow, Source is NAN, or Source is ±infinity
FMOVE to P	Source Exponent > 999 (Decimal) or k-Factor > +17
FMUL	One Operand is 0, Other Operand is ±infinity
FREM	Floating-Point Data Register is ±infinity or Source is 0, Other Operand is Not a NAN
FSCALE	Source is ±infinity
FSGLDIV	0/0 or Infinity/Infinity
FSGLMUL	One Operand is 0, Other Operand is Infinity
FSIN	Source is ±infinity
FSINCOS	Source is ±infinity
FSQRT	Source < 0
FSUB	Source and Floating-Point Data Register are +infinity or Source and FPn are −infinity
FTAN	Source is ±infinity

If the operand error is caused by an integer overflow, or if the floating-point data register being stored is infinity, the result is the largest positive or negative integer that can fit in the specified destination format size. If the destination is B, W, or L and the floating-point number being stored is a NAN, then the 8, 16, or 32 most significant bits of the NAN significand are stored as the result.

For packed decimal results, if the k factor is greater than +17, the result returned is a packed decimal string that assumes a k factor equal to +17. For packed decimal results where the absolute value of the exponent is greater than 999, the decimal string is returned with the three least significant exponent digits in EXP2, EXP1, and EXP0. The fourth digit, EXP3, is supplied in the most significant four bits of the third byte in the string. Refer to **2.8 DATA FORMAT DETAILS** for the packed decimal string format.

If traps are disabled and the destination is a floating-point data register, then a extended precision non-signaling NAN is stored in the destination floating-point data register.

Trap Enabled Results: For memory or MC68020 data register destinations, the destination operand is written as if the trap were disabled, and then a take exception primitive is returned to the MC68020. This can only occur for the FMOVE FPm,<ea> instruction, and the exception is reported as a mid-instruciton exception. If desired, the trap handler can overwrite the result generated by the MC68881.

If the destination is a floating-point data register, then the register is not modified by the MC68881. In this case, the trap handler should generate the appropriate result. To enable the trap handler to return a result, the MC68020 and MC68881 supply:

1) The address of the instruction where the error occured (in the FPIAR). By examining the instruction, the trap handler may determine the operation being performed, the value of the second operand (for dyadic instructions), and the destination location.
2) The address of the destination, if in memory, in the mid-instructions stack frame (at offset +$10). This allows the trap handler to overwritethe NAN, if necessary, without recalculating the effective address.
3) The FSAVE instruction that places the exceptional operand in a stack frame. For additional FSAVE stack frame information, refer to **4.3.2 State Frames**. The exceptional operand is the source input argument converted to extended precision.

Note that the trap handler should only use the FMOVEM instruction to read or write to the floating-point data registers, since FMOVEM will not generate further exceptions or change the condition codes.

4.1.2.4 OVERFLOW. Overflow is the condition that exists when an arithmetic operation creates a floating-point intermediate result that is too large to be represented in a floating-point data register or, in a store to memory, when the contents of the source floating-point data register are too large to be represented in the destination format.

Overflow is detected for a given data format and operation when the result exponent is greater than or equal to the maximum exponent value of the format. Overflow can only occur when the destination is in the S, D, or X formats. Overflows when converting to the B, W, or L

integer and packed decimal formats are included as operand errors. Refer to **2.8 DATA FORMAT DETAILS** for the maximum exponent value for each format. At the end of any operation that could potentially overflow and before the result is stored to the destination, the intermediate result is checked for underflow, then rounded, and then checked for overflow. If overflow occurs, then the OVFL bit is set in the FPSR exception byte.

NOTE

An overflow can occur when the destination is a floating-point data register even if the intermediate result is small enough to be represented as an extended precision number. This is due to the fact that the selected rounding precision is single or double, the intermediate result is rounded to that precision (both the mantissa and the exponent) and then the rounded result is stored in extended precision format. If the magnitude of the intermediate result exceeds the range of the selected rounding precision format, an overflow will occur.

Trap Disabled Results: The following values are stored at the destination, based on the current rounding mode:

Rounding Mode	Result
RN	Infinity, with the sign of the intermediate result
RZ	Largest magnitude number, with the sign of the intermediate result
RM	For positive overflow, largest positive number For negative overflow, –infinity
RP	For positive overflow, +infinity For negative overflow, largest negative number

Trap Enabled Results: The result stored in the destination is the same as the result stored when the trap is disabled, and a take exception primitive is returned to the MC68020. If the destination is memory or an MC68020 data register, the operand is stored, and then a take mid-instruction exception primitive is issued. If the destination is a floating-point data register, a take pre-instruction exception primitive is returned when the MC68020 attempts to initiate the next MC68881 instruction.

The address of the instruction that causes the overflow is available to the trap handler in the FPIAR. By examining the instruction, the exception type and operand location(s) may be determined. Additional information is available to the trap handler by executing the FSAVE instruction. When FSAVE is executed, the exceptional operand is stored in the stack frame. Refer to **4.3.2 State Frames** for details of the stack frame generated by FSAVE. The exceptional operand is defined differently for various destination types:

1) Memory or MC68020 data register destination—the value in the exceptional operand is the intermediate result mantissa rounded to the destination precision, with a 15-bit exponent biased as a normal extended precision number. In the case of a memory destination, the evaluated effective address of the operand is available in the MC68020 mid-instrucion stack frame (at offset +$10). This allows the trap handler to overwrite the default result, if necessary, without recalculating the effective address.
2) Floating-point data register destination—the value in the exceptional operand is the intermediate result rounded to extended precision, with an exponent bias of $3FFF-

$6000 rather than $3FFF. The additional bias of –$6000 is used to "wrap" the 17-bit intermediate value into a value that can be represented in 15 bits. To recover the 17-bit twos complement exponent of the intermediate result, the 15-bit exponent of the exceptional operand should be sign extended to at least 17 bits (i.e., if it is manipulated in an MC68020 data register, it is sign extended to a long word value) and then the bias of $3FFF-$6000 should be subtracted from that number. Note that for most operations, the intermediate exponent value will not exceed 32,767 and thus can be contained in a 16-bit integer. However, a completely general exception handler should calculate a 17-bit exponent value.

In addition to normal overflow, the exponential instructions implemented by the MC68881 (e^x, 10^x 2^x, hyperbolic sine and cosine) may generate results that overflow the 17-bit exponent used for intermediate results. For example, the e^x function can easily overflow the 17-bit intermediate exponent if the source value is very large ($x \geq +8192.0$). When such an overflow occurs (called a catastrophic overflow), the exceptional operand exponent value is set to $0000. This value is easily distinguished from the exponent values produced by normal overflow processing. The smallest exponent value that can be produced by a normal overflow is $1FFF ($04000 + $3FFF - $6000, truncated to 15 bits), while the largest exponent value is $7FFF ($0A000 + $3FFF - $6000, truncated to 15 bits). The catastrophic overflow exceptional operand exponent value of $0000 is produced any time that a calculation generates an intermediate result exponent value greater than $0A000.

Note that the trap handler should only use the FMOVEM instructions to read or write to the floating-point data registers since FMOVEM will not generate further exceptions or change the condition codes.

4.1.2.5 UNDERFLOW. Underflow is the condition that occurs when an arithmetic operation creates an intermediate result that is too small to be represented in a floating-point data register using the selected rounding precision or, in a store to memory, when the contents of the source floating-point data register are too small to be represented in the destination format as a normalized result. Underflow is detected for a given data format and operation when the intermediate result exponent is less than or equal to the minimum exponent value of the destination format. Underflow can only occur when the destination format is S, D, or X. When the destination format is packed decimal, underflows are included as operand errors. When the destination format is B, W, or L, the conversion underflows to zero without causing either an underflow or an operand error. See **2.8 DATA FORMAT DETAILS** for the minimum exponent value for each format.

At the end of any operation that could potentially underflow, the intermediate result is checked for underflow, rounded, and checked for overflow before it is stored to the destination. If an underflow occurs, then the UNFL bit is set in the FPSR exception status byte.

NOTE

An underflow can occur when the destination is a floating-point data register even if the intermediate result is large enough to be represented as an

extended precision number. This is due to the fact that if the selected rounding precision is single or double, the intermediate result is rounded to that precision (both the mantissa and the exponent) and then the rounded result is stored in extended precision format. If the magnitude of the intermediate result is too small to be represented in the selected rounding precision format, an underflow will occur.

Trap Disabled Results: The result that is stored in the destination is either a denormalized number or zero. Denormalization is accomplished by taking the intermediate result (which is always normalized, due to the 17-bit exponent used in the MC68881 ALU and temporary registers) and shifting the mantissa to the right while incrementing the exponent until it is equal to the denormalized exponent value for the destination format. After denormalization, the result is rounded to the destination precision.

If, in the process of denormalizing the intermediate result, all of the significant bits are shifted off to the right, then the following values are stored at the destination, based on the current rounding mode:

Rounding Mode	Result
RN	Zero, with the sign of the intermediate result
RZ	Zero, with the sign of the intermediate result
RM	For positive underflow, +zero
	For negative underflow, smallest denormalized negative number
RP	For positive underflow, smallest denormalized positive number
	For negative overflow, -zero

Trap Enabled Results: The result stored in the destination is the same as the result stored when traps are disabled, and a take exception primitive is returned to the MC68020. If the destination is memory or an MC68020 data register, the operand is stored and then a take mid-instruction exception primitive is issued. If the destination is a floating-point data register, a take pre-instruction exception primitive is returned when the MC68020 attempts to initiate the next MC68881 instruction.

The address of the instruction that caused the underflow is available to the trap handler in the FPIAR. By examining the instruction, the operation type and operand location(s) may be determined. Additional information is available to the trap handler by executing an FSAVE instruction. When an FSAVE instruction is executed, the exceptional operand is stored in the stack frame. Refer to **4.3.2 State Frames** for details of the stack frame generated by FSAVE. The exceptional operand is defined differently for various destination types:

1) Memory or MC68020 data register destination—the value in the exceptional operand is the intermediate result mantissa rounded to the destination precision, with a 15-bit exponent biased as a normal extended precision number. In the case of a memory destination, the evaluated effective address of the operand is available in the MC68020 mid-instruction stack frame (at offset +$10). This allows the trap handler to overwrite the default, if necessary, without recalculating the effective address.

2) Floating-point data register destination—the value in the exceptional operand is the intermediate result mantissa rounded to extended precision, with an exponent bias of $3FFF+$6000 rather than $3FFF. The additional bias of +$6000 is in 15 bits. To recover the 17-bit twos complement exponent of the intermediate result, the 15-bit exponent of the exceptional operand is sign extended to at least 17 bits (i.e., if it is manipulated in an MC68020 data register, it is sign extended to a long word value) and then the bias of $3FFF+$6000 is subtracted from that number. Note that for most operations, the intermediate exponent value will not be less than –32768, and thus can be contained in a 16-bit integer, but a completely general exception handler should calculate a 17-bit exponent value.

   In addition to normal underflow, the exponential instructions implemented by the MC68881 (e^x, 10^x, 2^x, hyperbolic sine and cosine) may generate results that underflow the 17-bit exponent used for intermediate results. For example, the e^x function can easily underflow the 17-bit intermediate exponent if the source value is a large negative number ($x \leq -8192.0$). When such an underflow occurs (called a catastrophic underflow), the exceptional operand exponent value is set to $0000. This value is equal to the smallest exponent value that can be produced by a normal underflow ($16001 + $3FFF + $6000, truncated to 15 bits), while the largest exponent value is $5FFF ($1C000 + $3FFF + $6000, truncated to 15 bits). The catastrophic underflow exceptional operand exponent value of $0000 is produced any time that a calculation generates an intermediate result exponent value less than or equal to $16001.

Note that the trap handler should only use the FMOVEM instructions to read or write to the floating-point data registers since FMOVEM can not generate further exceptions or change the condition codes.

NOTE

The IEEE standard defines two causes of an underflow: 1) when a result is so tiny that the absolute value of the number is less than the minimum number that can be represented by a normalized number in a specific format, and 2) when loss of accuracy occurs when attempting to calculate a very small number (a loss of accuracy also causes an inexact exception). The IEEE standard specifies that if the underflow trap is disabled, then an underflow should only be signaled when both of these cases are satisfied (i.e., the result is too small to represent with a given format, and there is a loss of accuracy during the calculation of the final result). If the trap is enabled, the underflow should be signaled any time a tiny result is produced, regardless of whether accuracy is lost in calculating it.

The MC68881 UNFL bit in the AEXC byte reflects the IEEE trap disabled definition, since it is only set when a tiny number is generated and accuracy has been lost when calculating that number. The UNFL bit in the EXC byte reflects the IEEE trap enabled definition, since it is set anytime a tiny number is generated. Thus, if the MC68881 underflow trap is enabled, a trap occurs when tininess alone is detected (as the IEEE standard specifies) to support the emulation of machines that underflow to zero, rather than using the IEEE

gradual underflow method (i.e., denormalized numbers). If the underflow trap is disabled, the UNFL bit in the AEXC byte may be polled at the end of a calculation to determine if any result produced during the operation required representation as a denormalized number, and accuracy was lost when denormalizing and rounding that result.

4.1.2.6 DIVIDE-BY-ZERO. This exception occurs when a zero divisor occurs in a division, or when a transcendental function is asymptotic with infinity as the asymptote. Table 4-3 lists the instructions that can generate the divide- by-zero exception. When a divide-by-zero is detected, the DZ bit is set in the FPSR exception status byte.

Table 4-3. Divide-by-Zero Exception Instructions

Instruction	Operand Value
FDIV	Source Operand = 0 and Floating-Point Data Register is Not a NAN
FLOG10	Source Operand = 0
FLOG2	Source Operand = 0
FLOGN	Source Operand = 0
FTAN	Source Operand is an Odd Multiple of $\pm\pi/2$
FSGLDIV	Source Operand = 0 and Floating-Point Data Register is Not a NAN

Trap Disabled Results: Store the following results in the destination floating-point data register: 1) for the FDIV and FSGLDIV instructions, return an infinity with the sign set to the exclusive OR of the signs of the input operands; 2) for the FTAN instruction, return infinity with the sign of the source operand; 3) for the FLOGx instructions, return minus infinity.

Trap Enabled Results: The destination floating-point data register is not modified, and a take pre-instruction exception primitive is returned when the MC68020 attempts to initiate the next MC68881 instruction. The trap handler must generate a result to store in the destination. To assist the trap handler in this function, the MC68881 supplies:

1) The address of the instruction where the divide-by-zero occurred (in the FPIAR). By examining this instruction, the trap handler can determine the operation being performed, the value of the source operand (for dyadic instructions), and the destination floating-point register number.
2) The FSAVE instruction that places the exceptional operand in a stack frame. For additional FSAVE stack frame information, refer to **4.3.2 State Frames**. The exceptional operand is the source input argument converted to extended precision.

Note that the trap handler should only use the FMOVEM instruction to read or write the floating-point data registers, since FMOVEM can not generate further exceptions or change the condition codes.

4.1.2.7 INEXACT RESULT. In a general sense, inexact result 2 (INEX2) is the condition that exists when any operation, except the input of a packed decimal number, creates a floating-point intermediate result whose infinitely precise mantissa has too many significant bits to be represented exactly in the current rounding precision or in the destination precision. If this condition occurs, the INEX2 bit is set in the status register EXC byte and the infinitely precise result is rounded as described below.

The MC68881 provides two inexact bits (INEX1 and INEX2) to help distinguish between inexact results generated by decimal input (INEX1) and other inexacts (INEX2). This is useful in instructions like:

```
FADD.P  #6.023E+24,FP3
```

where both types of inexacts can occur. In this case, the packed decimal to extended precision conversion of the immediate source operand causes an inexact error to occur which is signaled as INEX1. Furthermore, the subsequent add might also produce an inexact result and may cause INEX2 to be set. Therefore, the MC68881 provides two inexact bits in the FPSR exception status byte to distinguish these two cases.

Note that there is only one inexact exception vector number generated by the MC68881. If either of the two inexact exceptions is enabled, then the inexact exception vector is fetched by the MC68020, and the exception handler routine is initiated.

The IEEE standard specifies, and the MC68881 supports, four rounding modes. These modes are round to nearest (RN), round toward zero (RZ), round toward plus infinity (RP), and round toward minus infinity (RM). The rounding definitions are:

- RN — The representable value nearest to the infinitely precise intermediate value is the result. If the two nearest representable values are equally near (a tie), then the one with the least significant bit equal to zero (even) is the result. This is sometimes referred to as "round nearest, even".
- RZ — The result is the value closest to, and no greater in magnitude than, the infinitely precise intermediate result. This is sometimes referred to as the "chop mode", since the effect is to clear the bits to the right of the rounding point.
- RM — The result is the value closest to and no greater than the infinitely precise intermediate result (possibly minus infinity).
- RP — The result is the value closest to and no less than the infinitely precise intermediate result (possibly plus infinity). The RM and RP rounding modes are often referred to as "directed rounding modes" and are useful in interval arithmetic.

Rounding is accomplished using the intermediate result format shown in Figure 4-2.

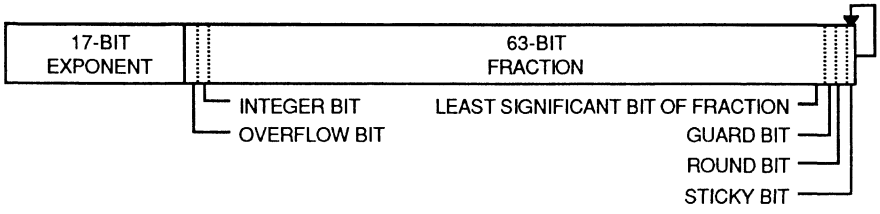


Figure 4-2. Intermediate Result Format

Depending on the rounding precision in effect, the location of the least significant bit of the fraction and the guard, round, and sticky bits in the 67-bit intermediate result mantissa varies.

The guard and round bits are always calculated exactly. The sticky bit is used to create the illusion of an infinitely wide intermediate result mantissa. As shown by the arrow in the figure above, the sticky bit is the logical OR of all the bits in the infinitely precise result to the right of the round bit. During the calculation stage of an arithmetic operation, any non-zero bits generated that are to the right of the round bit are stored as a one in the sticky bit (which is used in rounding). Because of the sticky bit, the rounded intermediate result for all required IEEE arithmetic operations in the round-to-nearest mode will be in error by no more than one half unit in the last place. For transcendental instructions, the result may not be this accurate (see **3.2 COMPUTATIONAL ACCURANCY**).

NOTE

When the MC68881 is programmed to operate in the single or double precision rounding mode, a method referred to as "range control" is used to assure correct emulation of a machine that only supports single or double precision arithmetic. When the MC68881 performs any calculation, the intermediate result is in the format shown above, and a rounded result stored into a floating-point data register is always in the extended precision format. However, if the single or double precision rounding mode is in effect, the final result generated by the MC68881 is always within the range of the selected precision format.

Range control is accomplished by not only rounding the intermediate result mantissa to the specified precision, but also checking the 17-bit intermediate exponent to assure that it is within the representable range of the selected rounding precision format. If the intermediate exponent exceeds the range of the selected precision, the exponent value appropriate for an underflow or overflow is stored as the resutt *in the 15-bit extended precision exponent.* For example, if the rounding precision and mode is single/RM and the result of an arithmetic operation overflows the magnitude of the single precision format, the largest normalized single precision value is stored as an extended precision number in the destination floating-point data register (i.e., an exponent of $00FE and a mantissa of $FFFFFF0000000000). If an infinity is the appropriate result for an underflow or overflow, the infinity value for the destination data type is stored as the result (i.e., an exponent with the maximum type and a mantissa of zero).

Figure 4-3 shows the algorithm that is used to round an intermediate result to the destination precision. If the rounded result of an operation is not exact and the MC68881 is not performing a decimal input operation, then the INEX2 bit is set in the FPSR EXC byte. For inexact conversions of decimal inputs, the INEX1 bit is set.

```
BEGIN
    IF GUARD, ROUND AND STICKY = 0
        THEN (RESULT IS EXACT)
            DON'T SET INEX1 OR INEX2
            DON'T CHANGE THE INTERMEDIATE RESULT

        ELSE (RESULT IN INEXACT)
            SET INEX1 OR INEX2 IN THE FPSR EXC BYTE

            SELECT THE ROUNDING MODE
                RM: IF INTERMEDIATE RESULT IS POSITIVE THEN ADD 1 TO LSB
                RN: IF GUARD = 1 AND ROUND AND STICKY = 0 (TIE CASE)
                        THEN IF LSB = 1 ADD 1 LSB
                        ELSE ADD 1 TO LSB
                    END IF
                RP: IF INTERMEDIATE RESULT IS NEGATIVE THEN ADD 1 TO LSB
                RZ: (FALL THROUGH; GUARD, ROUND AND STICKY ARE CHOPPED)
            END SELECT

            IF OVERFLOW = 1
                THEN
                    SHIFT MANTISSA RIGHT BY ONE BIT
                    ADD 1 TO THE EXPONENT
                END IF

        SET GUARD, ROUND AND STICKY TO 0
    END IF
END
```

Figure 4-3. Rounding Algorithm

Trap Disabled Results: The rounded result is delivered to the destination.

Trap Enabled Results: The rounded result is delivered to the destination, and an exception is reported to the MC68020. If the destination is memory or an MC68020 data register, a take mid-instruction exception primitive is returned immediately after the operand is stored. If the destination is a floating-point data register, a take pre-instruction exception primitive is returned when the MC68020 attempts to initiate the next MC68881 instruction.

The address of the instruction that generated the inexact result is available to the trap handler in the FPIAR. By examining the instruction, the location of the operand(s) may be determined. In the case of a memory destination, the evaluated effective address of the operand is available in the MC68020 mid-instruction stack frame. Unlike the other exceptions, when an FSAVE is executed by an inexact trap handler, the value of the exceptional operand in the stack frame is not defined. If an inexact condition is the only exception that occurred during the execution of an instruction, the value of the exceptional operand is invalid. If multiple exceptions occur during an instruction, the exceptional operand value is related to the other, higher priority exception.

Note that the trap handler should only use the FMOVEM instruction to read or write the floating-point data registers, since FMOVEM can not generate further exceptions or change the condition codes.

NOTE

The IEEE standard specifies that inexactness should be signaled on overflow as well as for rounding as described above. The MC68881 implements this via the INEX bit in the FPSR AEXC byte. However, the standard also indicates that the inexact trap should be taken if an overflow occurs with the overflow trap disabled and the inexact trap enabled. Therefore, the MC68881 takes the inexact trap if this combination of conditions occurs, even though the INEX1 or INEX2 bits may not be set in the FPSR EXC byte. In this case, INEX is set in the AEXC byte and OVFL is set in both the EXC and AEXC bytes.

4.1.2.8 INEXACT RESULT ON DECIMAL INPUT. In a general sense, inexact result 1 (INEX1) is the condition that exists when a packed decimal operand can not be converted exactly to extended precision in the current rounding mode. If this condition occurs, the INEX1 bit is set in the FPSR exception status byte, and the infinitely precise result is rounded as previously described. The MC68881 provides two inexact bits (INEX1 and INEX2) to help distinguish between inexact results generated by decimal input (INEX1) and other inexact results (INEX2).

Trap Disabled Results: If the instruction is an FMOVE to a floating-point data register, then the rounded result is stored in the floating-point data register. If the instruction is not an FMOVE, then the rounded result is used in the calculation.

Trap Enabled Results: The result is generated in the same manner as if traps were disabled, except that a take pre-instruction exception primitive is returned to the MC68020 when it attempts to initiate the next MC68881 instruction.

The address of the instruction that caused the inexact decimal conversion is available to the trap handler in the FPIAR. By examining the instruction, the location of the decimal string may be determined, although the effective address of the string must be recalculated (if possible) by the trap handler. Unlike the other exceptions, when an FSAVE is executed by an inexact trap handler, the value of the exceptional operand in the stack frame is not defined. If the inexact conversion is the only exception that occurs during the execution of an instruction, the value of the exceptional operand is invalid. If multiple exceptions occur during an instruction, the exceptional operand value is related to the other, higher priority exception.

Note that the trap handler should only use the FMOVEM instruction to read or write the floating-point data registers, since FMOVEM can not generate further exceptions or change the condition codes.

4.1.2.9 MULTIPLE EXCEPTIONS. Dual and triple instruction exceptions may be generated by a single instruction in a few cases. When multiple exceptions occur with traps

enabled for more than one exception class, only the highest priority exception trap is taken; the other enabled exceptions can not cause a trap. The higher priority trap handler must check for multiple exceptions. The priority of the traps are as follows:

BSUN ←——— Highest Priority
SNAN
OPERR
OVFL
UNFL
DZ
INEX2/INEX1 ←——— Lowest Priority

Below is a list of the multiple instruction exceptions that can occur:

SNAN and INEX1
OPERR and INEX2
OPERR and INEX1
OVFL and INEX2 and/or INEX1
UNFL and INEX2 and/or INEX1

4.1.2.10 IEEE EXCEPTION AND TRAP COMPATIBILITY. The IEEE standard only defines five exceptions. The MC68881 FPSR AEXC byte contains bits representing these five exceptions, which are defined to function exactly as the standard specifies the exceptions. However, it may be more useful to differentiate the IEEE required exceptions into the eight exceptions represented in the FPSR EXC byte. Since the MC68881 uses the bits in the FPSR EXC byte and the FPCR ENABLE byte to determine when to trap, there are seven possible instruction traps defined (INEX1 and INEX2 share a trap vector) instead of the five defined by the standard.

If it is necessary to write an application program that only supports the five IEEE specified traps, the BSUN, SNAN, and OPERR trap vectors should be set to point to the same trap handler. This allows the MC68881 to support the invalid operation exception defined in the IEEE standard, which is represented by the invalid operation (IOP) bit in the AEXC byte.

To satisfy other requirements in the IEEE standard, the MC68881 does the following:

1) A one is ORed into the AEXC byte IOP bit if the BSUN, SNAN, or OPERR bit is set in the EXC byte.
2) A one is ORed into the AEXC byte underflow (UNFL) bit only if both the UNFL and the INEX2 bits of the EXC byte are set. However, per the IEEE standard, the underflow trap is based only on the UNFL bit in the EXC byte.
3) A one is ORed into the AEXC byte inexact (INEX) bit if the INEX1, INEX2 or OVFL bit is set in the EXC byte.
4) The IEEE standard requires that an inexact trap be taken if it is enabled, an overflow occurs, and the overflow trap is disabled. Thus, if the overflow (OVFL) bit is set in the EXC byte, the OVFL bit is not set in the FPCR ENABLE byte, and the INEX2 bit is set in the FPCR ENABLE byte, then the inexact trap is taken.

The equations for items 1, 2, and 3 are:

AEXC(IOP) = AEXC(IOP) ∨ EXC(BSUN ∨ SNAN ∨ OPERR)
AEXC(UNFL) = AEXC(UNFL) ∨ EXC(UNFL ∧ INEX2)
AEXC(INX) = AEXC(INEX) ∨ EXC(INEX1 ∨ INEX2 ∨ OVFL)

where:

"∨" = logical OR
"∧" = logical AND

The equation for item 4 (inexact trap taken) is:

Inexact Trap =
[[EXC(OVFL) ∨ EXC(INEX2)] ∧ ENABLE(INEX2)] ∨ [EXC(INEX1) ∧ ENABLE(INEX1)]

4.1.3 Illegal Command Words

Illegal coprocessor commands are coprocessor command word bit patterns that are not implemented by the MC68881. The MC68881 reports illegal coprocessor commands as pre-instruction exceptions, using the F-line emulator vector number. The specific illegal command word bit patterns are defined in **3.5 INSTRUCTION ENCODING DETAILS.**

MC68881 instructions consist of an operation word, a coprocessor command word (if any), and extension words (if any). The MC68881 detects illegal command words, while the MC68020 detects illegal operation words.

For the case where a coprocessor detected instruction trap is pending when the MC68020 writes an illegal coprocessor command to the MC68881 command CIR, the following action is taken. First, the pending instruction exception is reported as a pre-instruction exception. Following exception processing of the instruction exception, the MC68020 resumes execution of the main program at the beginning of the illegal coprocessor command, by writing to the command CIR again. The illegal instruction exception is then reported by the MC68881.

4.1.4 MC68881 Detected Protocol Violation

All interprocessor communications in the coprocessor interface occur as standard M68000 bus cycles. A failure in this communication results in the MC68881 reporting a mid-instruction exception with the coprocessor protocol violation vector number. Once a protocol violation has been detected by the MC68881, the response CIR is encoded to the take pre-instruction primitive such that the next read of the response CIR by the main processor will terminate the dialog.

The MC68881 signals a protocol violation when unexpected accesses of the command, condition, register select, or operand CIRs occur. Coprocessor detected protocol violations occur when:

1) The MC68881 is expecting a write to the command or condition CIR, and instead an access of the register select or operand CIR occurs;
2) The MC68881 is expecting a read of the register select or operand CIR, and instead a write to the command, condition, or operand CIR occurs; and

3) The MC68881 is expecting a write to the operand CIR, and instead either a write to the command or condition CIR or a read of the register select or operand CIR occurs.

For the above three violations, the MC68881 maps the 16-bit register selector CIR onto the upper word of the 32-bit operand register. Thus, inconsistent data is read from the operand CIR, and write cycles can not store the correct value. Of course, this is of no consequence, since the protocol violation invalidates any operation being attempted by the MC68881 or the main processor.

During normal operation, the MC68881 synchronizes interprocessor communication by delaying the assertion of DSACK, if necessary. However, upon detection of a protocol violation, the MC68881 always terminates the access by immediately asserting DSACK. Thus, the system data bus can not lock up due to a coprocessor interface protocol violation.

A protocol violation can not occur as a result of an access to the reserved register locations, a read of a write-only register, or a write to a read-only register (a read of a reserved or write-only register always returns a value of all ones). One exception to this rule is that a write access to the register select CIR causes a protocol violation. Reads of the save or response CIR are always valid, as are writes to the restore or control CIR.

While the MC68881 requests that the MC68020 write to the instruction address CIR (by setting the PC bit in some primitives), accesses of this register are neither expected or unexpected. Thus, when the MC68881 is utilized as a peripheral processor where no concurrent instruction execution occurs, this request may be ignored without incurring a protocol violation exception. The FPIAR is updated by the MC68881 through the instruction address CIR by the MC68881. However, the FPIAR exists only to provide exception handlers with a pointer to faulty instructions following concurrent MC68020/MC68881 instruction execution.

A protocol violation is the highest priority MC68881-detected exception. It is also considered to be a fatal exception, since the MC68020 acknowledgment of the protocol violation exception clears any pending MC68881 instruction exceptions or illegal instructions.

NOTE

To distinguish between a protocol violation detected by the MC68020 or the MC68881, an exception handler may read the response CIR and evaluate the primitive. If the protocol violation is detected by the MC68881 due to an unexpected access, the operation being executed previously is aborted and the MC68881 assumes the idle state when the exception acknowledge is received. Therefore, the primitive read from the response CIR is null (CA = 0). If the protocol violation is detected by the MC68020 due to an illegal primitive, the MC68881 response CIR contains that primitive when the exception handler reads it (since the MC68881 **can not** internally generate an illegal primitive, an MC68020 detected protocol violation indicates a hardware failure).

To read the response CIR in an hardware independent manner, the move alternate address space (MOVES) instruction can be utilized. For example,

the following instruction sequence reads the response CIR of the coprocessor with CPID = 1 into an MC68020 data register:

```
MOVE.B    7,D0            Prepare the SFC register
MOVEC     D0,SFC          for a CPU space cycle...
MOVES.W   $00022000,D0    Execute a "coprocessor" cycle.
```

4.1.5 Recovery from Exceptions

When an MC68881-detected exception occurs, enough information is made available to the trap handler to perform the necessary corrective action and then resume execution of the program that caused the exception. Of course, in some instances it may not be valid to resume execution of the program; and for protocol violations, recovery is not possible. The information available to the exception handlers was described previously, and the following paragraphs summarize the methods used to resume execution of a program after an exception occurs.

In all cases, the stack frame generated by the MC68020 in response to an MC68881-detected exception contains a program counter value that points to the instruction to be executed upon return from the exception handler. In the case of pre-instruction exceptions, the instruction to be executed upon return is the MC68881 instruction that was attempted, but preempted by a pending exception. For mid-instruction exceptions (other than interrupts), two pointers are saved: the address of the MC68881 instruction that caused the exception and the address of the instruction immediately following that MC68881 instruction. Furthermore, the FPIAR contains a pointer to the MC68881 instruction that caused the exception in both cases. Thus an exception handler can always locate the instruction that caused an exception, and identify the next instruction to be executed upon return from the handler.

When the MC68020 executes a return from exception (RTE) instruction, it reads the stack frame from the top of the active system stack and restores that context. In the case where the stack frame was generated by an MC68881 pre-instruction exception, the context that is restored is that the MC68020 is ready to begin execution of the MC68881 instruction when the RTE is completed. The MC68881 instruction begins execution in the normal manner, with the MC68020 writing the coprocessor command word to the MC68881.

In the case where the RTE stack frame is generated by an MC68881-signaled mid-instruction exception (i.e., it is caused by an error during an FMOVE FPn,<ea> instruction, not by an interrupt during such an instruction), the context restore operation is slightly different than that just described. In this case, the MC68020 must complete execution of the instruction that was suspended by the exception. When the RTE instruction completes execution, the MC68020 first reads the response CIR of the MC68881 to determine the next appropriate action. Since the MC68881 always finishes execution of the instruction that causes this type exception before reporting it, the response that is returned is null (CA = 0, PF = 1), which releases the main processor to continue with the execution of the next instruction. Note that after a take mid-instruction exception primitive is returned, the main processor is not required by the MC68881 to perform a read from the resonse CIR before initiating the next floating-point instruction; but the MC68020 always performs this action when processing a mid-instruction stack frame.

An arithmetic exception handler (i.e., exception handlers other than the BSUN handler) routine is not required to perform any action to clear the cause of an exception. In fact, an arithmetic exception handler may consist of a single RTE instruction (which produces the same logical effect as disabling an exception). This is due to the fact that when the MC68881 signals an exception to the MC68020, the main processor acknowledges the exception by writing to the control CIR; and the exception acknowledge clears any pending exceptions in the MC68881. Thus an arithmetic exception handler is not required to clear any status bits or read any MC68881 registers in order to prevent the recurrence of an exception when an RTE instruction is executed. In the case of the BSUN exception handler, some action must be taken (as described in **4.1.2.1. BRANCH/SET ON UNORDERED (BSUN)**) by the exception handler to avoid an infinite loop condition.

If an exception handler is required to execute any MC68881 instruction other than an FMOVEM, an FSAVE should be the first MC68881 instruction to be executed. This assures that an exception handler can not generate any exceptions related to, or modify the context of, the program that caused the exception. It should also be noted that the FPIAR value must be saved before any instruction other than an FMOVEM is executed, so that the address of the instruction that caused the exception is not lost. When the exception handler completes the error recovery and is prepared to return to the suspended program, an FRESTORE is executed as the last MC68881 instruction; this restores the previous context of the program that caused the exception.

4.2 MAIN PROCESSOR DETECTED EXCEPTIONS

The following paragraphs describes exceptions which may be detected by the MC68020 during MC68881 instruction execution. Refer to the *MC68020 32-Bit Microprocessor User's Manual* for additional information on these exceptions, and the pre- and mid-instruction exception stack frames.

4.2.1 Trap on Coprocessor Condition Instructions

The MC68881 trap on condition instructions are initiated when the MC68020 writes a conditional predicate to the MC68881 for evaluation and reads a true/false condition evaluation in the MC68881 response primitive. If the MC68881 indicates that the condition is true, the MC68020 takes a post-instruction exception using the TRAPV/TRAPcc vector number.

The stack frame generated by the MC68020 in response to this exception contains two pointer values: 1) a pointer to the FTRAPcc instruction that caused the exception, and 2) a pointer to the instruction that follows the FTRAPcc (which is where the processor returns if an RTE instruction is executed).

4.2.2 Illegal Instructions

The MC68881 instructions consist of an operation word, a coprocessor command word (if any), and extension words (if any). The MC68881 detects illegal command words whereas the MC68020 detects illegal operation words. When the MC68020 detects an illegal operation word on a coprocessor instruction, it takes a pre-instruction exception using the F-line emulator vector number. Refer to **3.5 INSTRUCTION ENCODING DETAILS** for specific bit patterns which are illegal coprocessor operation words.

In addition to detecting an illegal operation word, the MC68020 may detect an illegal instruction even though the operation word is valid. This is due to the fact that the MC68881 either implicitly or explicitly indicates the valid addressing modes for an instruction whenever it returns a primitive response to the MC68020 that requests a data transfer to/from the effective address. Thus, the MC68020 may decide that properly formed MC68881 operation words and primitive responses are invalid if they specify operations which are illegal, such as writing a to non-alterable effective address.

When the MC68020 detects an invalid instruction in this manner, it terminates the MC68881 execution of the instruction by writing an abort to the control CIR. The MC68020 then takes a pre-instruction exception using the F-line emulator vector number. Termination of the MC68881 instruction execution in this manner does not alter any visible processor or coprocessor registers or status (such as pending coprocessor exceptions). Use of the F-line emulator trap allows the operating system to emulate any extensions to the MC68881 that are not supported by a specific processor.

4.2.3 MC68020 Detected Protocol Violations

If the MC68020 reads an MC68881 response primitive which it interprets as an illegal primitive, it does not terminate the MC68881 execution of the instruction by writing to the coprocessor interface control register. Instead, the MC68020 takes a mid-instruction exception using the coprocessor protocol violation vector number.

Since the MC68881 never issues an illegal response primitive, this feature of the MC68020 serves as a protection mechanism for interprocessor communications. If a protocol violation is taken on an MC68881 instruction, whether detected by the MC68881 or the MC68020, a system failure may be assumed. Refer to **4.1.4 MC68881 Detected Protocol Violation** for an example of how an exception handler can determine the cause of a protocol violation.

4.2.4 Trace Exceptions

To aid in program development, the MC68020 includes a facility to allow instruction-by-instruction tracing. In the single-step trace mode, after each instruction is executed, the MC68020 takes a post-instruction exception using the trace vector number. This allows a debugging program in the trace exception handler to monitor the execution of a program under test. Refer to the *MC68020 32-Bit Microprocessor User's Manual* for a complete description of how the trace mode is used.

Many MC68881 instructions may operate concurrently with MC68020 instructions, and defer the reporting of coprocessor detected instruction exceptions until the next MC68881 instruction is dispatched by the MC68020. This provides the user with a sequential instruction execution model even though concurrent instruction execution may occur. To guarantee that pending exceptions are always reported at the same point in an instruction sequence, regardless of whether tracing is enabled or not, the MC68881 always releases the MC68020 at the end of a concurrent instruction before reporting the exception. This is important, because the MC68020 (when in the trace mode) waits for an instruction to complete before proceeding.

In the trace mode, the MC68881 could report an exception as a post-instruction exception by issuing the null (CA = 0, PF = 0) primitive until the instruction is completed, and then issue the take post-instruction exception primitive to report the exception. However, this action by the MC68881 would change the point of detection of an exception from the beginning of the next instruction (with tracing disabled) to the end of the current instruction (with tracing enabled). This is undesirable because an error in a program might then occur with the trace mode disabled, and not occur in the trace mode. To provide consistent reporting of exceptions, the MC68881 always returns the null (CA = 0, PF = 1) primitive when it completes execution of a concurrent instruction, and then reports a pending exception only after a write to the command or condition CIR.

The synchronization of the two devices in the trace mode is accomplished through the PF bit in the null primitive (see **5.1.1 Null Primitives**). When the trace mode is enabled, the MC68020 repeatedly reads the response CIR to determine when the MC68881 completes instruction execution. If the null (CA = 0, IA = 1, PF = 0) primitive is read, then the MC68020 checks for pending interrupts, and if none are pending, reads the response CIR again. This process continues until the MC68020 receives a null (CA = 0, PF = 1) primitive from the MC68881.

In order for a trace exception to be transparent to normal program execution, the trace handler routine must take certain precautions in order to not disturb the context of the MC68881. When the main processor detects an exception, it automatically saves the most volatile portion of the current context and processes the exception immediately; thus the trace handler routine is not required to perform any MC68020 context save in order for the system to operate properly. The MC68881 does not operate in this manner, since it can not initiate exception processing until the MC68020 attempts to execute a new floating-point instruction. Also, the context information that must be saved for the MC68881 is more extensive than that of the main processor; thus, it is left up to software to perform the save only when necessary. The important consideration for a trace exception handler is that it must perform a more extensive context save for the MC68881 than for the MC68020 (since part of the MC68020 context save is automatic). Also, it should not execute any MC68881 instruction that may cause a pending exception to be reported,or a new exception to occur.

Given the constraints just described, it is apparent that the first and last MC68881 instructions that should be executed by a trace exception handler are the FSAVE and FRESTORE instructions, respectively. By executing the FSAVE instruction before any other floating-point instruction, any pending exceptions are saved in a state frame and then cleared internally; thus an exception generated by the main program can not be reported while the trace exception handler is executing. After the FSAVE instruction is executed, the

FMOVEM instruction can be used to save the user visible portion of the MC68881 context, and then the trace handler is free to utilize the coprocessor as desired, without affecting the main program context. When the trace handler is ready to return to the main program, the FMOVEM instruction is used to restore the user visible context, followed by an FRESTORE instruction to reinstate the exact context of the MC68881 prior to the trace exception processing. Note that since the MC68020 is forced to wait until the completion of an MC68881 instruction before processing a pending trace exception, the execution of the FSAVE instruction by the trace handler will always result in an idle state frame being saved and the user visible registers reflect the results of the last floating-point instruction. This is not the case if the trace exception handler is allowed to begin execution before the MC68881 instruction is completed. Processors other than the MC68020 must implement the trace synchronization mechanism in software (by polling the PF bit) in order to assure these conditions.

Another consideration important to a trace handler in an interrupting environment is that an interrupt can temporarily break the synchronization of the MC68020 and the MC68881. This can occur because when the MC68020 is in the trace mode and receives a null (CA = 0, IA = 1, PF = 0) primitive, it checks for interrupts and processes them if necessary before reading the response CIR again; if an interrupt is pending, the interrupt exception will be processed immediately. In response to the interrupt, the MC68020 saves a 10-word mid-instruction stack frame, with the trace pending status saved as part of the previous context information. When the interrupt handler completes execution and performs an RTE instruction, the MC68020 returns to the trace pending mode and reads the response CIR to determine if the previous coprocessor instruction is completed. In this manner, the exception processing for the interrupt is completely transparent to the handling of the trace exception by the MC68020/MC68881 pair. However, the interrupt handler must treat the MC68881 in the same manner as described above for the trace handler; i.e., if it requires the use of the MC68881, an FSAVE instruction must be executed to save the context of the coprocessor before any other floating-point instructions are executed.

4.2.5 Interrupts

When the MC68881 is busy executing an instruction, it may issue a null (CA = 1, IA = 1) primitive response which requests the MC68020 to continue polling the response register (this only occurs if the MC68881 requires additional services from the MC68020 for the current instruction). When this occurs, the MC68881 indicates to the MC68020 that it may sample interrupts between reads of the response CIR. If there is no interrupt pending, the MC68020 simply reads the response CIR again. If there is an interrupt pending, the MC68020 takes the interrupt exception using the mid-instruction stack frame. Upon exiting from the interrupt handler, the MC68020 re-polls the MC68881 response CIR to continue the suspended instruction dialog.

As described in **4.2.4 Trace Exceptions**, an interrupt handler must use the proper protocol to save the context of the MC68881 in order to be transparent to the main program. If the interrupt handler requires the use of the MC68881, or if a task switch requires that the context be saved, then an FSAVE instruction should be the first floating-point instruction executed by the routine. If an interrupt handler does not interact with the MC68881, then no context save operations are required.

Many MC68881 instructions can require a fairly long time to execute, and the MC68020 may be forced to wait until the MC68881 execution is complete before proceeding to the next instruction (because either the instruction does not allow concurrency or the main processor is in the trace mode). Normally, the MC68020 can only process pending interrupts when it reaches an instruction boundary, but this might adversely affect interrupt latency if it is not allowed to process interrupts while waiting on the MC68881. To reduce interrupt latency as much as possible, the MC68881 always sets the interrupts allowed (IA) bit in the null (CA = 1) and null (CA = 0, PF = 0) primitives; thus allowing interrupts to be processed while the MC68020 is waiting on the coprocessor to complete an operation. In fact, most MC68881 instructions, regardless of their overall execution time, provide for very small interrupt latency times. The worst case interrupt latency instruction for the MC68881 is the FRESTORE with a busy state frame (see **6.3 INTERRUPT LATENCY TIMES** for more information).

