This device, a hardware implementation of the proposed IEEE floating-point standard, can operate as a coprocessor on a 32-bit bus or as a peripheral on an 8- or 16-bit bus.

The MC68881 Floating-point Coprocessor

Clayton Huntsman and Duane Cawthron
Motorola, Inc.

Two things have happened that will have a strong influence on the design of modern numerical engines. First, a standard for floating-point arithmetic, now being proposed, is likely to become widely accepted. Second, currently available technology has made it economical to build VLSI floating-point processors. Here, we will present the highlights of the proposed IEEE standard for floating-point arithmetic[1,2] and discuss what its features do for real-number computations. We will briefly examine several ways to implement IEEE arithmetic, and we will then describe the MC68881, Motorola's floating-point processor for the M68000 family of microprocessors.

IEEE arithmetic

Motivations for standardizing floating-point arithmetic include the need for increased portability of numerical software and the need for graceful handling of anomalies and exceptions. Meeting these needs enables users with diverse numerical expertise to develop consistent and predictable software.

Data formats. Greater portability means that there must be a standard format to which programs using floating-point arithmetic must adhere. IEEE Task P754 proposes binary floating-point data formats, each consisting of a biased exponent, an unsigned mantissa, and a one-bit indicator for the sign of the mantissa. A mantissa consists of an explicit or implied leading bit to the left of the binary point and a fractional part to the right.

The proposed standard fully defines a 32-bit, single-precision format—which must be provided in all implementations—and a 64-bit, double-precision format. It also minimally specifies single-extended-precision and double-extended-precision formats, leaving it to the implementer to fully define them. If both of the nonextended formats are to be supported, and if one of the extended formats is to be provided as well, the standard requires that it be the wider of the two. The characteristics of the various formats are shown in Table 1.

The proposed standard also specifies the numerical characteristics for conversions between the specified binary formats and implementer-defined, decimal string formats. Because of the decimal conversion ranges, for which support is required by the standard, a minimal binary-coded decimal format is implied even though it is not explicitly specified.

Data types. The defined formats include five data types which permit representation of regular and particular numbers and of special situations.

0272-1732/83/1200-0044$01.00 © 1983 IEEE

Normalized numbers are the most common data type with which the user must deal. This type has the most significant bit of the mantissa positioned such that the one lies to the left of the radix point. In the nonextended formats, only the fractional part of the mantissa is stored in memory, which means that the most significant bit is implied and is equal to one. Thus, one more bit of significance can be extracted from the stored information than could be extracted otherwise. In the extended formats, which are required to meet the minimum fraction width specified by the standard, the integer bit may be either explicit or implied, but in either case it is equal to one.

Unlike a normalized number, a denormalized number, or denorm for short, has a nonzero mantissa with the most significant bit positioned somewhere to the right of the radix point. (Hence, the implied bit in nonextended formats is a zero.) A denorm's exponent is the format's minimum. Therefore, a denorm allows a graceful underflow to zero while maintaining some precision. (It is, in fact, sometimes possible to denormalize a number and not lose any precision.)

Zeroes are another data type. A number with a zero mantissa and the format's minimum exponent is a zero. Zeroes also have signs, which become important in mathematical packages such as those that provide complex arithmetic.

At the extremes of a format are the infinities. These numbers have zero fractions and the format's maximum exponent. Like zeroes, infinities are signed.

The last data type is called a NAN, for not-a-number. A NAN is simply what the name implies. It is a symbolic representation of a special number or situation in floating-point format. NANs include all numbers with nonzero fractions and the format's maximum exponent. There are two types of NANs—signaling and quiet. Quiet NANs provide a way to indicate uninitialized variables or illegal variable accesses as well as a means for floating-point routines to communicate error reports through successive calculations. Signaling NANs cause an exceptional condition to be reported. They were added so that extensions outside the scope of the proposed standard could be added. The standard does not define how a signaling NAN is distinguished from a quite NAN. The distinction is left to the implementer. For all operations required by the standard, the presence of one or more signaling NAN arguments causes an exceptional condition to be reported. A quiet NAN does not cause an exception. Instead, since the result of an operation with one or more NAN operands is a NAN, the information is propagated through any number of operations that may follow.

The various data types are summarized in Table 2.

Although these formats and associated data types allow numerical data to be transported from one machine to another, a floating-point system must be more than portable and more than just a format. It must ensure that the behavior of systems during format conversions is consistent. It must provide the user with a minimal set of exceptional conditions so that he can diagnose anomalies. The proposed IEEE floating-point standard defines such a system. It provides predictability and utility for the numerically naive user as well as for the expert. NANs and exceptional condition reporting provide debug and run-time diagnosis of packages developed under any implementation of the standard. Since Draft 10.0 of the standard requires that certain exceptions be announced by conforming sytems, users of such systems can easily trace algorithmic design flaws.

Exceptional conditions. There are five classes of exceptional conditions reported by conforming implementations. These are division by zero, underflow, overflow, inexact result, and invalid operation. Whenever an attempt is made to divide a finitely representable number by zero, the division-by-zero exception is reported. If an operation returns a result so small that it can be represented in the particular format only as a denorm and it loses significance during denormalization, the underflow exception is signaled. If an operation returns a rounded result that has an exponent greater than that of the largest finite number in the format, the overflow exception is reported. If an operation returns a rounded result that is imprecise, the inexact result is signaled. For instance, a number converted from BCD to single precision may not be exactly representable in the new format. If this is so, it is reported as an inexact result. Whenever arguments that will yield undefined results are presented to an operation, an invalid operation is signaled. Examples include operations such as attempting to multiply a zero times an infinity and attempting to divide two zeroes or two infinities. As stated above, the proposed standard requires all operations incorporating a signaling NAN to report exceptions. In these cases, the reporting vehicle is the invalid operation.

Basic operations. The proposed standard requires certain basic operations, including add, subtract, multiply, divide, find square root, find remainder, round to integer,

Table 1.
IEEE floating-point formats.

FORMAT	FIELD WIDTHS IN BITS		
	SIGN	EXPONENT	MANTISSA*
SINGLE	1	8	23
DOUBLE	1	11	52
SINGLE-EXTENDED	1	⩾11	⩾31
DOUBLE-EXTENDED	1	⩾15	⩾63

*THE MINIMUM NUMBER OF BITS SPECIFIED IN THIS FIELD REFLECTS AN IMPLIED INTEGER BIT IN THE MANTISSA.

Table 2.
IEEE floating-point data types.

