DRAFT: Do not distribute

MIT Concurrent VLSI Architecture Memo ??

Massachusetts Institute of Technology
Artificial Intelligence Laboratory

The Architecture of the MAP Floating Point Unit

Draft Version

Steve Keckler

January 20, 1996

Abstract

This document describes the microarchitecture of the floating point unit, which occupies several execute stages of the cluster pipeline in the MAP chip.

1 Overview

The MAP floating-point unit (FPU) implements floating point operations as well as some integer functions, such as multiply and divide. The FPU is split into two main units: the multiply add (MULA) unit and the divide/square root (DSR) unit. In addition, an FSEND unit sequences messages contents across the C-Switch in response to send operations in the FPU. Figure 1 shows a top level view of the FPU.

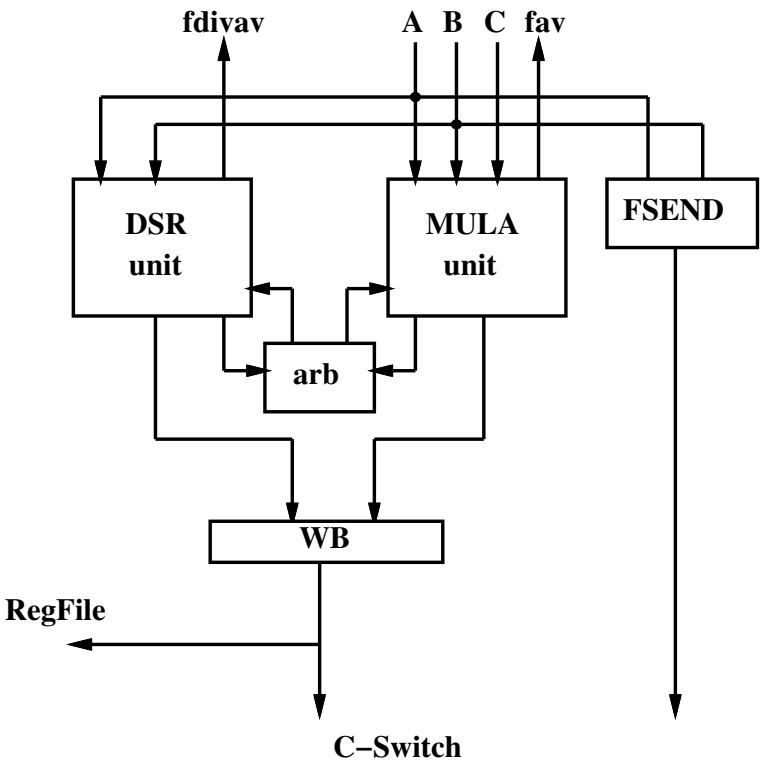

Figure 1: Top Level FPU diagram.

2 The IEEE Specification

2.1 Numerical Formats

The M-Machine will support only the IEEE double precision floating point format, as shown in Figure 2.

The value `V` of a number is specified as follows:

1. if $e = 2047$ and $f \neq 0$, then $V = NaN$

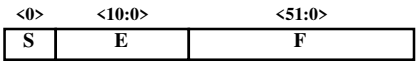

Figure 2: IEEE double precision format.

2. if $e = 2047$ and $f = 0$, then $V = (-1)^s \infty$
3. if $0 < e < 2047$, then $V = (-1)^s 2^{e-1023}(1.f)$
4. if $e = 0$ and $f \neq 0$, then $V = (-1)^s 2^{e-1022}(0.f)$
5. if $e = 0$ and $f = 0$, then $V = (-1)^s 0$ (zero)

The rounding performed at the output of the multiply-adder and the divider will supports denormalized numbers (case 4 above) for gradual underflow. Floating point overflow is handled by $\pm\infty$.

