The Design of the Fixed Point Unit for the z990 Microprocessor

Fadi Busaba Timothy Slegel Steven Carlough Christopher Krygowski John G Rell

IBM
2455 South Road
Poughkeepsie, NY
12601
(845)435-8697
busaba@us.ibm.com

ABSTRACT
The paper presents the design of the Fixed Point Unit (FXU) for the IBM eServer z990 microprocessor (announced in 2Q '03) that runs at 1.2 GHz [2]. The FXU is capable of executing two Register-Memory instructions including arithmetic instructions and a branch instruction in a single cycle. The FXU executes a total of 369 instructions that operate on variable size operands (1 to 256 bytes). The instruction set include decimal arithmetic with multiplies and divides, binary arithmetic, shifts and rotates, loads/stores, branches, long moves, logical operations, convert instructions, and other special instructions. The FXU consists of 64-bit dataflow stack that is custom designed and a control stack that is synthesized. The current FXU is the first superscalar design for the CMOS z-series machines, has a new improved decimal unit, and has for the first time a 16x64 bit binary multiplier.

Categories and Subject Descriptors
C.5.3 [**Computer System Organization**]: Computer System Implementation - microprocessors. Fixed Point Unit.

General Terms
Performance, Design.

Keywords
Superscalar FXU. Microprocessor.

1. INTRODUCTION
High end microprocessor (CP) development teams use custom circuits for both the dataflow and control logic, and dynamic logic to achieve high Giga-Hertz frequencies (GHz). However, a custom designed macro requires stable logic and is difficult to modify if a late logic bug is found in the design. On the other hand, synthesized macros using parameterized standard cells tend to be slower and larger than custom designed macros. Using custom logic for the dataflow and synthesized macros for the control macros, the FXU design was able to reach the desired frequency while maintaining design flexibility in the control stack for late logic adjustments that results in a shorter turn-around time between development and RITs (Release Interface Tape).

High speed synthesized macros are achieved by a "low-level" VHDL coding style and special tweaking of the synthesis and placement tools. These include macro decomposition, VHDL tuning for critical cones, logic restructuring, placement driven synthesis, dedicated wide wires, usage of tapered gates, and low Vt are all applied to our control macros to have the design meet its set objectives. The main objectives of the z990 FXU is to improve the execution cycles (or CPI) relative to previous FXU designs while meeting cycle time, area and power requirements. Other processor such as the Pentium 4 processor has an FXU design that targets high frequencies by concentrating on a subset of the instruction set and "critical" loops that are important for their workloads, but hurting the performance of other instructions [1]. For example, a small execution core with simple instructions within the FXU runs at double speed while a slower execution part of FXU runs at single speed and with higher execution latencies such as integer shift and rotate instructions which require up to 4 cycles of execution. The presented FXU design has a 64-bit dataflow stack that allows a 64-bit addition with sign extension, rotation, and shifting along with condition code setting to be performed in a just single cycle. This is chosen because of the commercial work-loads that the z990 processor targets.

This FXU design is the first superscalar processor for the CMOS z-series servers [2] [and has many other enhanced features over preceding designs [3-5]. These features include operand forwarding, condition code forwarding, muti-port instruction dispatch, enhanced decimal performance and a binary multiplier.

2. PIPELINE DESCRIPTION

Table 1 shows the six pipeline stages for the processor. These stages do not include instruction fetch cycles (before Dcd stage) and check point cycles (after PA cycle). Dcd is the decode cycle when two instructions can be decoded in parallel. Stage C0 is the address generation cycle (address = base register + index register + 20-bit signed displacement).

