VAX HARDWARE FOR THE PROPOSED IEEE FLOATING-POINT STANDARD*

George S. Taylor and David A. Patterson

Computer Science Division
University of California
Berkeley, California 94720

ABSTRACT

The proposed IEEE floating-point standard has been implemented in a substitute floating-point accelerator for the VAX** 11/780. We explain how features of the proposed standard influenced the design of the new processor. By comparing it with the original VAX accelerator, we illustrate the differences between hardware for the proposed standard and hardware for a more traditional floating-point architecture.

INTRODUCTION

We have designed an MSI processor to implement the proposed IEEE binary floating-point arithmetic standard [1]. The architecture specified by the standard is called *KCS*, after its initial proponents W. Kahan, J. Coonen and H. Stone. Several companies are implementing the proposal in special purpose microprocessors [2][3], but the implications of KCS for the high-speed combinatorial processors in larger computers are not well understood. This paper summarizes the cost of supporting the major features of KCS in a substitute floating-point accelerator(FPA) for the VAX 11/780 minicomputer. We believe that future implementations of KCS will need to make similar modifications in the functional units we discuss below. As a specific example of the relation between a KCS machine and a more traditional floating-point processor, we compare the Berkeley FPA to the DEC FPA. This comparison is interesting because traditional floating-point designs have been driven by considerations of implementation cost rather than the cost of writing programs to use them.

KCS attempts to simplify the task of writing high quality numerical software by increasing the demands made on system designers. We asked the following questions:

- How does KCS alter the requirements of functional units (e.g., rounder, shifter) usually found in floating-point processors?
- What new functions must be performed?
- Can appropriate hardware and microcode perform KCS operations without time penalty in the absence of arithmetic exceptions?
- How much additional microcode is required?

* This work was supported by the U. S. Department of Energy under contract DE-AT03-76SF00034, project agreement DE-AS03-79ER10358, and by the National Science Foundation under grant MCS78-07291. The first author was supported by an NSF Graduate Fellowship.

** VAX is a trademark of Digital Equipment Corporation.

The new design evolved from the DEC FPA. We accepted the constraints of a similar quantity of hardware and the same technology. The DEC FPA consists of 700 Schottky ICs on five 15" x 12" boards connected by a backplane with 180 signal lines. Our system's functional units also are distributed across five boards and interconnected by a single data bus. We should mention, however, that our goal was to build a tool for numerical software research rather than a plug compatible replacement. The processor supports all optional features of the proposed standard and allows us to experiment with mixed precision operands in a single instruction.

Although the debugging of our hardware prototype is not finished, we can draw conclusions from the completed design. We found that for a given level of performance, the hardware necessary to support KCS is slightly more intricate and slightly larger than a traditional floating point architecture.

UNUSUAL FEATURES OF KCS

KCS differs from prior floating-point architectures in several ways. Those which have the most impact on our processor include:

1) *Multiple exponent sizes* – KCS recommends three floating-point data types: single, double and (double) extended (see Figure 1). The range as well as the precision differs for each data type.
2) *Elaborate exception handling* – KCS defines five exceptions with corresponding flags and trap masks. The user may supply a trap handler for each exception or take the default response at his option.
3) *Six mode bits* – The user may control rounding, normalization, and the interpretation of signed infinities using six mode bits.†
4) *Denormalized numbers* – Underflow occurs gradually in KCS rather than abruptly. The default response to exponent underflow is to shift the fraction right until the exponent increases to the minimum value for that precision. Unless it is zero, the result is called a *Denormalized* number.

† Two bits select the rounding mode: toward zero, to nearest even, toward plus infinity, and toward minus infinity. Two bits control whether extended precision fractions are rounded at the width of single or double precision rather their full width. The normalizing mode bit determines whether Denormalized operands are treated within arithmetic routines as if they first had been normalized. The final mode bit selects affine or projective closure for comparisons in which one or both operands are infinities.

5) *Extended precision* – KCS defines extended versions of the basic precisions single and double. An extended has a wider exponent and fraction than its basic precision counterpart, but a much narrower fraction than that of the next larger basic precision. We follow the recommended practice of supporting only the extended version of the widest basic precision.

6) *Square root and remainder* – KCS requires correctly rounded square root and a remainder operation (defined as x rem y = x - y*n where n = x/y rounded to nearest).