TYPE	DESCRIPTION
NORMALIZED	MSB TO LEFT OF RADIX POINT, FORMAT'S EXPONENT RANGE
DENORMALIZED	MSB TO RIGHT OF RADIX POINT, MINIMUM EXPONENT
ZERO	ZERO MANTISSA, MINIMUM EXPONENT
INFINITY	ZERO FRACTION, MAXIMUM EXPONENT
NOT-A-NUMBER	NONZERO FRACTION, MAXIMUM EXPONENT

and compare. It specifies conversion between different floating-point formats, conversion between floating-point and integer formats, and conversion between binary and decimal formats. It also stipulates rules for the various rounding methods for these operations.

The proposed IEEE floating-point standard addresses the need for a functionally superior, floating-point methodology by providing various formats, regular error-handling mechanisms, fundamental operations, and simplicity. Compliance by major corporations will mean that scientific and business users will enjoy numerical software that is more producible, portable, and maintainable.

Implementing the proposed IEEE standard

The following discourse examines three general implementation schemes that are within the framework of the specification. The standard allows for software, hardware, or hybrid solutions to the implementation problem. It also stipulates that no hardware portion of a hybrid shall be said to conform apart from its supporting software.

So far we have discussed only the more salient features of the standard. Even so, it should already be obvious that a significant amount of overhead is required before and after the execution of each operation. We would expect, therefore, that a conforming software implementation would run slower than other, nonconforming software floating-point implementations currently in place. This is in fact the case. In an exhaustive evaluation of a conforming software package developed at Motorola for internal use, it was found that a significant portion of an operation's total execution time involved exception checking, preservation of rounding direction, and the like. Such performance degradation may make it difficult to convince some users of non-IEEE software implementations to convert to the philosophy of the proposed standard.

For VLSI implementations, at least, the hybrid approach could use a processor to perform certain kernel functions such as add, subtract, multiply, and divide, and rely on a software envelope to handle most of the special cases. If the hardware was able to multiply two normalized numbers, for example, the software would check the operands of each multiply instruction for zeroes, infinities, denorms, and NANs. Only normalized numbers might actually be passed to the hardware. This method definitely would be faster than a software-only approach and would be relatively inexpensive. Users could achieve the benefits provided by IEEE arithmetic while increasing system performance. However, they would still have to develop commonly used functions such as trigonometric and transcendental routines. These routines would represent a burden in addition to that imposed by the standard functions, and they would further degrade the overall performance of the hybrid implementation.

It seems, therefore, that if hardware could be used to solve part of the implementation problem, it should be allowed to provide a total solution. This means that special cases could be handled in an efficient manner. It means that the problems associated with the integration of commonly used functions such as trigonometrics and transcendentals could be handled with built-in expertise. And it means that maximum performance could be attained for about what a complete hybrid system would cost.

Partial list of M68000 family signals

SIGNAL	DESCRIPTION
A0-A31	Address bus
FC0-FC2	Processor status (function code)
D0-D31	Data bus
R/W	Read/write
AS	Address strobe
DS	Data strobe
UDS	Upper data strobe
LDS	Lower data strobe
CS	Chip select
SIZE	Bus size
DSACK0	Data and size acknowledge
DSACK1	Data and size acknowledge
DTACK	Data transfer acknowledge
RESET	Reset

The MC68881 floating-point machine

The MC68881, a single-chip HCMOS VLSI processor, is a hardware implementation of the proposed IEEE standard for floating-point arithmetic. It is available in either a 64-pin DIP or a 68-pin grid array. It not only provides those features required by the standard but includes most of the optional features suggested by it as well. It needs no software envelope. Its instruction set is a logical extension of the M68000 family architecture. This allows it to be used as a coprocessor with the next generation of Motorola microprocessors or as a peripheral for other central processing units.

The MC68881 is most effectively utilized when it is configured as a coprocessor for the MC68020, a 32-bit, general-purpose microprocessor (see Figure 1). The MC68020 and MC68881 contain bus interface units that utilize a protocol that allows them to exchange machine instructions, operands, and results along a 32-bit data bus. By executing asynchronously, the bus interfaces permit the MC68020 and the MC68881 to run at different clock speeds and to execute many operations concurrently. Because of the generality of its bus interface, the MC68881 can also be configured as a peripheral for the MC68008[3] (Figure 2) or as a peripheral for the MC68000[4] and MC68010[5] (Figure 3).

The following paragraphs discuss the protocol required to interface the floating-point coprocessor to an M68000 system. It should be understood that the operation word and extension word or words are peculiar to the M68000 family of processors. Nonfamily members as well as non-Motorola processors need only observe the actions required to pass the command word(s) to the coprocessor and to perform the necessary responses.

When the MC68000 was designed, certain operation code combinations were reserved for future expansion.

Of course, at that time the intent of the reserved instructions was not known, so these operation codes simply initiated traps. Later, the F-line codes (those with the uppermost four bits set to ones, or hexidecimal F) were relegated to the class of instructions that supports coprocessors.

Members of the microprocessor family supporting coprocessors (e.g., the MC68020) recognize coprocessor instructions and perform the protocol required for coprocessor instruction sequencing. The other members of the family (i.e., the MC68000, MC68008, and MC68010) simply perform an F-line trap and rely on the software to complete the interface between the machines. The coprocessor in the trapping instance is characterized as a peripheral.

An M68000 coprocessor instruction (see Figure 4) consists of an F-line operation word, a possible coprocessor command word, and zero or more extension words. The coprocessor command word (or words) provides the operation code specifically for the coprocessor. The operation word consists of four fields. The first field is the F-line indicator. The coprocessor identification, or

