A CHIP-SET FOR A HIGH-SPEED LOW-COST FLOATING-POINT UNIT.

J.B.Gosling B.Sc., Ph.D., C.Eng., M.I.E.E., J.H.P.Zurawski M.Sc., Ph.D.,
Prof. D.B.G.Edwards M.Sc., Ph.D., C.Eng., M.I.E.E.

Department of Computer Science
University of Manchester, Manchester, England. M13 9PL

Abstract

Although the advent of microprocessors has put considerable computing power in the hands of large numbers of users, there is still an important group who have yet to benefit fully from large scale integration. As a step in the direction of rectifying this situation, a highly flexible chip set is being designed, with a view to reducing the cost of a powerful floating point processor by a factor of about 4. Processing speed will be up to twice that of an equivalent unit built from MSI devices, before allowance is made for savings on wiring delays. It will be possible to construct a unit satisfying all published standards, proposed and existing (de facto), as well as permitting a number of extensions not specifically in these standards. At a cost between 100 and 150 ICs, and with a floating-point add time of around 120nS, the proposed unit is cost-effective compared to currently available coprocessors.

1. Introduction

Large scale integration of digital circuitry has produced 16-bit microprocessor systems, together with relatively large memories with up to eight bits in parallel. The computers so produced are of too low a throughput for a large group of users, notably the scientific community. To cope with more powerful processing facilities, and the longer word lengths used, it is necessary to partition the logic in such a way that the word length is extendable. This leads to the bit-slice approach. The resulting machines are still very slow in terms of conventional large computers. To achieve greater processing power, more specialised designs are needed, and for the speeds required, a lower scale of integration has to be accepted. This leads to more specialised packages and hence a smaller market, which in turn points to the use of uncommitted logic arrays (ULAs) or masterslices. This work describes an approach to the design of a high speed floating-point arithmetic unit, making use of a ULA in emitter coupled logic (ECL).

Experience of designing with the particular array has been obtained in an earlier project in which a slice of a high speed multiplier was produced [1]. This was proved on a 16-bit model, which indicated a 64-bit multiply time of around 280nS. Two additional, but smaller and faster ULAs were designed to handle end effects and control, replacing 17 SSI circuits. The total cost of this multiplier would be 56 packages. This compares with about 800 for a machine of 10 years ago, at a similar theoretical speed. The actual speed of the latter was 60% slower due to large wiring delays that will not be present in the newer unit. Current work is directed towards the 56-bit mantissa section of a 64-bit floating-point multiplier (112-bit product).

The authors have also been associated with the design of two floating-point arithmetic units. That of 1968 [2] consisted of about 3000 SSI circuits, and had a repertoire of about 50 functions including 32-bit fixed-point and 64-bit floating point arithmetic. Floating point addition time was an average of 250nS, and a multiply time of 500nS. A newer unit at present under construction in a machine known as MU6G [3], consists of about 650 ICs, mostly MSI. The unit will perform only floating-point arithmetic, but includes a capability for calculating a number of mathematical functions by polynomial evaluation under microprogram control. Floating-point add time is expected to be under 200nS. Multiplication speed is low, at 1.8μS, but the fast multiplier discussed above is to be connected as an 'optional extra'.

The present work is aimed at producing a set of medium scale integrated components that can significantly reduce the size of this unit. It is expected that the MU6G floating-point unit will be reduced to around 180 ICs, and a simple unit without the mathematical functions should require less than 120. The speed will be at least as good as MU6G, and initial indications are that the operation times will be halved.

The ULA to be used is the Plessey ULA2000 or ULA3000. They are similar to the Fairchild F100168. These devices each consist of 144 cells that are essentially 2-input OR-NOR gates (similar to those of [1]), and with capability for both wired-AND and wired-OR. The difference between the two arrays is solely one of speed and power dissipation. The ULA2000 has an OR-NOR time of 550pS, and dissipates up to 4W, while for the ULA3000 the figures are 3nS and 1W. There are a maximum of 64 connections available, including power. Of these, up to 24 may be outputs. These arrays are small by the standards of other

CH1630-3/81/0000/0050$00.75 © 1981 IEEE

technologies, but are currently available. Larger arrays in ECL are projected for the next few years. The proposed design is such that it will be a simple matter to incorporate more onto the larger chips, since pin limitation is not a serious problem.

2. Specification of a floating point unit

The design of the chip set is to be as flexible as possible in order to allow the system designer the maximum freedom in choosing the details of the system. It is hoped that the result will then be a commercial proposition. The cost of producing different designs for each system variation is still prohibitive, even using ULAs.

