Reducing the Instruction Queue Leakage Power in Superscalar Processors

Houman Homayoun and Ted H. Szymanski
Department of Electrical and Computer Engineering
McMaster University, Hamilton Ontario L8S 4K1, Canada
email: homayh@mcmaster.ca, teds@mail.ece.mcmaster.ca

Abstract

Today's high performance processors operate in the GHz frequency range and dissipate approximately 100W of power. According to Moore's Law, in the next generation of microprocessors we expect an exponential increase in the total dissipated power. CMOS technology scaling has been the primary factor responsible for the increase in processor performance. A smaller feature size enables designers to increase the clock frequency and transistor count which significantly affects the processor performance. The drawback of such technology scaling is the leakage power dissipation. As the semiconductor technology scales down, the leakage (standby) power also increases exponentially and accounts for an increasing share of a processor's total power dissipation. This issue will become a serious problem in mobile hardware where applications may generate long periods of inactivity.

In this paper we take a step towards reducing the leakage power dissipation of the Instruction Queue which allows the out of order execution in a superscalar processor. This unit is responsible for up to 27% of total chip power dissipation in typical superscalar microprocessors. In particular, we reduce the leakage power in the thousands of comparator units in the Instruction Queue by applying a power gating technique. We rely on detecting the idle time in all comparators. We show that the comparators in the instruction queue stay idle for typically 50% of the total program execution time. This figure is based on the observation that the whole processor pipeline approaches an idle state when a combination of instruction and data cache misses occur. When such idle time is detected we apply power gating to turn off all the comparator units thereby eliminating the leakage power. Our results show that by power gating the comparators using our idle time detecting algorithm it is possible to reduce their leakage power dissipation by up to 95%.

Keywords: *Instruction Queue, Leakage Power, Processor Idle Period.*

1. Introduction

The instruction queue (or in brief IQ) of a superscalar processor is a complex structure which is dedicated to out-of-order execution. Due to its high complexity, the instruction queue is responsible for a significant amount of overall processor power dissipation. According to previous studies this amount varies between 25 to 27 percent of processor total power dissipation (dynamic and static power) [11] and [15].

There are four tasks involved in instruction queue stage [12]:

- Set an entry for a new dispatched instruction.
- Read an entry to issue instructions to functional units
- Wakeup instructions waiting in the IQ once a result is produced by a functional unit
- Select instructions for issue when the available instructions exceed the processor issue limit (which we refer to as issue width).

The main complexity of the instruction queue stems from the associative search during the wakeup process. During this stage newly-produced results are broadcasted from functional units to all entries in the instruction queue. Figure 1 shows the structure of the wakeup logic. Tag drive lines are responsible to broadcast the results to all instructions waiting in the IQ. Each instruction compares its operand tags with the broadcasted tags. If a match is detected the instruction source operand is marked as ready. Once all source operands of an entry are marked as ready (rdyL and rdyR flags) the instruction can enter the execution stage. Finally the OR logic which is responsible for OR-ing the results of comparators sets the rdyL/rdyR flags.

While in the past, low power IQ design techniques mostly targeted the dynamic power, reducing leakage power in the instruction queue has received relatively little consideration. In an 8 way superscalar processor with a 128 entry instruction queue and 12 bit operand tag, provided that each instruction has two source operands, there are approximately 25K one-bit comparators. The combination of comparators and broadcast buses when idle dissipate moderate leakage power using today's technology. In future generations of processors the instruction queue size, processor issue width and number of operand tag bits will increase, thus we expect a substantial increase in the number of comparators and associated busses and consequently their dissipated leakage power.

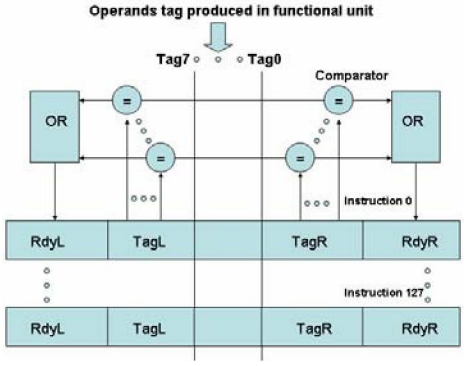

Figure 1: Instruction Queue in superscalar processor [12].

