IBM z10: The Next-Generation Mainframe Microprocessor

THE IBM SYSTEM Z10 INCLUDES FOUR MICROPROCESSOR CORES—EACH WITH A PRIVATE 3-MBYTE CACHE—AND INTEGRATED ACCELERATORS FOR DECIMAL FLOATING-POINT COMPUTATION, CRYPTOGRAPHY, AND DATA COMPRESSION. A SEPARATE SMP HUB CHIP PROVIDES A SHARED THIRD-LEVEL CACHE AND INTERCONNECT FABRIC FOR MULTIPROCESSOR SCALING. THIS ARTICLE FOCUSES ON THE HIGH-FREQUENCY DESIGN TECHNIQUES USED TO ACHIEVE A 4.4-GHZ SYSTEM, AND ON THE PIPELINE DESIGN THAT OPTIMIZES Z10'S CPU PERFORMANCE.

Charles F. Webb
IBM

••••• The IBM System z10 processor is the engine for the next generation of IBM System z mainframe servers. Its new microprocessor core leverages leading-edge high-frequency design technology but also maintains full compatibility with the existing z/Architecture instruction set architecture.[1] The processor chip incorporates several special-function units to accelerate specific operations and includes robust hardware fault detection and recovery mechanisms. In concert with a companion symmetric multiprocessing (SMP) hub chip, the z10 processor implements a shared-cache multiprocessor structure that is optimized for the enterprise data-serving workloads at the heart of the mainframe market.

Chip summary

The z10 processor chip (Figure 1) contains four microprocessor cores, each with a private 3-Mbyte second-level cache. The cores are arranged symmetrically on the chip, each surrounded on three sides by its second-level cache. There are two coprocessor units, each implementing cryptographic and data compression functions; each is shared by two cores. The chip also includes a memory controller, an I/O bus controller, and a switch that connects all four cores to a shared interface with the SMP hub chip and its shared cache. The z10 processor is implemented in IBM's 65-nm silicon-on-insulator (SOI) process, has a die size of 454 mm^2, contains 994 million transistors, and operates at 4.4 GHz in a multiprocessor system.

Relationship to Power6

The z10 and Power6[2] processors were developed side by side as part of a major development program within IBM's Systems and Technology Group to address the technology challenges and market requirements faced by IBM Systems i and p (built on the Power RISC architecture) and System z (built on the CISC z/Architecture). It was evident from the beginning that a single microprocessor core could not satisfy the requirements of both architectures, but it was also clear that those

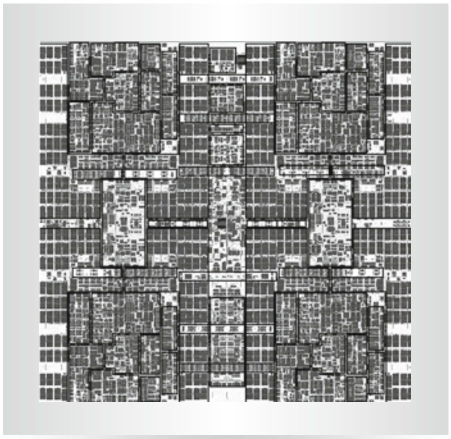


Figure 1. The z10 processor chip includes four microprocessor cores, each surrounded on three sides by its 3-Mbyte second-level cache. Two coprocessor units, each shared by two cores, provide cryptography and data compression.

requirements were driving both cores to conceptually similar designs, and that we could share many building blocks between the two designs without compromising either.

This led to the formation of a single "super team" that could pool the experience of the mainframe and Power design teams. A common design infrastructure (tools, methodology, libraries, latches, clocking structure, and so on) enabled the development of common building blocks (SRAMs, register files, basic data-flow elements) and allowed designer mobility between the two designs. At a higher level, common requirements and similar core frequencies let the two designs share significant portions of the fixed-point, binary floating-point, and decimal floating-point[3] units within the cores; and the I/O bus and memory controllers at the chip level. We kept the control designs for the cores, however, completely independent of one another, because of the very different instruction sets (CISC versus RISC), legacy code, and workload requirements.

At the chip and system levels, differing workload and product portfolio requirements drove very different cache and multiprocessor designs. The Power6 design, for example, includes two binary floating-point units per core and two memory controllers per chip to support the requirements of engineering and scientific workloads. The z10 design, on the other hand, includes on-chip hardware acceleration for data compression and encryption to improve performance of commercial data-serving workloads. These commercial workloads also benefit significantly from the relatively large on-chip caches implemented on the z10 chip, and from the shared-cache, centralized-control SMP structure enabled by the quad-core processor chip, SMP hub chip, and integrated packaging used in the System z10.