7) *Infinity and Not-a-Number symbols* – KCS has special symbols for infinities and mathematically undefined results (Not-a-Number or NaN).

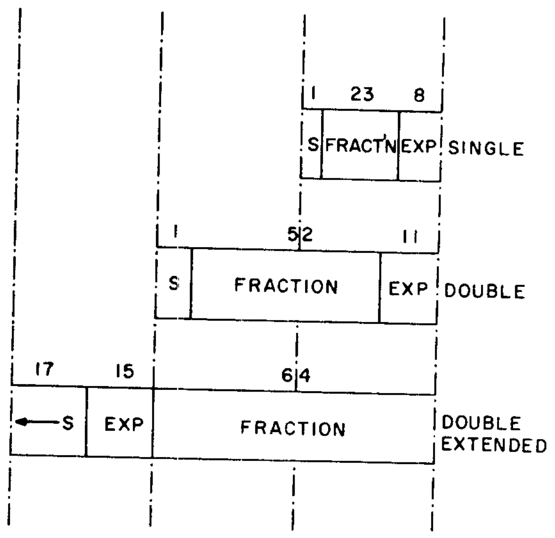

FIGURE 1 – KCS FLOATING POINT DATA FORMATS

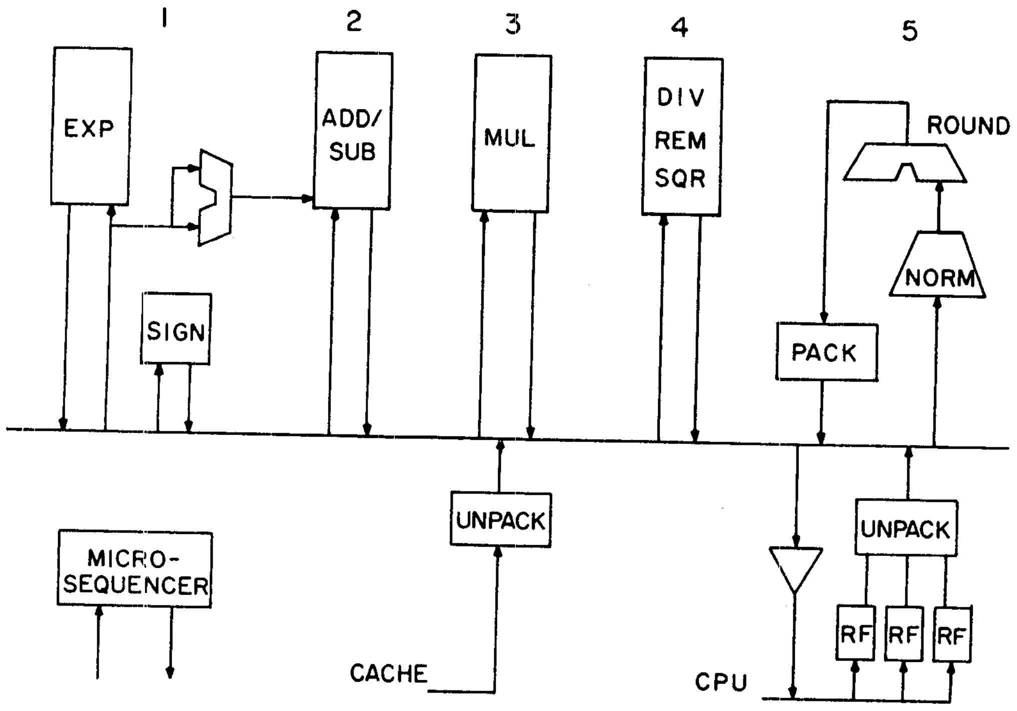

FIGURE 2 – KCS FPA BLOCK DIAGRAM

DESIGN OVERVIEW

A block diagram of the KCS FPA is shown in Figure 2. We use fourteen parts unlike those in the VAX: four LSI multipliers, nine programmable array logic (PAL) packages, and an LSI microcode sequencer. Table 1 compares the distribution of ICs in the KCS FPA with that of the DEC FPA. The six categories which differ are multiply, divide/remainder, square root, exponent, round, and pack/unpack. KCS is responsible for the increases in only the last four categories. Multiply is smaller in the Berkeley FPA due to the use of LSI mutipliers. We allocated an entire board to experiment with higher performance division. Square root merged inexpensively with this division hardware. The KCS exponent processor deals with 15-bit operands instead of 8-bit ones. KCS rounding, unpacking, and packing operations are more complicated than their DEC counterparts. The increase in unpacking logic is exaggerated because data enters the FPA through two ports. We duplicated the logic in order to maintain a similar system interface.

