

Design and microarchitecture of the IBM System z10 microprocessor

C.-L. K. Shum
F. Busaba
S. Dao-Trong
G. Gerwig
C. Jacobi
T. Koehler
E. Pfeffer
B. R. Prasky
J. G. Rell
A. Tsai

The IBM System z10™ microprocessor is currently the fastest running 64-bit CISC (complex instruction set computer) microprocessor. This microprocessor operates at 4.4 GHz and provides up to two times performance improvement compared with its predecessor, the System z9® microprocessor. In addition to its ultrahigh-frequency pipeline, the z10™ microprocessor offers such performance enhancements as a sophisticated branch-prediction structure, a large second-level private cache, a data-prefetch engine, and a hardwired decimal floating-point arithmetic unit. The z10 microprocessor also implements new architectural features that allow better software optimization across compiled applications. These features include new instructions that help shorten the code path lengths and new facilities for software-directed cache management and the use of 1-MB virtual pages. The innovative microarchitecture of the z10 microprocessor and notable differences from its predecessors and the IBM POWER6™ microprocessor are discussed.

Introduction

IBM introduced the System z10* Enterprise Class (z10 EC*) mainframes in early 2008. A key component of the system is the new z10* processor core [1]. It is hosted in a quad-core central processor (CP) chip, which together with an accompanying system controller (SC) chip (which includes both the L2 cache and storage controller functions) makes up the microprocessor subsystem. Both CP and SC chips are manufactured in IBM CMOS (complementary metal-oxide semiconductor) 65-nm SOI (silicon-on-insulator) technology. The resulting CP chip has a die size of 454 mm² and contains 993 million transistors.

The CP chip die is shown in **Figure 1**. Each core, shown in purple, is accompanied by a private 3-MB level 2 cache (L1.5), shown in gold. In addition, two coprocessors (COPs), each shared by a pair of cores, are shown in green. The I/O controller (GX) and memory controllers (MCs), shown in blue, an on-chip communication fabric responsible for maintaining interfaces between on-chip and off-chip components, and the external SC chip make up the cache subsystem. The cache subsystem mainly provides the symmetric multiprocessing (SMP) fabric and

interprocessor memory coherency. For a detailed description of the cache subsystem, see Mak et al. [2], in this issue.

A key z10 development goal was to provide performance improvements on a variety of workloads compared to its System z9* predecessor platform. Many of the performance gains achieved by prior System z* microprocessors were derived from technology improvements in both circuit density and speed. However, although the 65-nm technology provides a great improvement in circuit density, gate and wire delays no longer speed up proportionately to the density gain and thus cannot provide as much of a performance gain as was previously possible. An innovative microarchitecture was needed to meet our goal of a significant performance improvement.

Traditional mainframe workloads are characterized by large cache footprints, tight dependencies, and a sizable number of indirect branches. Newer workloads are typically processor centric, have relatively smaller cache footprints, and benefit greatly by raw processing or execution speed. Using the Large Systems Performance Reference (LSPR) workloads as a traditional workload

©Copyright 2009 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied by any means or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

0018-8646/09/\$5.00 © 2009 IBM

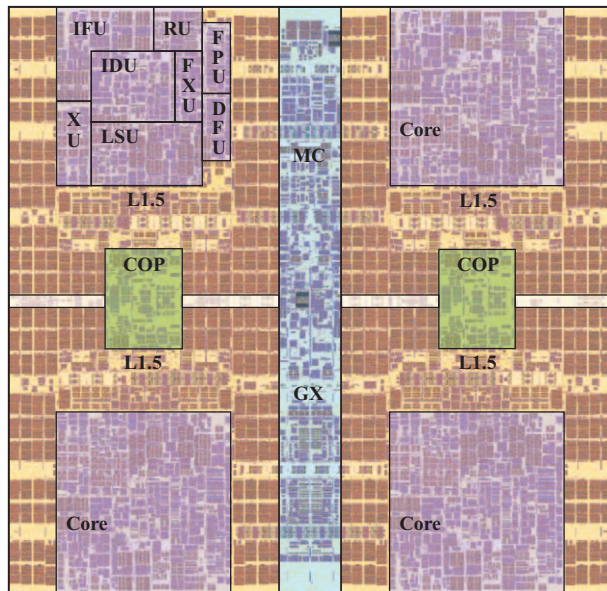


Figure 1

The z10 CP chip. (DFU: decimal floating-point unit; FPU: floating-point unit; FXU: fixed-point unit; IDU: instruction decode unit; IFU: instruction fetch unit; L1.5: the low-latency second-level private cache; LSU: load/store unit; RU: recovery unit; XU: address translation unit.) (A version of this figure appeared in [1], ©2008 IEEE, reprinted with permission.)

reference [3], the performance target of the z10 platform was set at about a 50% improvement over the z9* platform. At the same time, a separate goal of two times improvement was planned for processor-centric workloads.

After studying the impacts of various microarchitecture options and their effects on workload performance, area, power, chip size, system hierarchy, and time to market, an in-order high-frequency pipeline stood out to be the best microprocessor option that would satisfy the z10 goals. Since the introduction of the IBM s/390* G4 CMOS processor [4], the mainframe microprocessor pipeline had always been in order. Many insights leading to dramatic improvements could be gained by innovating upon the proven z9 microprocessor. The z10 core cycle-time target was eventually set to 15 FO4,¹ as explained in the next section.

Other microarchitecture options were also studied. For example, an out-of-order execution pipeline would give a good performance gain, but not enough to make a two times performance improvement. Besides, such a design would require a significant microarchitecture overhaul and an increase in logic content to support the inherently

¹FO4, or fanout of 4, is the delay of an inverter driving four equivalent loads. It is used as a measure of logic implemented in a cycle independent of technology.

rich and complex IBM z/Architecture* [5]. Also investigated was the possibility of three on-chip simultaneous multithreaded cores with each supporting two threads. The circuit area and performance throughput per chip would roughly equal that of a single-threaded quad-core design. However, single-threaded performance would be jeopardized. Although these options were not as effective for the z10 design, they remain possible enhancements for future mainframes.

In addition to an ultrahigh-frequency pipeline, many CPI (cycles per instruction) enhancements were also incorporated into the z10 microprocessor. For example, a state-of-the-art branch-prediction structure, a hardware data-prefetch engine, a selective overlapping execution design, and a large low-latency second-level private cache (L1.5) are some of the more prominent features. A total of more than 50 new instructions are provided to improve compiled code efficiency. These new instructions include storage immediate operations, operations on storage relative to the instruction address (IA), combined rotate and logical bit operations, cache management instructions, and new compare functionalities.