2.2 Rounding modes

The *IEEE-754* floating point specification requires 4 rounding modes, but the MAP will only implement one: round to nearest even. This means that the mantissa of a result will be rounded to the nearest 52 bit fraction. In case of a tie, it rounds to the nearest even fraction (lsb = 0). Rounding to nearest even reduces systematic rounding errors. The implementation of this will require computation of a *sticky* bit (the logical OR of all of the "insignificant" bits) to be used during rounding. Note that rounding must be performed after normalization in order to reduce the size of the carry chain used in rounding, and to make the rounding logic simpler.

2.3 Comparisons

The *IEEE-754* floating point specification requires defined behavior for four relations: less than, greater than, equal, and unordered. The MAP implements four comparison operations FLT, FLE, FEQ, FNE. The following table shows the results of comparisons of various operands.

X op Y	$-\infty$	Finite	∞	NaN	ERRVAL
$-\infty$	=	<	<	EXPT	EXPT
Finite	>	cmp	<	EXPT	EXPT
∞	>	>	=	EXPT	EXPT
NaN	EXPT	EXPT	EXPT	EXPT	EXPT
ERRVAL	EXPT	EXPT	EXPT	EXPT	EXPT

The entry `cmp` indicates that the result is based on $X - Y$. This behavior is similar to *affine* mode in the *IEEE-754* floating point specification.

2.4 Exceptions

The *IEEE-754* floating point specification requires exceptions to be taken on various conditions. It also requires these exceptions to be largely programmable, meaning that control bits are required to determine which exceptions are to be taken and which are to be ignored. On the MAP the exception behavior will not be programmable. Like the integer unit, the FPU will employ error values (ERRVALs) to defer exceptions. Error values will be generated if a valid result can not be formed. Error values will be propagated if an operand is an errval. NaN (not a number) will not be generated; instead error values will be used for this purpose. Arithmetic on ∞ is allowed: $(+\infty) + (+\infty) = (+\infty)$ and $(-\infty) + (-\infty) = (-\infty)$, but $(+\infty) + (-\infty) = ERRVAL$.

Exceptions will only occur on operations that cannot specify an ERRVAL as an output. This includes FNE, FEQ, FLT, FLE, and the family of FSEND operations. The FSEND operations have the same error semantics as the ISEND operations.

Note that this exception behavior largely precludes the use of exception handlers to extend the capability of the floating point unit in software.

3 Interface Signals

The FPU resides within the cluster. It receives instructions from the SZ pipeline stage and and writes results via a private register write bus or via the C-Switch. Two signals, (`fav` and `fdivav`) are delivered to the SZ stage indicating whether a floating point unit operation, or a floating point divide operation may issue. The standard C-Switch interface will be used. In addition the message composition registers will be accessed in the same manner for ISEND and FSEND.

4 Multiply and Add

The multiply and addition functions are implemented in a single multiply add unit, which must perform 64 bit operations in order to handle integer multiply. The Multiply-Add (MULA) unit performs $A \times B + C$. When only a multiply is desired, $C = 0$. When only an add is required, $B = 1$. The MULA unit consists of the following subunits.

- 64 bit Radix 8 Booth encoder (for B)
- 64 bit CPAs for generating -A, +3A, -3A
- Registers for $\pm A$ and $\pm 3A$ for partial product generation
- 64 bit two's complement multiply array (128 bits wide at result)

- 128 bit carry propagate adder to combine carry and save outputs of multiply array.
- Normalization and sticky bit calculation for rounding
- 52 bit shifter to align multiply result with C, or vice-versa (if necessary)
- 52 bit carry propagate adder $(A \times B) + C + S$ (sticky bit).
- 64 bit shifter for normalization of result
- 64 bit carry propagate incrementer for rounding of result
- Exponent adder and control logic
- Sign bit comparator and control logic

The effect of this is that FMULA is equivalent to FMUL followed by FADD. The components of the MULA unit is further described below, and are shown in Figure 3.

Booth Encoder: A radix 8 booth encoder reduces the number of partial products to 22. This requires that multiples of A (0, $\pm A$, $\pm 2A$, $\pm 3A$, and $\pm 4A$) be generated. If shifting is used for power of 2 multiples, only $\pm A$ and $\pm 3A$ need to be saved in registers and distributed to the partial product multiplexors. The negative parts allow the multiplier to use (mostly) simple carry save adder (CSA) cells.