Considerations that are to be born in mind include the following

a) The chip set should be capable of meeting the de facto standard of ICL/IBM floating point units. These make use of an exponent base of 16 rather than 2. It is believed that this format may be more acceptable to a number of large users rather than one in which the exponent base is restricted to 2.

b) The proposals of the committee on floating-point standards must be considered [4], since many users will wish to conform with any agreed standard. The maximum set is the aim.

c) The reasons for the choice of sign and magnitude representation of numbers by the major manufacturers and by the IEEE standards committee is not clear. The set will be designed to allow the choice of a complement number system.

d) At least two methods of exponent biasing are used; by 2^n or $2^n - 1$. The logic will be designed to handle both.

e) Different approaches to rounding are used by different manufacturers, and others have been proposed. MU6G contains options to allow multilength arithmetic to be performed, and to satisfy different high level language requirements with regard to real to integer conversion (truncate, round, entier etc). One of the floating-point standard proposals suggests the use of rounding towards $+\infty$ and towards $-\infty$ to enable the user to assess the error bounds of an algorithm.

f) Exceptions must be detected and used appropriately. Exceptions include overflows and underflows, indications of which may be collected together in a register of "odd" status bits, collectively designated here as AOD. This register will also contain exception override bits and could include 1-bit indications of "Not a numbers"[4]. "Graceful underflow" as proposed in one version of the floating point standard may also require a flag since numbers are then permitted to become subnormal.

g) The function set must include the operations add, subtract, multiply and divide. The operations 'reverse subtract' and 'reverse divide', defined as S-A and S/A (as opposed to subtract as A-S and divide as A/S), are also required when using a one-address architecture. For this architecture operations such as 'load accumulator' and 'store accumulator' are also included. The proposed floating-point standard includes square root, remainder (=x - yxn; n = nearest integer to x/y), compare, and conversions between fixed and floating-point

Store ACC	ST
Store AEX	STX
Load ACC with real	LDR
Load AEX (with real)	LDX
Load ACC with integer	LDD
Load exponent only	LSR
Add ACC to OP	ADD
Subtract OP from ACC	SUB
Subtract ACC from OP	RSB
Multiply ACC by OP	MLT
Divide ACC by OP	DIV
Divide OP by ACC	RDV
Scale ACC by 2^n (LS bits of OPM treated as a signed integer)	SCL
Square root of ACC	SQT
Compare ACC and OP	CMP
Remainder (ACC=ACC-OP*n n = nearest integer to ACC/OP)	REM
Convert integer to real	CIR
real to integer	CRI
real to floating point integer	CRF

Table 1 Proposed operations of a floating point unit
ACC - accumulator } the two operands in 2 or more
OP - operand } address formats
AEX - an extension to ACC used mainly for double length work.

formats. MU6G includes facilities to assist with multilength arithmetic, and certain double length operations are provided in microprogram. A function SCALE allows rapid multiplication by 2^n (n integral) and the exponent can also be manipulated as an entity on its own (LSR).

Table 1 lists a function set defined in terms of a one-address architecture. For two or more address systems (two addresses referring to two operands etc.) the ST, LDR, LDD, RSB and RDV operations will be unnecessary. The register AEX is the accumulator extension register. Its most obvious use is to receive the less significant half of a multiplication result, thereby simplifying multilength multiplication. It is also useful in other double length operations for holding intermediate results.

Other desirable features are:

a) Results should be as close as possible to the true result. In particular, operations with integers of comparable magnitude but expressed in floating-point format should give the correct integer result.

b) The result of A+(-B) and A-(+B) should be identical.

c) It is helpful if error propagation in common sequences, such as polynomial evaluation of functions, can be minimised.

3. A Floating-Point unit proposal

The design of a floating-point unit falls into three sections, which are, to some extent, independant. These are the mantissa, the exponent, and the "end effects". It is tempting to leave the latter until a late stage in design, but this can lead to a severe deterioration in both cost and performance.