The z10 core ultimately runs at 4.4 GHz, compared with the 1.7-GHz operating frequency of the z9 microprocessor. The z10 core, together with a novel cache and multiprocessor subsystem, provides an average of about 50% improvement in performance over the z9 system for the same *n*-way comparison. For processor-intensive workloads, the z10 core performs up to 100% better than the z9 core. Further performance improvements are also expected with new releases of software and compilers that utilize the new architectural features and are optimized for the new pipeline.

The remainder of this paper describes the main pipeline and features of the z10 core and its support units. The motivations and the analyses that led to the design choices are discussed, and comparisons are made with the z9 and IBM POWER6* [6] processors.

Decision for cycle-time target

The success of an in-order pipeline depends on its ability to minimize the performance penalties across instruction processing dependencies. These dependencies, which often show up as pipeline stalls, include fixed-point result to fixed-point usage (F–F), fixed-point result to storage access address generation for data cache (D-cache) load (F–L), D-cache load to fixed-point usage (L–F), and D-cache load to storage access address generation (L–L). The stalls resulting from address generation (AGen) dependencies are often referred to as *address generation interlocks* (AGIs).

In order to keep the execution dependencies (F–F and F–L) to a minimum, designs that allow the bypass of fixed-point results and go immediately back to

subsequent dependent fixed-point operations and AGen were analyzed. The best possible solution was found to be a 14-FO4 to 15-FO4 design, which was adopted for the z10 microprocessor. These physical cross-sections were validated against the POWER6 core, which was also designed with similar frequency and pipeline goals. The implementation of extra mainframe functionalities in the z10 microprocessor, which required extra circuits and thus more circuit loads and wire distances between various components, added roughly 1–2 FO4 of extra delay compared with the similar result bypass on the POWER6 microprocessor.

Since register-storage and storage-storage instructions² occur frequently in System z mainframe programs, a pipeline flow similar to the z9 microprocessor (which was the same as the z990 microprocessor [7]) was used such that the D-cache load fed directly into any dependent fixed-point operation, reducing the load-to-execute latency to zero.

Options were analyzed by varying the D-cache sizes and pipeline cycles required to resolve L–L latency (often encountered during pointer-chasing code). Eventually, a 128-KB D-cache design that led to a latency of four cycles was selected. This design fitted perfectly into a cycle time of 15 FO4. This latency was the same in the POWER6 microprocessor, but the extra FO4s in the z10 microprocessor allowed a double-sized L1 D-cache.

The z9 microprocessor design was also a good base for some high-level validations. For example, the decoding circuits for the vast number (~1,000) of instructions defined in both z/Architecture and millicode [8] hardware assists took about 28 FO4 in the z9 microprocessor. Although some of the decoding functions can be deferred, steering information required in instruction grouping and issue must still be extracted (with the minimum number of stages). A cycle time of 15 FO4 allowed decoding to be done in two cycles.

On the other hand, some z9 core units would not provide a proportional performance impact if they were redesigned into 15 FO4. The address translation unit (XU), which also includes a level 2 translation lookaside buffer (TLB), and the COP are such units. Their main design criteria are latencies (for TLB misses) and functionalities (cryptography and data compression), respectively. These two units were therefore slated to run at one half of the core frequency using mostly high V_t devices,³ requiring each to be a 26-FO4 design. Because of the small FO4 difference between the z9 and z10 design

²A *register-storage* instruction generally specifies a register as an operand source as well as the result destination, while a storage location is specified as the other operand source. A *storage-storage* instruction specifies storage as both operand source and result destination.

³High V_t devices have a higher threshold voltage such that the leakage current is reduced, but at the same time, performance is about 20% lower than nominal threshold devices.

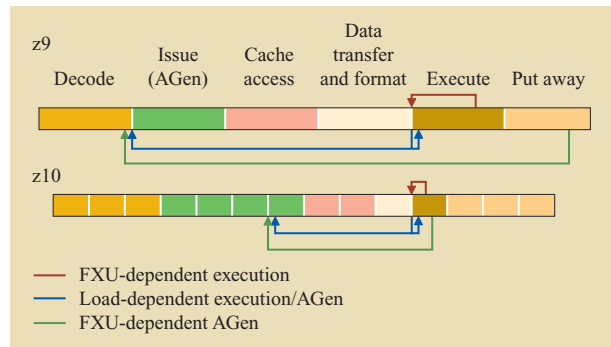


Figure 2

The z9 and z10 microprocessor pipelines compared with respect to cycle time. (AGen: address generation; FXU: fixed-point unit.)

requirements, these units were able to reuse much of the z9 design.

The 15-FO4 target was maintained throughout the design of the z10 microprocessor, which was 4.4 GHz at the time of the initial shipment. The frequency difference compared with the POWER6 4.7-GHz microprocessor is not proportional to the 2-FO4 difference, primarily because of differences in their operating conditions and circuit limited yield requirements.

Figure 2 shows a comparison between the z9 and z10 microprocessor pipelines with respect to their operating frequencies. The real-time latencies for critical dependencies, which were drastically reduced in time, are shown as arrows.

Design for power efficiency

The z10 core is designed to give the highest performance achievable within the power and heat dissipation constraints defined by the power and thermal system. The System z10 server includes a high-performance cooling system [9] that allows the processor core to run at about 55°C. Leakage current is therefore reduced in comparison with air-cooled systems that tend to run at higher temperatures. Such a leakage power advantage allows the z10 core more circuitries to achieve higher performance. To manage the overall power consumption and to keep a bound on the maximum sustainable power, power budgets across various design components were initially estimated and then validated throughout implementation.

Clock-gating efficiency targets were established for each unit inside the z10 core. This is the first time fine-grained clock gating (which turns off clocks to unused functional blocks to reduce dynamic power) was implemented throughout a System z microprocessor. The in-order design of the z10 core provided abundant opportunities and allowed straightforward implementation