4.2.6 Address and Bus Errors

Bus cycle faults may occur while processing MC68881 instructions during the MC68020 accesses of the coprocessor interface registers, or during memory cycles run by the MC68020 to access instructions or data. If the MC68020 receives a fault while running the bus cycle which initiates an MC68881 instruction (i.e., the initial write to the command or condition CIR), it assumes that no MC68881 is present in the system, and takes a pre-instruction exception using the F-line emulator vector number. Thus, an MC68020 system may utilize software emulation of the MC68881 or provide hardware floating point, and the actual configuration is transparent to the application program. If any other access to the MC68881 is faulted, it is assumed that the coprocessor has failed, and MC68020 takes a bus error exception.

If the MC68020 has a memory fault while executing an MC68881 instruction, it takes an address error or bus error exception. After the fault handler corrects the fault condition, it may return and communication with the MC68881 continues as if the fault had not occurred. If the processor is to be redispatched while the fault condition is being corrected, the state of the MC68881 may be saved via the FSAVE instruction (see **4.3 CONTEXT SWITCHING**).

4.2.7 Privilege Violations

The MC68020 operates in one of two states of privilege — the user state or the supervisor state. The privilege state determines which operations are legal, and the S bit in the MC68020 status register determines the privilege state. Most programs execute in user state where accesses are controlled, and effects on other parts of the system are limited. The operating system executes in supervisor state, has access to all resources, and may execute all instructions; hence, it performs the overhead tasks for the user state programs.

The MC68881 FSAVE and FRESTORE instructions are privileged instructions, while all others are non-priviliged. An attempt to execute the FSAVE or FRESTORE instructions while in the user privilege state will result in the MC68020 taking a pre-instruction exception using the privilege violation vector number.

4.2.8 Format Error Exceptions

When the FRESTORE instruction is executed, the MC68881 checks the validity of the format word written to the restore CIR by the MC68020. Refer to **4.3.2 State Frames** for information on the format word. If the format word is invalid, this is indicated to the MC68020 when it reads back the contents of the restore CIR. The MC68020 then takes a pre-instruction exception using the format error vector number.

4.3 CONTEXT SWITCHING

In most types of multitasking systems, it is often necessary to take control from one program and give control to another program. This requires the operating system to extract (from the MC68881) data corresponding to one program context and load the context corresponding to the next program to be executed. The information that must be exchanged is divided into two categories:

1) Programmer's model — consists of data accessible by the programmer using non-privileged instructions. This data is saved and restored using the FMOVEM instructions.
2) Internal state — consists of various internal flags and registers that the application program need not be concerned with, but is vital in restoring the MC68881 to the proper state. These internal lags and registers are accessed by the privileged FSAVE and FRESTORE instructions.

The following paragraphs describe how this context information is manipulated.

4.3.1 FSAVE and FRESTORE Instructions, Overview

The basic mechanism for performing a context switch on the MC68881 is provided through the FSAVE and FRESTORE instructions. These instructions provide a logical extension to the instruction continuation mechanism that is used by the MC68010 and MC68020 processors to support virtual memory. The FSAVE instruction is treated much like a microcode level interrupt to the MC68881, instructing it to suspend any operation that is being executed (at the earliest possible boundary) and make a complete copy of the internal state of the machine in memory. This is similar to the effect of the assertion of bus error to the main processor. To restore the internal state saved by the FSAVE instruction, the FRESTORE instruction is used, which is similar to the RTE instruction on the main processor.

The internal state information that is stored in memory by the FSAVE instruction contains the image of the non-user visible portion of the machine, including the address of the microprogram counter, temporary register values, and pending exception information. After the execution of an FSAVE, the MC68881 enters the idle state, and any pending exceptions are cleared. To perform a complete context save, the FMOVEM instruction can then be used to save the user visible portion of the machine; and then a new context may be loaded. When it is necessary to reload the context that was previously saved, these steps are reversed: first the FMOVEM instruction loads the user visible context, followed by an FRESTORE instruction which loads the non-user visible context. After the execution of the FRESTORE, the MC68881 returns to the exact context that existed just before the FSAVE instruction was executed, and execution continues from that point.

Depending on the state of the MC68881 when an FSAVE instruction is executed, the format of the internal state information written to memory may be in one of three forms. Also, the MC68881 may force the MC68020 to wait for a short time while the internal state is prepared for the save operation. During execution of an FRESTORE instruction, the MC68881 interprets the state information read from memory to determine the appropriate response action. The FRESTORE is a destructive command, in that the MC68881 immediately stops any operation that it may be performing and begins to load the next context; thus there is no need for a mechanism in the FRESTORE instruction to allow the MC68881 to make any service requests to the MC68020. The protocol of the FSAVE and FRESTORE instructions are detailed below, following a description of the various state frame formats generated by the MC68881.

4.3.2 State Frames

The three state frame formats that are generated by the MC68881 are shown in Figure 4-4. In all three state frames, the first long word of the frame has the same format. The least significant word of this long word is reserved for future definition by Motorola; it is included to allow long word alignment of a state frame in memory. The most significant word of the first long word (called the format word) contains the version number of the coprocessor that generated the state frame (in the most significant byte) and the size of the internal state stored in the frame (in the least significant byte). Although the version number and frame size values are defined by the MC68881, the M68000 Family coprocessor interface defines the null format word which is the one format word value that must be recognized by any coprocessor and is described below.

When an FSAVE instruction is executed, the format word is the first data item transferred to MC68020, and the main processor uses the size value to perform the correct address calculations. During an FRESTORE instruction, the format word is written to the MC68881 to initiate the restore operation. When this occurs, the MC68881 checks the version number and frame size values for validity and signals a format exception if they are not valid for this particular device. The version number is an 8-bit value that identifies the microcode version of the MC68881, and the format of this number is defined internally by the MC68881. Future devices will use a unique combination of the version number and frame size values in order to guarantee that various revisions of the device can not incorrectly utilize an internal state frame that is not valid for that revision.

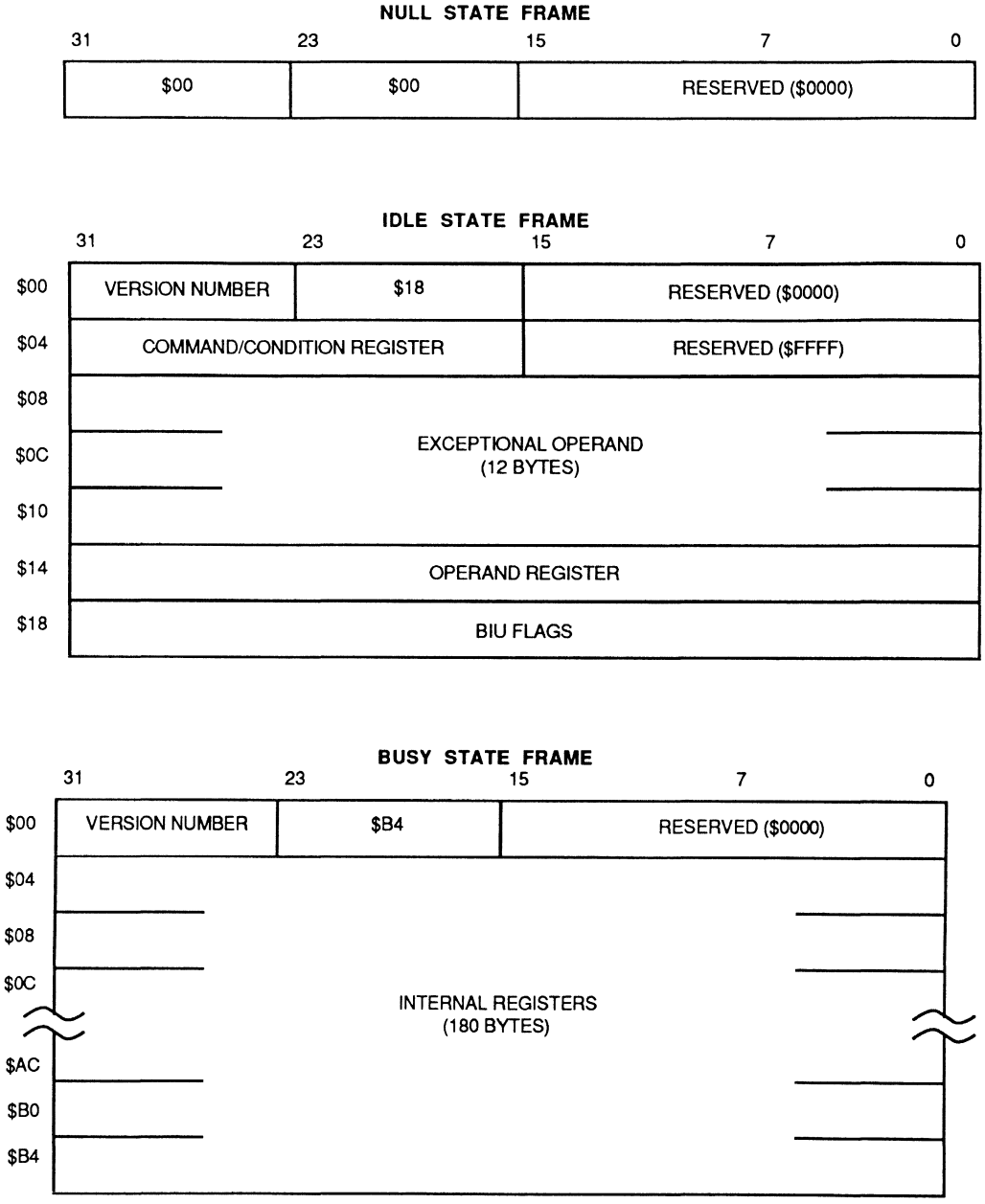


Figure 4-4. MC68881 State Frame Formats

In addition to being used by the MC68881 to validate a state frame before it is used in a restore operation, the format word may be used by a user program to identify the format of a state frame and the state of the MC68881. In the following descriptions of the three state frames, the data format within a frame is guaranteed only for those version number and frame size values given in the accompanying tables. Routines that utilize state frame information must examine the format word to correctly identify any data formats that are subject to change by Motorola.

NOTE

The state size value in the format word indicates the size (in bytes) of the MC68881 internal state information. This size value **does not** include the format word or the reserved word.

4.3.2.2 NULL STATE FRAME. As shown in Figure 4-4, no internal state information is saved in the null state frame. Only the coprocessor version number (0) and the state frame size (0 bytes) are indicated. Version number 0 is a wild card number, allowing this state frame type to be restored to a coprocessor of any version. The size value of a null state frame is not assumed to be valid during a save operation and is ignored by the MC68881 during a restore operation. A restore of the null state performs the reset function, with all floating-point data registers loaded with NANs and the FPCR set to zero. A save of the null state results when no MC68881 instructions have been executed since the last null state restore or hardware reset function. Note that a save of a null state indicates that the MC68881 programmer's model is empty, and thus does not need to be saved with a FMOVEM instruction.

4.3.2.1 IDLE STATE FRAME. As shown in Figure 4-4, 24 bytes of internal state are saved in the idle state format. The format word indicates the coprocessor version number and state size (24 bytes). The idle state is produced if an FSAVE occurs when a floating-point instruction is not being executed, or when the current instruction is in the end phase (refer to **4.3.3 FSAVE and FRESTORE Protocols** for a definition of the end phase).

In addition to being used for context switching, the idle state frame contains information that is useful to most floating-point exception handlers. First, it contains the exceptional operand value, which can be evaluated by an exception handler to determine the cause of an exception. Second, it contains the BIU flag word that indicates the status of the MC68881 at the time of an FSAVE instruction. For example, this can be useful to a trace exception handler, to allow a debug monitor routine to display the pending exception status along with the register state of the machine.

As shown in Figure 4-3, the idle state frame contains four data items—the command/condition register image, the exceptional operand, the operand register image, and the BIU flags. A reserved word is also included in order to long word align the state frame; it is written as $FFFF and ignored during restore operations. The command/condition word and operand register may contain temporary information, as indicated by the BIU flags.

The format of the BIU flag word is shown in Figure 4-5. Only 10 of the 32 bits in the BIU flag word are defined; the undefined bits are written as ones during save operations and ignored during restore operations. The definitions of the 10 flag bits are given below.

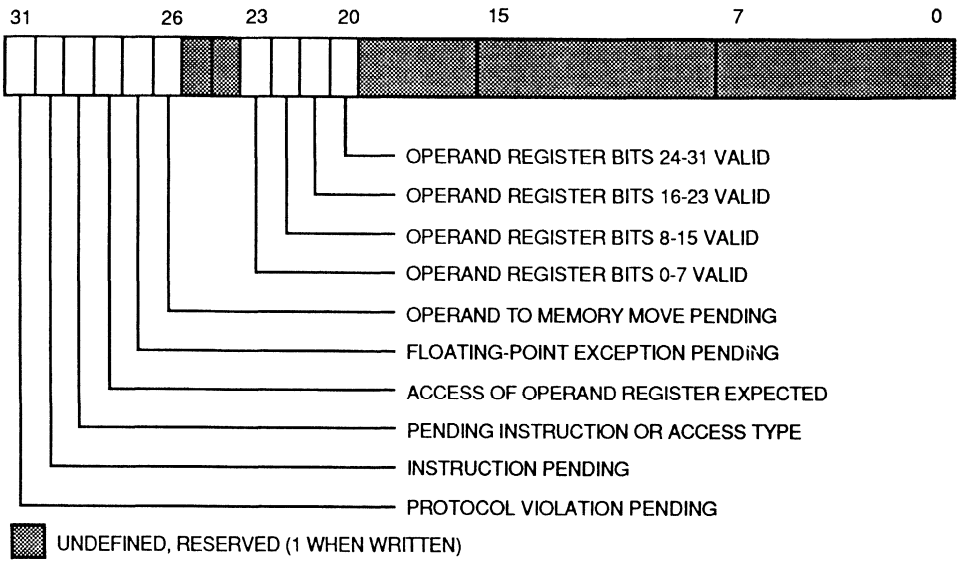


Figure 4-5. BIU Flag Format

Bits 20–23 These bits are set when valid data is contained in the operand register image of the state frame. There is one flag bit for each byte in the 32-bit operand register image; if a bit is one, there is valid data in the corresponding byte. If a bit is zero, then the data in the corresponding byte is assumed to be invalid. These bits can be used to qualify the image of the operand register and should not be modified.

Bit 26 This bit indicates that the MC68881 has completed any necessary operand conversions and is ready to write an operand to memory. If this bit is a zero, then an operand transfer to memory is pending. This bit should not be modified.

Bit 27 This bit indicates that a floating-point exception is pending which will be reported when the MC68020 attempts to initiate the next floating-point instruction (after an FRESTORE of this state frame). If this bit is zero, then an exception is pending, and the logical AND of the FPSR EXC and FPCR ENABLE bytes indicates the type of exception that is pending. This bit may be read by an exception handler (particularly a trace routine) to determine the

exception status of the MC68881. As described below, a user program may modify this bit and the FPSR EXC and FPCR ENABLE byte images to create a pending software generated exception pending.

Bit 28 This bit indicates that the MC68881 is expecting the next coprocessor interface register access to be to the operand CIR. This bit is used by the BIU as part of the protocol violation checking hardware, and should not be modified. If this bit is a zero, an access of the operand CIR is pending, and the state of bit 29 determines whether the expected access is a read or write cycle. Table 4-4 shows the definitions of the state of this bit.

Bit 29 This bit defines the type of pending operand access that is expected or the type of pending operation that is saved in the command/condition register image. This bit should not be modified. Table 4-4 shows the definitions of the state of this bit.

Bit 30 This bit indicates that the MC68881 has received a new command word or conditional predicate from the MC68020, but has not been able to begin execution of that operation. If this bit is zero, the command word or conditional prediciate that was received is contained in the command/condition register images of the state frame. This bit should not be modified. Table 4-4 shows the definitions of the state of this bit.

Bit 31 This bit indicates that a protocol violation has been detected by the MC68881, and the MC68020 has not responded with an exception acknowledge or abort operation. If this bit is a one, a protocol violation is pending. This bit should not be modified.

Table 4-4. BIU Flag Bit Definitions

30	29	28	Definition
0	0	0	(Undefined, Reserved)
0	0	1	Conditional Instruction Pending
0	1	0	(Undefined, Reserved)
0	1	1	General Instruction Pending
1	0	0	Write of Operand CIR Pending
1	0	1	(Undefined, Reserved)
1	1	0	Read of Operand CIR Pending
1	1	1	No Pending Instruction or Operand CIR Access

NOTE

The format of the idle state frame and the BIU flags given above are for the initial production version of the MC68881; this format is identified by the format word value $1F18. Motorola reserves the right to utilize different state frame formats and format word values to support future revisions to the MC68881.

The only bit in the BIU flag word that may be modified by software is bit 27, the pending exception bit. If this bit is zero, then an exception is pending and may be cleared by changing it to a one. Alternatively, the type of the pending exception may be changed by modifying the FPSR EXC and/or FPCR ENABLE byte(s) before executing an FRESTORE. Finally, if the pending exception bit is one (indicating that no exception is pending), it may be changed to make an exception pending; the type of exception pending is defined by the FPSR EXC and FPCR ENABLE bytes. In all of these cases, the change in the exception status takes effect when the state frame is utilized by an FRESTORE instruction.

The exception pending bit (referred to as EXC_PEND) in the BIU flag word is the image of the exception pending signal internal to the MC68881. Normally EXC_PEND is negated by the MC68881 execution unit when an instruction (other than an FMOVEM, FMOVE control register, FSAVE, FRESTORE) begins execution, and is asserted if an exception occurs during the instruction. The bus interface unit uses EXC_PEND to determine the primitive response that is encoded in the response CIR after a write to the command or condition CIRs, or after the completion of the transfer of a floating-point operand to memory. If EXC_PEND is true when an attempt is made to initiate an MC68881 instruction (other than an FMOVEM, FMOVE control register, FSAVE, or FRESTORE), then the response CIR is encoded to the take pre-instruction exception primitive; otherwise, the dialog for the instructon is started. If EXC_PEND is true at the end of the move of a floating-point operand to memory, then the response CIR is encoded to the take mid-instruction exception primitive; otherwise it is encoded to the null (CA = 0, PF = 1) primitive. The vector number that is encoded in the take exception primitive is determined by the state of the FPSR EXC and FPCR ENABLE bytes, corresponds to the highest priority exception that is enabled. When the MC68020 responds to the take exception primitive, by writing an exception acknowledge to the control CIR, EXC_PEND is cleared by the MC68881.

With this understanding of how EXC_PEND (and its image in the BIU flag word) affects the operation of the MC68881, a programmer can make exceptions pending in the MC68881 under software control. Or, conversely, a pending exception type may be changed or cleared if necessary.

4.3.2.3 BUSY STATE FRAME. As shown in Figure 4-4, 180 bytes of internal state are saved in the busy state format. The format word indicates the coprocessor version number and state size (180 bytes). The busy state is produced if an FSAVE occurs when a floating-point instruction is in the initial or middle phase. Due to the volatile nature of the MC68881 internal state during calculation, this state frame does not contain any information useful to applications programs, and the frame should not be modified in any way.

4.3.3 FSAVE and FRESTORE Protocols

The following paragraphs describe the bus cycles that occur between the MC68020 and the MC68881 during FSAVE and FRESTORE instructions and the conditions under which each size of state frame is produced. To describe the response of the MC68881 to an FSAVE instruction, the execution state is identified as one of five phases shown in Table 4-5.

Table 4-5. MC68881 Responses to Save Command

Phase Name	Response Time	State Frame Type
Reset	Immediate	Null
Idle	Immediate	Idle
Initial	Immediate	Busy
Middle	Periodic	Busy
End	Delayed	Idle

4.3.3.1 FSAVE PROTOCOL. When the MC68020 encounters an FSAVE instruction, it attempts to initiate a save operation in the MC68881 by first reading from the save CIR. If the MC68881 is ready to perform the save, it responds with a valid state frame format word which informs the MC68020 that the state frame transfer may begin and what size frame is to be saved. If the MC68881 is not ready to begin the transfer of the state frame, it returns the come-again format word, forcing the MC68020 to wait. When the MC68020 receives the come-again format word, it checks for pending interrupts and processes them if necessary. Otherwise, it repeatedly reads the save CIR until a non-come-again format word is returned. When a valid format word is received by the MC68020, it reads the number of bytes indicated by the format word, four bytes at a time, from the operand CIR and writes them to memory.

The MC68881 always returns one of five format words when the save CIR is read by the MC68020. Table 4-6 shows the five format word values and their meanings. In this figure, the version number of the idle and busy format words, $1F, corresponds to the version number of the first production version of the MC68881; future revisions of the device will utilize different version numbers to identify unique state frame formats. If the format of the idle state frame of a future version of the MC68881 differs from that of version $1F, Motorola will provide the new format information when the new version is available.

The come-again format word is returned by the MC68881 to force the MC68020 to wait, as described above. When the MC68881 is ready to complete a save operation, one of the other valid format words (null, idle, or busy) is returned to the main processor, and then the appropriate state frame is transferred to memory. The only time that the MC68881 uses the illegal format word is when a read of the save CIR occurs when the MC68881 is in the process of performing a state save or state restore. Normally, this only occurs when the execution of an FSAVE or FRESTORE instruction is suspended (e.g., due to a page fault during the save or restore operation) and an attempt is made to execute a new FSAVE instruction. If this happens, the illegal format word is returned to cause a format exception to be taken by the main processor. When the MC68020 receives the illegal format word, an

Table 4-6. MC68881 Format Word Definitions

Format Word	Definition and Frame Size
$0018*	Null State
$0118*	Come Again
$0218*	Illegal, Format Error
$1F18	Idle State
$1FB4	Busy State

*The frame size byte for these format words is undefined for the M68000 Family coprocessor interface. Version $1F of the MC68881 returns the value of $18 for the frame size in these format words; however, future devices are not guaranteed to do this.

abort is written to the control CIR and then exception processing is initiated. In this case, the format error handler routine examines the instruction that was being executed when the format error occurred and can determine that the second FSAVE instruction failed due to "nesting" of save or restore operations. Such an error is considered to be a catastrophic system error, since the MC68881 context is lost and can not be recovered.

When an idle or busy format word is received by the MC68020, it transfers the number of bytes in the frame (four bytes at a time) to memory. First, the format word is written to memory at the evaluated effective address. For the predecrement addressing mode, the value of the specified address register is saved in a temporary register, the size of the state frame is subtracted from the address register, and the format word is pushed to that address (thus the required stack space is allocated before the save operation is started). The state frame is then filled, from higher addresses to lower addresses, using the temporary register as a pointer. For the control alterable addressing modes, the format word is written to the specified address; then the address of the last word of the frame is calculated (in a temporary register) and the frame is filled from higher addresses to lower addresses. After the last byte of the state frame is written to memory, the MC68881 is in the idle state with no pending exceptions, and the MC68020 executes the next instruction (it does not read the save or response CIR after the save operation).

The following paragraphs describe the response of the MC68881 to an FSAVE instruction for the various phases of instruction execution.

4.3.3.1.1 Reset Phase. In this phase, no MC68881 instructions have been executed since the last hardware reset or FRESTORE of a null state frame. When the MC68881 is in this state and an FSAVE is executed, a null format word is returned immediately.

4.3.3.1.2 Idle Phase. In this phase, the MC68881 is not executing an instruction, but at least one instruction has been executed by it since the last hardware reset or FRESTORE of a null state frame. When the MC68881 is in this state and an FSAVE is executed, an idle format word is returned immediately, and an idle state frame is stored.

4.3.3.1.3 Initial Phase. In this phase, the MC68881 is acquiring instruction and operand words from the MC68020. In virtual memory systems, a memory fault can occur during this phase due to an attempt to access an operand that is not resident in main memory. In this case, the MC68020 traps to a fault handler to initiate a transfer from secondary storage, typically involving one or more disk accesses. After initiating the transfer, the operating system will usually switch the main processor and coprocessor(s) to another program, thus necessitating a save of the coprocessor state and restoration of the state of the coprocessor relative to the next program. To facilitate this, the MC68881 responds immediately to a save command during the initial phase with a dump of a busy state frame.

4.3.3.1.4 Middle Phase. The middle phase occurs only in MC68881 instructions that take significant processing time (i.e., remainder, transcendental functions, and BCD conversions). During this phase, the internal microcode sequence of the MC68881 provides for periodic checkpoints to determine if the MC68020 has issued a save command. If the MC68020 initiates a save command to the MC68881 between check-points, the MC68881 sets an internal flag denoting the receipt of the command and returns a come-again format word to the MC68020. The MC68020 repeatedly reads the save CIR until a valid format word is received. The MC68881 continues internal processing up to the next checkpoint, at which time processing stops, and the next read of the save CIR acquires the appropriate format word to start the save operation. At this point, the save command proceeds to completion, with the MC68881 supplying a busy state frame.

4.3.3.1.5 End Phase. This phase begins when the MC68881 is almost finished with a long instruction. The length of the end phase is approximately equal to the amount of time required to perform a save of a busy state frame. Once the MC68881 reaches the end phase, it takes less time to complete execution of the instruction and then save an idle frame than to immediately save a busy state. During this phase, the MC68881 uses the come-again format word to force the MC68020 to wait for the completion of the instruction., and then an idle state frame is saved.

Note that most of the MC68881 instructions proceed directly from the initial phase to the end phase, and thus most state frames that are generated by the MC68881 are idle frames.

4.3.3.2 FRESTORE PROTOCOL. When the MC68020 encounters an FRESTORE instruction, it evaluates the effective address to locate the format word for the state frame, and writes that format word to the restore CIR of the MC68881. In response to this write cycle, the MC68881 aborts any operation that may be in progress and prepares to load a new internal state. The format word that is written to the restore CIR is checked for validity (it must be a null, idle, or busy format word with a version number that matches that of the specified device) before the restore operation begins. After the MC68020 writes the format word to the MC68881, it then reads the restore CIR to verify that the format word is valid. If the format word is valid, the MC68881 returns the same format word that was written; if the format word is not valid, an illegal format word ($02xx) is returned. If the format word is successfully verified, the MC68020 begins to transfer the state frame, four bytes at a time, from memory to the MC68881 operand CIR.

When transferring the state frame from memory to the MC68881, the MC68020 first transfers the format word and, after it is verified, transfers the remainder of the state frame. The order of transfer is the same for both the postincrement and the control addressing modes, with the long word at the lowest address transferred first and then proceeding up through higher addresses. For the postincrement addressing mode, the specified address register is not updated by the MC68020 until the entire frame has been successfully transferred. Thus, a fault during an FRESTORE will generate a stack frame that does not overwrite any part of the MC68881 state frame.

After the entire state frame has been transferred to the MC68881, the MC68020 continues with the execution of the next instruction (it does not read the response CIR). If an exception related to the MC68881 caused the suspension of the task earlier, an RTE instructon will eventually be executed to return to the origianl context. Depending on the exception type, the RTE may re-establish the MC68020/MC68881 protocol of the suspended operation, or begin the execution of a new MC68881 instruction.

4.3.4 Context Switching Summary

To perform a complete context save or restore operation, three MC68881 instructions are required. First the FSAVE and FRESTORE instructions are used to transfer the non-user visible portion of the machine state between the MC68881 and memory. Second, the FMOVEM instruction may be used to transfer the user visible portion of the machine, including the floating-point data and control register. An important aspect of these instructions is that they can not cause or report a pending exception; thus the context of the MC68881, including pending exceptions, can be saved and restored in a manner that is completely transparent to a user program. Note that if an FSAVE instruction results in the generation of a null state frame, then the format of the floating-point data and control registers is known, and the FMOVEM instructions are not needed. Figure 4-6 illustrates the manner in which a full context switch might be performed.

```
SAVE OLD CONTEXT:
            FSAVE       -(An)                   SAVE MC68881 STATE FRAME
            TST.B       (An)                    CHECK FOR NULL FRAME
            BEQ         NULL_SV                 SKIP PROGRAMMER'S MODEL SAVE IF NULL
            FMOVEM      FP0-FP7,-(An)           ELSE, SAVE DATA REGISTERS
            FMOVEM      FPCR/FPSR/FPIAR,-(An)   AND SAVE CONTROL REGISTERS
            ST          -(An)                   PLACE NOT-NULL FLAG ON STACK
NULL_SV     ...

RESTORE NEXT CONTEXT:
            TST.B       (An)                    CHECK FOR NULL FRAME OR NOT-NULL FLAG
            BEQ         NULL_RST                SKIP PROGRAMMER'S MODEL RESTORE IF NULL
            ADDQ.L      #2,An                   ELSE, THROW AWAY THE NOT-NULL FLAG
            FMOVEM      (An)+,FPCR/FPSR/FPIAR   RESTORE THE CONTROL REGISTERS
            FMOVEM      (An)+,FP0-FP7           RESTORE THE DATA REGISTERS
NULL_RST    FRESTORE    (An)+                   RESTORE THE MC68881 STATE FRAME
```

Figure 4-6. Full Context Save/Restore Instruction Sequences

SECTION 5
COPROCESSOR INTERFACE

This section describes the coprocessor interface with respect to the communication protocol utilized by the MC68881 and MC68020. This communications protocol includes electrical and command level mechanism that allow a coprocessor to act as an extension to the main processor.

5.1 COPROCESSOR INTERFACE SIGNAL CONNECTION

The connection between the MC68020 and the MC68881 is a simple extension of the M68000 bus interface with the MC68881 connected as a peripheral to the MC68020. The selection of the MC68881 is based upon a chip select ($\overline{CS}$) signal that is decoded from the MC68020 function code and address bus lines.

The MC68881 contains a set of coprocessor interface registers (CIRs) by which the main processor and coprocessor communicate. These registers are not related to the programming model implemented by the MC68881. Rather, they are used as communication ports that have specific functions associated with each register. When the MC68881 is used as a coprocessor to the MC68020, the programmer is never required to explicitly access these interface registers, since the coprocessor interface is implemented in the hardware and microcode of the MC68020. When the MC68020 is not used as the main processor, the MC68881 CIRs are explicitly accessed by a software routine that simulates the behavior of the MC68020 with respect to the coprocessor interface.

For more information on the electrical interconnection between the main processor and the MC68881, refer to **SECTION 9 INTERFACING METHODS**.

5.1.1 Chip-Select Decode

The MC68020 does not require any special bus signals, beyond the normal M68000 Family bus control signals, for connection to the MC68881. The former MC68000 interrupt acknowledge address space (function code 111) is extended in the MC68020 to be the CPU address space. A portion of this space, identified by the MC68020 address bus, is dedicated to coprocessor devices. Figure 5-1 illustrates the information presented on the MC68020 address bus for coprocessor accesses in the CPU address space.

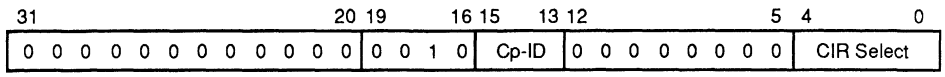


Figure 5-1. MC68020 Address Bus Encoding for Coprocessor Accesses

During CPU space cycles, address bits A16-A19 indicate the CPU space function that the main processor is performing. The MC68020 utilizes four of the possible 16 encodings of A16-A19 as listed in Table 5-1.

Table 5-1. MC68020 CPU Space Type Field Encoding

CPU Space Type Field (A19-A16)	CPU Space Transaction
0000	Breakpoint Acknowledge
0001	Access Level Control
0010	Coprocessor Communications
1111	Interrupt Acknowledge

The coprocessor identification (Cp-ID), A13–A15, is taken from the coprocessor instruction operation word (refer to **5.2 COPROCESSOR INSTRUCTIONS**). The coprocessor interface register (CIR select) field, A0–A4, is decoded by the MC68881 to select the appropriate CIR. For a map of the MC68881 coprocessor interface registers in the CPU address space, refer to Figure 5–2. Since address bits A20–A31 are not present on all implementations of M68000 processors, these bits are not essential for decoding CPU space transactions and therefore are don't care bits.

The MC68881 chip select decode is therefore based upon the MC68020 function code outputs (FC0–FC2), the CPU space type field (A16–A19), and the Cp–ID field (A13–A15). The MC68881 decodes the address bits A0–A4 to determine the function of any coprocessor access.

5.1.2 Coprocessor Interface Registers

Table 5-2 identifies the MC68881 coprocessor interface register locations in the CPU space which are used for communications between the MC68020 and the MC68881. Figure 5-2 illustrates the memory map of the CIRs on a 32-bit bus. When $\overline{CS}$ is asserted, the MC68881 decodes the CIR select field of the address bus (A0-A4) to select the appropriate coprocessor interface register.

When the MC68881 is used on a 32-bit bus, the coprocessor interface registers appear at the logical addresses shown in Figure 5-2 and Table 5-2. The M68000 dynamic bus sizing protocol is used to place all word registers on the upper word of the data bus (D16-D31). This is accomplished by asserting $\overline{DSACK1}$ and leaving $\overline{DSACK0}$ negated when any word register is accessed, regardless of the value of A1.

Table 5-2. Coprocessor Interface Register Characteristics

Register	A4–A0	Offset	Width	Type
Response	0000x	$00	16	Read
Control	0001x	$02	16	Write
Save	0010x	$04	16	Read
Restore	0011x	$06	16	R/$\overline{W}$
Operation Word*	0100x	$08	16	R/$\overline{W}$
Command	0101x	$0A	16	Write
(Reserved)	0110x	$0C	16	—
Condition	0111x	$0E	16	Write
Operand	100xx	$10	32	R/$\overline{W}$
Register Select	1010x	$14	16	Read
(Reserved)	1011x	$16	16	—
Instruction Address	110xx	$18	32	Write
Operand Address*	111xx	$1C	32	R/$\overline{W}$

*These CIRs are optionally implemented by a coprocessor only if they are needed; since they are not used by the MC68881, they are not implemented. Writes to these locations are ignored, and reads always return all ones.

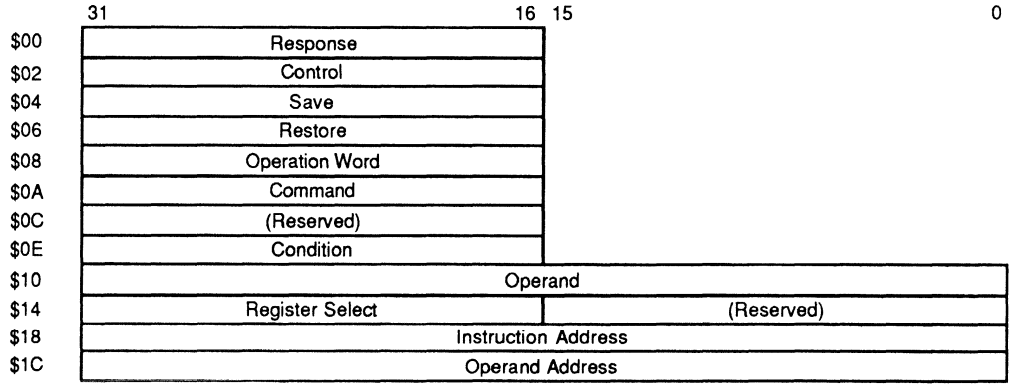


Figure 5-2. MC68881 Coprocessor Interface Register Map

The following paragraphs describe the characteristics of each of the coprocessor interface registers as implemented by the MC68881. In these descriptions, the read/write attributes of each register are given. If a register is read only, write accesses to that location are ignored; while read accesses of a write-only register always returns all ones. In all cases, the MC68881 asserts DSACK in response to the assertion of $\overline{CS}$ in order to terminate the bus cycle.

5.1.2.1 RESPONSE CIR ($00). This 16-bit read-only register is used to communicate service requests from the MC68881 to the main processor. A read of the response CIR is always legal, regardless of the state of an instruction dialog. The format of the response

primitive that are returned through this register are detailed in **5.2.2 Response Primitives.**

The execution of an instruction by the MC68881 does not start until the main processor reads the response CIR for the first time after a write to the command CIR. Furthermore, a read of a primitive from the response CIR usually causes the MC68881 to proceed to the next state in an instruction dialog. For example, if an evaluate effective address and transfer data primitive is encoded in the response CIR and the main processor reads that primitive, it is assumed that the primitive was successfully transferred (and saved for later use, if necessary) and that the requested service will be performed. In this case, the MC68881 then changes the encoding of the response CIR to the null primitive and waits for an access of the operand CIR to transfer the operand.

5.1.2.2 CONTROL CIR ($02). This 16-bit write-only register is utilized by a main processor to issue an exception acknowledge or instruction abort to the MC68881. Figure 5–3 illustrates the format of this register. Only two of the 16 bits are defined: the exception acknowledge (XA) and abort (AB) bits.

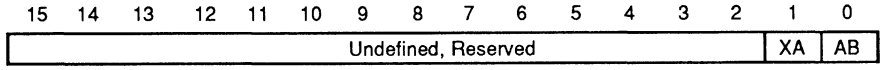


Figure 5-3. Control CIR Register

The implementation of the MC68881 does not utilize these two bits; but instead, simply interprets a write to this CIR address as an abort command, regardless of the data pattern written. Thus, an exception acknowledge (in response to a take exception primitive) or abort (in response to an illegal format word or an invalid request primitive) issued during any MC68881 instruction protocol, or an explicit write (e.g., with the MOVES instruction) to the control CIR always has the same effect on the MC68881. Also, write cycles to this register are never illegal, since the MC68881 always responds in the same manner.

The response of the MC68881 to a write of the control CIR is to:

1) Immediately terminate processing on any instruction that may be in progress. If an arithmetic instruction is in progress when an abort is issued, the content of the destination floating-point data register is undefined. No other user visible registers are disturbed,
2) Clear any pending exceptions, and
3) Reset the bus interface unit to the idle condition. Thus, the MC68881 is ready to begin a new instruction protocol following the write cycle.

5.1.2.3 SAVE CIR ($04). This 16-bit read-only register is used by the main processor to issue a context save command to the MC68881, and to return the format word of the MC68881 state frame to the main processor. A read of this register causes the operation currently being executed by the MC68881 (except a state save or restore) to be suspended, and a state save operation is initiated.

Following the read of a not ready, come again format word from the save CIR, the next expected access is a read of the save CIR. After the read of an idle or busy format word, the next expected access is to the operand CIR (to transfer the state frame). After the read of a null format word, the MC68881 is in the reset state and the next expected access is to the command or condition CIR.

The only time that a read of this register is illegal is when the MC68881 is executing a state frame transfer for an FSAVE or FRESTORE instruction; a read of the save CIR is legal at any other time. If the main processor reads the save CIR at an illegal time, the invalid format word is returned. In response to the invalid format word, the main processor may write an abort to the MC68881 to return it to the idle state.

5.1.2.4 RESTORE CIR ($06). This 16-bit read/write register is used by the main processor to issue a context restore command to the MC68881 and to validate the format word of a state frame. A write of this register causes the MC68881 to immediately stop any operation that may be executing and prepare to load a new internal state context from the memory resident state frame.

After the main processor writes a format word to the restore CIR, it must read the restore CIR to receive the results of the format word verification. If the previously written format word is valid, that format word will be read back from the restore CIR to indicate the successful verification. If the format word is invalid, the invalid format, take exception value is placed in the restore CIR to indicate the verification failure. After a successful verification is signaled, the next expected access is to the operand CIR (to transfer the state frame). After a verification failure is signaled, the main processor should write an abort to the control CIR in order to return the MC68881 to the idle state (the MC68020 does this automatically).

5.1.2.5 OPERATION WORD CIR ($08). This 16-bit write-only register is not used by the MC68881. The only time that this CIR location is used by the M68000 Family coprocessor interface is when a coprocessor issues the transfer operation word primitive, in which case the main processor writes the F-line word of the instruction to the operation word CIR. Since the MC68881 never issues the transfer operation word primitive, the operation word CIR location should never be written by the main processor. If a write to this location occurs, it will be ignored and will not cause a protocol violation.

5.1.2.6 COMMAND CIR ($0A). This 16-bit write-only register is used by the main processor to initiate the dialog for a general type coprocessor instruction. When the MC68881 detects a write to this CIR location, the data value is latched from the data bus. If the MC68881 is executing a previous instruction when the command CIR is written, the latched command word is saved for later use and the response CIR is encoded with the null (CA = 1, IA = 1) primitive. If the MC68881 is in the idle or reset state when a write to the command CIR occurs, it encodes the first primitive for the selected instruction dialog in the response CIR in order to begin the execution of the new instruction.

A write to this CIR location is legal at any time except when the MC68881 is in the initial phase of a general instruction or before the read of the conditional evaluation for a previous conditional instruction. If a write to the command CIR occurs when it is not expected, a protocol violation occurs, and the command word that is written is not saved by the MC68881.

5.1.2.7 CONDITION CIR ($0E). This 16-bit write-only register is used by the main processor to initiate the dialog for a conditional type coprocessor instruction. When the MC68881 detects a write to this CIR location, the data value is latched from the data bus. If the MC68881 is executing a previous instruction when the condition CIR is written, the latched conditional predicate is saved for later used, and the reponse CIR is encoded with the null (CA = 1, IA = 1) primitive. If the MC68881 is in idle or reset state when a write to the condition CIR occurs, it evaluates the selected condition and returns the null (CA = 0, TF = x) primitive (where the TF bit indicates whether the conditional evaluation is true or false).

A write to this CIR location is legal at any time except when the MC68881 is in the initial phase of a general instruction, or before the read of the conditional evaluation for a previous conditional instruction. If a write to the condition CIR occurs when it is not expected, a protocol violation occurs, and the conditional predicate that is written is not saved by the MC68881.

5.1.2.8 OPERAND CIR ($10). This 32-bit read/write register is used by the main processor to transfer data to and from the MC68881. The MC68881 transfers data through this CIR location in the following cases:

1) following an evaluate effective address and transfer data primitive,
2) following the read of the register select CIR after a transfer multiple coprocessor registers primitive,
3) following a transfer single main processor register primitive,
4) following a read of an idle or busy format word from the save CIR, and
5) following a write of an idle or busy format word to the restore CIR.

These five cases are the only times when an access to the operand CIR is legal. At any other time, an access to this CIR location causes a protocol violation.

The MC68881 expects all operands which are to be transferred through this CIR location to be aligned with the most significant byte of the register. Any operand larger than four bytes is transferred through this register using a sequence of long word transfers. If the operand is not a multiple of four bytes in size, the portion remaining after the initial long word transfers is aligned with the most significant byte of the operand CIR. Figure 5-4 illustrates the operand CIR data alignment expected by the MC68881 when transferring data through the operand CIR.

5.1.2.9 REGISTER SELECT CIR ($14). This 16-bit read-only register is read by the main processor to transfer the register mask from the MC68881 during a move multiple floating-point data registers operation. The only time that an access to this register is legal is immediately following the issue of a transfer multiple coprocessor registers primitive to the main processor; at any other time, an access of this CIR location causes a protocol violation.

Although this is a 16-bit register, the MC68881 only utilizes the least significant eight bits; the most significant eight bits are always read as zeros. The least significant eight bits contain

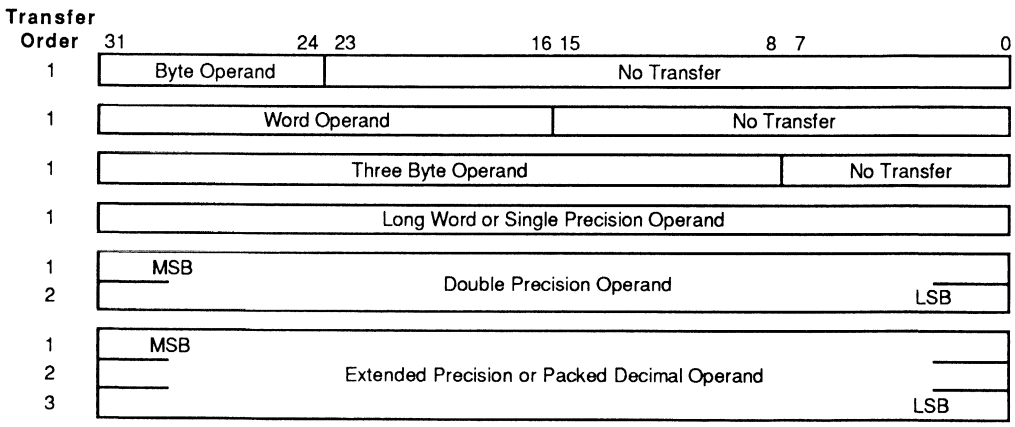


Figure 5-4. Operand CIR Data Alignment

the register mask for the multiple register transfer, with each bit set if the corresponding floating-point register is to be transferred. The main processor should not interpret the order of the bits in the register mask, but rather count the number of ones in the mask to determine the number of registers to transfer. Each MC68881 floating-point data register is 12 bytes long, and thus requires three long word transfers.