Table 1. Pipeline stages related to the FXU

DcD	C0	C1 (E-1)	C2 (E0)	E1	PA

Cycles C1 and C2 are the memory cache access cycle and E1 stage is the execution stage. PA stage is the put-away stage when the GPR and cache write buffer are updated. The execution stage of z990 processor is placed a cycle after the data cache return. This was necessary since the z990 processor executes register-Memory and memory-memory instructions where one or both of the operands are from storage, and thus the instruction cannot be executed until the data is returned from the cache. The instruction dispatch occurs on the C1 (or E-1) stage which coincides with the D-cache access. During the E-1 cycle, the FXU controls receives the instruction text and any previously decoded bits and starts its own decoding to set the controls of the dataflow. During the E0 stage the operands are read from the GPR or they are returned from storage and all data dependencies are resolved. During cycle E1, instructions are executed and the condition code is set. Also, branch resolution, and the final exception determination and exception type are decided in E1 stage. An instruction stays in E1 stage as many cycles as is needed to finish its execution. Also, in the case of a D1-cache or translation miss, an instruction is held in the E1 stage until its data becomes available. If the instruction with a cache miss is the oldest instruction in a group, all instructions in that group are held in E1 stage. However, if the instruction with a miss is not the oldest, older instructions are allowed to complete and update architected facilities. Each one of stages E-1, E0, E1 and PA can be held without the need of instruction recycle or re-dispatch in order to achieve the best CPI. The FXU dataflow and control wraps the FXU results directly into the next executing group of instructions without any stalls or stages.

3. FXU DATAFLOW

The FXU consists of 4 execution pipes labeled R, X, Y and Z, where pipe R is a control pipe used for executing conditional branches. The dataflow stack, as shown in Figure 1, consists of three execution pipes (X, Y and Z), a General Purpose Register (GPR 32x64 bits), an Address Register File (AR 32x32 bits), and an operand buffer to hold data cache returns. The GPR has four 64-bits read ports and two 64-bits write ports. The GPR updates occur on either a word boundary (32-bits) or double word boundary (64-bits). A write port can be used to update bits 0:31, bits 32:63, or bits 0:63 of the same GR. The same write port can also be used to update a pair of even-odd register with write port bits 0:31 updating even-GR bits 32:63 and write port bits 32:63 updating odd-GR bits 32:63. The special write ports for the GPR are used to execute many architected instructions that operate on 32-bits of a GR, 64-bits or a GR, or on 32-bits of even-odd GR pair. The FXU also has 2 64-bits reads from the cache allowing two Register-Memory operations or single Memory-Memory instruction to be executed in any specific cycle. Fetches from the cache can be made ahead of execution and the data returns are saved in the operand buffer until they are used by the instructions. The FXU dataflow also receives two inputs from the address generation in the dispatch unit (Addr. Gen. Outputs). Address generation is used for memory address calculation, load address calculations and branch target address calculations. Finally, the FXU has one 64-bit read port from architected control registers (RUbus). Figure 2 shows a layout for the entire FXU.

Pipes X and Y, which are almost identical, are used to execute most of the superscalar single cycle instructions. Each of these pipes consist of a 64-bit binary adder/subtractor (Bin) with a built in byte sign extension, a 64-bit rotator (Rot), a 64-bit bit-logical-and-insert-under-mask (Blu) and a 64-bit mask generator used for shifts and merge instructions. The binary adder performs addition, subtraction as well as byte-size sign extension. For example, a 16 or 32-bits signed number can be added/subtracted to/from a 64-bits signed number in just a single execution cycle. Pipe Z consists of a decimal arithmetic support unit, 16x64 bit binary multiplier and other miscellaneous logic.

4. FXU EXECUTION

The current FXU is the first superscalar design for the new CMOS generations of z990 processor, offers better performance