Table 1.
IC Count by function for KCS and DEC FPAs

	KCS	DEC
Add,Subtract	145	150
Multiply	**105**	**220**
Divide, Remainder	**135**	**45**
Square Root	**25**	**—**
Exponent, Sign	**75**	**50**
Round	**75**	**30**
Normalize	45	35
Control	85	80
Data Interface	70	65
Pack, Unpack	**110**	**25**
Total	870	700

Table 2.
Speed of operations for KCS and DEC FPAs

DEC times in parentheses

	SGL	DBL	EXT
ADD	1.0(0.8)	1.8(1.4)	2.2
MUL	1.4(1.2)	2.6(3.4)	3.0
DIV	2.6(4.2)	4.8(8.8)	5.8
SQR	4.0	7.4	9.0

Table 2 shows that the KCS FPA is comparable in speed to the DEC FPA. We use more ICs, but support three data types instead of two and achieve higher performance in the wider precisions. Extended precision results take 0.4 μsec longer to return to the main CPU than double precision ones because a 32-bit bus is used.

Single precision ADD would be the same speed if we handled overflow checking cleanly. We would be able to do so by changing the microcode in the central processor to which the FPA is attached.

As in the DEC FPA, our boards are connected by a data bus which transfers two single, one double, or one extended precision number at a time. The bus is 87 bits wide:

64 for the operand fraction(s);

2 for the sign bit and overflow bit in fractions of intermediate results;

3 for guard, round, and Sticky bits in fractions of intermediate results;

16 for the operand exponent(s);

2 sign bits.

The original bus was 68 bits wide.

Exponent Board

The EXP board, shown in Figure 3, calculates the sign and exponent of the result and contains the microprogram control unit. Since KCS exponents are biased by $2^n - 1$ rather than 2^n, they can be converted more easily to two's complement - 1 than to two's complement. Consequently the exponent unit treats its data as two's complement - 1.* This internal form simplifies conversion from one floating point data type to another.

Although the extended precision exponent is 15 bits wide, the exponent ALU must be 17. One extra bit is required for pre-normalization of Denormalized extended precision numbers and the second bit is required for multiplication and division.

The exponent difference logic controls the ADD board pre-alignment shifter in addition and subtraction. Denormalization of a product or quotient and conversion of a floating-point number to an integer, however, require calculation of a shift distance. Thus the exponent ALU can also control the shifter.

The microsequencer is built around an Am2910. Unlike traditional floating-point units such as the VAX, the KCS one requires stored constants to perform several operations.** Thus our 64-bit microinstruction includes a 16-bit constant field not found in the 48-bit VAX FPA microinstruction. The 2910 includes a microprogram counter which allows us to share the next address field with the constant field. Data path control accounts for 39 of 48 bits in the VAX FPA and 48 of 64 in the KCS FPA.

* A two's complement - 1 ALU performs A+B+1 and A-B-1 instead of A+B and A-B.

** 16-bit constants are used in the exponent unit for conversion between floating-point precisions, rounding a floating point number to a integer value, and scaling after overflow or underflow before going to a trap handler.

Add Board

Figure 4 is a block diagram of the ADD board. Its primary difference from traditional units is that two guard bits and the so-called Sticky bit are formed in the barrel shifter. More later about the use of these bits in the rounding logic. Only the Sticky bit adds significant cost. The basic shifter consists of 48 ICs in three stages. The guard bits require four additional ICs. The Sticky bit logic uses 21, including two PALs. We were able to save eight ICs by using bits from the intermediate stages of the main barrel shifter.

