Extension of the MC68000 Architecture to Include Standard Floating-point Arithmetic

Gregory Walker

Motorola, Inc., Microprocessor Design, Mail drop M2880
3501 Ed Bluestein Blvd., Austin, TX 78712
(512) 928-6212

ABSTRACT

The synthetic aspect of designing a computer architecture is particularly evident when the design is highly constrained from two independent directions. Floating-point extensions of the MC68000 architecture incorporate the IEEE Proposed Floating-point Standard into the existing MC68000 architecture.

The creation of a computer architecture is a synthetic process. The various components of the architecture, such as registers and operations, must be combined into a unified whole. A properly designed computer architecture should exhibit a certain esthetic appeal as well as providing functionality and economy: simplicity of conception and consistency of features can do much to ease the use of a computer. Such is the case for floating-point extensions to the MC68000. On the one hand the design was required to be in conformance with the IEEE Proposed Standard for Binary Floating-point Arithmetic [1]. At the same time we desired to maintain the integrity of the MC68000 architecture [2] across the new extensions. Many persons have contributed to the design and implementation of floating-point for the MC68000. Throughout this paper, the pronoun "we" will be used to refer to the MC68000 design group in whole or part.

1. IEEE Proposed Floating-Point Standard

The IEEE Proposed Standard is independent of any particular implementation strategy. It describes three formats of floating-point numbers: single precision, double precision, and double-extended precision. A rather extensive set of required operations on floating-point numbers is specified, along with criteria for rounding and detection of exceptional conditions. Several modes of calculation, some available at the implementor's option, may be chosen by the floating-point user. An implementation of IEEE floating-point may optionally provide traps to user-supplied software in the case of errors or exceptional conditions.

Early in the design process, Motorola committed to support the IEEE Proposed Standard. Consistent with the MC68000 philosophy of providing the highest-possible state-of-the-art computing power, we chose to implement the entirety of the Proposed Standard.

2. MC68000 Architecture

The MC68000 architecture provided the major constraints to the design of the floating-point architecture. We decided to provide floating-point capability as a direct extension of the MC68000 architecture. This decision was made to maximize the overall system performance and to take advantage of expected advancements in semiconductor technology.

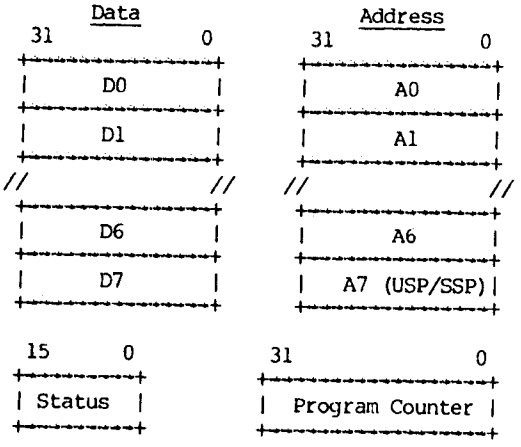

Figure 1. MC68000 registers.

The ingredients offered by the existing MC68000 architecture are shown in Figure 1. There are two sets of eight registers, each set offering slightly different functionality. The program counter needs no comment, but the status register combines functions of both control and status, allowing the masking of lower priority

interrupts as well as reporting the results of arithmetic comparisons. Examples of MC68000 instructions show that it is fundamentally a one-and-a-half address architecture; each instruction specifies a register as well as a full effective address, which may address a register as well as any memory location. One major exception, which will prove significant later, is the two-address MOVE instruction. It allows data in any of the three formats to be moved between any two effective addresses, i.e., between any register and memory, between any two registers, or between any two memory locations. The MC68000 provides a powerful interrupt structure which will play several roles in the final architecture and its implementation.

3. The Floating-point Architecture

The implementation details not addressed by the IEEE Proposed Standard were developed out of the MC68000 architectural concepts.

3.1 Registers

We decided to provide floating-point for the MC68000 as an additional set of floating-point registers which can contain only double-extended precision numbers, as shown in Figure 2. There are up to eight floating-point registers with a set of instructions that operate on them, analogous to the data and address register sets. A separate control register and status register are provided for floating-point operations.

