IBM zEnterprise 196 microprocessor and cache subsystem

The IBM zEnterprise® 196 (z196) system, announced in the second quarter of 2010, is the latest generation of the IBM System z® mainframe. The system is designed with a new microprocessor and memory subsystems, which distinguishes it from its z10® predecessor. The system has up to 40% improvement in performance for traditional z/OS® workloads and carries up to 60% more capacity when compared with its z10 predecessor. The memory subsystem has four levels of cache hierarchy (L1 through L4) and constructs the L3 and L4 caches with embedded DRAM silicon technology, which achieves approximately three times the cache density over traditional static RAM technology. The microprocessor has 50% more decode and dispatch bandwidth when compared with the z10 microprocessor, as well as an out-of-order design that can issue and execute up to five instructions every single cycle. The microprocessor has an advanced branch prediction structure and employs enhanced store queue management algorithms. At the date of product announcement, the microprocessor was the fastest complex-instruction-set computing processor in the industry, running at a sustained 5.2 GHz, executing approximately 1,100 instructions, 220 of which are cracked into reduced-instruction-set computing-type operations, to achieve large performance gains in legacy online transaction processing and compute-intensive workloads.

Introduction

The IBM zEnterprise* 196 (z196) System z* platform represents the latest in a long lineage of IBM complementary metal-oxide semiconductor (CMOS) enterprise servers. It features up to 96 processors operating at 5.2 GHz distributed across four fully connected processing nodes, 3 TB of physical memory, up to 24 input/output (I/O) hubs, and a new four-level cache design that provides up to 60% more box capacity than the preceding z10* platform [1]. Together, these features enable the platform to better support the traditional Large Systems Performance Reference (LSPR) workload set, to better utilize the hardware/software synergy that emerged on the preceding z10 platform, and to provide a highly competitive platform for emerging workloads while continuing the trend for increased box capacity within a constant energy footprint from one generation to the next. The sections that follow contain a comparative analysis between the z196 platform and its predecessor z10 platform [2, 3].

Processor cache subsystem

Figure 1 shows the z196 and z10 cache subsystem hierarchies, highlighting the common elements between both machines. These common elements are the two chip-type system designs composed of a central processor (CP) chip containing four processor cores and a system controller (SC)/interconnect chip, packaged on a glass-ceramic multichip module (MCM) containing up to six processor chips and two SC chips. The MCM, along with up to 30 dual inline memory modules (DIMMs), and eight I/O card slots are packaged together to form the basic system pluggable unit of a processor book. In both platforms, up to four processor books are attached through a passive backplane, in a fully interconnected topology, to create a system that
scales from a single processor to the respective maximum configuration.

Figure 1 also highlights a number of changes that were introduced to the platform from z10 to z196. The platform added a sixth processor chip, increasing the maximum system processor capacity from 80 to 96 cores. This was enabled by a change in interconnect protocols between each processor chip and the SC chips, i.e., from separate control and data buses to a unified dynamically shared busing structure [4].

At the same time, the number of DIMM channels per memory port was increased from four to five with the introduction of a redundant array of independent memory on the physical memory [5], requiring the number of memory ports per node to be reduced from four to three because of connector constraints in the MCM package. In order to compensate for the potential reduction in aggregate memory bandwidth, the platform shifted from double-data-rate-2-533 (DDR2-533) to DDR3-800 DIMMs, with a redesigned on-DIMM advanced memory buffer (AMB) chip [6], resulting in a net generational memory bandwidth increase of approximately 13%. In addition, with the change in memory DIMM technology, the z196 platform shifted to two-deep cascading from three-deep on z10 through the use of denser DIMMs with four times the capacity of that used in the prior generation of the machine, effectively reducing the number of DIMMs required in the maximum memory configuration from 48 to 30. A complete comparative set of system bandwidths from z10 to z196 is shown in Table 1. The general trend is upward or increasing, with the exception of the processor chip to the SC chip interface, which was offset by a change in the system caching structure that effectively reduced the bandwidth requirement for the interface.

A new fourth level of cache is introduced to the processor cache subsystem to address three major sensitivity points...
that are key to our typical large shared workloads, namely, access latency, bandwidth, and working set size. As shown in Figure 2 and Table 2, the new cache level on z196 (L3) is effectively inserted between the z10 L1.5 and L2 cache levels, providing one-half the z10 L2 cache size at less than one-half the access latency, in processor clocks (pclk)s and time in nanoseconds. The z196 L3 cache is shared among four cores on the same processor chip, whereas the z10 L2 cache was shared among 20 cores in the node, more than doubling the amount of cache bits per core provided at the first-level shared cache, allowing the platform to persist a significant portion of the software working set in closer proximity to the requesting cores. This third-level cache is packaged on the processor chip, making it possible to provide a wide and robust interconnect to the processor cores, thus allowing for rapid movement of data among the four cores with minimal contention. With the introduction of the L3 cache, a careful tradeoff was made between aggregate on-chip cache size and effective cache latency. This resulted in a reduction in the second-level cache (i.e., L1.5 on z10 and L2 on z196) from 3 to 1.5 MB while still providing an optimal combination for system performance.

