An Oldest-First Selection Logic Implementation for Non-Compacting Issue Queues

Alper Buyuktosunoglu, Ali El-Moursy, David H. Albonesi
Department of Electrical and Computer Engineering
University of Rochester

Abstract— **Microprocessor power dissipation is a growing concern, so much so that it threatens to limit future performance improvements. A major consumer of microprocessor power is the issue queue. Many microprocessors, such as the Compaq Alpha 21264 and IBM POWER4^{TM}, use a compacting latch-based issue queue design which has the advantage of simplicity of design and verification. The disadvantage of this structure, however, is its high power dissipation.**

In this paper, we propose a new selection logic implementation in conjunction with a non-compacting issue queue. This scheme achieves comparable delays to the existing position-based selection approach used for compacting issue queues, yet results in far less power with a small performance loss.

I. Introduction

Power dissipation is a serious constraint for designers of high-end microprocessors [2], [6], [12]. Many complex trade-offs must be made to achieve the goal of power-efficient, yet high performance design. As more processor performance is needed with the given power dissipation constraints, it becomes important to consider how these different areas affect one another in order to find the optimum design. This overlap is particularly noticeable in the issue queues of microprocessors.

Modern dynamic super-scalar microprocessors use out-of-order issue and execution in order to extract greater instruction-level parallelism (ILP) from common programs [8], [11], [14]. A key hardware structure in out-of-order designs is the issue queue. Figure 1 shows a pipeline overview of the dynamic scheduling path as well as the issue queue organization and operation. Here, each functional unit has its own dedicated issue queue, although in some designs, some of the functional units may share the same issue queue. The issue queue holds decoded and renamed instructions until they issue to appropriate functional units. The size of the issue queue represents the window of instructions that may be eligible for issue (or woken up) when both of its source operands have been produced and an appropriate functional unit has become available. The selection logic determines which instructions (up to the maximum issue width of processor) should issue out of these woken up in a given cycle. Wake-up and select together constitute an atomic operation. Atomic implies that the entire operation must finish before the wake-up and select operations for dependent instructions can begin. Thus, if dependent instructions are to be executed in consecutive cycles (necessary for achieving the highest performance) the issue queue logic (scheduling) performs this operation in one cycle. After an instruction is chosen for execution, a tag associated with the instruction destination register is broadcast over the tag buses to all entries in the issue queues. This tag broadcast signals dependent instructions that the instruction's result will be available soon (wake-up logic). After an instruction completes its execution, it broadcasts its result over the result bus to the register file and to any dependent instructions starting execution. The dependent instructions read register values from the register file or receives them from the bypass path.

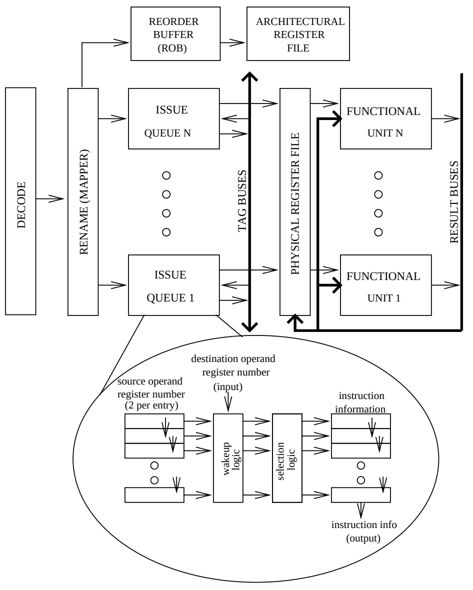

Fig. 1. Pipeline overview of the dynamic scheduling path and issue queue organization and operation.

Although crucial for achieving high performance in out-of-order processors, the issue queues are often a major contributor to the overall power consumption of the chip, potentially affecting both thermal issues related to hot spots and energy issues related to battery life. Figure 2 shows the power breakdown for the Compaq Alpha 21264 and also depicts the power breakdown of the Ibox unit within this processor. According to these figures, 50% of the Ibox power is consumed in the queue whereas 23% is consumed in the mapper and early datapath [13].