Multiply Board

The multiply board is shown in Figure 5. KCS' only requirement for this board is a 17-bit OR gate to form the Sticky bit, but we include a description for completeness. We use four 16x16 multipliers organized as a two-stage pipeline that processes 16 bits per 200 ns cycle.* The first stage of the pipeline performs a 64x16 multiply yielding two 64-bit partial products. The second stage adds these products into the partial result accumulated in previous cycles. The first stage is entirely within the LSI multipliers because they contain input and output registers. Three cycles are required for a 64x32 multiplication in single precision floating-point or 32-bit integer arithmetic. In double and extended floating-point, a 64x64 multiply is completed in five cycles.

The interface to the VAX cache is on the bottom of this board, so one of the duplicate sets of unpacking logic is included.

*The multiplier configuration is 64x16 instead of 32x32 because this requires fewer carry save adders in the pipeline and eliminates conflicts on shared input and output pins.

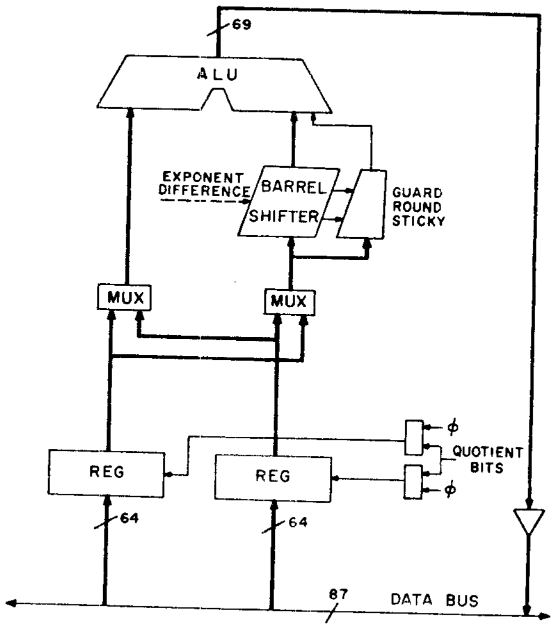

FIGURE 4-ADD BOARD

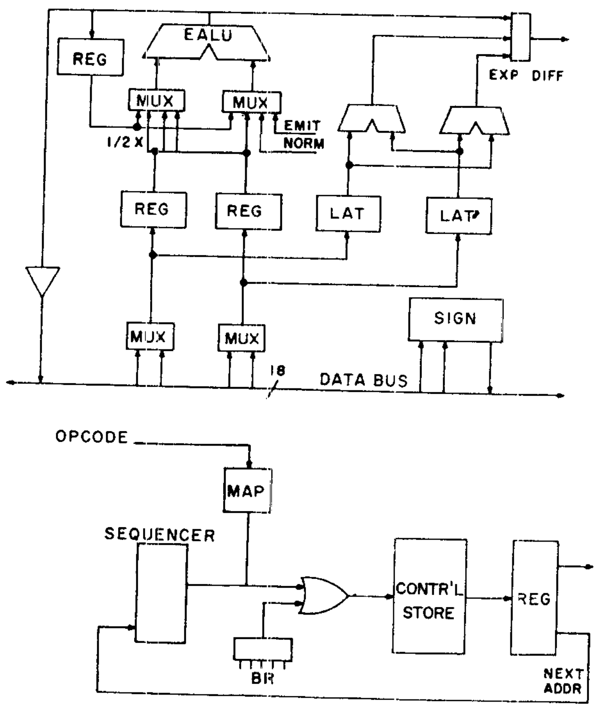

FIGURE 3-EXPONENT BOARD

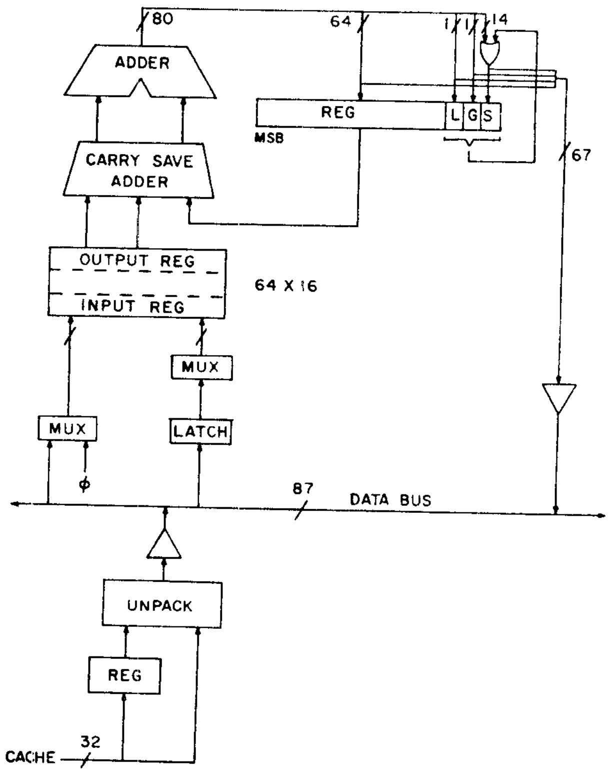

FIGURE 5 - MULTIPLY BOARD

Divide Board

