The floating-point unit of the PowerPC 603e microprocessor

by R. M. Jessani
C. H. Olson

The IBM PowerPC 603e™ floating-point unit (FPU) is an on-chip functional unit to support IEEE 754 standard single- and double-precision binary floating-point arithmetic operations. The design objectives are to be a low-cost, low-power, high-performance engine in a single-chip superscalar microprocessor. Using less than 15 mm^2 of the available silicon area on the chip (the size of the PowerPC 603e microprocessor is 98 mm^2) and operating at the peak clock frequency of 100 MHz, an average single-pumping multiply–add–fuse instruction has one-cycle throughput and four-cycle latency. An average double-pumping multiply–add–fuse instruction has two-cycle throughput and five-cycle latency. The estimated performance at 100 MHz is 105 against the SPECfp92™ benchmark.

Introduction

The skeleton of the IBM PowerPC 603e™ FPU architecture is optimized to perform a multiply and an add operation [1–3] in a single floating-point instruction:

$$FRT = FRA * FRC + FRB, \quad (1)$$

where *FRT* is the target operand, and *FRA*, *FRB*, and *FRC* are the three source operands. Each of the four operands can be any one of the 32 user-accessible floating-point registers (FPRs). The floating-point move, add, subtract, and multiply instructions can easily be derived from this multiply–add–fuse instruction by forcing *FRC* to the constant 1.0 or *FRB* to the constant 0.0.

The 603e[1] FPU data flow architecture comprises three independent pipeline stages—the multiply, the carry-propagate-add (CPA), and the Writeback (WB). Each stage requires only one clock cycle to execute in a normal situation. In addition to the 32 user-accessible FPRs, there are four floating-point rename buffers to provide temporary storage for the execution result. These rename buffers are used to allow fast result forwarding for the next instruction and to avoid storage dependency problems. There is hardware to support floating-point divide, floating-point-to-integer conversion, denorm input operands, denorm result, IEEE exception handlers, three new graphics instructions, and non-IEEE mode for fast execution.

Figure 1 shows the 603e FPU as a function unit which takes instructions from the dispatch unit. The operands can come from either the FPR or the rename buffers. The result is placed at the preassigned rename buffer only.

[1] "PowerPC 603e" and "603e" denote the same microprocessor.

0018-8646/96/$5.00 © 1996 IBM

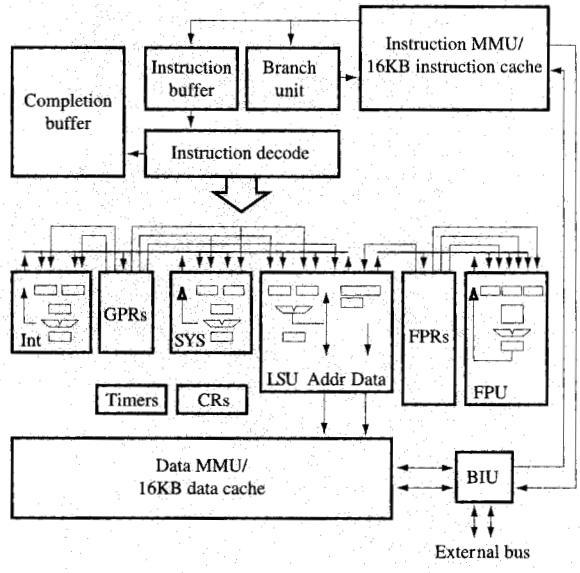

Figure 1

Block diagram of the PowerPC 603e microprocessor.

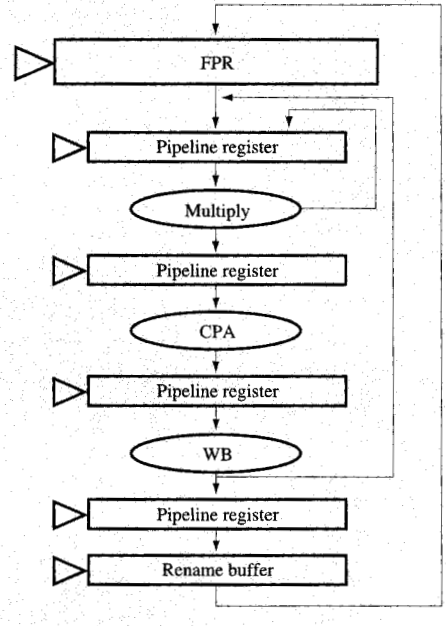

Figure 2

Major stages of the 603e FPU.