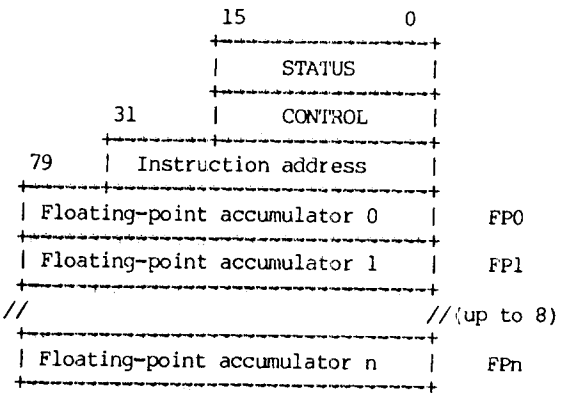

Figure 2. MC68000 floating-point architecture.

One rejected choice would have been to make floating-point accumulators out of one or more combined data registers; the choice of separate floating-point registers offers several significant advantages. The single internal format simplifies design of the floating-point algorithms. A hardware floating-point arithmetic unit may be separate from the original MC68000 ALU and may even run concurrently with it. Various partitionings of the MC68000 central processor are now possible. In particular, the floating-point operations may be implemented on a separate piece of silicon and yet be more tightly coupled to the CPU than is the usual peripheral component. Such a tightly-coupled processor is known as a co-processor. I will examine the co-processor implementation strategies in greater detail later.

3.2 Arithmetic instructions

The floating-point arithmetic operations fall easily into the model of MC68000 instructions, as shown by Figure 3. Here floating-point instructions are compared with analogous MC68000 instructions. Monadic operations use a single floating-point register as both source and destination. They correspond to MC68000 instructions such as EXT and CLR. The dyadic floating-point arithmetic instructions take the same form as the binary arithmetic instructions on the MC68000. In both cases one data register acts as one operand and as the result destination.

MC68000		Floating-point	
	Monadic		
CLR	D0	FSQRT	FP0
EXT	D3	FNEG	FP3
	Dyadic		
ADD	D0,D1	FADD	FP0,FP1
ADD.B	#120,D2	FADD.B	#120,FP2
MUL.L	CAT(A5),D4	FMUL.L	DOG(A5),FP4
SUB.W	D3,D4	SUB.W	D3,FP4

Figure 3. Arithmetic instructions for the original MC68000 and for the floating-point extensions.

For floating-point instructions, the effective address field has been expanded to include the ability to address another floating-point register as the second operand, as well as using any of the existing MC68000 addressing modes. We see that each operation has a format modifier attached to it. The three binary formats in the original MC68000, byte, word, and long, were extended to include the three floating-point formats mentioned earlier. However, the data typing associated with the floating-point instructions is interpreted slightly differently from the data typing in original MC68000 instructions, as will be seen as we examine the floating-point analogue of the MOVE instruction.

3.3 FMOVE instruction and data formats

Figure 4 shows examples of the FMOVE instruction. In the first example, data is

moved between two floating-point registers. Since all numbers in floating-point registers are in double-extended (internal) format, there is no format conversion on this move and no data type is specified.

```
FMOVE     FP0,FP1          FP register -->
FMOVE     FP3,FP2            FP register

FMOVE.B   CAT(A0),FP0      Memory integer ->
FMOVE.L   D0,FP1             FP register
FMOVE.W   (A4)+,FP3

FMOVE.S   D2,FP3           Memory FP -->
FMOVE.D   -(A7),FP4          FP register
FMOVE.X   BIGNUM(A0),FP2

FMOVE.P   (A1),FP0      Memory decimal-->
                           FP register
```

Figure 4. FMOVE's into floating-point registers.

The second set of examples show data moved from MC68000 memory or MC68000 registers into a floating-point register. Here a data type is specified for the instruction and is required. Unlike the MC68000 MOVE instruction, the FMOVE performs an implicit conversion to or from the internal floating-point register format. This difference in interpretation between MC68000 instructions and the analogous floating-point instructions is annoying at first, but as it develops, the difference leads to an extremely powerful floating-point instruction set.