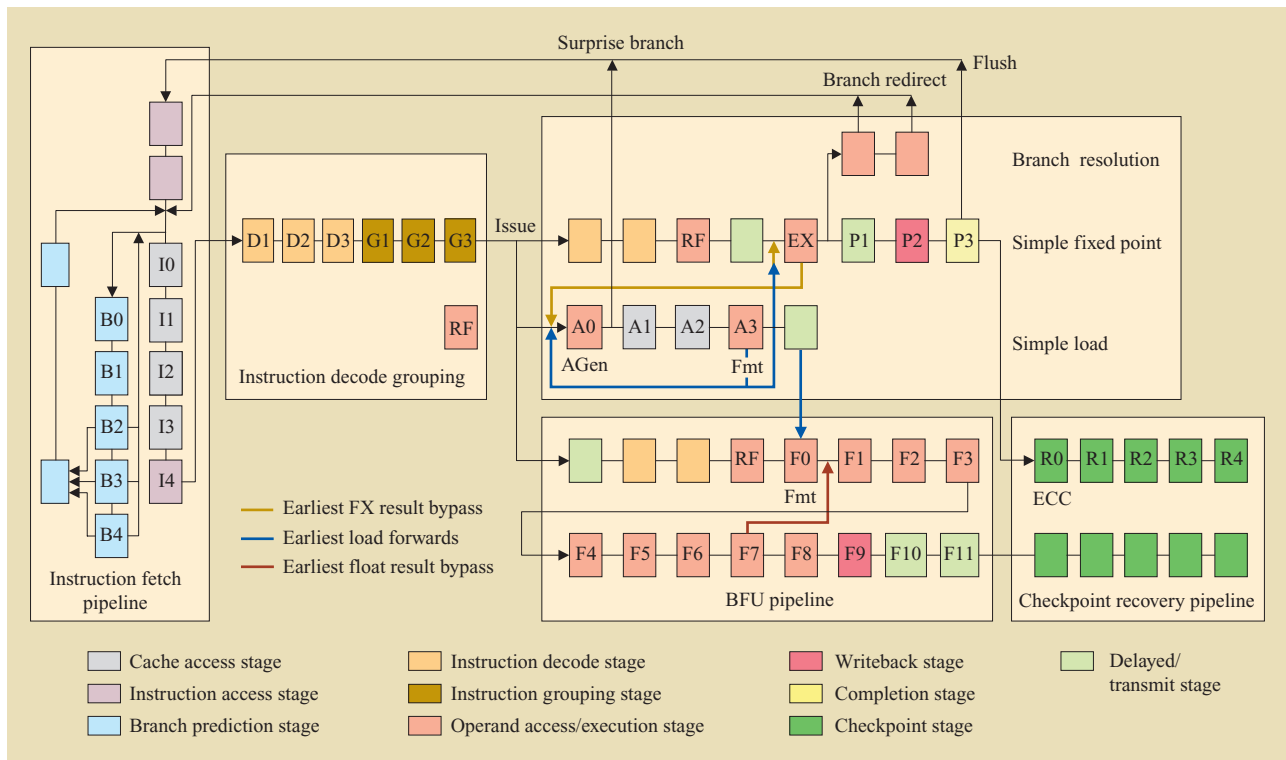


Figure 3

The z10 microprocessor pipeline. (AGen: address generation; BFU: binary floating-point unit; ECC: error-correcting code; EX: execution; Fmt: format; RF: general-purpose register [GPR] file/floating-point register [FPR] file access.)

to maximize clock-gating efficiency. Verification-based tools helped tabulate the amount of clock gating achieved using pseudo-idle simulation runs. For example, about 50% of all latches are not clocked when a processor is waiting for a long cache miss.

By measuring the power consumed by either enabling or disabling clock gating in all cores and coprocessors in a quad-core chip, an average of about 20% difference in active power was observed across various workloads.

The use of devices of various threshold voltages (V_t) was also budgeted. Devices with lower V_t consume more leakage current but perform faster. While the core (except for the XU) was designed mainly with nominal V_t devices (with a small amount of low V_t devices used in some frequency-critical paths), the rest of the chip was populated by high V_t devices to reduce leakage power. Toward the end of the design phase, noncritical circuits inside the core were selectively replaced with high V_t devices to further reduce core leakage power.

In addition, latches were designed to operate in pulsed mode, which involves holding the master clock on while the slave clock is pulsed. The switching power of the master clock signals is reduced, while operating frequency

can be improved. Such an improvement results from the fact that signals that are too slow in certain pipeline stages can now steal time from the immediate next pipeline stages. However, pulsed-mode design can sometimes become quite challenging because a relatively large early mode padding is required because of the approximately 40-ps pulse width. The decrease in power consumed in pulsed mode is about 25%, as measured in actual chips.

Microprocessor pipeline

The main processor pipeline is shown in **Figure 3**. The pipeline is largely separated into instruction fetching, instruction decoding and issuing, storage access through data cache, execution including fixed-point and floating-point operations, and results checkpointing. (The alphanumeric labels referring to the pipeline stages are referenced throughout this paper.)

Instruction fetching and branch prediction

The instruction fetch unit (IFU) is designed to deliver instructions far ahead of processor execution along either the sequential or the predicted branch path. The IFU also

redirects the instruction fetch addresses during pipeline flushes, millicode execution entries and exits, and interrupts.

The IFU incorporates a 4-way 64-KB instruction cache (I-cache) and a 2-way 128-entry TLB. During instruction fetch, an instruction fetch request is first made from the instruction fetching engine (I0). The following two cycles (I1 and I2) are used for the I-cache pipelined access. The set select for each access to the I-cache is obtained through a 4K-entry set prediction array indexed with a portion of the logical IA. Although a set prediction miss would cause an extra delay of four cycles, it is highly accurate and more beneficial than having to wait for the TLB and directory lookup results. Although the I-cache size is reduced from 256 KB in the z9 system, the CPI delta is small and, to a large extent, mitigated by the z10 enhanced branch-prediction structure and by the low-latency L1.5.

Cache data fetched is then processed through the instruction buffer to extract the instruction texts (I3 and I4). The instruction buffer is made up of three super basic block buffers (SBBB). Each SBBB retains up to four quadwords (QWs) of instruction text from possibly two different cache lines. A predicted taken branch will be marked within an SBBB while another SBBB is set up to handle the branch target stream. The z10 I-cache contains no predecode or grouping information, unlike some other popular processor designs. To support variable instruction lengths, and to efficiently support the strong storage consistency (one that allows self-modifying code) in the z/Architecture, the mechanisms to maintain extra information in the I-cache could become extremely complicated. Instead, the SBBB incorporates dataflow parsing of instruction boundaries based on the instruction lengths, so that a steady stream of two instructions can be delivered to the instruction decode (and issue) unit (IDU).

Upon any surprise branches, the z9 IFU would wait for the branch address calculated from the downstream AGen adders before it could redirect its fetching patterns. In the z10 system, the IFU can react to a new target stream once any surprise relative branches are decoded by either the SBBBs or the IDU and thus speeds up the branch target redirection.

As the core pipeline becomes longer, any branch redirect becomes more costly in terms of processor cycles. Therefore, a highly aggressive z10 branch-prediction design is used to improve both branch-prediction accuracy and timeliness.