5.1.2.10 INSTRUCTION ADDRESS CIR ($18). This 32-bit write only register is used by the main processor to transfer the address of the MC68881 instruction being executed when the PC bit of any primitive is set. The MC68881 only sets the PC bit in the first primitive returned during the dialog for an instruction that can cause an exception, or a take pre-instruction exception primitive for a BSUN exception. The main processor may optionally transfer the program counter value to the instruction address CIR at that time or ignore the request (this is left to the discretion of the system designer in order to support exception handlers in the most efficient manner. The MC68020 always transfers the PC when needed).

Accesses to the instruction address CIR are neither expected or unexpected at any point in an instruction dialog; thus, an access to this CIR location never causes a protocol violation. A write to the instruction address CIR updates the FPIAR register in the MC68881 programming model, while a read always returns all ones.

5.1.2.11 OPERAND ADDRESS CIR ($1C). This 32-bit read/write register is used by the main processor to transfer an operand address in response to the evaluate and transfer effective address or take address and transfer data primitives. Since the MC68881 does not utilize either of these primitives, this CIR is not required for operation and is not implemented. An access to this CIR location does not cause a protocol violation; read cycles always return all ones, while write accesses are ignored.

5.1.3 Interprocessor Transfers

All interprocessor transfers are initiated by the MC68020. During the processing of a MC68881 instruction, the MC68020 transfers instruction information and data to the MC68881 via standard M68000 write bus cycles; and receives data, requests for service, and status information from the MC68881 via standard M68000 read bus cycles. A detailed description of the electrical characteristics of the MC68881 bus interface is contained in **SECTION 8 BUS OPERATION** and **SECTION 10 ELECTRICAL SPECIFICATIONS**.

5.2 COPROCESSOR INSTRUCTIONS

MC68881 instructions are from one to eight words in length. The first word of the instruction is called the operation word, and the second word of the instruction is called the coprocessor command word. Additional words specify the operands, and are either extensions to the effective addressing mode specified in the operation word, or immediate operands which are part of the instruction. The general format of an MC68881 instruction is illustrated in Figure 5-5.

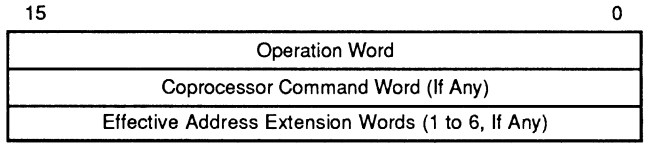


Figure 5-5. Coprocessor Instruction General Format

All coprocessor operations are based on the F-line operation codes (i.e., operation words with bits [15:12] = $F) which instruct the MC68020 to call upon a coprocessor for execution of the instruction. Figure 5-6 illustrates the format of this word.

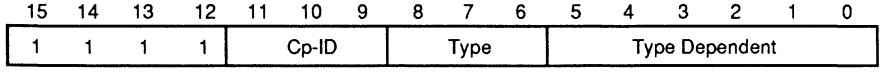


Figure 5-6. MC68881 Instruction Operation Word

The Cp-ID field indicates which coprocessor is to be selected. Cp-IDs of 000-101 are reserved by Motorola, and Cp-IDs 110 and 111 are reserved for user definition. The Motorola MC68020/MC68881 assembler defaults the Cp-ID for MC68881 instructions to 001. The type field indicates to the MC68020 which type of coprocessor operation is selected: general, branch, conditional, save, or restore. The type and type dependent fields and the coprocessor command word for all MC68881 instructions are described in **3.5 INSTRUCTION ENCODING DETAILS**.

5.2.1 Instruction Protocol

All MC68881 instructions have a typical protocol which the MC68020/MC68881 follows. This communication protocol is as follows:

1) When the MC68020 detects an F-line operation word, communication is initiated by writing information (a command, condition selector, or restore format word) to the appropriate MC68881 coprocessor interface register location. (The MC68881 save instruction is initiated by a read operation.)
2) The MC68020 then reads the coprocessor response to the previous write operation. The response may indicate any of the following:
   a) The MC68881 is busy. MC68020 will check for interrupts, process them if any are pending, and then query the coprocessor again. This allows the main processor and coprocessor to synchronize operation.
   b) An exception condition exists, and the MC68881 instructs the MC68020 to take an exception, using a specific exception vector. The MC68020 acknowledges the exception and initiates exception processing.
   c) There is an MC68881 service request; for example, to evaluate the effective address and transfer data to/from the effective address from/to the MC68881. The MC68881 may also request that the MC68020 query the coprocessor after the service is performed.
   d) The MC68020 is not needed for further processing of the coprocessor instruction. Communication is terminated and the MC68020 is free to begin execution of the next instruction. If the MC68020 is in the trace mode, the MC68020 does not take the trace exception until the MC68881 completes the processing of the coprocessor instruction.

Each MC68881 instruction type has specific requirements based upon this simplified protocol. The main processor service requests required for each MC68881 instruction are described in **3.5 INSTRUCTION ENCODING DETAILS.** All MC68881 main processor service requests (response primitives) are described in the following paragraphs. In addition, the dialog used by the MC68020 and the MC68881 during the execution of each instruction is detailed in **5.3 INSTRUCTION DIALOGS.**

5.2.2 Response Primitives

Data read from the MC68881 coprocessor interface response register is referred to as a primitive. Although the M68000 Family coprocessor interface defines 18 response primitives, the MC68881 only uses six of those primitives. For additional information on the complete set of response primitives and how they are serviced, refer to the *MC68020 32-Bit Microprocessor User's Manual.* The following paragraphs summarize all MC68881 response primitives and how they are used.

The M68000 coprocessor response primitives are encoded in a 16-bit word which is transferred to the main processor through the response CIR. Figure 5-7 illustrates the general format of a response primitive.

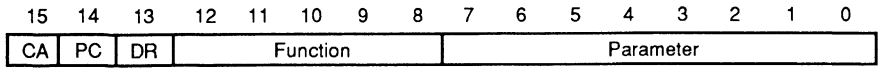


Figure 5-7. M68000 Coprocessor Response Primitive General Format

The encoding of bits [12-0] of a coprocessor response primitive is dependent on the individual primitive being implemented. Bits [15-13], however, are used to specify particular attributes of the response primitive which can be utilized in most of the primitives defined for the M68000 coprocessor interface.

Bit [15] in the primitive format, denoted by CA, is used to specify the come- again operation of the main processor. Whenever the main processor receives a response primitive from the MC68881 with the CA bit set to one, it should perform the service indicated by the primitive and then return to read the response CIR again.

Bit [14] in the primitive format, denoted by PC, is used to specify the pass program counter operation. If the main processor reads a primitive from the MC68881 that has the PC bit set, the main processor should immediately pass the current value of the program counter to the instruction address CIR as the first operation when servicing the primitive request. The value of the program counter passed from the main processor is usually the address of the operation word of the coprocessor instruction executing when the primitive is received (this is always the case if the main processor is the MC68020). The MC68881 always sets the PC bit in the first primitive of a general type instruction that might cause an exception (i.e., all of the arithmetic and move single floating-point data register instructions when exceptions are enabled), or the take pre-instruction exception primitive for a BSUN trap during a conditional instruction. By updating the FPIAR in this manner, the MC68881 can release the main processor for concurrent execution after all operands are fetched, and exception handlers can later locate an instruction that causes an exception. It should be noted that the PC bit is set in only one primitive response during any instruction dialog, and that the request to transfer the PC can be ignored without incurring a protocol violation.

Bit [13] in the primitive format, denoted by DR, is the direction bit; and is used in conjunction with operand transfers between the main processor and the MC68881. If DR is zero, the direction of the transfer is from the main processor to the MC68881 (a main processor write). If DR is one, the direction of the transfer is from the MC68881 to the main processor (a main processor read). If the operation indicated by a given response primitive does not involve an explicit operand transfer, the value of this bit is dependent on the particular primitive encoding.

NOTE

All primitives issued by the MC68881, with the exception of the null primitive, have the CA bit equal to one, such that the MC68020 checks the response CIR after any service is performed. This allows the MC68881 to assure correct internal operation, and to report exceptions immediately after a service is performed.

The following paragraphs detail the response primitive encodings used by the MC68881 and the expected main processor response to each one.

5.2.2.1 NULL PRIMITIVE. This primitive is used by the MC68881 to synchronize operation with the main processor and to allow concurrent execution by the main processor. The format of the null primitive is shown in Figure 5-8. In addition to the variable bits CA and PC that are discussed above, the null primitive uses three other variable bits to identify the required action to be taken by the main processor. Bit [8], denoted by IA, is used to specify that the main processor may process pending interrupts if necessary. Bit [1], denoted by PF, is used to indicate the status of the MC68881 during concurrent instruction execution; if the PF bit is zero, then the MC68881 is executing an instruction, otherwise it is idle. Bit [0], denoted by TF, is used to communicate the true or false of a conditional evaluation; if TF equals one, then the condition is true, otherwise it is false.

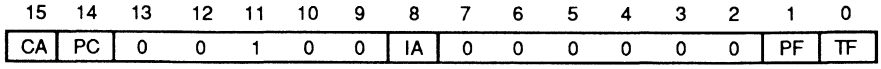


Figure 5-8. Null Primitive Format

As indicated by the format of this primitive, there are 32 possible null primitive encodings of which the MC68881 uses only seven. Table 5-3 lists the MC68881 null primitive encodings, and the circumstances in which they are used.

Table 5-3. Null Primitive Encodings

CA	PC	IA	PF	TF	Usage
0	0	0	0	x	Returned by the MC68881 in response to the write of a conditional predicate to the Condition CIR. The TF bit indicates the result of the conditional evaluation; TF=1 if the condition is true, TF=0 if the condition is false.
0	0	0	1	0	Returned when the MC68881 is in the idle state. The PF bit indicates that no instruction is being executed; thus, there is no expected response to this primitive.
0	0	1	0	0	Returned when the MC68881 enters the middle or end phase of an instruction to allow concurrent execution by the main processor. The CA bit indicates that no further service is required of the main processor, but the PF bit indicates that the MC68881 has not completed execution of the instruction. The IA bit indicates that if the main processor is in the trace mode, it may process interrupts while waiting for the MC68881 to complete execution of the instruction. Since this primitive does not request any specific service, there is no expected response from the main processor.
0	1	1	0	0	The same as the preceding response, except that the main processor is requested to pass the current program counter before proceeding with the next instruction. This response is returned only as the first response of a dialog. The transfer of the program counter is not required, and the PC bit is cleared by the read of this response, thus the PC bit can be ignored.
1	0	1	0	0	Returned when the MC68881 is executing an instruction and requires further service from the main processor before the next instruction can be executed. This response is also used when a new MC68881 instruction is initiated while a previous one is still being executed. The expected response is for the main processor to re-read the Response CIR (after servicing pending interrupts).
1	1	1	0	0	The same as the preceding response, except that the main processor is requested to pass the current program counter before processing any pending interrupts and re-reading the Response CIR. This response is returned only as the first response of a dialog. The transfer of the program counter is not required, and the PC bit is cleared by the read of this response, thus the PC bit can be ignored.

The meaning of the CA and PC bits are as described above. If IA equals one, then the main processor may process pending interrupts as part of the service for the null primitive, otherwise interrupts should be ignored. The IA bit is set to a one by the MC68881 for most null responses thus allowing the main processor to process pending interrupts anytime that it is "waiting" on the MC68881.

The PF bit is an indicator that reflects the processing state of the MC68881 during concurrent instruction execution. In normal operation, the PF bit is of no concern to the main processor. However, if the main processor is in the trace mode, it should wait until the MC68881 has completed execution of an instruction before taking the trace exception. By monitoring the PF bit in the null response primitive, the main processor can synchronize with the MC68881 in this manner. If PF equals zero, then the MC68881 is executing an instruction, otherwise it is idle.

The TF bit is utilized only for the conditional instructions. When the main processor writes a conditional predicate to the condition CIR, the MC68881 uses the null primitive to return the true or false result of the conditional evaluation. If TF equals one, then the condition is true; otherwise it is false. For all reads of the response CIR for other instruction types, the TF bit is a don't care.

5.2.2.2 EVALUATE EFFECTIVE ADDRESS AND TRANSFER DATA PRIMITIVE. This primitive is used by the MC68881 to request the transfer of a data item between the floating-point data and control registers and an external location (either memory or a main processor register). The format of this primitive is shown in Figure 5-9. The main processor services this request by evaluating the effective address indicated by the F-line word of the instruction and transferring the number of bytes indicated by the length field of the primitive to or from the operand CIR.

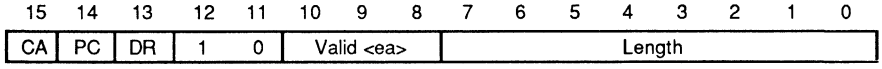


Figure 5-9. Evaluate Effective Address and Transfer Data Primitive Format

Note that the MC68881 returns this primitive only once during an instruction dialog. When this primitive is read from the response CIR, it is discarded by the MC68881 and the response encoding is changed to the null primitive. By doing this, the MC68881 avoids spurious service request primitives in systems where the MC68020 is not the main processor.

The meaning of the CA and PC bits are as described above. The DR bit indicates the direction of data transfer between the effective address location and the operand CIR of the coprocessor. If DR equals zero, the operand is transferred from the effective address location

to the coprocessor. If DR equals one, the operand is transferred from the coprocessor to the effective address location.

The effective address that is to be evaluated is specified in the F-line operation word, and any required extension words are fetched by the main processor, as needed. If the predecrement or postincrement addressing mode is used, the address register is decremented or incremented before or after the transfer by the size of the operand, as indicated in the length field.

The valid EA field specifies various classes of addressing modes with the encodings shown in Table 5-4. If the effective address in the operation word is not of the specified class, then the main processor should write an abort to the control CIR and take an F–line emulator trap. The addressing categories below are as defined for all M68000 Family processors.

Table 5-4. Coprocessor Valid Effective Address Codes

000	Control Alterable
001	Data Alterable
010	Memory Alterable
011	Alterable
100	Control
101	Data
110	Memory
111	Any Effective Address

The number of bytes transferred to or from an effective address location is indicated in the length field. If the effective address is a main processor register (register direct), then only lengths of one, two, or four bytes are used. If the effective addressing mode is immediate, the length is always one or even, and the transfer is effective address to coprocessor. If the effective address is a memory location, any length is legal (including odd). If the effective address mode is predecrement or postincrement, with A7 as the specified register and a length of one, the transfer causes the stack pointer to be decremented or incremented by two, in order to keep the stack aligned on a word boundary.

Table 5-5 lists the encodings of the evaluate effective address and transfer data primitive that are used by the MC68881 and the cases for which they are used.

Table 5-5. Evaluate Effective Address and Transfer Data Primitive Encoding

Usage	CA	PC	DR	Valid <ea>	Length
F<op> <ea>,FPn (OPCLASS 010)					
Issued as the first primitive of an instruction dialog to request the transfer of an operand from memory or a main processor data register to the MC68881. The length field indicates the size of the operand; byte, word, long or single, double, and extended.	1	*	0	101	1
	1	*	0	101	2
	1	*	0	101	4
	1	*	0	110	8
	1	*	0	110	12
FMOVE FPm,<ea> (OPCLASS 011)					
Issued after the conversion from the internal extended precision format to the destination format is completed to request the transfer of an operand from the MC68881 to memory or a main processor data register. The length field indicates the size of the operand; byte, word, long or single, double, and extended.	1	0	1	001	1
	1	0	1	001	2
	1	0	1	001	4
	1	0	1	010	8
	1	0	1	010	12
FMOVE <ea>,FPcr and FMOVEM <ea>,FPcr_list (OPCLASS 100)					
Issued as the first primitive of an instruction dialog to request the transfer of one or more control registers from memory or a main processor register to the MC68881. The length field indicates the total size of all control registers to be moved, 4 bytes per register.	1	0	0	111	4
	1	0	0	101	4
	1	0	0	110	8
	1	0	0	110	12
FMOVE FPcr,<ea> and FMOVEM FPcr_list,<ea> (OPCLASS 101)					
Issued as the first primitive of an instruction dialog to request the transfer of one or more control registers from the MC68881 to memory or a main processor register . The length field indicates the total size of all control registers to be moved, 4 bytes per register.	1	0	1	011	4
	1	0	1	001	4
	1	0	1	010	8
	1	0	1	010	12

*PC = 1 if any arithmetic exceptions are enabled; otherwise PC = 0.

5.2.2.3 TRANSFER SINGLE MAIN PROCESSOR REGISTER PRIMITIVE. This primitive is used by the MC68881 to request the transfer of one main processor register. The format of this primitive is shown in Figure 5-10. The main processor services this request by writing a long word to the operand CIR.

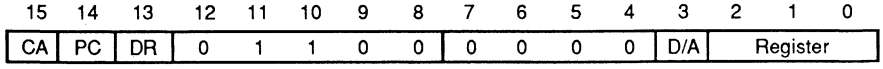


Figure 5-10. Transfer Single Main Processor Register Primitive Format

This primitive is only utiliized for the move multiple floating-point data register instruction when the register list is specified as dynamic. Therefore, when this primitive is issued by the MC68881 (to fetch the register list), CA is always set, DR, PC, and D/A are always clear (D/A indentifies the selected register as a data or address register; zero indicates it is a data

Note that the MC68881 returns this primitive only once during an instruction dialog. When this primitive is read from the response CIR, it is discarded by the MC68881 and the response encoding is changed to the null primitive until the request has been serviced. By doing this, the MC68881 avoids spurious service requests in systems where the MC68020 is not the main processor.

5.2.2.4 TRANSFER MULTIPLE COPROCESSOR REGISTERS PRIMITIVE. This primitive is used by the MC68881 to request the transfer of a list of floating-point data registers to or from memory. The format of this primitive is shown in Figure 5-11. The main processor services this request by performing an implied effective address evaluation, reading a register list from the register select CIR, and transferring the selected registers (where each register is the size indicated by the length field of the primitive) between the operand CIR and memory. The MC68020 uses long-word transfers whenever possible while servicing this primitive.

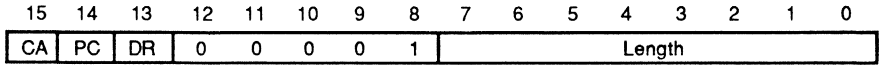


Figure 5-11. Transfer Multiple Coprocessor Registers Primitive Format

Note that the MC68881 returns this primitive only once during an instruction dialog. When this primitive is read from the response CIR, it is discarded by the MC68881 and the response encoding is changed to the null primitive until the request has been serviced. By doing this, the MC68881 avoids spurious service requests in systems where the MC68020 is not the main processor.

The meaning of the CA and PC bits are as described above. For this primitive, CA is always set, and PC is always clear (since the FMOVEM instruction cannot cause an exception). The DR bit indicates the direction of the transfer between the effective address location and the operand CIR. If DR equals zero, the listed registers are transferred from the effective address location to the MC68881. If DR equals one, the listed registers are transferred from the MC68881 to the effective address location.

The length field indicates the size, in bytes, of each register to be transferred; for the MC68881 the length is always 12 bytes. The register list that is read from the register select CIR is used by the main processor to determine the number of registers to be transferred (but not the order of the transfer). For each bit that is set in the 16-bit register list, one operand of the size indicated by the length field in transferred. Thus, the total number of bytes transferred in response to this primitive is the product of the length and the number of ones in the register list. Note that since the MC68881 has only eight floating-point data registers, the register list will always have zeros in the most significant byte.

The effective address that is evaluated by the main processor is specified in the F-line operation word, and any required extension words are fetched by the main processor, as needed. If the predecrement or postincrement addressing mode is used, the address

The effective address that is evaluated by the main processor is specified in the F-line operation word, and any required extension words are fetched by the main processor, as needed. If the predecrement or postincrement addressing mode is used, the address register is decremented or incremented (before or after the the transfer) by the size of the operand, as indicated by the length field. The effective addressing modes that are valid for this primitive are determined by the DR bit, and the mode is validated by the main processor. If DR equals zero, then the control and postincrement addressing modes are allowed. If DR equals one, then the control alterable and predecrement addressing modes are allowed. If the effective address field of the operation word is not valid for the selected multiple register transfer, the main processor writes an abort to the control CIR (before reading the register select CIR) and takes an F-line trap.

For the control and postincrement addressing modes, the registers are transferred using ascending addresses. For the postincrement addressing mode, the address register is incremented by the length value after each register is transferred. Thus, the final value of the address register is the initial value plus the total number of bytes transferred during the primitive execution.

For the predecrement addressing mode, the operands are written to memory with descending addresses, while the bytes within each operand are written to memory with ascending addresses. For example, Figure 5-12 illustrates the transfer of two floating-point data registers to a stack, using the -(An) addressing mode. The designated stack pointer is decremented by 12 bytes before the transfer of each register, then the bytes within each register are written to memory with ascending addresses. Thus, the address register is decremented by the total number of bytes transferred by the end of the primitive execution.

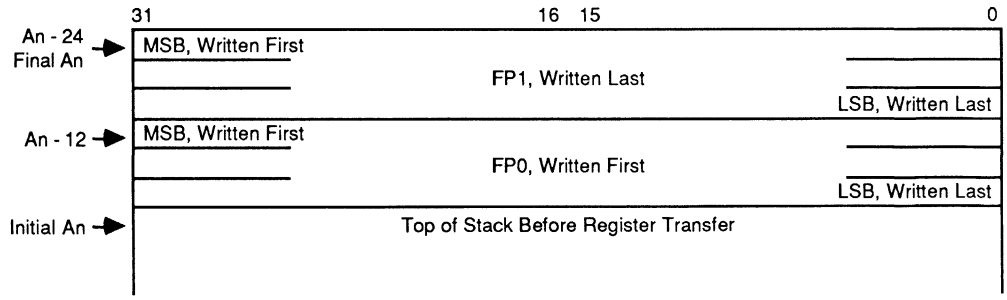


Figure 5-12. Transfer Multiple Floating-Point Data Register to Stack Example

5.2.2.5 TAKE EXCEPTION PRIMITIVES. These primitives are used by the MC68881 to instruct the main processor to abort the current operation and initiate exception processing. The main processor services these requests by writing an exception acknowledge to the control CIR (which clears the pending exception in the MC68881), creates the appropriate stack frame on the currently active supervisor stack, and begins execution of an exception

Table 5-6. MC68881 Vector Numbers

Vector Number (Dec.)	Vector Offset (Hex.)	Assignment
11	$02C	F-Line Emulator
13	$034	Coprocessor Protocol Violation
48	$0C0	Branch or Set on Unordered Condition
49	$0C4	Inexact Result
50	$0C8	Floating-Point Divide by Zero
51	$0CC	Underflow
52	$0D0	Operand Error
53	$0D4	Overflow
54	$0D8	Signalling NAN

Note that the MC68881 returns one of these primitives only once during the instruction dialog. When an exception acknowledge is written to the control CIR, the take exception primitive is discarded by the MC68881, and the response encoding is changed to the null primitive. By doing this, the MC68881 assures that the take exception request is received by the main processor, but avoids spurious service request primitives in systems where the MC68020 is not the main processor.

While the M68000 coprocessor interface defines three take-exception primitives, the MC68881 utilizes only two of them. The following paragraphs describe the two take-exception primitive that are used by the MC68881.

5.2.2.5.1 Take Pre-Instruction Exception Primitive. This primitive is used by the MC68881 when an arithmetic (OPCLASS 000, 010, and 011) or conditional instruction is initiated and an exception is pending from a previously executed, concurrent instruction. This primtive is also returned if an illegal command word is written to the command CIR or if a protocol violation occurs. Finally, this primitive is issued when a conditional instruction is executed that utilizes one of the IEEE non-aware conditional predicates, and the NAN bit in the FPSR condition code byte is set. The format of this primitive is shown in Figure 5-13.

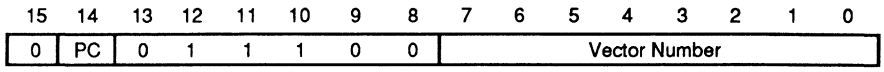


Figure 5-13. Take Pre-Instruction Exception Primitive Format

The CA bit is always zero for this primitive, since there is an implied protocol preemption in this service request. The PC bit is zero if the exception is preempting the execution of a new MC68881 instruction. The PC bit is one if the exception is due to an illegal command word, or if it is reported during the execution of a conditional instruction, in lieu of the true/false

result of the conditional evaluation. The vector number identifies the type of the exception and is used by the main processor to locate the exception handler routine.

In response to this primitive, the MC68020 creates a four word stack frame on top of the currently active supervisor stack. The format of this stack frame is shown in Figure 5-14. The value of the program counter in the stack frame is the address of the F-line operation word of the MC68881 instruction that was preempted by the exception (i.e., the arithmetic or conditional instruction attempted in the case of an exception pending from a previous instruction or an F-line exception, or the conditional instruction in the case of a BSUN exception). Thus, if no modifications are made to the stack frame within the exception handler, an RTE instruction causes the MC68020 to return and re-initiate the instruction that was being attempted when the primitive was received. Refer to the *MC68020 32-Bit Microprocessor User's Manual* for further details on exception handling by the MC68020.

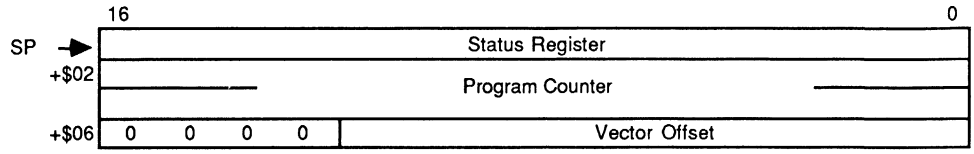


Figure 5-14. Pre-Instruction Exception Stack Frame

5.2.2.5.2 Take Mid-Instruction Exception Primitive. This primitive is used by the MC68881 when an exception occurs during the execution of an FMOVE FPm,<ea> instruction. The format of this primitive is shown in Figure 5-15.

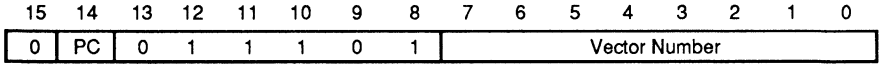


Figure 5-15. Take Mid-Instruction Exception Primitive Format

The CA bit is is always zero for this primitive, since there is an implied protocol preemption in this service request. The PC bit is always zero, since a null primitive earlier in the dialog for the move-out instruction is used to request the program counter transfer. The vector number identifies the type of the exception, and is used by the main processor to locate the exception handler routine.

In response to this primitive, the MC68020 creates a ten-word stack frame on top of the currently active supervisor stack. The format of this stack frame is shown in Figure 5–16. The ScanPC value is the address of the instruction immediately following the FMOVE instruction. The value of the program counter in the stack frame is the address of the F-line operation word of the MC68881 instruction that caused the exception. The operation word image contains the F-line word of the FMOVE instruction. The effective address value is the memory address of the destination operand. Note that the take mid-instruction exception

primitive is used in this case solely for the purpose of placing the evaluated effective address in the stack frame, such that an exception handler is not required to recalculate it.

If no modifications are made to the stack frame within the exception handler, an RTE instruction causes the MC68020 to return to read the response CIR, which will contain the null (CA = 0, PF = 0) primitive. Thus, the main processor is free to begin execution of the instruction following the FMOVE upon return.

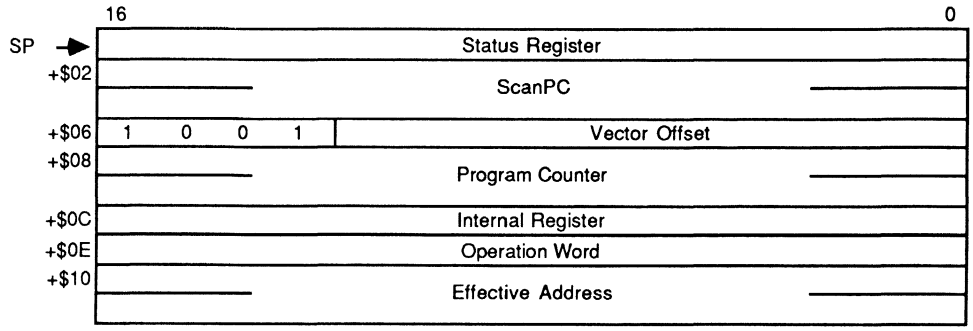


Figure 5-16. Mid-Instruction Stack Frame

5.2.2.6 RESPONSE PRIMITIVE SUMMARY. Table 5-7 lists a summary of all primitive responses utilized by the MC68881 in numeric order.

5.3 INSTRUCTION DIALOGS

The following paragraphs detail the coprocessor communications dialogs that are executed by the MC68881 and MC68020 during each floating-point instruction. In this discussion, a dialog refers to the sequence of command and data transfers to the MC68881, and the service request primitives that are returned to control that sequence. Although the following discussion assumes that the main processor is an MC68020, information is also presented that may be used by designers of systems that utiltize a different main processor.

The diagrams presented in the following paragraphs represent the activity of the MC68020 and the MC68881 during the execution of a floating-point instruction. In these diagrams, boxes are used to depict periods of time during which a device is actively participating in the execution of an instruction; while the absence of a box during a period indicates that a device is waiting on the other one to complete an operation, or that concurrent execution of unrelated instructions may take place.

Table 5-7. MC68881 Primitive Responses (Sheet 1 of 2)

Primitive Value	Primitive Type	Comments
$0800 $0801 $0802 $0900	Null	CA=0, PC=0, IA=0, PF=0, TF=0 CA=0, PC=0, IA=0, PF=0, TF=1 CA=0, PC=0, IA=0, PF=1, TF=0 CA=0, PC=0, IA=1, PF=0, TF=0
$1C31 $1C32 $1C33 $1C34 $1C35 $1C36	Take Pre-Instruction Exception PC=0	Inexact Result Floating-Point Divide By Zero Underflow Operand Error Overflow Signalling NAN
$1D0D $1D31 $1D33 $1D34 $1D35 $1D36	Take Mid-Instruction Exception PC=0	Coprocessor Protocol Violation Inexact Result Underflow Operand Error Overflow Signalling NAN
$4900	Null	CA=0, PC=1, IA=1, PF=0, TF=0
$5C0B $5C30	Take Pre-Instruction Exception PC=1	F-Line Emulator Branch Or Set On Unordered
$810C	Transfer Multiple Coprocessor Registers CA=1, PC=0, DR=0 (Memory to MC68881)	
$8900	Null	CA=1, PC=0, IA=1, PF=0, TF=0
$8C00 $8C01 $8C02 $8C03 $8C04 $8C05 $8C06 $8C07	Transfer Single Main Processor Register CA=1, PC=0, DR=0 (Main Processor to MC68881)	D0 D1 D2 D3 D4 D5 D6 D7
$9501 $9502 $9504 $9608 $960C $9704	Evaluate <ea> and Transfer Data CA=1, PC=0, DR=0 (External to MC68881)	Byte Word Long, Single, FPCR or FPSR Double or Two FPcr's (Memory Only) Extended, Packed or Three FPcr's (Memory Only) FPIAR
$A10C	Transfer Multiple Coprocessor Registers CA=1, PC=0, DR=1 (MC68881 to Memory)	

Table 5-7. MC68881 Primitive Responses (Sheet 2 of 2)

Primitive Value	Primitive Type	Comments
$B101 $B102 $B104 $B208 $B20C $B304	Evaluate <ea> and Transfer Data CA=1, PC=0, DR=1 (MC68881 to External)	Byte Word Long, Single, FPCR or FPSR Double or Two FPcr's (Memory Only) Extended, Packed or Three FPcr's (Memory Only) FPIAR
$C900	Null	CA=1, PC=1, IA=1, PF=0, TF=0
$D501 $D502 $D504 $D608 $D60C	Evaluate <ea> and Transfer Data CA=1, PC=1, DR=0 (External to MC68881)	Byte Word Long or Single Double (Memory Only) Extended or Packed (Memory Only)

Each box in the following diagrams is labled to indicate the activity depicted by that box. The lables above the boxes identify the actions taken by the main processor, while the lables below the boxes identify the encoding of the response CIR at any time during a dialog. When a response CIR encoding is indicated, that encoding will be received by the main processor any time that the response CIR is read until the next primitive encoding is indicated.

In all of the following paragraphs, the following assumptions are made:

1) Before the start of an instruction dialog, except for the FSAVE and FRESTORE instructions, the MC68881 is in the idle state.
2) The MC68020 and the MC68881 communicate via a 32-bit data bus.
3) The memory width is 32 bits, and all memory operands are long-word aligned.

Also, for periods during which the MC68020 is forced to wait on the MC68881, (i.e., during move to memory operation, or if the MC68020 is in the trace mode) only one of the response CIR reads is explicitly indicated. In actual operation, numerous reads of the response CIR may occur in these cases. Similary, if the MC68881 is not idle before the initiation of a new instruction, multiple reads of the null (CA = 1, IA = 1; $8900) primitive may occur after the command or condition CIR write, and before the read of the first primitive shown in a diagram.

5.3.1 General Instructions

This group of instructions includes all of the arithmetic instructions, the move system control register instructions, and the move multiple floating-point register instructions. The common factor between these instructions is the format of the F-line operation word, which uses the cpGEN format of the M68000 Family coprocessor instruction set. Thus, the initial phase of the communications dialog for these instructions is identical, with the MC68020 writing the

command word to the MC68881 and then relying on the MC68881 to control the remainder of the dialog through the use of the coprocessor interface response primitive set. The following paragraphs discuss the five different protocols that are used by the MC68881 for this group of instructions.

For each of the general instruction dialogs, with the exception of the register-to-register dialog, there is an important consideration for systems that do not utilize the MC68020 as the main processor. This consideration is that the come-again request in any evaluate effective address and transfer data primitive or transfer multiple coprocessor registers primitive should not be ignored. The MC68881 sets the CA bit in these primitives to assure correct operation regardless of the frequency relationship between the MC68881 clock and the main processor clock. By requiring the main processor to perform a read of the response CIR (which is a cycle that is synchronous with the MC68881 CLK signal) after the last operand CIR access, and before continuing with the next instruction, the MC68881 assures that the operand transfer is completed internally before the main processor can initiate the next instruction by writing the command or condition CIRs. This sequence assures that spurious protocol violations (detected by the MC68881) do not occur in systems where the main processor clock frequency is much faster than the MC68881 clock frequency.

5.3.1.1 REGISTER-TO-REGISTER (OPCLASS 000). This dialog is utilized for all of the arithmetic and move instructions that use floating-point data registers for both the source and destination operands, and the FMOVECR instruction. Since the MC68881 contains both operands when such an instruction is initiated, no external data references are required before the calculation can be performed. Thus, after the MC68020 has written the command word to the MC68881, it is released to execute the next instruction. If any arithmetic exceptions are enabled, the MC68881 requests the transfer of the program counter; however, this request can be ignored, and the program counter write cycle does not affect instruction execution time (since it occurs concurrently with the MC68881 instruction execution).

The dialog for this instuction type is shown in Figure 5-17. Also shown in this figure is the key for all of the dialog figures presented in subsequent paragraphs.

5.3.1.2 EXTERNAL-TO-REGISTER (OPCLASS 010). This dialog is utilized for all of the arithmetic and move instructions that reference memory or a main processor register for the source operand. The dialog for this instruction type is shown in Figure 5-18. Since the MC68881 does not contain both operands when such an instruction is initiated, external data references are required before the calculation can be performed. The MC68881 requests the fetch of the required external operand with the first primitive of the dialog. The second primitive of the dialog is then used to release the main processor to execute the next instruction (once the operand transfer is completed). Note that the read of the first primitive causes the response CIR encoding to be changed to the null primitive, thus avoiding spurious request primitives in non-MC68020 based systems.

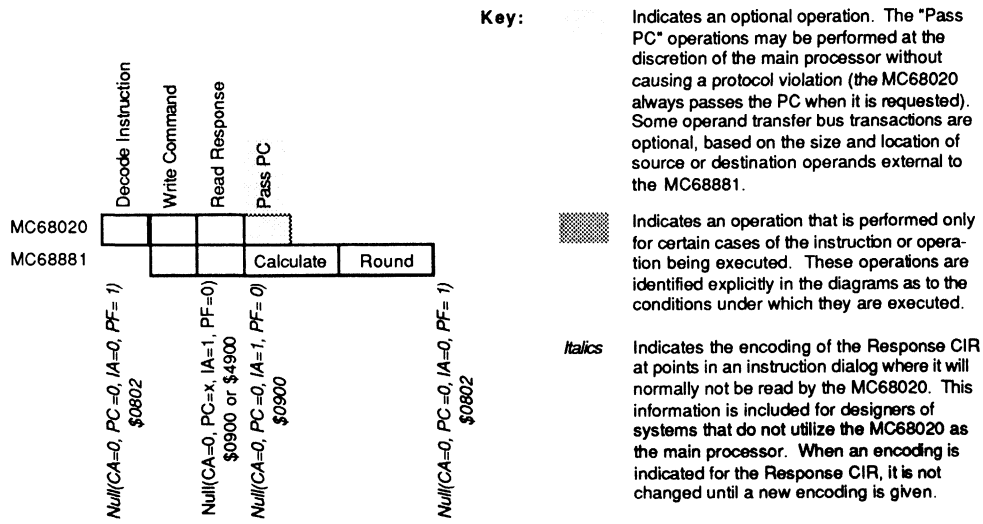


Figure 5-17. Register-to-Register Instruction Dialog

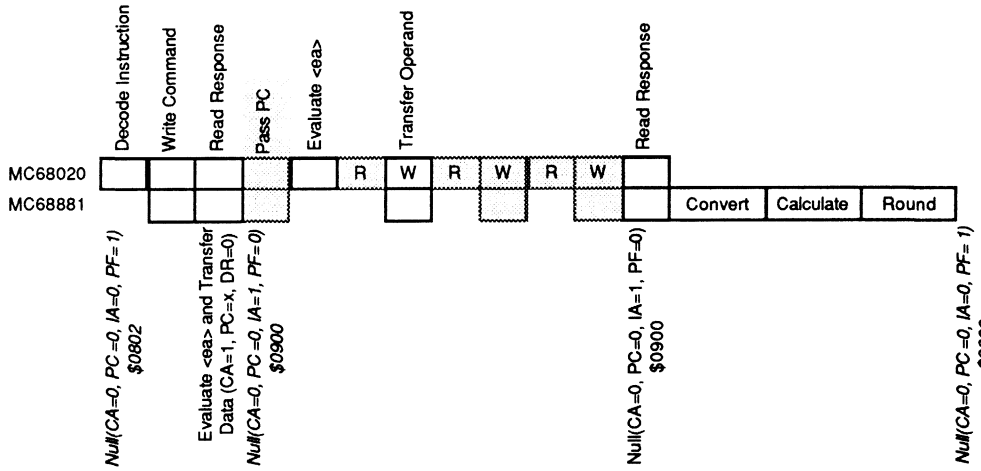


Figure 5-18. External-to-Register Instruction Dialog

If any arithmetic exceptions are enabled, the MC68881 requests the transfer of the program counter with the first primitive. However, this request can be ignored, and the program counter write cycle does not affect instruction execution time (since it occurs concurrently with the MC68020 effective address calculation).

The operation boxes that are marked "R" and "W" indicate an operand read or write cycle, respectively, by the MC68020. Those operand transfer boxes that are shaded are optionally executed, depending on the size and location of the source operand. For example, none of the shaded boxes are executed for source operands that reside in the MC68020 registers. Also, note that the FMOVECR instruction, while it is an opclass 010 instruction, uses the register-to-register protocol described above.

5.3.1.3 REGISTER-TO-EXTERNAL (OPCLASS 011). This dialog is utilized only for the move from floating-point data register instruction. The dialog for this instruction type is shown in Figure 5-19. The first primitive returned depends on the destination data format; since additional format information is required to generate a packed decimal destination operand when a dynamic k-factor is specified. If the destination data type is packed decimal and a dynamic k-factor is used, the first response is the transfer single main processor register primitive (to transfer the contents of the register containing the k-factor). For all other destination data formats, the first request is the null(CA=1) primitive, since a conversion from the internal data format to the desired destination format must be performed before any further action is required of the main processor. For the dynamic k-factor case, the conversion will start after the main processor register transfer is completed; while the conversion starts immediately after the first read of the response CIR for all other destination operand formats. The main processor is allowed to service pending interrupts while it is waiting for the conversion to complete.

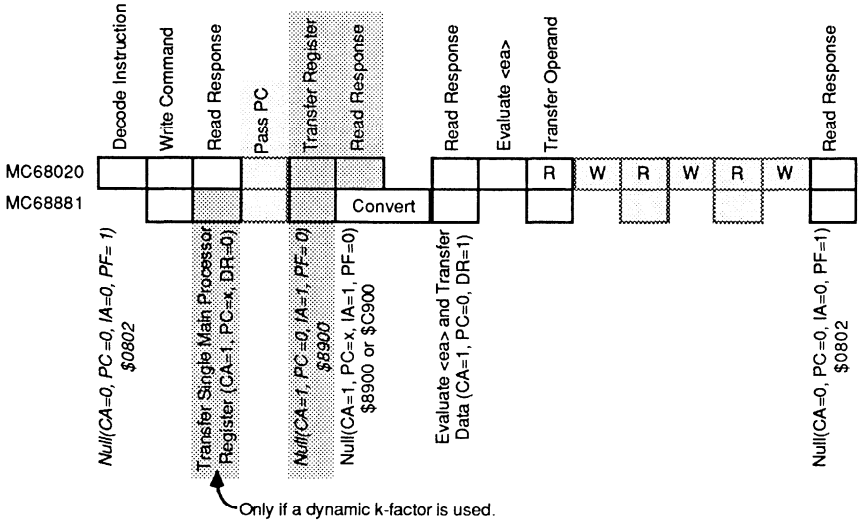


Figure 5-19. Register-to-External Instruction Dialog

If any arithmetic exceptions are enabled, the first primitive will request the transfer of the program counter. However, this request can be ignored, and the program counter write cycle may not affect instruction execution time (since it can occur concurrently with the operand conversion if the destination format is not packed decimal with a dynamic k-factor). Only the first primitive will request the transfer of the program counter; thus, if the transfer single main processor register primitive is issued first, the PC bit will not be set in any subsequent null primitive. If the first primitive is the null primitive, the PC bit is set if the program counter transfer is required.

The optional pass program counter operation is requested in one of two of the primitive encodings (shown in Figure 5–19 with the notation "PC=x"). For the packed decimal with a dynamic k-factor case, the dark shaded operations are always performed, with the PC bit set if necessary. The MC68020 services the transfer single main processor primitive with the PC bit set by first transferring the program counter and then transferring the requested register. For all other destination data formats, the dark shaded operations are not performed, and the box labeled "Convert" is 'folded under' the first box labeled "Read Response". For these cases, the null primitive may request the transfer of the PC, and that transfer occurs concurrently with the operand conversion by the MC68881.

When the operand conversion is completed, the next read of the response CIR returns the evaluate effective address and transfer data primitive. The main processor then reads the conversion result from the operand CIR and writes it to the appropriate destination location. Note that the read of the evaluate effective address and transfer data primitive causes the response CIR encoding to be changed to the null primitive, thus avoiding spurious request primitives in non-MC68020 based systems.

The operation boxes that are marked "R" or "W" indicate an operand read or write cycle, respectively, by the MC68020. Those operand transfer boxes that are lightly shaded are optionally executed, depending on the size and location of the source operand. For example, none of the shaded boxes are executed for destination operands that reside in the MC68020 registers.

5.3.1.4 MOVE CONTROL REGISTERS (OPCLASS 100 and 101). This dialog is utilized for the move single or multiple floating-point system control registers instructions. The dialog for this instruction type is shown in Figure 5-20. The first primitive of the dialog requests that the main processor evaluate the effective address and transfer the appropriate number of bytes to or from the operand CIR. The read of the first primitive causes the response CIR encoding to be changed to the null primitive, thus avoiding spurious request primitives in non-MC68020 based systems. When the transfer data primitive service is complete, the main processor is released to begin execution of the next instruction. Note that since this instruction type cannot cause an exection, the PC bit is not set in any primitive; thus, these instructions can be used to read or write the control registers without overwriting the FPIAR contents.

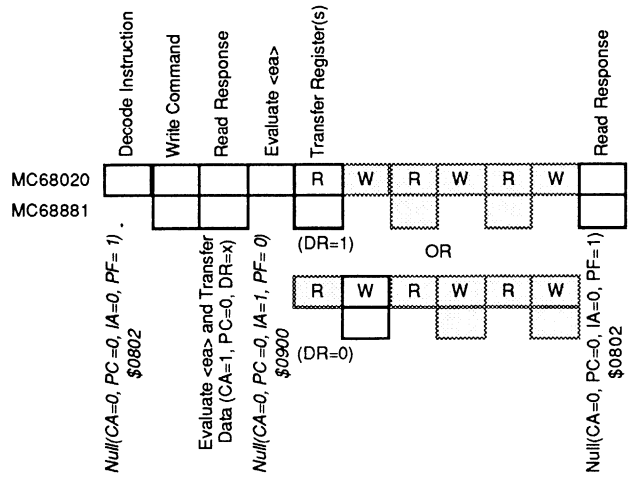


Figure 5-20. Move Control Register Instruction Dialog

The operation boxes that are marked "R" or "W" indicate an operand read or write cycle, respectively, by the MC68020. Those operand transfer boxes that are shaded are optionally executed, depending on the size and location of the source operand. For example none of the shaded boxes are executed for source operands that reside in the MC68020 registers.