In addition, with the maturation of embedded dynamic RAM (eDRAM) technology and its incorporation into the processor subsystem, the normal cache density doubling effect that would have occurred with the transition from 65- to 45-nm technology was further doubled, resulting in quadrupling of the last-level cache in the system (i.e., L2 on z10 and L4 on z196) from 48 to 192 MB. The z10 L2 and z196 L4 caches act as the multiprocessor chip and multinode interconnect, providing low latency, high bandwidth, uniform access latencies for the processor chips within a book, and nodes within the system for the requests not resolved in the upper level caches (i.e., L1, L2, and L3). Of the two above changes, the shift from static RAM (SRAM) caches to eDRAM caches presented more challenges to the design because it resulted in two new industry innovations. These innovations are the first processor store-in eDRAM cache design [8], which effectively managed the intersection of store bandwidth and relatively long eDRAM cache busy times, and the introduction of a method for managing array refresh for more than 1,500 array macros in the L4 eDRAM cache design, which is composed of a pair of address-sliced SC chips [9]. With the processor store bandwidth being addressed at the L3 cache level, the L4 design was able to switch from full-line interleaving on z10 to a banked cache scheme on z196, reducing the average cache array utilization significantly while optimizing the design for power performance. These challenges and changes presented significant opportunity for the platform because the inclusion of this technology bolstered system performance significantly and reduced power consumption by roughly 40% over the preceding z10 SC chip cache using SRAM technology.

### Table 1

Aggregate speeds and feeds. (Note: All bandwidths are unidirectional in a maximally configured system.)

<table>
<thead>
<tr>
<th></th>
<th>z10</th>
<th>z196</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bus width</td>
<td>Bus speed</td>
</tr>
<tr>
<td>Core to/from L1.5</td>
<td>2×16 B</td>
<td>4.40 Gb/s</td>
</tr>
<tr>
<td>4× L1.5 to/from L2 (CP chip to/from SC)</td>
<td>2×8 B</td>
<td>2.93 Gb/s</td>
</tr>
<tr>
<td>L2 to/from L2 (SC to/from SC)</td>
<td>3×2×8 B</td>
<td>1.47 Gb/s</td>
</tr>
<tr>
<td>Max single node memory store BW</td>
<td>4×4 B</td>
<td>2.13 Gb/s</td>
</tr>
<tr>
<td>Max single node memory fetch BW</td>
<td>4×8 B</td>
<td>2.13 Gb/s</td>
</tr>
</tbody>
</table>
In addition to the structural changes in the z196 cache storage hierarchy, the design also incorporated a number of system protocol enhancements focused on reducing event latency, queuing, contention, and cache miss rates.

In the SC chip level, these enhancements included the introduction of the industry’s first asynchronous multinode coherency protocol lacking a combined response broadcast [10], which was accomplished through broadcasting the per-node partial responses throughout the system, while maintaining order-based merge stacks within each node for all outstanding or observed remote requests. This protocol change sped-up remote node combined response generation latency by 10 ns on average, along with certain directory hit state case resolution, while effectively reducing remote resource utilization at the same time.

A multinode horizontal cache persistency algorithm was added to the design, which effectively manages all of the individual congruence classes across the SC chips as one, dynamically moving the least recently used (LRU) eviction target line address directory state and data among the

Table 2  L1-miss access penalty. z10 and z196 storage hierarchy, cache size, nominal latency. (p: private cache; s4: shared cache among four cores; s20: shared cache among 20 cores; and s24: shared cache among 24 cores.)

<table>
<thead>
<tr>
<th></th>
<th>z10 (20 cores/node)</th>
<th>z196 (24 cores/node)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1.5p 3 MB</td>
<td>2.9 ns (13 pckls)</td>
<td>2.7 ns (14 pckls)</td>
</tr>
<tr>
<td>L2 48 MB</td>
<td>20 ns (88 pckls)</td>
<td>22.0 ns (114 pckls)</td>
</tr>
<tr>
<td>RmtL 48 MB</td>
<td>52 ns (229 pckls)</td>
<td>124.8 ns (254 pckls)</td>
</tr>
<tr>
<td>LclMem</td>
<td>134.3 ns (591 pckls)</td>
<td>LclMem</td>
</tr>
<tr>
<td>RmtMem</td>
<td>161.8 ns (712 pckls)</td>
<td>147.1 ns (765 pckls)</td>
</tr>
</tbody>
</table>
nodes within a multinode system through an algorithm of redundant directory state merging or invalid or empty compartment spilling, which effectively delays the write back of LRU eviction data to memory while increasing cache hit rates for workloads with long-latency reentrant data [11].