The 603e FPU is a fully static-register base implementation with LSSD testability. The implementation employs an ASIC design methodology approach to speed up the design cycle time for chip integration; however, most of the building elements of the 603e FPU are fully custom designed for the specific application in order to achieve peak performance. The control logic is synthesized, placed, and routed automatically. The logic is verified by a random test generation program through both an FPU stand-alone simulation model and a whole-chip simulation model.

The 603e FPU has two types of execution modes—scientific mode and real-time mode. The scientific mode is implemented for precise scientific computations and engineering applications that require high precision and execution power. This mode conforms to ANSI/IEEE Standard 754-1985 [4], the "IEEE Standard for Binary Floating-Point Arithmetic" (hereafter referred to as the "IEEE standard"), including all four rounding modes and exception status reporting, but it does have a dependency on supporting software in order to do so. All floating-point operations conform to that standard, except when software sets the floating-point non-IEEE mode (NI) bit in the floating-point status and control register (FPSCR) to 1, i.e., real-time mode, in which case floating-point operations conform to a subset of the IEEE standard—all results are produced without software assistance (without causing a floating-point-enabled-type program interrupt, a floating-point-assist interrupt, or a fast-trap). All exceptions are handled by hardware, and default numbers such as quiet not-a-number (QNaN), maximum, and zero are output according to the exception.

Basic instruction pipeline stage and timing

The PowerPC 603e FPU comprises three major stages (**Figure 2**). The first stage of the FPU engine is the multiply stage. It performs the main multiply function of *FRA* times *FRC* using a 53 by 28-bit Booth recoding Wallace tree multiplier array [5, 6] to generate the accumulated partial product in the sum-and-carry format. Concurrently, *FRB* is only right-shifted to align with the result of *FRA* times *FRC*. Finally, the aligned *FRB* and the result of *FRA* times *FRC* are compressed by a 3-to-2 carry-save adder. To save silicon area, the multiplier array is only implemented with half the size of a full 53 by 53-bit array using three levels of 4-to-2 carry-save adders. For a double-precision multiply operation, it requires double pumping through the multiplier array.

The second stage is the carry-propagate-add stage, which performs a 161-bit one's complement add using a carry-lookahead adder. The result of the operation goes through a leading-zero detector for normalization shift-count calculation.

The third stage is the WB stage, which performs the normalization left shift, rounding, and status generation operations. The final result is stored in one of the four

rename buffers and/or forwarded back to the multiply stage for the next instruction execution. The rename buffer content does not update the FPR until the instruction is architecturally completed.

Figures 3 and **4** respectively show typical timing examples involving three single-pumping and three double-pumping instructions. For single-precision multiply instructions (and instructions which do not have the multiply function incorporated in them) without data dependency, the throughput is one per clock cycle, and the latency is four cycles. For double-precision multiply instructions without data dependency, the throughput is one per two clock cycles, and the latency is five cycles.

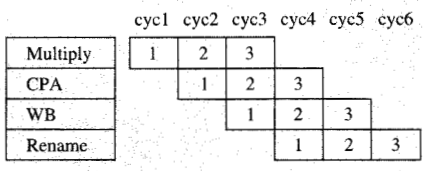

Figure 3

Timing for single-precision multiply in the 603e FPU.

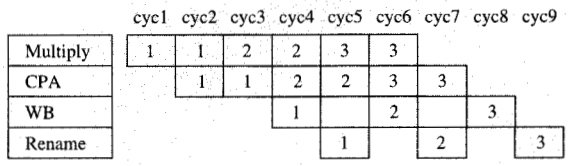

Figure 4

Timing for double-precision multiply in the 603e FPU.

Architecture/implementation

Figure 4 shows the major building elements in the 603e FPU for the multiply–add–fuse instruction. The three independent pipeline stages (multiply, CPA, WB) require one clock cycle to execute under a normal situation. Extra cycles are required for situations such as the following:

- For a full double-precision multiply operation, an extra cycle is required for iterating in the multiply stage to double-pump the operand or intermediate result through the multiplier array.
- The normalization shifter is implemented with a 63-bit left shifter, and in the event of mass cancellation in the mantissa calculation, the WB stage could require one or two extra clock cycles to execute.
- Underflow and overflow exceptions require an extra cycle in the WB stage for exponent correction.
- Denormalization of results requires an extra cycle.