Current generation high-end processors like the IBM POWER4 [11] are performance-driven designs where power limits are still below the 0.5 watts/mm^2 power density limit afforded by the package/cooling solution of choice in the server markets targeted by such processors. Figure 3 shows

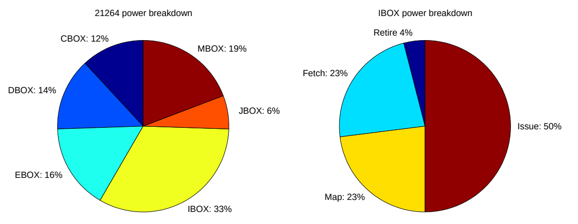

Fig. 2. Compaq Alpha 21264 and Ibox power breakdown. CBOX:BIU, Data & Control Busses; DBOX: Dcache, EBOX: Integer Units; IBOX: Integer Mapper & Queue, FP Mapper & Queue, Instruction Data Path; JBOX: Instruction Cache; MBOX: Memory Controller.

the power density across some of the major units of a simulated processor that is similar to a POWER4 core [1]. We note that although on a unit basis the power density numbers are under 0.5 watts/mm^2, there are smaller hot-spots, like the issue queues that are above the limit without any clock-gating assumptions.

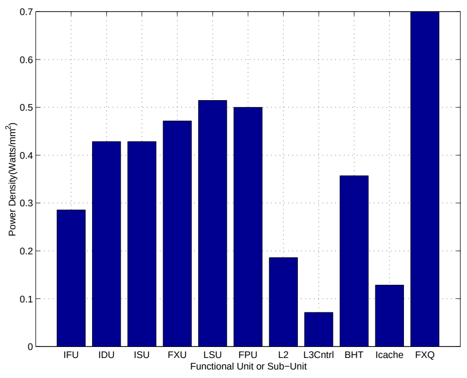

Fig. 3. IBM POWER4 power density profile across major units and sub-units. IDU: Instruction Decode Unit; FXU: Fixed Point Unit; IFU: Instruction Fetch Unit; ISU: Instruction Sequencing Unit; LSU:Load-Store Unit: includes Level1 (L1) Data Cache; FPU: Floating Point Unit; L2: Level2 (L2) Cache; L3 Cntrl: Level3 (L3) Cache Control; BHT: Branch History Table; Icache: Instruction Cache; FXQ: Fixed-Point Issue Queue.

The Compaq Alpha 21264 and IBM POWER4 implement a compacting *latch-based* issue queue in which each entry consists of a series of latches [1], [5]. Figure 4 shows a general latch-based issue queue design similar to that used in these processors. Each bit of each entry consists of a latch and a multiplexer as well as comparators (not shown in this figure) for the source operand IDs. Each entry feeds-forward to the next queue entry, with the multiplexer used to either hold the current latch contents or load the latch with the contents of the next entry. The design shown in Figure 4 can load up to four dispatched instructions in a cycle into the uppermost unused queue entries. The queue is *compacting* in that the outputs of each entry feed-forward to the next entry to enable the filling of holes created by instruction issue. New instructions are always added to the tail position of the queue. In this manner, the queue maintains an oldest to youngest program order within the queue. This simplifies the implementation of an oldest-first issue priority scheme as well as the squashing of queued instructions that are in the shadow of a mis-predicted branch.

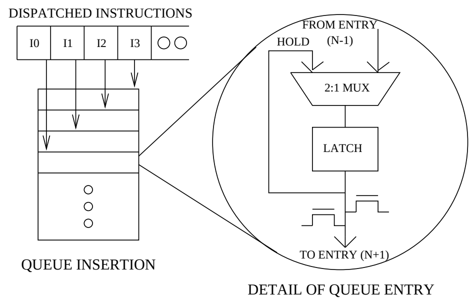

Fig. 4. Queue insertion and detail of queue entry.

However, the high price of this approach is its power consumption. Compaction entails moving instructions around in the queue every cycle and depending on instruction word width may therefore be a source of considerable power consumption. Each time an instruction is issued, all entries are shifted down to fill the hole, resulting in all of these latches being clocked. Because lower entries have issue priority over upper entries, instructions often issue from the lower positions, resulting in a large number of shifts and therefore, a large amount of power dissipation.

To eliminate these power-hungry operations, we can make the issue queue non-compacting. In a non-compacting queue, holes that result from an instruction issue from a particular entry are not immediately filled. Rather, these holes remain until a new entry is dispatched into the queue. At this point, the holes are filled in priority order from bottom to top. The obvious disadvantage of a non-compacting queue is that oldest to youngest priority order of instructions in the queue is lost. So, although a compaction strategy is unsuitable for low power operation, it may be critical to achieving good performance with a simple position-based selection logic.