A 10K-entry 5-way set-associative branch target buffer (BTB) is the nucleus of such a design. Each BTB entry contains the branch address, target, type, and an eight-state branch history used to predict the branch direction (either taken or not taken). A novel design allows three

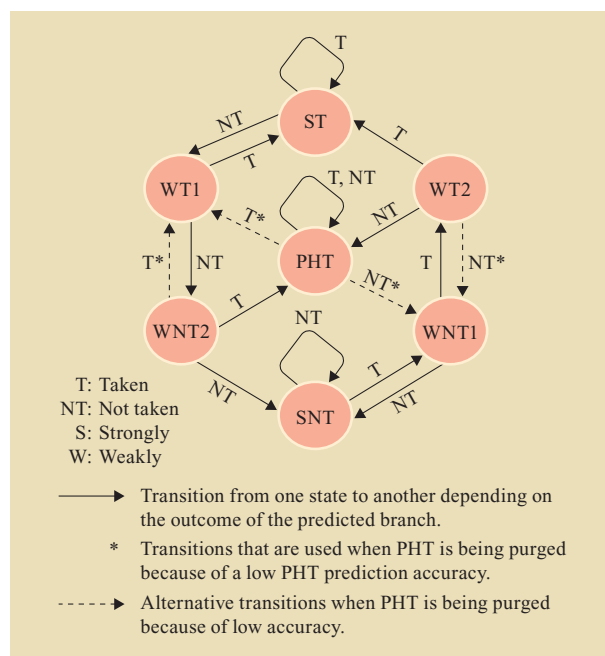


Figure 4

Branch history table (BHT) states. (PHT: pattern history table.)

1-MB target offsets and two 1-GB target offsets intermixed among the five sets in a BTB row.

Predictions can be made once every four cycles. One cycle is required to index the two-cycle pipelined BTB. The fourth cycle determines which set of the array, if any, contains the desired prediction. By presuming the common case that the branch of interest is in the most recently used column, the hit-detection logic can increase the throughput to a prediction every third cycle for such cases.

To further increase prediction throughput, a branch target queue (TQ) is used to store four recently taken branches. This small TQ requires a single cycle to locate a possibly matching branch. The TQ can, therefore, increase branch-prediction throughput to a prediction every other cycle, providing a throughput capability similar to that of the z9 microprocessor.

To improve direction prediction accuracy, a 512-entry filtered pattern history table (PHT) has been added for branches that exhibit poor accuracies from predictions made only from the branch history table (BHT). Upon multiple mispredictions, as shown in Figure 4, the BHT state enters a PHT override state. The PHT, thus, contains predictions that are filtered down from the BHT on the basis of the BHT states tracked.

To improve target prediction accuracy mainly involving branches with multiple targets, a multiple target branch target buffer (MTBTB) is also added. It is a 2K-entry

array that is indexed in parallel to the BTB using a hashed function of taken branch history. Such indexing allows for multiple branch targets to be tracked for a single branch. Only upon the initial detection of a wrong target on a BTB branch can the MTBTB be used.

The branch-prediction logic runs fairly asynchronously to instruction fetching. Branch searching and isolation occurs at a rate of 32 sequential bytes per cycle, regardless of the instruction fetch bandwidth. A predicted branch target is sent both to the I-cache for data and back to the BTB for the next branch search. The various branch search latencies are shown in Figure 3 as the (B0–B4) stages. Through this searching method, the target of branches can be fetched without actual fetching or decoding of those branches. With the BTB having a larger branch footprint than that of the I-cache, the BTB performs instruction prefetching using the predicted branch targets.

Instruction decode/issue

Once valid instructions pass from the IFU to the IDU, they enter into the two-wide superscalar pipeline.

As shown in Figure 2, z9 decode and issue take only two pipeline stages, while the z10 decode and issue take considerably more stages. Since no predecode information is stored inside the I-cache, two pipeline cycles (D1 and D2) are required to decode the full instruction set needed for instruction grouping and steering. This and other control information, such as storage access controls and branch handling, are then fed into an eight-deep instruction queue (IQ) and address queue (AQ).

From the IQ/AQ, two instructions are brought into the grouping logic (D3). Three grouping stages (G1, G2, and G3) are needed to schedule up to two instructions into an execution group and to create a pipeline hazard-handling scheme that scales well with high frequency. This scheme assumes instruction execution to be done within a fixed number of cycles after issue and then involves rejecting and reissuing instructions upon the detection of any pipeline hazard (or reject) conditions. All stall conditions are, by design, evaluated in the first two stages (G1 and G2) to determine whether the group can be issued. These stall conditions primarily include address generation interlocks (AGIs), floating-point register dependencies, multicycle execution waits, or post-pipeline reject waits. If no stall condition is detected, instructions in a group will be issued (G3); otherwise, they will be held.

In comparison, the z9 pipeline is an execution-stalling design. To handle any possible pipeline hazards, these conditions have to be known early enough to block the execution of the affected instruction until these conditions are resolved. As any execution is blocked, subsequent instructions are also stalled from being issued. This design

requires global stalling controls that are very difficult to fit into a short cycle time. By moving the z10 stalling controls to preissue stages and reacting to any hazards by reissuing instructions, frequency criticalities of post-issue flow controls are reduced, allowing a design that scales well with high frequency.

A minimal penalty of nine cycles is incurred for any rejected instruction. With the most common pipeline rejecting conditions being TLB and cache misses, an instruction being rejected will often be ready for reissue by the time these conditions are resolved. Copies of instructions are kept by the IDU in a reject queue such that rejected instructions (and any subsequently issued younger instructions) can be routed back to the G1 stage for reissue.

In the z9 microprocessor, only one branch instruction can be decoded per cycle, and branches have to be executed as the first instruction in a group. These restrictions are removed in the z10 microprocessor. A branch instruction can be decoded in either pipe, or two at the same time. Although only one branch can be executed per group, as in the z9 microprocessor, simple branches can be executed in any pipe. The more symmetric and less restrictive z10 design enables more efficient instruction grouping and increases pipeline utilization and thus performance.

Storage accesses

Any issued instruction that requires storage access is processed through one of the two superscalar pipes where the D-cache is first accessed and is then followed immediately by fixed-point execution. Although the z10 microprocessor is only 2-way superscalar, every pass through the pipeline can include both load and execute functions, allowing full advantage of the z/Architecture. For instructions that do not require any storage access, executions start in the same pipeline stage as if there are storage accesses. This consistent pipeline simplifies overall control and allows more precise bypass operations.

The load/store unit (LSU) includes an 8-way 128-KB D-cache and a 2-way 512-entry TLB, supporting two concurrent read accesses. Any storage access is handled first by calculating the access address through a 3-way (base + index + displacement) adder (A0). This address is then used to start the pipelined two-cycle D-cache access (A1) and the one-cycle directory and TLB accesses on the next cycle (A2).