3.1 Mantissa

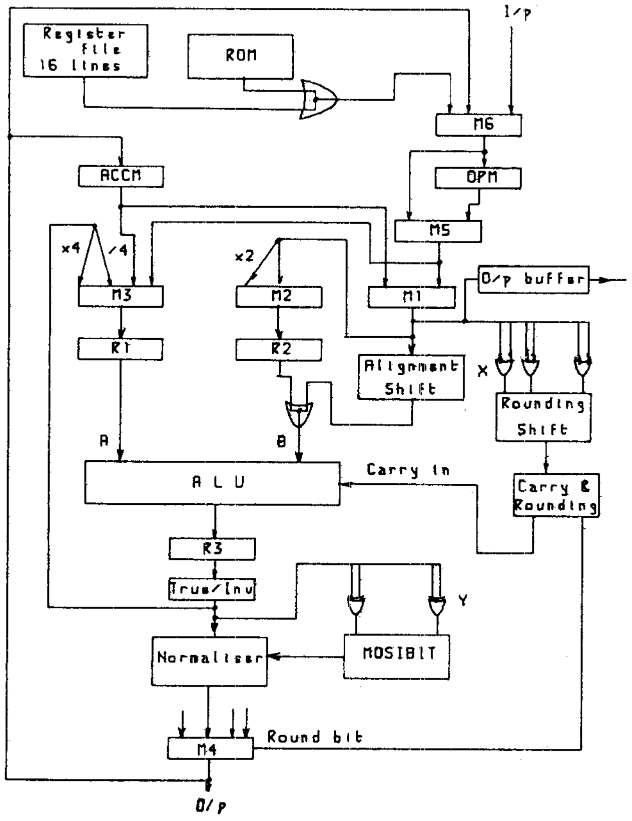

Fig. 1 Mantissa section

Fig. 1 is a schematic diagram of the mantissa section, and includes an indication of some of the end effect logic. It is arranged in a form suitable for a one-address system, but may be adapted easily for other forms of order code by addition of an extra multiplexer on the input to the register ACCM. For one-address orders, an existing number is held in a register ACC, whose mantissa section is ACCM. A second operand enters the unit via M6 and is set to register OPM. The use of this register as a buffer allows the operand fetch unit to prepare the next instruction - operand pair. For an addition, the exponents will be compared, and one operand is routed via M3 to R1, and the other to the alignment shifter. The output of these two devices feeds an adder subtractor unit at the inputs A and B. It is envisaged that this would consist of an arithmetic logic unit (ALU) such as the 10181. This unit can perform A+B and A-B. The latter is performed internally by inverting data on the B input and adding. With floating-point arithmetic the operand reaching the B input is the one with the smaller exponent, and not necessarily the subtrahend of a subtraction operation. If sign and magnitude representation is used, it is immaterial which operand is inverted, and the result may also require inversion. With twos complement the correct result can be obtained by use of the relation [5]

$$-A+B = \overline{A+\overline{B}}$$

which does not have a forced carry as is normally required. The result of the add is set to a register R3, true or inverse selected, and then normalised and returned to ACCM.

Multiplication takes place by using the loop consisting of the ALU, R3 (and the true / inverse logic), M3 and R1 to accumulate the product. ACCM is used as the multiplier, and is decoded two bits per cycle [1,5]. The decoder selects OPM or 2xOPM to be gated via M2 to R2. The least significant bits of the product fill up ACCM as the multiplier is shifted down. When the product is complete, the most significant half is in R3, and the least significant half in ACCM. ACCM is transferred to R1, R3 via the normalisation to ACCM, and R1 via the same normalisation to one line of the register file RF (if present).

In division the divisor is in OPM and dividend in ACCM. Again the loop consisting of the ALU, R3, M3 and R1 is used, with shifting in the opposite direction. A redundant 2-bit at a time algorithm is used [6].

Convert to integer orders make use of a special operand with a fixed exponent and zero mantissa. An appropriate addition without the subsequent normalisation will then give the correct result. Provision to hold the fractional part of the result in floating-point form in AEX can be made. Convert from integer to floating-point format operates similarly, but includes the normalisation.

Fig. 1 also includes an indication of the logic for rounding and carry generation to be described later, logic for controlling the normalising shifter, and for performing other functions. The latter consists of a read only memory to hold polynomial constants, and a register file to hold intermediate results mainly in multilength operations. One of these registers would be designated as AEX. The input route bypassing OPM to M5 is used by certain of these functions to avoid destroying the value in OPM. Finally a buffer from the output of M1 allows data to be fed to extra optional units, results returning via M4. A fast multiplier would be one such unit, where the low speed of the multiplier described above is insufficient for the application. For a 64-bit unit, a fast multiplier would add at least 50% to the chip count, and more to the PCB area. The multiplier speed would be up by a factor of about 6.