- *Bypass unit*

The block diagram in **Figure 5** shows a bypass unit—used to handle abnormal executions (i.e., the abnormal operands NaN and infinity, and abnormal operations such as divide by zero, infinity minus infinity, and infinity multiplied by zero). The normal multiply and CPA stages are bypassed, and the default result is fed to the WB state to simplify the datapath logic. The bypass unit also manipulates the FPSCR for FPSCR instructions, and handles the graphic-instruction floating-point conditional select register.

- *Multiplier array*

The 603e multiplier is a high-performance fully pipelined multiplier which uses radix-4 Booth recoding [5] to halve the summands (partial products) which must be added and a Wallace tree [6] of 4-to-2 CSAs to minimize the time necessary to add the summands together. The multiplier configuration is a 54 × 28 arrangement capable of unsigned operations. The 603e single-precision floating-point multiplication can be operated at the rate of one instruction per cycle. For double-precision multiplications, the operands are double-pumped at each stage to obtain the desired results with the required precision.

Figure 6 shows the multiplier array structure. The two feedback paths for the most significant bits of the sum and carry each use a 2:1 mux which allows the results of the first pass to be incorporated into the second pass.

- *CPA (carry-propagate adder)*

The CPA is implemented as a 161-bit one's complement adder with an end-around-carry adjustment. It accepts one's complement input from the aligned *FRB*, and it accepts the result of *FRA* * *FRC*, which is always positive relative to *FRB* from the multiply stage. The output is represented in sign-magnitude format with the sign bit handled in control logic. The CPA consists of three pieces, as shown in **Figure 7**. Since the lower 26-bit product from the multiply array is generated from the first CPA cycle by the carry-lookahead adder (CLA) and saved, the lower 26-bit portion of the 161-bit CPA is an incrementer instead of a CLA. Since carry-lookahead incrementers are smaller and faster than carry-lookahead adders, the actual CLA is only 88 bits wide.

The propagate from the carry-lookahead incrementer and the group identifier generated by the 87-bit CLA are fed back to the carry-in to correct the end-around-carry problem without a closed loop. XOR gates are used at