Architecture

The z10 processor fully implements the IBM z/Architecture[1]—the current generation of the mainframe instruction set architecture that dates back to the S/360, which celebrated its 40th anniversary in 2004. Although the architecture has undergone many major extensions during this span, IBM has rigorously maintained upward binary compatibility at the application program level to preserve customer investment and ensure a smooth transition from one generation to the next. For z10, we added more than 50 instructions to z/Architecture, chiefly to improve the efficiency of compiled applications. The z10 processor also adds support for 1-Mbyte page frames and software-hardware interfaces to improve cache efficiency.

Core summary

The z10 microprocessor core is organized into eight units, shown in Figure 2. The instruction fetch unit (IFU) contains a 64-Kbyte instruction cache, branch prediction logic, instruction-fetching controls, and instruction buffers. This unit's size is a result of the rich branch prediction design required to minimize branch mispredictions in a high-frequency pipeline across a broad range of programs.

The instruction buffers in the IFU feed the instruction decode unit (IDU) in the center of the core. This logic parses and decodes almost 900 distinct opcodes defined in z/Architecture (668 of which are implemented entirely in hardware in z10), identifies dependencies among instructions, forms instruction pairs for superscalar execution when possible, and issues instructions to the operand access and execution logic.

The load-store unit (LSU) includes a 128-Kbyte data cache and handles operand accesses across the broad range of lengths, modes, and formats included in z/Architecture, and it supports two quadword fetches per cycle. It also buffers operand store results between instruction execution and completion, and it interfaces with the multiprocessor fabric (through the second-level cache) to maintain the required strongly ordered cache coherence. The LSU is coupled with a translation unit (XU), which consists of a large second-level translation look-aside buffer (TLB) and a hardware translation unit. The latter handles access register translation and dynamic address translation (DAT) to convert logical addresses to real addresses, including the nested DAT required for operating systems running as guests under the z/VM (virtual machine) hypervisor.

Three units handle the actual instruction execution: The fixed-point unit (FXU) performs fixed-point arithmetic, logical, and branch instructions. The FXU executes most of these in a single cycle, and in pairs, with a full forwarding network to allow back-to-back execution of dependent operations. The binary floating-point unit (BFU) is a multistage pipeline that handles all binary (IEEE-754 compliant) and hexadecimal (S/360 legacy) floating-point operations. This unit can start one operation per cycle, and uses intrapipeline forwarding of results to minimize pipeline delays between dependent instructions. The BFU also executes fixed-point multiplication and division instructions. The decimal floating-point unit (DFU) executes both floating-point (IEEE-754R compliant) and fixed-point (S/360 legacy) decimal operations. Decimal floating-point functionality first appeared in the z/Architecture on IBM System z9,[4] which implemented these instructions with a combination of hardware and internal code (millicode); on the z10 processor, these are implemented fully in hardware, which improves performance

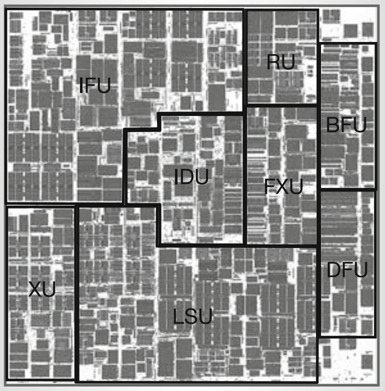


Figure 2. The z10 microprocessor core, organized into eight units. (IFU: instruction fetch unit; IDU: instruction decode unit; LSU: load-store unit; XU: translation unit; FXU: fixed-point unit; BFU: binary floating-point unit; DFU: decimal floating-point unit; RU: recovery unit.)

for business applications requiring decimal computation. (See the "Why Decimal Floating Point?" sidebar.)

Finally, the recovery unit (RU) maintains a complete copy of the processor architected

Why Decimal Floating Point?

Mike Cowlishaw, IBM Fellow

Binary floating-point numbers can only approximate common decimal numbers. The value 0.1, for example, would need an infinitely recurring binary fraction. This causes subtle problems; for instance, consider the calculation of adding a 5 percent sales tax to a $0.70 telephone call, rounded to the nearest cent. Using binary floating-point numbers, the result of 0.70×1.05 before rounding is just less than the correct result (0.735) and hence would be rounded down to $0.73. With decimal floating-point numbers, the intermediate result would be exactly 0.735, which would then round up correctly to $0.74.

For this and other reasons, binary floating-point computation cannot be used safely for financial calculations, or indeed for any calculations where the results achieved are required to match those which might be calculated by hand.

IBM's mainframe processors have always had binary-coded decimal (BCD) instructions, but these are hard to use for anything other than fixed-point calculations. However, in recent years, decimal calculations are more common (interest rates change daily, for instance) and more complicated (more analysis is done on currency transactions, for example).

The decimal floating-point unit in the z10 processor allows all calculations, including mathematical and statistical, to be done in the new decimal formats, so no conversions to binary are needed, and exact decimal results are given where expected. Conversion to and from BCD or strings is easy too.

For more examples and information, see http://www2.hursley.ibm.com/decimal.