In this paper, we propose a new oldest-first selection logic implementation for overcoming the drawback (lost instruction order) of non-compacting queues. This implementation relies on the instruction sequence numbers, or the reorder buffer (ROB) numbers that typically tag each dispatched instruction. Our analysis with this circuit structure suggests that comparable delays with existing selection logic implementations can be achieved, while still providing the power savings of a non-compacting queue, with a small cycles per instruction (CPI) performance degradation.

II. Non-Compacting Issue Queue

The obvious disadvantage of a non-compacting queue is that the oldest to youngest priority order of the instructions in the queue is lost. Thus, the use of a simple position-based selection mechanism like that described in [10] will not give priority to older instructions as in the compacting design. Figure 5 shows a simple example of a four-entry issue queue from which a single instruction can issue each cycle and in which lower instructions have priority over upper ones. Initially, both the compacting and non-compacting queues maintain the oldest to youngest order

(A, B, C, D). When instruction B issues from the compacting queue, instructions C and D are shifted down and a new instruction (E) is put into the uppermost location. By contrast, in the non-compacting queue, instruction E is placed in the position formerly occupied by instruction B, giving E higher priority than instructions C and D. If both of the operands for instruction E are ready when it is inserted into the queue, then in the non-compacting queue it will issue in the next cycle, ahead of the older instruction C. Thus, these older instructions C and D may stay in the queue for a significant period of time, which may have a non-negligible impact on CPI performance (shown in Section V).

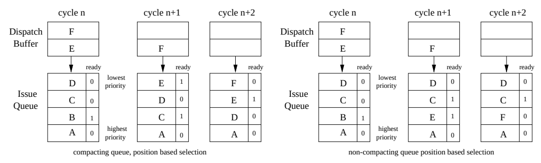
cycle n
cycle n+1
cycle n+2
Dispatch Buffer
Issue Queue
ready
lowest priority
highest priority
compacting queue, position based selection
non-compacting queue position based selection

Fig. 5. An example showing the differences between compaction and non-compaction with position-based selection.

Yet another alternative to a latch-based non-compacting issue queue is a CAM/RAM-based issue queue which does not support compaction. The baseline latch-based issue queue is redesigned as a CAM/RAM structure in which the source operand numbers are placed in the CAM structure and the remaining instruction information is placed in the RAM structure (shown in Figure 6). Because of the lower power of CAM/RAM logic relative to random logic, the CAM/RAM-based issue queue has the potential to reduce the average power dissipation of the queue.

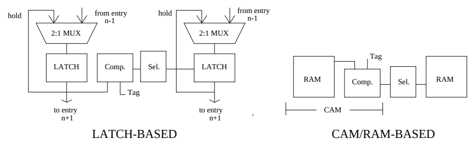

Fig. 6. Latch-based (with compaction) and CAM/RAM-based issue queues.

III. Position-Based Selection Logic

Since the number of ready instructions in the issue queue may be greater than the number of functional units available, some form of selection logic is required to select the instructions for execution from a ready pool of instructions. A position-based selection policy based on a mix of static and domino gates is proposed in [10]. Figure 7 shows the arbitration logic which is structured as a decision tree which must be traversed both ways. The inputs to the selection logic are the request signals, each of which is raised when all operands of the corresponding instruction are available. The outputs of the selection logic are the grant signals, one per request signal. Once an instruction is granted, it can be issued to the functional unit and the issue queue entry it occupies is freed.

The selection logic shown in Figure 7 works in two phases. In the first phase, the request signals are propagated up the tree. Each cell raises the *anyreq* signal if any of its input request signals is high. This also raises the input request signal of its parent arbiter cell. It is therefore possible that at the root cell one or more input request signals are high if there are one or more instructions that are ready. The root cell then grants the functional unit to one of its children by raising one of its grant outputs. This triggers the second phase in which the grant signal is propagated down the tree to the instruction that will be selected. The selection policy implemented in this structure is static and is strictly based on location of the instruction in the queue. Due to compaction, the bottom entries which are older instructions have the highest priority in selection.

The approach uses static arbitration gates in segments of four queue entries. The arbitration gate is structured as an AND function with a maximum fan-in of five for the lowest priority grant output signal. Domino OR gates are used to propagate request signals up the tree to provide the root node with information of which segments of the queue have ready instructions in them. The static arbitration gates then propagate the root nodes decision back to the queue entries.