3.2 Partitioning of the mantissa logic

The unit illustrated in Fig. 1 is capable of providing all the features described in section 2. In an ideal world one would wish to incorporate a slice of all of this into a chip. Unfortunately ULAs of the required speed are not yet large enough to incorporate more than one bit of such a slice. The number of pins that would be needed for the two shifters, and the problem of adder carry propagation, particularly with the F100K Series, would make such a chip virtually impossible to design, and difficult to use.

In deciding what to omit from the logic of Fig. 1, two factors are considered. Firstly, certain relatively large scale circuits are available commercially. These include the ALU, the shifter (16-bit shift matrix, 10808), ROM and register file. Secondly, a less ambitious unit could be constructed without the ROM and register file. In this case M6 would not be needed. In some cases the output buffer and M4 would not be required either. Thus the ROM, RF, the output buffer, M6 and M4 will not be considered further.

Considering the shifter, the pin requirements of a shift matrix can be reduced by using a shift register approach. This will make shifting very slow, and add times, in particular, will be operand dependent. With exponent bases other than 2, the one-bit-at-a-time shift is particularly galling. This is one disadvantage of the 10800 when used in this application. The 10808 shift matrix is capable of shifting both ways, of shifting by the twos complement of the control input, and of backfilling with zeros or ones. This unit is therefore particularly suitable for the purpose, since the alignment shift is by the exponent difference or by its (effective) twos complement (a property of the bias system). The backfill arrangements allow for mantissae in either sign and magnitude or in complement form. Suitable wiring from M1 to the shifter, and from shifter to ALU, and of the normalise shifter, will allow the use of any exponent base without difficulty. Omitting the shifter from the ULA thus leads to a much more flexible system.

The second large section of logic is the adder-subtractor. A four bit slice of this will itself occupy most of a ULA of the type being considered. A reduced device performing only addition and subtraction requires about 50 cells. Consideration of block carries over a 50 plus bit mantissa, particularly when using the F100179, leads to the conclusion that a four bit (rather than a two bit) section is necessary. Three bits may be acceptable for this purpose, but has other disadvantages. It is therefore proposed that the adder-subtractor should be constructed from commercially available ALUs (10181, 10179 or equivalents).

The rounding, carry generation, and normalise distance (MOSIBIT) logic will be discussed in connection with the end effects. The remaining logic will be considered as candidate for inclusion in the mantissa chip.

Details of the circuits and their use will be found in [1]. Basic cells consist of 2-input OR-NOR gates, or 3 or 4-input ORs. A latch can be made from 2 cells, and a multiplexer-latch with one more cell than would be required by the multiplexer alone.

On this basis the mantissa section requires about 101 cells for a 3-bit slice, or 130 cells for a 4-bit slice. For ease of wiring it is advisable to aim for about 75% occupancy of the cells, or about 108 cells. The three bit slice fits into this, but the four bit one does not.

It appears intrinsically better to aim for a four-bit slice, as this fits better with commercially available devices. However, there is a more important reason. In constructing the rounding logic and the normalise distance (MOSIBIT of fig.1) the initial gating requires many SSI components. The section marked X in fig.1 consists of a set of 4-input OR gates, or 7 packages in ECL10K, for a 56-bit mantissa. That marked Y looks at each group of four bits and decides if they are all zeroes for positive numbers and negative if sign and magnitude numbers, or all ones for negative numbers in complement form. This requires over one package per four bit group. At the cost of four or five cells, this logic can be included in the mantissa slice, thus saving about 25 packages. For binary exponents the number of bits in the ULA would be immaterial, and some extra logic is needed to handle the odd few bits (3 bits for a 4 bit slice). For base 16 it is important to have 4-bit groups (or 8 etc.). Since this is a de facto standard, a four-bit slice is chosen. This can be achieved by omitting either ACCM or the OP-M5 combination. ACCM is left out, since the resulting structure is more appropriate to two and three-address order codes.

3.3. Exponent

Fig. 2 is a schematic diagram of the exponent section of the unit, and is suitable for 1, 2 or 3 address codes. ACCE and OPE are compared by the adder subtractor, and the difference, RD, is used to control the alignment shifter of the mantissa. The larger of ACCE and OPE is set to ACCE. After mantissa addition, the normalisation distance, MOSIBIT, is fed to OPE and subtracted from ACCE. Multiplication requires the addition of the exponents, and division and reverse division the subtraction or reverse subtraction of exponents respectively.