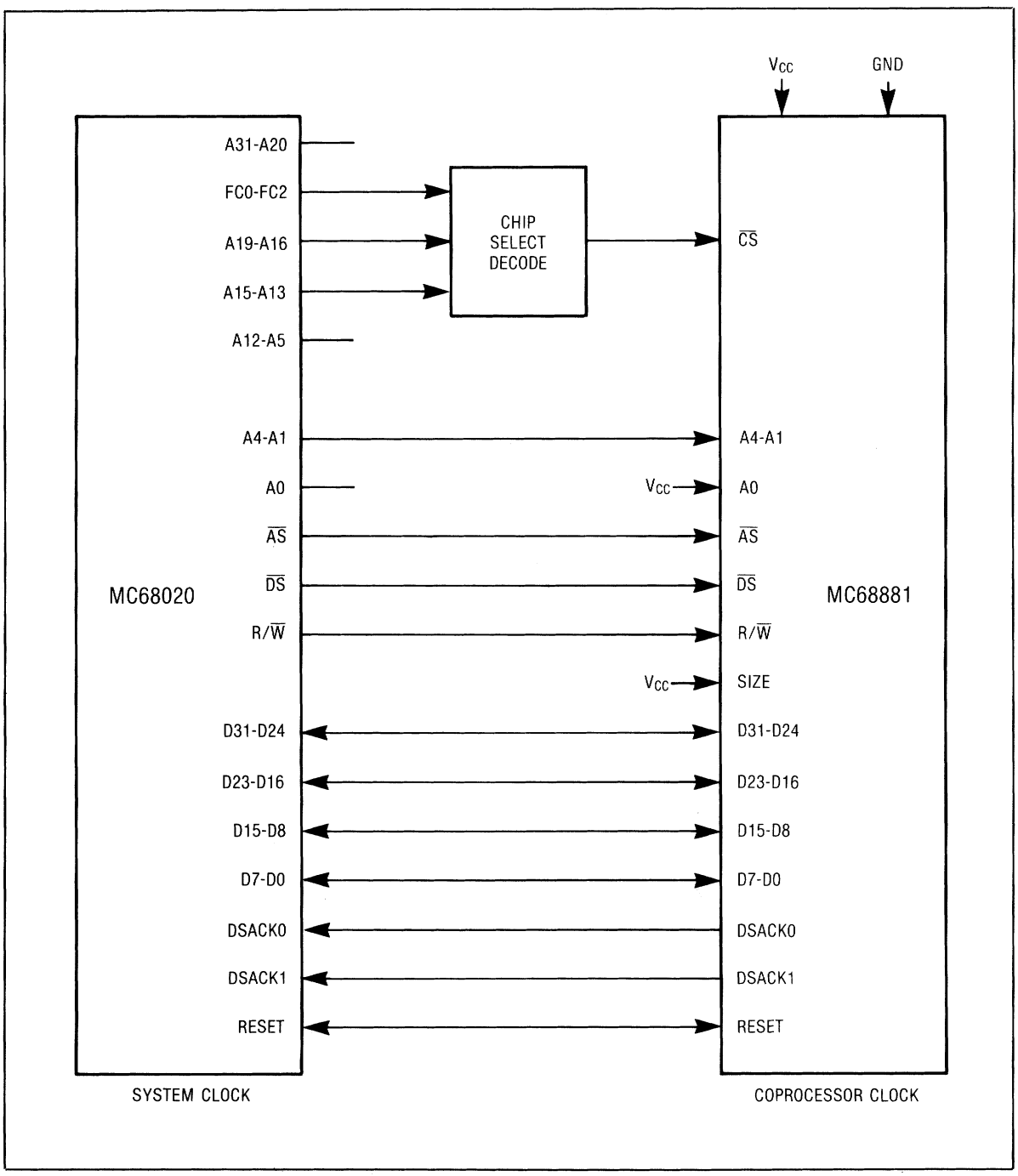


Figure 1. The MC68881 used as a coprocessor on a 32-bit data bus.

CP-ID, field determines the particular coprocessor for which the instruction is intended. Up to eight coprocessors may exist in a system. The MC68881 can be one of eight different coprocessors, or all eight coprocessors can be MC68881s. The type field declares the class of coprocessor instruction and includes the general type (for the most common coprocessor operations), the conditional type, the save type, and the restore type. The modifier field provides additional information about a particular operation word type—the modifier for the general instruction type, for example, is an effective address field. (This modifier can also specify that additional extension words are needed to determine the effective address.)

Coprocessor instruction processing begins with the fetching of the F-line instruction. If the main processor does not directly support the coprocessor interface, the F-line trap handler is entered. In any case, the main processor must then interpret the operation word and write the command word to the coprocessor command register. The MC68881 must then decode the command word and provide a response for the host machine in the coprocessor response register. The main processor then reads the response register. The register indicates either that no further action is required or that some additional service must