5.3.1.5 MOVE MULTIPLE FPn (OPCLASS 110 and 111). This dialog is utilized for the move multiple floating-point data registers instruction. The dialog for this instruction type is shown in Figure 5-21. The first primitive of the dialog depends on the type of register list specified by the instruction. If the static register list form of the instruction is used, the first service request issued is the transfer multiple coprocessor registers primitive. For the dynamic register list form, the first primitive requests the transfer of the main processor data register that contains the register mask, and then the transfer multiple coprocessor registers primitive is issued. In response to the transfer multiple coprocessor registers primitive, the main processor reads the register list from the register select CIR and transfers one register for each bit that is set in the list (note that the register list can be equal to zero, in which case no register transfer occurs).

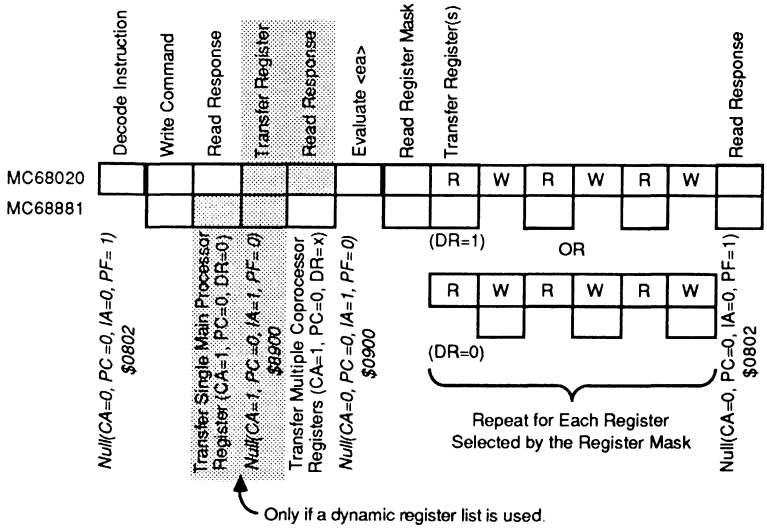


Figure 5-21. Move Multiple Floating-Point Data Registers Instruction Dialog

The read of the transfer single main processor register and transfer multiple coprocessor registers primitives cause the response CIR encoding to be changed to the null primitive, thus avoiding spurious request primitives in non-MC68020 based systems. If the transfer single main processor register primitive is issued, the transfer multiple coprocessor register primitive is not issued until the first service request is completed. When the transfer multiple coprocessor register primitive service is complete, the main processor is released to begin execution of the next instruction. Note that since this instruction type cannot cause an exception, the PC bit is not set in any primitive; thus, these instrucution can be used to read the floating- point data registers without overwriting the FPIAR contents.

The operation boxes that are marked "R" and "W" indicate an operand read or write cycle, respectively, by the MC68020.

5.3.2 Conditional Instructions

This group of instructions includes the FBcc, FDBcc, FNOP, FScc, and FTRAPcc instructions. The common factor between these instructions is that the execution of each one is inherent in the M68000 Family coprocessor instruction set definition, and the dialog used for all of them is the same. The dialog consists of only one write cycle and one read cycle; the main processor writes the conditional predicate to the MC68881 and then reads the response CIR to receive the result of the evaluation. The MC68881 always responds immediately with a

true or false result, and the main processor then proceeds with the appropriate conditional action. This dialog is shown in Figure 5-22.

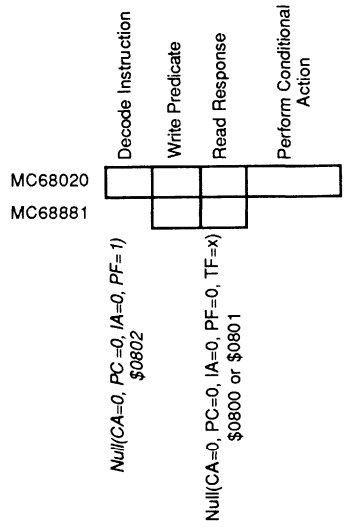


Figure 5-22. Conditional Instruction Dialog

5.3.3 Context Switch Instructions

This group of instructions includes the FSAVE and FRESTORE instructions. The common factor between these instructions is that the execution of each one is inherent in the M68000 Family coprocessor instruction set definition, and the coprocessor does not control the dialog in the flexible manner available with the general and conditional instruction types. The dialog consists of the save or restore command, followed by the transfer of the appropriate state frame. The only control that the MC68881 has over this dialog is for the FSAVE instruction, in which case it may request that the main processor delay the save operation until the MC688881 is ready to perform it. These dialogs are discussed in the following paragraphs.

5.3.3.1 FSAVE. This dialog is utilized for the context save instruction. The dialog for this instruction is shown in Figure 5-23. There are no primitive responses during this dialog; but rather, the MC68881 controls the frame transfer to a limited extent through the use of the format word encoding.

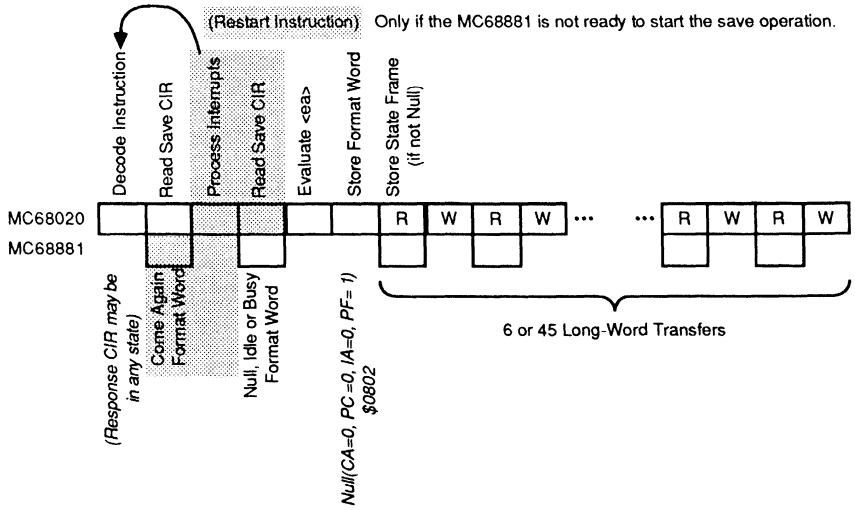


Figure 5-23. FSAVE Instruction Dialog

The main processor initiates this dialog by reading from the save CIR. During this read cycle, the MC68881 returns a format word that indicates the current state of the machine. For most cases of this dialog, the first read of the save CIR returns the idle format word, and the main processor then proceeds to transfer six long words from the operand CIR to memory. Optionally, the first primitive may be a null or busy format word, in which case, no state frame is transferred or 45 long words are transferred, respectively. Finally the save CIR read may return the not-ready, come-again format word. In this case, the main processor may process pending interrupts and restart the instruction, or simply re-read the save CIR, until a different format word is received. The invalid format word may also be returned, as discussed in **5.3.4.6 FORMAT EXCEPTION, FSAVE INSTRUCTION.**

After the MC68020 receives a valid format word, it then evaluates the effective address and writes the format word to that address. The appropriate state frame is then transferred to the effective address, and the main processor is free to proceed with the execution of the next instruction. Note that by using this sequence, the MC68020 can take a pre-instruction exception in response to a pending interrupt (if a not-ready, come-again format word is received) and then return to restart the instruction rather than taking a mid-instruction exception.

Note that after the state save operation is complete, the MC68881 is in the idle state with no pending exceptions.

5.3.3.2 FRESTORE. This dialog is utilized for the context restore instruction. The dialog for this instruction is shown in Figure 5-24. There are no primitive responses during this dialog; but rather, the MC68881 controls the frame transfer to a limited extent through the use of the format word encoding.

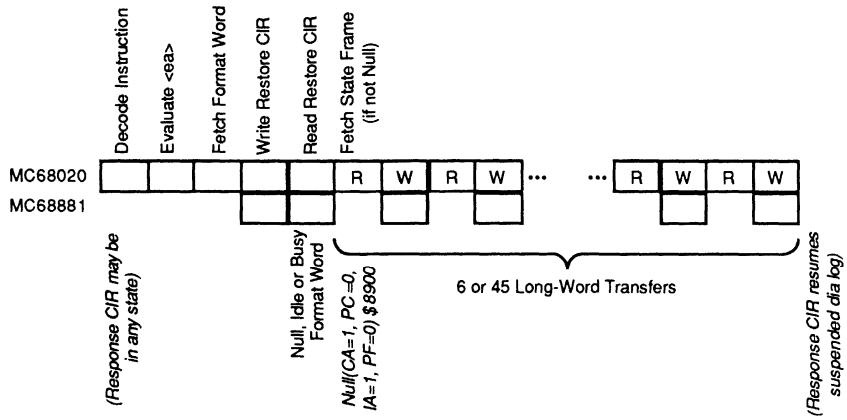


Figure 5-24. FRESTORE Instruction Dialog

The main processor initiates this dialog by evaluating the effective address, fetching a format word from that address, and writing the format word to the restore CIR. The main processor then reads the restore CIR to verify that the format word is valid. During this read cycle, the MC68881 returns a format word that indicates if the format word that was written is valid for the current revision of the device. If the format word is valid, the same value that was written is read back from the restore CIR, and the main processor proceeds to transfer the state frame appropriate for the format word. The state frame size is 0, 6, or 45 long words for the current implementation of the MC68881. The invalid format word may also be returned, as discussed in **5.3.4.7 FORMAT EXCEPTION, FRESTORE INSTRUCTION**.

Note that after the state restore operation is complete, the MC68881 is in the state of the instruction that was previously suspended with an FSAVE instruction.

5.3.4 Exception Processing

This group of dialogs is actually a set of special cases of the dialogs described previously; they are grouped here for quick reference, and to simplify the preceding discussions. For each of the exception processing dialogs, only the differences from the normal instruction dialogs shown above are discussed here. Also, it should be noted that these dialogs do not include all exception processing sequences that involve the MC68881; they only include those exceptions that are directly related to the coprocessor interface operation. For

example, main processor detected F-line exceptions are not included, since no coprocessor interface dialog occurs as part of the exception processing for this type of an exception.

Also not included in the diagrams below is the dialog for the coprocessor protocol violation exception. This is due to the fact that these exceptions are not expected to occur during the normal operation of a fully debugged system, and that the dialog is not readily predicted (either before or after the protocol violation occurs). For main processor detected protocol violations, the cause of the exception is by definition a hardware failure (since the MC68881 **can not** return an illegal response primitive).

For MC68881 detected protocol violations, the cause is most likely a software failure that causes a new instruction to be initiated before the previous instruction dialog is completed. In this case, the new instruction dialog is aborted immediately, but the previous instruction dialog may not terminate for some time (the previous dialog may be completed incorrectly, since the protocol violation is never reported to the previous instruction).

5.3.4.1 TAKE PRE-INSTRUCTION EXCEPTION. This dialog is utilized by the MC68881 when an arithmetic (OPCLASS 000, 010, or 011) or conditional instruction is initiated and an arithmetic exception is pending from a previous instruction, or when the main processor writes an undefined, reserved command word to the command CIR. In either case, this dialog consists of two write cycles and one read cycle, as shown in Figure 5-25. First, the main processor attempts to initiate a new instruction by writing to the command CIR; it then reads the response CIR to determine the next required action. The MC68881 returns the take pre-instruction exception in this case, indicating the appropriate vector number. The main processor services this primitive by writing an exception acknowledge to the control CIR and initiating exception processing.

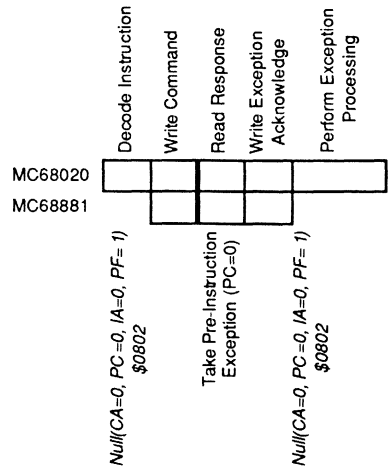


Figure 5-25. Take Pre-Instruction Exception Dialog

Note that the write of the exception acknowledge causes the response CIR encoding to be changed to the null primitive, thus assuring that the take exception primitive is received by the main processor while avoiding spurious request primitives in non-MC68020 based systems.

5.3.4.2 TAKE MID-INSTRUCTION EXCEPTION. This dialog is utilized by the MC68881 only if an exception occurs during the execution of the FMOVE FPm,<ea> instruction. In this case, the protocol for the normal execution of the instruction is followed; and then the mid-instruction exception is reported with the last primitive (in lieu of the null primitive normally used to terminate the dialog). The main processor services this primitive by writing an exception processing acknowledge to the control CIR and initiating exception processing.

The dialog for this operation is shown in Figure 5-26. (For simplicity, this diagram assumes that the destination data format is not packed decimal with a dynamic k-factor.) Note that a write of the exception acknowledge causes the response CIR encoding to be changed to the null primitive, thus assuring that the take-exception primitive is received by the main processor while avoiding spurious request primitive in non-MC68020 based systems.

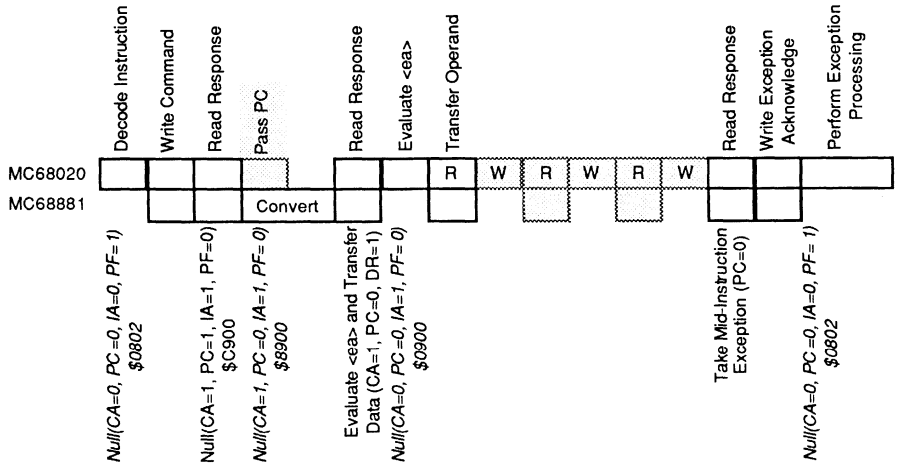
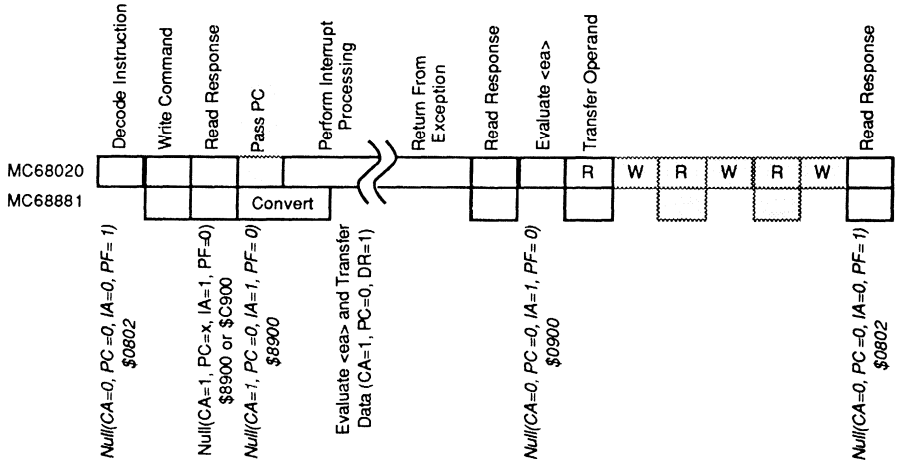


Figure 5-26. Take Mid-Instruction Exception Dialog

5.3.4.3 MID-INSTRUCTION INTERRUPT. This dialog is utilized by the MC68881 only if an interrupt is pending during the calculation phase of the FMOVE FPm,<ea> instruction. In this case, the protocol for the normal execution of the instruction is followed, except that the main processor performs exception processing for the interrupt, executes the interrupt handler, and returns to the point where the dialog was suspended in the middle of the execution of the instruction by the MC68881. From the perspective of the MC68881, the dialog appears to follow the normal sequence, with a long delay between successive reads of the response CIR by the main processor.

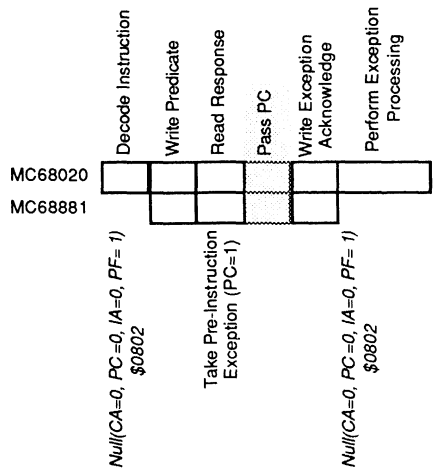
The dialog for this operation is shown in Figure 5-27. (For simplicity, this diagram assumes that the destination data format is not packed decimal with a dynamic k-factor.) Note that it is possible (and indeed probable) that the conversion of the output operand will be completed before the main processor returns from the interrupt. Thus, the response CIR is prepared to return the evaluate effective address and transfer data primitive as soon as the main processor returns. Since the read of this primitive causes the MC68881 to discard it and change the response CIR encoding to the null primitive, the interrupt handler (or any other routine) **must not** casually read the reponse CIR to determine the status of the MC68881, or the suspended protocol will be disrupted. Rather, the only valid method for checking the status of the MC68881 is to execute the FSAVE instruction and examine the state frame that is generated; followed by an FRESTORE instruction to reinstate the previous context of the MC68881.



(A similar sequence is followed during the initial phase of an FSAVE instruction, as indicated in the FSAVE protocol diagram.)

Figure 5-27. Mid-Instruction Interrupt Dialog

5.3.4.4 TAKE BSUN EXCEPTION. This dialog is utilized by the MC68881 when a conditional instruction is initiated by writing one of the IEEE non-aware conditional predicates to the condition CIR, the SNAN enable bit is set, and the NAN condition code bit is set. In this case, this dialog consists of three write cycles and one read cycle, as shown in Figure 5-28.



5

Figure 5-28. Take BSUN Exception Dialog

When the main processor reads the response CIR to receive the true/false result of the conditional evaluation, the MC68881 returns the take pre-instruction exception primitive instead of the null primitive. In order to update the FPIAR, the PC bit of this primitive is also set. The main processor services this primitive by transferring the program counter, writing an exception acknowledge to the control CIR, and then initiating exception processing. Although the MC68020 always performs the program counter transfer when it is requested, other main processors may choose to ignore this request without incurring a protocol violation.

Note that the write of the exception acknowledge causes the response CIR encoding to be changed to the null primitive, thus assuring that the take exception primitive is received by the main processor while avoiding spurious request primitives in non-MC68020 based systems.

5.3.4.5 F-LINE EMULATOR EXCEPTION. This dialog is utilized by the MC68881 when a general instruction is initiated and the value written to the command CIR is not a legal MC68881 command word encoding. In this case, this dialog consists of two write cycles and one read cycle, as shown in Figure 5-29. First the main processor attempts to initiate a new instruction by writing to the command CIR; it then reads the response CIR to determine the appropriate action to be taken. In this case, the first read of the response CIR returns a take exception primitive with the F-line emulator vector number. The main processor services this primitive by writing an exception acknowledge to the control CIR and initiating exception processing.

Note that the write of the exception acknowlege causes the response CIR encoding to be changed to the null primitive while avoiding spurious request primitive in non-MC68020 based systems.

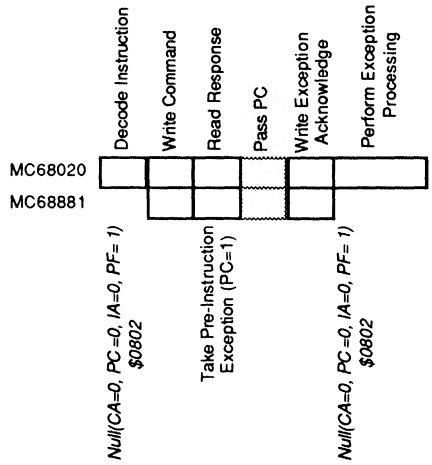
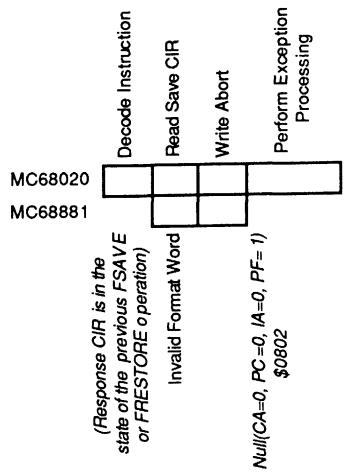


Figure 5-29. Take F-Line Emulator Exception Dialog

5.3.4.6 FORMAT EXCEPTION, FSAVE INSTRUCTION. This dialog is utilized by the MC68881 when an FSAVE or FRESTORE instruction dialog is interrupted by an attempt to initiate a new FSAVE instruction (by reading from the save CIR). In this case, the MC68881 returns the invalid format word to signal the illegal nesting of the FSAVE instruction. The main processor services this format word by writing an abort to the control CIR and initiating exception processing. The dialog for this operation is shown in Figure 5-30.

Since the MC68020 writes an abort to the MC68881 in response to the illegal format word, the FSAVE or FRESTORE that was interrupted by the nested FSAVE is destructively aborted, with no indication to the suspended instruction that this has occurred. Thus, a suspended save operation continues to read the "frame" from the operand CIR if it is resumed; even though the data in the operand CIR is not valid. Likewise, a suspended restore operation writes the remainder of the frame to the operand CIR if it is resumed; even though the data written is ignored and the restore operation is not performed. Due to the destructive behaviour of a nested FSAVE instruction, programmers must be certain that the MC68881 is not executing an FSAVE or FRESTORE instruction prior to an attempt to execute a new FSAVE instruction. If there is a possibility that a nested FSAVE might occur, the MC68020 MOVES instruction might be used to read the save CIR before the FSAVE is executed. If the value returned from the save CIR is the illegal format word, then the new FSAVE should be postponed. Reading the save CIR in this manner is not destructive.

5.3.4.7 FORMAT EXCEPTION, FRESTORE INSTRUCTION. This dialog is utilized by the MC68881 when an FRESTORE instruction is initiated by writing an invalid format word value to the restore CIR (in this context, the term invalid format value refers to any value that



5

Figure 5-30. FSAVE Format Exception Dialog

is not a null, idle, or busy format word value recognized by the MC68881). In this case, the MC68881 returns the explicit invalid format word ($02xx) to signal the unrecognized format word value. The main processor services this format word by writing an abort to the control CIR and initiating exception processing. The dialog for this operation is shown below. Note that this is a destructive exception, since any instruction that was executing is aborted when the FRESTORE instruction is initiated. However, this should not be detrimental since a successful restore operation also aborts any previously executing instruction. The dialog for this operation is shown in Figure 5-31.

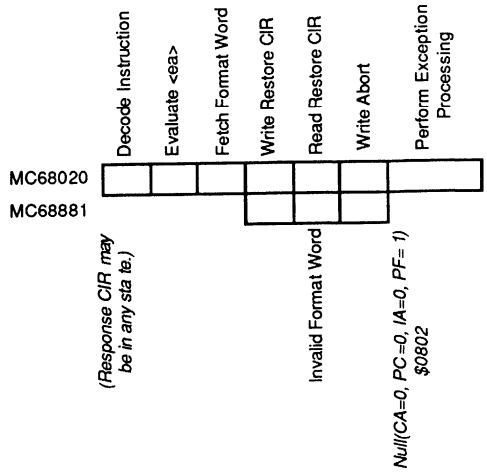


Figure 5-31. FRESTORE Format Exception Dialog

SECTION 6
INSTRUCTION EXECUTION TIMING

This section gives the instruction execution times for the MC68881 in terms of external clock cycles. This section provides the user with some reasonably accurate execution timing guidelines, not exact timings for every possible circumstance. This approach is used since the exact execution time for an instruction is highly dependent on such things as external data formats, input operand values, operand type combinations, and timing relationships with respect to the main processor. The timing numbers presented in the following tables allow the assembly language programmer or compiler writer to predict worse case timings needed to evaluate the performance of the MC68881, or to optimize code for concurrent execution. Also, the effect that various data formats and operand values have on execution times can be observed, such that data structures can be optimized for the highest performance for a given application. Finally, the timings for exception processing, context switching, and interrupt processing are included so that designers of multitasking or real-time systems can predict such things as task switch overhead and maximum interrupt latency due to floating-point operations.

6.1 FACTORS AFFECTING EXECUTION TIMES

When investigating instruction execution timing for the MC68881, it is assumed that a system designer requires the following information in order to make informed engineering trade-offs:

- Best case instruction execution timings, for determining whether or not an MC68881-based system can meet certain data processing performance criteria.
- Worst case instruction execution timings and how they affect execution concurrency, such that programs and compilers can be optimized to take maximum advantage of overlap under any timing circumstances.
- Guidelines to indicate how various programming practices can be utilized to improve upon the worst case execution times; and thus allow performance to approach, as closely as possible, the best case execution times for a given task.
- The effects that an MC68881 might have on system related timings such as context switch overhead time in multitasking systems, or interrupt latency time in a real-time system.

First of all, when defining the performance of any machine that can operate in an asynchronous manner, or where data dependencies affect execution times, a set of assumptions must be made in order to provide a measurable environment. In this manual, instruction execution times are given in clock cycles to remove clock frequency dependencies from the times given, and the following assumptions are used to define the context of the times given.

- The main processor is an MC68020, acting as the host to the MC68881, and the two devices use the same clock input.
- When the main processor initiates a command to the MC68881, any previous floating-point instruction has been completed and the MC68881 is in the idle state.
- All operands in memory, as well as the system stack, are long word aligned.
- A 32-bit data bus is used for communications between the MC68020 and both the MC68881 and the system memory.
- All memory accesses occur with no wait states (ie., three clock cycle reads and writes).
- All coprocessor accesses, except those to the response and save CIRs occur with no wait states. Accesses to the response and save CIRs require two wait cycles (five clock reads).

Note that the clock signal relationship between the MC68020 and the MC68881 assumed for these discussions is not a system requirement, but merely a simplification that allows easy measurement of instruction times. In general, the clock frequency of the MC68881 will affect absolute instruction timing more than that of the MC68020, since floating-point operations are usually computation intensive. However, the clock frequency relationship of the MC68020 and MC68881 can affect the execution time of an instruction due to the time needed to transfer operands of various sizes and due to actual activity of the two devices. The magnitude of the dependency of execution times on the clock frequency of the MC68020 varies with instruction types, since some instructions spend a relatively small amount of their overall execution time in communication with the main processor; whereas other instructions spend almost all of their execution time in communication with the main processor.

With this set of assumptions as a starting point, several factors must be defined that contribute to the overall execution time for a given instruction. Some of these factors are common to all instructions, while others are only applicable to certain instructions or data types. Particularly, the execution times for the conditional and system control instructions are not widely variable, but the execution time for an arithmetic or data movement instruction is heavily affected by things such as data values and exception checking. In order to better understand how these factors are combined to calculate the execution time for an arithmetic or move-to-floating-point register instruction, it is helpful to divide coprocessor instruction execution into the following steps:

1) Receive the command word from the host processor, decode it, and return the first service request primitive.
2) Receive the main processor program counter, if required.
3) Receive an external operand, if required.
4) Convert the operand to the internal extended format.
5) Perform the algorithm specified by the command word on the operand(s).
6) Round the result to the correct precision, check the result of the computation for conditions such as overflow, then store the result into a floating-point data register.

The first three of these steps require approximately the same amount of time for any instruction, but the last three steps can require widely varying amounts of time, even when comparing the execution time for a given instruction with different data inputs. For purposes of this discussion, the first three steps will be referred to as the instruction start-up phase, the

fourth step as the conversion phase, the fifth step as the calculation phase, and the sixth step as the round/store phase. The following sections discuss the factors that affect the execution time of an arithmetic instruction during each of these phases.

6.1.1 Instruction Start-up Phase

The factor that affects execution time most heavily during this phase of an instruction is the location and format of an external operand. The three possible locations for an input operand are 1) in a floating-point data register, 2) in a main processor data register, or 3) in external memory. If an operand resides in a floating-point data register before an instruction starts, then no data movement operations is required to prepare it for the calculation phase, and thus the start-up phase is very short. If an operand resides in a main processor data register, then the MC68881 uses the evaluate effective address and transfer data response primitive to request it from the MC68020. In this case, the MC68020 does not generate any operand memory cycles, and the operand is transferred to the MC68881 with a single bus cycle. The MC68881 then converts the signed integer or single precision floating-point number to extended precision and proceeds to the calculate phase.

For the third operand location case, execution time can vary widely due to two separate mechanisms, the addressing mode and alignment of the operand in memory, and the data format and value of the operand. The addressing mode used to locate an operand affects execution time in a straight forward manner due to the fixed nature of effective address calculations by the MC68020. For example, if the addressing mode used is address register indirect, (An), then no instruction prefetches or external bus cycles are required to calculate the address of the operand. If the addressing mode used is memory indirect with post-indexing, ([d,An],Xn.sz*scl,d), then up to five instruction prefetch words and one long word indirect address fetch may be required to calculate the final address of the operand. Then, once the operand is located, up to three long word fetches may be required to transfer the operand to the MC68881. The execution times for these operations are quite predictable (i.e., there are no data dependencies involved), although they are affected by instruction stream alignment, MC68020 cache hits, memory access times, memory width, and operand alignment. As mentioned earlier, certain assumptions are made with regard to these factors (for the purposes of this discussion) so that the tables in this section may be simplified. In order to include the effects of these factors, refer to the MC68020 User's Manual for more information regarding bus operation.

The second mechanism that can affect execution times for operands in memory is the data format. For the integer and binary floating-point formats, the execution times for conversions required to prepare the operand for the calculation are relatively free from data dependencies. However, for the packed decimal floating-point format, execution times can vary significantly due to the value of the input operand.

6.1.2 Calculation Phase

This is the most volatile portion of an instruction with respect to execution times. The main factor that affects the calculation time is the operation to be performed (e.g., a sine operation requires far more time than an add operation), but for a given operation, the execution time is data dependent. For the monadic operations, the data dependency is limited to the type

and value of the input operand; for the dyadic operations, the combination of the types and values of the two operands can also affect execution time (in this context, data type refers to the MC68881 extended precision representation of one of the five IEEE data types: normalized, denormalized, zero, infinity, and not-a-number). Because execution times vary due to data values and type combinations, the following tables indicate the execution time for each arithmetic operation for typical arguments, along with timing values for special case operand types such as zero, infinity, etc.

6.1.3 Round/Store Result Phase

The execution time for this phase of an instruction is dependent on the mode of operation that is programmed into the floating-point control register, as well as the value of the result. For example, if the rounding precision is programmed to be extended, execution will be faster than if it is single. Also, if the result of a calculation overflows or underflows the destination precision, then more time is required to handle that exception. In the following subsections, the overall execution times for the arithmetic operations assume the best case round/store phase time, and a separate table is included that allows the calculation of execution times for various rounding precisions and exception handling operations.

6

6.2 CONCURRENT INSTRUCTION EXECUTION

An important factor that should be considered when optimizing MC68020/MC68881 programs is the amount of concurrent execution time that an instruction allows. In general, the start-up time of an arithmetic instruction simultaneously occupies both the MC68020 and MC68881 (except as described in **6.4 COPROCESSOR INTERFACE OVERHEAD**). When the MC68881 enters the calculation phase, it allows the main processor to proceed with the execution of the next instruction by returning the null (CA=0, PF=0) response primitive. As long as the main processor does not attempt to initiate another floating-point instruction to the same MC68881 until the MC68881 has completed execution of the calculation and round/store phases, the two devices achieve fully concurrent execution (also, more than one MC68881 may be used in a system, allowing concurrent floating-point instruction execution). Figure 6-1 illustrates the overlapped execution timing for an arithmetic instruction. For the data movement, conditional, and system control instructions, relatively little concurrency is allowed, since the main processor is normally involved with these operations from start to finish (an important exception to this is the FMOVE.P <ea>,FPn instruction, which allows concurrent processing during the decimal-to-extended conversion).

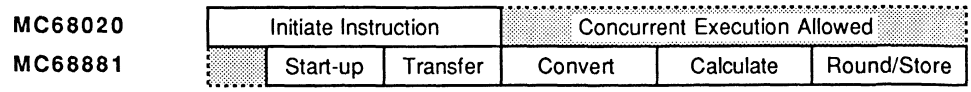


Figure 6-1. Concurrent MC68020/MC68881 Instruction Execution

In the following tables, the amount of time that is allowed for concurrent execution is included for each instruction, along with the overall execution time. This concurrent

execution time includes two periods: 1) from when the MC68020 begins execution of the instruction to when the first read of the response CIR begins, and 2) from the point that the MC68881 first returns the null (CA=0, PF=0) primitive to when it completes execution of the instruction. Due to the asynchronous nature of the MC68020 and MC68881 bus interfaces and variable instruction execution times, it may not be possible to achieve exact overlap of integer and floating-point instructions for extended periods of time (i.e., where a new floating-point instruction is dispatched by the MC68020 exactly as the MC68881 completes execution of the previous one), but significant performance increases can be achieved by careful interleaving of integer and floating-point instructions. In most cases (i.e., for an even distribution of input operand values), the maximum benefit can be achieved by using the typical instruction execution times to derive the amount of overlap time to be built into an instruction sequence. In this way, the MC68020 is required to wait for completion of a floating-point instruction only when it takes longer than the typical time to execute; while the MC68881 is not required to wait for very long to receive the next instruction if the previous one executed under best case conditions.

6.3 INTERRUPT LATENCY TIMES

In real-time systems, a very important factor pertaining to overall system performance is the response time required for a processor to handle an interrupt. In the M68000 Family of processors, interrupts are allowed to be asserted to the processor asynchronously, and they are handled on the next instruction boundary. While the average interrupt latency for the MC68020 is quite short, the maximum latency is often of critical importance, since real-time interrupts cannot require servicing in less than the maximum interrupt latency. The maximum interrupt latency for the MC68020 is approximately 250 clock cycles (for the MOVEM.L ([d32,An],Xn,d32),D0-D7/A0-A7 instruction where the last data fetch is aborted with a bus error; refer to the MC68020 User's Manual for more detailed information), but there are MC68881 instructions that may take two or three times that long to execute with typical operand types, combinations and values.

It may be unacceptable in a real-time system to have a worst-case interrupt latency time as large as 600 or more clock cycles (the length of some long floating-point instructions), therefore the MC68881 allows interrupts to be processed in the middle of the execution of a floating-point instruction, whenever possible, to reduce the latency time. The MC68881 does this in four ways:

1) by returning the null (CA=0, IA=1, PF=0) primitive when it enters the calculate phase of an instruction that allows concurrency. If the MC68020 is not in the trace mode, it is then free to fetch the next instruction, and process any pending interrupts at the instruction boundary. If the MC68020 is in the trace mode, it will wait for the MC68881 to complete execution and return the null (CA=0, PF=1) primitive before continuing with the next instruction, but it will service pending interrupts while it is waiting.
2) by returning the null (CA=1, IA=1) primitive when the main processor attempts to initiate a floating-point instruction before the previous concurrent operation has been completed; thus allowing the MC68020 to service interrupts while waiting for the coprocessor to start execution of the new instruction.
3) by returning the null (CA=1, IA=1) primitive during internal conversions for non-concurrent instruction execution (eg., FMOVE.<fmt> FPn,<ea>) before returning service request primitives to complete the operation.

4) by returning the not ready, come again format code during internal operations required by the FSAVE instruction. The MC68881 returns this format code in some cases (as described in **4.3 CONTEXT SWITCHING**) to enable it to store a smaller state frame, and the MC68020 may process interrupts while waiting for the save operation to begin.

For the first two cases, the MC68020 is allowed to process interrupts during the period labeled "Concurrent Execution Allowed" in Figure 6-1. For the third case, the period during which the MC68020 can process interrupts is illustrated in Figure 6-2. The timing for the fourth case is similar to the third case, except that the periods labeled "Convert," "Round" and "Transfer" for the MC68881 are not used for those purposes, but instead for saving of the internal state.

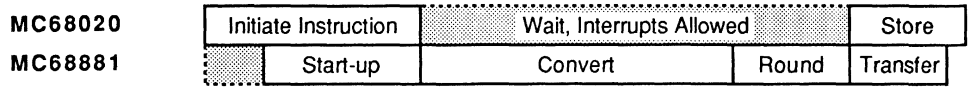


Figure 6-2. Non-Concurrent Instruction Execution, Interrupts Allowed

6

Basically, the maximum interrupt latency time for any MC68881 instruction is equal to the worst case execution time minus the interrupts allowed time, where both of these values are calculated using the tables below. For concurrent instructions, the execution time and allowed concurrency times are given, and the interrupt latency is the difference between those two values. For non-concurrent instructions, the amount of time during which interrupts are allowed is given in the tables as the number of allowed overlap clock cycles, and the interrupt latency is approximately equal to the total execution time minus the allowed overlap time. However, as shown in Figure 6-2, there may be two separate time periods during which the MC68020 is not allowed to process interrupts. For some instructions, such as the FMOVE.P FPn,<ea> instruction, these two periods are approximately equal and make up a small fraction of the overall execution time for the operation. On the other hand, for the FRESTORE and FSAVE instructions, the time required to transfer a Busy state frame is roughly equal to the overall execution time. In fact, the worst case interrupt latency due to an MC68881 instruction is for the FRESTORE instruction with a busy state frame.

6.4 COPROCESSOR INTERFACE OVERHEAD

For all of the instruction timings given in the following tables, all coprocessor interface bus cycle timing and associated processing are included in the overall execution times. However, it is assumed that when the main processor begins execution of a floating-point instruction, that the MC68881 has completed execution of any previous instruction. Thus, the MC68020 is never required to wait while the MC68881 completes an instruction. Also, it is assumed that when the MC68020 is waiting on the MC68881 during a non-concurrent instruction, the main processor reads the response register at exactly the moment when the MC68881 is prepared to return a service request primitive to complete or continue the instruction. If these conditions are not met, then the actual instruction execution time can be shorter or longer than the values shown in the tables, due to synchronization of the two devices.

First, it must be noted that the MC68881 does not begin execution of an instruction until the start of the read cycle from the response CIR in which the first primitive of the instruction dialog is returned to the main processor. If the MC68020 attempts to initiate a floating-point instruction before the previous one has completed, the MC68881 queues the command word or conditional predicate, and then instructs the MC68020 to wait (by encoding the null (CA=1, IA=1) primitive in the response CIR) until the previous instruction is completed. When the previous instruction has completed execution, the MC68881 does not begin execution of the queued instruction until the next read of the response CIR. The sequence of events for this situation is:

1) The MC68881 allows concurrent instruction execution by returning the null (CA=0, IA=1, PF=0) primitive to the MC68020.
2) The MC68020 encounters the next MC68881 instruction and attempts to initiate execution by writing to the command or condition CIR. The MC68020 then starts a read from the response CIR to determine what further action should be taken.
3) The MC68881 queues the instruction initiation request and changes the encoding of the response CIR to null (CA=1, IA=1), causing the MC68020 to wait.
4) The MC68020 continues to read the response CIR repeatedly until a new primitive is encoded, or an interrupt becomes pending (if an interrupt occurs, the MC68020 resumes polling of the response CIR after the interrupt handler executes an RTE instruction).
5) The MC68881 completes execution of the previous instruction and waits for the next read of the response CIR.
6) The MC68020 reads the response CIR, which either results in the return of a take exception primitive (due to an exception during the previous instruction) or causes the MC68881 to begin execution of the new instruction by returning the first primitive required for that operation.

The timing relationship of the main processor and the MC68881 during this sequence can affect the overall execution time of the new instruction, due to synchronization between the two devices. Specifically, if the MC68020 begins a read of the response CIR exactly one clock cycle before the MC68881 completes the execution of the previous instruction, then the MC68881 immediately begins execution of the new instruction by returning the first primitive of the new instruction dialog during that read cycle. This case is shown in Figure 6–3, which illustrates the best case timing for coprocessor interface overhead: two clock cycles.

Figure 6-3 also illustrates the typical coprocessor interface overhead timing which occurs when the MC68020 initiates a new instruction and the MC68881 is in the idle state. For this case, there is no overlap with a previous instruction at the beginning of the instruction dialog, and the coprocessor interface overhead for the new instruction is eleven clock cycles. Also note that this example assumes that the instruction prefetch requested by the cpGEN start-up operation was not satisfied by the previous prefetch bus cycle, and it does not hit in the MC68020 on-chip instruction cache. Refer to **6.5.2 Detail Timing Tables** for further discussion of the effects of instruction prefetching by the MC68020.

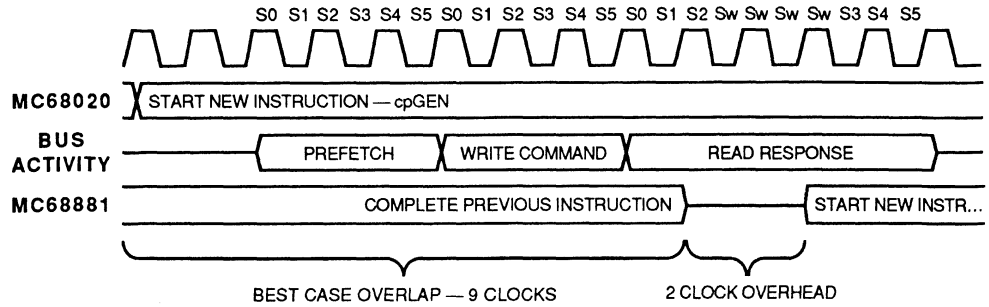


Figure 6-3. Best-Case Coprocessor Interface Overhead Timing

If the read cycle to the response CIR occurs before the MC68881 has completed execution of the previous instruction, then the MC68020 processes the null (CA=1, IA=1) primitive that is returned (by checking for pending interrupts and re-reading the response CIR if there are none). This operation requires 10 MC68020 clock cycles, and thus there is a 10 clock cycle worst case synchronization period that is part of the effective execution time for the new instruction (the worst case will occur if the response CIR read cycle starts 2 clock cycles before the previous instruction is completed). The worst case timing is shown in Figure 6-4. The exact amount of synchronization time required is dependent on the system environment, such as the clock signal relationship between the MC68020 and the MC68881, and the context of an instruction sequence. In all of the following tables, the typical case of no overlapped execution is assumed; thus, a coprocessor interface overhead value of eleven clock cycles is included in the timing numbers. If an attempt is made to optimize an instruction sequence for overlapped execution, the coprocessor interface overhead may be reduced by as much as nine clock cycles. However, incorrect "optimization" may result in an eleven clock cycle overhead, which is no worse than the no overlap case described above.

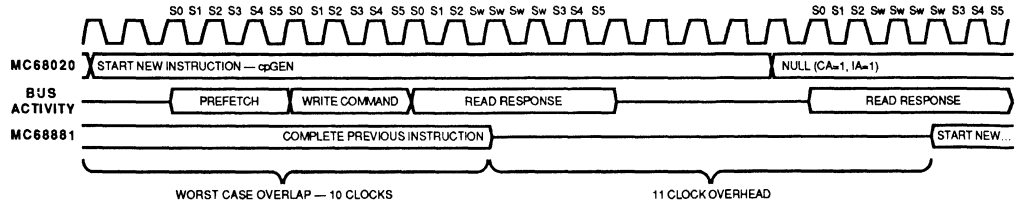


Figure 6-4. Worst-Case Coprocessor Interface Overhead Timing

A similar overhead effect will occur for nonconcurrent instructions that allow interrupt processing by returning the not ready format code, or the null (CA=1, IA=1) primitive in the middle of instruction execution (i.e., the FSAVE and FMOVE FPn,<ea> instructions). In these cases, the MC68881 will complete as much of the instruction as possible while allowing interrupts, and then prepare to encode a valid format code or service request

primitive in the save or response CIR during the next read by the MC68020. The timing relationship between the start of the read cycle and the completion of internal operations by the MC68881 is identical to the timing described above for the instruction start up phase. Thus, the same 10 clock cycle overhead factor might be added to the execution time for these instructions.