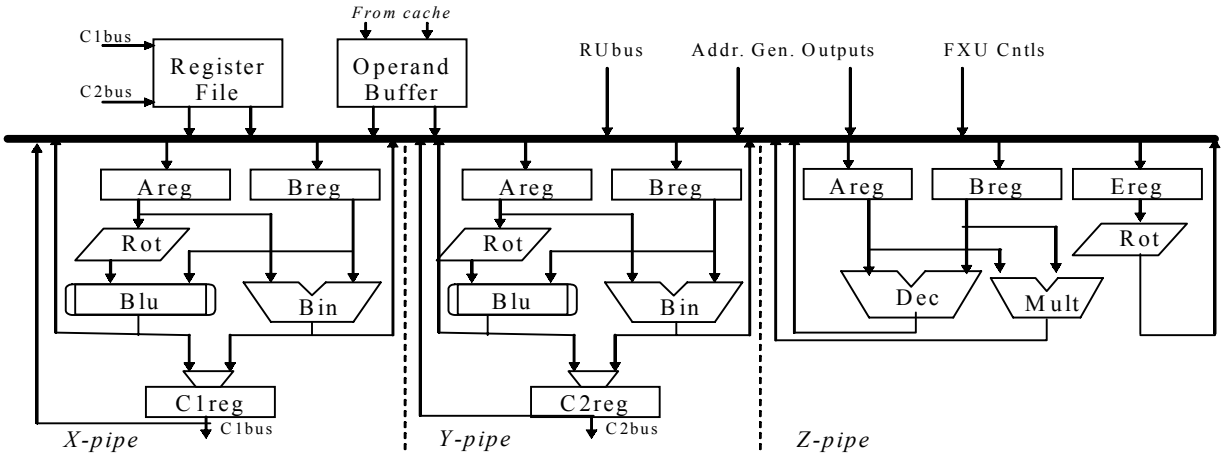

Figure 1. Block Diagram for the FXU dataflow stack.

for decimal arithmetic, better performance for various critical complex instructions with execution algorithms that target multi-execution pipes, includes capabilities to alter execution behavior to alleviate logic bugs during processor testing in the lab as well as problems that may be encountered at the customer sites.

The FXU executes a total of 369 instructions. The instruction set the FXU executes include load/stores, decimal arithmetic including multiply and divide, binary arithmetic, shifts and rotates, standard and multi-register loads/stores, branches, complex memory addressing, long moves, logical operations, convert instructions, and other special instructions necessary to support hardware virtualization and tightly coupled parallel processing. Most of the instructions executed in the FXU require one cycle of execution.

The FXU design allows flexibility on grouping and dispatching of multi-cycle and/or complex instructions. It allows a multi-cycle execution instruction to be dispatched to one execution port but executed in multiple execution pipes without cracking the instruction and without limiting it to a single execution pipe. The dispatch unit dispatches the instruction to one of the FXU pipes, but replicates the instruction text information such as opcode to all of the execution pipes in parallel. Few control signals are generated to control which pipes will be used for execution of the instruction. The FXU then decides how the instruction is to be executed in the available FXU pipes. This provides the FXU with ability to utilize all the hardware resources, in all of the FXU execution pipes, for the execution of this instruction. This method results in optimum performance and little or no complication to the exception logic, error detection or recovery logic. It also places the flexibility of how these instructions will be executed in the FXU, where the actual execution takes place, instead of in the instruction dispatch unit (or compiler in the case of VLIW).

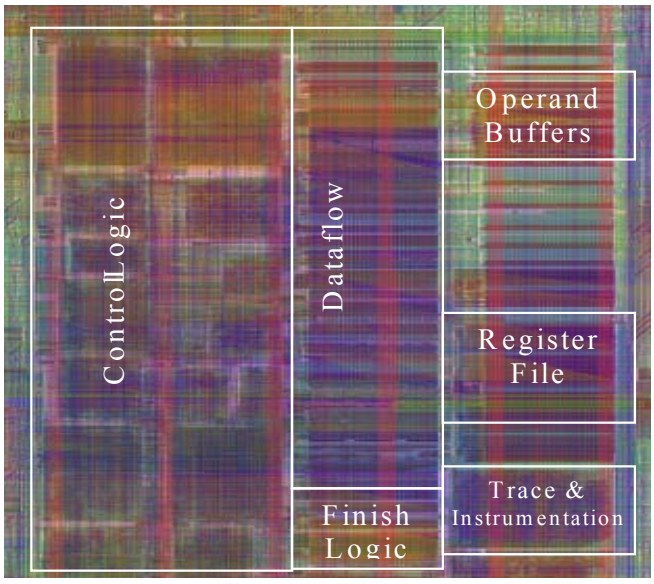

Figure 2. Layout for the entire FXU.