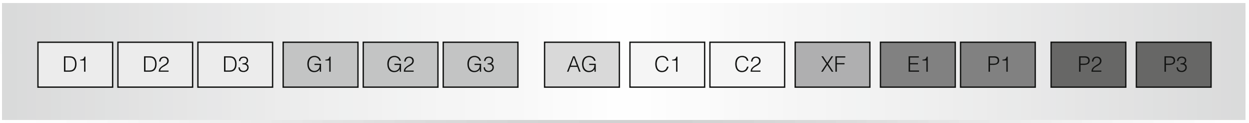


Figure 3. The z10 core pipeline for a typical FXU instruction. (D1-D3: parse, decode, and identify inter-instruction dependencies, and deliver results to the instruction queue and the address generation queue; G1-G3: determine stalls and group for superscalar execution; AG, C1, C2, and XF: deliver operands for the instruction (or instruction pair) to the FXU; E1 and P1: generate results and condition code; P2 and P3: resolve conditional branches, write results to register files, check results for hardware faults, and forward results to the RU.)

state, protected by error-correcting code (ECC). This state includes all z/Architecture registers as well as various mode and state registers used by hardware and millicode to implement z/Architecture functions. The RU collects all hardware fault detection signals and oversees hardware recovery actions if these signals indicate any fault.

High-frequency design

The most distinctive feature of the IBM z10 core design relative to its predecessors is the giant leap in operating frequency—from 1.7 GHz on System z9 to 4.4 GHz on z10-based systems. Starting with the S/390 CMOS G4 processor in 1997,[5] IBM mainframe CPU cores have had a cycle time of roughly 27 to 29 FO4 (fanout of 4 inverter delays) and a six-cycle pipeline (counting from instruction decode through register put-away). Through six generations of systems and silicon technology, the design team has maintained that cycle size and pipeline depth while adding substantial functions (such as IEEE-compliant floating-point capability, branch target prediction, full 64-bit architecture extensions, superscalar operation, and cryptography).[6–8] The design of the z10 processor, however, started from a clean sheet, aiming for a far shorter 15-FO4 cycle width, and balancing performance, power, area, and design complexity considerations.[9,10] This change required innovation in the design methodology, pipeline structure, and architecture implementation.

A comprehensive high-frequency design infrastructure was essential to the success of the z10 project. As noted earlier, because the Power6 and z10 designs faced the same challenges and were using the same process technology, the two projects could share this foundation, including low-level building blocks such as latches and data-flow elements. (Another publication describes this approach to designing a high-frequency processor.[2])

Pipeline

Figure 3 shows the pipeline for the z10 core, from instruction decoding through results put-away, for a typical FXU instruction. The IFU feeds instructions into this pipeline, where they are processed in program order.

The first three cycles—D1, D2, and D3—parse and decode the instructions (up to two per cycle), identify inter-instruction dependencies, and deliver the results to the instruction queue (IQ) for execution and the address generation queue (AQ) for storage operand access. The next three cycles—G1, G2, and G3—determine the required stalls between instructions and group pairs of instructions for superscalar execution, if possible. When the instruction dependencies and downstream pipeline allow, instructions are issued from the IQ and AQ to the FXU and LSU, respectively. In conjunction with the issuing of instructions from the AQ, the necessary access registers and general registers are read from the register files and supplied to the operand access controls.

Once an instruction (or instruction pair) has been issued from the IQ and AQ, it proceeds through the remainder of the pipeline without further stalls. If some condition, such as a cache or TLB miss, is detected that inhibits the instruction's completion, rather than being stalled, it is recycled to the grouping stages (G1 through G3) and reissued. This most commonly occurs for TLB or cache directory misses on operand accesses, or for inter-instruction

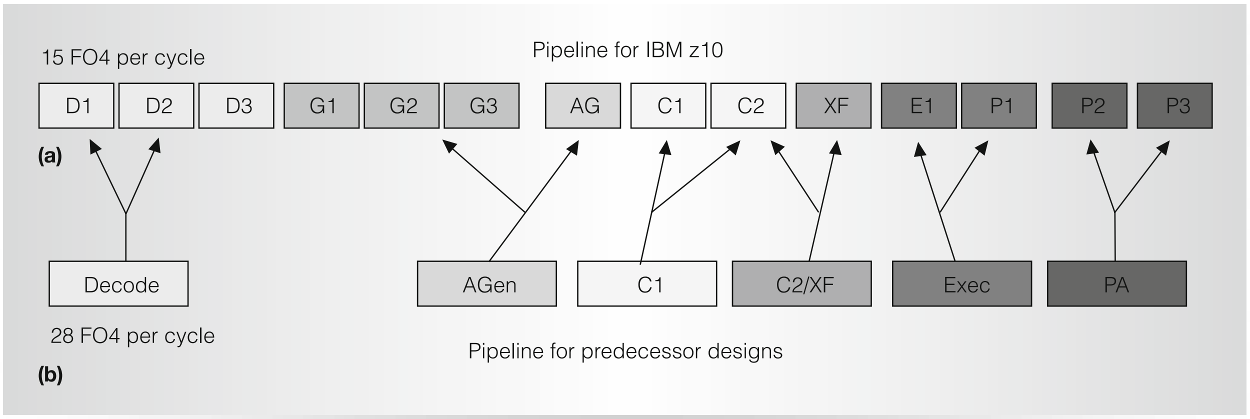


Figure 4. Pipeline comparison: z10, with a 15-FO4 cycle time (a), and predecessor designs, which had a 28-FO4 cycle time (b).