In addition, the fact that the transistor leakage power is increasing exponentially in future generations of CMOS technology reveals the importance of applying low power techniques to reduce the comparators leakage power [5], [7], [8] and [10].

2. Simulation Framework

Through this study we report for a representative subset of SPEC'2K benchmarks. We used both floating point (art, equake and ammp) and integer (vpr, gcc, mcf, gzip, bzip2, parser and twolf) programs from the SPEC CPU2000 suite compiled for the MIPS-like PISA architecture used by the Simplescalar v3.0 simulation tool set [3]. The benchmarks studied here include different programs including high and low IPC and those limited by memory and branch misprediction. We used GNU's gcc compiler. We simulated 200M instructions after skipping 200M instructions.

We detail the base processor model in Table 2. In our baseline architecture the fetch unit stalls until the instruction miss resolves and retrieves from the main memory.

Table 1: Base processor configuration.

Integer ALU	# 8	***Scheduler***	128 entries, RUU-like
FP ALU	# 8	***OOO Core***	any 8 instructions / cycle
Integer Multipliers/ Dividers	#4	***Fetch Unit***	Up to 8 instr./cycle. 64-Entry Fetch Buffer
FP Multipliers/ Dividers	#4	***L1 - Instruction Caches***	64K, 4-way SA, 32-byte blocks, 3 cycle hit latency
Instruction Fetch Queue	#32	***L1 - Data Caches***	32K, 2-way SA, 32-byte blocks, 3 cycle hit latency
Branch Predictor	2k Gshare bimodal w/selector	***Unified L2***	256K, 4-way SA, 64-byte blocks,16-cycle hit
Load/Store Queue Size	64	***Main Memory***	Infinite, 100 cycles
Reorder Buffer Size	128	***Memory Port***	#4

3. Motivation

Instruction cache misses (Imiss) and data cache misses (Dmiss) are the two major barriers in increasing processor throughput. When an instruction or data cache miss happens, the processor communicates with the main memory to fetch the instruction or data. This process takes hundreds of cycles during which the processor performance drops significantly. In Figure 2 we report the processor performance for each benchmark when any of instruction cache miss, data cache miss or a combination of both occurs (which we refer to the combination of both instruction and data cache misses as IDmiss). We measure performance from the time a cache miss happens until the information (data or instruction) is retrieved from the main memory (we refer to this period as cache miss interval). As figure 2 reports, across all benchmarks performance drops significantly during cache miss intervals. The ideal performance is 8 IPC and the average performance is 2 IPC which during cache miss interval drops to 0.15 IPC.

This considerable performance drop can be translated to the fact that during the cache miss interval there are many cycles in which no instructions issue to the functional units and no instructions enter processor pipeline; i.e. the processor pipeline stalls. Accordingly during such idle period the processor pipeline including instruction queue stalls until the information is retrieved from the main memory (we refer to this time as processor idle period). The fact that retrieving information from the main memory to instruction or data caches may require hundreds of clock cycles provides the motivation to apply power gating techniques and reduce leakage power in the instruction queue. However, identifying idle periods early enough with minimum extra hardware cost is a challenging problem. Moreover, reactivating the gated units soon enough is critical since stalling instructions in any pipeline stage could come with a performance penalty [7] and [9].

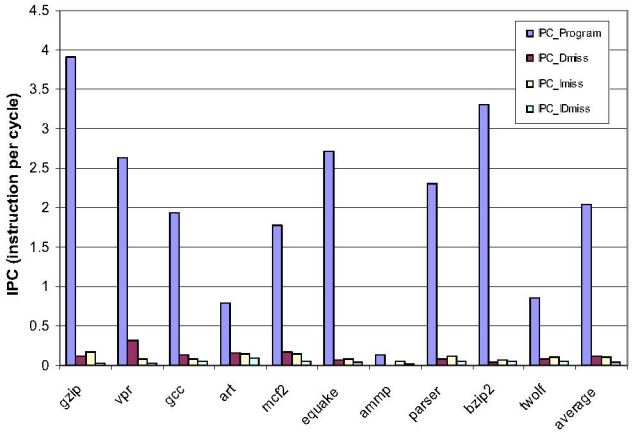

Figure 2. Processor performance for cache miss interval.