The DIV board supports division, remainder, and square root. Figure 6 shows the block diagram and data paths. The division algorithm is radix four nonrestoring, while the square root algorithm is radix two restoring. The inner loop of both operations is clocked at 100 ns intervals. Details of this board are given in a companion paper [4].

Normalization Board

The NRM board, shown in Figure 7, is used for normalization and rounding. Its design is similar to the DEC FPA counterpart, but the rounding, unpacking and packing logic is more complex. The rounding logic is programmed into two PALs, while the data unpacking/packing logic requires 65 multiplexers and gates.

The normalization shifter is used for pre-normalization of Denormalized operands as well as its regular function of normalizing the result of an operation. Special control logic limits normalization after multiplication and division to one bit.

The extended precision data type suggests the ability to read 80 bits in parallel from the register set. Thus 2½ copies of the VAX register set (32 bits each) are held on this board.

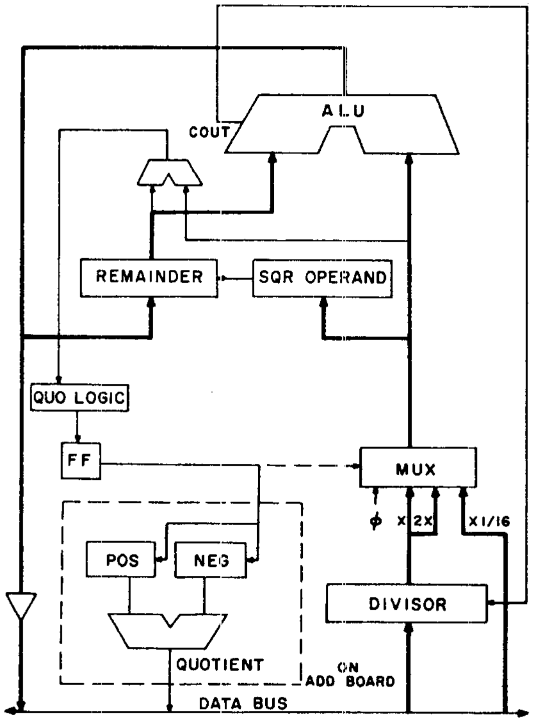

FIGURE 6 – DIVISION BOARD

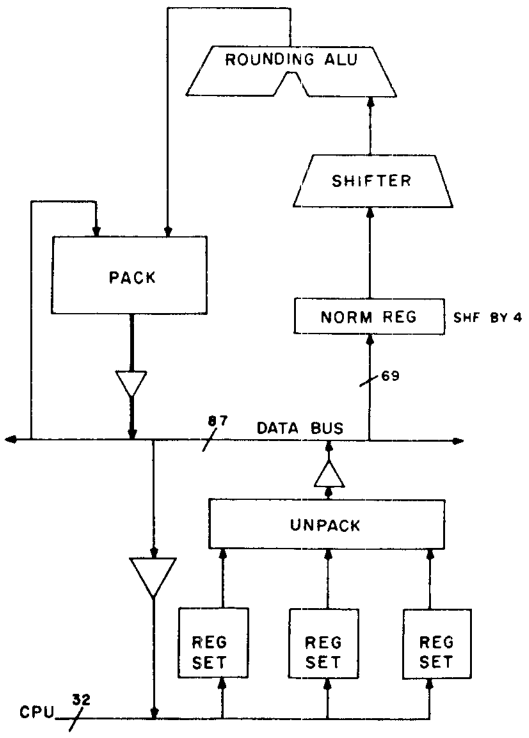

FIGURE 7 – NORMALIZATION BOARD

IMPACT OF UNUSUAL FEATURES

This section describes the impact which the seven KCS features mentioned above have on our design.