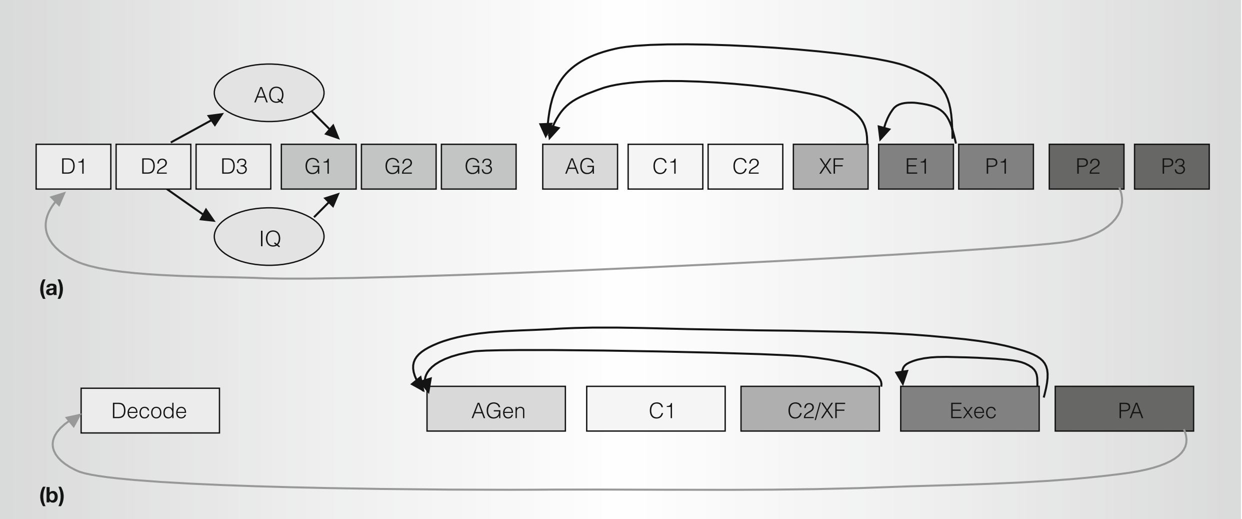
hazards that could not be detected at instruction decode time, such as address-based dependencies. This nonstalling design of the next portion of the pipeline avoids the need for global stall signals and enables the high-bandwidth flow of instructions when the instruction sequence allows. In most cases, an instruction can be recycled and is ready to be reissued by the time the recycling condition is resolved, so that the effective performance is the same as it would be if the hazard had been precisely predicted upstream.

The next four stages of the pipeline—AG, C1, C2, and XF—are responsible for delivering the operands for the instruction (or instruction pair) to the FXU. As with prior mainframe CPU designs, the pipeline is optimized for instructions that get one or both of their operands from cache, and for the flow of operands from *load* instructions to subsequent register-operand instructions. This gives register-register, register-storage, and storage-storage instruction types the same pipeline timing, and also eliminates any pipeline latency for forwarding operands from *load* instructions. Performance studies have shown that this structure provides robust performance for a broad range of programs, particularly those optimized for the z/Architecture instruction set.[9]

For instructions that require one or more operands from the data cache, or that will store a result to the cache, the AG, C1, and C2 cycles generate the operand address (or addresses) and access the TLB, cache directory, and cache array. The XF cycle is used to align, format, and transfer the operand data from the LSU to the FXU.

During the AG, C1, and C2 cycles, the FXU is preparing for execution by performing additional FXU-specific decoding of the instruction, reading any register operands, and setting up controls for the FXU data flow. The actual execution takes place in the E1 cycle, and results can be forwarded immediately to a dependent FXU instruction or to a dependent address generation. Some execution functions extend into the P1 cycle, including condition code generation. Finally, the P2 and P3 cycles resolve conditional branches, write results to register files, check results for hardware faults, and forward results to the RU.