Switching the focus from the SC chip enhancements to the changes in the system-level protocols, z196 introduced centralized fairness queues at the L3 and L4 cache levels [12], which expanded the use of the z10 floating–pending queues to all requests in the system, effectively creating a multitude of request serialization queues based on the resource type required by an operation, by address comparisons among operations, or by request type. The queues have a dynamic depth management hardware algorithm since requests may be canceled before normal operational resolution, effectively allowing the queue to dynamically rethread itself, resulting in optimal allocation of system resources and more efficient serialization of system operations [13] compared with prior designs.

In addition, with the introduction of the L3 cache level, the platform expanded the traditional strongly ordered store cache management protocols and directory states to create a multilitered inclusive shared cache management protocol between the L3 and L4 caches, enabling the L3 cache to function in a similar manner as the traditional z10 L2 cache, but among a group of four processor cores, while having the L4 cache function in a more traditional manner as the SC chip, but with six processor chips in comparison to the 24 processor cores on the z10 platform. In doing so, a number of new cache directory states were introduced, including the “exclusive-to-processor chip” state at the L3 cache level, as shown in Table 3, which effectively allowed within the L3 cache the resolution of complex cache coherency management operations among the group of four processor cores. New hardware-based cache management interlocks between the L3 and L4 caches were introduced, including the L3 LRU notification command, which maintains consistency of exclusive line ownership between the L3 and L4 caches in order to reduce system intervention traffic and the associated latency penalty. These protocol changes and new interlocks effectively reduced the aggregate amount of communication required between the L3 and L4 caches, allowing the bandwidth of the system buses to be predominantly consumed with data transfers and not communication overhead.

Being the first level of store-in cache within the system and interconnected to four processor chips through their private store-through L2 caches, the L3 cache faced significant challenges in effectively managing the store bandwidth sent from the cores to the L3 eDRAM cache design, as previously mentioned. In overcoming this challenge, the L3 cache topology was altered in comparison to the prior fully interleaved SRAM cache design in z10, resulting in a dual absolute-address sliced eDRAM cache design containing subinterleaved interleaves, allowing for dual concurrent octword (32 bytes) store dismissals on any double-word permutation [9]. This created a hardware design where store operations are only busy for a fraction of a full cache interleave and enabled the storage hierarchy to drain stores at a rate comparable to that of the most store-intense workloads, effectively preventing any visible store contention issues under the normal LSPR workload mix.

**Processor cache hierarchy**

We now turn our attention from the shared system caches to the private caches in the system. Above the L3 cache is the 1.5-MB absolute-address indexed L2 cache, which acts as the interface between each core and the shared nest cache hierarchy. The L2 cache is shared between the three L1 caches, namely, instruction cache (I-cache), data cache (D-cache), and compression cache (i.e., I$, D$, and COP$, respectively).

The L2 directory maintains the consistency between the L1 caches. If, for example, a cache line is owned by both I-cache and D-cache and a store operation is executed, the D-cache requests exclusive ownership of the cache line from the L2. Before the L2 grants exclusivity, the L2 sends an invalidation request to the I-cache so that stores into the instruction stream are recognized. Once the store is completed, the updated instruction text can be refetched by the I-cache (causing the D-cache to lose exclusivity again). Similarly, the L2 handles invalidation requests from the L3 cache by forwarding the requests to the appropriate L1 caches; upon acknowledgment from the L1, the L2 then invalidates the L2 directory for the particular cache line.
The L2 cache can handle six L1 cache outstanding requests. On an L2 cache hit, the first data is returned to the L1 after 13 cycles in the fastest case, and the entire line transfer takes only 8 cycles. For L2 misses, the L2 can have up to six outstanding requests to the L3 caches, three to each slice. Data from each of the L3 slices is received on a 16-byte-wide bus that is clocked every other cycle; the bus into the L1 cache can transfer 32 bytes every other cycle. Thus, the aggregate peak bandwidth from the L3 into the L2 is 83 GB/s, and the aggregate peak bandwidth from the L2 into the L1 is 166 GB/s.

The I-cache is a 64-KB four-way set-associative cache. Accesses are requested by the instruction fetch unit (IFU) with a logical address in cycle i0, and on a cache hit, 32 bytes of instruction text is delivered two cycles later. On a cache miss, a request is sent to the L2 cache.

The D-cache is a 128-KB eight-way set-associative cache within the load–store unit (LSU). The four-cycle access pipeline is basically maintained from the z10 processor. It consists of address generation in cycle A0, address transfer and cache SRAM access in cycles A1 and A2, and data transfer and rotation or formatting in cycles A2 and A3. Set prediction is applied to predict which set and line will have the data before the directory compare is completed in cycle A4. On a mispredicted set and line, on a cache miss, or other reject conditions, the uop is rejected to the instruction sequencing unit (ISU), which then recycles the operation at a later time when the reject condition is resolved.