Multiply Array: The multiply array must handle 64 bit inputs for integer multiply. The inputs to the array are two's complement numbers. The 54 bit fractional part of a floating-point number is positive. The adjustments to the sign bit are performed in parallel. The multiplier array is split into two halves: one half accumulates the partial products from the low 33 bits of B, while the other accumulates the partial products from the high 31 bits of B. This structure is shown in Figure 4. The delay through the array is:

$$10 \times t_{CSA} + t_{CSA} + t_{CSA} = 12 \times t_{CSA}$$

where t_{CSA} is the delay to two XOR gates ($t_{CSA} \cong 600ps$).

An alternative would be a radix 4 multiplier. This requires only two registers ($\pm A$), but results in 32 partial products and 30 CSAs instead of 20.

The multiplier array is followed by a 128 bit carry propagate adder (CPA) with a similar structure as that found in the Integer unit. This adder sums the outputs of the last CSA resulting in a 128 bit partial sum. The integer multiply instruction returns only the bottom 64 bits, discarding the overflow.

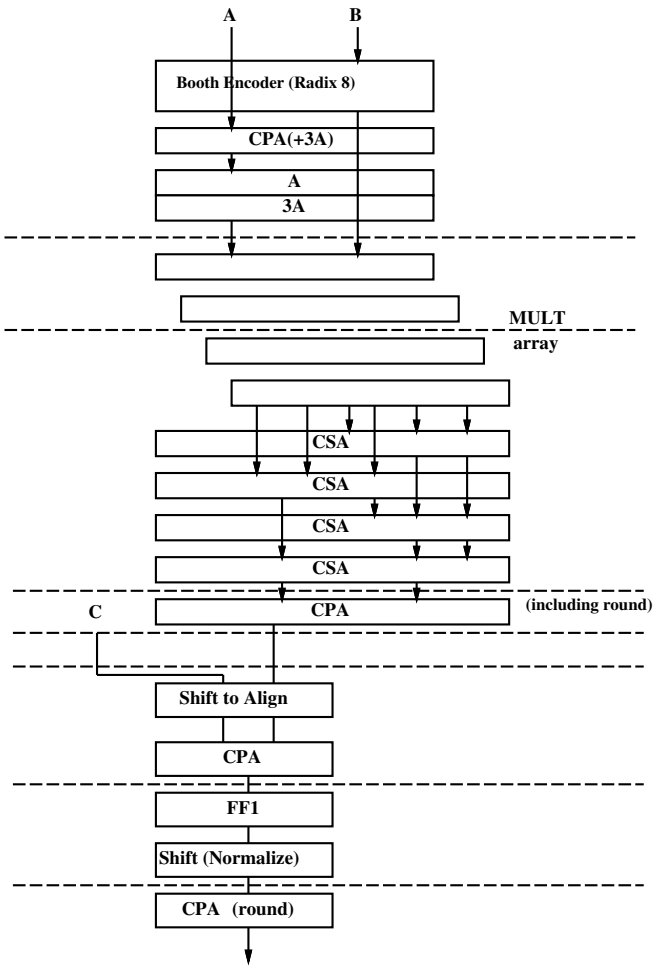

Figure 3: Block diagram of MULA unit.

Preparation for addition: After the multiply, the 128 bit sum is normalized (shift of at most 1) and the sticky bit is computed (logical OR of the least significant bits of the mantissa). Then a right shift is performed on the smaller of AB or on C (as determined by their exponent) to align them. At this point, only the fractional portion of a floating point number is computed, so the datapath is reduced to 52 bits. However, a 64-bit wide path is required to deliver the integer multiply result to the MULA output and to deliver the HMUL result to the rounder.

Addition: A 52 bit CPA adds AB, C, and the sticky bit. Additional add inputs are provided for the necessary inversion of inputs for addition or subtraction, as a standard two's complement CPA is used.