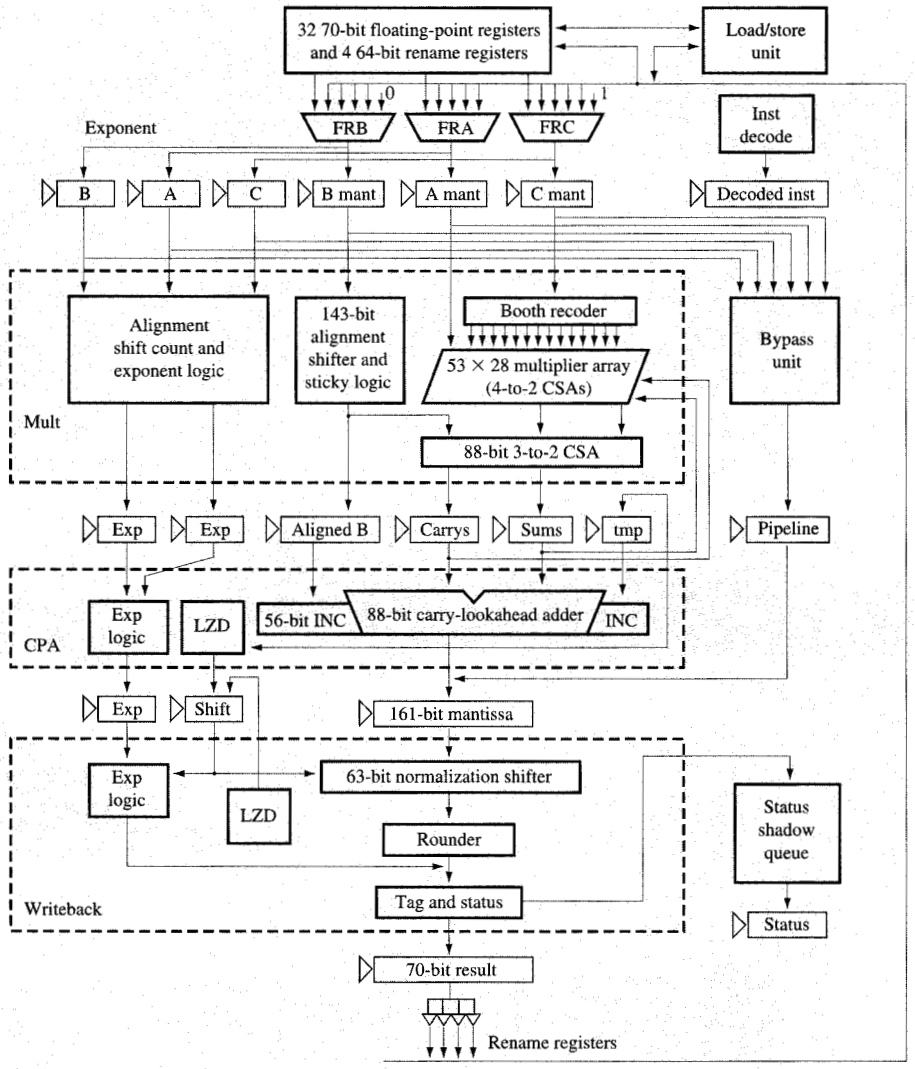

Figure 5

Block diagram of the 603e FPU.

the output to invert the result back to sign-magnitude format. The one's complement adder is used in this implementation to make it easier to convert the result back to sign-magnitude format. The upper 63-bit portion of the 161-bit CPA result is examined for the number of leading zeros for normalization shift count in the next stage.

• *Alignment shifter*

The alignment shifter for the B mantissa is reduced to a 53-bit-input 136-bit-output right shifter through double-pumping. It is implemented as a partial decode having three levels in a multistage structure with partial-shift groups, or as a modulo shifter (maximum shift count of 143), since each nested shift amount is calculated with modulo arithmetic. **Figure 8** shows the three levels, which carry out shifts of binary (0, 1, 2, 3); multiples of 4 bits (0, 4, 8, 12); and multiples of 16 bits (0, 16, 32, 48, 64, 80, 96, 112, 128), respectively. Shifting arising from a negative shift count is prevented by a bypass port in the third level that takes advantage of critical timing. Bypassing enables a quicker shift count calculation for the first two levels. The

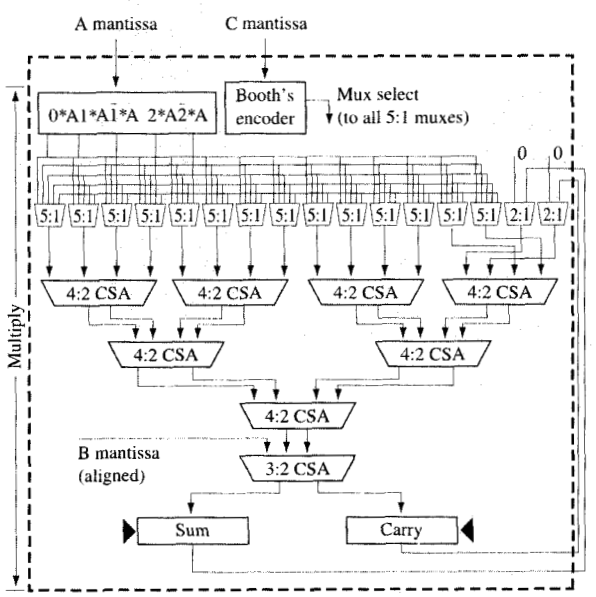

Figure 6

Multiplier array.