The LSU also contains the store queue (STQ) and forwarding logic. Significant effort was put into the forwarding capabilities since typical System z workloads tend to perform a lot of operations on memory and often need to refetch what was stored just a few instructions before. The store-forwarding logic consists of a fully associative search through the 16-deep STQ to see whether a fetch–uop overlaps with an older store. If an overlap is detected, the data from the STQ is used instead of the cache data. If the load only partially overlaps with the store, then the result of the load is merged between the cache and STQ data. If the load overlaps with multiple stores, then for each byte of the load, the STQ searches for the youngest store older than the load and forwards the data from that store to the load result. In contrast to most other microarchitectures, there is no performance penalty for obtaining data from the store-forwarding logic in comparison to cache data.

The I-cache and D-cache also contain first-level translation look-aside buffers (TLBs) to store dynamic address translations. New in z196 is the support for 1-MB pages in the first-level TLBs. The instruction TLB is two-way set-associative and 64 entries deep per set. Each entry can contain either a 4-KB or a 1-MB page translation information. Page-size prediction is assisted by the branch prediction logic by predicting the target page size in addition to the target addresses of taken branches. On a TLB miss, the TLB is reaccessed for the other page size to confirm or correct page-size prediction. If the TLB does not hit for either page size, then a request is sent to the translation unit. The IFU accurately predicts the page size in contrast to the data TLB since patterns of operand data accesses are more scattered. Therefore, the data TLB consists of two independent TLBs, one for the 4-KB and the other for the 1-MB translations. The 4-KB TLB is two-way set-associative with 256 entries per set; the 1-MB TLB is two-way set-associative with 32 entries. The two TLBs are concurrently accessed for each fetch or store microoperation.

### Processor microarchitecture

The z196 processor contains a new aggressive out-of-order execution, featuring a high-frequency core that improves upon the prior-generation System z complex-instruction-set computing (CISC) architecture microprocessors in several ways [14]. It increases the frequency from 4.4 GHz in the
prior-generation z10 core to 5.2 GHz and achieves the highest sustained frequency in the history of the microprocessor industry thus far. This is the first System z core in two decades to execute instructions out of program order and is the most aggressive out-of-order System z core in terms of the number of instructions in flight and out-of-order instruction window size. The System z predecessor S/360 Model 91 pioneered out-of-order execution in the 1960s, and several generations of bipolar mainframes in the early 1990s employed out-of-order techniques [15]. The z196 processor design follows another generation of a deep-pipelining high-frequency z10 microprocessor design. The prior z10 microprocessor, mainly an in-order processor, has a decode and an issue bandwidth of two instructions with a pipeline structure geared toward CISC register-memory (RX) instructions. In the z196 microprocessor, the traditional System z CISC (RX) pipelines are split into multiple shorter latency reduced-instruction-set computing (RISC)-like execution units, and the complex z/Architecture* instructions are cracked into RISC-like microoperations. Register-register (RR) instructions directly issue to the appropriate execution unit without being staged through the load–store pipeline, thus allowing other loads to be issued in parallel.

**Figure 4** shows the pipeline stages of the z196 processor from the first decode stage (D1) to completion. In the prior z10 microprocessor pipeline design [2], the fixed-point unit (FXU) execution stage is placed after the cache data return stage of the LSU; hence, there is no need to stall the execution of an FXU instruction if it is dependent on a load data event. However, this caused results of the RR-type instructions1 to go through many nonrequired stages (e.g., LSU pipe stages A0–A3), delaying the availability of results and eventually delaying the branch resolution cycle. The pipeline structure in z10 is suitable for an in-order issue. In the z196 pipeline, an RR-FXU instruction that is not dependent on a cache data return can issue and finish ahead of older instructions, thus freeing up issue queue (IQ) entry resources. The z196 processor can issue up to five instructions, or uops [two FXUs, two LSUs, one binary floating-point unit (BFU) or decimal floating-point unit (DFU)] in a single cycle. RX-type arithmetic instructions are issued twice from a single IQ entry. The first issue is to the LSU to load a data into a scratch register, and the second

---


---
issue, skewed by four cycles, is to the FXU where the loaded data from the first issue is used. Because of the physical location of the BFU/DFU, the register read (RF) is skewed by a cycle when compared to the RF stage of LSU and FXU execution pipelines.

The z196 core consists of multiple functional units: instruction fetch unit (IFU), instruction decode and dispatch unit (IDU), instruction sequencing unit (ISU), fixed point unit (FXU), load-store unit (LSU), hexadecimal and binary floating point unit (FPU), decimal floating point unit (DFU), translator unit (XU), compression (COP), recovery unit (RU), and pervasive (PC). The IFU and IDU are described in greater detail below. The ISU manages the out-of-order execution with concepts similar to those of the IBM POWER7® processor [16]. It maps the logical registers (LREGs) specified within each instruction to physical registers and assigns a new physical register to each target register. Ready-to-issue instructions are age ranked via an age matrix, based on the program order, and the oldest ready-to-fetch instruction for each execution type (fixed point, floating point, and load/store) issues to that execution unit. The ISU also manages in-order group completion; a group completes when the prior group has completed and all uops in this group have finished execution. In addition, at completion, the associated transient register mapper state is committed to an architected mapper. The two FXUs execute fixed-point instructions, perform store data moves to the STQ inside the LSU, generate condition codes, and resolve branch directions. The BFU executes all nondecimal floating-point instructions and fixed-point divides. The DFU executes all decimal floating-point instructions and the arithmetic piece of storage–storage (fixed point) decimal instructions. The RU handles the time of day, much of the processor state checkpointing, global error collection, and error recovery sequencing. The XU maintains a second-level TLB and performs address translation. The shared coprocessor subsystem, or COP, consists of two compression/decompression engines, which execute compression/decompression instructions, two small caches for table data, one data decryption/encryption (crypto) engine, and an adaptive lossless data compression engine that performs database index compression. The core PC unit controls trace, logic built-in self-test, scan, and clock operations. Other aspects of the z196 processor microarchitecture were published in *IEEE Micro* [17].