As mentioned earlier, the FXU has many capabilities that enhance the overall performance of the processor CPI. Operand forwarding and result forwarding, for example, allow the grouping and dispatching of two dependent instructions together by forwarding the operand or result of the older instruction executing on pipe X to the younger instruction executing on pipe Y. The following are examples of the description and implementation of operand forwarding and result forwarding.

Example 1: Cache data forwarding: This type of forwarding is used when the older instruction is a Load instruction to a GR-R1 and the GR-R1 is needed by younger instruction.

Instr. A (X-pipe) : Load R1, R3 (R1 ← cache data)
Instr. B (Y-pipe) : Add R1, R2 (R1 ← R1 + R2)

In this case, the instruction B gets the value of GR-R1 from the cache return data for instruction A instead of the GPR read data port. The dataflow of the FXU is modified so that the cache return for X-pipe is also an input to Y-pipe registers.

Example 2: GR data forwarding: This type of forwarding is used when the older instruction is a Load register instruction to a GR-R1 and the GR-R1 is needed in younger instruction as follows:

Instr. A (X-pipe) : Load R1, R3 (R1 ← R3)
Instr. B (Y-pipe) : Add R1, R2 (R1 ← R1 + R2)

Instruction B is dependent on Instruction A. To solve dependency and group instructions together, the GR-read port for Instruction B is set to R3 instead of R1.

Example 3: condition code forwarding:

Instr. A (Y-pipe) : Add R1, R2 (R1 ← R1 + R2, sets CC)
Instr. B (Y-pipe) : LTR R1, R1 (R1 ← R1, sets CC)

Instructions A and B are dependent but are grouped together. To solve the dependency, instruction A is allowed to update the GR while instruction B update to the GR is suppressed. However, instruction B uses the result of instruction A to update the condition code.

4.1 Decimal Assist Unit

The decimal unit has many enhancements over the last generation unit [6]. The current decimal unit includes a 16 digit BCD adder/subtractor with sign insertion, conversion tables to convert between binary and BCD arithmetic formats and other support logic to perform packing and unpacking between BCD, ASCII and Unicode formats. The current decimal unit can execute the add-instruction, subtract-instruction or compare-instruction in just 3 cycles. It is also capable of performing decimal division and multiplication with variable size operands (size = 1 to 15 bytes). For decimal multiplication, leading zero count is performed on both operands, and the one with least number of non-zero digits is iterated upon. After initialization (4-6 cycles), each iteration in multiplication algorithm requires an average of 1.8 cycles The number of cycles required for the decimal division is dependent on the number of significant quotient digits with an average of 4.4 cycles for each quotient digit.

4.2 Binary Multiplication

The current FXU has a binary multiplier than be configured to either perform a 16x32 signed multiplication or a 16x64 unsigned multiplication. The multiplier is pipelined with two stages. During the 1st stage, all partial products are generated and added to form two final products, and in the second stage the two products are added together to form the final result. The output of the multiplier feeds the output register and also wraps back to the input registers of pipes X, Y and Z. The wrapping paths are added to perform a quick and efficient implementation of long multiples. Table 2 shows the typical execution cycles for various multiply

instructions. A leading zero count is performed on both operands for long multiplies to potentially shorten the execution cycles.

Table 2. Execution cycles for various multiply instructions.