In the following tables, the assumptions stated earlier are used (i.e., the main processor is an MC68020 running on the same clock as the MC68881) and the coprocessor interface overhead for all operations other than instruction initiation is included based on those assumptions. If an instruction is executed under conditions other than those described, the execution time may be increased in increments of 10 clock cycles if necessary.

6.5 EXECUTION TIMING TABLES

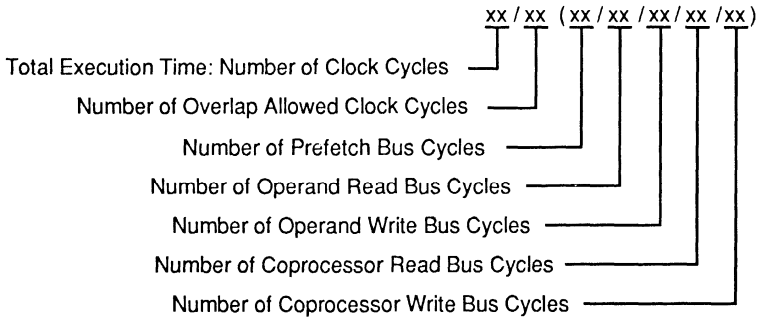
In the following subsections, timing tables are presented that allow the calculation of best case, typical and worst case execution times for any MC68881 instruction. These tables are based on the assumptions stated above, and include the total execution time for each instruction. In other words, the numbers that are calculated using these tables indicate the time from when the MC68020 begins execution of the coprocessor instruction (i.e., when the instruction has been prefetched and loaded into the instruction decode register) to when the MC68881 and/or MC68020 completes execution of that instruction (i.e., when a read of the response CIR indicates a null (CA=0, PF=1), when conditional processing is completed, or when the last operand transfer to or from the MC68881 has been completed).

Bus cycle activity is also indicated by the tables, and includes all bus cycles generated by a particular operation. Note that instruction prefetch and operand write cycles requested by the execution of a given instruction may not actually be executed during the execution of the instruction, but are queued by the MC68020 bus interface unit for completion as soon as the bus is available (refer to the MC68020 User's Manual for more information on bus cycle overlap). When a floating-point operation is completed, a prefetch request will have been generated by the MC68020 to replace each word of instruction stream used by the instruction, or to refill the instruction pipe in the case of a conditional branch or trap instruction, or an exception.

The timing information given in the following tables for some operations includes three numbers that depend on the context of the instruction (i.e., whether the MC68020 cache is enabled and contains the instruction, and the alignment of the instruction stream):

1) The best case value, where prefetches hit in the MC68020 on-chip cache and the instruction benefits from the maximum overlap, in the MC68020 pipeline, with other instructions. Due to the highly volatile nature of the instruction pipeline, this case is not easy to achieve intentionally, but occurs occasionally.
2) The cache-only-case, where prefetches hit in the MC68020 on-chip cache, but the instruction does not overlap with preceding or following instructions.
3) The worst case, where prefetches do not hit in the MC68020 on-chip cache, or the cache is disabled, and there is no instruction overlap. It is further assumed that the instruction is aligned such that a prefetch is executed before the MC68020 writes to the MC68881 command CIR.

The execution time entries in most of the following tables contain seven numbers. The left-most number is the total execution time for the operation in clock cycles, followed by the number of clock cycles of the total execution time that are allowed to overlap with execution of other operations by the main processor. Then, in parenthesis, the bus cycle activity is included, which indicates the number of instruction prefetch, operand read, operand write, coprocessor read and coprocessor write bus cycles that will be generated by the execution of the instruction. An example of the format of an entry from the timing tables is:



The total number of clocks required for the bus activity in each entry can be derived by multiplying the total number of bus cycles by three (this does not account for the fact that reads from the response and save CIRs require five clocks rather than three, but the two clock cycle discrepancy is usually negligible compared with the overall execution time for an instruction). For some instructions, the number of coprocessor read cycles indicated by the tables may not reflect the actual number of read cycles that are executed during the dialog for the instruction. This is due to the fact that only the first occurrence of a series of null (CA=1, IA=1) or null (CA=0, PF=0, IA=1) primitives is included in the tables. For example, the MC68881 forces the main processor to wait during the conversion phase of the FMOVE FPn,<ea> instruction by using the null response primitive. The MC68020 may perform numerous response CIR read cycles during the time that it waits on the MC68881, but only the first of this series of read cycles is included in the timing table entry. Although this simplification may fail to indicate the true number of coprocessor read cycles executed by the MC68020, it allows the tables to accurately indicate the minimum number of different response primitive and operand reads that must be executed by a main processor, regardless of its type or clock and bus speed.

The timing tables in the following sections are divided into two major groups. First, several tables are presented that allow quick determination of the typical execution time for all instructions. These tables assume typical operand inputs and operating conditions, for simplicity, and are comprehensive. No more than two of the first six tables are used to determine the typical execution time for a given instruction. One table is used to determine the basic execution time for the selected instruction; and a second table may be used to determine the additional time required for the calculation of the effective address by the MC68020 (for those instructions that require an effective address calculation).

The second group of tables is used to calculate a more precise execution timing value for a specific instruction, addressing mode, and operand type combination than is available in the first group of tables. This group of tables is also useful for the calculation of execution times where the main processor is not an MC68020, since the timing for each phase of instruction execution is included in a separate table. This allows timings that are only dependent on the MC68881 to be calculated and added to the timing characteristics of the main processor.

6.5.1 Timing Tables for Typical Execution

This set of tables allows a quick determination of the typical execution time for any MC68881 instruction when the MC68020 is used as the main processor. The first table presented is for effective address calculations performed by the MC68020. Entries from this table are added to the entries in the other tables in this subsection, if necessary, to obtain the overall execution time for an operation. The assumptions for the following tables are:

- The main processor is an MC68020 and operates on the same clock as the MC68881. Instruction prefetches do not hit in the MC68020 cache (or it is disabled) and the instruction is aligned such that a prefetch occurs before the command CIR is written by the MC68020.
- A 32-bit memory interface is used, and memory accesses occur with zero wait states. All memory operands, as well as the stack pointers, are long-word aligned.
- Accesses to the MC68881 require 3 clock cycles, with the exception of read accesses to the response and save CIRs, which require 5 clock cycles.
- No instruction overlap is utilized, so the coprocessor interface overhead is 11 clocks. This can be reduced to 2 clock cycles if optimized code sequences are used, or may be 11 clock cycles if overlap is attempted and a synchronization delay is required.
- Typical operand conversion and calculation times are used (i.e., input operands are assumed to be normalized numbers in the legal range for a given function).
- No exceptions are enabled or occur, and the default rounding mode and precision of round-to-nearest, extended precision, is used.

6.5.1.1 EFFECTIVE ADDRESS CALCULATIONS. For any instruction that requires an operand external to the MC68881, an evaluate effective address and transfer data response primitive is issued by the MC68881 during the dialog for that instruction. The amount of time that is required for the MC68020 to calculate the effective address while processing this primitive for each addressing mode, excluding the transfer of the data to the MC68881, is shown below. The times given in this table include all bus cycles required to perform the address calculation (such as instruction prefetches and memory indirect fetches).

For the FMOVEM instruction, the following table is also used to determine the time required for the MC68020 to perform an address calculation (implied by the transfer multiple coprocessor registers primitive). For the FScc, FRESTORE and FSAVE instructions, the request to evaluate an effective address is implied by the F-line instruction word; therefore no response primitive is issued by the MC68881 to request the evaluation. The following table is used for these three instructions to adjust the basic instruction execution time to reflect the addressing mode that is used.

Note that the following table applies only to the MC68020 effective address calculation time for coprocessor instructions. The execution times included in this table are not the same as the calculate effective address times given in the *MC68020 32-Bit Microprocessor User's Manual* for normal instruction execution .

Addressing Mode	Best Case	Cache Case	Worst Case
Dn or An	0/0 (0/0/0/0/0)	0/0 (0/0/0/0/0)	0/0 (0/0/0/0/0)
(An)	0/0 (0/0/0/0/0)	2/0 (0/0/0/0/0)	2/0 (0/0/0/0/0)
(An)+	3/0 (0/0/0/0/0)	6/0 (0/0/0/0/0)	6/0 (0/0/0/0/0)
-(An)	3/0 (0/0/0/0/0)	6/0 (0/0/0/0/0)	6/0 (0/0/0/0/0)
(d_{16},An) or (d_{16},PC)	0/0 (0/0/0/0/0)	2/0 (0/0/0/0/0)	3/0 (1/0/0/0/0)
(xxx).W	0/0 (0/0/0/0/0)	2/0 (0/0/0/0/0)	3/0 (1/0/0/0/0)
(xxx).L	1/0 (0/0/0/0/0)	4/0 (0/0/0/0/0)	5/0 (1/0/0/0/0)
#<data>	0/0 (0/0/0/0/0)	0/0 (0/0/0/0/0)	0/0 (0/0/0/0/0)
(d_8,An,Xn) or (d_8,PC,Xn)	1/0 (0/0/0/0/0)	4/0 (0/0/0/0/0)	5/0 (1/0/0/0/0)
(d_{16},An,Xn) or (d_{16},PC,Xn)	3/0 (0/0/0/0/0)	6/0 (0/0/0/0/0)	7/0 (1/0/0/0/0)
(B)	3/0 (0/0/0/0/0)	6/0 (0/0/0/0/0)	7/0 (1/0/0/0/0)
(d_{16},B)	5/0 (0/0/0/0/0)	8/0 (0/0/0/0/0)	9/0 (1/0/0/0/0)
(d_{32}B)	11/0 (0/0/0/0/0)	14/0 (0/0/0/0/0)	16/0 (2/0/0/0/0)
([B],I)	8/0 (0/1/0/0/0)	11/0 (0/1/0/0/0)	12/0 (1/1/0/0/0)
([B],I,d_{16})	8/0 (0/1/0/0/0)	11/0 (0/1/0/0/0)	12/0 (1/1/0/0/0)
([B],I,d_{32})	10/0 (0/1/0/0/0)	13/0 (0/1/0/0/0)	15/0 (2/1/0/0/0)
([d_{16},B],I)	10/0 (0/1/0/0/0)	13/0 (0/1/0/0/0)	14/0 (1/1/0/0/0)
([d_{16},B],I,d_{16})	10/0 (0/1/0/0/0)	13/0 (0/1/0/0/0)	15/0 (2/1/0/0/0)
([d_{16},B],I,d_{32})	12/0 (0/1/0/0/0)	15/0 (0/1/0/0/0)	17/0 (2/1/0/0/0)
([d_{32},B],I)	16/0 (0/1/0/0/0)	19/0 (0/1/0/0/0)	21/0 (2/1/0/0/0)
([d_{32},B],I,d_{16})	16/0 (0/1/0/0/0)	19/0 (0/1/0/0/0)	21/0 (2/1/0/0/0)
([d_{32},B],I,d_{32})	18/0 (0/1/0/0/0)	21/0 (0/1/0/0/0)	24/0 (3/1/0/0/0)

B = Base address; 0, An, PC, Xn, An + Xn, PC + Xn. Form does not affect timing.
I = Index; 0 or Xn.
Note that Xn cannot be in B and I at the same time. Scaling and size of Xn does not affect timing.

6.5.1.2 ARITHMETIC OPERATIONS. The following table gives the typical instruction execution time for each arithmetic instruction. This group of instructions includes the majority of the MC68881 operations such as FADD, FSUB, etc. In addition to the instructions that perform arithmetic calculations as part of their function, the FCMP, FMOVE and FTST instructions are also included, since an implicit conversion is performed by those operations. For memory operands, the timing for the appropriate effective addressing mode must be added to the numbers in this table to determine the overall instruction execution times. In order to simplify these tables the overall execution time, overlap allowed time, and bus cycle activity numbers are separated into individual tables.

Overall Execution Time:

Instruction	FPm Source	Memory Source or Destination Operand Format*				
		Integer**	Single**	Double	Extended	Packed
FABS	35	62	54	60	58	872
FACOS	625	652	644	650	648	1462
FADD	51	80	72	78	76	888
FASIN	581	608	600	606	604	1418
FATAN	403	430	422	428	426	1240
FATANH	693	720	712	718	716	1530
FCMP	33	62	54	60	58	870
FCOS	391	418	410	416	414	1228
FCOSH	607	634	626	632	630	1444
FDIV	103	132	124	130	128	940
FETOX	497	524	516	522	520	1334
FETOXM1	545	572	564	570	568	1382
FGETEXP	45	72	64	70	68	882
FGETMAN	31	58	50	56	54	868
FINT	55	82	74	80	78	892
FINTRZ	55	82	74	80	78	892
FLOGN	525	552	544	550	548	1362
FLOGNP1	571	598	590	596	594	1408
FLOG10	581	608	600	606	604	1418
FLOG2	581	608	600	606	604	1418
FMOD	67	94	86	92	90	902
FMOVE to FPn	33	60	52	58	56	870
FMOVE to memory***	—	100	80	86	72	1996
FMOVECR****	29	—	—	—	—	—
FMUL	71	100	92	98	96	908
FNEG	35	62	54	60	58	872
FREM	67	94	86	92	90	902
FSCALE	41	70	62	68	66	878

* Add the appropriate effective address calculation time. (Continued)

** If the source or destination is an MC68020 data register, subtract 5 or 2 clock cycles, respectively.

*** Assumes a static k-factor is used if the destination data format is packed decimal. Add 14 clock cycles if a dynamic k-factor is used.

**** The source operand is from the constant ROM rather than a floating-point data register.

Overall Execution Time (continued):

Instruction	FPm Source	Memory Source or Destination Operand Format*				
		Integer**	Single**	Double	Extended	Packed
FSGLDIV	69	98	90	96	94	906
FSGLMUL	59	88	80	86	84	896
FSIN	391	418	410	416	414	1228
FSINCOS	451	478	470	476	474	1288
FSINH	687	714	706	712	710	1524
FSQRT	107	134	126	132	130	944
FSUB	51	80	72	78	76	888
FTAN	473	500	492	498	496	1310
FTANH	661	688	680	686	684	1498
FTENTOX	567	594	586	592	590	1404
FTST	33	60	52	58	56	870
FTWOTOX	567	594	586	592	590	1404

* Add the appropriate effective address calculation time.
** If the source or destination is an MC68020 data register, subtract 5 or 2 clock cycles, respectively.

6

Overlap Allowed Time:

Operation Type	FPm Source	Memory Source or Destination Operand Format				
		Integer	Single	Double	Extended	Packed
FPn Destination*	–12	–31	–31	–37	–43	–43
Move to Dn or Memory**	—	50	38	38	18	~1942

* Subtract these numbers from the overall execution time value in the previous table to determine the allowed overlap time for a particular instruction. The result of this calculation includes 9 clocks of allowed overlap with the previous MC68881 instruction.
** These numbers represent the amount of time in the middle of the instruction during which the MC68020 may process interrupts. An additional period of 9 clocks is allowed to overlap with the execution of the previous MC68881 instruction.
~ Indicates a typical time for the binary-to-decimal conversion.

Bus Cycle Activity:

Operation Type	FPm Source	Memory Source or Destination Operand Format*				
		Integer**	Single**	Double	Extended	Packed
FPn Destination	(1/0/0/1/1)	(1/1/0/2/2)	(1/1/0/2/2)	(1/2/0/2/3)	(1/3/0/2/4)	(1/3/0/2/4)
Move to Memory***	—	(1/0/1/4/1)	(1/0/1/4/1)	(1/0/2/5/1)	(1/0/3/6/1)	(1/0/3/6/1)

* Add the appropriate effective address calculation bus cycle activity.
** If the source or destination is an MC68020 data register, subtract (0/1/0/0/0) or (0/0/1/0/0), respectively.
*** Includes the read of one null (CA=1, IA=1) primitive when the conversion starts, the evaluate effective address and transfer data primitive when the conversion is complete, and a null (CA=0) primitive after the transfer is complete. The MC68020 reads additional null (CA=1, IA=1) primitives while waiting for the transfer to start. If no interrupts occur, the number of additional response CIR read cycles is 5 for integer, 3 for single or double, 1 for extended and ~194 for packed.

6.5.1.3 MOVE CONTROL REGISTER AND FMOVEM OPERATIONS. The following table gives the execution times for the FMOVE FPcr and FMOVEM instructions. The timing for the appropriate effective addressing mode must be added to the numbers in this table to determine the overall instruction execution times.

Operation*		Best Case	Cache Case	Worst Case
FMOVE	FPcr,Rn	29/6 (0/0/0/3/1)	31/6 (0/0/0/3/1)	34/9 (1/0/0/3/1)
	FPcr,<ea>	31/6 (0/0/1/3/1)	33/6 (0/0/1/3/1)	36/9 (1/0/1/3/1)
	Rn,FPcr	26/6 (0/0/0/2/2)	28/6 (0/0/0/2/2)	31/9 (1/0/0/2/2)
	<ea>,FPcr	31/6 (0/1/0/2/2)	33/6 (0/1/0/2/2)	36/9 (1/1/0/2/2)
	#<data>,FPcr	30/6 (0/0/0/2/2)	30/6 (0/0/0/2/2)	31/9 (2/0/0/2/2)
FMOVEM	FPcr_list,<ea>	25+6n/6 (0/0/n/2+n/1)	27+6n/6 (0/0/n/2+n/1)	30+6n/9 (1/0/n/2+n/1)
	<ea>,FPcr_list	25+6n/6 (0/n/0/2/1+n)	27+6n/6 (0/n/0/2/1+n)	30+6n/9 (1/n/0/2/1+n)
	#<data>,FPcr_list	24+6n/6 (0/0/0/2/1+n)	25+6n/6 (0/0/0/2/1+n)	29+6n/9 (1+n/0/0/2/1+n)
FMOVEM	FPdr_list,<ea>	35+25n/6 (0/0/3n/3+3n/1)	37+25n/6 (0/0/3n/3+3n/1)	40+25n/9 (1/0/3n/3+3n/1)
	<ea>,FPdr_list	33+23n/6 (0/3n/0/3/1+3n)	35+23n/6 (0/3n/0/3/1+3n)	38+23n/9 (1/3n/0/3/1+3n)
	Dn,<ea>	49+25n/6 (0/0/3n/4+3n/2)	51+25n/6 (0/0/3n/4+3n/2)	54+25n/9 (1/0/3n/4+3n/2)
	<ea>,Dn	47+23n/6 (0/3n/0/4/2+3n)	49+23n/6 (0/3n/0/4/2+3n)	52+23n/9 (1/3n/0/4/2+3n)

* Add the appropriate effective address calculation time.
n is the number of registers transferred.

FPcr or FPdr indicates any one of the floating-point control or data registers, respectively. FPcr_list or FPdr_list indicates a list of any combination of the floating-point control or data registers, respectively.

6.5.1.4 CONDITIONAL INSTRUCTIONS. The following table gives the execution times for the MC68881 conditional instructions. Each entry in this table, except those for the FScc instruction, is complete and does not require the addition of values from any other table. For the FScc instruction, the only additional factor that must be included is the calculate effective address time for the operand to be modified.

Since the conditional instructions are intrinsic to the M68000 Family Coprocessor Interface (i.e., they are not defined by the MC68881 through the use of response primitives), the MC68020 performs most of the processing associated with these instructions. The only part of the instruction that is performed by the MC68881 is the evaluation of the conditional predicate written to the condition CIR. Thus, the execution times given in the table below are heavily dependent on the environment in which the main processor executes.

The overlap allowed times given for these instructions indicates the time at the beginning of the instruction that can overlap with the execution of the previous instruction by the MC68881. There is no overlap allowed at the end of the instruction, since the MC68881 is always idle during the period where the MC68020 is completing the operation.

Operation	Comments	Best Case	Cache Case	Worst Case
FBcc.W	Branch Taken	18/6 (0/0/0/1/1)	20/6 (0/0/0/1/1)	23/6 (2/0/0/1/1)
	Branch Not Taken	16/6 (0/0/0/1/1)	18/6 (0/0/0/1/1)	19/6 (1/0/0/1/1)
FBcc.L	Branch Taken	18/6 (0/0/0/1/1)	20/6 (0/0/0/1/1)	23/6 (2/0/0/1/1)
	Branch Not Taken	16/6 (0/0/0/1/1)	18/6 (0/0/0/1/1)	21/6 (2/0/0/1/1)
FDBcc	True, Not Taken	18/6 (0/0/0/1/1)	20/6 (0/0/0/1/1)	24/9 (2/0/0/1/1)
	False, Not Taken	22/6 (0/0/0/1/1)	24/6 (0/0/0/1/1)	32/9 (4/0/0/1/1)
	False, Taken	18/6 (0/0/0/1/1)	20/6 (0/0/0/1/1)	26/9 (3/0/0/1/1)
FNOP	No Operation	16/6 (0/0/0/1/1)	18/6 (0/0/0/1/1)	19/6 (1/0/0/1/1)
FScc	Dn	16/6 (0/0/0/1/1)	18/6 (0/0/0/1/1)	21/9 (2/0/0/1/1)
	(An)+ or –(An)*	18/6 (0/0/1/1/1)	22/6 (0/0/1/1/1)	25/9 (2/0/1/1/1)
	Memory**	16/6 (0/0/1/1/1)	20/6 (0/0/1/1/1)	23/9 (2/0/1/1/1)
FTRAPcc	Trap Taken	36/6 (0/1/4/1/1)	39/6 (0/1/4/1/1)	47/9 (3/1/4/1/1)
	Trap Not Taken	16/6 (0/0/0/1/1)	18/6 (0/0/0/1/1)	22/9 (2/0/0/1/1)
FTRAPcc.W	Trap Taken	38/6 (0/1/4/1/1)	41/6 (0/1/4/1/1)	45/9 (3/1/4/1/1)
	Trap Not Taken	18/6 (0/0/0/1/1)	20/6 (0/0/0/1/1)	23/9 (2/0/0/1/1)
FTRAPcc.L	Trap Taken	40/6 (0/1/4/1/1)	43/6 (0/1/4/1/1)	52/9 (4/1/4/1/1)
	Trap Not Taken	20/6 (0/0/0/1/1)	22/6 (0/0/0/1/1)	27/9 (3/0/0/1/1)

* For condition true; subtract one clock for condition false.
** Add the appropriate effective address calculation time.

6.5.1.5 FSAVE AND FRESTORE INSTRUCTIONS. The time required for a context save or restore operation is given in the table below. The appropriate calculate effective address times must be added to the values in this table to obtain the total execution time for these operations. For the FSAVE instruction, the MC68881 may use the not ready format code to force the MC68020 to wait while internal operations are completed in order to reduce the size of the saved state frame or reach a point where a save operation can be performed. The idle (minimum) time occurs if the MC68881 is in the idle phase when the save CIR is read (refer to **4.3.3 FSAVE and FRESTORE Protocols** for a definition of instruction phases). A time between the idle (minimum) and the idle (maximum) occurs if an instruction is in the end phase when the save CIR is read. The busy (minimum) time occurs if the MC68881 is in the initial phase, or at a save boundary in the middle phase, when the save CIR is read. Finally, the busy (maximum) time occurs if the MC68881 has just passed a save boundary in the middle phase when the save CIR is read.

Operation*	State Frame	Best Case	Cache Case	Worst Case
FRESTORE	Null	19/4* (0/1/0/1/1)	21/4* (0/1/0/1/1)	22/4* (1/1/0/1/1)
	Idle	55/4* (0/7/0/1/7)	57/4* (0/7/0/1/7)	58/4* (1/7/0/1/7)
	Busy	289/4* (0/46/0/1/46)	291/4* (0/46/0/1/46)	292/4* (1/46/0/1/46)
FSAVE	Null	14/1 (0/0/1/1/0)	16/1 (0/0/1/1/0)	18/1 (1/0/1/1/0)
	Idle (min.)	50/1 (0/0/7/7/0)	52/1 (0/0/7/7/0)	54/1 (1/0/7/7/0)
	Idle (max.)**	__/1, __ (0/0/7/7/0)	__/1, __ (0/0/7/7/0)	__/1, __ (1/0/7/7/0)
	Busy (min.)	284/1 (0/0/46/46/0)	286/1 (0/0/46/46/0)	288/1 (1/0/46/46/0)
	Busy (max.)**	__/1, __ (0/0/46/46/0)	__/1, __ (0/0/46/46/0)	__/1, __ (1/0/46/46/0)

* Add the appropriate effective address calculation time. Note that the overlap time available for the FRESTORE instruction is of little use, since this operation destroys the previous context of the MC68881.

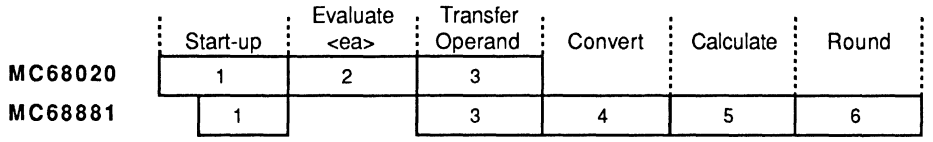
** The second overlap allowed number represents the period during which the MC68881 is preparing to perform the save operation and the MC68020 may process interrupts.

6.5.2 Detail Timing Tables

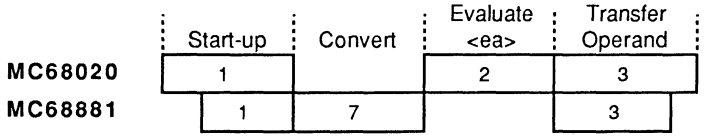
This set of tables allows the calculation of a more precise execution time for an instruction, based on the input operand format and type, than can be obtained with the typical timing tables given previously. Also, these tables contain the information necessary to determine instruction execution timing for a system that does not utilize the MC68020 as the main processor. The assumptions given previously are used for these tables, with further restrictions described separately for each table. Note that the timing numbers in the typical timing tables are derived, in most cases, by using the following set of tables.

In order to better understand the relationship of each table in this group, diagrams are included below that break each basic instruction type into separate execution components. For each component, the appropriate tables that are used to calculate the execution time are identified. These diagrams can also be used to clarify the distribution of responsibility for instruction execution between the MC68020 and the MC68881 and to more clearly illustrate the periods of time during which overlapped execution may occur. In these diagrams, the numbers inside each box indicates the table that is used to determine the timing for that phase of the instruction; the identification key for these tables follows the diagrams.

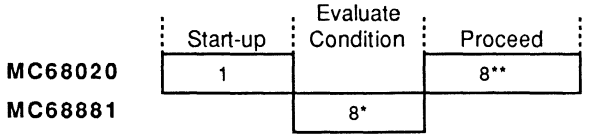
Memory-to-Register, Register-to-Register Operations



Move Register-to-Memory



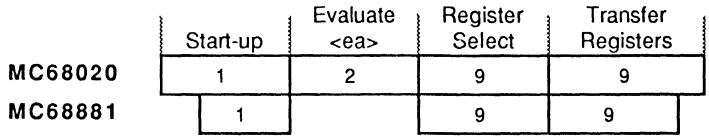
Conditional Operations



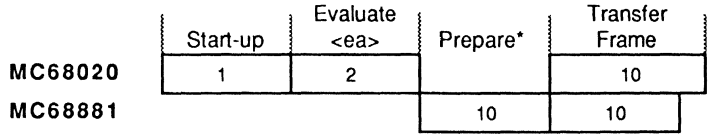
* The timing for the evaluation of the conditional predicate is given separately in section **6.5.2.7 CONDITIONAL TERMINATION**.

** The action taken by the MC68020 after the conditional predicate is evaluated by the MC68881 depends on the instruction (FBcc, FDBcc, FScc or FTRAPcc).

Move Control or Multiple Registers

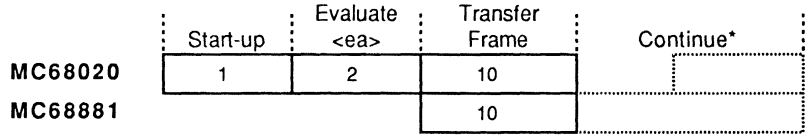


Context Save Operation



* During this period, the MC68881 may force the MC68020 to wait while an internal operation is completed, or reaches a point where a save operation can be performed.

Context Restore Operation



* When the context restore operation is completed, the MC68881 continues with any operation that was suspended by a previous context save. The MC68020 does not reestablish communications with the MC68881 during the FRESTORE instruction, but the execution of a subsequent RTE instruction restores the MC68020 context to the state of the previously suspended operation if necessary.

Table Identification:

1 — Instruction Start-Up
2 — Effective Address Calculations
3 — Operand Transfers
4 — Input Operand Conversions
5 — Arithmetic Calculations
6 — Rounding And Exception Handling
7 — Output Operand Conversions
8 — Conditional Instructions
9 — Multiple Register Transfers
10 — State Frame Transfers
11 — Exception Processing

As an example of how to use the information given in the following paragraphs, consider the FADD.P (A0)+,FP0 instruction. First the instruction start-up table is used to determine the time required by the MC68020 to initiate the instruction (by writing the command word and reading the first response primitive). In this case, the first response is evaluate effective address and transfer data (with the PC bit set if any exceptions are enabled). The operand transfer table is then used to determine the time required to transfer the packed decimal string from memory to the MC68881, and this table requires the addition of the effective address calculation time. Thus, the calculate effective address table is used to determine the time required by the MC68020 to calculate the effective address, (A0)+, and those numbers are added to the start-up and transfer timing numbers. Note that these first three values are

almost entirely dependent on the MC68020, and as such are not used if the main processor is not an MC68020.

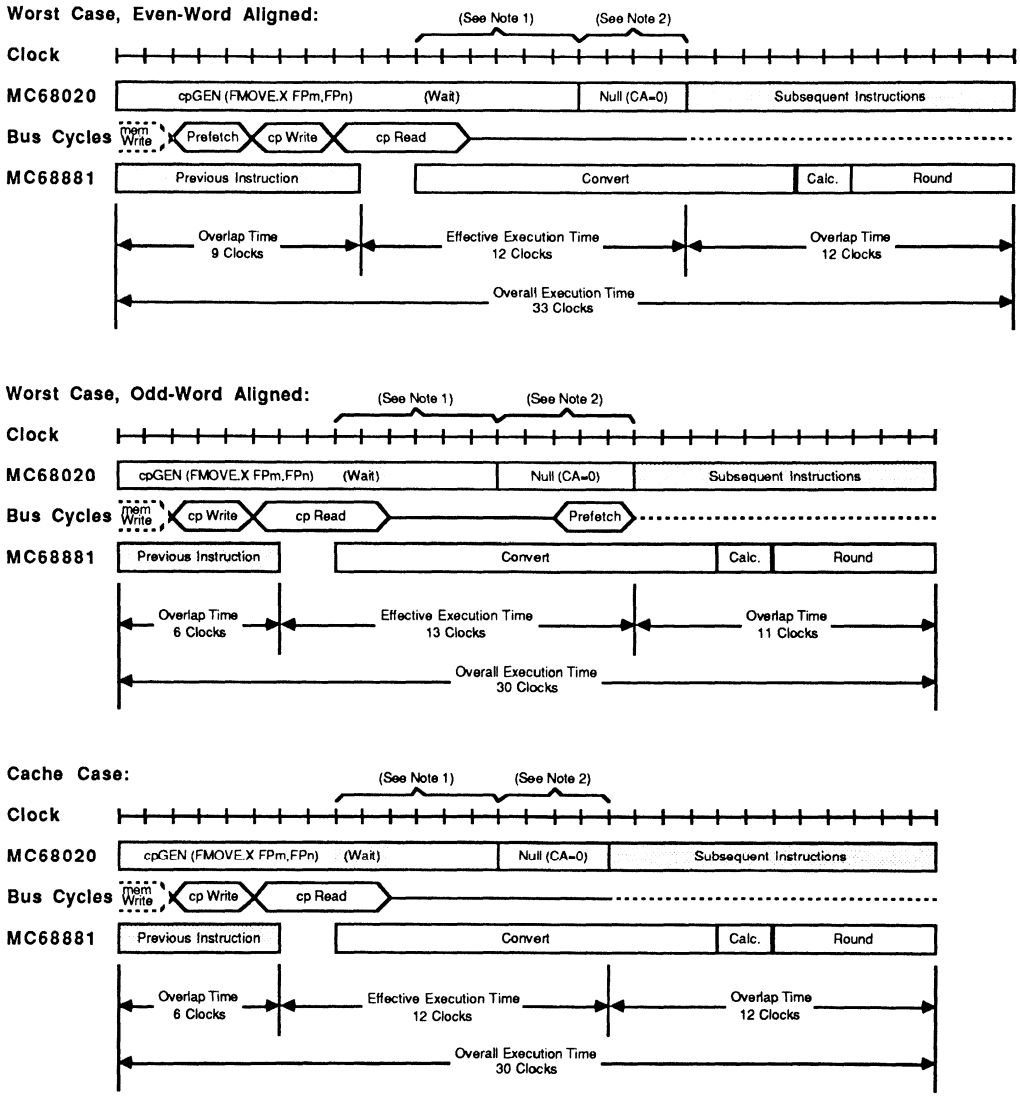
To complete the timing calculation, a fourth table is used to determine the decimal-to-binary conversion time, based on the input operand data type and value. Finally, the fifth and sixth tables used determine the time required for the addition and rounding operations. The second set of three operations are totally independent of the main processor, such that the timing numbers derived for them can be utilized by non-MC68020 based system designers.

As a further aid to the understanding of how the MC68020 and MC68881 interract during the execution of an instruction, four diagrams are presented in Figures 6–5 and 6–6. The bus cycle activity and overlapped execution that is allowed during the communications dialog is shown in the diagrams, in addition to illustrations of the effect of instruction alignment, enabled exceptions and device synchronization. These diagrams represent the clock-cycle-by-clock-cycle activity of the two devices for four cases of the FMOVE instruction. The first three diagrams describe the FMOVE.X FPm,FPn instruction for worst case and cache case operation, and the fourth diagram describes the FMOVE.S (An),FPn instrucion.

The three diagrams in Figure 6–5 show three cases of the FMOVE.X FPm,FPn instruction. The first and second cases show worst case operation (where the instruction prefetches required to replace the FMOVE instruction do not hit in the MC68020 on-chip cache) for the two possible alignments of the instruction. If the first word of the instruction is at an even word address, then the prefetch request generated by the cpGEN start-up operation (to replace the F-line operation word) will cause an external bus cycle to be executed. This prefetch aquires 2 words, one of which fills the cpGEN request, and one that is held in a temporary register. The time required to execute this prefetch cycle adds directly to the overall execution time for the instruction, as well as the front-end overlap allowed time. When the null (CA=0) primitive is processed by the MC68020, a second prefetch request is generated (to replace the command word) which is filled with the word from the temporary register. Thus, the null operation prefetch request does not generate an external bus cycle.

When the MC68020 polls the response CIR, the MC68881 begins execution of the instruction in the fourth clock cycle of the read cycle. As the MC68881 proceeds with the conversion operation, the MC68020 then completes the cpGEN start-up operation and processes the null (CA=0) primitive. The 10 clock cycles required to perform these operations overlap with the execution of the instruction by the MC68881, and thus are not included in the overall execution time calculation (although they are included in the effective execution time calculation). The same consideration applies to the second and third diagrams in Figure 6–5.

As shown in the second diagram of Figure 6–5, if the first word of the instruction is at an odd word address then the prefetch requested by the cpGEN start-up will be filled from the temporary register (which was loaded by a prefetch requested during the previous MC68020 instruction) and an external bus cycle is not required. When the null (CA=0) primitive is processed, a second prefetch request is generated which must be filled by the execution of an external bus cycle. Thus, the start-up operation for this case is a minimum of 3 clock cycles shorter than the first case (although the overlap allowed time is also shorter) while the time required to process the null primitive is at least 1 clock cycle longer. Since the null



Notes: 1) These six clocks do not add to the overall execution time for the instruction, only the effective execution time for the MC68020.

2) This operation does not add to the overall execution time for the instruction, only the effective execution time for the MC68020.

Figure 6-5. Instruction Overlap Examples — FMOVE.X FPm,FPn.

processing overlaps with the execution of the operand conversion by the MC68881, the overall execution time for the instruction is shorter, although the overlap allowed time at the end of the instruction is reduced.

For the third case, both of the instruction prefetch reqests generated during the instruction execution are satisfied by either the temporary register or the on-chip instruction cache. Thus, the overall execution time achieves the absolute best case while allowing the maximum possible overlap between the two devices.

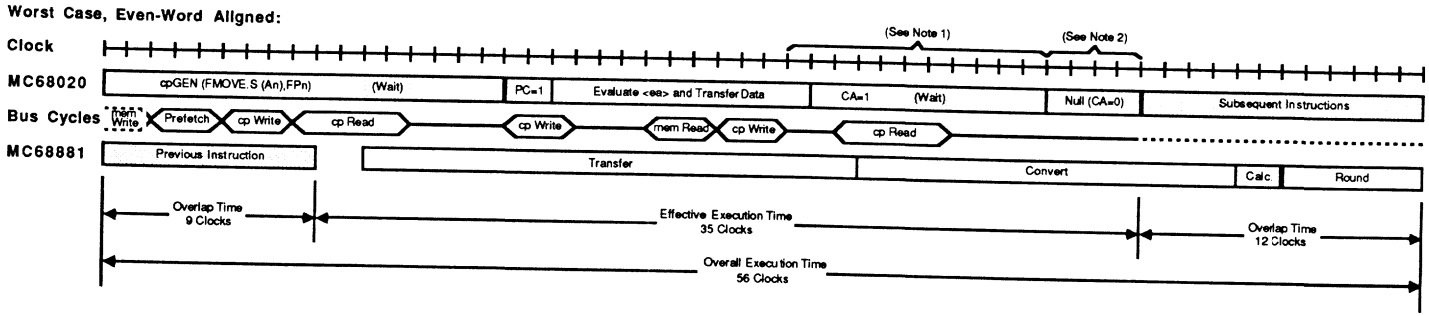
The diagram in Figure 6–6 illustrates the execution of the FMOVE.S (An),FPn instruction where the instruction is even word aligned, the MC68020 cache is disabled, and at least one of the arithmetic exceptions is enabled. Under these conditions, the cpGEN start-up operation is identical to the first diagram in Figure 6–5, except that the primitive returned by the MC68881 is evaluate effective address and transfer data with the PC and CA bits set. Thus, the first operation performed by the MC68020 while processing this primitive is to pass the program counter; which adds two clock cycles to both the effective and overall execution times (note that the third clock cycle of the coprocessor write cycle overlaps with the effective address calculation). The MC68020 then evaluates the effective address, (An), which requires two clock cycles, and transfers the 32–bit single precision operand from memory. The come again operation is then performed, which requires 10 clock cycles, followed by a four clock period during which the null (CA=0) primitive is processed.

The MC68881 does not start the input conversion operation until the single precision operand is internally passed to the execution unit. The MC68881 bus interface unit requires 3 clocks from the end of the operand write cycle to transfer the operand to the execution unit; thus the conversion does not begin until 3 clock cycles after the end of the write cycle. This three clock cycle transfer operation and part of the conversion operation occur simultaneously with the completion of the CA=1 and null processing by the MC68020. Thus, 15 clock cycles of the MC68020 effective execution time do not contribute to the overall execution time for the instruction.

The previous four examples are intended to clarify the meaning of the detailed execution timing tables that follow. The only difference between the FMOVE instruction examples presented and any of the monadic or dyadic instructions is that the convert and calculate times are different (the round time is also different if an exception occurs). Also, the effective address calculation and operand transfer times are different. Notice that the timing prior to the start of the conversion operation is almost entirely dependent on the execution characteristics of the main processor, while the timing for the rest of the instruction is dependent solely on the MC68881. This distinction is useful when the execution timing for a main processor other than the MC68020 is to be determined.

NOTE

The term "not normalized" is used frequently in the following tables. This term is used where conditions allow the input of a denormalized or unnormalized number; and the term denormalized is used where only a denormalized input is possible. Refer to **2.4.2 Denormalized Numbers** for a description of the denormalized and unnormalized data types.



Notes: 1) These eleven clocks do not add to the overall execution time for the instruction, only the effective execution time for the MC68020.
2) This operation does not add to the overall execution time for the instruction, only the effective execution time for the MC68020.

Figure 6-6. Instruction Overlap Example — FMOVE.S (An),FPn

6.5.2.1 INSTRUCTION START-UP. When the MC68020 encounters an MC68881 instruction, it decodes the type of the coprocessor instruction and then initiates communications with the MC68881 via the appropriate coprocessor interface bus cycle. The following table gives the execution timing of the MC68020 for the start-up phase of each of the coprocessor instruction types. For the general instruction type, the start-up time includes the command CIR write and response CIR read cycles that initiate the instruction dialog between the MC68020 and the MC68881. For the conditional instruction types, the start-up time includes the condition CIR write and response CIR read cycles.

For the FSAVE instruction, the start-up time includes the read of the save CIR and the write of the format word to memory. For the FRESTORE instruction, the start-up time includes the read of the format word from memory, the write of the restore CIR, and the read of the restore CIR to validate the format word. The effective address calculation time is not included for the FSAVE and FRESTORE instructions; the appropriate values must be obtained from the calculate effective address table and added to the start-up values for these instructions.

If an enabled pre-instruction exception is pending when the MC68020 attempts to initiate an MC68881 instruction, the instruction start-up operation is performed for the general or conditional instruction types, and then the MC68020 proceeds to perform exception processing (at the request of the MC68881). In this case, the start-up timing numbers are added to the values from the exception processing tables to determine the time required to begin execution of the exception handler.

Instruction Type	Best Case	Cache Case	Worst Case
General*	12/6 (0/0/0/1/1)	14/6 (0/0/0/1/1)	17/9 (1/0/0/1/1)
FBcc	12/6 (0/0/0/1/1)	14/6 (0/0/0/1/1)	14/6 (0/0/0/1/1)
FDBcc, FScc and FTRAPcc	12/6 (0/0/0/1/1)	14/6 (0/0/0/1/1)	17/9 (1/0/0/1/1)
FSAVE**	13/1 (0/0/1/1/0)	15/1 (0/0/1/1/0)	15/1 (0/0/1/1/0)
FRESTORE**	16/4** (0/1/0/1/1)	18/4** (0/1/0/1/1)	18/4** (0/1/0/1/1)

* These execution time numbers represent the overall execution time for this operation with respect to the MC68020, and therefore are used to calculate the *effective* execution time of the instruction. However, 6 clock cycles always overlap with the execution of a register-to-register instruction (OPCLASS 000) by the MC68881, and therefore should not be included in the calculation to generate the *overall* execution time.

** Add the appropriate effective address calculation time. Note that the overlap time available for the FRESTORE instruction is of little use, since this operation destroys the previous context of the MC68881.

The MC68020 terminates all instructions except FSAVE and FRESTORE by processing a null (CA=0) primitive (unless a mid-instruction exception occurs). Therefore, the following timing values should be included in the calculation of the effective execution time for the MC68020, where appropriate.

Primitive Type	Best Case	Cache Case	Worst Case
Null (CA=0) with no tracing	4/4* (0/0/0/0/0)	4/4* (0/0/0/0/0)	5/5* (1/0/0/0/0)

* Overlap is allowed for register-to-register and external-to-register instructions only (OPCLASS 000 and 010).

6.5.2.2 TRANSFER OPERAND. The following tables give the timing for the transfer of an operand to or from the MC68881 by the MC68020. Two tables are given, one for external source or destination operands that reside in an MC68020 register or in memory, and one for immediate source operands. For input transfers, the timing numbers given include the time required by the MC68020 to process the evaluate effective address and transfer data (with CA=1) primitive, and for the MC68881 to perform the internal transfer of the operand to the execution unit. For the MC68020, the last clock cycle of the transfer operation and the processing for CA=1 always overlaps with the input operand transfer and conversion operations by the MC68881, and therefore are not added to the overall execution time for the instruction (although these operations are included in the calculation of the effective execution time for the MC68020).

For output operand transfers, the timing numbers include the processing for the evaluate effective address and transfer data primitive (with CA=1). Since no overlap occurs during an output transfer, the values below are used directly in the overall execution time calculation. Note that the bus cycle activity numbers include the read of the evaluate effective address and transfer data primitive at the end of the conversion (even though the execution time for that operation is not included). This is due to the fact that null (CA=1, IA=1) primitives are read during the instruction start-up operation and while waiting for the conversion to complete, and the evaluate effective address and transfer data primitive is read during the processing of one of those primitives.