**Instruction fetching and branch prediction**

The IFU is designed to deliver instructions as fast and as speculative-path-accurate as possible into the IQ of the ISU via the IDU. The greater the backlog of instructions in the IQ, the greater the opportunity to find instructions that can be issued out of order given an in-order dependence stall. The z10 IFU design (which is described in detail in [2]) was designed on a foundation that was meant, with enhancements, to support an out-of-order bandwidth machine. The foundation of the z10 design is a branch predictor, which, based on a branch address, predicts the direction [branch history table (BHT)] and the target [branch target buffer (BTB)] of the given branch. This predictor has the capacity of 10,240 branches. Additionally, for those branches whose targets go to more than a single location or whose direction is based on a prior code direction, there exists a 2,048-entry changing target buffer (CTB) and a 512-entry pattern history table (PHT), respectively, to assist accuracy. To achieve the full potential of an out-of-order design, beyond keeping the IQ full, wrong branches resolution and direction need to be minimized, as they are magnified in an out-of-order design because the amount of time needed to execute the correct path has decreased, but the amount of time to recover from a wrong branch has increased with regard to the added out-of-order stages in the pipeline (see Figure 4).

The instruction delivery rate and accuracy improvements have been accomplished via large-page I-TLB, as described in the section “Processor cache hierarchy,” third-level direction prediction, $8 \times$ larger PHT with tagging, modified filtered PHT algorithm, CTB index tweaking, speculative-side copy prediction updates, address mode prediction, lower latency branch prediction, improved branch prediction synchronization, and 3-wide instruction processing.

The low-latency delivery of instructions to the IDU first requires instructions to be in the L1 I-cache. The BTB (and BHT) not only acts as a target and direction predictor of branches but also as an instruction prefetcher. By having the BTB asynchronously run to the I-cache and having the BTB size capacity exceed the footprint of the L1 I-cache, the BTB provides addresses of taken branches to the I-cache, which triggers speculative path fetching beyond the L1. Technology scaling and BTB latency reduction complexities have resulted in the BTB containing 8,192 branches.

With the reduction of branches stored in the BTB, an additional level of prediction capacity has been added to the design. The additional level is the surprise-guess-override (SGO) predictor, which is a direction-only predictor that is accessed in parallel to the I-cache. The SGO predictor is a 1-bit predictor for 32,768 branches, which is applied only to branches that were not predicted by the BTB and were to be guessed, not taken, as a function of the opcode. The SGO is updated whenever the most significant bit of the 2-bit saturating BHT is modified.

A branch predictor that asynchronously runs to the I-cache is valuable to performance since it allows the branch predictor to run ahead of the I-cache to: 1) act as an instruction prefetcher for I-cache misses as described above; and 2) stitch together instruction streams such that the branch to target redirected penalty can be, in the best case,
eliminated. In addition, in an asynchronous branch predictor with respect to the I-cache, it is possible for the predictor to get behind the instruction stream based on the number of branches it has to process. To keep the branch predictor ahead of the instruction stream, branch double pumping and the branch latency array (BLA) have been introduced. Branch double pumping allows the BTB to predict a not-taken branch, along with a sequential not-taken/taken branch with a single access to the BTB, whereas indexing into the PHT accounts for the different pattern as per these two branches. The BLA provides an enhanced means to “lockstep” the branch prediction logic and the instructions being sent to the IDU from the IFU. When the branch prediction logic is behind the instruction flow from the I-cache, the BLA provides the means to stall the instruction stream to allow the extra cycles needed for the branch prediction to provide the missing or latent dynamic prediction, thereby greatly reducing the potential to encounter a wrong branch at a later point in the pipeline.

The instruction flow from the IFU to the IDU has been increased from two to three instructions per cycle. The super basic block buffer introduced in z10 retains a similar instruction flow pattern for these three instructions. Given an I-cache and BTB that can change behavior in the time domain, attention has been given to the flow pattern so that grouping patterns can remain repetitive, thereby allowing the software compiler to best optimize for the instruction grouping that takes place downstream.