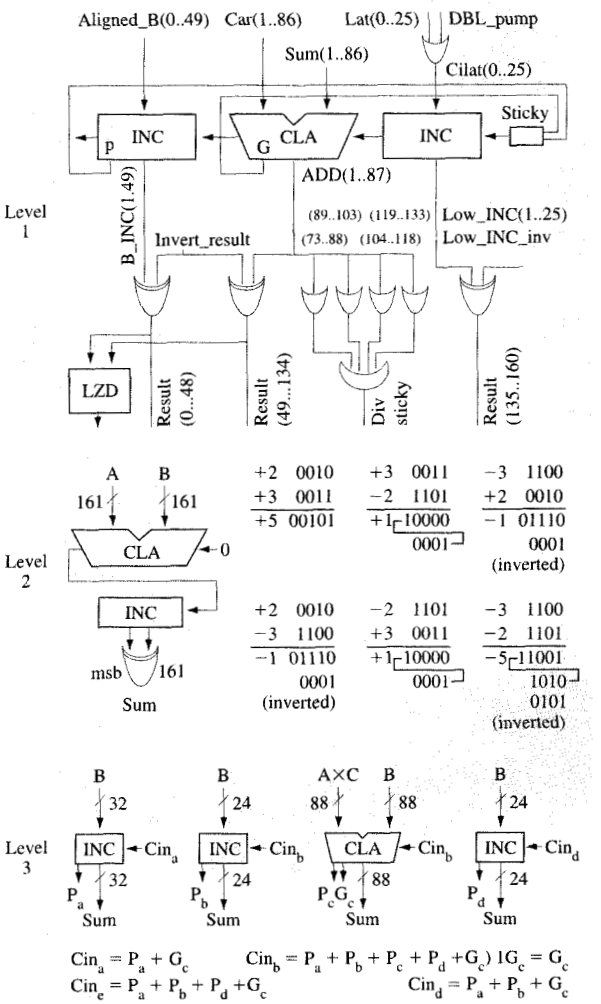

Figure 7

Carry-propagate adder. Level 1: CPA dataflow; Level 2: One's complement adder examples with end-around-carry adjustments; Level 3: Actual implementation for the 161-bit adder divided into four sections.

alignment shifter also includes the automatic sticky-bit detection logic for the bits that are shifted off.

- *LZD (leading-zero detector)*

Three 63-bit LZDs inside the FPU are used to calculate the number of leading zeros. The most time-critical one is at the CPA stage after the 161-bit add. It is implemented by three levels of coarse and fine circuits. The first level contains sixteen 4-bit leading-zero-detect circuits. The second level contains four 16-bit leading-zero-detect circuits. The final level contains one 64-bit leading-zero-detect circuit. The other two LZDs are not time-critical and are implemented by synthesis.

- *Normalization shifter*

The normalization shifter comprises three levels and is not a complete shifter. The maximum shift count is 63. For a normal CPA result, it requires only one pass through to generate the normalization result. For the case of mass cancellation from unlike-sign add operations, it requires one to two extra cycles to normalize the result. A sticky bit is also accumulated for bits beyond bit 53.

- *FPR file*

There are 32 floating-point registers, each containing a 70-bit floating-point number. The format of this 70-bit number includes a 3-bit tag, a 1-bit sign, a 13-bit exponent, and a 53-bit mantissa. The FPR register array has one write port from one of the four rename buses. The data transfer control signals and address for the write come from the completion unit, which allows updates from either the writeback unit or the load/store unit. The FPR register array also has three read ports for latching the three operands (*FRA*, *FRB*, and *FRC*) needed to execute an instruction. The data transfer control signals and addresses for the three reads come from the dispatch unit. Finally, the FPR register array also has one more read port to the load/store unit. The data transfer control signals and address for this read come from the load/store

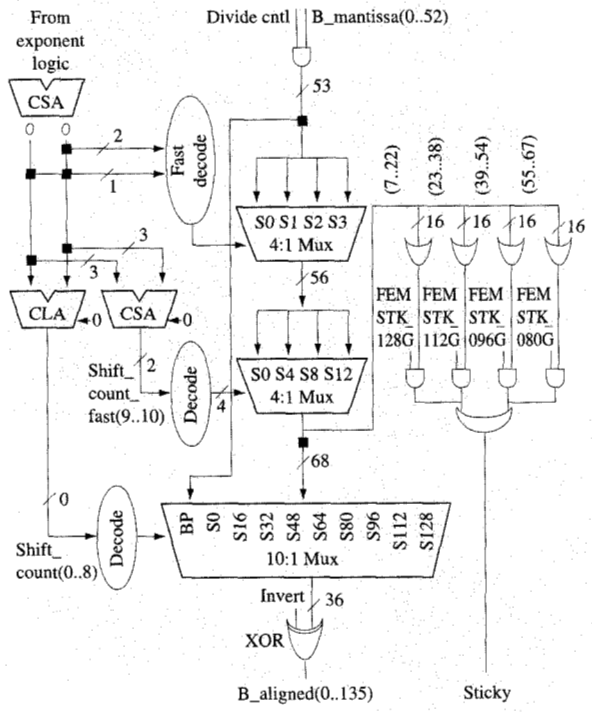

Figure 8

Alignment shifter.

unit. The four reads and one write can take place at the same time.