In order to calculate the effective execution time for the MC68020 for either input or output transfers, the processing time for the null (CA=0) primitive that terminates the dialog must be included. For output conversions that cause an enabled exception, the take mid-instruction exception primitive is returned after the operand transfer is complete. In this case, the appropriate exception processing execution time values must be included in lieu of the null (CA=0) processing time in the calculation of the overall execution time.

Transfer Type	Operand Format				
	Byte	Word	Long, Single	Double	Ext., Packed
From MC68020 Dn	14/0 (0/0/0/1/1)	14/0 (0/0/0/1/1)	14/0 (0/0/0/1/1)	—	—
From Memory*	19/0 (0/1/0/1/1)	19/0 (0/1/0/1/1)	19/0 (0/1/0/1/1)	25/0 (0/2/0/1/2)	31/0 (0/3/0/1/3)
To MC68020 Dn	17/0 (0/0/0/3/0)	17/0 (0/0/0/3/0)	17/0 (0/0/0/3/0)	—	—
To Memory**	19/0 (0/0/1/3/0)	19/0 (0/0/1/3/0)	19/0 (0/0/1/3/0)	25/0 (0/0/2/4/0)	31/0 (0/0/3/5/0)

* Add the appropriate effective address calculation time. Eleven clocks of the MC68020 processing overlap with execution by the MC68881, which requires 5 or 3 clock cycles after the last coprocessor write cycle to complete the internal transfer for double or any other format, respectively. Thus, reduce the numbers above by 6 clocks for double or 8 clocks for any other format for calculation of the overall execution time.

** Add the appropriate effective address calculation time. If the destination is packed decimal and a dynamic k-factor is used, add 14/0 (0/0/0/1/1).

Immediate Operand Format	Best Case	Cache Case	Worst Case
Byte, Word	14/0 (0/0/0/1/1)	14/0 (0/0/0/1/1)	17/0 (1/0/0/1/1)
Long, Single	18/0 (0/0/0/1/1)	18/0 (0/0/0/1/1)	19/0 (1/0/0/1/1)
Double	22/0 (0/0/0/1/2)	22/0 (0/0/0/1/2)	24/0 (2/0/0/1/2)
Extended, Packed	26/0 (0/0/0/1/3)	26/0 (0/0/0/1/3)	30/0 (3/0/0/1/3)

6.5.2.3 INPUT OPERAND CONVERSION. All MC68881 instructions that require an input operand execute an implied conversion to the 80-bit extended precision format that is used internally. The amount of time required to perform this conversion depends on the format, value, and type of the input operand. The following tables give the amount of time required to convert an input operand to the internal data format, starting from the end of the internal operand transfer after the last write cycle to the operand CIR.

For dyadic operations, one table for conversions from each combination of source data format and type versus destination data type is included. For monadic operations, one table includes the conversion timing for any data format and type. Only one number is given in each entry, since the total number of clock cycles required is equal to the number of overlap allowed clock cycles, and no bus cycles are generated during this stage of an instruction (since the MC68881 does not require any further services of the MC68020 once this stage of an instruction starts).

Dyadic Input Conversions — Source Operand is Byte, Word or Long:

Destination \ Source	Normalized +	Normalized -	Not Normalized	Zero	Infinity	NAN
Normalized	24	26	—	22	—	—
Unnormalized	36	38	—	34	—	—
Zero	30	32	—	28	—	—
Infinity	28	30	—	26	—	—
NAN	30	32	—	28	—	—

Dyadic Input Conversions — Source Operand is Single Precision:

Destination \ Source	Normalized	Not Normalized	Zero	Infinity	NAN
Normalized	18	36	22	24	26
Unnormalized	30	48	34	36	38
Zero	24	42	28	30	32
Infinity	22	40	26	28	30
NAN	24	42	28	30	32

Dyadic Input Conversions — Source Operand is Double Precision:

Destination \ Source	Normalized	Not Normalized	Zero	Infinity	NAN
Normalized	16	34	20	22	24
Unnormalized	28	46	32	34	36
Zero	22	40	26	28	30
Infinity	20	38	24	26	28
NAN	22	40	26	28	30

Dyadic Input Conversions — Source Operand is Extended Precision:

Destination \ Source	Normalized	Not Normalized	Zero	Infinity	NAN
Normalized	10	26	12	12	14
Unnormalized	22	38	24	24	26
Zero	16	32	18	18	20
Infinity	14	30	16	16	18
NAN	16	32	18	18	20

Monadic or Dyadic Input Conversions — Source Operand is Packed Decimal:

Destination \ Source	Normalized	Not Normalized	Zero	Infinity	NAN
Normalized	~822	~822	22	22	24
Unnormalized	~848	~848	34	34	36
Zero	~842	~842	28	28	30
Infinity	~840	~840	26	26	28
NAN	~842	~842	28	28	30

~ indicates a typical conversion time. The minimum and the maximum conversion times are ??? and 954 clock cycles, respectively.

Monadic or Dyadic Input Conversions — Source Operand is FPm:

Destination \ Source	Normalized	Not Normalized	Zero	Infinity	NAN
Normalized	14	30	16	16	18
Unnormalized	26	42	28	28	30
Zero	20	36	22	22	24
Infinity	18	34	20	20	22
NAN	20	36	22	22	24

Monadic Input Conversions — Source Operand is in Memory:

Format \ Type	Normalized +	Normalized -	Not Normalized	Zero	Infinity	NAN
Byte, Word, Long	22	24	—	20	—	—
Single	16		30	20	24	24
Double	14		28	18	22	22
Extended	8		20	10	12	12

6.5.2.4 ARITHMETIC CALCULATION. The following tables give the time required to perform any of the general purpose arithmetic operations supported by the MC68881. One table is given for each dyadic instruction with respect to the combination of input operand data types; and one table is given for all of the monadic operations. Each entry in these tables includes the time from the end of the input operand conversion to when the calculation is complete. Only one number is given for each entry, since no bus cycles are generated during this stage of an instruction. Also, the total number of clock cycles required for the calculation is equal to the number of overlap allowed clock cycles, since the MC68881 does not require any further services of the MC68020 once this stage of an instruction starts.

Some entries contain a reference to a footnote that contains more detailed timing information for an operation (e.g., the table for addition contains an entry that references the ADD footnote, which contains three numbers, based on the the input operands). Furthermore, in some cases, an entry references another table that contains the execution time required to handle certain input operands. For example, if an entry contains NAN1, refer to the entry of the same name in **6.5.2.9 EXCEPTION PROCESSING**.

If an entry below is appended with a plus sign (+), then the appropriate timing numbers from the rounding and exception handling table (in section **6.5.2.9 EXCEPTION PROCESSING**) must be used to calculate the overall execution time for an instruction. Otherwise, the numbers from the following tables include the time to handle exceptional operand cases and produce the final result.

FADD Calculation Time:

Source / Destination	Normalized +	Normalized -	Zero +	Zero -	Infinity +	Infinity -	NAN +	NAN -
Normalized +	ADD		2+		6		NAN2	
Normalized -								
Zero +	2+		6	26	6		NAN2	
Zero -			26	6				
Infinity +	6		6		6	20	NAN2	
Infinity -					20	6		
NAN +	NAN1		NAN2		NAN2		NAN3	
NAN -								

ADD:
- 18+ if the source and destination exponents are equal, and the source mantissa is less than the destination mantissa.
- 20+ if the source and destination exponents are equal, and the source mantissa is greater than or equal to the destination mantissa.
- 22+ if the source and destination exponents are not equal.

FCMP Calculation Time:

Destination \ Source	Normalized +	Normalized -	Zero +	Zero -	Infinity +	Infinity -	NAN +	NAN -
Normalized +	CMP	6	6		8	6	NAN4	
Normalized -	6	CMP			6	8		
Zero +	8	6	6		8	6	NAN4	
Zero -	6	8			6	8		
Infinity + -	6		6		6		NAN4	
NAN + -	NAN1		NAN2		NAN2		NAN3	

CMP: 8 if the source exponent is greater than the destination exponent.
10 if the source exponent is less than or equal to the destination exponent.

FDIV CalculationTime:

Destination \ Source	Normalized +	Normalized -	Zero +	Zero -	Infinity +	Infinity -	NAN +	NAN -
Normalized + -	DIV		20		6	8	NAN2	
					8	6		
Zero + -	6	8	20		6	8	NAN2	
Infinity + -	6	8	6	8	20		NAN2	
NAN + -	NAN1		NAN2		NAN2		NAN3	

DIV: 72+ if the intermediate result is normalized.
74+ if the intermediate result is denormalized.

FMOD Calculation Time:

Destination \ Source	Normalized +	Normalized -	Zero +	Zero -	Infinity +	Infinity -	NAN +	NAN -
Normalized + -	MOD		20		6+		NAN2	
Zero + -	6+		20		6+		NAN2	
Infinity + -	IOP		20		20		NAN2	
NAN + -	NAN1		NAN2		NAN2		NAN3	

MOD: 12+ if the quotient is zero; else:

$$\left(36 + 70\left(\mathrm{INT}\left(\frac{1 + \text{destination exponent} - \text{source exponent}}{64}\right)\right)\right) +$$

FMUL Calculation Time:

Source / Destination	Normalized +	Normalized -	Zero +	Zero -	Infinity +	Infinity -	NAN +	NAN -
Normalized +	MUL		6		6		NAN2	
Normalized -			8		8			
Zero +	6	8	6	8	20		NAN2	
Zero -								
Infinity +	6	8	20		6	8	NAN2	
Infinity -								
NAN +	NAN1		NAN2		NAN2		NAN3	
NAN -								

MUL: 40+ if the intermediate result is normalized.
42+ if the intermediate result is not normalized.

FREM Calculation Time:

Source / Destination	Normalized +	Normalized -	Zero +	Zero -	Infinity +	Infinity -	NAN +	NAN -
Normalized +	REM		20		6+		NAN2	
Normalized -								
Zero +	6+		20		6+		NAN2	
Zero -								
Infinity +	IOP		20		20		NAN2	
Infinity -								
NAN +	NAN1		NAN2		NAN2		NAN3	
NAN -								

REM: 12+ if the quotient is zero; else:

$$\left(36 + 70\left(\mathrm{INT}\left(\frac{1 + \text{destination exponent} - \text{source exponent}}{64}\right)\right)\right)+$$

FSCALE Calculation Time:

Source / Destination	Normalized +	Normalized -	Zero +	Zero -	Infinity +	Infinity -	NAN +	NAN -
Normalized +	SCALE		6+		20		NAN2	
Normalized -								
Zero +	6		6		20		NAN2	
Zero -								
Infinity +	6		6		20		NAN2	
Infinity -								
NAN +	NAN1		NAN2		NAN2		NAN3	
NAN -								

SCALE: 6+ if the source exponent (unbiased) is less than zero.
10+ if the source exponent (unbiased) is in the range [0...15].
14+ if the source exponent (unbiased) is greater than 15.

FSGLDIV Calculation Time:

Source / Destination	Normalized +	Normalized -	Zero +	Zero -	Infinity +	Infinity -	NAN +	NAN -
Normalized +	SGLDIV		20		6	8	NAN2	
Normalized -					8	6		
Zero +	6	8	20		6	8	NAN2	
Zero -								
Infinity +	6	8	6	8	20		NAN2	
Infinity -								
NAN +	NAN1		NAN2		NAN2		NAN3	
NAN -								

SGLDIV:
- 44 if no extended precision underflow or overflow occurs.
- 62 if an extended precision overflow occurs.
- 90 if an extended precision underflow occurs.

FSGLMUL Calculation:

Source / Destination	Normalized +	Normalized -	Zero +	Zero -	Infinity +	Infinity -	NAN +	NAN -
Normalized +	SGLMUL		6		6		NAN2	
Normalized -			8		8			
Zero +	6	8	6	8	20		NAN2	
Zero -								
Infinity +	6	8	20		6	8	NAN2	
Infinity -								
NAN +	NAN1		NAN2		NAN2		NAN3	
NAN -								

SGLMUL:
- 34 if no extended precision underflow or overflow occurs.
- 52 if an extended precision overflow occurs.
- 80 if an extended precision underflow occurs.

FSUB Calculation:

Source / Destination	Normalized +	Normalized -	Zero +	Zero -	Infinity +	Infinity -	NAN +	NAN -
Normalized +	SUB		2+		8		NAN2	
Normalized -								
Zero +	4+		26	8	8		NAN2	
Zero -			8	26				
Infinity +	6		6		20	8	NAN2	
Infinity -					8	20		
NAN +	NAN1		NAN2		NAN2		NAN3	
NAN -								

SUB:
- 18+ if the source and destination exponents are equal, and the source mantissa is less than the destination mantissa.
- 20+ if the source and destination exponents are equal, and the source mantissa is greater than or equal to the destination mantissa.
- 22+ if the source and destination exponents are not equal.

Monadic Calculations:

Operation \ Source	Normalized +	Normalized -	Zero +	Zero -	Infinity +	Infinity -	NAN +	NAN -
FABS	4+		4+		8		NAN2	
FACOS	594+		121		20		NAN2	
			202					
FASIN	550+		6		20		NAN2	
FATAN	372+		6		121	141	NAN2	
					202	222	NAN2	
FATANH	662+		6		20		NAN2	
FCOS	360+3		8		20		NAN2	
FCOSH	576+		8		8		NAN2	
FETOX	466+		8		6		NAN2	
FETOXM1	514+		6		6	8	NAN2	
FGETEXP	exponent = 0: 16		6		20		NAN2	
	exponent > 0: 20							
	exponent < 0: 22							
FGETMAN	6		6		20		NAN2	
FINT, FINTRZ	fraction = 0: 8		6		6		NAN2	
	fraction ≠ 0: 30							
	result = 0: 28							
FLOGN	494+	IOP	22		6	20	NAN2	
FLOGNP1	540+4		6		6	20	NAN2	
FLOG10	550+	IOP	22		6	20	NAN2	
FLOG2	550+	IOP	22		6	20	NAN2	
FMOVE to FPn	2+		6		6		NAN2	
FMOVECR	181		—		—		—	
	262							
FNEG	4+		4+		8		NAN2	
FSIN	360+3		6		20		NAN2	
FSINCOS	420+3		20		26		NAN6	
FSINH	656+		6		6		NAN2	
FSQRT	76+	IOP	6		6	20	NAN2	
FTAN	442+3		6		20		NAN2	
FTANH	630+		6		8		NAN2	
FTENTOX	536+		8		6		NAN2	
FTST	8		8		8		NAN5	
FTWOTOX	536+		8		6		NAN2	

NOTES:
1. If the extended precision rounding mode is used.
2. If the single or double precision rounding mode is used.
3. This assumes that the source operand is in the range [-π...+π]. If the source operand is outside of that range, the appropriate REM calculation time required to perform the argument reduction must be added to this value.
4. If the source operand is less than or equal to -1, use the IOP time.

6.5.2.5 OUTPUT OPERAND CONVERSION. The FMOVE.<fmt> FPn,<ea> instruction performs an implicit conversion from the 80-bit extended precision format used internally by the MC68881 to an external data format. The first table gives the conversion times for most output operations. Since the execution timing for conversions from the internal extended precision format to either single or double precision is highly data dependent, a second table is used to present the timing for these operations (for in-range, non-zero input values).

The amount of time required to perform this conversion depends on the value and type of the input operand and the format of the desired output. The values given in the following tables, in MC68881 clock cycles, include the time from the fourth clock cycle of the first response CIR read (which returns a null (CA=1, IA=1) primitive) to when the conversion is complete (when a read of the response CIR returns an evaluate effective address and transfer operand primitive). Only one number is given for each entry, since no bus cycles are generated during this stage of an instruction. Also, the total number of clock cycles required for operand conversion is equal to the number of overlap allowed clock cycles (during which time interrupts may be handled; normal program execution is not allowed), since the MC68881 does not require any services of the MC68020 during this stage of an instruction.

Source Type / Dest. Format	Normalized +	Normalized -	Not Normalized +	Not Normalized -	Zero +	Zero -	Infinity +	Infinity -	NAN +	NAN -
Integer, No Overflow	50	52	60	62	18		—		24	
Integer, Overflow	52	56	62	66	18		24	26	24	
Single	(see below)		(see below)		16		18		NAN7	
Double	(see below)		(see below)		16		18		NAN7	
Extended	18		(see note 1)		16		16		NAN7	
Packed	(see note 2)		(see note 2)		24		24		NAN2	

NOTES: 1. 26 clocks if the source operand is an unnormalized number.
56 clocks if the source operand is a denormalized number.
2. 1942 clocks is the typical time required for the conversion, if no overflow occurs. The minimum and maximum times are ??? and 3674 clocks, respectively.

Source Type / Conversion Result	Normalized	Not Normalized
No Underflow, Overflow or Round Overflow	38	48
No Underflow or Overflow; Round Overflow	42	52
Overflow; RN or RZ Mode; No Round Overflow	44	54
Overflow; RN or RZ Mode; Round Overflow	48	58
Overflow; RM or RP Mode; No Round Overflow	46	56
Overflow; RM or RP Mode; Round Overflow	50	60
Underflow; No Round Overflow	66	76
Underflow; Round Overflow	70	80

6.5.2.6 ROUNDING AND EXCEPTION HANDLING. Two tables are given below that contain the execution times for various exception handling operations. For the typical execution time tables given previously, it is assumed that the MC68881 uses the default operating mode of round to extended precision, and no overflow or underflow exceptions occur. If this is not the case, the round/store phase of most arithmetic instructions takes longer to execute. The entries in the typical execution time tables include the processing time for no underflow, overflow or round overflow as indicated in the table below.

The first table below indicates the number of clock cycles that should be added in the calculation of the execution time for an arithmetic instruction (both the total and the overlap allowed numbers) to account for the various rounding precision and exception handling combinations. The entries in the first table below include the time from the end of the calculation phase to when the MC68881 completes instruction execution (i.e., when the PF bit in the null (CA=0) primitive is clear if the response CIR is read).

The second table below includes the entries referenced previously in the arithmetic calculation and output operand conversion tables for exceptional operand inputs. The values in this table are used for the calculation or conversion timing in lieu of a value from the appropriate table. For example, if an output operand conversion table entry references NAN7, then the timing number from the NAN7 entry below is used as the conversion time value.

Rounding Precision	Result	Clock Cycles
Extended	No Underflow, Overflow or Round Overflow	6
	No Underflow or Overflow; Round Overflow	6
	Underflow	34
	Overflow; RN or RZ Mode; No Round Overflow	14
	Overflow; RM or RP Mode; No Round Overflow	16
	Round Overflow (Not Caused By Rounding); RN or RZ Mode	16
	Round Overflow (Not Caused By Rounding); RM or RP Mode	18
	Round Overflow (Caused By Rounding); RN or RZ Mode	20
	Round Overflow (Caused By Rounding); RM or RP Mode	22
Single or Double	Result is Zero	6
	No Underflow, Overflow or Round Overflow	24
	No Underflow or Overflow; Round Overflow	28
	Underflow; No Round Overflow	56
	Underflow; Round Overflow	60
	Overflow; RN or RZ Mode; No Round Overflow	30
	Overflow; RM or RP Mode; No Round Overflow	32
	Overflow; RN or RZ Mode; Round Overflow	34
	Overflow; RM or RP Mode; Round Overflow	36

Exception Identifier	Conditions	Clock Cycles
IOP	Source Operand is Not Denormalized	20
	Source Operand is Denormalized	32
NAN1	Destination is a QNAN, Source is not Denormalized	28
	Destination is a QNAN, Source is Denormalized	52
	Destination is an SNAN, Source is not Denormalized	30
	Destination is an SNAN, Source is Denormalized	54
NAN2	The NAN is a QNAN	28
	The NAN is an SNAN	30
NAN3	Both NANs are QNANs	28
	Source is a QNAN, Destination is an SNAN	30
	Source is an SNAN, Destination is a QNAN	32
	Both NANs are SNANs	30
NAN4	The NAN is a QNAN	30
	The NAN is an SNAN	32
NAN5	The NAN is a QNAN	8
	The NAN is an SNAN	10
NAN6	The NAN is a QNAN	38
	The NAN is an SNAN	40
NAN7	The NAN is a QNAN	22
	The NAN is an SNAN	24

Where QNAN refers to a non-signalling not-a-number.

6.5.2.7 CONDITIONAL TERMINATION. The effective execution time for the conditional and context switch instructions is not heavily dependent on the MC68881, since the execution of these operations is performed, for the most part, by the MC68020. In order to calculate the effective execution time for these instructions, the following table gives the termination timing for the MC68020. The termination processing starts four MC68020 clock cycles after the end of the response CIR read and ends when the MC68020 begins execution of the next instruction. Note that the allowed overlap time in this table is always zero, since the MC68881 is in the idle state when these instructions reach the termination phase. However, if multiple coprocessors are used in a system, the execution of other coprocessors may overlap with the execution of these instructions.

In order to determine the execution time for a conditional operation performed by a processor other than an MC68020, it is necessary to know the timing for the conditional evaluation by the MC68881. This value is given below (in MC68881 clock cycles) and indicates the best case time from the start of the condition CIR write to the end of the response CIR read (which are the only two coprocessor accesses required).

MC68881 Conditional Test Evaluation Time: 8/0 (0/0/0/1/1)

Instruction Type		Best Case	Cache Case	Worst Case
FBcc.W	Branch Taken	6/0 (0/0/0/0/0)	6/0 (0/0/0/0/0)	9/0 (2/0/0/0/0)
	Branch Not Taken	4/0 (0/0/0/0/0)	4/0 (0/0/0/0/0)	5/0 (1/0/0/0/0)
FBcc.L	Branch Taken	6/0 (0/0/0/0/0)	6/0 (0/0/0/0/0)	9/0 (2/0/0/0/0)
	Branch Not Taken	4/0 (0/0/0/0/0)	4/0 (0/0/0/0/0)	7/0 (2/0/0/0/0)
FDBcc	True, Not Taken	6/0 (0/0/0/0/0)	6/0 (0/0/0/0/0)	7/0 (1/0/0/0/0)
	False, Not Taken	10/0 (0/0/0/0/0)	10/0 (0/0/0/0/0)	15/0 (3/0/0/0/0)
	False, Taken	6/0 (0/0/0/0/0)	6/0 (0/0/0/0/0)	9/0 (2/0/0/0/0)
FScc	Dn	4/0 (0/0/0/0/0)	4/0 (0/0/0/0/0)	4/0 (1/0/0/0/0)
	(An)+ or –(An)*	6/0 (0/0/1/0/0)	8/0 (0/0/1/0/0)	8/0 (1/0/1/0/0)
	Memory**	4/0 (0/0/1/0/0)	6/0 (0/0/1/0/0)	6/0 (1/0/1/0/0)
FTRAPcc	Trap Taken	24/0 (0/1/4/0/0)	25/0 (0/1/4/0/0)	30/0 (2/1/4/0/0)
	Trap Not Taken	4/0 (0/0/0/0/0)	4/0 (0/0/0/0/0)	5/0 (1/0/0/0/0)
FTRAPcc.W	Trap Taken	26/0 (0/1/4/0/0)	27/0 (0/1/4/0/0)	28/0 (2/1/4/0/0)
	Trap Not Taken	6/0 (0/0/0/0/0)	6/0 (0/0/0/0/0)	6/0 (1/0/0/0/0)
FTRAPcc.L	Trap Taken	28/0 (0/1/4/0/0)	29/0 (0/1/4/0/0)	35/0 (3/1/4/0/0)
	Trap Not Taken	8/0 (0/0/0/0/0)	8/0 (0/0/0/0/0)	10/0 (2/0/0/0/0)

* For condition true; subtract one clock for condition false.
** Add the appropriate effective address calculation time.

6.5.2.8 MULTIPLE REGISTER TRANSFER. The following table gives the number of clock cycles and bus cycles required for the MC68020 to perform a multiple register transfer to or from the MC68881. These transfers occur during the FMOVEM instruction for either the floating-point control register or floating-point data register form of the instruction. The timing values given in the following table includes the processing time for either the evaluate effective address and transfer data or transfer multiple coprocessor registers primitive (for the control or data register form, respectively) with CA=1, assuming that the main processor is an MC68020. In order to calculate the effective execution time for the MC68020, the time required to process the null (CA=0) primitive after the transfer is complete must be included.

For the transfer of multiple control registers, the register select list is included in the instruction, and all of the selected registers are transferred as a single operand (from the perspective of the main processor). For the transfer of multiple data registers, the MC68020 must read the register select mask before starting the register transfer. The amount of time required by the MC68020 to read and process the register mask is included in the following table entries. If a dynamic register list is used, the time required by the MC68020 to process the transfer single main processor register primitive must be included and is given below.

MC68020 Transfer Dynamic Register List Time: 14/0 (0/0/0/1/1)

Transfer Type		Timing
Move Single Control Register	To an MC68020 Register	17/0 (0/0/0/2/0)
	To Memory*	19/0 (0/0/1/2/0)
	From an MC68020 Register	14/0 (0/0/0/1/1)
	From Memory	19/0 (0/1/0/1/1)
	#<data>	19/0 (1/0/0/1/1)*
Move Multiple Control Registers	To Memory	13+6n/0 (0/0/n/1+n/0)
	From Memory	13+6n/0 (0/n/0/1/n)
	#<data>	12+6n/0 (n/0/0/1/n)*
Move Multiple Data Registers	To Memory	23+25n/0 (0/0/3n/2+3n/0)
	From Memory	21+23n/0 (0/3n/0/2/3n)

n is the number of registers transferred.

* if the immediate operand resides in the MC68020 cache, the number of clock cycles is reduced by 3n and the number of instruction prefetch bus cycles is zero.

6.5.2.9 STATE FRAME TRANSFER. The following table gives the number of clock cycles and bus cycles required for the MC68020 to transfer an internal state frame to or from the MC68881. These transfers occur during the FSAVE and FRESTORE instructions. The timing values given in the following table include the time from the end of the instruction start-up operation to the end of the last operand write cycle, assuming that the main processor is an MC68020.

Before the transfer of a state frame to the MC68881 during an FRESTORE instruction, the MC68020 must read the format word from memory, write it to the restore CIR, and verify that it is valid by reading the restore CIR. Likewise, during an FSAVE instruction, the MC68020 must read the format word from the save CIR and store it in memory. The instruction start-up timing table entries include these operations for MC68020 based systems.

During an FSAVE operation, the MC68881 may require the main processor to wait until the current instruction is completed or a save boundary is reached before starting the state frame transfer. The maximum time that the main processor may be forced to wait is given below, and should be included in the calculation of the worst case FSAVE execution time.

MC68881 Maximum FSAVE Delay Time: ___

Operation	Frame Type	Timing
State Save	Idle	36/0 (0/0/6/6/0)
	Busy	270/0 (0/0/45/45/0)
State Restore	Idle	36/0 (0/6/0/0/6)
	Busy	270/0 (0/45/0/0/45)

In order to calculate overall execution time for the MC68020 during an FSAVE or FRESTORE instruction, the instruction termination processing time must be included. The following table gives the timing values for this processing, which is from the end of the last operand write cycle to when the MC68020 begins execution of the next instruction.

Instruction Type	Best Case	Cache Case	Worst Case
FSAVE	1/0 (0/0/0/0/0)	1/0 (0/0/0/0/0)	3/0 (1/0/0/0/0)
FRESTORE	3/0 (0/0/0/0/0)	3/0 (0/0/0/0/0)	4/0 (1/0/0/0/0)

6.5.2.10 EXCEPTION PROCESSING. The following table indicates the time required for exception processing related to the execution of MC68881 instructions. For the second and third entries, the values in the table indicate the time required to process the take exception primitive to when the MC68020 resumes normal instruction execution in the appropriate exception handler.

To determine the overall exception latency for a pre-instruction exception, the instruction start-up time (for the arithmetic or conditional instruction that is preempted by the exception) is added to the exception processing time from the table below. The exception processing time for a take mid-instruction exception primitive is added to the overall execution time for the FMOVE to memory instruction that caused the exception. For conditional instructions that cause a BSUN exception, the pass program counter time, given below, is also added to the instruction start-up and exception processing time to calculate the overall exception latency for the instruction.

For the take interrupt operations, the values in the table below include the time from the end of the processing of a response primitive that allows interrupts to when the MC68020 resumes normal instruction execution in the interrupt handler (the possible responses are the null (CA=1, IA=1) and null (CA=0, IA=1, PF=0) primitives, or the not ready format code). If an interrupt is processed during an FMOVE to memory instruction or when the main processor is in the trace mode and receives a null (CA=0, IA=1, PF=0) primitive, a mid-instruction stack frame is used. A pre-instruction stack frame is used for interrupts processed during an FSAVE instruction. The M-stack and I-stack designation indicates whether the the M bit of the MC68020 status register was set or clear, respectively, before the interrupt occurred.

The processing time for an FRESTORE format error includes the time from the end of the FRESTORE start-up operation to when the MC68020 resumes nomal instruction execution in the format error exception handler. Since the characteristics of an FSAVE format error exception are not predictable (and since such an occurance is catastophic), execution timing required to handle such an error is not included in the table below.

The entries in the table for the return from exception (RTE) instruction include the time from when the MC68020 begins execution of the RTE to when the previously aborted operation is resumed. If the RTE instruction processes a pre-instruction frame, the time in the table includes the time required to restore the processor context and prepare to execute the instruction at the address contained in the stack frame program counter image. For the mid-instruction frame, the time in the table includes the time required to restore the processor context and read the response CIR to continue the previously suspended operation. The "RTE, throwaway frame" entries include the time required to read and process the throwaway stack frame (normally from the top of the interrupt stack) and then perform RTE processing for the stack frame on top of the resulting active stack (normally either the master or user stack). Thus, if the MC68020 must return from an interrupt that occurred while the M bit in the MC68020 status register was set, a throwaway frame is first processed from the interrupt stack, followed by the processing of the appropriate frame from the master stack (which returns the processor to the context saved by the interrupt processing). For such a case, the "RTE, throwaway frame" times are added to the RTE execution times for the second stack frame to derive the overall execution times for the operation.

In addition to the occurrence of an exception, whether exceptions are enabled or not can also affect instruction execution time. This is due to the fact that the MC68881 requests the transfer of the program counter at the start of any arithmetic instruction if any exception (other than the BSUN exception) is enabled. If the source operand resides in a floating-point data register, the transfer of the PC does not affect overall execution timing, since it takes place concurrently with the execution of the operation by the MC68881. However, for source operands external to the MC68881, the MC68020 first passes the PC, and then passes the operand; thus, execution time is affected for this case. The time required by the MC68020 to pass the program counter is given below, and should be added to the instruction execution time where appropriate.

Operation	Best Case	Cache Case	Worst Case
Pass Program Counter	2/2* (0/0/0/0/1)	3/3* (0/0/0/0/1)	3/3* (0/0/0/0/1)
Take Pre-Instruction Exception	22/0 (0/1/4/0/0)	22/0 (0/1/4/0/0)	24/0 (2/1/4/0/0)
Take Mid-Instruction Exception	32/0 (0/1/7/0/0)	32/0 (0/1/7/0/0)	38/0 (2/1/7/0/0)
Process Pre-Instruction Interrupt (I-stack)	26/26 (0/2/4/0/0)	26/26 (0/2/4/0/0)	33/33 (2/2/4/0/0)
Process Pre-Instruction Interrupt (M-stack)	41/41 (0/2/8/0/0)	41/41 (0/2/8/0/0)	48/48 (2/2/8/0/0)
Process Mid-Instruction Interrupt (I-stack)	35/35 (0/2/6/0/0)	36/36 (0/2/6/0/0)	42/42 (2/2/6/0/0)
Process Mid-Instruction Interrupt (M-stack)	46/46 (0/2/9/0/0)	47/47 (0/2/9/0/0)	53/53 (2/2/9/0/0)
Process FSAVE Interrupt (I-stack)	26/26 (0/2/4/0/0)	26/26 (0/2/4/0/0)	33/33 (2/2/4/0/0)
Process FSAVE Interrupt (M-stack)	41/41 (0/2/8/0/0)	41/41 (0/2/8/0/0)	48/48 (2/2/8/0/0)
Format Error, FRESTORE Instruction	23/0 (0/1/4/0/0)	24/0 (0/1/4/0/0)	29/0 (2/1/4/0/0)
RTE, Pre-Instruction Frame	20/20 (0/4/0/0/0)	21/21 (0/4/0/0/0)	24/24 (2/4/0/0/0)
RTE, Mid-Instruction Frame	31/24 (0/6/0/1/0)	32/25 (0/6/0/1/0)	33/26 (1/6/0/1/0)
RTE, Throwaway Frame	15/15 (0/4/0/0/0)	16/16 (0/4/0/0/0)	19/19 (0/4/0/0/0)

* Overlap is allowed only for floating-point register-to-register and register-to-external operations.

SECTION 7
FUNCTIONAL SIGNAL DESCRIPTIONS

This section contains a brief description of the input and output signals for the MC68881 floating-point coprocessor. The signals are functionally organized into groups as shown in Figure 7-1

NOTE

The terms assertion and negation are used extensively. This is done to avoid confusion when describing "active-low" and "active-high" signals. The term assert or assertion is used to indicate that a signal is active or true, independent of whether that level is represented by a high or low voltage. The term negate or negation is used to indicate that a signal is inactive or false.

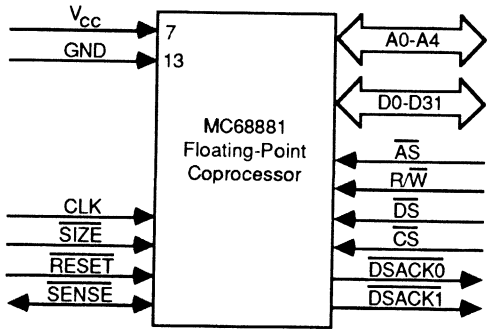


Figure 7-1. MC68881 Input/Output Signals

7.1 ADDRESS BUS (A0 through A4)

These active-high address line inputs are used by the main processor to select the coprocessor interface register locations located in the CPU address space. These lines control the register selection as listed in Table 7-1.

Table 7-1. Coprocessor Interface Register Selection

A4-A0	Offset	Width	Type	Register
0000x	$00	16	Read	Response
0001x	$02	16	Write	Control
0010x	$04	16	Read	Save
0011x	$06	16	R/$\overline{W}$	Restore
0100x	$08	16	—	(Reserved)
0101x	$0A	16	Write	Command
0110x	$0C	16	—	Reserved)
0111x	$0E	16	Write	Condition
100xx	$10	32	R/$\overline{W}$	Operand
1010x	$14	16	Read	Register Select
1011x	$16	16	—	(Reserved)
110xx	$18	32	R/$\overline{W}$	Instruction Address
111xx	$1C	32	R/$\overline{W}$	Operand Address

When the MC68881 is configured to operate over an 8-bit data bus, the A0 pin is used as an address signal for byte accesses of the coprocessor interface registers. When the MC68881 is configured to operate over a 16- or 32-bit system data bus, both the A0 and $\overline{SIZE}$ pins are strapped high and/or low as listed in Table 7-2.

Table 7-2. System Data Bus Size Configuration

A0	$\overline{SIZE}$	Data Bus
--	Low	8-Bit
Low	High	16-Bit
High	High	32-Bit

7.2 DATA BUS (D0 through D31)

This 32-bit, bidirectional, three-state bus serves as the general purpose data path between the MC68020 and the MC68881. Regardless of whether the MC68881 is operated as a coprocessor or a peripheral processor, all interprocessor transfers of instruction information, operand data, status information, and requests for service occur as standard M68000 bus cycles.

The MC68881 may be configured to operate over an 8-, 16-, or 32-bit system data bus. Depending upon the system data bus configuration, both the A0 and $\overline{SIZE}$ pins are configured specificially for the applicable bus configuration. (Refer to **7.1 ADDRESS BUS (A0 through A4)** and **7.3 SIZE ($\overline{SIZE}$)** for further details.)

7.3 SIZE ($\overline{SIZE}$)

This active-low input signal is used in conjunction with the A0 pin to configure the MC68881 for operation over an 8-, 16-, or 32-bit system data bus. When the MC68881 is configured to operate over a 16- or 32-bit system data bus, both the $\overline{SIZE}$ and A0 pins are strapped high and/or low as listed in Table 7-2.

7.4 ADDRESS STROBE ($\overline{AS}$)

This active-low input signal indicates that there is a valid address on the address bus, and both the chip select ($\overline{CS}$) and read/write (R/$\overline{W}$) signal lines are valid.

7.5 CHIP SELECT ($\overline{CS}$)

This active-low input signal enables the main processor access to the MC68881 coprocessor interface registers. When operating the MC68881 as a peripheral processor, the chip select decode is system dependent (i.e., like the chip select on any peripheral). The $\overline{CS}$ signal must be valid (either asserted or negated) when $\overline{AS}$ is asserted. Refer to **8.3 CHIP SELECT TIMING** for further discussion of timing restrictions for this signal.

7.6 READ/WRITE (R/$\overline{W}$)

This input signal indicates the direction of a bus transaction (read/write) by the main processor. A logic high (1) indicates a read from the MC68881, and a logic low (0) indicates a write to the MC68881. The R/$\overline{W}$ signal must be valid when $\overline{AS}$ is asserted.

7.7 DATA STROBE ($\overline{DS}$)

This active-low input signal indicates that there is valid data on the data bus during a write bus cycle.

7.8 DATA AND SIZE ACKNOWLEDGE ($\overline{DSACK0}$, $\overline{DSACK1}$)

These active-low, three-state output signals indicate the completion of a bus cycle to the main processor. The MC68881 asserts both the $\overline{DSACK0}$ and $\overline{DSACK1}$ signals upon assertion of $\overline{CS}$.

If the bus cycle is a main processor read, the MC68881 asserts $\overline{DSACK0}$ and $\overline{DSACK1}$ signals to indicate that the information on the data bus is valid. (Both DSACK signals may be asserted in advance of the valid data being placed on the bus.) If the bus cycle is a main processor write to the MC68881, $\overline{DSACK0}$ and $\overline{DSACK1}$ are used to acknowledge acceptance of the data by the MC68881.

The MC68881 also uses $\overline{DSACK0}$ and $\overline{DSACK1}$ signals to dynamically indicate to the MC68020 the "port" size (system data bus width) on a cycle-by-cycle basis. Depending upon which of the two DSACK pins are asserted in a given bus cycle, the MC68020 will assume data has been transferred to/from an 8-, 16-, or 32-bit wide data port. Table 7-3 lists the DSACK assertions that are used by the MC68881 for the various bus cycles over the various system data bus configurations. Refer to **8.1 BASIC TRANSFER MECHANISM OVERVIEW** for details of how the data bus is utilized by the MC68881.

Table 7-3. DSACK Assertions

Data Bus	A4	$\overline{DSACK1}$	$\overline{DSACK0}$	Comments
32-Bit	1	L	L	Valid Data on D31-D0
32-Bit	0	L	H	Valid Data on D31-D16
16-Bit	x	L	H	Valid Data on D31-D16 or D15-D0
8-Bit	x	H	L	Valid Data on D31-D24, D23-D16, D15-D8, or D7-D0
All	x	H	H	Insert Wait States in Current Bus Cycle

Table 7-3 indicates that all accesses over a 32-bit bus where A4 equals zero are to 16-bit registers. The MC68881 implements all 16-bit coprocessor interface registers on data lines D16-D31 (to eliminate the need for on-chip multiplexers); however, the MC68020 expects 16-bit registers that are located in a 32-bit port at odd word addresses (A1 = 1) to be implemented on data lines D0-D15. For accesses to these registers when configured for 32-bit bus operation, the MC68881 generates DSACK signals as listed in Table 7-3 to inform the MC68020 of valid data on D16-D31 instead of D0-D15.

An external holding resistor is required to maintain both $\overline{DSACK0}$ and $\overline{DSACK1}$ high between bus cycles. In order to reduce the signal rise time, the $\overline{DSACK0}$ and $\overline{DSACK1}$ lines are actively pulled up (negated) by the MC68881 following the rising edge of $\overline{AS}$ or $\overline{DS}$, and both DSACK lines are then three-stated (placed in the high-impedance state) to avoid interference with the next bus cycle.

7.9 RESET ($\overline{\text{RESET}}$)

This active-low input signal causes the MC68881 to initialize the floating-point data registers to non-signaling not-a-numbers (NANs) and clears the floating-point control, status, and instruction address registers.

When performing a power-up reset, external circuitry should keep the $\overline{\text{RESET}}$ line asserted for a minimum of four clock cycles after Vcc is within tolerance. This assures correct initialization of the MC68881 when power is applied. For compatibility with all M68000 Family devices, 100 milliseconds should be used as the minimum.

When performing a reset of the MC68881 after Vcc has been within tolerance for more than the initial power-up time, the $\overline{\text{RESET}}$ line must have an asserted pulse width which is greater than two clock cycles. For compatability with all M68000 Family devices, 10 clock cycles should be used as the minimum.

7.10 CLOCK (CLK)

The MC68881 clock input is a TTL-compatable signal that is internally buffered for development of the internal clock signals. The clock input must be a constant frequency square wave with no stretching or shaping techniques required. The clock should not be gated off at any time and must conform to minimum and maximum period and pulse width times.

7.11 SENSE DEVICE ($\overline{\text{SENSE}}$)

This output pin may be used optionally as an additional GND pin, or as an indicator to external hardware that the MC68881 is present in the system. This signal is internally connected to the GND of the die, but it is not necessary to connect it to the external ground for correct device operation. If a pullup resistor is connected to this pin, external hardware may sense the presence of the MC688881 in a system as described in **8.6 USE OF THE $\overline{\text{SENSE}}$ PIN**.

7.12 POWER (Vcc and GND)

These pins provide the supply voltage and system reference level for the internal circuitry of the MC68881. Care should be taken to reduce the noise level on these pins with appropriate capacitive decoupling.

7.13 NO CONNECT (NC)

One pin of the MC68881 package is designated as a no connect (NC). This pin position is reserved for future use by Motorola, and should not be used for signal routing or connected to Vcc or GND.

7.14 SIGNAL SUMMARY

Table 7-4 provides a summary of all the MC68881 signals described in this section.

Table 7-4. Signal Summary

Signal Name	Mnemonic	Input/Output	Active State	Three State
Address Bus	A0-A4	Input	High	—
Data Bus	D0-D31	Input/Output	High	Yes
Size	$\overline{SIZE}$	Input	Low	—
Address Strobe	$\overline{AS}$	Input	Low	—
Chip Select	$\overline{CS}$	Input	Low	—
Read/Write	R/$\overline{W}$	Input	High/Low	—
Data Strobe	$\overline{DS}$	Input	Low	—
Data Transfer and Size Acknowledge	$\overline{DSACK0}$, $\overline{DSACK1}$	Output	Low	Yes
Reset	$\overline{RESET}$	Input	Low	—
Clock	CLK	Input	—	—
Sense Device	$\overline{SENSE}$	Input/Output	Low	No
Power Input	V_{CC}	Input	—	—
Ground	GND	Input	—	—

SECTION 8
BUS OPERATION

This section describes the functional characteristics of the MC68881 bus interface and the mechanisms used to execute data transfers between the MC68881 and the main processor. This discussion includes a description of the functional characteristics of individual bus cycles as well as a description of the operand transfer protocols that require multiple bus cycles.

Although the MC68881 is designed primarily for use as a coprocessor to the MC68020, there are no characteristics of the bus operation that preclude the use of the MC68881 as a peripheral device with any other processor. This is due to the fact that the M68000 Family coprocessor interface utilizes standard bus cycles to transfer instructions and data between the main processor and coprocessors in a system, with no special signals required for these transfers. Because of this general purpose transfer mechanism, the type of the main processor and the nature of the system bus interface are transparent to the MC68881.

8.1 BASIC TRANSFER MECHANISM OVERVIEW

In order to execute a floating-point instruction, the MC68881 and the main processor communicate, via a series of bus cycles, instructions and data according to a predefined protocol as described in **5.3 INSTRUCTION DIALOGS**. Most of these bus cycles transfer an entire item in a single transfer, although large items such as extended precision floating-point numbers require multiple bus cycles to transfer the entire operand. Also, if an MC68881 port size of 8 or 16 bits is selected, multiple bus cycles may be required to transfer items that can be transferred with a single cycle over a 32-bit port.

The communications mechanism utilized by the MC68881 and the main processor uses a set of 'mail-box' registers, called the coprocessor interface registers (CIRs), to move data, instructions, and control information between the devices. The characteristics of the CIRs and the manner in which they are used by the MC68881 and a main processor are described in **SECTION 5 COPROCESSOR INTERFACE**. The discussions in the following sections are not specific to any particular CIR or instruction protocol, except where noted.

When a single bus cycle is able to accomodate an entire item, the transfer mechanism is obviously quite simple and the only requirement that must be met is that the bit alignment of the MC68881 and main processor match. Figure 8-1 shows the bit assigment and significance of the 32-bit data bus of the MC68881, which must be matched to the main processor (for the MC68020, this matching is accomplished by connecting D31 of the MC68881 to D31 of the MC68020, D30 to D30, etc.).