Multiple exponent sizes have the largest impact. Much of the cost we pay in packing and unpacking logic would be necessary in any system with multiple exponent sizes. The cost is doubled in our case by the FPA's two port interface. We also provide the hardware to execute instructions with operands of different precisions. The main support is left justifying the fractions of all operands to give proper alignment in the arithmetic units.

The alternative is to adjust operand fractions to the right. If fractions are right aligned as they enter the arithmetic units, only those of the same precision will have correctly aligned binary points. The right aligned system has a drawback even if mixed precision operands are not allowed: three sets of logic are required to find the normalization distance after cancellation in subtraction. The normalization logic is already duplicated to find either the first one or the first zero because the sign of the intermediate fraction is unpredictable if the exponents are equal. A total of six sets of normalization logic would be required, two for each of the three precisions. Nevertheless, right alignment could substantially reduce the amount of unpacking logic.

KCS recommends that the order of components in a floating-point number be sign, exponent, fraction (SEF) from left to right. This would have worked against our intention to align fractions left and exponents right, so we store numbers as SFE, with the fraction before the exponent. The packing and unpacking logic is reduced by 40 ICs. Although the SFE format is not conforming for data interchange, it causes no complications within the machine since floating-point compare instructions remove the need to take advantage of the lexicographic ordering of numbers by value in SEF.

Exception checking and handling has its primary impact on microcode. We allocated twice as much microcode space (1024 words) as the DEC FPA, but have not yet completed microcoding of special cases for all floating-point instructions.

Three of the five exceptions defined by KCS are dealt with in most systems: underflow, overflow, and division by zero. The inexact result exception is detected with a small piece of logic in the rounding unit. The invalid operation exception covers all other cases.

KCS allows users to write their own exception handling programs. The hardware is thus designed so that the original operands of an instruction can be recovered by a user's trap handler. This involves either preventing an overwrite when an exception is detected or saving the operands in hardware so that they can be passed to the trap handler. We chose to prevent overwrites. KCS does not require that exceptions be handled by hardware. Only the inexact exception occurs frequently enough (with the trap masked off) to merit a hardwired default response. The other exceptions permit a slow response.

KCS rounding modes require new hardware. The three modes other than round toward zero rely on the Sticky bit, which is equal to zero only when the result is exact, i.e., just when the infinite precision result would contain only zeros beyond the guard bit. Means to form the Sticky bit on each board were discussed above. The total hardware cost is 45 ICs (5%).*

Normalizing mode also has a significant impact on processor design. It determines whether Denormalized operands are treated as if they first had been normalized. As mentioned above, this costs an extra bit of exponent processor width. It also requires the capability to update each of the exponent operand registers.

Normalizing mode can be ignored in addition/subtraction, but it requires actual prenormalization of the operands in multiplication and division. We use the regular normalization unit for this purpose. Since the system has a common data bus, only one operand can be pre-processed at a time. There is no new cost in the fraction hardware.

Denormalized numbers must be created by a KCS machine whether or not the Normalizing mode is in effect. Denormalization amounts to shifting right the fraction of a preliminary result. We use the alignment shifter on the ADD board for this purpose. The exponent unit is made slightly more powerful so that it can calculate the shift distance.

Denormalized numbers are relatively simple to deal with in single and double precision. They account for a small part of the unpacking logic. After unpacking, they are usually treated as ordinary numbers. Exceptions are that they cannot be divisors or arguments to the square root instruction, and single Denormalized numbers cannot be converted to double. So that single and double Denormalized numbers can be converted to extended, its fraction has an explicit leading bit. This implies that unnormalized numbers can exist throughout the range. These unnormalized numbers are responsible for many of the non-standard cases that must be handled by microcode.

Extended precision serves to backup double precision in a chain of computations by allowing intermediate values to have greater range and precision. The KCS FPA has 12% more ICs with extended than it would have if just single and double precision were supported. As Table 2 demonstrates, extended precision operations take only a little longer than their double precision counterparts. One reason is that the extended fraction stops at the next natural boundary after the double fraction; in our case 64-bits. 64 bit ALUs and multiplier slices are as fast as 53 bit ones would be.