- *Division*

The 603e FPU employs a 2-bit nonrestoring division algorithm which produces two correct mantissa bits per cycle. A normal single-precision divide requires 18 cycles and a normal double-precision divide requires 33 cycles to execute, utilizing most of the existing hardware.

Exception detection and handling

The 603e FPU detects exceptions in two parts—an early detect in the first cycle of instruction execution (multiply) in the bypass unit, and a late detect in the last cycle of instruction execution (WB). All seven invalid operations exceptions (Inf − Inf, Inf * Zero, SNan, etc.) and the zero-divide exception fall in the early-detect category, while all overflow, underflow, and inexact exceptions fall in the late-detect category. For cases of exceptions with corresponding exception-enable bits that are clear in the FPSCR, early detection in the bypass unit is carried out with exponent and mantissa checking, and appropriate default results are generated, while late detection is handled by generating a meaningful result. For enabled exceptions, early-detect exceptions stall the dispatch unit from dispatching further FPU instructions, whereas for late-detect exceptions an adjusted result is prepared and loaded into the rename bus.

Figure 9 shows the exception module in the 603e FPU. The IEEE standard assumes that it should be possible to identify the instruction that triggers an exception trap. The 603e FPU identifies each executed instruction with a 3-bit IDN (ID number assigned during dispatch), which is passed on to the completion unit along with the finish signal for an instruction. The completion unit handles situations such as misplaced branches and out-of-order completion among the FPU, the fixed-point unit (FXU), and the load store unit (LSU), and fires a complete signal for the FPU to update architected registers.

FPSCR control

The floating-point status and condition register (FPSCR) is implemented inside the FPU. Each bit in the FPSCR belongs to one of three categories: 1) report exception (sticky), 2) status, and 3) enable and programming mode control bits. The FPSCR is accessed by the following two groups of instructions:

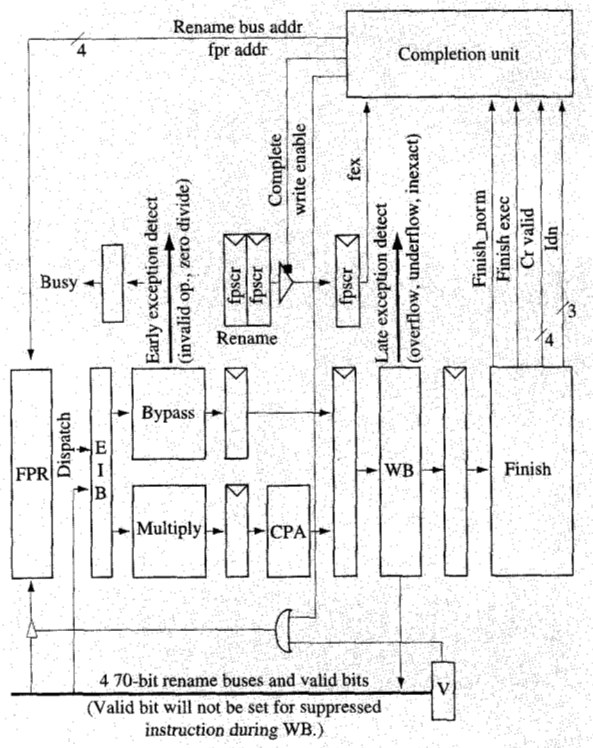

Figure 9

Exception module.

1. *FPSCR instructions* This group requires machine serialization.
2. *Floating-point arithmetic, multiply–add, rounding, and conversion and compare instructions* This group does not require instruction serialization. However, two rename FPSCRs and one master FPSCR are required to maintain the FPSCRs in program order. The update from one of two rename FPSCRs to the master FPSCR takes place when the completion unit announces that the associated instruction is completed.

The FPSCR is implemented with a full 32-bit master architectural FPSCR and two 17-bit FPSCR rename buffers which are implemented with a stack structure as shown in **Figure 10**. The FPSCR rename buffers contain the non-sticky-status bits of *ox, ux , zx, xx, vxsnan, vxisi, vxidi, vxzdz, vximz, vxvc, vxcvi, fr, fi,* and *fprf*.