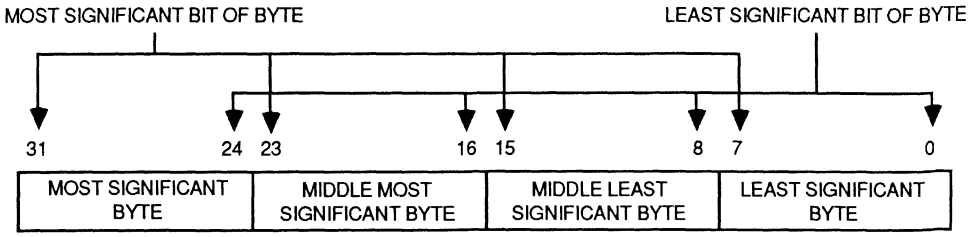


Figure 8-1. MC68881 Data Bus Bit Assignments

When multiple bus cycles are required to transfer an item, the additional requirements of correct transfer order and port alignment must also be met. Figure 8-2 shows the data alignment of the MC68881 for each port size. In this figure, if a section of the data bus is shaded for a particular encoding of $\overline{SIZE}$, A4, A1 and A0, that section of the data bus is active during the transfer (i.e., valid data is expected during a write cycle, and the bus is driven during a read cycle), otherwise it is idle. Note that the port size is not determined by the $\overline{SIZE}$ pin alone, but by the combination of the $\overline{SIZE}$ pin and A0. The following paragraphs describe the transfer order that is used for each port size.

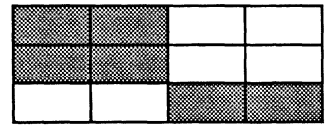
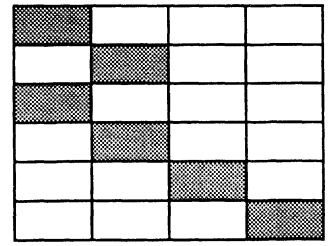
$\overline{SIZE}$	A4	A1	A0	PORT SIZE	$\overline{DSACK1/0}$		ACTIVE DATA BUS SECTIONS
H	0	x	1	32 BITS	L	H	
H	1	x	1		L	L	
H	0	x	0	16 BITS	L	H	
H	1	0	0		L	H	
H	1	1	0		L	H	
L	0	x	0	8 BITS	H	L	
L	0	x	1		H	L	
L	1	0	0		H	L	
L	1	0	1		H	L	
L	1	1	0		H	L	
L	1	1	1		H	L	

Figure 8-2. Data Bus Activity vs. Port Size and Operand Alignment

8.1.1 32-Bit Port Size

When $\overline{SIZE}$ and A0 are both high, the MC68881 port size is defined to be 32 bits wide. In most cases, this configuration is statically selected by connecting the $\overline{SIZE}$ and A0 pins directly to Vcc; although dynamic port size selection is possible if the proper timing constraints are followed for the SIZE and A0 pins. Although this configuration selects a 32-bit port size, the MC68881 utilizes the dynamic bus sizing capabilities of the MC68020 in order to reduce the amount of multiplexing logic on-chip. The value of A4 during a bus cycle determines which bytes of the 32-bit port are used to drive or receive data. Since all of the coprocessor interface registers in the lower half of the CIR address range (A4 = 0, offsets $00 through $0F) are 16-bit registers, dynamic bus sizing is utilized to place all of those CIRs on data bus pins D16 through D32, and the DSACK encoding returned indicates a 16-bit port size. All of the CIRs in the upper half of the CIR address range (A4 = 1, offsets $10 through $1F) are either 32-bit registers or 16-bit registers paired with an undefined register location. Therefore, the DSACK encoding used to terminate accesses in this range indicates a 32-bit port (during a read of the register select CIR, data bits 15-0 are undefined, reserved, and are driven high). In both of these cases, A4 determines the DSACK encoding that is returned and A1 selects the appropriate word location. A0 is always one, to select a 32-bit MC68881 port size, and thus individual bytes cannot be accessed in this configuration. Furthermore, the MC68881 always expects a full 16 or 32 bits of data to be transferred during a bus cycle when $\overline{SIZE}$ is high, A0 is 1, and A4 is 0 or 1, respectively (with the exception of immediate byte or word operands, as discussed below).

When the MC68881 is used in a 32-bit configuration, most CIR accesses transfer an entire instruction or data item in a single bus cycle. The one exception to this is for accesses to the operand CIR, which is used to transfer large items such as floating point numbers and state frames. When an item is larger than four bytes, multiple accesses of the operand CIR are required to complete the transfer. In this case, the correct transfer order must be observed, in addition to the bit and byte alignment discussed above. In all cases, each part of an item is transferred with the most significant bit aligned with bit 31 of the operand CIR (i.e., they are transferred across D31-D24, D31-D16, or D31-D0 for bytes, words or long-words, respectively). With the exception of byte and word immediate operands, the MC68881 never requests the transfer of an item that is not a multiple of four bytes in length. Immediate byte or word operands are transferred in a single bus cycle and are left-aligned with the operand CIR. All other operands are transferred through the operand CIR in 32-bit units until the entire item is transferred.

When multiple bus cycles are required to transfer an item, the first operand CIR access transfers the most significant long word of the item; with each successive access transferring the next least significant long word. For example, when an extended precision number is moved, the first operand CIR access is used to transfer bits 95-64 of the operand, the second access transfers bits 63-32, and the third access transfers bits 31-0 to complete the operand transfer. Note that the manner in which the operand is read from or written to memory is transparent to the MC68881, such that the operand can be stored in memory in the native format of the main processor.

The amount of data transferred with each access to the operand CIR is dependent on the state of an instruction dialog and is determined by the MC68881, not the main processor. For example, if the MC68881 issues an evaluate effective address and transfer data primitive with a length of 12 bytes, three accesses of the operand CIR are expected (with

each access transferring four bytes). Thus, the main processor is not allowed to transfer the operand with a series of word or byte transfers, but must use long word transfers to move the operand (remember, of course, that this discussion is for the case where SIZE is high and A0 is 1 to select a 32-bit port; see the following discussions for the 16 and 8-bit port cases).

8.1.2 16-Bit Port Size

When $\overline{SIZE}$ is high and A0 is low, the MC68881 port size is defined to be 16 bits wide. In most cases, this configuration is statically selected by connecting the $\overline{SIZE}$ and A0 pins directly to Vcc and GND, respectively; although dynamic port size selection is possible if the proper timing constraints are followed for the $\overline{SIZE}$ and A0 pins. Although A0 = 0 in this case, this value is not specifically used to select even byte addresses; rather, it is used to configure the data port to be 16 bits wide. When the MC68881 is configured in this manner, all CIR accesses are assumed to transfer a full 16 bits to the word address selected by A1 (except for the case of an immediate byte operand, as discussed below) and the DSACK encoding returned always indicates that the port is 16 bits wide; individual bytes cannot be accessed in this configuration.

In order to eliminate the need for on-chip multiplexing, the MC68881 drives data on or receives data from only 16 bits of the data bus, depending on the encoding of A1 and A4 (thus allowing D31 and D15 of the MC68881 to be tied together, D30 to be tied to D14, D29 to D13, etc., refer to **SECTION 9 INTERFACING METHODS** for more information on connection to various data bus sizes). For all accesses where A4 is 0, or where A4 is 1 and A1 is 0, data is transferred across D31-D16, while it is transferred across D15-D0 when A4 and A1 are both 1.

When the MC68881 is used in the 16-bit configuration, most CIR accesses transfer an entire instruction or data item in a single bus cycle. The one exception to this is for accesses to the operand CIR, which is used to transfer large items such as floating point numbers and state frames. When an item is larger than two bytes, multiple accesses of the operand CIR are required to complete the transfer. In this case, the correct transfer order must be observed, in addition to the bit and byte alignment discussed above. In all cases, each part of an item is transferred with the most significant bit aligned with bit 31 or bit 15 of the operand CIR, depending on the value of A4 and A1 as described in the previous paragraph. With the exception of byte and word immediate operands, the MC68881 never requests the transfer of an item that is not a multiple of four bytes in length. Immediate byte operands are transferred in a single bus cycle and are left-aligned with the operand CIR (i.e., they are transferred across D31-D24). All other operands are transferred through the operand CIR in 16bit units until the entire item is transferred.

When multiple bus cycles are required to transfer an item, the first operand CIR access transfers the most significant word of the item; with each successive access transferring the next least significant word. For example, when an extended precision number is moved, the first operand CIR access is used to transfer bits 95-80 of the operand, the second access transfers bits 79-64, and the third through sixth accesses transfer bits 63-48, 47-32, 31-16 and 15-0, respectively, to complete the operand transfer. Note that the manner in which the operand is read from or written to memory is transparent to the MC68881, such that the operand can be stored in memory in the native format of the main processor.

The amount of data transferred with each access to the operand CIR is dependent on the state of an instruction dialog and is determined by the MC68881, not the main processor. For example, if the MC68881 issues an evaluate effective address and transfer data primitive with a length of 12 bytes, six accesses of the operand CIR are expected (with each access transferring two bytes). Thus, the main processor is not allowed to transfer the operand with a series of long-word or byte transfers, but must use word transfers to move the operand (remember, of course, that this discussion is for the case where $\overline{SIZE}$ is high and A0 is 0 to select a 16-bit port; see the preceding and following discussions for the 32 and 8-bit port cases).

8.1.3 8-Bit Port Size

When the $\overline{SIZE}$ pin is low the MC68881 port size is defined to be 8 bits wide. In most cases, this configuration is statically selected by connecting the $\overline{SIZE}$ pin directly to GND; although dynamic port size selection is possible if the proper timing constraints are followed for the $\overline{SIZE}$ and A0 pins. In this case, the value of A0 is used to select the correct byte address, rather than to configure the data port size. When the MC68881 is configured in this manner, all CIR accesses transfer one byte to the address selected by A4-A0, and the DSACK encoding returned always indicates that the port is 8 bits wide. In order to eliminate the need for on-chip multiplexing, the MC68881 drives data on or receives data from only 8 bits of the data bus, depending on the encoding of A0, A1 and A4 (thus allowing D31, D23, D15 and D7 of the MC68881 to be tied together; D30 to be tied to D22, D14 and D6; D29 to D21, D13 and D5, etc., refer to **SECTION 9 INTERFACING METHODS** for more information on connection to various data bus sizes). Figure 8-2 shows which bytes of the data bus are driven or received for each encoding of the A0, A1 and A4 lines.

When the MC68881 is used in the 8-bit configuration, most transfers require multiple CIR transfers to move an entire instruction or data item. The one exception to this is for accesses to the operand CIR to transfer a byte immediate operand. When an item is larger than one byte, multiple accesses of the appropriate CIR are required to complete the transfer. In this case, the correct transfer order must be observed, in addition to the bit and byte alignment discussed above. In all cases, each part of an item is transferred with the most significant bit aligned with bit 31, 23, 15 or 7 of the MC68881, depending on the value of A0, A1 and A4 as described in the previous paragraph. With the exception of byte and word immediate operands, the MC68881 never requests the transfer of an item that is not a multiple of four bytes in length. Immediate byte operands are transferred in a single bus cycle and are left-aligned with the operand CIR (i.e., they are transferred across D31-D24). All other operands are transferred through the appropriate CIR in 8-bit units until the entire item is transferred.

When multiple bus cycles are required to transfer an item, the first operand CIR access transfers the most significant word of the item; with each successive access transferring the next least significant word. For example, when an extended precision number is moved, the first operand CIR access is used to transfer bits 95-88 of the operand, the second access transfers bits 87-80, and the third through twelfth accesses transfer bits 79-72, 71-64, 63-56, 55-48, 47-40, 39-32, 31-24, 23-16, 15-8 and 7-0, respectively, to complete the operand transfer. Note that the manner in which the operand is read from or written to memory is transparent to the MC68881, such that the operand can be stored in memory in the native format of the main processor.

The amount of data transferred with each access to the operand CIR is dependent on the state of an instruction dialog and is determined by the MC68881, not the main processor. For example, if the MC68881 issues an evaluate effective address and transfer data primitive with a length of 12 bytes, twelve accesses of the operand CIR are expected (with each access transferring 1 byte). Thus, the main processor is not allowed to transfer the operand with a series of word or long-word transfers, but must use byte transfers to move the operand (remember, of course, that this discussion is for the case where $\overline{SIZE}$ is low to select an 8-bit port; see the preceding discussions for the 32 and 16-bit port cases).

8.2 RESET OPERATION

Before the MC68881 can be used for any operation after power has been applied to the system, it must be initialized through a hardware reset function. This is done when power is initially applied to the system by asserting $\overline{RESET}$ for at least four clock cycles (with reference to the MC68881 CLK signal) after Vcc has reached the nominal operating level. After power has been stable and the MC68881 has executed a power-up reset operation, a subsequent reset operation may be initiated by asserting $\overline{RESET}$ for at least two cycles of the MC68881 CLK signal. Note that in order to maintain compatibility with all M68000 Family devices, the power-on reset pulse for a system should be a minimum of 100 ms, while a 10 clock minimum (with respect to the clock signal of the slowest M68000 Family device in the system) should be used for reset operations after power is stable.

When a hardware reset operation is performed, the MC68881 immediately aborts any operation that may have been in progress and returns to the idle state. All of the floating-point data registers are loaded with non-signalling NANs, and the FPCR and FPSR are cleared to all zeroes (thus clearing any old status information and selecting the IEEE standard default operating modes). An identical operation may be performed under software control by an FRESTORE of a null state frame (although a hardware reset must be executed at power-up in order to initialize the MC68881).

One consideration that should be given to the $\overline{RESET}$ signal of the MC68881 is the treatment of a RESET instruction by an M68000 Family processor. When the RESET instruction is executed by an M68000 Family processor, the internal state of the processor is not affected, while the external system should respond to the reset operation. Since the MC68881 is considered to be part of the internal state of the main processor, prudent system design suggests that the MC68881 should not respond to the assertion of the $\overline{RESET}$ signal by the main processor. This can be accomplished in many ways, depending on the requirements of the system. A simple circuit to support this operation is shown in Figure 8-3.

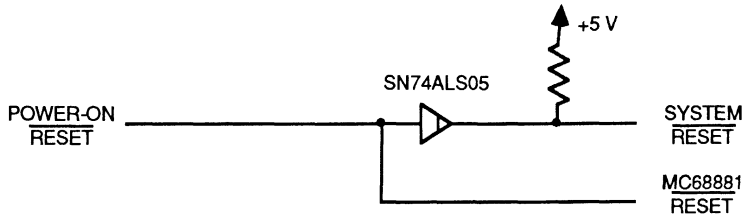


Figure 8-3. MC68881 Reset Logic Example

8.3 BUS CYCLE FUNCTIONAL DESCRIPTIONS

The MC68881 executes three types of bus cycles, based on the direction of the transfer and the CIR that is selected by the main processor. The three bus cycle types are: synchronous read cycles, asynchronous read cycles, and asynchronous write cycles. As described below, the terms synchronous and asynchronous as used here convey slightly different meanings than when they are used to describe the bus transfer characteristics of a microprocessor (e.g., where the M68000 Family of microprocessors is said to utilize an asynchronous bus structure). The following paragraphs describe the functional characteristics of each bus cycle type (for an AC parametric description of the MC68881 bus interface, refer to **SECTION 10 ELECTRICAL SPECIFICATIONS**).

In the following discussions, the main processor is assumed to be an MC68020, with the MC68881 and the MC68020 both driven by the same clock signal. Thus, the terminology and conventions used are identical to the bus description for the MC68020. This clock frequency relationship is not required, but the following discussions are simplified by assuming that both devices use the same clock signal. Where appropriate, referrences are made to variations in bus cycle operations if the main processor is not an MC68020.

8.3.1 CHIP SELECT TIMING

For the most part, the bus cycle timing requirements of the MC68881 are straightforward, with all signal timing following the normal M68000 Family conventions. The only signal timing that is peculiar to the MC68881 bus interface is the relationship of the assertion of chip select to the assertion of the address and data strobes. Unlike most M68000 Family peripherals that require the assertion of chip select to follow the assertion of address or data strobes (in fact, most peripherals do not utilize the strobe signals, since the chip select equation includes the address and/or data strobes), the MC68881 allows the chip select assertion to precede the assertion of the address and data strobes.

In order to detect the start or end of an access, the MC68881 monitors all three of the signals $\overline{AS}$, $\overline{DS}$ and $\overline{CS}$. A cycle start is detected when all three of these signals are asserted, and a cycle end is detected when the first strobe ($\overline{AS}$ or $\overline{DS}$) is negated. The order in which these three signals are sequenced is not critical to correct operation, and in the case of $\overline{CS}$ the occurance of a negated or asserted edge is not needed to detect a new access. For example, it is not required that $\overline{CS}$ be negated between succesive accesses to the MC68881, since the negation and assertion of the $\overline{AS}$ and $\overline{DS}$ signals will cause the MC68881 to detect the end of one access and the start of the next.

The only timing restriction that is placed on $\overline{CS}$ is that it must be either negated or asserted during transitions of the $\overline{AS}$ and $\overline{DS}$ signals. Thus, $\overline{CS}$ must either be asserted before $\overline{AS}$ or $\overline{DS}$ is asserted at the start of a bus cycle or it must remain negated until after the strobe signals are stable (although it may be asserted between the assertion of $\overline{AS}$ and $\overline{DS}$ during a write cycle). For M68000 Family processors, this timing restriction can be satisfied easily by decoding the necessary function code and address bus signals to generate $\overline{CS}$ and not including the $\overline{AS}$ or $\overline{DS}$ signals in the $\overline{CS}$ generation logic. Note that if the MC68020 is used as the main processor and this decoding scheme is used, the function code and address bus signals used to generate $\overline{CS}$ may not change between back-to-back accesses of the

MC68881; thus $\overline{CS}$ may not negate between successive bus cycles. For example, only the following terms must be included in the $\overline{CS}$ equation in an MC68020 based system:

- FC2-FC0 = 7 CPU Space
- A19-A16 = 2 Coprocessor Communications
- A15-A13 = 1 Cp-ID One (Motorola Assembler Default)

8.3.2 Synchronous Read Cycles

When the main processor performs a read access to either the response or save CIR, the MC68881 responds by executing a synchronous read bus cycle. In this context, the term synchronous signifies that the bus cycle timing is directly related to the MC68881 clock signal; but the MC68881 clock is not required to be synchronous with the main processor clock during such a transfer. By synchronizing the bus cycle to the MC68881 clock, the appropriate response primitive or format word is always returned based on the current status of the MC68881. Also, since these bus cycles are used to transmit service requests to the main processor, the synchronous bus cycle timing provides a mechanism to allow the main processor and MC68881 to be synchronized at critical points in an instruction dialog, without requiring that the clock signals for the two devices be synchronous.

The functional timing for the synchronous read cycle is shown in Figure 8-4. The MC68881 detects the start of a synchronous read cycle when chip select and address strobe are asserted, read/write is high, and the address pins are encoded to $00 (to select the response CIR) or $04 (to select the save CIR). When one of these conditions is met, the MC68881 then begins to sample the address strobe, data strobe and chip select lines on each rising edge of the CLK signal. When all three of these signals are sampled as asserted, the MC68881 latches certain internal state flags and uses those flags to determine the appropriate response primitive or format word to be placed on the data bus. One and one-half clock cycles later, the MC68881 begins to drive the data value onto the bus and assert the appropriate data transfer and size acknowledge encoding. The data value remains on the data pins and $\overline{DSACKx}$ remains asserted until the first of the two signals, $\overline{AS}$ or $\overline{DS}$, is negated; at which time the bus cycle is terminated by placing the data bus in the high impedance state and negating $\overline{DSACKx}$.

As shown in Figure 8-4, this type of a bus cycle requires five clock cycles (two wait cycles) when the MC68020 and the MC68881 share the same clock. There is also the possibility that, under certain conditions, these bus cycles may require six or seven clock cycles. Two separate mechanisms determine whether additional clock cycles are reqired for this type of a bus cycle; 1) the relationship between the assertion of $\overline{AS}$, $\overline{DS}$ or $\overline{CS}$ and the rising edge of the MC68881 clock signal, and 2) the relationship between the assertion of $\overline{DSACKx}$ by the MC68881 and the falling edge of the MC68020 clock signal.

As described above, $\overline{DSACKx}$ is triggered to assert one and one-half clock cycles after $\overline{AS}$, $\overline{DS}$ and $\overline{CS}$ are sampled as asserted; thus, the best case timing will occur when all three of these signals are asserted as early in the bus cycle as possible. Since the MC68020 triggers the assertion of $\overline{AS}$ and $\overline{DS}$ with the falling edge of the CLK signal (which is

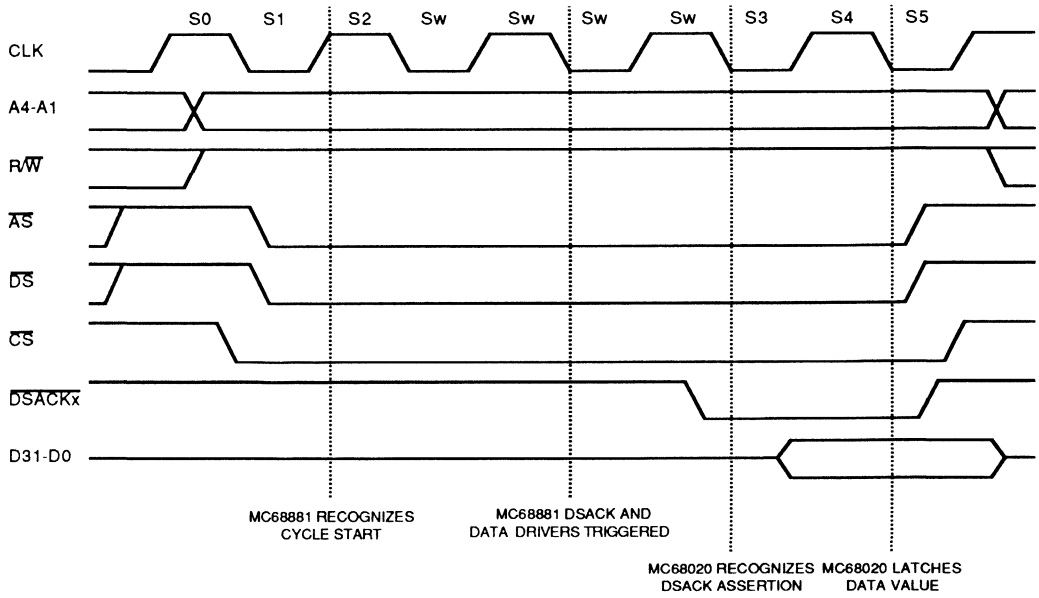


Figure 8-4. Synchronous Read Cycle Timing Diagram

assumed to be the same for both devices) and the MC68881 samples those signals, along with $\overline{CS}$, on the rising edge of the CLK signal, the best case cycle timing will occur only if $\overline{AS}$, $\overline{DS}$ and $\overline{CS}$ are all asserted by the required setup time to the next rising edge of the clock. Thus, the maximum assertion and propagation delays for these signals must be less than the clock pulse width low in order to guarantee the best case bus cycle timing. Although the maximum specifications for the assertion, by the MC68020, of $\overline{AS}$ or $\overline{DS}$ from the falling edge of the clock do not guarantee the best case timing for operation at 16.67 MHz under worst case system environments, the best case timing will normally occur under typical system conditions. In order to assure the possibility that the best case timing will occur, system designers should utilize the $\overline{CS}$ generation methods described above to prevent propagation delays of the $\overline{CS}$ logic from lengthening the bus cycle by one clock.

In the same manner as just described (where the MC68881 "misses" the assertion of $\overline{AS}$, $\overline{DS}$ or $\overline{CS}$), one clock cycle may be added to the bus cycle timing if the MC68020 "misses" the assertion of $\overline{DSACKx}$ by the MC68881. The assertion of $\overline{DSACKx}$ by the MC68881 is triggered by the falling edge of the clock, and the propagation delay for this assertion can be quite long (slightly longer than one 16.67 MHz clock cycle under worst case system conditions). Since the MC68020 samples $\overline{DSACKx}$ on the falling edge of the clock, the assertion of $\overline{DSACKx}$ triggered by a given falling clock edge may not be completed by the setup time to the next falling clock edge. There is very little that a system designer can do to assure that the $\overline{DSACKx}$ assertion is recognized on the first falling clock edge after it is triggered, since the propagation delay is dependent on individual device characteristics as well as system conditions such as temperature and power supply levels.

Due to the nature of the two mechanisms just described, it is possible that for an individual system the bus cycle timing for synchronous read cycles will be different under varying system conditions. For example, when a system is first turned on (and thus the devices are at room temperature) it is quite likely that synchronous read cycles will require five clock cycles as shown in Figure 8-4. As the temperature increases to the normal operating range, the synchronous read cycle timing may change to six clock cycles. If the temperature rise affects both of the synchronization mechanisms enough (particularly if the $\overline{CS}$ generation logic causes the assertion of $\overline{CS}$ to follow the assertion of $\overline{AS}$ and/or $\overline{DS}$), the timing for these operations may increase to seven clock cycles, or even vary on a cycle-by-cycle basis between six and seven clock cycles. Some other factors that may affect the timing for synchronous reads cycles are the power supply levels for the MC68881 and MC68020, the individual device characteristics (due to manufacturing variances) and the capacitive loading of the control signals.

It should be noted that the timing variances for synchronous read cycles will not affect the overall performance of a system significantly. Specifically, one or two additional clock cycles per synchronous read cycle results in a small percentage change in the overall execution time for an instruction (since most instructions typically require over 50 clock cycles to execute). The only environment where these timing variances may be of concern is when a programmer is attempting to optimize an instruction sequence for maximum overlap; in which case these factors should be added to the instruction execution timing variability mechanisms discussed in **SECTION 6 INSTRUCTION EXECUTION TIMING.**

8.3.3 Asynchronous Bus Cycles

When the main processor performs any access to the MC68881 other than a read of the response or save CIR, the MC68881 will respond by executing an asynchronous bus cycle (either a read or a write). In this context, the term asynchronous signifies that the bus cycle timing is not related to the MC68881 or MC68020 clock signals in any way. The MC68881 supports this type of operation by implementing all of the CIRs, except the response and save CIRs, as dual ported structures. Thus, the main processor can access these CIRs at the maximum speed regardless of the clock frequency of the MC68881, while the MC68881 internally accesses these CIRs in a synchronous manner.

The MC68881 executes two types of asynchronous bus cycles based on the level of the R/$\overline{W}$ signal. The following paragraphs describe each of these bus cycle types; the asynchronous read cycle, and the asynchronous write cycle.

8.3.3.1 Asynchronous Read Cycles. The functional timing for the asynchronous read cycle is shown in Figure 8-5. The MC68881 detects the start of an asynchronous read cycle when chip select, address strobe and data strobe are asserted, read/write is high, and the address pins are not encoded to $00 or $04 (which selects the response or save CIR, respectively). When this condition is met, the MC68881 responds by placing the data from the selected CIR on the data bus and asserting the appropriate data transfer and size

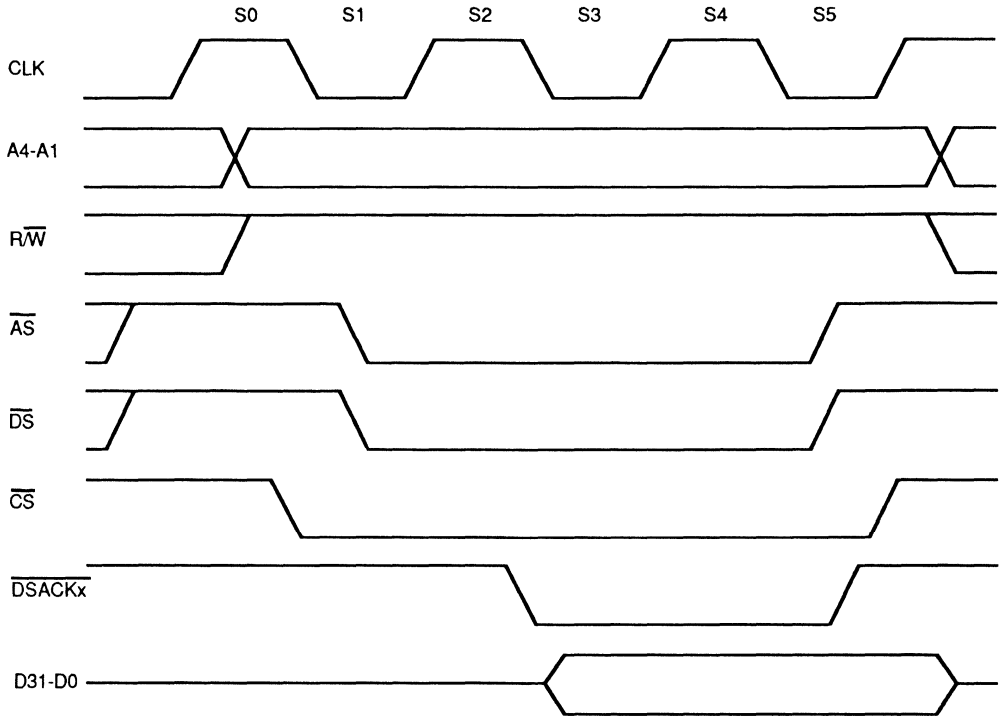


Figure 8-5. Asynchronous Read Cycle Timing Diagram

acknowledge encoding. The data value remains on the data pins and DSACKx remains asserted until the first of the two signals, AS or DS, is negated; at which time the bus cycle is terminated by placing the data bus in the high impedance state and negating DSACKx.

As shown in Figure 8-5, this type of a bus cycle requires three clock cycles (no wait cycles) when the MC68020 and the MC68881 share the same clock. Due to the asynchronous timing of the data transfer and size acknowledge assertion by the MC68881, this bus cycle timing does not depend on the clock frequency of the MC68881 (there are some exceptions to this rule, as discussed in **8.4 INTER-CYCLE TIMING RESTRICTIONS**). For example, if the MC68881 clock frequency is 12.5 MHz and the MC68020 clock frequency is 16.67 MHz, this bus cycle will require three MC68020 clock cycles since the assertion of DSACKx will be recognized by the MC68020 on the falling edge of S2. This assumes that the chip select logic causes the assertion of CS to preceed the assertion of AS and DS so that the AS/DS assertion to DSACKx assertion delay is not lengthened by the chip select logic propagation time

8.3.3.2 Asynchronous Write Cycles. The functional timing for the asynchronous write cycle is shown in Figure 8-6. The MC68881 detects the start of an asynchronous write cycle when chip select and address strobe are asserted and read/write is low. When this condition is met, the MC68881 responds by asserting the appropriate data transfer and size acknowledge encoding and latching the value of the data bus into the selected CIR when an asserted pulse occurs on $\overline{DS}$. The $\overline{DSACKx}$ encoding remains asserted until $\overline{AS}$ is negated; at which time the bus cycle is terminated by negating $\overline{DSACKx}$.

As shown in Figure 8-6, this type of bus cycle requires three clock cycles (no wait cycles) when the MC68020 and the MC68881 share the same clock. Due to the asynchronous timing of the data transfer and size acknowledge assertion by the MC68881, this bus cycle timing does not depend on the clock frequency of the MC68881 (there are some exceptions to this rule, as discussed in **8.4 INTER-CYCLE TIMING RESTRICTIONS**). For example, if the MC68881 clock frequency is 12.5 MHz and the MC68020 clock frequency is 16.67 MHz, this bus cycle will require three MC68020 clock cycles since the assertion of $\overline{DSACKx}$ will be recognized by the MC68020 on the falling edge of S2. This assumes that the chip select logic causes the assertion of $\overline{CS}$ to preceed the assertion of $\overline{AS}$ and $\overline{DS}$ so that the $\overline{AS}$/$\overline{DS}$ assertion to $\overline{DSACKx}$ assertion delay is not lengthened by the chip select logic propagation time.

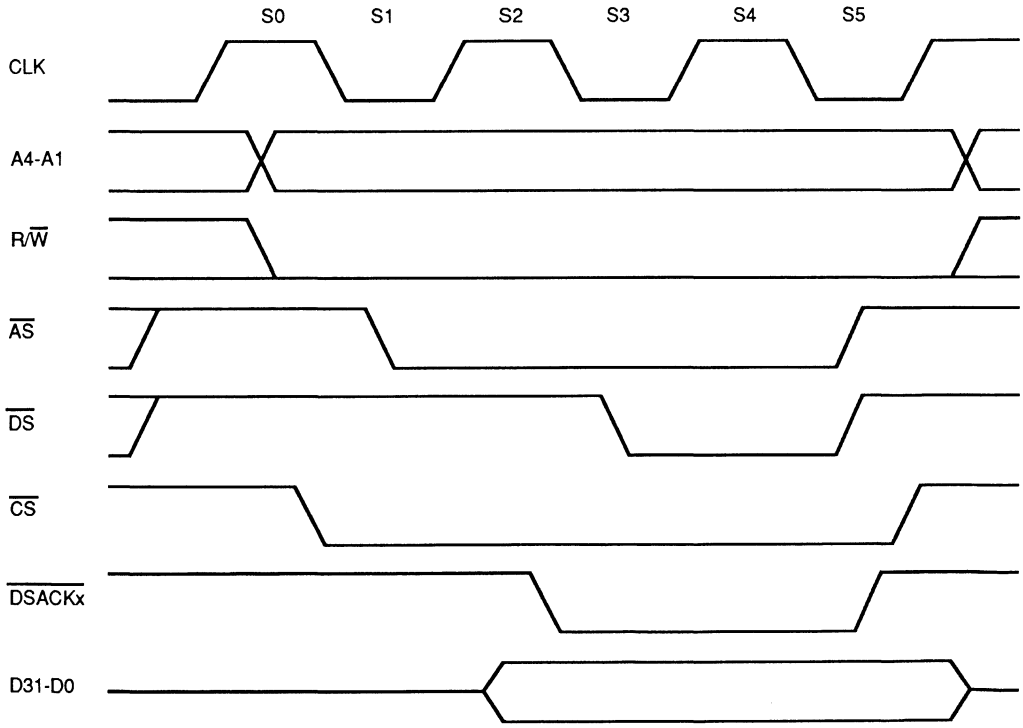


Figure 8-6. Asynchronous Write Cycle Timing Diagram

8.4 INTER-CYCLE TIMING RESTRICTIONS

The bus interface of the MC68881 is designed such that there are no restrictions on the clock frequency relationship between the MC68881 and the main processor. In most cases, differences in the clock frequency of the two devices does not affect the operation of the bus; and particularly, it does not affect the timing of individual bus cycles. However, there are some cases where the timing of a bus cycle is modified if the MC68881 is "overrun" by the main processor.

During coprocessor interface dialogs, certain bus cycles trigger actions by the MC68881 on the negated edge of data strobe. Operations internal to the MC68881 that are initiated in this manner are completed within four clock cycles after the negation of $\overline{DS}$, but the main processor may initiate a subsequent asynchronous bus cycle before those internal operations are completed. In these cases, the MC68881 delays the subsequent asynchronous access by not responding to the bus cycle (and thus not asserting $\overline{DSACK}$) until the internal operations are completed. Synchronous accesses (i.e., accesses to the response or save CIR) execute in the normal manner regardless of preceding accesses. The following is a list of the bus cycles that initiate internal operations on the negated edge of $\overline{DS}$, where a subsequent asynchronous bus cycle might overrun the MC68881 and necessitate a delay in the assertion of DSACK:

1) A write cycle to the least significant byte of the control CIR.
2) A write cycle to the least significant byte of the restore CIR.
3) The last write cycle to the least significant byte of the operand CIR during a restore operation with a busy state frame.
4) The first read from the least significant byte of the operand CIR during a save operation with an idle or busy state frame.

In all of these cases, the term "least significant byte" indicates a transfer of any size that includes the least significant byte of the referrenced CIR, and does not indicate that only byte transfers cause conditions that require delays in subsequent bus cycles.

In addition to the cases just described, there is also the possibility that the main processor may overrun the MC68881 if the main processor clock frequency is greater than that of the MC68881. There are two cases where this might occur:

1) The main processor reads the operand or register select CIR before the MC68881 has data ready for transfer to the main processor.
2) The main processor writes to the operand CIR before the data from the previous write cycle has been stored internally.

In both of these cases, the MC68881 does not respond to the initiation of an asynchronous bus cycle until the internal data transfers are completed (synchronous bus cycles are not delayed).

8.5 COPROCESSOR INTERFACE PROTOCOL RESTRICTIONS

As just described, the MC68881 will delay asynchronous bus cycles, if necessary, until internal operations are completed. However, even though the response to these bus cycles

is delayed, the MC68881 bus interface unit control logic does detect the beginning of each access regardless of the state of the execution unit. Thus, it is possible that an access to a CIR may be detected before the bus interface unit has completed previous operations and updated status flags to reflect the state of an instruction dialog. This can result in spurious protocol violations if the coprocessor interface protocol is not strictly observed.

The most important protocol that must be observed is that the come again request included by the MC68881 in every evaluate effective address and transfer data primitive must not be ignored by the main processor. For example, if the come again request is ignored and the main processor clock is much faster than the MC68881 clock, the following situation may occur:

1) The main processor receives the evaluate effective address and transfer data request primitive, processes it, and begins to transfer the operand.
2) The last operand part is written to the operand CIR.
3) The main processor ignores the come again request and begins execution of the next instruction immediately.
4) The next instruction is an MC68881 instruction, such that the main processor writes the command word to the command CIR to initiate the instruction.
5) Since the internal operand transfer is not complete, the BIU flags still indicate that the next expected access is to the operand CIR; thus the access to the command CIR is deemed illegal, and a protocol violation occurs.

In this case, if the main processor follows the protocol and services the come again request by reading the response CIR immediately after the last operand CIR access, a null (CA = 1, IA = 1) primitive may be returned by the MC68881. Since the response CIR read cycle timing is synchronous with the MC68881 clock signal, this read cycle allows the main processor to be synchronized to the MC68881 internal operations. Thus, the next read of the response CIR will normally occur after internal operations are completed. At that time, the response encoding is changed to null (CA = 0) to allow the main processor to proceed, and the subsequent access to the command CIR is deemed as legal.

8.6 USE OF THE $\overline{\text{SENSE}}$ PIN

In order to allow hardware in a system to detect the presence of the MC68881, one pin may be optionally used as a device sense signal. By using a circuit similar to the following, external hardware may determine if the MC68881 is present in a socket prepared for it.

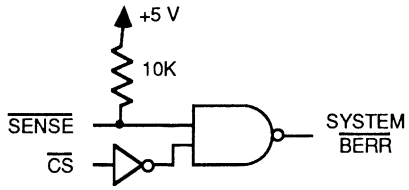


Figure 8-7. Sense Device Circuit Example

If the $\overline{\text{SENSE}}$ pin is not needed for this function, it should be connected to system ground, and functions as an extra GND pin to provide additional noise immunity.

8.7 POWER AND GROUND CONSIDERATIONS

The MC68881 is fabricated in Motorola's advanced HCMOS process, contains approximately 140,000 transistors, and is capable of operating at clock speeds of 16.67 MHz. While the use of CMOS for a device containing such a large number of transistors allows significantly reduced power consumption in comparison to an equivalent NMOS device, the high clock speed makes the characteristics of the power supplied to the part quite important. The power supply must be as free from noise as possible, and it must be able to supply large amounts of instantaneous current when the MC68881 performs certain operations. In order to meet these requirements, more detailed attention should be given to the power supply connection to the MC68881 than is required for older NMOS devices that operate at slower clock rates.

In order to provide a solid power supply interface, four Vcc pins, eight primary GND pins, and two secondary GND pins are provided. This allows two Vcc and GND pins to supply the power for the data bus, while the remaining Vcc and GND pins are used by the internal logic and DSACK drivers. The two secondary GND pins are not intended to provide the main power supply interface, but merely to augment it as required (one of these pins is the $\overline{\text{SENSE}}$ pin, which may be used as an optional GND connection). Three Vcc and four GND pin positions are reserved for future use by Motorola, and should be connected appropriately in order to maintain pin compatability with all future versions of the MC68881. The Table 8-1 lists the Vcc and GND pin assignments.

Table 8-1. Vcc and GND Pin Assignments

Devices Supplied	V_{CC}	GND
D31-D16	H8	J8
D15-D00	B8	B7
Internal Logic, $\overline{\text{DSACK1}}$, $\overline{\text{DSACK0}}$	E2, E9	A2, B2, B3, B4*, C3, E10, K3
Separate	—	C1
Extra	A1, B1, J2	A10, D2, F2, H9

* B4 is the $\overline{\text{SENSE}}$ pin, and may be used optionally as a ground pin or to detect the presence of the MC68881 in the system.

In order to reduce the amount of noise in the power supplied to the MC68881, common capacitive decoupling techniques should be observed. While there is no recommended layout for this capacitive decoupling, it is suggested that a combination of low, middle and high frequency filter capacitors be placed as close to the chip as possible (for example, a set of 10 µf, 0.1 µf and 330 pf capacitors in parallel provides filtering for nearly the entire frequency spectrum present in a digital system). In a system that utilizes the MC68020 as the main processor, these capacitive decoupling practices should also be observed for the main processor. In particular, the 10 µf "tank" capacitor should be reasonably close to both

devices (since the two devices are typically placed next to each other on a board) to provide for the high instantaneous current requirements of both the MC68020 and the MC68881.

In addition to the capacitive decoupling of the power supply, care should be taken to ensure a low resistance connection between the MC68881 Vcc and GND pins and the system power supply traces. In particular, the connections to pins B7 and J8 (the GND pins for the data bus pins) must have very low resistance. This is due to the fact that when a read of the MC68881 occurs, the data bus drivers can sink very large amounts of current to ground in order to pull the data bus signals low (if the data pattern that is read contains mostly zeroes). If low resistance connections are not provided on pins B7 and J8, the ground potential internal to the package may rise, the fall time of the data signals may be increased, and the low output voltage noise margin may be reduced.

SECTION 9
INTERFACING METHODS

This section contains information on the interface logic required to connect the MC68881 to an MC68020 as a coprocessor, or to an MC68000, MC68008, MC68010, or MC68012 as a peripheral processor.

9.1 MC68881/MC68020 INTERFACING

The following paragraphs describe how to connect the MC68881 to an MC68020 for coprocessor operation via an 8-, 16-, or 32-bit data bus.

9.1.1 32-Bit Data Bus Coprocessor Connection

Figure 9-1 illustrates the coprocessor interface connection of an MC68881 to an MC68020 via a 32-bit data bus. The MC68881 is configured to operate over a 32-bit data bus when both the A0 and $\overline{\text{SIZE}}$ pins are connected to Vcc.

9.1.2 16-Bit Data Bus Coprocessor Connection

Figure 9-2 illustrates the coprocessor interface connection of an MC68881 to an MC68020 via a 16-bit data bus. The MC68881 is configured to operate over a 16-bit data bus when the $\overline{\text{SIZE}}$ pin is connected to Vcc, and the A0 pin is connected to GND. The sixteen least significant data pins (D0-D15) must be connected to the sixteen most significant data pins (D16-D31) when the MC68881 is configured to operate over a 16-bit data bus (i.e., connect D0 to D16, D1 to D17, ... and D15 to D31). The DSACK pins of the two devices are directly connected, although it is not necessary to connect the $\overline{\text{DSACK0}}$ pin since the MC68881 never asserts it in this configuration.