Instruction	Result ← operand size	cycles
MH, MHI,	32 ←16 x 32	2
MGHI	64 ←16 x 64	2
ML, MLR	64 ←32 x 32 (logical)	4
M, MR	64 ←32 x 32 (signed)	4
MLG, MLGR	128 ←64 x 64 (logical)	17

5. FXU CONTROL STACK

There are 12 unique custom macros and 19 control macros in a design area dimension of 8.6 mm by 11.5 mm. A summary of the control macros are shown in Table 3. For typical control macros, the number of latches per unit area or area per k-cells is supposed to be the same. For decode macros, there tend to be more logic work (k-cells) done per latch. These numbers will be used to size future generation of the FXU and to perform power estimates based on area and latch count. The control is decomposed into 19 interacting macros (or state machines). The decomposition minimizes the overall macro crossing, eliminates macro crossing for timing critical logic cones, reduces the overall area, and allows multiple designers to work simultaneously on implementing different instructions. Control macros are not logically stable until very late in the design phase since additional functions and logic fixes are expected even days before sending the processor for manufacturing. A substantial effort is placed on structuring the control and altering the micro-architecture of some FXU states to meet cycle time.

Table 3. Statistic on FXU control macros.

Number of macros	19
Total number of latches	3984
Total number of k-cells	488k
Area per latch (mm^2 per 1000 latch)	11.3
k-cells per latch	0.15

6. FXU VERIFICATION

Verification of the FXU in the z990 microprocessor proceeds at four basic levels defined by the breadth of logic being tested. The lowest level, designer macro verification, contains a single designer's hardware description language (in VHDL). The usage of formal verification is incorporated in the macro level verification [7]. The second level is the unit-level verification, where the entire unit (many dataflow and control macros) is verified. The third level of verification is done at the microprocessor level. A strong architectural-level instruction stream test-case generator, AVPGEN is used for the unit level as well as for the CP level [8]. Finally, system level verification is performed on symmetric multiprocessor (SMP) configurations. A substantial time and effort is placed on unit-level simulation development and usage.

7. CONCLUSION

This paper presents the design challenges for the FXU in the z990 enterprise microprocessor. The FXU dataflow supports executing two Register-memory instructions and a branch in a single cycle or a single memory-memory instruction with variable operand length. The FXU is also capable of executing decimal arithmetic instructions including multiplies and divides as well as binary multiples. The FXU in z990 processor improved the performance of decimal arithmetic, binary multiplies and other critical instructions. The FXU design includes advanced features such as operand forwarding that allows dependent instructions to be grouped and dispatched simultaneously. Finally, verification of the FXU is quite challenging that required multiple levels of verification by using random-based and formal verification.

8. ACKNOWLEDGMENTS

Our thanks are to the rest of the z990 microprocessor design and management teams.

9. REFERENCES

[1] Glenn Hinton et. al, "A 0.18-um CMOS IA-32 Processor With a 4-GHz Integer Execution Unit," IEEE Journal of Solid State Circuits, Vol. 36, No. 11, Nov. 2001.

[2] T. J. Slegel, E. Pfeffer, J. A. Magee, "The IBM eServer z990 microprocessor," IBM Journal of Research and Development, vol. 48, no. 3/4 (accepted for publication in the May/July 2004 issue)

[3] T. McPherson and et al. "760 MHz G6 S/390 Microprocessor Exploiting Multiple Vt and Copper Interconnects", Solid-State Circuits Conference, Feb. 2000.

[4] G. Northrop and et. Al "600 MHz G5 S/390 Microprocessor", 1999 International Solid-State Circuits Conference, Feb. 1999, pp. 88-89.

[5] Timothy . Slegel and et al. "IBM S/390 G5 Microprocessor", 1998 Hot Chips Symposium, Stanford. Aug., 1998.

[6] F. Busaba, et. al, "The IBM z900 Decimal Arithmetic Unit, " 35th Asilomar Conference on Signals, Systems and Computers, Nov. 2001.

[7] F. Busaba, and et. al, "Designer-Level Logic Verification Using RuleBase, " 4th International Workshop of Testing Embedded Core-based System-Chips, Montreal, May 2000.

[8] A Chandra, V. Avenger, D.. Gist and Y. Wolfs Hal, "AVPGEN-A Test Generator for Architecture Verification," IEEE Trans. Very Large Scale Integration (VLSI) sys. Vol. 3, No. 2, pp. 188-200, June 1985.