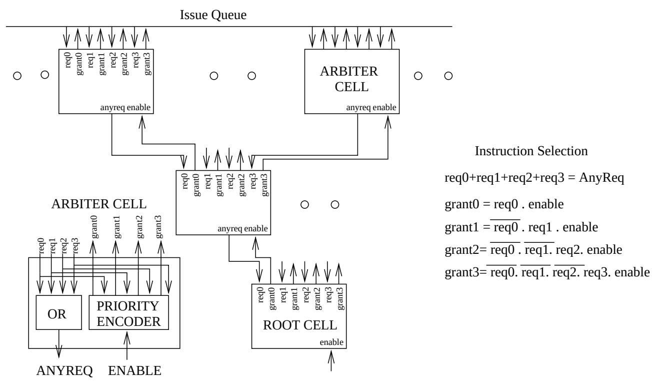
Issue Queue
req0 grant0 req1 grant1 req2 grant2 req3 grant3
anyreq enable
ARBITER CELL
OR
PRIORITY ENCODER
ANYREQ
ENABLE
ROOT CELL
enable
Instruction Selection
req0+req1+req2+req3 = AnyReq
grant0 = req0 . enable
grant1 = $\overline{\text{req0}}$. req1 . enable
grant2= $\overline{\text{req0}}$. $\overline{\text{req1}}$. req2. enable
grant3= $\overline{\text{req0}}$. $\overline{\text{req1}}$. $\overline{\text{req2}}$. req3. enable

Fig. 7. Position-based selection logic.

IV. Oldest-First Selection Logic

To solve the problem of lost instruction ordering while maintaining much of the power-efficiency advantages of a non-compacting queue, the reorder buffer (ROB) numbers (sequence numbers) that typically tag each dispatched instruction can be used to identify oldest to youngest order. (The ROB commits instructions in program order by permitting an instruction to commit only if all the preceding instructions have committed.) However, a problem arises with this scheme due to the circular nature of the ROB which may be implemented as a RAM with head and tail pointers. For example, assume for simplicity an 8-entry ROB where the oldest instruction lies in location 111 and the youngest in 000. When an instruction commits, the head pointer of the ROB is decremented to point to the next entry. Similarly, the tail pointer is decremented when an instruction is dispatched. With such an implementation, the oldest instruction may no longer lie in location 111 in our working example, but in any location. In fact, the tail pointer may wrap around back to entry 111 such that newer entries (those nearest to the tail) may occupy a higher-numbered ROB entry than older entries [7]. When this occurs, the oldest-first selection scheme will no longer work properly.

This problem can be solved by adding an extra high-order sequence number bit which we call the *sorting bit* that is kept in the issue queue. As instructions are dispatched, they are allocated a sequence number consisting of their ROB entry number appended to a sorting bit of 0. These sequence numbers are stored with the entry in the issue queue. Whenever the ROB tail pointer wraps around to entry 111 in our example, all sorting bits are flash set to 1 in the issue queue. Newly dispatched instructions, however, including the one assigned to ROB entry 111, continue to receive a sorting bit of 0 in their sequence numbers. These steps, which are summarized in Figure 8, guarantee that these newly dispatched instructions will have a lower sequence number than prior (older) instructions already residing in the queue.

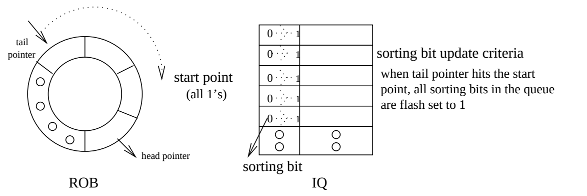
tail pointer
start point (all 1's)
head pointer
ROB
sorting bit update criteria
when tail pointer hits the start point, all sorting bits in the queue are flash set to 1
sorting bit
IQ

Fig. 8. Mechanism for updating the sorting bit in the issue queue.

Once the sorting bit adjustment is in place, older instructions can properly be selected from the ready instructions as follows. The most significant bit of the sequence numbers of all ready instructions are OR-ed together. If the result of the OR is 1, all ready instructions whose most significant bits are 0 will be removed from consideration. In the next step, the second most significant bit of the sequence numbers of all ready instructions that are still under consideration are OR-ed together. If the result of the OR is 1, all ready instructions still under consideration whose second most significant bits are 0, will be removed from consideration. The Nth step is the same as step 2, except the least significant bit of the sequence number is used. At the end of this step, all ready instructions will have been removed from consideration except for the oldest.