The D-cache directory is virtually indexed, but absolute address tagged. If the cache access had to wait for the serial lookup hit results through the TLB and directory, the pipeline cycles added would be detrimental to performance. A set prediction array is, therefore, used to obtain an early predicted cache set by doing partial logical tag compares. This set predict scheme (similar to

that in the POWER6 microprocessor) is highly accurate and allows data to be returned without waiting for the full TLB and directory searches. If a misprediction is found afterward, a pipeline reject will be generated to allow the necessary correction and then a re-access.

Cache data is formatted and returned (A3) directly to the fixed-point unit (FXU) as source operands or to the AGen adders as potential bypassable inputs, keeping all of these latencies to a minimum.

In addition to a cache lookup pipeline that is as short as possible (while providing a cache as large as possible), ways to reduce cache-miss penalties are needed. In contrast to the z9 solution, in which the LSU can return data out of order to hide miss latencies, the z10 LSU always returns data in program order. However, the z10 reject pipeline allows the LSU to honor cache-miss requests from younger instructions that are already issued after a demand cache miss. By providing more cache-miss handling capabilities, cache misses can be overlapped to hide their penalties. The extra miss-handling capabilities include support for software and hardware prefetching.

Software cache management—Prefetch data
instructions are introduced so software can improve on performance by either directly prefetching a particular storage address with corresponding usage (read/store) into L1 or preemptively relinquishing its cache-line ownership when usage is over. The prefetching helps to hide cache-miss latencies by initiating a cache miss as early as possible. The relinquishment allows another processor to gain fast access to a line that was previously owned, reducing delays needed for multiprocessor coherency management, thus improving performance in an SMP environment. These instructions are implemented as nonblocking to avoid unnecessary pipeline rejects and delays. For more software optimization insights, see Reference [10].

Hardware prefetch engine—The LSU included an IA-based stride prefetching engine to prefetch data for repetitive storage-accessing patterns. The engine (Figure 5) is made up of a 32-entry stride history buffer. The buffer maintains a history of storage access addresses and types. The access addresses are the absolute addresses corresponding to any logical storage access addresses after dynamic address translation. These histories are used to detect strides (distances between memory access locations) among a history of previous accesses or strides. When a valid entry is read from the history buffer using part of the IA, its saved address is compared to the current storage access address to form an 11-bit stride. Stride detection using byte addresses is used because it enables more precise prefetches than line-address-based calculations.

If a stride is detected, the new address and stride will be written into the history buffer; otherwise, the new address

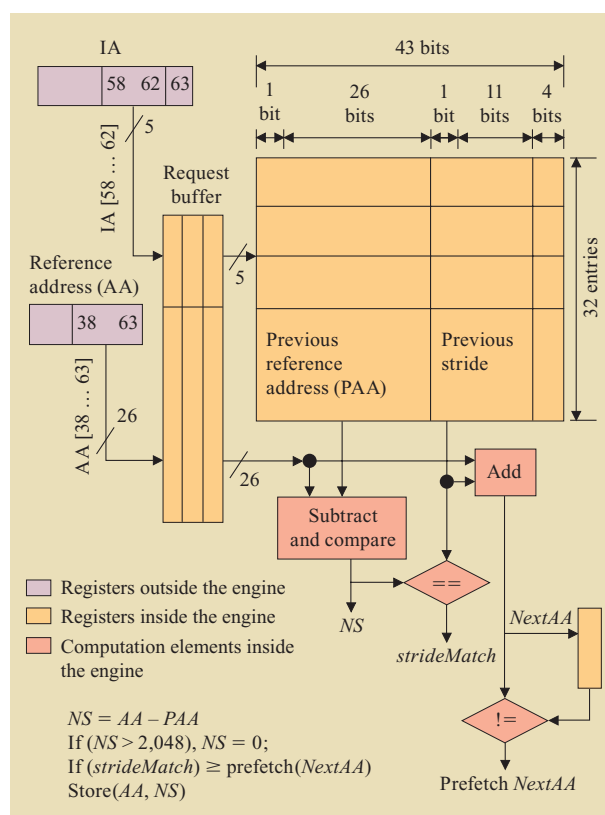


Figure 5

IA-based stride prefetch engine. (IA: instruction address; AA: absolute address; PAA: previous absolute address.)

and a stride of 0 will be recorded. When a new stride matches the previous stride, a single prefetch to the next strided absolute address (calculated by incrementing the current memory access address by the stride amount) will be attempted. If the strides are small, the engine will preemptively prefetch the next cache line.

This design is inherently less aggressive than other existing streaming prefetch designs that are more suited for technical computing workloads. The z10 design is tuned to benefit commercial workloads. A capability allowing millicode to initiate a 4-KB page prefetch is also provided to speed execution of very long instructions, such as `move character long (MVCL)`.

Instruction execution

Fixed-point unit execution

After performing instruction decodes, various dataflow preparation controls, register accesses, and potential result bypasses, the FXU starts execution immediately after the pipeline cycle of cache data return (EX). Most instructions can be performed in either of the two FXU

execution pipes in a single cycle; some less-frequently used instructions, such as control instructions, can be done only in the first pipe. For multicycle instructions, the FXU may spin in its execution state by wrapping partial results back into its input registers.

Results obtained are staged through two put-away cycles (P1 and P2) before they are written into the general-purpose register (GPR) file or the LSU store buffer. The put-away delays are added to allow time to suppress any premature GPR update from the second pipe in the case of a branch misprediction in the first pipe. If there are no reject conditions, interruptions, or branch misprediction, the instruction is considered to be complete once all the results are available (P3).

The FXU can execute dependent instructions back-to-back with full bypass capabilities. Techniques used to implement these superfast execution dataflows are described by Curran et al. [11]. Some additional System z functions, such as pack- or unpack-type instruction supports, are implemented outside the main dataflows.

Although millicode is used to implement complicated instructions, many simple multioperand and multiresult instructions remain implemented by hardware in the z10 microprocessor to optimize performance. To support the reject and reissue design, in which a long instruction can be rejected while its execution is only partially completed, state machines are incorporated into the IDU, LSU, and FXU that can synchronously restart from the same point within that instruction processing. For some cases, results are stored in either unused millicode GPRs or in special registers within control logic for a correct restart.

The z10 FXU can perform branch resolution at two different points within the pipeline. Normally, the branch resolution occurs two cycles after execution, but a branch mispredict indication can be sent to the IFU one pipeline cycle earlier if that branch has no immediate dependencies. These earlier cases include incorrect branch target prediction and incorrect branch direction predictions for branch-on-count instructions or branch-on-condition instructions in which the condition code is not updated in the immediately prior instruction within the same group.