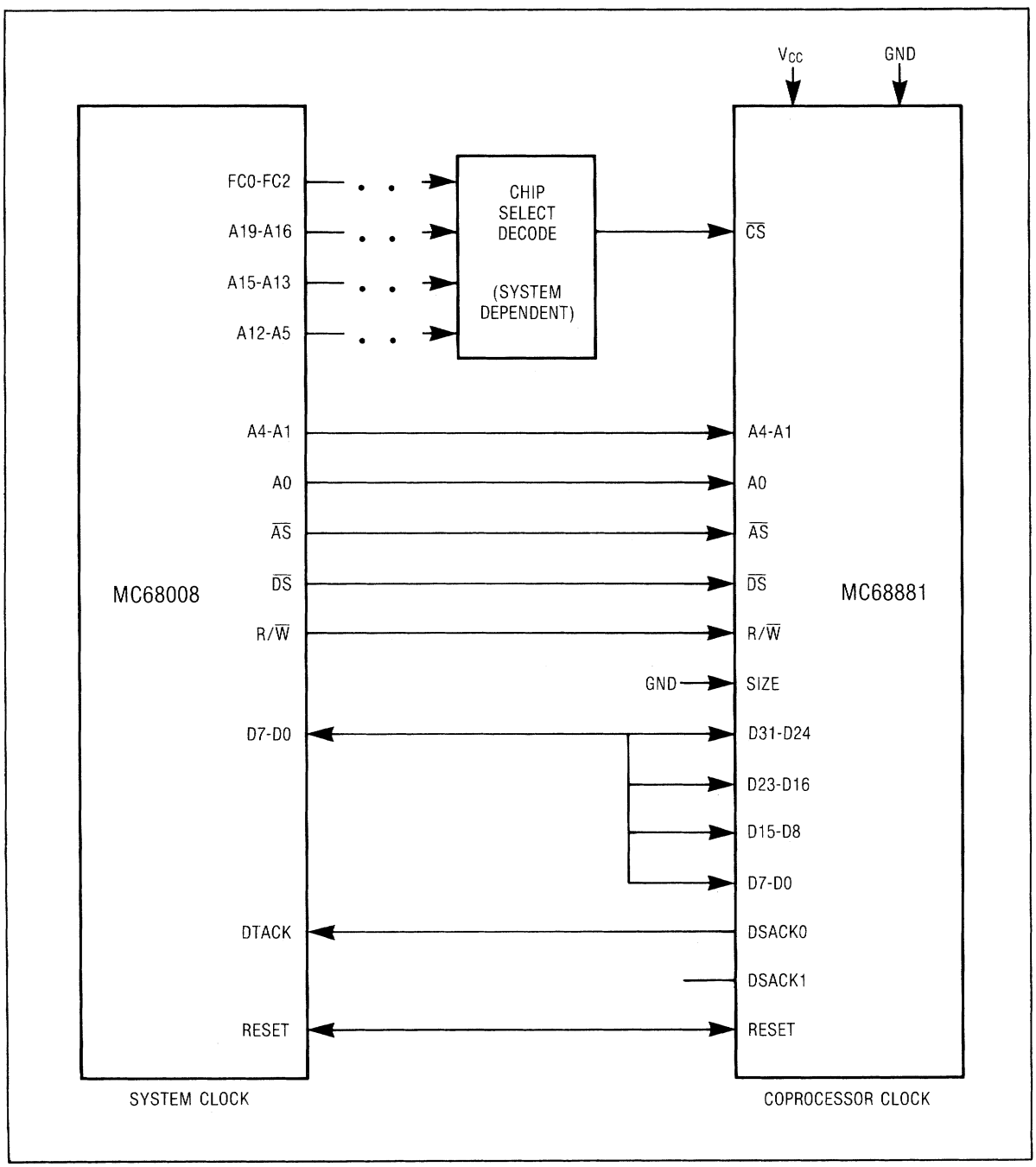


Figure 2. The MC68881 used as a peripheral on an 8-bit data bus.

be performed. Such a service may be the evaluation of an effective address or the commencement of exception processing.

In the M68000 family, coprocessor interface registers are, by definition, memory-mapped. Therefore, the protocol for coprocessor operation is the same whether the MC68881 is used as an M68000 family coprocessor or peripheral. Because the burden of operation word decoding and effective address evaluation is on the main processor, instruction sequencing external to the coprocessor is necessarily slower when the part is used as a peripheral. However, once the host-to-coprocessor protocol software has been installed, the architecture of the host processor has been effectively extended with additional instructions and register resources.

Programmer's model. The MC68881 programmer's model appears as an extension of the host processor and provides eight floating-point data registers, a status register, a control register, and an instruction address register (Figure 5).

Floating-point registers. The floating-point data registers are 80-bit, extended-procision registers. All floating-point operations are performed via these

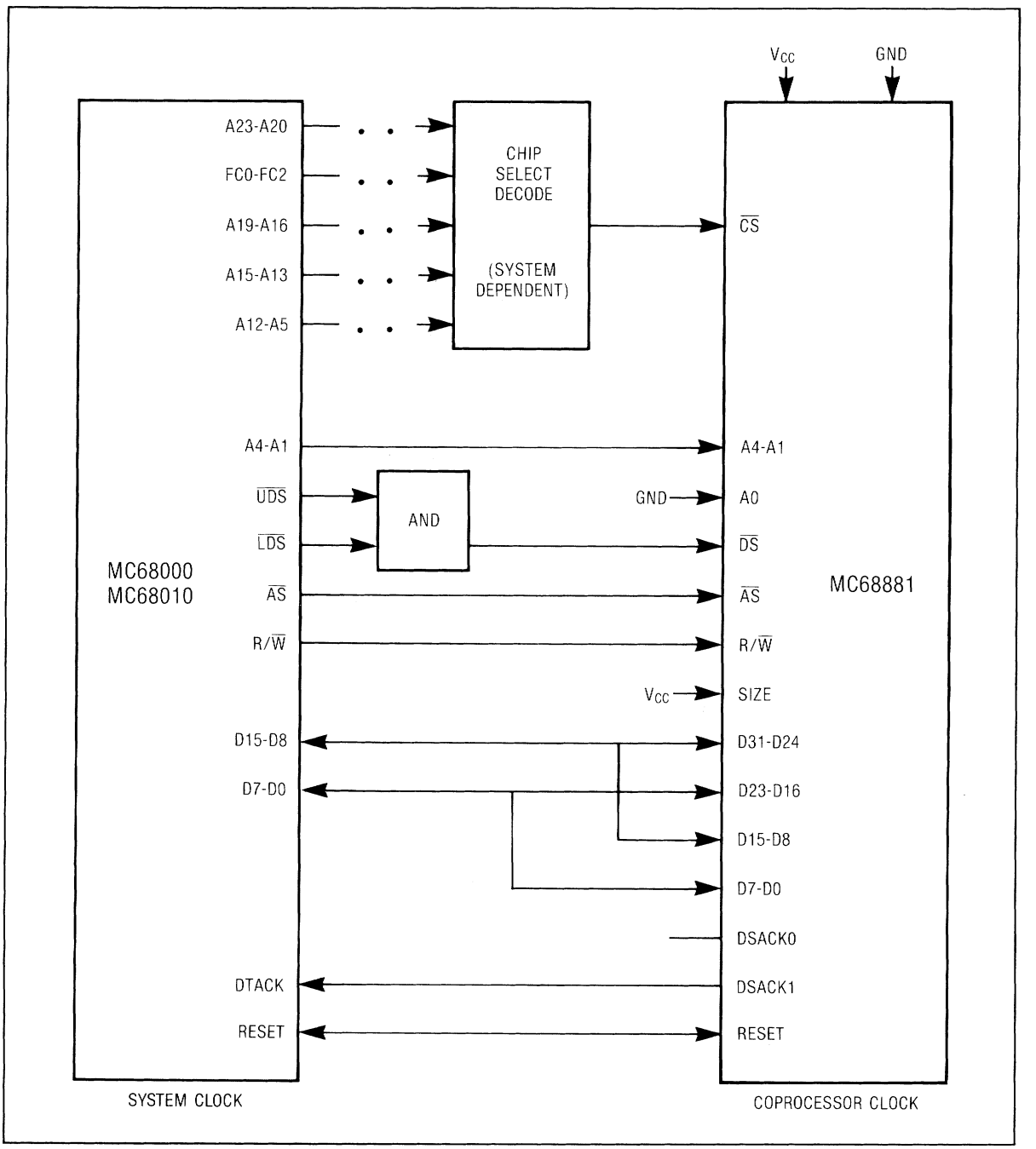


Figure 3. The MC68881 used as a peripheral on a 16-bit data bus.

registers. During execution of a floating-point instruction, the MC68881 execution unit obtains its operands from the floating-point register block, an MC68020 data register, or a memory area. It performs the calculation specified by the floating-point coprocessor command word and then deposits the result into a floating-point register. (Exceptions to this are the move-out instructions.) The result may be left in the register for another calculation, or it may be moved off the chip.