However, this OR-based arbitration mechanism requires a final linear O(N) chain from the highest order to the lowest order bit. This significantly increases the delay of the selection logic compared to the selection logic described by Palacharla [10] after 4 bits with a 32 entry queue. Note that for a processor that has up to 128 instructions (ROB of 128 entries) in flight, a full sequence number consists of 7 bits and a sorting bit. The lack of full age ordering with 4-bit sequence numbers results in a CPI degradation (shown in Section V), although it is an improvement over the CPI degradation incurred with no age ordering (position-based selection with non-compaction).

Figure 9 shows the 4-bit implementation of the oldest-first selection logic for a 32 entry queue. These four bits consist of three low order bits of the sequence number in addition to the sorting bit. (In Figure 9, A4 is the sorting bit and A3-A1 are the three least significant bits.) 32-input OR gates are implemented as 32-input domino logic. So for each stage the corresponding logic (OR gate, mux and AND gate) makes the decision of passing all the entries to the next stage (if they have the same age) or filter them (by passing only the greatest ones according to the comparison up to that point). In the scenario where ready instructions each have the same four bit value (tie situation), a priority encoder described in [5], [9] is used to grant one of those instructions. This priority encoder is an array of kill signals in which the one in a lower row discharges all upper row grants (not shown in Figure 9).

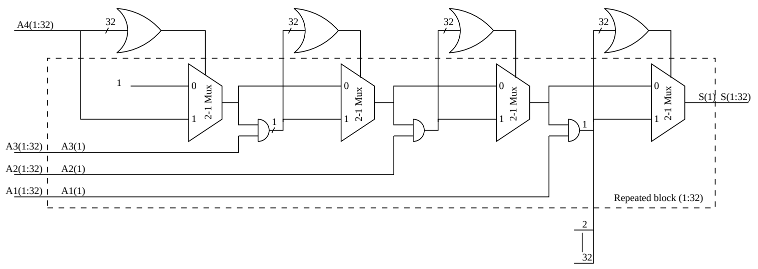

Fig. 9. Oldest-first selection logic.

V. Evaluation

A. Methodology

We perform circuit-level delay analysis and energy estimation using the IBM AS/X circuit simulator and next generation process parameters. The baseline latch-based issue queue and other circuit designs borrow from existing circuit libraries where appropriate. Both position-based selection logic and oldest-first selection logic implementations are optimized for speed, and then the corresponding energy measurements are performed. For the microarchitectural simulations, we used SimpleScalar-3.0 [3] to simulate an aggressive out-of-order super-scalar processor. The simulator has been modified to model separate integer and floating point queues. We focus on an integer issue queue with 32 entries in this paper.

B. Results

The trade-offs between a compacting and non-compacting issue queue are complex, as a degradation in CPI performance can potentially occur with non-compaction due to the lack of an oldest-first selection scheme. We modified SimpleScalar to model the holes created in a non-compacting issue queue, the filling of these holes with newly dispatched instructions, and a selection mechanism strictly based on location within the queue (rather than the oldest-first mechanism used by default). With such a scheme, older instructions may remain in the queue for a long time period, thereby delaying the completion of important dependence chains. The left-most bar in Figure 10 shows the CPI degradation for our six SPEC2000 integer benchmarks. The degradation is significant, around 8% for mcf and parser and 5.5% overall. The right-most bar shows the CPI degradation when the described oldest-first selection scheme is implemented by using four bit sequence number (including the sorting bit). On average, the partial oldest-first selection scheme reduces the CPI degradation from 5.5% to 2.3%.

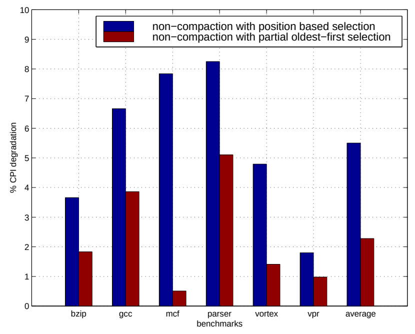

Fig. 10. CPI degradation incurred via non-compaction with position-based selection, and non-compaction with partial oldest-first selection.

One drawback with our selection scheme is that it dissipates 2.2 times more energy compared to the position-based selection logic under worst-case switching activity. However, depending on the number of ready instructions that are candidates for selection each cycle, the selection logic dissipates between 1.6%-5.1% of the total issue queue energy. Under such circumstances, the extra energy overhead of the proposed selection logic is well compensated by the extra energy savings. Figure 11 shows the overall energy savings with the proposed oldest-first selection logic for non-compacting issue queues (latch-based and CAM/RAM-based). The results are averaged over the six SPEC2000 benchmarks shown in Figure 10, with the relative energy calculations of different issue queue alternatives gathered from [4] and the extra energy overhead of oldest-first selection logic factored in. Latch-based and CAM/RAM-based designs with no compaction save considerable energy (30% for the latch-based design with the selection logic overhead at 5.1%) compared with the latch-based compacting issue queue with a small performance degradation and no cycle time impact.