A set of compare-and-branch instructions are added to facilitate common instruction sequences in which a branch may be taken depending on the result of testing a register. This two-instruction sequence can now be accomplished with one instruction and without affecting the condition code.

A set of compare-and-trap instructions are also provided to accelerate the checking of software error conditions, such as invalid pointers or out-of-bound array indices. These new instructions eliminate the prior use of branch instructions. In turn, the corresponding branch-prediction entries are now freed up in order to provide more prediction capacity.

Floating-point unit execution

Most instructions executed in the binary floating-point unit (BFU) can be operated in pipeline mode, while instructions executed in the decimal floating-point unit (DFU) are nonpipelined.

During BFU and DFU operations, instruction information and operand data (from either the GPR or cache) is staged through the FXU. The resulting condition code or arithmetic exceptions are also sent back to the FXU for consolidation (F9/P1 and F10/P2) before instruction completion (F11/P3).

BFU

The BFU performs binary (i.e., IEEE 754-compliant) and hexadecimal floating-point arithmetic and fixed-point integer multiplication and division.

All the main z10 dataflows are developed on a design common with the POWER6 microprocessor.

Because the dataflow is based on a common internal format, a cycle (F0) is needed to map from the various z/Architecture floating-point formats to this internal execution format. Similarly, a cycle (F8) is needed to reformat a result back into the architectural formats before the floating-point register (FPR) is updated. Although two extra cycles are added compared to the POWER6 microprocessor design, there is only a one-cycle impact for dependent instructions because results of the internal format can be bypassed without the first reformatting cycle. This is shown in **Figure 6**.

Extra dataflow and control sequences are also added to support IBM mainframe processor execution of extended precision arithmetic.

DFU

Decimal floating-point arithmetic (IEEE P754-compliant) was introduced in IBM z9 machines but was performed mainly with millicode emulation using special hardware assists. In the z10 core, all decimal floating-point instructions are implemented by the DFU hardware, providing better performance. Fixed-point decimal instructions are also implemented using the same dataflow. Schwarz et al. [12], in this issue, present additional detail.

Overlapped execution

In the mostly in-order z9 design, there are no execution overlaps between any floating-point and fixed-point instructions. Any fixed-point instruction will be held from issue until a prior floating-point instruction reaches its last BFU execution stage, and then any subsequent floating-point instruction will issue only after the fixed-point instruction is issued. Such a design is not efficient for floating-point programs that are characterized largely by having floating-point instructions bounded by fixed-

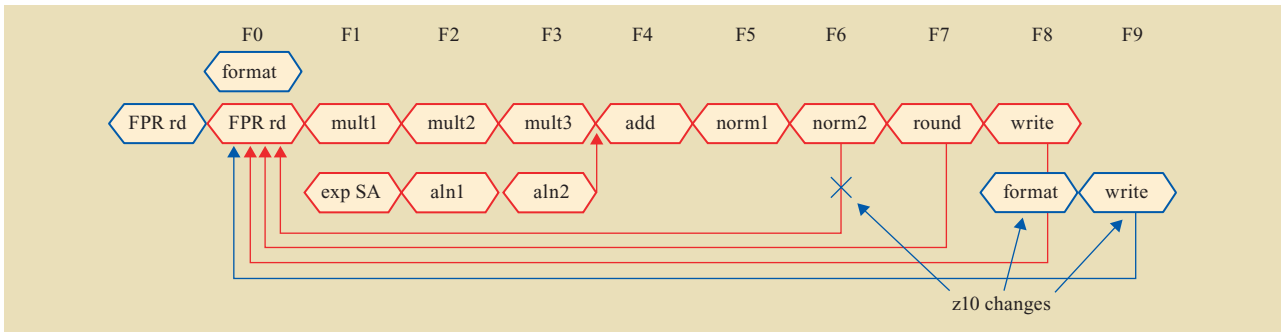


Figure 6

Differences between the POWER6 processor pipelines and the z10 BFU pipelines, with the POWER6 pipelines in red and the z10 differences in blue. (Aln: align; exp SA: exponent shift amount calculation; norm: normalize.)

point branch loops. To alleviate this in the z10 design, up to one group of FXU instructions can be executed in a pipelined fashion among a sequence of floating-point instructions, allowing them all to be fully pipelined. This allowed a simplified form of out-of-order or *overlapped* execution. The fixed-point instructions that can be overlapped are restricted to a small and performance-sensitive set (such as branch on index, branch on count, load address, or simple register loads).

Store data forwarding

Upon processing a store instruction, a store queue in the LSU is allocated by a storage access request similar to a load. Then, during store execution, the store data is sent to the LSU to be stored in its store buffer and to a special dataflow reserved for store data forwarding. This innovative design improves performance by forwarding buffered store data to subsequent loads at the same addresses to avoid the load-hit-store penalties that are common performance problems when store data needed by the load has not yet been written into the D-cache.

Instruction checkpointing and recovery

The z10 core continues to employ the same reliability features that are characteristic of a mainframe processor. Its design is vigorously checked throughout, and proven error-handling techniques that are largely software-transparent are still supported. To implement these error-handling techniques, an ECC (error-correcting code)-protected processor state, called *checkpointed state*, is backed up inside the recovery unit (RU).

The RU includes a checkpointing pipeline (R0 to R3) that starts right after any instruction completion (P3). ECC is generated on results data (R0) and is then written into a write queue (R1). Upon the absence of any error conditions (R2), that instruction and its results are checkpointed. Results in the write queue are then drained

into the ECC-protected checkpoint array (R3). Checkpointed architecture states are now ready (R4) for any recovery refresh or any state read usage.

For transient errors, the processor is reset and refreshed from the checkpointed state, and then execution resumes from the next noncheckpointed instruction. For cases in which the processor cannot make forward progress from an error, such as a stuck fault, the checkpointed state is loaded into another processor, and instruction processing resumes from there.

Unlike the z9 core, in which the CP chip (which includes two cores) is a sparing unit, each z10 core can undergo retry or transparent central processing unit sparing individually without impacting other cores, further improving availability.

Support units

In addition to mainline units and the pipeline already described, extra supporting units are required to implement a robust mainframe processor.

Address translation unit

The XU contains the second-level TLB (TLB2) and a programmable translation engine that supports dynamic address translation (DAT) and access register translation (ART). The design is similar to the z9 design, efficiently supporting large working sets and the virtualization architecture.

Referring to **Figure 7**, the TLB2 is composed of three hierarchically structured subunits:

- A small-attribute content-addressable memory that serves for purge operations of entries stored in the lower-level subunits with equal logical partition (LPAR) identifications and other attributes (such as “guest level”).