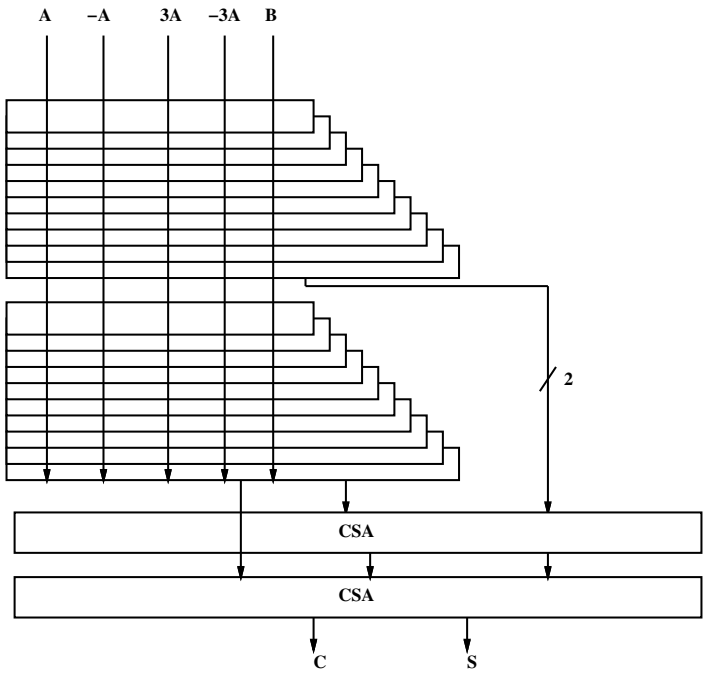

Figure 4: Configuration of multiply array.

Result generation: After the add is performed, the fraction is in a two's complement representation and must be aligned and returned to signed magnitude. The result is first translated into signed magnitude (by inversion) but the carry-in is saved. Then the first-ones finder determines how far to shift the result and the carry-in. Finally the result is rounded with a final carry propagate adder, adding the result, carry in, and sticky bit, if necessary.

Exponent adder: The exponent adder determines adds the exponents from A and B and determines which to shift: AB or C. Other modifications to the final exponent are performed during normalization of AB and of AB+C.

Sign bit comparator: The sign bit determines the conversions necessary prior to addition or subtraction of floating point numbers. All subtraction is performed using two's complement addition. The sign bit is set as a result of the addition. If the result of the add is a negative two's complement number, the sign bit is set to 1 and the fractional portion is converted back to a positive number.

Pipeline Registers FMUL, IMUL, HMUL, and FMULA will each take 3 cycles to traverse the MULA unit. The pipeline registers will be placed as needed, but the approximate division of the time into cycles is displayed in Figure 3. FADD, and FSUB need only the last stage

and if no other operation is occupying the second or third stage, may complete in a single cycle.

HMUL: The HMUL operation is a fixed point multiply. The range of representable numbers in fixed point notation is $-2.0 \leq X \leq 2.0 - 2^{-62}$. The operands from the HMUL operation run through the multiplier just like the operands for IMUL. Unlike IMUL, HMUL returns the high 64 bits of the result. At the output, bit 126 is removed and bits 125–0 are shifted up one position (for realignment of fixed point representation). The normal round to nearest even is performed for the high 64 bits, using the existing rounding hardware. (Note, rounding two's complement negative numbers is more difficult than rounding signed magnitude, as a subtraction is required–truncation would certainly be easier).

A simple 5 bit example follows:

$$\begin{array}{rrcr} & 00010 & = & 1/4 \\ \times & 11100 & = & -1/2 \\ \hline & 1111111100 & & \\ & 11111 & = & -1/8 \end{array}$$

4.1 Conversion

The conversion operations are also handled in the MULA unit. Both ITOF, FTOI, and FTOIU may use the existing hardware in the third stage of the pipeline. If no operation are ahead in the pipeline, each of these operations may complete in a single cycle.

ITOF: ITOF is performed directly by setting the exponent to 64 and passing the entire number to the stage 3 normalizer which converts the number into floating point representation, including conversion to signed magnitude. Some bits of precision may be lost due to roundoff in the 52 bit fractional portion.