The FPSCR renames are updated by the instruction which is at the final cycle of the WB stage. When the completion unit retires the finished FPU instruction, the architectural FPSCR is updated from the bottom of the FPSCR rename stack, and the stack is decremented. When both FPSCR renames in the stack are full, the FPU will be stalled until the FPSCR rename which is on the top of the FPSCR rename stack is free again. Figure 10 and **Figure 11** show the implementation of the FPSCR rename and the state diagram for the FPSCR rename control. Under the assumptions listed below, the performance lost due to stall caused by the lack of FPSCR rename buffers is minimal:

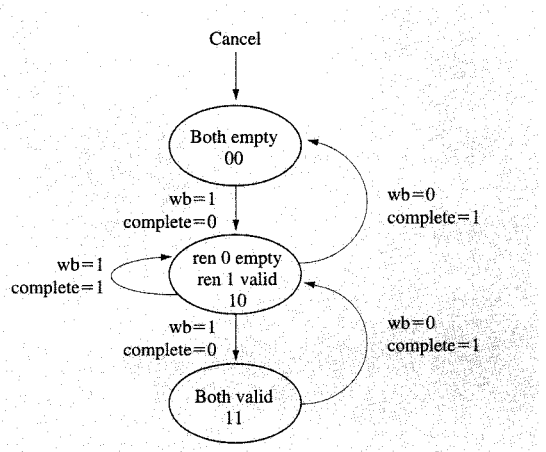

Figure 11

State diagram for the FPSCR rename control.

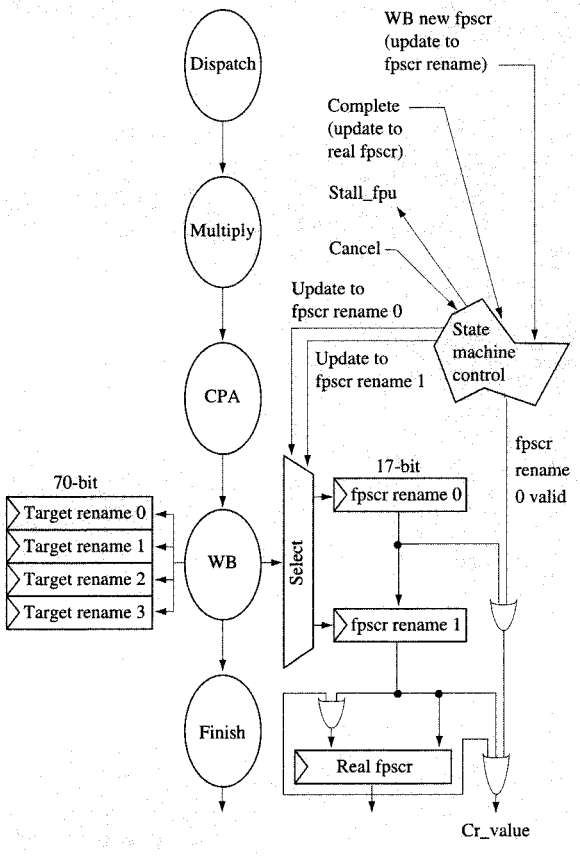

Figure 10

Implementation of the FPSCR rename stack and the architectural FPSCR.

1. Given that there are only four target rename buffers which are being shared with the LSU, and assuming that on the average, the FPU uses only half the number of the target rename buffers, the FPU will be stalled in any case by the lack of target rename buffers. Therefore, only two FPSCR rename buffers are needed to match the four target rename buffers.
2. The probability of having a long instruction (e.g., FXU divide, LSU with a miss) in front of a floating-point instruction is very small. Most FPU applications will not be delayed.

Prenormalization/denormalization

In the 603e FPU, denormalized source operands are prenormalized by using the normalization shifter in the writeback stage.[2] The bypass unit contains the logic which detects denormalized source operands and stalls the FPU from accepting further instructions. The denormalized operand bypasses the multiply and CPA stages and is fed

[2] R. Jessani, C. Olson, and M. Putrino, work in progress.

to the normalization shifter. Depending on the number of denormalized sources, three to five extra cycles are required before the instruction commences execution. A single denormalized source utilizes three extra cycles, where two and three denormalized sources take four and five extra cycles, respectively. Prenormalization is carried out for all instructions which could potentially cause an error in certain cases with denormalized source operands. The prenormalized number is fed back to the EIB for normal execution via the rename bus that was pre-assigned for the target register of that instruction.