The relatively large number of cycles required for decoding, grouping, and issuing instructions reflects both the complexity of the z/Architecture instruction set and a drive to push as much as possible of the associated hardware complexity into the front end of the pipeline. This minimizes the amount of work required in the subsequent stages, which are far more critical to the performance of a pipelined processor. This becomes more apparent through a comparison, shown in Figure 4, of the z10 pipeline to that used in its predecessors, from S/390 CMOS G4 through System z9. The total number of cycles from decode through put-away grew from 6 to 14—a ratio slightly larger than the FO4 ratio—reflecting the cost of



(c)

Pipeline loop	Cycles		FO4	
	IBM z9	IBM z10	IBM z9	IBM z10
Load-FXU	0	0	0	0
FXU-FXU	1	1	28	15
Load-load	3	4	84	60
FXU-load	4	5	112	75
Branch wrong	6+	13+	168+	195+

Figure 5. Performance-critical pipeline latencies: z10 and its latencies (a), z9 and its latencies (b), and comparison table (c).

additional latch levels in the z10 pipeline. Almost all the growth, however, was in the front end of the pipeline and in the put-away cycles. In the pipeline's performance-critical core, which encompasses address generation, operand access, and FXU execution, the z10 pipeline is only one cycle longer than its predecessors. This is crucial to processor performance because most of the inter-instruction dependencies occur within this scope, and the latencies to resolve these dependencies play a large role in determining the pipeline's effectiveness.

Figure 5 shows the key pipeline latencies. These are for

- FXU operands dependent on FXU results;
- operand address generation inputs (for a *load* or register-storage instruction) dependent on cache data (*load* results) or FXU results; and
- branch misprediction.

The latency for forwarding cache data to the FXU is zero cycles in both the prior designs and z10 because of the pipeline's optimization for register-storage operations.

As the table in Figure 5 shows, latencies within the core of the pipeline grew by at most one cycle despite z10's large frequency increase. When measured in FO4, these latencies decreased dramatically with the new design. This is a result of removing control complexity from the central cycles, which also drove disproportionate growth in the pipeline's front end. This, in turn, affected the branch misprediction latency, which grew not only in cycles but also in FO4 from the prior designs.

Operand access

The redistribution of control complexity in the z10 pipeline enabled a drastic reduction in the control logic latency in the central pipeline cycles. To leverage this change in the control flow, however, we had

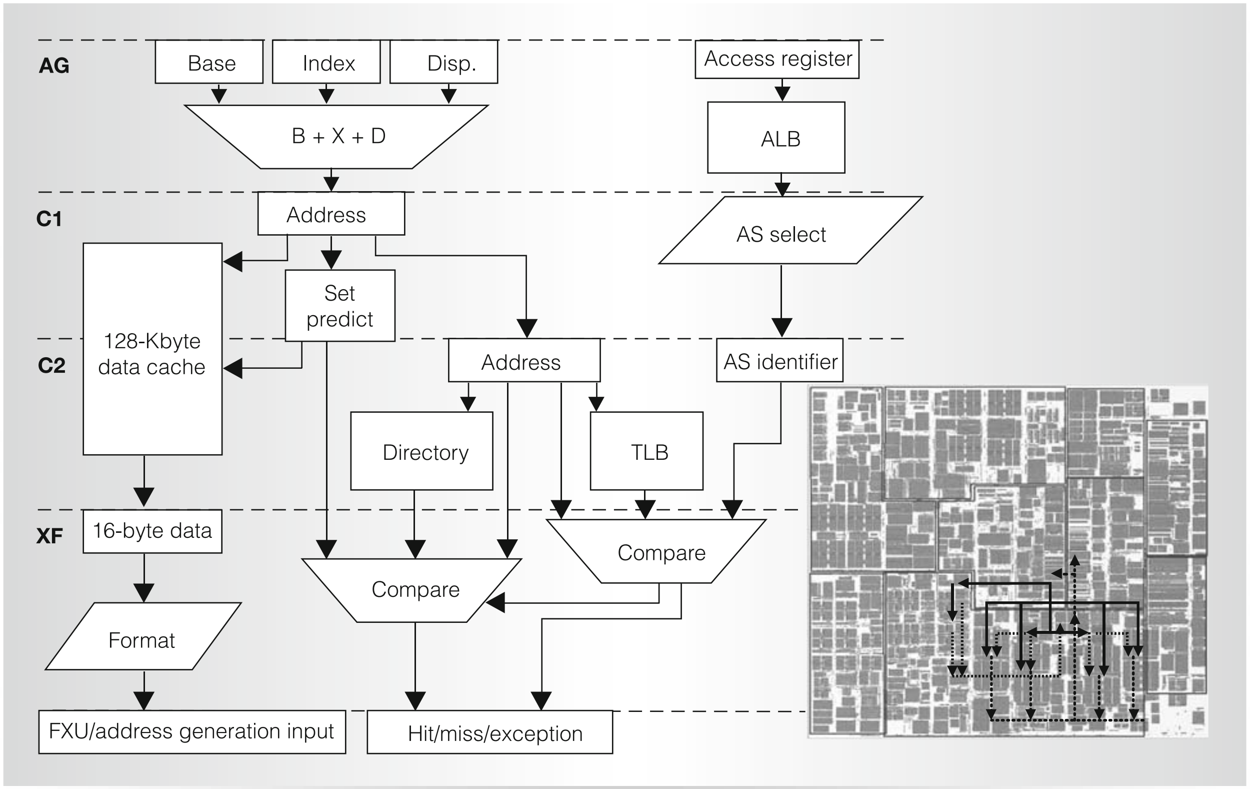


Figure 6. Operand access path. Dashed lines indicate cycle boundaries. (ALB: access register translation look-aside buffer; AS: address space.)