FTOI: FTOI is performed by shifting the floating point number by the appropriate amount in the alignment shifter. High bits which overflow are discarded. If a sign conversion is required, the CPA before the normalization shifter is used.

FTOIU: This is the same as FTOI, except that one more bit of integer magnitude is maintained, and the mantissa is assumed to be unsigned.

4.2 Comparison:

The four comparison operations (FNE, FLT, FLE, FEQ) each use the third stage of the MULA pipeline. A subtraction is performed on the mantissas and on the exponents, and the sign bits are compared. The result of the mantissa check is the same as that comparison performed in the integer unit.

5 Divide/Sqrt

The divide/square root (DSR) unit is separate from the MULA unit. It is not pipelined and uses the SRT algorithm to compute both the divide and square root function. A radix 4 SRT will be used, which retires 2 bits every iteration through the select/add cycle. In addition, registers are provided to double-cycle the adder (ie. use the adder both on *clk* and $\overline{clk}$). Thus 4 bits of will be retired every clock cycle. This results in a 64 bit divide in 16 processor cycles. The schematic of the double cycling logic required is shown in Figure 5. Two additional cycles are required: one at the beginning for alignment of the operands for IDIV and IDIVU, and one cycle at the end for normalization for FDIV and FSQRT. Like the multiplier, division is performed using two's complement arithmetic.

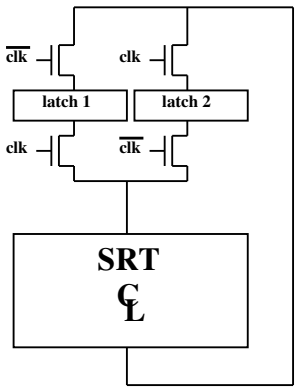

Figure 5: Double cycling over the combinational SRT logic.

Every sub-cycle, the following equations are computed:

$$R_{j+1} = \begin{cases} 4R_j + 2|D| & & & 4R_j & < & -2D \\ 4R_j + |D| & -2D & \leq & 4R_j & < & -D \\ 4R_j & -D & \leq & 4R_j & \leq & D \\ 4R_j - |D| & D & < & 4R_j & \leq & 2D \\ 4R_j - 2|D| & 2D & < & 4R_j & & \end{cases}$$

$$q_{j+1} = \begin{cases} -2 & & & 4R_j & < & -2D \\ -1 & -2D & \leq & 4R_j & < & -D \\ 0 & -D & \leq & 4R_j & \leq & D \\ 1 & D & < & 4R_j & \leq & 2D \\ 2 & 2D & < & 4R_j & & \end{cases}$$

where R_{j+1} is the residual remainder and q_{j+1} is the next set of quotient bits computed. The components required for the SRT divider are:

- +D register for divisor
- 1 bit shifter to generate +2D
- 64 bit adder (CPA) with carry in to generate $X-D$ and $X-2D$ (also used for rounding)
- Remainder register R
- Quotient register Q
- Comparison logic (7 bits) to select D, 2D
- Register for $1.5 \times D$ (7 bits)
- Exponent subtraction
- 64 bit shifter for pre-divide alignment and post-divide normalization.
- Converter from signed digit to integer or FP

These components are shown in Figure 6.