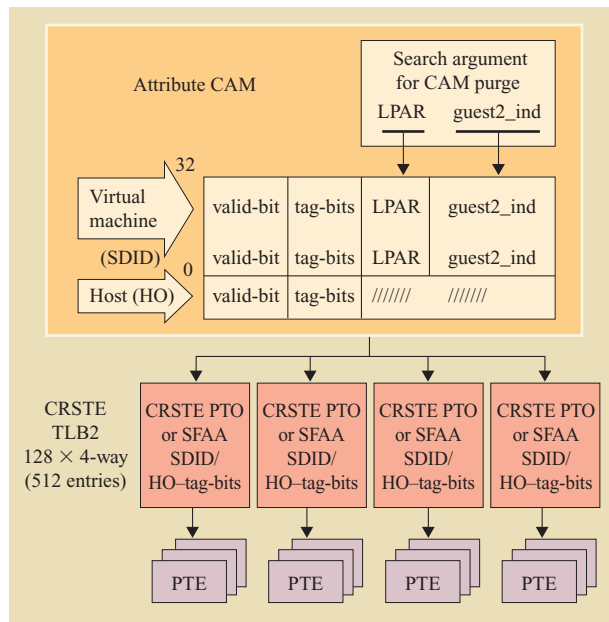


Figure 7

Attribute CAM with CRSTE and PTE TLB2. (CAM: content-addressable memory; LPAR: logical partition; PTO: page table origin; SDID: state description identification; SFAA: segment frame absolute address.)

- A 4-way set-associative common region segment table entry (CRSTE) TLB2 that covers the higher-level address translation levels.
- Four 3-way set-associative page table entry (PTE) TLB2s, each connected to a CRSTE compartment, covering the lowest translation levels.

Support for a new enhanced-DAT architecture that provides a 1-MB large page frame is added to the unit. Although the TLB1s are still maintained using entries of 4-KB pages to reduce design complexity, all 4-KB pages within the same large page are managed consistently through TLB2 and millicode. Performance gains are readily expected through reductions in both the TLB2 miss rate and software time spent managing pages [13].

L1.5 unit

The L1.5 unit is a private ECC-protected 3-MB 12-way set-associative second-level cache. It is designed as two parallel and independently running cache interleaves, one for even lines and one for odd lines, based on address bit 55. For L1.5 hits, 16 QWs (quadwords, 1 QW = 16 bytes) of a line are returned 1 QW per cycle back-to-back to the L1s, with an average cache-hit latency of 14.5 cycles.

The L1.5 directory includes indications tracking L1 cache-line ownerships to ensure valid entries in L1s are proper subsets of entries in L1.5. In particular, self-modifying code is supported such that updated code storage can be passed from the D-cache through the L1.5 to the I-cache. It guarantees that while the D-cache is storing, the same line cannot be used in the I-cache, and vice versa.

Since both the I-cache and the D-cache are virtually indexed (and absolute-address tagged) and the z/Architecture allows multiple virtual addresses to be translated to the same absolute address, storage of the same absolute address could possibly coexist in different entries in the L1 caches. To avoid such multiple existences of a line, the L1.5 maintains a single value of a virtual address that is part of the synonym values (bits 50, 51 used for L1 cache indexing) and allows only one such value to exist for each absolute address in its cache. By actively invalidating L1 copies with different synonym values, it ensures a single image of each absolute address.

Cross interrogates (XIs) from L2 are first processed in the L1.5. Acting as an ownership filter, the L1.5 forwards only XIs to the L1s if required, thus reducing XI processing time in the core for lines that are already replaced in the L1.5 (and the L1s).

Coprocessor

The z10 coprocessor (COP) for data compression and cryptography⁴ is based on the previous z9 design. The COP is still driven mainly by millicode. Each COP consists of two compression (and expansion) units, one cryptographic cipher engine, and one cryptographic hash engine. Each compression unit includes a 32-entry TLB and a 4-way 16-KB local cache for compression table fetching and is dedicated to a core. Each cryptographic engine, however, is shared between two cores. Such sharing of the COP is fully transparent to software. A possible increase of instruction execution latency is noticed only if both sharing cores are trying to use the same cryptographic engine.

This method of transparent sharing of the COP is a highly efficient way to provide new and complex functionality. With this construct, new COP features can easily be provided in the future that do not proportionally increase area per core, but still allow software use.

The z9 Data Encryption Standard and Triple Data Encryption Standard functions remain in the z10 design, and in addition, enhancements are added to the execution of the Advanced Encryption Standard and the secure hash algorithm (SHA-2) to support up to 192-/256-bit and 384-/512-bit keys, respectively. The COP is capable of performing up to an 8.8-GB/s expansion, a 240-MB/s

⁴Cryptographic functions implemented are called the *CP assist for cryptographic function* (CPACF).

compression, and a 290-MB/s to 960-MB/s bulk encryption rate.

Conclusion

In addition to the high-frequency pipeline that runs at 4.4 GHz, other distinctive innovations within the z10 core have also been described. These innovations address various aspects of a microprocessor design. The enhanced branch prediction reduces misprediction penalties and initiates I-cache prefetching. The L1.5 cache and the support for both software cache management and hardware data prefetching reduce the overall cache-miss penalties. The second-level TLB and the large page provision reduce overall TLB-miss latencies and software overhead. New instructions have been added to support software optimization. In addition, decimal floating-point operations are done in hardware, and COP functionalities are enhanced. Finally, many power-saving techniques are incorporated for an energy-efficient design suitable for a mainframe system.

The z10 core, together with a robust cache hierarchy and an SMP system design, provides a significant performance increase over its predecessor z9 core for enterprise database and transaction processing workloads as well as for new processor-centric workloads. As software is optimized for the z10 pipeline and to make use of the new architectural features, further performance gains are expected.

Acknowledgments

The development of the z10 microprocessor was made possible by many dedicated engineers. In particular, we thank the system chief architect, Charles Webb; core architects, Jane Bartik, Mark Farrell, Bruce Giamei, Lisa Heller, Tim Koprowski, Barry Krumm, Martin Recktenwald, Dave Schroter, Scott Swaney, Hans-Werner Tast, and Michael Wood; performance modeling team, James Bonanno, David Hutton, and Jim Mitchell; physical design leads, Robert Averill, Sean Carey, Chris Cavitt, Yiu-Hing Chan, Adam Jatkowski, Mark Mayo, and Joseph Palumbo; and technical leaders, Hung Le and Brian Konigsburg. There were many other individuals who are not mentioned because of space, but their contributions are certainly appreciated.