4. Detecting Comparators Idle Time

Based on the observation made in the previous section we propose an algorithm to detect IQ comparator idle times. Our algorithm to detect idle time relies on monitoring the instruction queue incoming and outgoing buffers. We monitor the dispatch buffer which sends the recently fetched instruction to the instruction queue. An "idle" dispatch buffer, for some consecutive cycles, indicates that an instruction cache miss has happened, provided that the number of consecutive idle cycles exceeds the branch misprediction penalty. We also monitor the issue buffer, which sends instructions to appropriate functional

units. An "idle" issue buffer, for some consecutive cycles, could indicate that a data cache miss has happened.

An idle dispatch and an idle issue buffer, for some consecutive clock cycles, is an indication that the IQ comparators are in an idle state provided that none of the functional units are busy. If any of the functional units are busy and the issue and dispatch buffer are idle the instruction queue comparators will not necessarily be idle as there might be some instruction in the IQ that are dependent on the instruction executing in the functional unit. In this case the IQ comparators are active to compare the forthcoming generated results in the functional unit with all available instructions operands in the instruction queue.

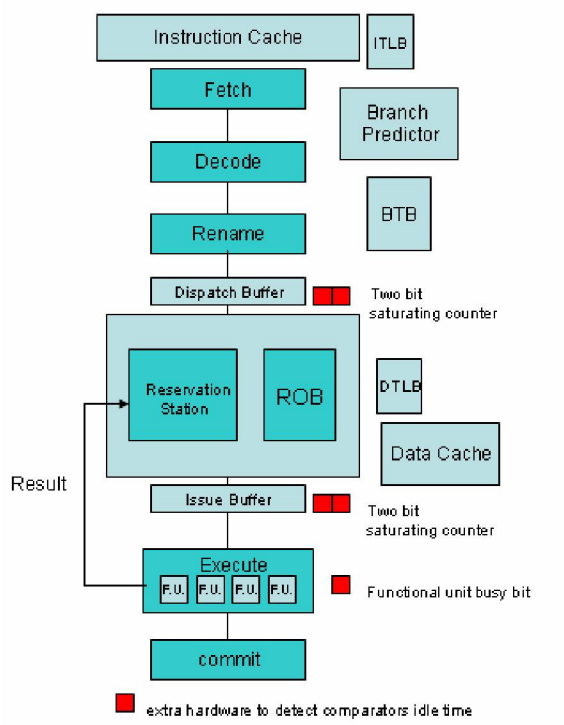

Figure 3. Proposed configuration to detect comparators idle time.

Figure 3 shows the configuration we propose. A two bit saturating counter is associated with each of the issue buffer and dispatch buffer. The counter increases by one per clock cycle when each buffer is empty otherwise the counter is reset. Also a busy bit is associated with the functional unit logic. The busy bit is set when any of the execution units is busy executing an instruction. An idle time is detected when the two counters are saturated and the busy bit it zero.

In figure 4 we report the percentage of cycles the IQ comparators remain idle using our detection algorithm for three different value of saturating counter threshold; 5 cycles, 10 cycles and 20 cycles. As mentioned earlier, this threshold should be more than the branch misprediction penalty to avoid detecting pipeline flush (due to branch misprediction) instead of IQ idle cycles. As it appears, increasing the saturating counter threshold doesn't have a major impact on the percentage of detected idle cycles. This observation indicates that our algorithm can detect long idle cycles; i.e. when the dispatch and issue buffer remain empty for enough consecutive cycles and no instruction is executing in the functional units, the IQ comparators stay idle for long cycles. To provide better insight, in figure 5 we report the average number of cycles the comparators remain idle. This figure shows that the average detected idle time using our algorithm is more than 120 cycles. This observation could explain why variation in saturating counter thresholds in figure 4 doesn't change the percentage of detected idle cycles.