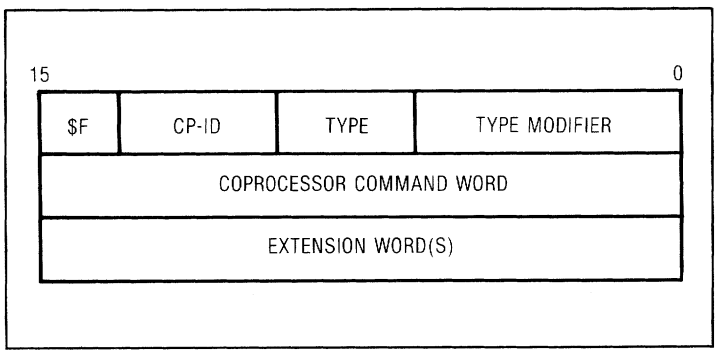


Figure 4. MC68881 instruction format.

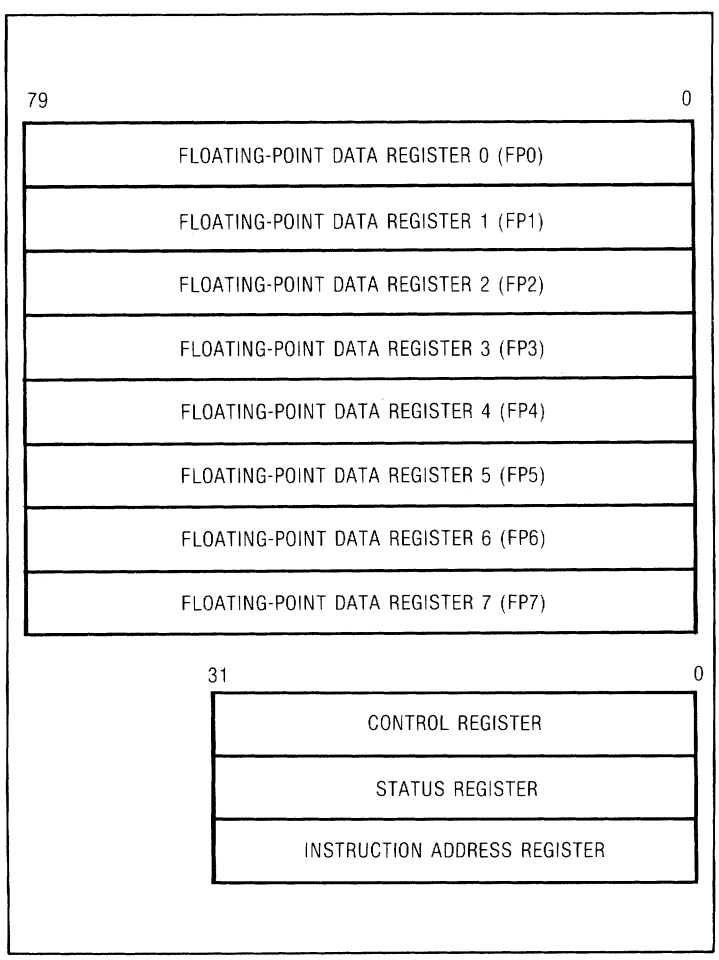


Figure 5. MC68881 programmer's model.

Status register. In the MC68881, the status register

- reports exceptions which have occurred during the execution of an instruction,
- indicates exceptions which have accrued since the last time the exception history was cleared,
- reports floating-point condition codes, and
- provides a portion of the quotient generated by the modulo and IEEE remainder instructions.

An exception is reported in the exception byte. Each bit position in this byte represents a particular exception type. Any exception that occurs during the execution of an instruction sets the appropriate bit. The exception byte is cleared before the start of any instruction capable of causing an exception.

The floating-point standard requires conforming implementations to maintain a history of exception occurrences. The control of this history must be under the user's auspices. The MC68881 incorporates the accrued exception (AEXC) byte to accomplish this. Whenever an exception occurs, the corresponding bit in the AEXC byte is set to one and remains set until the user explicitly clears it.

The value of the AEXC bytes lies in the way it simplifies exception handling. During expression evaluation, it is not always desirable to trap just because an exception has occurred, nor is it desirable to poll for the occurrence of an exception at the completion of each instruction. The accrued exception byte allows the user to check only once, at the end of a set of calculations, for the occurrence of an exception. Table 3 shows the exceptions that can be recorded in the accrued exception byte.

A quotient byte is also provided in the status register. The information written into the quotient byte is provided by the modulo and IEEE remainder operations. The quotient byte provides a sign-magnitude quantity that may be used by user-defined periodic functions to classify the portion of the period in which the remainder lies. The least significant seven bits of the quotient are placed in the least significant bits of the quotient byte, while the sign information (a one indicates that the quantity is negative) is placed in the most significant bit of the quotient byte.

The condition code byte is also contained in the status register. It indicates the data type of the result of a move to a floating-point data register and the data type of the result of a floating-point arithmetic instruction. The IEEE floating-point standard requires only the determination of the data type resulting from a compare operation. However, data type determination at the completion of an instruction allows conditional branching, conditional seting of a byte, decrementing and branching on condition, and trapping on condition to occur without an explicitly performed compare operation. The MC68881 implements all conditional predicates specified by the proposed standard.

Control register. The control register is written to enable or disable traps on exceptions, to select the round-

ing mode, or to select the rounding precision. It and the status register are shown in Figure 6.

When an exceptional condition has occurred, as indicated in the exception byte of the status register, either program execution continues unabated or exception processing begins. The choice of program flow is determined by the value of the exception enable byte. For each exception indicated in the status register exception byte, there is a corresponding exception enable bit in the control register exception byte. When an exception occurs and the associated enable for the exception is set, exception processing begins in the host processor.

The control register also contains a byte of information that specifies the precision (single, double, or extended) to which a floating-point result will be rounded (Table 4). Rounding to single or double precision facilitates the emulation of machines that support only single-precision or double-precision calculations. This byte also specifies which of four rounding modes is used. The rounding modes allow the floating-point mantissa to be adjusted by one unit in the last place, according to the rounding rules for the mode selected.