**Instruction decoding and grouping**

The IDU decodes and dispatches up to three instructions in parallel. Stages D1 and D2, as shown in Figure 4, receive up to three instruction texts (referred to as a “clump” of instructions) from instruction fetch and generate basic decode information required for dispatch group formation (e.g., alone, cracked, expanded, first, and last), dispatch, and redirect instruction fetch. Branch instructions, including millicoded instructions, are identified and cross checked with the branch prediction information associated with these branch instructions. Surprise branches (a branch that is not predicted by the branch prediction logic) and wrong prediction instructions (e.g., a nonbranch instruction predicted as a branch due to aliasing) are identified, and then instruction fetching is redirected to the correct stream. In addition, instructions younger than the instruction causing redirection are dropped and prevented from flowing down the pipeline. As an example to a grouping rule, branch instructions, explicit program status word update instructions (e.g., SAC, SPKA, etc.), and architecture serializer instructions are dispatched last in the group since these instructions may result in a flush after being completed, and a flush in the middle of a group is not supported (no partial group completion is allowed). A new dispatch operation mode, i.e., scalar mode, is introduced if a set of grouped instructions cannot complete together due to reasons such as a program store compare (PSC) when the storing instruction and its victim are grouped together, or due to unsupported store-hit-load bypass between a store and a load in the same dispatched group. The request to go to the scalar mode for the next three dispatched instructions is generated from the execution units and is accompanied by a flush request to the whole group. After the flush, the IDU dispatches the next three instructions in the scalar mode (each instruction is in a group by itself).

Stages G1 and G2 of the IDU are the grouping and dispatching sections, where up to three instructions or uops of a cracked instruction are dispatched in a given cycle to the ISU. Each dispatch pipe specifies up to four architected LREGs. One of the LREGs can be simultaneously identified as a register source and as a register target, whereas the other three LREGs can be identified only as register sources. This LREG arrangement covers most of the frequently used RX, RS, and RR type of instructions without the need to crack them at dispatch. For example, a commonly used RX-arithmetic operation [e.g., AG R1, D2 (B2, X2)] dispatches without cracking with LREG0, set to operand R1, specifying a register source and a register target, and the rest of the LREGs specifying register sources for index register (X2), base register (B2), and access register (B2). Each LREG assignment can be for general-purpose registers (GPRs), floating-point registers (FPRs), access registers (ARs), scratch GPRs, scratch FPRs, millicode GPRs, or millicode ARs. Each dispatch pipe can read and/or write an architected or scratch condition code, assign a load queue entry, an SQ entry, and/or a store buffer entry (or entries).

Table 4 shows an example of a nine-instruction stream flow from D1 stage to G2 stage, with three instructions [branch on count (BCT), load multiple (LM), and move character (MVC)] requiring cracking and/or expansion. Clumps of three instructions provided from IFU are decoded together, as shown in cycles 1, 2, and 3. Cracked and expanded instructions go through an overflow queue located between D2 and G1. The overflow queue allows an instruction to be replicated from one D2 pipe to all the G1 pipe stages. The first clump of three instructions (RX-add AG, load LG, branch BRC) is decoded in D2 and moved to G1 bypassing the overflow queue cycle and eventually dispatched in cycle 3. The second clump of instructions (BCT, LM, MVC) consists of three cracked instructions that need to be dispatched alone. Instructions of clump 2 decode together and move to the overflow queue in cycle 3. Cracked instruction BCT is then moved to the G1 stage, and then to the G2 stage where the two uops of BCT are dispatched in cycle 5. The LM instruction in this example is expanded to three dispatch groups, with a total of seven uops. The LM instruction is moved from the overflow queue to all of the three pipes in the G1 stage and spins in the G2 stage for three cycles (cycles 6, 7, and 8) for its three group
dispatches. As LM spins in the G2 stage, the cracked MVC moves to the G1 stage and stays there until the last group of LM is dispatched. The three uops of MVC are dispatched in cycle 9. Clump 3 from the IFU (AR, BC, and LG) decodes together in cycle 3 but cannot be dispatched together because the middle is a branch instruction that ends a group.

The z196 processor executes approximately 1,100 instructions. About 220 of these instructions are either cracked or expanded based on the following: instruction operand length field (instruction text 8:15) [18], instruction base and displacement fields (text bits 16:19, 30:31, 32:35, and 36:47) [19], instruction R2-field (instruction text bits 12:15 or text bits 28:31) [20], history-based operand store compare prediction (used for store multiple instructions) [21], as well as control and program status word bits and hardware disables [22]. In addition to the 220 hardware-executed cracked and expanded instructions, there are approximately 200 complex instructions that are forced to be executed in millicode (similar to vertical microcode) and thus require expansion to three groups encompassing nine uops.