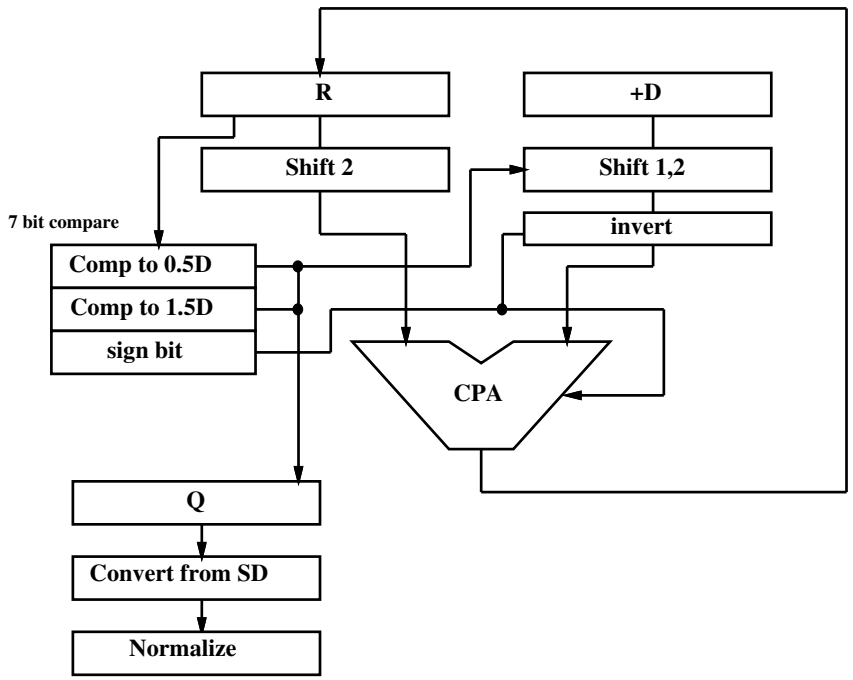

Figure 6: SRT divide and square root unit.

FSQRT: Floating point square root is performed using the same hardware as the divider. Some additional logic is required for selecting the square root digits (q_i), which is different from that used in for the divide. In addition a small amount of logic may be required to predict the first few bits of the root. The exact algorithm to be used has not yet been selected.

6 FSEND

FSEND is implemented in the same manner as ISEND. If the FSEND operands (destination and dispatchIP) are not valid then an exception occurs. Otherwise, the operands flow through to the FSEND controller and are dispatched across the C-Switch. The body of the message is fetched using the write-back stage wires to the register file, as in ISEND.

7 Write-back stage

The write-back stage merges the results from the DSR and MULA units. It performs arbitration between these units. The winner of the arbitration marks its destination register present and on the next cycle delivers the result. The same arbitration applies to access to the C-Switch. In arbitration, the result of the DSR always wins (since it has a maximum throughput of 1 result every 18 cycles). If an FSEND is in progress when the result of the DSR unit becomes available, then the DSR stalls.

8 Area Estimate

Using the component area estimates provided by Andrew Chang, the overall area estimate of the FPU is provided in Table 1. Only the large components (ie. wide datapath) are included as significant.

9 Issues:

- Remainder instruction can be implemented without too much trouble in DSR unit.
- Addition of immediate generating instructions in MULA unit
- Detailed design of the two's complement FP array
- Details surrounding exactly where and how to round.
- Can we remove the CPA for combining the outputs of the multiplier array (ie. keep the result in carry save form) and add in C in this form. This would required two large shifters (one for carry and one for sum).

- Do we need to perform rounding on HMUL results, or can we get away with truncation. Rounding may require subtraction.
- Determine SQRT parameters.
- Do one's complement arithmetic in multiplier (convert two's complement integers into signed magnitude.
- Use two adders in two different cycles (4 cycles?) Add then round.

Component	bits	width(λ)	length(λ)	area(λ^2)
Booth encoder	64	7680	500	3.8×10^6
Adders ($\times 3$)	64	7680	900	2.7×10^7
Registers ($\pm A$, $\pm 3A$)	64	7680		6.8×10^6
CSAs ($\times 20$)	128	7680	7000	5.2×10^7
CPA (combine CSA output)	128	7680	1800	1.3×10^7
Shift (align)	64	7680	735	5.6×10^6
CPA (AB + C)	64	7680	900	6.8×10^6
Shift (normalize)	64	7680	735	5.6×10^6
CPA (round)	64	7680	900	6.8×10^6
total				1.3×10^8
Registers (D, R, Q)	64	7680		2×10^7
CPA	64	7680	900	6.8×10^6
Shift (normalize)	64	7680	735	5.6×10^6
CPA (round)	64	7680	900	6.8×10^6
total				3.9×10^7
overall total				1.7×10^8

Table 1: Area estimate of FPU.