The four rounding modes are shown in Table 5. Round-to-nearest rounds to the nearest representable value, unless the result is exactly halfway between two consecutive points. In that instance the result becomes the value having a zero in the least significant bit position (the even value). Round-toward-zero simply truncates the result. This is the rounding mode inherent in Fortran programs. Round-toward-plus-infinity and round-toward-minus-infinity force the result to be rounded in the direction of the associated infinity. This rounding mode is used in interval arithmetic, which can be employed to determine the largest interval on which an infinitely precise answer lies.

Instruction address register. The last element in the programmer's model is the instruction address register, a pointer to the last floating-point instruction executed. The host processor uses this pointer to locate an instruction that has caused an exception. (Were it not for this pointer, the host—because it operates concurrently with the coprocessor—might not be able to locate such an instruction.)

Data formats and types. Internal number representation in the MC68881 is in an extended format conforming to the IEEE specification. However, representation of data that are moved and converted outside the device can be in the IEEE format or in any of a number of other supported formats. Each datum, regardless of initial memory-resident format, is coerced into the extended floating-point format as it is brought on board the processor. Once the internal set of operations has been performed and the final datum is ready to be moved into any of the other supported formats. (This scheme inherently supports coercion in high-level languages.)

Table 3.
MC68881 accrued exception byte.

EXCEPTION	DESCRIPTION
IOP	INVALID OPERATION
OVFL	OVERFLOW
UNFL	UNDERFLOW
DZ	DIVISION BY ZERO
INEX	INEXACT RESULT

Table 4.
MC68881 rounding precision.

ROUNDING PRECISION	DESCRIPTION
SGL	ROUND TO SINGLE PRECISION
DBL	ROUND TO DOUBLE PRECISION
EXT	ROUND TO EXTENDED PRECISION

Table 5.
MC68881 rounding modes.

ROUNDING MODE	DESCRIPTION
RN	ROUND TO NEAREST EVEN
RZ	ROUND TOWARD ZERO
RP	ROUND TOWARD PLUS INFINITY
RM	ROUND TOWARD MINUS INFINITY

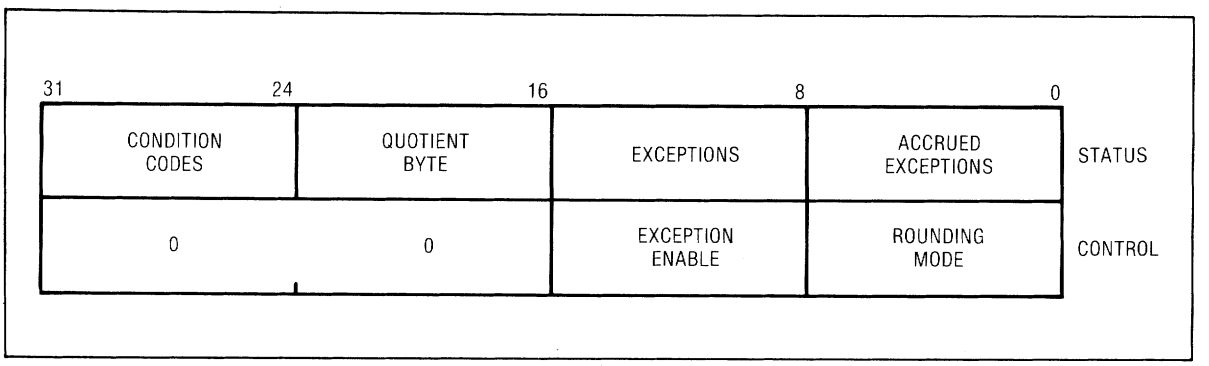


Figure 6. MC68881 status and control registers.

Table 6.
Move instructions.

INSTRUCTION	DESCRIPTION
FMOVE	FLOATING-POINT MOVE TO/FROM FP DATA REGISTER
FMOVE	MOVE TO/FROM MC68881 CONTROL/STATUS REGISTER
FMOVE	MOVE TO/FROM MC68881 IADDR REGISTER
FMOVECR	FLOATING-POINT MOVE FROM ON-CHIP CONSTANT ROM
FMOVEM	FLOATING-POINT MOVE MULTIPLE TO/FROM CONTROL/ STATUS/IADDR REGISTERS
FMOVEM	FLOATING-POINT MOVE MULTIPLE TO/FROM FP DATA REGISTERS

Table 7.
Conditional instructions.

INSTRUCTION	DESCRIPTION
FBcc	FLOATING-POINT CONDITIONAL BRANCH
FDBcc	FLOATING-POINT CONDITIONAL TEST, DECREMENT, AND BRANCH
FScc	FLOATING-POINT CONDITIONAL SET
FTcc	FLOATING-POINT CONDITIONAL TRAP
FTPcc	FLOATING-POINT CONDITIONAL TRAP WITH PARAMETER

Seven data formats are supported. Byte, word, and long integers are 8, 16, and 32 bits long, respectively. These are the same integer formats supported by the M68000 family of central processing units.

Single-, double-, and extended-precision binary floating-point numbers are 32, 64, and 80 bits long, respectively. Extended-precision numbers external to the MC68881 (those residing in the memory space) are actually padded with zeroes so that they will occupy 96 bits.

The MC68881 provides a packed binary-coded decimal format which fully supports the conversion operations described by the proposed standard. Each number in this format occupies a 96-bit region in memory space (see Figure 7). The specifications for conversion between this format and any other format are stringent and involve complicated algorithms. Therefore, hardware support of BCD conversions relieves the programmer of a large burden. In addition, hardware support of decimal strings assists financial calculations in business programming languages.

The MC68881 supports all of the IEEE-specified data types. Ordinary numbers are represented by the normalized data type, and gradual underflow is provided by the denormalized data type. Special data types are plus and minus zero, plus and minus infinity, and not-a-number (NAN).

Instruction set. Besides the floating-point arithmetic operations required by the proposed standard, the MC68881 instruction set includes operations for moving data into and out of the machine, for branching, and for supporting virtual memory. It also includes a full set of trigonometric and transcendental functions.