The IDU has a postdispatch decode area that runs in parallel with ISU pipeline stages M0–M2, S0–S1, as shown in Figure 4. This section decodes dataflow control bits required for execution units FXU, BFU, and DFU and approximately 100+ bits of decode information for the LSU. The postdispatch decode bits are written to the IQs and are read and transmitted to the execution units at the issue time of the corresponding instruction or uop. The IQ structure in the ISU (physically consistent of many queues) is written from three different stages. The first write path is from the M2 stage where information pertaining to physical registers’ sources and target is written. The second write path is from postdispatch decodes, and the third write path is from the LSU address generation dataflow. The third write path is what made shift and rotate instructions noncracked and single-cycle executed. A shift or rotate instruction has the rotate amount specified as the least significant bits of operand2 address (B2 + D2). The lower 6 bits of operand2 address forms the rotate amount and the FXU mask required for such operations. The shift or rotate instructions are dual issued (an LSU issue followed by an FXU issue) from a single IQ entry, and the control information is passed through the IQ between the two issues.

### Conclusion

The IBM zEnterprise 196 system represents a major advancement in multiple aspects over its z10 predecessor. It features up to 96 processors operating at a sustained speed of 5.2 GHz distributed across four fully connected

### Table 4

| IDU D1 to G2 instruction flow for various instructions. (Note: BCT-uop0: (R1 ← R1 − 1) + set sCC; BCT-uop1: (branch targ ← B2 + X2 + D2) + branch resolution; LM-uop0: (R1 ← Mem|B2 + D2|); LM-uop1: (R1 + 1 ← Mem|B2 + D2 + 4|); LM-uop2: (R1 + 2 ← Mem|B2 + D2 + 8|); LM-uop6: (R1 + 6 ← Mem|B2 + D2 + 24|); MVC-uop0: (sGR ← Mem|B2 + D2|); MVC-uop1: (Mem|B1 + D1| ← sGR); MVC-uop2: (Pretest Mem|B1 + D1 + L|); sGR: scratch GPR; and sCC: scratch condition code.) |
|---|---|---|---|---|---|---|---|
| Pipe0-D2 | AG | BCT | AR |
| Pipe1-D2 | LG | LM | BC |
| Pipe2-D2 | BRC | MVC | LG |
| Overflow queue | — | BCT | LM | MVC | AR | AR | AR | LG |
| 2 deep × 3 wide | — | LM | MVC | AR | BC | BC | BC |
| | — | MVC | AR | BC | LG | LG | LG | BC | LG |
| Pipe0-G1 | AG | — | BCT | LM | MVC | MVC | MVC | AR |
| Pipe1-G1 | LG | — | BCT | LM | MVC | MVC | MVC | BC |
| Pipe2-G2 | BRC | — | — | LM | MVC | MVC | MVC | — |
| Pipe0-G2 | AG | BCT-uop0 | LM-uop0 | LM-uop3 | LM-uop6 | MVC-uop0 |
| Pipe1-G2 | LG | BCT-uop1 | LM-uop1 | LM-uop4 | — | MVC-uop1 |
| Pipe2-G2 | BRC | — | LM-uop2 | LM-uop5 | — | MVC-uop2 |
processing nodes, 3 TB of physical memory, up to 24 I/O hubs, and a new four-level cache design that provides up to 60% more box capacity than the preceding z10 platform. The combination of the fast core, aggressive microarchitecture, memory subsystems structure, system reliability, and availability in the zEnterprise 196 system provides an environment for managing the world’s most demanding workloads.

Acknowledgments
The development of the z196 chip was a major investment by IBM that involved considerable team effort. Numerous engineers, managers, and support staff at various IBM laboratories across the world have contributed to various phases of this design. The authors would like to specially thank Thomas McPherson, Shukla Kapur, Kelly Hopmeier, and the management team for their support and guidance throughout this development process. In addition to the authors, other key microarchitecture leads and contributors included Gregory Alexander, Khary Alexander, Ekaterina Ambroladze, Brian Barrick, Michael Billeci, James Bonanno, Tim Bronson, Steven Carlough, Kevin Calabrese, Mike Chow, Bing Chu, Adam Collura, Lawrence Curley, Deanna Dunn, Garrett Drapala, Lee Eisen, Thomas Fuchs, Ute Gaertner, Guenter Gerwig, Bruce Giamei, Rebecca Gott, Willm Hinrich, Jonathan Hsieh, David Hutton, Christine Jones, Edward Kamiinski Jr, Kenneth Klapproth, Matthias Klein, Barry Krumm, Thomas Koehler, Wen Li, Frank Lehnert, William Lewis, Edward Malley, Uli Mayer, Arthur O’Neill, Martin Recktenwald, John Rell, Anthony Saporito, David Schroter, Eric Schwarz, Robert Sonnelitter, Gary Strait, Son Dao Trong, Aaron Tsai, Sherylly Veneracion, and Vesselin Papazova. There are many others who cannot be mentioned due to space limitations, but their contributions and dedication were integral to the successful launch of this platform and their efforts are greatly appreciated.

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

References
Received January 15, 2011; accepted for publication
July 25, 2011

Fadi Busaba  IBM Systems and Technology Group, Poughkeepsie, NY 12601 USA (busaba@us.ibm.com). Dr. Busaba received his Ph.D. degree in computer engineering from North Carolina A&T State University in 1993 and joined the Department of Electrical Engineering at North Carolina A&T State University as assistant professor. He joined IBM in 1997 and worked on various aspects of zSeries® microprocessor design and development from high-level design concepts through lab support. He has held various leadership positions and led the fixed point unit and instruction decode and grouping unit, including address generation and operand fetch controls. Dr. Busaba has reached the eleventh IBM Invention Plateau and is the recipient of the IBM Master Inventor award.