80 bits is not a natural size for the 32-bit registers and busses in the VAX architecture. We allocate three registers or storage words to hold an extended number. The potentially greater memory access time is not a problem if the number of extendeds is small enough for them to be held in registers. Extended precision adds a burden to the microcode primarily because its explicit leading fraction bit introduces unnormalized numbers.

* The Sticky bit has one use besides rounding. It simplifies subtraction because it catches nonzero bits shifted out the subtrahend. There is no need to take an extra cycle to complement the subtrahend before shifting it.

KCS square root and remainder operations are supported with a 4% marginal increment to the hardware. Correctly rounded square root could be implemented in software, but we support it in hardware using 25 additional ICs on the division board. The remainder operation is exactly like division except for initialization and fix-up steps. These require about 10 additional microinstructions. Remainder is defined so that its potential for long latency (while generating and discarding to 32,000 leading quotient bits) is handled with a software loop. The microcode stops the instruction every 64 bits by overwriting the dividend with the current remainder and restoring the program counter. Thus interrupts can occur normally.

The **symbols for infinity and Not-a-Number** have no hardware impact other than detection of an exponent value equal to all ones. The microcode handles infinity and NaN operands as special cases in each instruction. The normal fraction and exponent hardware is not used for these symbols.

CONCLUSIONS

None of unusual features of the KCS dominated the cost or complexity of the Berkeley FPA. The greatest problem we encountered was in unpacking operands to a precision-independent internal form. High speed mixed precision operations are expensive with the KCS formats.

Rounding logic increased from 4% in the DEC FPA to 8% in ours, primarily due to the Sticky bit. Gradual underflow and Normalizing mode cost little because the single bus architecture allows us to reuse shifters. Extended precision adds only 12% to the hardware but executes at the the speed of double precision. If extended is an adequate backup for double, it will be far more cost effective than quadruple precision.

We find that KCS floating-point is feasible for mainframes and minicomputers, that it is about the same speed for about the same hardware, but that it has inherently greater design time due to the greater complexity.

STATUS

The processor boards have been fabricated and populated with chips. They are now being debugged. We have written the basic microcode sequences and simulated them using the ISPS system [5].

ACKNOWLEDGEMENTS

Several people have worked on this design and several others have supported this research. Eric Shienbrood developed software tools needed to complete the design, wrote some of the microcode, and performed the ISPS simulation. Marianne Picha developed the microcode assembler DAPL [6]. David Cary helped with the fabrication and (is still) debugging the hardware. Terry Allison of Lawrence Livermore produced wire lists from signal lists. Electronic Manufacturing Service wire-wrapped the boards.

Support from commercial firms has been crucial to this project. We thank Sam Fuller and Dick Eckhouse of Digital Equipment Corporation for a grant and the donation of cabinets and boards. Shlomo Waser of Monolithic Memories donated ALUs and PALs, and provided the services of his company to program the latter. William Koral of TRW supplied the LSI multipliers. Gene Hill of Intel provided a microprocessor development system for use in the standalone prototype. AMD and National Semiconductor supplied several critical ICs. We thank each individual and company for their generosity.

We would like to thank Professor Richard Fateman for his continuing support of the project.

The final thanks goes to Professor William Kahan whose persistence pushed floating-point arithmetic and this project further than most of us planned.

REFERENCES

[1] IEEE Computer Society Microprocessor Standards Committee Task P754, "A Proposed Standard for Binary Floating Point Arithmetic," *Computer* 14, No. 3, March, 1981, pp. 51-62.

[2] Intel, "The 8086 Family User's Manual Numerics Supplement", July, 1980.

[3] Motorola, "MC68A39 Advance Information", 1981.

[4] G. Taylor, "Compatible Hardware for Division and Square Root," Fifth IEEE Symposium on Computer Arithmetic, May, 1981.

[5] M. Barbacci, "Instruction Set Processor Specifications (ISPS): The Notation and Its Application," *IEEE Transactions on Computers* 30, No. 1, January, 1981, pp. 24-40.

[6] M. Picha, "DAPL: A Microprogramming Assembler," M.S. Project Report, University of California, Berkeley, September, 1980.