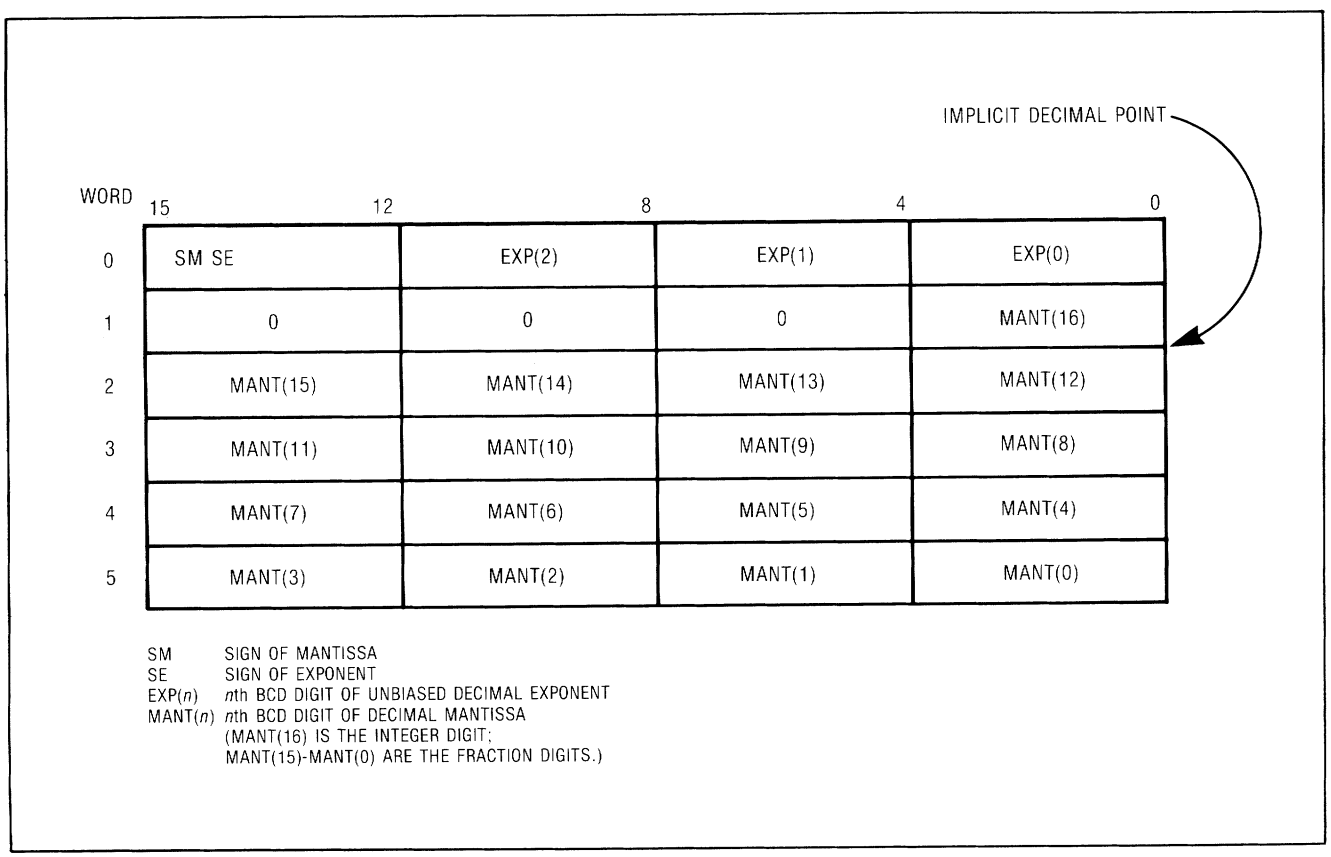


Figure 7. MC68881 decimal floating-point data format.

Floating-point move instructions (Table 6) allow data to be transferred into and out of the floating-point data, control, status, and instruction address registers. Each operand is converted to extended precision when it is moved into a floating-point data register. When the operand is moved out of the floating-point data register and into memory, it is converted to the format of the destination. This permits calculations between arguments having differing formats.

Move multiple register (FMOVEM) instructions speed the movement of data to or from multiple registers. One instruction can move data into or out of any or all of the floating-point data registers. Another instruction can move data into or out of any or all of the control, status, and instruction address registers.

Frequently used constants such as pi and one are stored in the MC68881 execution unit and are made available to the user by the move constant ROM (FMOVECR) instruction. This instruction always moves information from an internal read-only store to one of the floating-point data registers.

The branch according to condition instruction (FBcc) and the trap according to condition instructions (FTcc and FTPcc) alter the flow of a program according to the floating-point condition code byte in the status register. The set according to condition instruction (FScc) allows a byte to be either set or cleared according to the floating-point condition codes. If the condition is true, the byte specified by the effective address is set to all ones; if the condition is false, the byte is set to all zeroes. The conditional instructions are listed in Table 7.

The MC68881 supports monadic and dyadic floating-point operations (Tables 8 and 9). Monadic operations are those that take only one input operand. Square root and the trigonometric and transcendental functions are in this category. The operand is taken from memory, from a floating-point register, or from an MC68020 data register, and the result is placed in a floating-point register called the destination register. Dyadic operations are those that require two operands. One operand is taken from memory, from a floating-point register, or from an MC68020 data register; the other is always taken from one of the floating-point registers, which is also the destination address for the result of the operation.

Trigonometric and transcendental instructions are not specified by the standard. However, the MC68881 instruction set includes a group of trigonometric and transcendental functions commonly used in high-level-language libraries (Table 10). Each instruction in this group is completely supported by input bounds checking and exception processing in the same manner that the IEEE-defined functions are. Each supported instruction accepts inputs over its domain. Each is completely defined and requires no additional software envelope.

For speed-critical applications requiring only single-precision calculations, the MC68881 provides single-precision multiply (FSGLMUL) and divide (FSGLDIV) instructions (Table 11). These ignore the rounding precision selected by the user and produce results rounded correctly to single precision. Single-precision versions of add and subtract are not included, since no significant improvement in throughput would be observed.

Table 8.
Floating-point monadic instructions.

INSTRUCTION	DESCRIPTION
FABS	FLOATING-POINT ABSOLUTE VALUE
FGETEXP	GET FLOATING-POINT EXPONENT
FGETMAN	GET FLOATING-POINT MANTISSA
FINT	FLOATING-POINT INTEGER PART
FNEG	FLOATING-POINT NEGATE
FNOP	FLOATING-POINT NO OPERATION
NSCALE	FLOATING-POINT SCALE EXPONENT BY INTEGER
FSQRT	FLOATING-POINT SQUARE ROOT
FTST	FLOATING-POINT TEST

Table 9.
Floating-point dyadic instructions.