In the next section we utilize the idle time detection algorithm to reduce comparator leakage power.

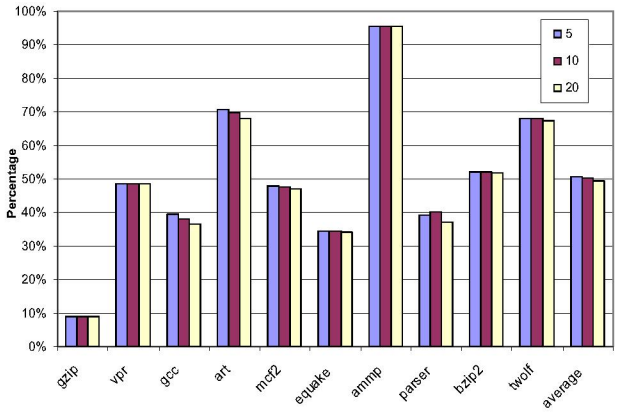

Figure 4. Percentage of execution cycles IQ comparators remains idle for different saturating counter threshold.

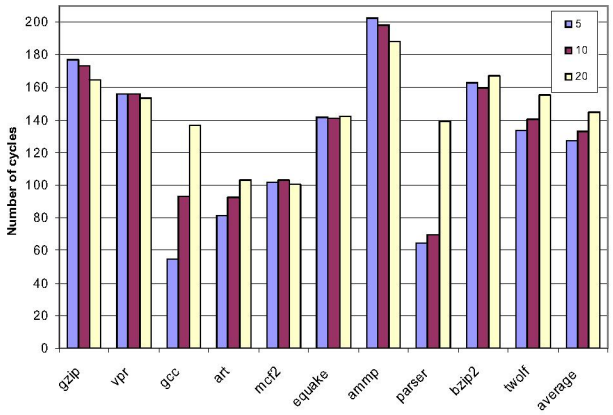

Figure 5. Average number of cycles IQ comparators stays idle for different saturating counter threshold.

5. Reducing Comparators Leakage Power by Detecting Their Idle Time

In the previous section we have shown that the comparators stay idle for long consecutive cycles. We also detected such idle cycles. In this section we use power gating to reduce the comparators leakage power.

Figure 6 shows a traditional pull down comparator logic used in various pipeline stages of a modern superscalar

processor including the instruction queue. To power gate the comparator logic we use a header transistor (which is referred to as a sleep transistor) to block the voltage supply from reaching the comparators. When the sleep transistor receives the power gating signal, the voltage at virtual Vcc starts decreasing. As the virtual Vcc decreases the leakage current reduces and the leakage power saving starts. The power gating signal is asserted when IQ idle time is detected. The power gating continues until the information (data or instruction) is retrieved from the main memory. Assuming deterministic memory access latency we can eliminate the timing overhead associated with turning on a power gated comparator unit.

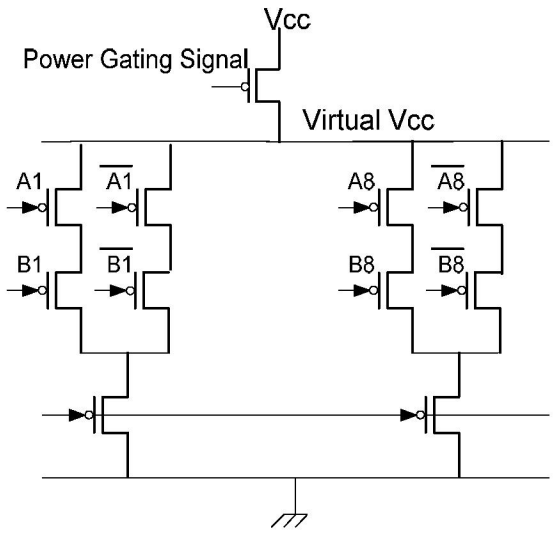

Figure 6. Power gating instruction queue comparators.