For the formats of "B", "W", or "L", a binary byte, word, or long word is converted to internal floating-point format. These formats are native to the MC68000 and so any memory location or MC68000 register may be the source of the data. The next three formats, "S", "D", and "X", are the three IEEE specified floating-point formats: single precision, double precision, and double-extended precision. The double-extended precision format contains exactly the same range and precision as the internal floating-point registers. Note that a single precision floating-point number, being the same length as a long word (32 bits), may be stored in any MC68000 register that can accept LONG format data. The double and double-extended precision numbers can only be stored in memory, being 64 bits and 80 bits long, respectively. There is no provision in the MC68000 architecture for combining registers to hold results longer than 32 bits, and the floating-point extensions maintain that philosophy.

The final example shows an FMOVE of a data type "P" into a floating-point register. The "P" format is that of a BCD floating-point number consisting of a signed mantissa and a signed exponent to the base 10. This FMOVE instruction performs a decimal-to-binary floating-point conversion! Obviously, in this case there is a lot of processing involved and the result may contain rounding errors, but the utility of this instruction is truly astounding. The complex operation of conversion between binary and decimal representations has become, to the programmer, a simple move between two different formats for representing the same numeric value.

Figure 5 shows some examples of FMOVE instructions that move data out of floating-point registers into memory or MC68000 registers. Once again, an implicit conversion is performed from the internal format to the destination format, as specified by the data type on the FMOVE instruction.

```
FMOVE.L   FP0,D0         FP register -->
FMOVE.W   FP1,-(A5)        Memory integer
FMOVE.B   FP2,CAT(A1)

FMOVE.S   FP0,D0         FP register -->
FMOVE.D   FP0,(A5)         Memory FP
FMOVE.X   FP0,-(A7)

 FP register --> Decimal with formatting

FMOVE.P   #5,FP0,DECNUM(A1) Static format
FMOVE.P   D1,FP0,DECNUM(A1) Dynamic format
```

Figure 5. FMOVE's out of floating-point registers.

FMOVE's out exhibit more complications than FMOVE's in to floating-point registers. Only in the case of a move to an "X" format is it guaranteed that the result can be exactly represented in the destination format. Two error conditions are possible: the number is too large in magnitude for the destination, or some bits of precision will be lost in the destination format. In the first case, an OVERFLOW or UNDERFLOW, error will be signalled. In the second case, an INEXACT error will be signalled to indicate that rounding occurred during format conversion. The final example of FMOVE.P shows a conversion to BCD representation of the number. This instruction is complicated by the IEEE requirement that a format specification be included for decimal output. The MC68000 architecture does not contain a three-parameter instruction, but an analogy was found with the MC68000 "shift" instructions. The decimal formatting parameter, which I will call "K", appears in the FMOVE instruction as the first argument, much like the shift count of the MC68000 LSL instruction. Like the shift count, "K" may be either static or dynamic, i.e. it may

be included as immediate data in the instruction, or it may be specified by the contents of a MC68000 data register.

3.4 Compare and branch

Properly speaking, the FCMP instruction is a dyadic arithmetic operation with two source operands that remain unchanged. The result of an FCMP is to set the floating condition code bits to reflect the numeric relationship between the two operands.

Another class of instructions allow numbers to be tested for the special symbols provided by the IEEE Proposed Standard. Figure 6 shows these "ISXX" instructions. They test for values of +/-infinity, +/-zero, or Not-a-Number, by setting the "equal" flag in the floating-point condition codes if the number is the same as the special value tested for.

```
FCMP.L    D0,FP0          Compare with integer
FCMP      FP0,FP1         Compare with FP reg.

ISNAN     FP1             Test for special
ISZERO.D  NUMBER(A2)        values
ISINF.S   D1

FBEQ      ARE_EQUAL       Branch on
FBUN      ARE_UNORDERED     condition

FSLT      -(A7)           Set byte on
FSORD     ORDERED_FLAG      condition
```

Figure 6. Instructions involving the floating-point condition code bits.

Two classes of instructions make use of the floating-point condition codes. The "FBcc" instructions branch to their effective address if the condition specified in the instruction is true, otherwise, execution continues with the next sequential instruction. This class is entirely analogous with the MC68000 "Bcc" instructions, the conditions tested have been expanded to include the "unordered" condition required by the IEEE Proposed Standard.