INSTRUCTION	DESCRIPTION
FADD	FLOATING-POINT ADD
FCMP	FLOATING-POINT COMPARE
FDIV	FLOATING-POINT DIVIDE
FMOD	FLOATING-POINT MODULO REMAINDER
FMUL	FLOATING-POINT MULTIPLY
FREM	FLOATING-POINT IEEE REMAINDER
FSUB	FLOATING-POINT SUBTRACT

Table 10.
Trigonometric and transcendental instructions.

INSTRUCTION	DESCRIPTION
FACOS	FLOATING-POINT ARC COSINE
FASIN	FLOATING-POINT ARC SINE
FATAN	FLOATING-POINT ARC TANGENT
FATANH	FLOATING-POINT HYPERBOLIC ARC TANGENT
FCOS	FLOATING-POINT COSINE
FCOSH	FLOATING-POINT HYPERBOLIC COSINE
FETOX	FLOATING-POINT *e* TO THE *x*
FETOXM1	FLOATING-POINT (*e* TO THE *x*) MINUS 1
FLOG10	FLOATING-POINT LOG TO THE BASE 10
FLOG2	FLOATING-POINT LOG TO THE BASE 2
FLOGN	FLOATING-POINT LOG TO THE BASE *e*
FLOGNP1	FLOATING-POINT LOG TO THE BASE *e* OF (*x* + 1)
FSIN	FLOATING-POINT SINE
FSINCOS	SIMULTANEOUS FLOATING-POINT SINE AND COSINE
FSINH	FLOATING-POINT HYPERBOLIC SINE
FTAN	FLOATING-POINT TANGENT
FTANH	FLOATING-POINT HYPERBOLIC TANGENT
FTENTOX	FLOATING-POINT TEN TO THE POWER *x*
FTWOTOX	FLOATING-POINT TWO TO THE POWER *x*

Table 11.
Miscellaneous instructions.

INSTRUCTION	DESCRIPTION
FSGLMUL	FLOATING-POINT SINGLE-PRECISION MULTIPLY
FSGLDIV	FLOATING-POINT SINGLE-PRECISION DIVIDE
FSAVE	FLOATING-POINT SAVE
FRESTORE	FLOATING-POINT RESTORE

Virtual memory support requires a processor to be able to stop in the middle of an instruction that is trying to access a logical address not currently mapped into a physical memory location. Since a memory fault can occur during a floating-point operand fetch, the floating-point coprocessor, as well as the main processor, must provide this capability.

The MC68881 can be halted during the execution of an instruction. Its current internal state can be saved and later restored so that instruction execution can be restarted from its stopping point. FSAVE and FRESTORE provide this support to virtual memory systems (see again Table 11). Three sizes of internal state can be managed. If the coprocessor has not executed an instruction since it was last reset, FSAVE causes a minimal amount of the internal state to be preserved in the memory space. More information must be saved if the machine has executed an instruction since it was last reset and is currently idle. A maximum amount of the internal state must be saved whenever the coprocessor is interrupted while executing an instruction. FRESTORE allows for saved-state movement back to the MC68881.

Exception processing. The MC68881 supports the handling of all IEEE-defined exceptions. There are also other exceptions that can occur during system operation. These include exceptions arising from bus interface protocol violations and from illegal coprocessor commands. The protocol violation is the highest-priority exception and is considered a fatal error. Because of this, it preempts all other exceptions. Illegal commands are reported by the coprocessor, since it decodes the command portion of a floating-point instruction.

The IEEE proposal for standardizing arithmetic will bring stability to the floating-point world. Programmers will be able to create software which will be safe and reliable and which will provide consistent answers when moved across system boundaries. Systems designers requiring high-performance mathematical computation at low cost will be able to take advantage of hardware implementations of the standard such as the MC68881. The rich instruction sets and advanced system features the standard will make possible will provide architectural extensions for generations of central processing units to come. ■

References

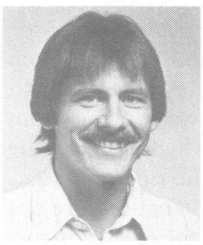
1. "A Proposed Standard for Binary Floating-point Arithmetic" (Draft 8.0 of IEEE Task P754), *Computer,* Vol. 14, No. 3, Mar. 1981, pp. 51-62.
2. "A Proposed Standard for Binary Floating-point Arithmetic" (Draft 10.0 of IEEE Task P754), Dec. 2, 1982, 17 pp. (Available from David Stevenson, Chairman, IEEE Task P754, c/o Zilog, 1315 Dell Ave., Campbell, CA 95008.)
3. W. Browne, Jr., and B. Moyer, "μP Fits 16-bit Performance into 8-bit Systems," *Electronic Design,* Vol. 30, No. 8, April 15, 1982, pp. 183-187.
4. E. Stritter and T. Gunter, "A Microprocessor Architecture for a Changing World—The Motorola 68000," *Computer,* Vol. 12, No. 2, Feb. 1979, pp. 20-29.
5. D. MacGregor and D. S. Mothersole, "Virtual Memory and the MC68010," *IEEE Micro,* Vol. 3, No. 3, June 1983, pp. 24-39.

For further reading

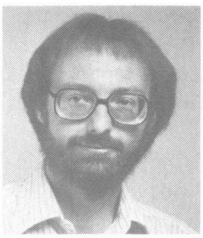
J. Boney, V. Shahan, and P. Harvey, "Floating-point Power for the M68000 Family," *Proc. Maecon,* Sept. 1983, Vol. 9/2, pp. 1-9.

J. T. Coonen, "An Implementation Guide to a Proposed Standard for Floating-point Arithmetic," *Computer,* Vol. 13, No. 1, Jan. 1980, pp. 68-79.

MC68881 Design Specifications, Motorola, Inc., Austin, TX, July 1983.



Clayton Huntsman, a former submariner and nuclear reactor operator, has been working since 1977 in Motorola's Microprocessor Design Group in Austin, Texas. His primary responsibility is transcendental algorithm implementation in the MC68881 project. He is interested in alternative computer architectures and artificial intelligence, and he enjoys delivering soapbox orations concerning the shortage of user-friendly software and operating systems. His hobbies include running, swimming, and bicycling.



Duane Cawthron has been working since 1979 in Motorola's Microprocessor Design Group in Austin, Texas. He is currently involved with the system-level design of the MC68881. His interests include microcoded instruction sequencers, efficient VLSI design methodology, and the Unix operating system. He received his BSEE from the University of Texas at Austin in 1979.

The authors' address is MOS Microprocessor Design Group, Mail Drop M2, Motorola MOS Integrated Circuits Division, 3501 Ed Bluestein Blvd., Austin, TX 78721.