The extra hardware required for handling denormalized operands consists of a leading-zero detector (LZD) in the bypass unit, and minimal logic to detect denormalized operands and to stall the FPU. The LZD in the CPA stage fell in a time-critical path and was not used, since introducing a 2:1 mux in the path would compromise timing. Since the bypass LZD was not time-critical, it was implemented using synthesis.

The denormalization operation for denormalized results is handled at the writeback stage. The normalization shifter can only left-shift from 0 to 63 bit positions. After the normalization operation is completed and the final exponent is computed, if a denorm operation is required, the result is looped back to the WB pipeline register with a 56-bit hardwired right shift while the multiply and CPA stages are stalled. Another left shift is performed by left-shifting the result by (56 minus the number of the denorm bit position). The denormalizing operation requires an extra cycle. The extra hardware required is an extra port in the writeback register.

Design methodology

The 603e FPU was described in a proprietary high-level register-transfer design language called Design Structure Language (DSL). The DSL compilers accept hardware constructs in a programlike manner and support many levels of hierarchy for a macro design approach. Customized, hand-placed circuits (off-the-shelf, or OTS, macros) were used to lead to optimal timing and density for memory elements, multiplexors, and challenging functions. Control logic was developed by allowing synthesis to map the high-level, functional model for each random logic macro (RLM) to primitive gates in the technology library. The timing of critical paths was improved by reworking the function, hiding a desired implementation from synthesis, or altering synthesis timing assertions. MCSPICE was used for detail circuit simulation on critical speed paths, before and after layout.

Summary

The IBM PowerPC 603e floating-point unit is an IEEE-754-compliant fused multiply–add design. With complete hardware support for denormalized numbers, overflow and underflow processing, and invalid operation exceptions, the 603e requires the same minimum software envelope as high-end RS/6000™ processors. Requiring less than 15 mm^2 area and operating at 100 MHz, the 603e floating-point unit represents an excellent cost–performance implementation.

References

1. E. Hokenek, R. K. Montoye, and P. W. Cook, "Second-Generation RISC Floating Point with Multiply–Add Fused," *IEEE J. Solid-State Circuits* **25,** No. 5, 1207–1213 (October 1990).
2. R. K. Montoye, E. Hokenek, and S. L. Runyon, "Design of the IBM RISC System/6000 Floating-Point Execution Unit," *IBM J. Res. Develop.* **34,** No. 1, 59–70 (January 1990).
3. T. Olson and B. Stewart, "Floating-Point Architecture of the AM29050," *WESCON/90 Conference Record*, November 13–15, 1990, pp. 214–217.
4. *ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic*, American National Standards Institute, Washington, DC, 1988.
5. A. D. Booth, "A Signed Binary Multiplication Technique," *Quart. J. Mech. Appl. Math.* **4,** Part 2, 236-EOA (1951).
6. C. S. Wallace, "A Suggestion for Fast Multipliers," *IEEE Trans. Electron. Comput.* **EC-13,** 14–17 (February 1964).

Received February 21, 1996; accepted for publication June 7, 1996

Romesh M. Jessani *Motorola, Inc., Somerset Design Center, 6300 Bridgepoint Parkway No. 4, Austin, Texas 78730 (romesh@ibmoto.com).* Mr. Jessani received his B.E. degree in electronics and communication engineering from the Regional Engineering College, Kurukshetra, India, and his M.S. degree in electrical engineering from the University of Texas at Dallas. After graduation, he worked at Advanced Micro Devices on the 80C51 microcontroller, subsequently joining Motorola to work on development of the PowerPC® microprocessor architecture and logic design. He has since worked on several PowerPC portable microprocessors. Mr. Jessani's professional experience includes the floating unit, L1 and L2 cache controllers, and array built-in-self-test design and computer architecture. He is a member of the IEEE.

Christopher H. Olson *IBM Corporation, Somerset Design Center 9730, 11400 Burnet Road, Austin, Texas 78758 (colson@ibmoto.com).* Mr. Olson graduated with a B.S. degree in electrical engineering from the University of Wisconsin at Madison in 1988. He interned with IBM Kingston in 1986 and 1987. After graduation, he joined the IBM RISC System/6000® microprocessor development group in Austin, Texas. Since 1992, Mr. Olson has worked on a number of PowerPC processors as part of the IBM/Motorola/Apple alliance.