In figure 7 we show the leakage power saving for the IQ comparators when the idle detect saturating counter threshold is 5 cycles. The results are consistent with the results in figure 4. On average for around 50% of a program execution time the power can be gated which translate to on average 50% leakage power reduction. In the application program ammp we witnessed the highest leakage power savings, i.e. a 95% reduction, which is in consistent with its very low IPC mentioned in figure 2.

It should be noted that applying power gating comes with timing overhead. The power gating process includes three separate intervals. The first interval starts the moment we decide to power gate a unit and ends when the voltage supply is completely blocked. The second interval is the period where the unit is gated and therefore does not dissipate power. Power dissipation reduction depends on how often and for how long units stay in the second interval. We have to wakeup a unit as soon as its idle period ends. Turning on the voltage supply to wakeup a unit takes time. The third interval represents this timing overhead and is the time needed to reactivate a unit. While saving leakage power during the first and third intervals is possible, the power reduction benefits are mostly achievable when a unit is in the second interval. Hu et. al, provide a detailed explanation of the three intervals [6].

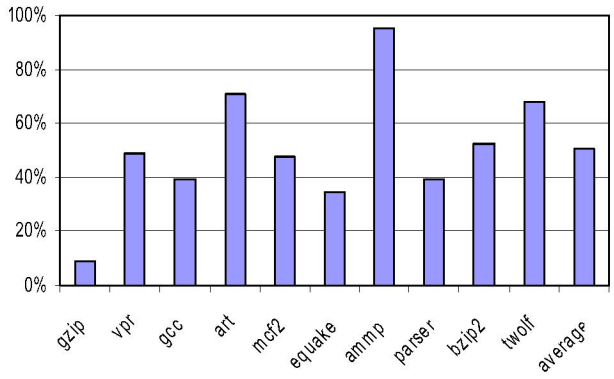

Figure 7. Percentage of execution cycles during which IQ comparators are power gated.

7. Related Work

Several approaches have been proposed to reduce the power dissipation of the associative search related to wakeup logic. Folegnani and Gonzalez [15] proposed a new scheme which avoids waking up empty entries in the instruction queue. Brown et al. introduced methods to remove the select logic from the critical path [18]. Homayoun and Baniasadi predicted "lazy instructions" which stay for long cycles in the instruction queue. They reduced IQ power dissipation by waking up lazy instruction every two cycles. Moreover they reduced wakeup power dissipation by reducing the fetch rate when the number of lazy instructions in the pipeline exceeds a dynamically decided threshold [2]. Canal and Gonzalez proposed a scheme, which schedules instructions based on their expected issue time [13] and [14]. Raasch et al. suggested adapting the issue queue size and exploiting partitioned issue queues to reduce the wakeup activity [19]. Brekelbaum et al., introduced a new scheduler, which exploits latency tolerant instructions in order to reduce implementation complexity [17]. Stark et al., used grandparent availability time to speculate wakeup [20]. Ernst et al., suggested a wakeup free scheduler which relied on predicting the instruction issue latency [21]. Hu et al., studied wakeup-free schedulers such as that proposed in [21] and explored how design constrains result in performance loss and suggested a model to eliminate some of those constrains [16]. To the best of our knowledge our work is the first attempt to reduce leakage power in the instruction queue. Previous studies including all aforementioned techniques try to reduce dynamic power in the instruction queue or leakage power in other processor structure such as functional units [1], [4] and [6].

6. Conclusion and Future Studies

In this paper we present a simple technique to reduce leakage power in the instruction queue of superscalar processor. We showed that it is possible to reduce leakage power in the comparators unit considerably.

While through out this work we only focused on the instruction queue it is also possible to detect idle time in other processor structures such as register renaming unit, functional unit and decode unit using our proposed technique. In future work we will extend our technique to the entire processor pipeline. We will also study variation in timing overhead associated with the power gating.

7. Acknowledgements

This work was supported by the Natural Sciences and Engineering Research Council of Canada, Discovery Grants Program, Canada Foundation for Innovation, New Opportunities Fund and Ontario Centre of Excellence Research Grants Program.