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

References

1. C. F. Webb, "IBM z10: The Next-Generation Mainframe Microprocessor," *IEEE Micro* **28**, No. 2, 19–29 (2008).
2. P. Mak, C. R. Walters, and G. E. Strait, "IBM System z10 Processor Cache Subsystem Microarchitecture," *IBM J. Res. & Dev.* **53**, No. 1, Paper 2:1–12 (2009, this issue).

3. IBM Corporation, *Large Systems Performance Reference*, Document No. SC28-1187-12, February 2008; see http://www-03.ibm.com/systems/resources/servers_eserver_zseries_lspr_pdf_SC28118712.pdf.
4. C. F. Webb and J. S. Liptay, "A High-Frequency Custom CMOS S/390 Microprocessor," *IBM J. Res. & Dev.* **41**, No. 4/5, 463–473 (1997).
5. K. E. Plambeck, W. Eckert, R. R. Rogers, and C. F. Webb, "Development and Attributes of z/Architecture," *IBM J. Res. & Dev.* **46**, No. 4/5, 367–379 (2002).
6. H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden, "IBM POWER6 Microarchitecture," *IBM J. Res. & Dev.* **51**, No. 6, 639–662 (2007).
7. T. J. Slegel, E. Pfeffer, and J. A. Magee, "The IBM eServer z990 Microprocessor," *IBM J. Res. & Dev.* **48**, No. 3/4, 295–309 (2004).
8. L. C. Heller and M. S. Farrell, "Millicode in an IBM zSeries Processor," *IBM J. Res. & Dev.* **48**, No. 3/4, 425–434 (2004).
9. A. Bieswanger, M. Andres, J. Van Heuklon, T. B. Mathias, H. Osterdorf, S. A. Piper, and M. R. Vanderwiell, "Power and Thermal Monitoring for the IBM System z10," *IBM J. Res. & Dev.* **53**, No. 1, Paper 14:1–9 (2009, this issue).
10. K. M. Jackson, M. A. Wisniewski, D. Schmidt, U. Hild, S. Heisig, P. C. Yeh, and W. Gellerich, "IBM System z10 Performance Improvements with Software and Hardware Synergy," *IBM J. Res. & Dev.* **53**, No. 1, Paper 16:1–8 (2009, this issue).
11. B. Curran, B. McCredie, L. Sigal, E. Schwarz, B. Fleischer, Y.-H. Chan, D. Webber, M. Vaden, and A. Goyal, "4GHz+ Low-Latency Fixed-Point and Binary Floating-Point Execution Units for the POWER6 Processor," *Proceedings of the IEEE International Solid-State Circuits Conference*, San Francisco, CA, 2006, pp. 1728–1734.
12. E. M. Schwarz, J. S. Kapernick, and M. F. Cowlishaw, "Decimal Floating-Point Support on the IBM System z10 Processor," *IBM J. Res. & Dev.* **53**, No. 1, Paper 4:1–10 (2009, this issue).
13. E. Tzortzatos, J. Bartik, and P. Sutton, "IBM System z10 Support for Large Pages," *IBM J. Res. & Dev.* **53**, No. 1, Paper 17:1–8 (2009, this issue).

Received February 14, 2008; accepted for publication June 16, 2008

Chung-Lung K. Shum *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (cshum@us.ibm.com)*. Mr. Shum received his B.S. and M.S. degrees in electrical engineering from Columbia University. He joined IBM in 1988 and has been working on IBM zSeries* processor development. He was the chief architect and lead for the z10 microprocessor core. He previously led the team for the L1 cache units of the z900 and z990 mainframe processors.

Fadi Busaba *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (busaba@us.ibm.com)*. Dr. Busaba received his Ph.D. degree in computer engineering from North Carolina State University. He joined IBM in 1997. He has worked on cross-coupling noise analysis, logic synthesis, and CAD tools. He led the z10 instruction decode unit (IDU) team and previously led the fixed-point unit (FXU) team for the z990 mainframe processor.

Son Dao-Trong *IBM Systems and Technology Group, IBM Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (daotrong@de.ibm.com)*. Dr. Dao-Trong received his M.S. and Ph.D. degrees in electronic engineering from Technische Universitaet Karlsruhe, Germany. He joined IBM in 1985 and has worked on different areas of computer design. He was the team leader for the z10 binary floating-point unit (BFU).

Guenter Gerwig *IBM Systems and Technology Group, IBM Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (ggerwig@de.ibm.com)*. Mr. Gerwig received both his B.S. and M.S. degrees in electrical engineering from University of Stuttgart, Germany. He joined IBM in 1981 to work on chip card readers for banking systems. He was the team leader for the z10 recovery unit (RU) and previously led the BFU design for the G3, z990, and z9 mainframe processors.

Christian Jacobi *IBM Systems and Technology Group, IBM Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (cjacobi@de.ibm.com)*. Dr. Jacobi received both his M.S. and Ph.D. degrees in computer science from Saarland University, Germany. He joined IBM in 2002 where he has worked on floating-point implementation for various IBM processors. He led the team for the z10 L1.5 unit.

Thomas Koehler *IBM Systems and Technology Group, IBM Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (thkoehler@de.ibm.com)*. Mr. Koehler received his M.S. degree in electrical engineering from University of Stuttgart, Germany. He joined IBM in 1986 to work on memory and I/O adapter development for IBM server systems. In 1991 he began work on data compression engines and has been leading the coprocessor (COP) unit team since 1998.

Erwin Pfeffer *IBM Systems and Technology Group, IBM Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (epfeffer@de.ibm.com)*. Mr. Pfeffer received his graduate degree in electrical engineering from Johannes-Kepler-Polytechnikum in Regensburg, Germany. He joined IBM in 1971 and worked on printer and inspection tool development and processor microcode development. He subsequently led the execution unit team of an early CMOS processor, and then the address translation unit (XU) team for multiple zSeries processors.

Brian R. Prasky *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (brprasky@us.ibm.com)*. Mr. Prasky received his B.S. and M.S. degrees in electrical and computer engineering from Carnegie Mellon University. He joined IBM in 1998 and has been working in zSeries processor

development. He led the z10 IFU team and previously worked on the z900, z990, and z9 mainframe processors.

John G. Rell *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (relljohn@us.ibm.com)*. Mr. Rell received his B.S. degree in electrical engineering from Rensselaer Polytechnic Institute. He joined IBM in 1999 and has had various logic design and simulation responsibilities for IBM zSeries processors. He led the z10 FXU team and previously led the verification of the z990 mainframe processor.

Aaron Tsai *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (atsai@us.ibm.com)*. Mr. Tsai received his B.S. degree in electrical engineering from Cornell University. He joined IBM in 1997 and has worked on various aspects of zSeries processor design including front-end and back-end tools and methodology, circuit design, and logic design. He was the lead microarchitect of the z10 load/store unit (LSU).