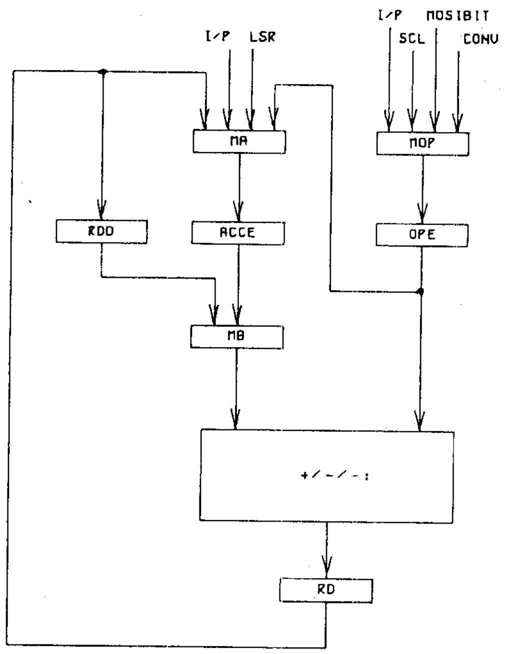

Fig. 2. Exponent section

The inputs to the exponent unit, other than the 'normal' ones are:

a) LSR. Used to load the exponent from an 'unusual' place in the operand. This is intended to allow the exponent to be manipulated as a separate entity. It is also needed with two and three-address order codes.
b) SCL. This is the entry for a part of the 'n' of a scale operation. Scale is performed by adding all or part of n to the exponent (depending on the exponent base) and shifting the mantissa under control of the rest of n.
c) CONV. This is the special exponent required in orders to convert between integer and floating-point formats. The value will depend on the exponent base, the mantissa length and whether the mantissa is an integer or a fraction.

On the assumption that a 4-bit adder subtractor will require about 50 cells, a 4-bit exponent slice will require about 120 cells. This is a little on the high side.

3.4. End Effects.

There are five major sections of logic which are placed under the general heading of end effects. These are:

'Carry' generation.
Rounding.
MOSIBIT (determination of normalisation distance).
AOD register.
Multiply and divide decoders.

By 'carry' is implied the carry to the least significant bit of the adder. The implementation of this affects the problem of getting the same answer to the two calculations A+(-B) and A-(+B) (twos complement). It also affects the rounding operations for convert orders, and is closely associated with the rounding of other functions.

Rounding can be done in three main ways.
By truncation (IBM/ICL).
By forcing the least significant bit of the mantissa to one whenever there is any 'spill' from the less significant end of the adder.
By adding one if the 'spill' is greater than one half the least significant bit of the mantissa.
The first of these is seriously biased if numbers of the same sign are being used (e.g. air pressure), and is the least accurate. The second is unbiased providing the condition is observed. The last has the least deviation from the correct result, but takes longer to execute and/or is more expensive in equipment. It is not biased if the case when the spill is exactly half the least significant bit is treated as round to even [4]. In some special circumstances it may be required to round towards zero, towards $+\infty$, or towards $-\infty$, rather than to the nearest, which is what has been under discussion.

A full discussion of carry generation and rounding is beyond the scope of this paper. However, the following points must be noted.

a) The end around carry for sign and magnitude, or forced carry for twos complement should only be added in if the spill from the less significant end of the mantissa is zero. If the spill is not zero then a carry at the proper place in a theoretically infinite length adder would not reach the bottom of the finite implementation.
b) Rounding by forcing a one is recommended for two reasons.
   (i) It is unbiased.
   (ii) It is a good compromise between accuracy and speed of implementation.

Rounding must be performed after normalisation, but determination of rounding action does not need to wait until then [5,7]. It is noted that if the alignment shift is zero or one place, a large normalisation may be needed, but there is at most two digits of spill. This may be due to either alignment, or mantissa 'overflow' or both. If the alignment shift is greater than one place, then the normalisation is, at most, one place. Thus one guard digit at the least significant end of the adder will be needed, and suitable logic can prepare the round bit prior to the end of normalisation. Where rounding by addition is required, the addition must wait until after normalisation. This adds considerably to the time, as a second cycle of the system of Fig. 1 is needed. Alternatively, a second adder can be included. In either case mantissa overflow could reoccur, and thus further normalisation and rounding.