to correspondingly reduce the address and data flow through those cycles. The techniques used to accomplish this for FXU execution are essentially the same in z10 as in Power6,[2] but the design of the operand access pipeline is unique to the z10 processor and reflects z/Architecture's rich operand addressing capabilities. We established these two paths (one cycle for FXU execution, four cycles for operand access) very early in z10's design as the frequency-limiting paths, and we designed the rest of the pipeline to ensure that no other paths would drive a longer cycle time.

Figure 6 shows the design of the operand access path. In the first cycle (AG), a three-operand address addition (base + index + displacement) generates the logical address, while the access register translation look-aside buffer (ALB) is accessed to begin determining the operand address space. Across the next two cycles (C1 and C2), the data cache, TLB, and cache directory are accessed in parallel. A history-based cache set prediction table (indexed by a portion of the logical address) decouples the cache access from the TLB and directory lookup, and the address space selection occurs at the same time using the ALB results and other program controls. In the fourth cycle (XF), the operand is aligned and formatted as needed while the logical address and cache set prediction are compared to the TLB and directory contents to determine whether the correct data has been read.

The z10 design team devoted intense effort not only to this path's logical organization, but also to its physical design. The address adder is optimized for the delay on the bits needed to access the cache arrays (bits 50 to 59 of the 64-bit address). The two-port cache SRAM uses variable cycle stealing to enable optimal cycle boundary alignment. The address adders, cache arrays, and set prediction arrays are arranged to minimize maximum wire length on critical address paths, and the cache arrays are organized to facilitate byte-level rotation,

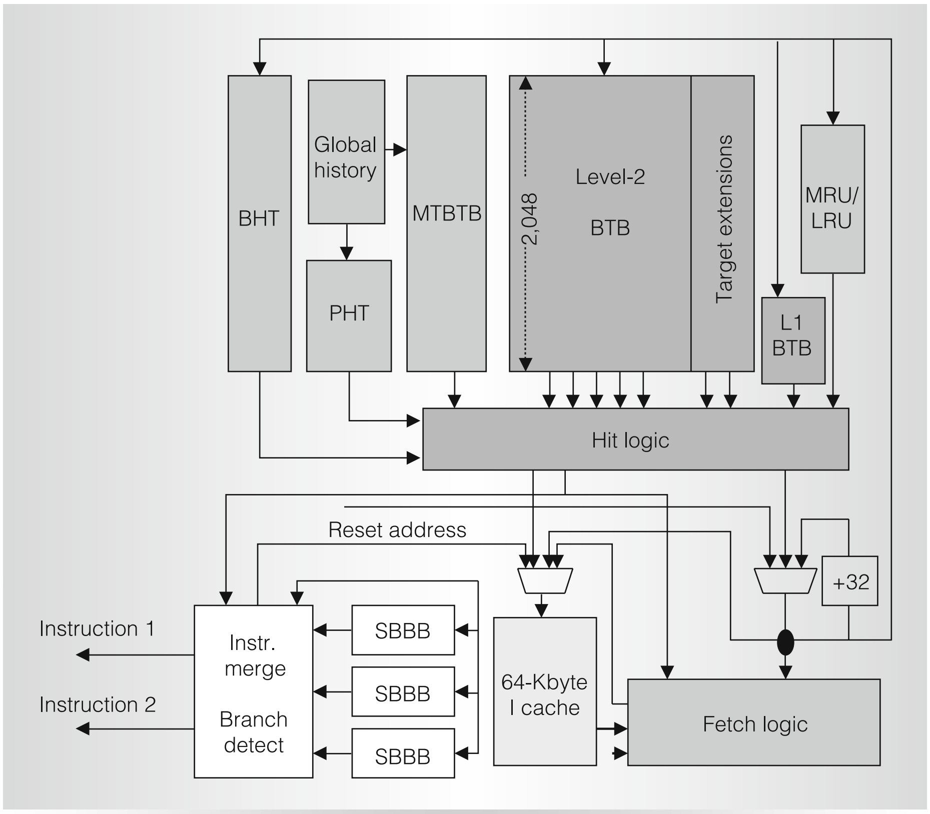


Figure 7. Branch prediction logic and instruction fetch structure in z10. (BHT: branch history table; PHT: pattern history table; MTBTB: multiple-target predictor; BTB: branch target buffer; SBBB: super-basic-block buffer.)

alignment, and merging of partial results (for multi-access operands) within the formatter. The entire operand access path is confined to a compact area within the z10 processor core, as the Figure 6 inset shows, minimizing the time of flight for address, control, and data signals.