Michael A. Blake  IBM Systems and Technology Group, Poughkeepsie, NY 12601 USA (mablake@us.ibm.com). Mr. Blake joined IBM in 1981 after receiving his B.S. degree in electrical engineering from Rensselaer Polytechnic Institute. He has held hardware design positions on numerous System z servers throughout his career, including his role as the chip design lead and chief architect for the z196 system controller chip. He is currently a Senior Technical Staff Member in server hardware development working on future System z servers. Mr. Blake has received four IBM Outstanding Technical Achievement awards, including one for his work on the z196 system controller chip, as well as two IBM Outstanding Innovation awards and an IBM Corporate Award for his work on the z900 system controller element. He currently holds 15 U.S. patents and has 11 patents pending.

Brian Curran  IBM Systems and Technology Group, Poughkeepsie, NY 12601 USA (curranb@us.ibm.com). Mr. Curran is a Distinguished Engineer in the IBM Systems and Technology Group. He joined IBM in 1984 after receiving a B.S. degree in electrical engineering from the University of Wisconsin–Madison. He has been a member of 11 processor/microprocessor design teams serving in various capacities: logic design, circuit design, and microarchitect. He was the z196 microprocessor design lead and is currently leading the design of a future System z microprocessor.

Michael Fee  IBM Systems and Technology Group, Poughkeepsie, NY 12601 USA (fee@us.ibm.com). Mr. Fee is a Senior Engineer in zSeries nest custom hardware design. He was the lead microarchitect of the IBM zEnterprise 196 24-MB shared level-3 DRAM cache as well as overall nest functions of this 96-way system. His responsibilities varied through the development cycle from creating and evaluating new concepts to providing detailed specifications for hardware operations. He worked with verification teams to ensure proper function and to achieve the 5.2-GHz target frequency as well as performance targets. He defined hardware support for system-wide operations such as processor quiescing, memory move, and storage protect features and improved support for performance measuring to help design better future systems. He has received IBM Outstanding Technical Achievement and IBM Outstanding Innovation awards for his contributions to the G4 through G7 and z10 systems and has recently received his 11th IBM Invention Plateau. Mr. Fee has worked exclusively on shared-cache system designs for IBM since receiving his B.S. degree in electrical engineering from Manhattan College in 1989.

Christian Jacobi  IBM Systems and Technology Group, Schoenaicherstrasse, Boeblingen, Germany (cjacobi@de.ibm.com). Dr. Jacobi received his M.S. and Ph.D. degrees in computer science from Saarland University, Germany, in 1999 and 2002, respectively. He joined IBM in 2002, working on floating point implementation and formal verification for various IBM processors. For the IBM System z10 mainframe, he led the L1.5 cache unit team. From 2007 to 2010, he was on international assignment to Poughkeepsie, New York, where he led the design team of the load-store unit for the z196 mainframe. Dr. Jacobi is now responsible for the cache design of IBM mainframe processors and for future architecture extensions.

Pak-Kin Mak  IBM Systems and Technology Group, Poughkeepsie, NY 12601 USA (pmak@us.ibm.com). Mr. Mak is the Nest Architect in the System z processor subsystem development team. He received his B.S. degree in electrical engineering from Polytechnic University of New York and his M.B.A degree from Union College. He joined IBM in 1981, working on his first cache design in the ES/3090 buffer control element, and has since been involved with high-performing cache hierarchy and symmetric multiprocessor designs. Mr. Mak is currently involved in developing future System z processor cache subsystem designs.

Brian R. Prasky  IBM Systems and Technology Group, Poughkeepsie, NY 12601 USA (brprasky@us.ibm.com). Mr. Prasky joined IBM in 1998 after receiving his M.S. degree in electrical and computer engineering from Carnegie Mellon University. With expertise in instruction flow, he has designed, architected, and/or led the instruction fetch unit over the last five generations. He is now additionally responsible for core performance. Mr. Prasky, a recipient of the IBM Master Inventor Award, is at the tenth IBM Invention Plateau.

Craig R. Walters  IBM Systems and Technology Group, Poughkeepsie, NY 12601 USA (walters1@us.ibm.com). Mr. Walters is a Senior Engineer. He joined IBM in 1999, working as a logic designer on the z990, z9®, z10, and z196 system controller designs, with a focus on cache management controllers and multinode transport/coherence protocols. In 2005, he joined the System z Processor Cache Subsystem Performance team, focusing on the z10, z196, and future system designs. He is an author or coauthor of 22 granted or pending patents. He holds a B.S. degree in electrical engineering from the New Jersey Institute of Technology and an M.B.A. degree from the State University of New York at New Paltz.