For both these problems it is necessary to detect mantissa spill. The proposed approach to this is to OR together groups of four bits in the mantissa slice [see section 3.2], thus giving a one out if any of the four bits are ones [6]. The resulting outputs are fed to a shift matrix (see fig.1) consisting of a single 10808 IC, and shifted <u>left</u> by an amount that is complementary to the mantissa alignment. Thus for a 56-bit mantissa and an exponent base of 16, the shift will be 56/4 + 1 - RD. The OR of the outputs of the shifter will then be the spill if RD is less than 15. For values of RD of 15 or greater, the spill is obtained by forcing the shift to be a circular, rather than a logical shift. This method gives the effect of an infinite length shifter. For an exponent of base 2, some additional logic will be necessary to cater for shifts which are not multiples of 4 bits. In fact three or four 'guard' bits at the least significant end of the mantissa will handle this. Some of these guard bits are needed for rounding anyway.

c) The MOSIBIT logic is best performed using priority encoders. Again, use can be made of a 4-bit combination from the mantissa slice, plus some extra logic for the last four bits when the exponent base is not 16.

It has been estimated that the carry, rounding and AOD logic, together with multiplier decoding, could be implemented in 107 ULA cells, and with 52 pins. Although only one of these chips is required per unit, it does replace around 20 small scale integrated circuits, and is well worth while.

	ICs	Pins/IC.	
Mantissa slice (56 bits)	14	60 (est)	chip
exponent slice	2	40 (est)	carriers
end effects	1	52 (est)	"
Shifter (10808)	9	48 quil	
Adder (10181/10179)	24	24/16	
ACC register (10141)	16	16	
MOSIBIT	8	16	
Divide decode	20		
	94		

Table 2 Cost of a floating-point unit

4. Conclusions.

The overall cost of a floating point unit designed to handle 64-bit numbers in ICL/IBM format, and with no double length or mathematical function facilities would be around 94 packages, excluding control. Table 2 lists the details. It is believed that the necessary control would allow the unit to be implemented in under 120 packages. The estimated speed is under 120ns for floating point add and subtract; multiplication time is in the region of 1.2μs, and a divide time around 4μs. The major portion of the times are made up of the shifter times for addition, and the ALU times for multiplication. They are, therefore, dependent on which of the two versions of the ULA is used only to the extent of about 30%. If the support circuits are F100K rather than 10K, the package count reduces from 94 to 73, and times are correspondingly improved, notably for multiplication (as yet there is no F100K version of the 10808).

Addition of a register file, a ROM and a multiplexer to enable a more powerful unit to be built would increase the cost by about 50 ICs. This is a factor of about 4 less than in a current MSI design, and a factor of about 20 down on a 1968 SSI implementation with a less powerful function set. The speed may be about twice that of the MSI design, not allowing for reductions in layout problems (F100K version).

Compared to single chip floating point processors, the number of ICs is obviously much greater. The majority of these processors normally work with only 32 bit numbers and need to load them 8 or 16 bits at a time, a severe limitation. Floating point add times are usually in the region of tens to hundreds of microseconds, and are severely operand dependent. Thus the time cost product of this proposal compares very favourably with these other devices.

The project described is one step in a longer term intention to implement a powerful processor in a small number of ICs. The ULA approach puts the costs of LSI within the budget of relatively small quantity production. The lowering of the cost of an individual floating point unit also makes it easier to contemplate moderate to large numbers of such processors linked so as to perform high throughput array processing.The numbers of ICs are still larger than might be wished, due to the relatively small ULAs at present available at this speed. The arrival of larger devices would be welcomed, since it would enable larger slices to be placed on a chip. Units with still longer word lengths also become economical.

<u>Acknowledgemenys</u>

Thanks are due to the Science Research Council for funding this project.

<u>References.</u>

[1] Gosling,J.B., Kinniment,D.J., Edwards,D.B.G. : 'Uncommitted logic array provides cost effective multiplication even for long words'. IEE J Comp.& Dig. Tech. 2 1979 pp 113-120.

[2] Morris,D.M., Ibbett,R.N. : 'The MU5 Computer System' Macmillan 1979.

[3] Edwards,D.B.G., Knowles,A.E., Woods,J.V.; MU6G : 'A new design to achieve mainframe performance from a mini-sized computer'. ACM SIGARCH Newsletter 1980 p161-167.

[4] ACM SIGNUM Newsletter. Special issue on the proposed IEEE Floating Point Standard. October 1979.

[5] Gosling,J.B : 'Design of arithmetic units for digital computers' Macmillan 1980 (Springer-Verlag).

[6] Zurawski,J.H.P : 'High performance evaluation of division and other elementary functions' Ph.D Thesis, University of Manchester 1980.

[7] Sterbenz,P. : 'Floating Point Computation'. Prentice Hall 1974.