Branch prediction

As noted earlier, one critical pipeline latency that grew as a consequence of z10's high-frequency pipeline design is that associated with branch misprediction, which spans the entire length of the pipeline. This high penalty for misprediction motivated a substantial investment in branch prediction mechanisms. In addition to a robust direction-prediction scheme, z10 maintains an emphasis on target prediction, as required by both the latency in the instruction fetching pipeline and the prevalence of branch instructions using register-based target addresses in z/Architecture programs, particularly older programs that still make up a large portion of commercial data-serving workloads.[11]

Figure 7 shows the structure of the z10 branch prediction logic. It includes two levels of branch target prediction: a small, low-latency buffer for tight loop handling, and a 10K-entry full-function branch target buffer (BTB) for maintaining longer-term history. To optimize area and performance, the target addresses are compressed and presumed to lie within a certain range of the branch instruction address; three of the five BTB associativity sets support a 1-Mbyte range; the other two sets support a 1-Gbyte range.

The BTBs are complemented by a traditional branch-history table (BHT) to predict the direction of conditional branches; a multiple-target predictor (MTBTB) to predict branch targets for switch tables, subroutine returns, and so on; and a pattern history table (PHT) to enhance BHT direction prediction with path-based context. The MTBTB and PHT are indexed by unique hash values derived from the global path history, reflecting the directions of the last several branches processed. Embedded into these mechanisms are special controls for handling the z/Architecture *execute* instruction and for optimizing transitions into and out of millicode mode, which, as in prior designs, serves to implement complex instructions and control functions.

As in prior mainframe designs, the branch prediction is designed to run ahead of instruction fetching and find the most likely path to follow. This enables the predicted stream to be fetched before the branch instructions are decoded, and provides a very accurate form of instruction cache prefetching. The instructions fetched are fed into a set of super-basic-block buffers (SBBBs), from which the z/Architecture instructions are extracted, merged, and sent to the IDU.

Special-function coprocessor

External to the microprocessor core, the z10 processor includes a pair of coprocessor units. Each of these units implements zArchitecture's data compression and cryptographic acceleration functions, which are provided to the software as conventional synchronous instructions. Each coprocessor serves two of the microprocessor cores and contains two compression engines (each with a 16-Kbyte local cache), a cryptographic cipher engine, and a cryptographic hash engine. This combination balances the area and power efficiency of a shared coprocessor with the need to minimize the performance impact of sharing in high-utilization workloads. Working in concert with millicode running in the microprocessor core, the z10 coprocessor can sustain throughput rates of 290 to 960 Mbytes/s for encryption (depending on the cryptographic protocol in use), up to 240 Mbytes/s for data compression, and up to 8.8 Gbytes/s for expansion of compressed data.

Reliability, availability, and serviceability

As in all prior mainframe processor designs, hardware reliability was a top concern during the design of the z10 processor. The design uses a rich suite of fine-grained redundancy and checking mechanisms, including ECC and parity, residue, functional, and state checking. The choice of checking method for each portion of the processor was determined by the type of logic and by local timing, area, and wiring constraints—for example, parity for address and data stacks, ECC for persistent state, residue checks for arithmetic elements, and functional consistency checks or local replication for complex controls. Regardless of the method used for the various portions, the entire design meets the requirement for thorough checking. The processor's architected state is maintained with ECC protection in the RU and is used to drive both CPU retry and dynamic CPU sparing, as pioneered on the S/390 CMOS G4 and G5 processors.[5,6] The top priority is preserving absolute integrity of program results, followed by continuous system availability.

Energy efficiency

Although it was not a primary design objective, power efficiency was important in the z10's design to maximize the performance attainable within the power delivery and cooling constraints of the system package. System-level cooling and selective use of high-V_T devices for noncritical paths limit the power wasted on leakage, and dynamic clock gating reduces active power. To further reduce power spent on spare or idle cores in the system, the hardware supports a millicode-invoked sleep mode that can shut down the instruction pipeline until an interruption is pending. System z achieves system-level power-performance efficiency by operating with consistent performance at a sustained high level of utilization, enabled by virtualization capabilities provided by the hardware, z/OS operating system, and PR/SM (processor resource/system manager) and z/VM hypervisors. The design emphasis therefore was on minimizing the sustainable maximum operating power, which determines the upper bound for operating voltage and frequency.