The class of "FScc" instructions perform analogously to the MC68000 "Scc" instructions. A byte of all one's is stored at the effective address if the specified condition is true, else a byte of all zeroes is stored.

3.5 Exception handling

Several times I have mentioned errors or exceptions that may arise during the course of a floating-point operation. The IEEE Proposed Standard requires a set of exception flags that are set whenever the associated exception occurs. These flags can only be reset by the user program and are therefore called "sticky" flags. These are provided in the 16-bit Status register in the floating-point processor. The Proposed Standard also requires that the user be able to set bits to mask any or all of the exception traps. These bits are provided in the Control register of the floating-point processor. If an exception is masked, then a default result specified by the Standard is returned to the destination and execution of the user program continues. If an exception occurs and is not masked, then a trap must be taken to user-supplied software to handle the exception. These traps are easily incorporated into the M68000 architecture. One or more of the MC68000's "unassigned, reserved" interrupt vectors will be assigned to floating-point exceptions.

4.0 Implementation

The MC68000 was designed to allow easy extension of its instruction set by providing two classes, or "lines", of unimplemented instructions. These are called the A-line and F-line emulator instructions. On the current MC68000 CPU there are two trap vectors defined for each of these emulator lines. An instruction with a hexadecimal "A" or "F" in its 4 most-significant bits will cause the appropriate supervisor trap to be taken. The door is immediately open to implement the MC68000 floating-point extensions by way of software emulation. This software project is, in fact, under way. The F-line emulator trap has been reserved for floating-point and other extensions to the MC68000 instruction set.

A software emulation may provide a suitable cost-benefit ratio for many applications, but the real excitement these days centers on the promise of high-speed floating-point arithmetic on a chip. Conceptually, a co-processor is an extension to an existing microprocessor architecture which is implemented on a separate silicon chip. With advances in semiconductor technology, a co-processor could be incorporated onto the same chip as the original CPU. I have already shown how the original instruction set must be extended systematically and rationally without doing violence to the "philosophy" of the original architecture. Care must be taken with implementation if overall system performance is to be maximized. First and foremost, the co-processor's arithmetic unit must be optimized for floating-point arithmetic. The speed of execution of a floating-point instruction will set the limit on system throughput.

Equal care must be given to the interface to the CPU and the system bus. In particular, the bus interface functions must be partitioned between the CPU and the co-processor so as to provide the highest overall performance while minimizing the duplication of expensive circuitry. It is a great advantage if the co-processor can execute concurrently with the CPU.

Once the co-processor has begun concurrent execution of its instruction, the CPU may continue executing non-co-processor instructions and may respond to interrupts. Of course, this approach raises the problems of CPU and co-processor synchronization.

A co-processor should use the same instructions as its software emulation. It is a commonplace of operating system design that protection mechanisms provide a boundary that hides the operating system from the user, giving it the appearance of hardware. These same mechanisms can be used with the floating-point emulation to give the user object-code compatibility between hardware and software floating-point. The advantages of compatibility need some emphasis: user programs need not be recompiled or relinked when moving from emulation to floating-point hardware. For many microprocessor applications, this means that a large investment in mask-programmed ROM's need not be thrown away. Additionally, a vendor can sell two versions of his product offering different cost/speed trade-offs. The software emulation will be slower, but relatively less expensive than the faster hardware.

5.0 Conclusion

The design of MC68000 floating-point started from two independent constraints: the IEEE Proposed Standard for Binary Floating-point Arithmetic and the existing MC68000 architecture. These architectural constraints were sucessfully merged to produce a uniform and useful unity. Two different paths of implementation for the floating-point extensions were identified: software emulation and co-processor hardware. Both paths are being actively explored, with the software emulation nearing fruition in the MC68341 Floating-Point ROM.

References

1. IEEE Task P754, "Proposed Standard for Binary Floating-Point Arithmetic," Computer, Vol. 14, No. 3, March, 1981, pp. 51-62.

2. Motorola, Inc., MC68000 16-Bit Microprocessor User's Manual.
Available from: Motorola Literature Distribution Center, Box 20924, Phoenix, AZ 85036.