References

[1] H. Homayoun, K. F. Li and S. Rafatirad, "Functional Unit Power gating in Simultaneous Multithreaded Processors," The IEEE Pacific Rim Conference on Communications, Computers and Signal Processing. Aug. 2005.

[2] H. Homayoun, A. Baniasadi, "Using Lazy Instruction Prediction to Reduce Processor Wakeup Power Dissipation,", In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-2006).

[3] D. Burger, T. M. Austin and S. Bennett, "Evaluating Future Microprocessors: The SimpleScalar Tool Set," Technical Report CS-TR-96-1308, University of Wisconsin-Madison, July 1996.

[4] H. Homayoun and A. Baniasadi, "Analysis of Functional Unit Power Gating in Embedded Processors," IFIP International Conference on Very Large Scale Integration System on Chip. Oct. 2005.

[5] S. Borkar, "Design Challenges of Technology Scaling," IEEE Micro.vol. 19, pp. 23--29, July--Aug. 1999.

[6] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson and P. Bose, "Microarchitectural Techniques for Power Gating of Execution Units," International Symposium on Low Power Electronics and Design, 2004.

[7] J. A. Butts and G. S. Sohi, "A static power model for architects," In Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture, pages 191--201, Dec. 2000.

[8] O. S. Unsal and I. Koren, "System-Level Power-Aware Design Techniques in real-Time Systems," In proceedings of the IEEE, VOL. 91, NO. 7, July 2003.

[9] K. S. Khouri and N. K. Jha, "Leakage power analysis and reduction during behavioral synthesis," IEEE Transactions on Very Large Scale Integration (VLSI) Systems Volume 10, Issue 6, Dec. 2002 Page.

[10] S. Thompson, P. Packan and M. Bohr, MOS Scaling: Transistor Challenges for the 21st Century. Intel Technology Journal, Q3 1998.

[11] G. Kucuk, D. Ponomarev and K. Ghose, "Low-Complexity Reorder Buffer Architecture," 16th ACM International Conference on Supercomputing (ICS'02), New York, June, 2002, pp. 57-66.

[12] S. Palacharla, N. P. Jouppi and J. Smith, "Complexity-effective superscalar processors," In Proc.of the 24th Annual International Symposium on Computer Architecture, pages 206–218, June 1997.

[13] R. Canal and A. Gonzalez, "A low-complexity issue logic," In Proceedings of 2000 International Conferences on Supercomputing, May 2000.

[14] R. Canal and A. Gonzalez. "Reducing the complexity of the issue logic," In Proceedings of 2001 International Conferences on Supercomputing, June 2001.

[15] D. Folegnani and A. González, "Energy-effectiveissue logic," In Proceedings of the 28th annual international symposium on Computer architecture, May 2001.

[16] J. S. Hu, N. Vijaykrishnan, and M. J. Irwin, "Exploring Wakeup-Free Instruction Scheduling," In Proceedings of the 10th International Conference on High-Performance Computer Architecture (HPCA-10 2004), 14-18 February 2004, Madrid, Spain.

[17] E. Brekelbaum, J. R. II, C. Wilkerson and B. Black, "Hierarchical scheduling windows," In Proc. of the 35th Annual IEEE/ACM International Symposium on Microarchitecture, Nov. 2002.

[18] M. D. Brown, J. Stark and Y. N. Patt, "Select-free instruction scheduling logic," In Proc. of the International Symposium on Microarchitecture, Dec. 2001.

[19] S. Raasch, N. Binkert and S. Reinhardt, "A scalable instruction queue design using dependence chains," In Proc. of the 29th Annual International Symposium on Computer Architecture, May 2002.

[20] J. Stark, M. D. Brown and Y. N. Patt, "On pipelining dynamic instruction scheduling logic," In Proc. of the International Symposium on Microarchitecture, Dec. 2000.

[21] D.Ernst, A.Hamel and T.Austin, "Cyclone:a broadcast-free dynamic instruction scheduler selective replay," In Proc. of the 30th Annual International Symposium on Computer Architecture, June 2003.