SMP hub chip

A hub chip designed specifically for use with z10 processors enables the construction of high-performance scalable SMP systems for enterprise data serving. The hub chip connects multiple z10 processor chips, enabling data bandwidth of 48 Gbytes/s between each hub-processor pair. The hub chip includes a 24-Mbyte SRAM cache that is shared among the connected processors, and pairs of hub chips can be combined to form a flat 48-Mbyte shared cache for that set of processors. Multiple pairs of hub chips can be connected to form larger SMP systems, and a low-latency fabric enables efficient SMP scaling and supports z/Architecture's strong coherency requirements. This central-switch, shared-cache structure provides significant performance advantages over more distributed fabric topologies for workloads with a high degree of data sharing and process interaction, which are common in many commercial transaction-processing environments.

Implemented in the same process technology as the z10 processor, the SMP hub chip consists of 1.6 billion transistors on a 445-mm^2 die.

The z10 processor shares high-frequency design techniques and building blocks with Power6, but has unique core, chip, and multiprocessor designs developed specifically for the System z enterprise data server market. The 4.4-GHz z10 core yields up to twice the performance of System z9 for CPU-intensive applications, which are a growing part of commercial data-processing workloads. Coupled with 3-Mbyte private and 48-Mbyte shared caches, the z10 design provides a 50 percent performance gain across a broad set of the traditional data-intensive database transactions that continue to drive real-world enterprise computing. This design provides a basis for extending the IBM mainframe platform through the next several generations of silicon technology. Future designs will build on this high-frequency foundation to improve pipeline efficiency, enhance hardware-software synergy and scalability, and further optimize performance within increasingly critical power density constraints. MICRO

Acknowledgments

The IBM z10 processor is the fruit of the innovation and labor of many individuals across IBM. For their contributions to the features described here and for their assistance in preparing this article, I extend particular thanks to Sean Carey, Ralf Fischer, Thomas Koehler, Pak-kin Mak, Frank Malgioglio, Mark Mayo, Tom McPherson, Brian Prasky, Eric Schwarz, Dave Schroter, Kevin Shum, Tim Slegel, Dave Webber, and Mike Wood.

References

1. *z/Architecture Principles of Operation,* document no. SA22-7832, IBM, 2000.
2. J. Friedrich et al., ''Design of the Power6 Microprocessor,'' *Proc. IEEE Int'l Solid-State Circuits Conf.* (ISSCC 07), IEEE Press, 2007, pp. 96-97.
3. L. Eisen et al., ''IBM Power6 Accelerators: VMX and DFU,'' *IBM J. Research and Development*, vol. 51, no. 6, Nov. 2007, pp. 663-684.
4. A. Duale et al., ''Decimal Floating-Point in z9: An Implementation and Testing Perspective,'' *IBM J. Research and Development*, vol. 51, no. 1 and 2, Jan.-Mar. 2007, pp. 217-228.
5. C.F. Webb and J.S. Liptay, ''A High-Frequency Custom CMOS S/390 Microprocessor,'' *IBM J. Research and Development*, vol. 41, no. 4 and 5, 1997, pp. 463-474.
6. T.J. Slegel et al., ''IBM's S/390 G5 Microprocessor,'' *IEEE Micro*, vol. 19, no. 2, Mar.-Apr. 1999, pp. 12-23.
7. E.M. Schwarz et al., ''The Microarchitecture of the IBM eServer z900 Processor,'' *IBM J. Research and Development*, vol. 46, no. 4 and 5, 2002, pp. 381-396.
8. T.J. Slegel, E. Pfeffer, and J.A. Magee, ''The IBM eServer z990 Microprocessor,'' *IBM J. Research and Development*, vol. 48, no. 3 and 4, 2004, pp. 295-310.
9. A. Hartstein and T.R. Puzak, ''The Optimum Pipeline Depth for a Microprocessor,'' *Proc. 29th Ann. Symp. Computer Architecture* (ISCA 02), IEEE CS Press, 2002, pp. 7-13.
10. M.S. Hrishikesh, N.P. Jouppi, and K.I. Farkas, ''The Optimal Logic Depth per

Pipeline Stage Is 6 to 8 FO4 Inverter Delays,'' *Proc. 29th Ann. Symp. Computer Architecture* (ISCA 02), IEEE CS Press, 2002, pp. 14-24.

11. C.F. Webb, ''S/390 Microprocessor Design,'' *IBM J. Research and Development*, vol. 44, no. 6, 2000, pp. 899-908.

Charles F. Webb is an IBM Fellow in IBM's Systems and Technology Group in Poughkeepsie, New York, working on processor and system design for future IBM System z servers. He was the overall technical leader for the z10 processor, and his technical interests include various aspects of mainframe processor design, including performance analysis, microarchitecture, instruction set architecture, and system design. He has a BS and an MEng in computer and systems engineering from Rensselaer Polytechnic Institute. He is a member of the IEEE Computer Society.

Direct questions concerning this article to Charles F. Webb, IBM Corp., MS P310, 2455 South Rd., Poughkeepsie, NY 12601; cfw@us.ibm.com.