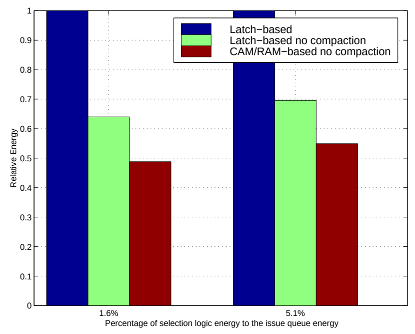

Fig. 11. Overall energy savings with latch-based no-compaction strategy and CAM/RAM-based issue queues compared to latch-based compacting issue queue with the proposed oldest-first selection logic.

VI. Conclusion

In this paper, we present a new selection logic scheme for non-compacting issue queues that achieves considerable power savings with a small performance degradation. Through a detailed quantitative comparison of our circuit technique with the well-known position-based selection logic used for compacting issue queues, we determine that this new selection logic meets the cycle time constraints of high clock rate microprocessors.

VII. Acknowledgments

This research was supported in part by DARPA/ITO under AFRL contract F29601-00-K-0182, NSF under grants CCR-9701915 and CCR-9811929, and by an IBM Faculty Partnership Award.

References

[1] P. Bose, D. M. Brooks, A. Buyuktosunoglu, P. W. Cook, K. Das, P. Emma, M. Gschwind, H. Jacobson, T. Karkhanis, S. E. Schuster, J. E. Smith, V. Srinivasan, V. Zyuban, D. H. Albonesi, S. Dwarkadas. Early-Stage Definition of LPX: A Low Power Issue-Execute Processor Prototype. Power Aware Computer Systems Workshop in conjunction with HPCA-8, February 2002.
[2] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta, P. Cook. Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors. IEEE Micro, pp. 26-44, November/December 2000.
[3] D. Burger and T. Austin. The Simplescalar toolset, version 2.0. Technical Report TR-97-1342, University of Wisconsin-Madison, June 1997.
[4] A. Buyuktosunoglu, D. H. Albonesi, S. E. Schuster, D. M. Brooks, P. Bose, P. W. Cook. Power-Efficient Issue Queue Design. Power Aware Computing, R. Graybill and R. Melhem(Eds), Kluwer Academic Publishers, Chapter 3, pp. 37-60, 2002.
[5] J. A. Farrell, T. C. Fischer. Issue Logic for a 600-MHz Out-of-Order Execution Microprocessor. IEEE Journal of Solid-State Circuits, 33(5): 707-712, May 1998.
[6] M. K. Gowan, L. L. Biro, D. B. Jackson. Power considerations in the design of the Alpha 21264 microprocessor. Design Automation Conference, June 1998.
[7] P. J. Jordan, B. R. Konigsburg, H. Q. Le, S. W. White. Data Processing System and Method for Using an Unique Identifier to Maintain an Age Relationship Between Executing Instructions. IBM Corporation, U.S. patent 5805849, 1997.
[8] R. Kessler. The Alpha 21264 microprocessor. IEEE Micro, 19(2): 24-36, March/April 1999.
[9] J. Leenstra, J. Pille, A. Mueller, W. Sauer, R. Sautter, D. Wendel. 1.8 GHz Instruction Window Buffer. International Symposium on Solid-State Circuits, 2001.
[10] S. Palacharla, N. Jouppi, J. E. Smith. Complexity-Effective Superscalar Processors. International Symposium on Computer Architecture, 1997.
[11] J. M. Tendler, S. Dodson, S. Fields, H. Le, B. Sinharoy. POWER4 System Architecture. IBM Journal Research and Development, 46(1): 5-27, 2002.
[12] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, F. Baez. Reducing power in high-performance microprocessors. Design Automation Conference, June 1998.
[13] K. Wilcox and S. Manne. Alpha Processors: A history of power issues and a look to the future. Proceedings of the Cool Chips Tutorial, in conjunction with MICRO-32, 1999.
[14] K. Yeager. The Mips R10000 Superscalar Microprocessor. IEEE Micro, 16(2):28-41, April 1996.