9.1.3 8-Bit Data Bus Coprocessor Connection

Figure 9-3 illustrates the connection of an MC68881 to an MC68020 as a coprocessor over an 8-bit data bus. The MC68881 is configured to operate over an 8-bit data bus when the $\overline{\text{SIZE}}$ pin is connected to GND. The twenty four least significant data pins (D0-D23) must be

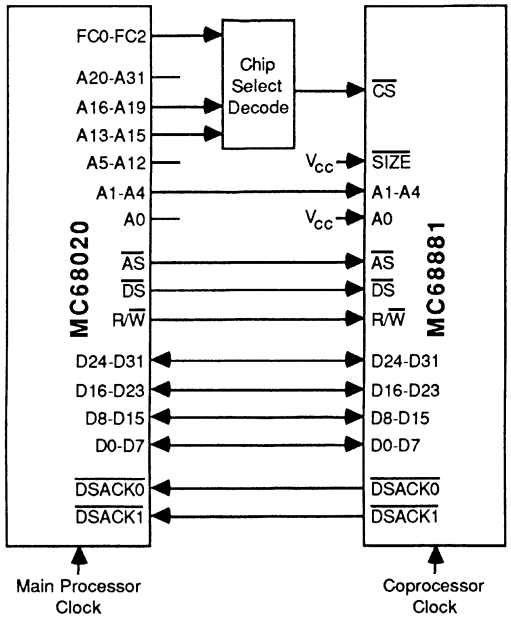


Figure 9-1. 32-Bit Data Bus Coprocessor Connection

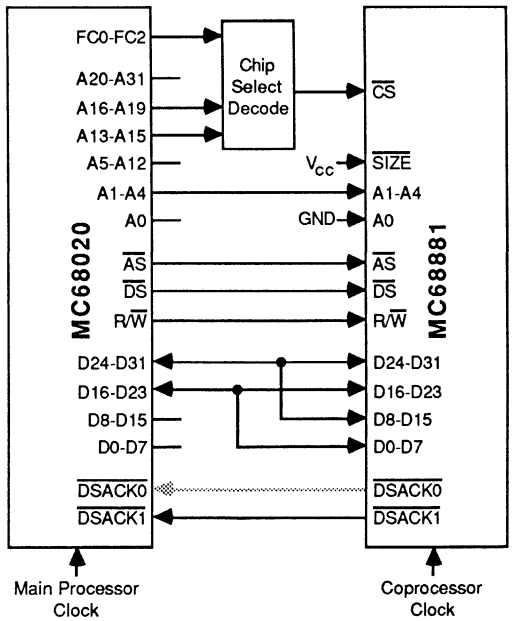


Figure 9-2. 16-Bit Data Bus Coprocessor Connection

connected to the eight most significant data pins (D24-D31) when the MC68881 is configured to operate over an 8-bit data bus (i.e., connect D0 to D8, D16 and D24; D1 to D9, D17, and D25; ... and D7 to D15, D23, and D31). The DSACK pins of the two devices are directly connected, although it is not necessary to connect the $\overline{\text{DSACK1}}$ pin since the MC68881 never asserts it in this configuration.

9.2 MC68881-MC68000/MC68008/MC68010 INTERFACING

The following paragraphs describe how to connect the MC68881 to an MC68000, MC68008, MC68010, or MC68012 processor for operation as a peripherial via an 8- or 16-bit data bus.

9.2.1 16-Bit Data Bus Peripheral Processor Connection

Figure 9-4 illustrates the connection of an MC68881 to an MC68000, MC68010, or MC68012 as a peripheral processor over a 16-bit data bus. The MC68881 is configured to operate over a 16-bit data bus when the $\overline{\text{SIZE}}$ pin is connected to Vcc, and the A0 pin is connected to GND. The sixteen least significant data pins (D0-D15) must be connected to the sixteen most significant data pins (D16-D31) when the MC68881 is configured to operate over a 16-bit data bus (i.e., connect D0 to D16, D1 to D17, ... and D15 to D31). The $\overline{\text{DSACK1}}$ pin of the MC68881 is connected to the $\overline{\text{DTACK}}$ pin of the main processor, and the $\overline{\text{DSACK0}}$ pin is not used.

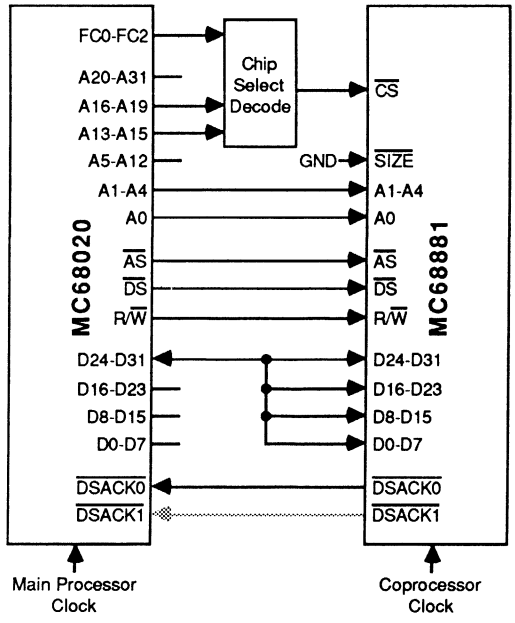


Figure 9-3. 8-Bit Data Bus Coprocessor Connection

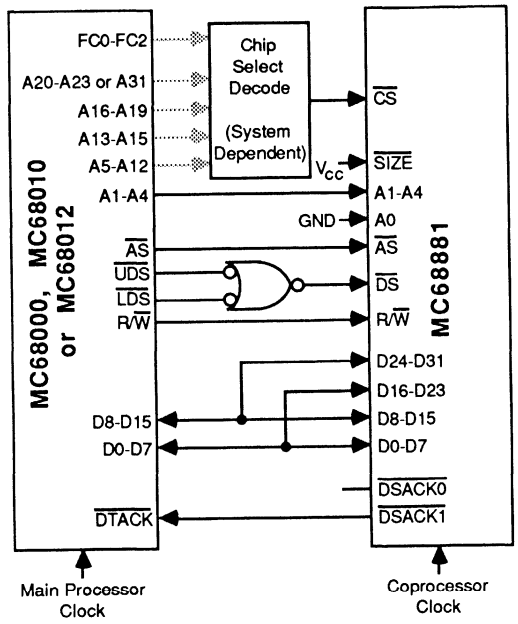


Figure 9-4. 16-Bit Data Bus Peripheral Processor Connection

When connected as a peripheral processor, the MC68881 chip select ($\overline{CS}$) decode is system dependent. If the MC68000 is used as the main processor, the MC68881 $\overline{CS}$ must be decoded in the supervisor or user data spaces. However, if the MC68010 or MC68012 is used for the main processor, the MOVES instruction may be used to emulate any CPU space access that the MC68020 generates for coprocessor communications. Thus, the $\overline{CS}$ decode logic for such systems may be the same as in an MC68020 system, such that the MC68881 will not use any part of the data address spaces.

9

9.2.2 8-Bit Data Bus Peripheral Processor Connection

Figure 9-5 illustrates the connection of an MC68881 to an MC68008 as a peripheral processor over an 8-bit data bus. The MC68881 is configured to operate over an 8-bit data bus when the $\overline{SIZE}$ pin is connected to GND. The eight least significant data pins (D0-D7) must be connected to the twenty four most significant data pins (D8-D31) when the MC68881 is configured to operate over an 8-bit data bus (i.e., connect D0 to D8, D16, and D24; D1 to D9, D17, and D25; ... and D7 to D15, D23, and D31). The $\overline{DSACK0}$ pin of the MC68881 is connected to the $\overline{DTACK}$ pin of the MC68008, and the $\overline{DSACK1}$ pin is not used.

When connected as a peripheral processor, the MC68881 chip select ($\overline{CS}$) decode is system dependent, and the $\overline{CS}$ must be decoded in the supervisor or user data spaces.

9.3 PERIPHERAL PROCESSOR OPERATION

The MC68881 may be used as a peripheral processor on systems where the main processor does not have a coprocessor interface by using instruction sequences which emulate the protocol of the coprocessor interface. When an MC68881 instruction is encountered by an MC68000, MC68008, MC68010, or MC68012, the instruction will cause an F-line emulator trap to be taken. The trap handler may then emulate the coprocessor interface protocol. Refer to **SECTION 5 COPROCESSOR INTERFACE** for details on the communications protocol.

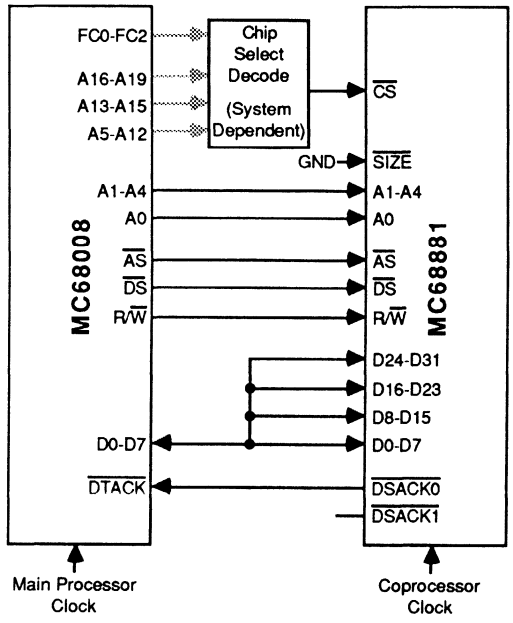


Figure 9-5. 8-Bit Data Bus Peripheral Processor Connection

The MC68881 requests services from the main processor via coprocessor interface response register primitives. **SECTION 5 COPROCESSOR INTERFACE** describes the main processor service requests required for the execution of each MC68881 instruction type. Also included in **SECTION 5 COPROCESSOR INTERFACE** is a summary of all MC68881 response primitives.

SECTION 10
ELECTRICAL SPECIFICATIONS

This section contains electrical specifications and associated timing information for the MC68881.

10.1 MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V_{CC}	–0.3 to +7.0	V
Input Voltage	V_{in}	–0.3 to +7.0	V
Operating Temperature	T_A	0 to 70	°C
Storage Temperature	T_{stg}	–55 to +150	°C

This device contains protective circuitry against damage due to high static voltages or electrical fields; however, it is advised that normal precautions be taken to avoid application of any voltages higher than maximum-rated voltages to this high-impedance circuit. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage level (e.g., either GND or V_{CC}).

10.2 THERMAL CHARACTERISTICS – PGA PACKAGE

Characteristic	Symbol	Value	Rating
Thermal Resistance - Ceramic			
Junction to Ambient	θ_{JA}	30*	°C/W
Junction to Case	θ_{JC}	15*	°C/W

* Estimated

10.3 POWER CONSIDERATIONS

The average chip-junction temperature, T_J, in °C can be obtained from:

$$T_J = T_A + (P_D \cdot \theta_{JA}) \quad (1)$$

Where:

$T_A \equiv$ Ambient Temperature, °C

$\theta_{JA} \equiv$ Package Thermal Resistance, Junction-to-Ambient, °C/W

$P_D \equiv P_{INT} + P_{PORT}$

$P_{INT} \equiv I_{CC} \times V_{CC}$, Watts — Chip Internal Power

$P_{I/O} \equiv$ Port Power Dissipation, Watts — User Determined

For most applications $P_{I/O} \ll P_{INT}$ and can be neglected.

An approximate relationship between P_D and T_J (if $P_{I/O}$ is neglected) is:

$$P_D = K \div (T_J + 273°C) \quad (2)$$

Solving equations 1 and 2 for K gives:

$$K = P_D \cdot (T_A + 273°C) + \theta_{JA} \cdot P_D^2 \quad (3)$$

Where K is a constant pertaining to the particular part. K can be determined from equation (3) by measuring P_D (at equilibrium) for a known T_A. Using this value of K, the values of P_D and T_J can be obtained by solving equations (1) and (2) interatively for any value of T_A.

The total thermal resistance of a package (θ_{JA}) can be separated into two components, θ_{JC} and θ_{CA}, representing the barrier to heat flow from the semiconductor junction to the package (case) surface (θ_{JC}) and from the case to the outside ambient (θ_{CA}). These terms are related by the equation:

$$\theta_{JA} = \theta_{JC} + \theta_{CA} \quad (4)$$

θ_{JC} is device related and cannot be influenced by the user. However, θ_{CA} is user dependent and can be minimized by such thermal management techniques as heat sinks, ambient air cooling and thermal convention. Thus, good thermal management on the part of the user can significantly reduce θ_{CA} so that θ_{JA} approximately equals θ_{JC}. Substitution of θ_{JC} for θ_{JA} in equation (1) will result in a lower semiconductor junction temperature.

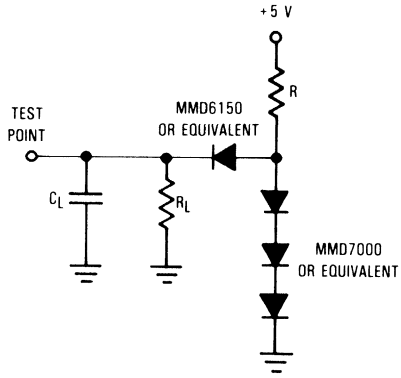
Values for thermal resistance presented in this data sheet, unless estimated, were derived using the procedure described in Motorola Reliability Report 7843, "Thermal Resistance Measurement Method for MC68XX Microcomponent Devices," and are provided for design purposes only. Thermal measurements are complex and dependent on procedure and setup. User derived values for thermal resistance may differ.

10.4 DC ELECTRICAL CHARACTERISTICS

(V_{CC} = 5.0 Vdc, ±5%; GND = 0 Vdc, T_A = 0°C to 70°C) (See Figure 10-1)

Characteristic	Symbol	Min	Max	Unit
Input High Voltage	V_{IH}	2.0	V_{CC}	V
Input Low Voltage	V_{IL}	GND–0.3	0.8	V
Input Leakage Current @ 5.25 V CLK, $\overline{RESET}$, R/$\overline{W}$, A0-A4, $\overline{CS}$, $\overline{DS}$, $\overline{AS}$, $\overline{SIZE}$	I_{in}	—	10	µA
Hi-Z (Off-State) Input Current @ 2.4V/0.4V $\overline{DSACK0}$, $\overline{DSACK1}$, D0-D31	I_{TSI}	—	20	µA
Output High Voltage (I_{OH} = –400 µA) $\overline{DSACK0}$, $\overline{DSACK1}$, D0-D31	V_{OH}	2.4	—	V
Output Low Voltage (I_{OL} = 5.3 mA) $\overline{DSACK0}$, $\overline{DSACK1}$, D0-D31	V_{OL}	—	0.5	V
Power Dissipation	P_D	—	0.75	W
Capacitance * (V_{in} = 0, T_A = 25°C, f = 1MHz)	C_{in}	—	20	pF

* Capacitance is periodically sampled rather than 100% tested.



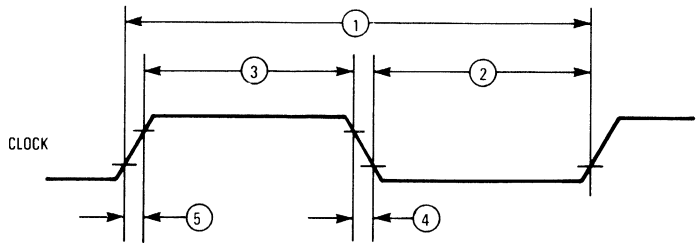
C_L = 130 pF (includes all parasitics)
R = 740 ohms for $\overline{DSACK0}$, $\overline{DSACK1}$, D0-D31
R_L = 6.0 kohms for $\overline{DSACK0}$, $\overline{DSACK1}$, D0-D31

Figure 10-1. Test Loads

10.5 AC ELECTRICAL CHARACTERISTICS – CLOCK INPUT

(V_{CC} = 5.0 Vdc ±5%; GND = 0 Vdc, T_A = 0°C to 70°C; see Figure 10-2)

No.	Characteristic	Symbol	MC68881RC12		MC68881RC16		Unit
			Min	Max	Min	Max	
	Frequency of Operation	f	4.0	12.5	4.0	16.67	MHz
1	Cycle Time	t_{cyc}	80	250	60	250	ns
2, 3	Clock Pulse Width	t_{CH}, t_{CL}	32	120	24	120	ns
4, 5	Clock Rise and Fall Time	t_{Cr}, t_{Cf}	—	5	—	5	ns



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

Figure 10-2. Clock Input Timing Diagram

10.6 AC ELECTRICAL CHARACTERISTICS – READ AND WRITE CYCLES

(V_{CC} = 5.0 Vdc ±5%; GND = 0 Vdc, T_A = 0°C to 70°C; see Figures 10-3–10-5)

No.	Characteristic	Symbol	MC68881RC12		MC68881RC16		Unit
			Min	Max	Min	Max	
6	Address Valid to $\overline{AS}$ Asserted (Note 5)	t_{AVASL}	20	—	15	—	ns
6a	Address Valid to $\overline{DS}$ Asserted (Read) (Note 5)	t_{AVRDSL}	20	—	15	—	ns
6b	Address Valid to $\overline{DS}$ Asserted (Write) (Note 5)	t_{AVWDSL}	65	—	50	—	ns
7	$\overline{AS}$ Negated to Address Invalid (Note 6)	t_{ASHAX}	15	—	10	—	ns
7a	$\overline{DS}$ Negated to Address Invalid (Note 6)	t_{DSHAX}	15	—	10	—	ns
8	$\overline{CS}$ Asserted to $\overline{AS}$ Asserted or $\overline{AS}$ Asserted to $\overline{CS}$ Asserted (Note 8)	t_{CVASL}	0	—	0	—	ns
8a	$\overline{CS}$ Asserted to $\overline{DS}$ Asserted or $\overline{DS}$ Asserted to $\overline{CS}$ Asserted (Read) (Note 9)	t_{CVRDSL}	0	—	0	—	ns
8b	$\overline{CS}$ Asserted to $\overline{DS}$ Asserted (Write)	t_{CVWDSL}	45	—	35	—	ns
9	$\overline{AS}$ Negated to $\overline{CS}$ Negated	t_{ASHCX}	10	—	10	—	ns
9a	$\overline{DS}$ Negated to $\overline{CS}$ Negated	t_{DSHCX}	10	—	10	—	ns
10	R/$\overline{W}$ High to $\overline{AS}$ Asserted (Read)	t_{RVASL}	20	—	15	—	ns
10a	R/$\overline{W}$ High to $\overline{DS}$ Asserted (Read)	t_{RVDSL}	20	—	15	—	ns
10b	R/$\overline{W}$ Low to $\overline{DS}$ Asserted (Write)	t_{RLSL}	45	—	35	—	ns
11	$\overline{AS}$ Negated to R/$\overline{W}$ Low (Read) or $\overline{AS}$ Negated to R/$\overline{W}$ High (Write)	t_{ASHRX}	15	—	10	—	ns
11a	$\overline{DS}$ Negated to R/$\overline{W}$ Low (Read) or $\overline{DS}$ Negated to R/$\overline{W}$ High (Write)	t_{DSHRX}	15	—	10	—	ns
12	$\overline{DS}$ Width Asserted (Write)	t_{DSL}	50	—	40	—	ns
13	$\overline{DS}$ Width Negated	t_{DSH}	50	—	40	—	ns
13a	$\overline{DS}$ Negated to $\overline{AS}$ Asserted (Note 4)	t_{DSHASL}	40	—	30	—	ns
14	$\overline{CS}$, $\overline{DS}$ Asserted to Data-Out Valid (Read) (Note 2)	t_{DSLDO}	—	110	—	80	ns
15	$\overline{DS}$ Negated to Data-Out Invalid (Read)	t_{DSHDO}	0	—	0	—	ns
16	$\overline{DS}$ Negated to Data-Out High Impedance (Read)	t_{DSHDZ}	—	70	—	50	ns
17	Data-In Valid to $\overline{DS}$ Asserted (Write)	t_{DIDSL}	20	—	15	—	ns
18	$\overline{DS}$ Negated to Data-In Invalid (Write)	t_{DSHDI}	20	—	15	—	ns
19	$\overline{START}$ True to $\overline{DSACK0}$ and $\overline{DSACK1}$ Asserted (Notes 2,10)	t_{SLDAL}	—	70	—	50	ns
19a	$\overline{DSACK0}$ Asserted to $\overline{DSACK1}$ Asserted (Skew) (Note 7)	t_{DADAS}	–20	20	–15	15	ns
20	$\overline{DSACK0}$ or $\overline{DSACK1}$ Asserted to Data-Out Valid (Read)	t_{DALDO}	—	60	—	50	ns
21	$\overline{START}$ False to $\overline{DSACK0}$ and $\overline{DSACK1}$ Negated (Note 10)	t_{SHDAH}	—	70	—	50	ns

10.6 AC ELECTRICAL CHARACTERISTICS (Continued)

No.	Characteristic	Symbol	MC68881RC12		MC68881RC16		Unit
			Min	Max	Min	Max	
22	$\overline{START}$ False to $\overline{DSACK0}$ and $\overline{DSACK1}$ High Impedance (Note 10)	t_{SJDAZ}	—	90	—	70	ns
23	$\overline{START}$ True to Clock High (Synchronous Read) (Notes 3, 10)	t_{DSLCH}	0	—	0	—	ns
24	Clock Low to Data-Out Valid (Synchronous Read) (Note 3)	t_{CLDO}	—	140	—	105	ns
25	$\overline{START}$ True to Data-Out Valid (Synchronous Read) (Notes 3, 10, and 11)	t_{DSSLDO}	1.5 Clks	140 + 2.5 Clks	1.5 Clks	105 + 2.5 Clks	ns
26	Clock Low to $\overline{DSACK0}$ and $\overline{DSACK1}$ Asserted (Synchronous Read) (Note 3)	t_{CLDAL}	—	100	—	75	ns
27	$\overline{START}$ True to $\overline{DSACK0}$ and $\overline{DSACK1}$ Asserted (Synchronous Read) (Notes 3, 10, and 11)	t_{DSLDAL}	1.5 Clks	100 + 2.5 Clks	1.5 Clks	75 + 2.5 Clks	ns

NOTES:

1. Timing measurements are referenced to and from a low voltage of 0.8 volts and a high voltage of 2.0 volts, unless otherwise noted. The voltage swing through this range should start outside, and pass through, the range such that the rise or fall will be linear between 0.8 volts and 2.0 volts.
2. These specifications only apply if the MC68881 has completed all internal operations initiated by the termination of the previous bus cycle when DS was negated. Refer to **8.4 INTER-CYCLE TIMING RESTRICTIONS** for exceptions to this case.
3. Synchronous read cycles occur **only** when the save or response CIR locations are read.
4. This specification only applies to systems in which back-to-back accesses (read-write or write-write) of the operand CIR can occur. When the MC68881 is used as a coprocessor to the MC68020, this can occur when the addressing mode is Immediate.
5. If the $\overline{SIZE}$ pin is **not** strapped to either Vcc or GND, it must have the same setup times as do addresses **plus** 5 ns.
6. If the $\overline{SIZE}$ pin is **not** strapped to either Vcc or GND, it must have the same hold times as do addresses.
7. This number is reduced to 5 ns if $\overline{DSACK0}$ and $\overline{DSACK1}$ have equal loads.
8. $\overline{CS}$ must be either asserted or negated when $\overline{AS}$ is asserted.
9. $\overline{CS}$ must be either asserted or negated when $\overline{DS}$ is asserted (read).
10. $\overline{START}$ is not an external signal; rather, it is the logical condition that indicates the start of an access. The logical equation for this condition is: $\overline{START} = \overline{CS}\ (\overline{AS} + (R/\overline{W} \cdot \overline{DS}))$.
11. Value depends on actual clock input waveform used, not clock input specifications.

Timing Diagrams (Figures 10-3, 10-4, and 10-5)
are located on foldout pages at the end of this document.

SECTION 11
ORDERING INFORMATION AND MECHANICAL DATA

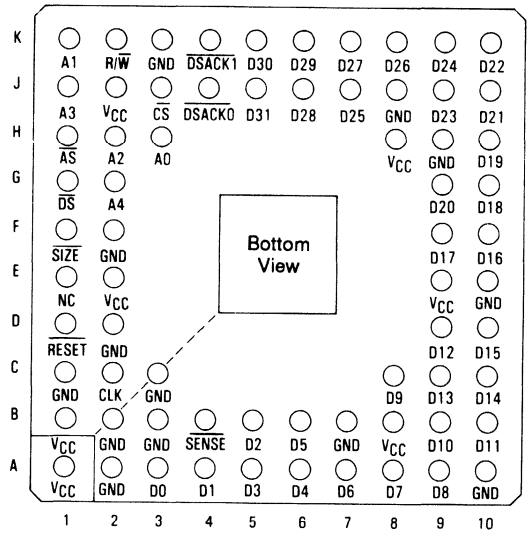
This section contains the pin assignments and package dimensions of the MC68881. In addition, detailed information is provided to be used as a guide when ordering.

11.1 STANDARD MC68881 ORDERING INFORMATION

Package Type	Frequency (MHz)	Temperature	Order Number
Pin Grid Array	12.5	0°C to 70°C	MC68881RC12
RC Suffix	16.67	0°C to 70°C	MC68881RC16

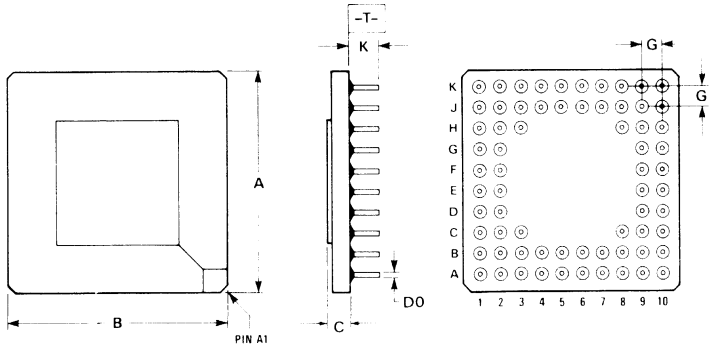
11.2 PIN ASSIGNMENTS

68-PIN GRID ARRAY



11.3 PACKAGE DIMENSIONS

RC SUFFIX
PIN GRID ARRAY
CASE 765A-03



NOTES:
1. DIMENSIONS A AND B ARE DATUMS AND T IS DATUM SURFACE.
2. POSITIONAL TOLERANCE FOR LEADS (68 PLACES)
   ⌖ ⌀ 0.13 (.005) Ⓜ T Ⓐ Ⓑ
3. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
4. CONTROLLING DIMENSION: INCH.

	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	26.67	27.17	1.050	1.070
B	26.67	27.17	1.050	1.070
C	1.91	2.66	0.075	0.105
D	0.43	0.60	0.017	0.024
G	2.54 BSC		0.100 BSC	
K	4.32	4.82	0.170	0.190

11

APPENDIX A
GLOSSARY OF TERMS

ALGORITHM A procedure for solving a mathematical problem in a finite number of steps that frequently involves repetition of an operation.

ASYMPTOTE A straight line associated with a curve such that as a point moves along an infinite branch of the curve the distance from the point to the line approaches zero and the slope of the curve at the point approaches the slope of the line.

BCD (Binary-Coded-Decimal) Number The representation of cardinal numbers 0 through 9 by 10 binary codes of any length. The minimum length is four and that there are over 29×10^9 possible four-bit BCD codes.

BIASED EXPONENT The sum of the exponent and a constant (bias) chosen to make the biased exponents range non-negative.

BINARY FLOATING-POINT NUMBER A bit string characterized by three components: a sign, a signed exponent, and a significand. (See single, double, and extended precision terms.) The numerical value of the bit string is the signed product of the significand and two raised to the power of the exponent.

DENORMALIZED NUMBER (1) Numbers having all zeros in the exponent and non-zero values in the fraction/mantissa. (2) A non-zero floating-point number whose exponent has a reserved value, usually the formats minimum, and whose explicit or implicit leading significand bit is zero.

DOUBLE PRECISION A 64-bit binary floating-point operand format composed of fields; one sign bit, an 11-bit biased exponent field, and an 52-bit fraction (significand) field.

DYAD Pair, an operator indicated by writing the symbols of two vectors without a dot or cross between (as AB).

DYADIC A sum of mathematical dyads.

DYADIC OPERATION An operation on two operands.

E FIELD See exponent (E field).

EXPLICIT FUNCTION A mathematical function defined by an expression containing only independent variables.

E FIELD See exponent (E field).

EXPLICIT FUNCTION A mathematical function defined by an expression containing only independent variables.

EXPONENT A symbol written above and to the right of a mathematical expression to indicate the operation of rising to a power.

EXPONENT (E FIELD) The component of a binary floating-point number that normally signifies the integer power to which two is raised in determining the value of the represented number. Occasionally the exponent is called the signed or unbiased exponent.

EXTENDED PRECISION A 96-bit binary floating-point operand format composed of four fields; one sign bit, an 15-bit biased exponent field, an16-bit undefined field, and an 64-bit mantissa (significand) field.

F FIELD See fraction (F field).

FIXED-POINT Pertaining to a numeration system in which the position of the point is fixed with respect to one end of the numerals, according to some convention.

FLOATING-POINT Pertaining to a system in which the location of the point does not remain fixed with respect to one end of the numerical expressions, but is regularly recalculated. The location of the point is usually given by expressing a power of the base.

FRACTION (F FIELD) The field of the significand that lies to the right of its implied binary point.

IMPLICIT FUNCTION A mathematical function that is not expressed with the dependent variable on one side of an equation and the one or more independent variables on the other. (in the expression $X + 3XY + Y = 0$, Y is an implicit function of X).

INTEGER Any of the natural numbers, the negatives of these numbers, or zero.

MANTISSA (1) The decimal part of logarithm. (2) Terms mantissa and significand are interchangable throughout this manual. See the term significand.

MODULO A mathematical operation that yields the remainder function of division. Thus 39 modulo 6 = 3.

MONADIC Unit, one.

MONADIC OPERATION An operation on one operand, for example, negation.

NAN (Not-A-Number) A symbolic entity encoded in floating-point format. There are two types of NANs; signaling and quiet. Signaling NANs signal the valid operation exception whenever appearing as operands. Quiet NANs propagate through almost every arithmetic operation without signaling exceptions.

NORMALIZE (1) To multiply a quantity by a suitable constant or scalar so that the quantity then has norm one; that is, the quantity norm is then equal to one. (2) Are numbers which can be expressed as floating-point operands where the exponent is neither all zeros nor all ones.

OPERAND That which is, or is to be operated upon. Note: An operand is usually identified by an address part of an instruction.

ORTHOGONALLY (1) Mutually perpendicular. (2) Having a sum of products or an integral that is zero or sometimes one under specified conditions: (a.) of real valued functions – having the integral of the product over a specific interval equal to zero. (b.) of vectors – having the scalar product equal to zero. (c.) of a square matrix – having the sum of products of corresponding elements in any two rows or any two columns equal to one if the rows or columns are the same and equal to zero otherwise, having a transpose with which the product equals the identity matrix. (d.) of a linear transformation – having a matrix that is orthogonal – preserving length and distance. (3) Composed of mutually orthogonal elements (basis of a vector space). (4) Statistically independent.

S BIT See sign bit (S field).

S FIELD See sign bit (S field).

SIGN BIT (S FIELD) Denotes the sign of the operand zero for positive and one for negative. Floating-point numbers are in sign-magnitude form, such that only the S bit is complemented in order to change the sign of the represented number.

SINGLE PRECISION A 32-bit binary floating-point operand format composed of three fields; one sign bit, an 8-bit biased exponent field, and an 23-bit fraction (significand) field.

SIGNIFICAND (1) As that component of a binary floating-point number which consists of an explicit or implicit leading bit to the left of the implied binary point and a fraction field to the right of the implied binary point. (2) Terms significand and mantissa are interchangable throughout this manual. (3) See fraction (F field).

TRANSCENDENTAL (1) Incapable of being the root of an algebraic equation with rational coefficients. (2) Being, involving, or representing a function (sin x, log x) that cannot be expressed by a finite number of algebraic operations.

UNNORMALIZED NUMBER An extended precision external operand that contains an explicit integer part bit (j) of zero and an exponent which is neither the formats maximum or minimum.

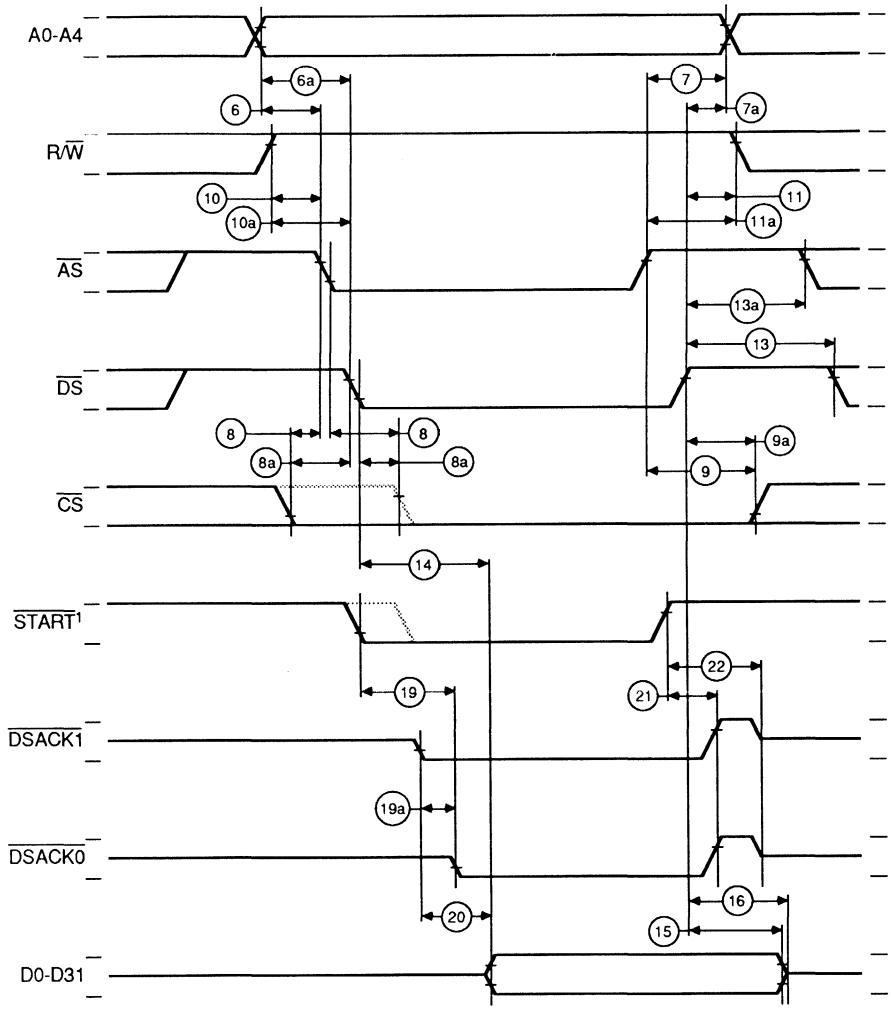
APPENDIX B
ABBREVIATIONS AND ACRONYMS

A	Address
Abs	Absolute
AEXC	Accrued EXCeption
ALU	Arithmetic Logic Unit
AS	Address Strobe
B	Byte integer
BCD	Binary Coded Decimal
BIU	Bus Interface Unit
BSUN	Branch/Set on Unordered
cc	condition code
CLK	CLocK
CMP	CoMPare
cp	coprocessor
CPU	Central Processor Unit
CPRED	Conditional PREDicate
CS	Chip Select
d	displacement
D	Data
D	Double precision binary real floating point
DIV	DIVide
DMA	Direct Memory Access
DS	Data Strobe
DSACK	Data and Size ACKnowledge
DZ	Divide by Zero
e	exponent
ea	effective address
ECU	Execution Control Unit
EQ	EQual
EXC	EXCeption
EXP	EXPonent
ENAB	ENABle

f	fraction
F	False
F	prefix for floating-point instruction (i.e., FADD)
FBcc	Floating-point Branch Branch on condition code
FDBcc	Floating-point Decrement and Branch on condition code
FP	Floating Point
FPCC	Floating-Point Condition Code
FScc	Floating-point Set on condition code
FTRAPcc	Floating-point Trap on condition code
GE	Greater than or Equal
GL	Greater than or Less than
GLE	Greater or Less or Equal
GND	GrouND
GT	Greater Than
I/O	Input/Output
I	Infinity
IADDR	Instruction ADDRess
ID	IDentification
Imm	Immediate
INEX	INEXact
INEX1	INEXact arithmetic
INEX2	INEXact conversion
IOP	Invalid OPeration
j	integer part
L	Long word integer
LE	Less than or Equal
LSB	Least Significant Bit/Byte
LT	Less Than
MANT	MANTissa
MCU	Microcode Control Unit
MOD	MODulo
MSB	Most Significant Bit/Byte
MUL	MULtiply
n	number
N	Negative
NAN	Not-A-Number
NEQ	Not EQual
NGE	Not Greater than or Equal

NGL	Not Greater or Less than
NGLE	Not Greater or Less or Equal
NGT	Not Greater Than
NLE	Not Less than or Equal
NLT	Not Less Than
OGE	Ordered Greater than or Equal
OGL	Ordered Greater or Less than
OGT	Ordered Greater Than
OLE	Ordered Less than or Equal
OLT	Ordered Less Than
OPERR	OPerand ERRor
OR	ORdered
OVFL	OVerFLow
P	Packed binary coded decimal real string
PC	Program Counter
QUOT	QUOTient
R/W	Read/Write
REM	REMainder
RM	Round toward Minus infinity
RN	Round to Nearest
RP	Round toward Plus infinity
RZ	Round toward Zero
s	sign
S	Single precision binary real floating point
SE	Sign of Exponent
SEQ	Signaling EQual
SF	Signaling False
SGL	SinGLe
SM	Sign of Mantissa
SNAN	Signaling NAN
SNEQ	Signaling Not EQual
SOC	Set On Condition
ST	Signaling True
SUB	SUBtract
T	True
TTL	Transistor-Transistor Logic

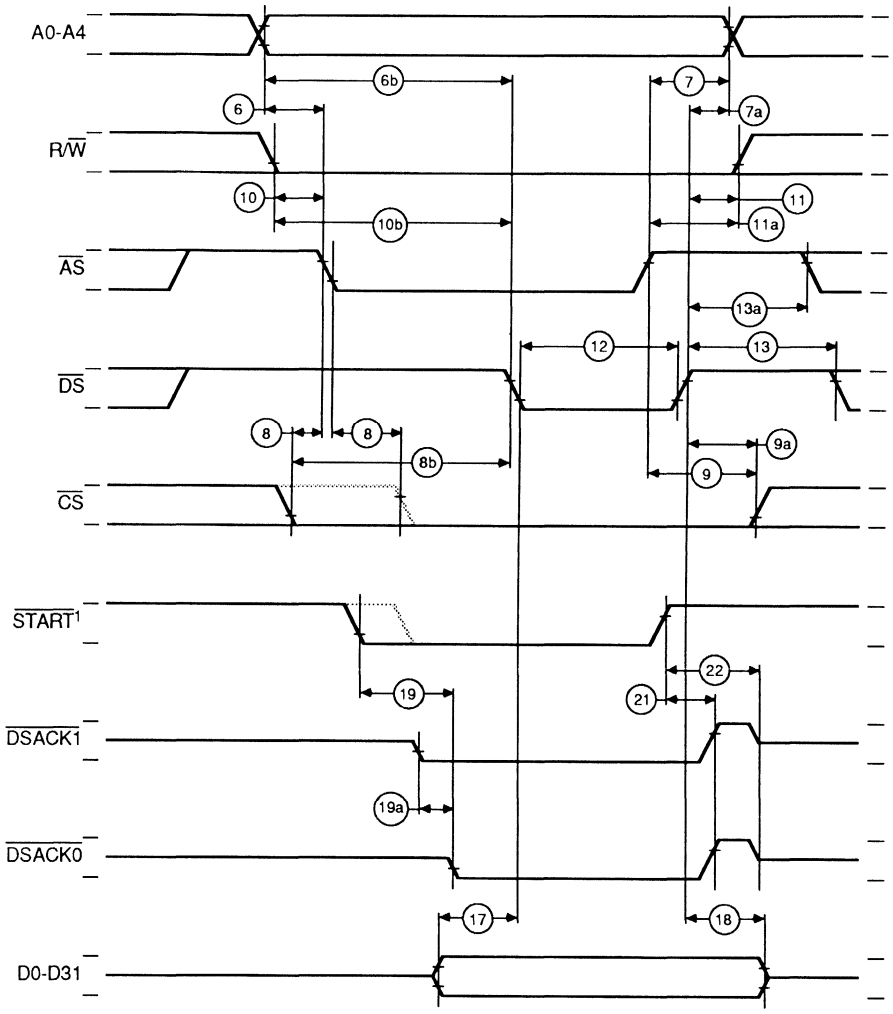
UEQ	Unordered or EQual
UGE	Unordered or Greater or Equal
UGT	Unordered or Greater
ULE	Unordered or Less or Equal
ULT	Unordered or Less Than
UNFL	UNderFLow
W	Word integer
x	don't care, irrelevant
X	eXtended precision binary real floating point
Z	Zero
$	hexadecimal
+inf	positive infinity
-inf	negative infinity



NOTES:

1. START is actually a logical condition, but is shown as an active low signal for clarity. The logical equation for this signal is START = CS (AS + R/W•DS).
2. Timing measurements are referenced to and from a low voltage of 0.8 volts and a high voltage of 2.0 volts, unless otherwise noted. The voltage swing through this range should start outside, and pass through, the range such that the rise or fall will be linear between 0.8 volts and 2.0 volts.

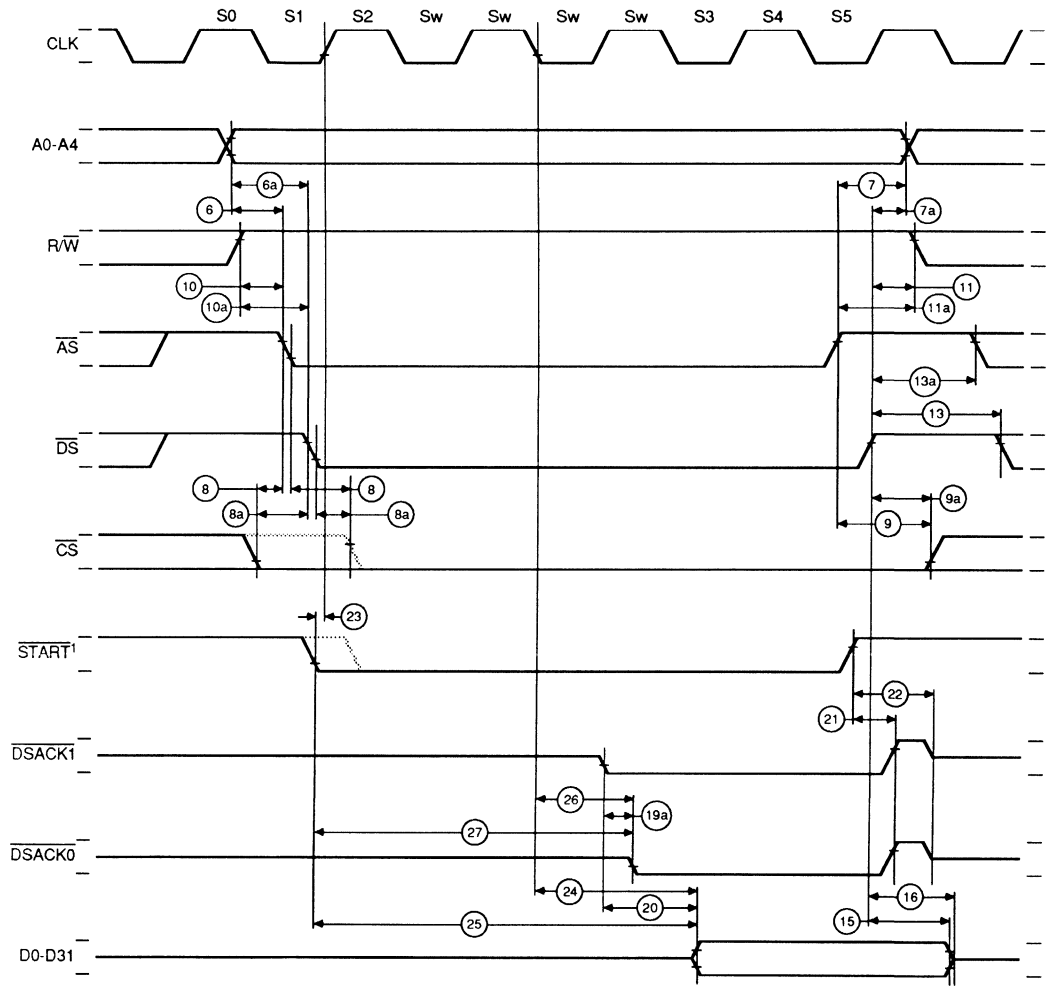
Figure 10-3. Asynchronous Read Cycle Timing Diagram



NOTES:

1. START is actually a logical condition, but is shown as an active low signal for clarity. The logical equation for this signal is START = CS (AS + R/W•DS).
2. Timing measurements are referenced to and from a low voltage of 0.8 volts and a high voltage of 2.0 volts, unless otherwise noted. The voltage swing through this range should start outside, and pass through, the range such that the rise or fall will be linear between 0.8 volts and 2.0 volts.

Figure 10-4. Asynchronous Write Cycle Timing Diagram



NOTES:

1. $\overline{START}$ is actually a logical condition, but is shown as an active low signal for clarity. The logical equation for this signal is $\overline{START} = \overline{CS}\ (\overline{AS} + R/\overline{W} \cdot \overline{DS})$.
2. Timing measurements are referenced to and from a low voltage of 0.8 volts and a high voltage of 2.0 volts, unless otherwise noted. The voltage swing through this range should start outside, and pass through, the range such that the rise or fall will be linear between 0.8 volts and 2.0 volts.

Figure 10-5. Synchronous Read Cycle Timing Diagram