

Power-Aware Issue Queue Design for Speculative Instructions

Tali Moreshet and R. Iris Bahar
Brown University, Division of Engineering, Providence, RI 02912
{tali,iris}@lems.brown.edu

ABSTRACT

Speculatively issued instructions may be particularly sensitive to increases in pipeline depth. Our results indicate that as pipeline depth increases, speculation increases the percentage of issue queue instructions that are waiting to be potentially re-issued in case of a mis-speculation. To compensate, issue queues are larger and thus more power hungry. We propose an alternative design called the *Dual Issue Queue*, that retains pre- and post-issue instructions in separate, smaller queues, saving 18% of issue queue power dissipation without degrading performance.

Categories and Subject Descriptors

C.1.3 [Processor Architectures]: Other Architecture Styles—
Pipeline processors, Adaptable architectures

General Terms

Design

Keywords

Low power design, microarchitecture, speculation

1. INTRODUCTION

Current modern superscalar processors rely on executing instructions out of program order to enable more instruction level parallelism and higher performance. In order to execute more instructions per cycle, it is necessary to minimize false dependencies among instructions, hide instruction latencies, and speculate across true dependencies. Among the techniques implemented are branch prediction, value and address prediction [4], memory dependence prediction [16], [4], [5], and latency prediction [16], [12].

To facilitate these speculation techniques, some means of recovering from a misprediction must also be employed. In the case of branch misprediction, the correct path instructions must be fetched from the instruction cache. For other types of speculation, the same instructions need to be re-executed (this time with correct data). Although these mispredictions may be handled in a similar manner as branch mispredictions, this can be costly in terms of performance. Instead, another way of enabling re-execution is to keep all “post-issue” instructions in the issue queue (a.k.a. instruction window) until the prediction is verified. While this approach may lead to some inefficiencies in the design of

the issue queue, it provides for a quick means of accessing instructions that need to re-execute.

Current trends in microprocessor designs show increasing pipeline depth in order to keep up with higher clock frequencies and increased architectural complexity. By employing speculation techniques, deeper pipelines will increase the number of cycles between the speculation and verification stages. In addition, as pipelines get deeper, a larger percentage of the issue queue will be needed to hold post-issue instructions—most of which will never be needed, assuming low misprediction rates. This limits the queue’s effectiveness in exposing available instruction level parallelism (ILP).

In this paper we focus on load hit speculation as an example of a prediction technique that uses the issue queue to hold post-issue instructions for misprediction recovery. Load hit speculation predicts the latency of a load instruction’s access to memory. In general, there is a non-zero delay between the point an instruction is issued to the time it can begin execution. This delay is due to register file access and moving of data across buses. For instance, in the Pentium4 processor it takes 2 cycles to move data across buses and 4 cycles to access the register file. Because of this non-zero delay, if load data is to be used immediately after it is available, an instruction dependent on the load must issue *before* it is known whether the load actually hit in the first level cache. The *load resolution loop* is the delay between the issue of a load instruction until its hit/miss information is passed back to the load’s dependent instructions. This loop delay increases as the delay between instruction issue and execute increases (i.e., as pipeline depth increases).

Since in practice most load instructions do hit in the first level cache, it is reasonable to schedule all instructions dependent on the load assuming that the load hits in the cache, and therefore has minimal latency. This approach requires that post-issue instructions remain in the issue queue until a load hit is verified. If the load misses, all post-issue instructions dependent on the load (i.e. those issued in the *speculative window* of the load) must be re-issued, or *replayed* from the queue. The Alpha 21264 [6], as well as the Pentium4 [10], use load hit speculation. They allow instructions dependent on a load to be issued assuming the load instruction hit in the first level cache, and therefore has minimal latency. If the load hits, then its dependent instructions benefit from the possibility of issuing early. If the load misses, the dependent instructions need to be re-issued.

Although keeping instructions in the issue queue allows for a fast recovery path, it requires that the issue queue be designed with enough entries to hold post-issue instructions while at the same time expose a large amount of instruction-level parallelism. In fact, we found that the portion of the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC 2003, June 2–6, 2003, Anaheim, California, USA.

Copyright 2003 ACM 1-58113-688-9/03/0006 ...\$5.00.

issue queue holding post-issue instructions grows from less than 5% to over 55% with the employment of load hit speculation. In addition, the issue queue is on the critical path, thus requiring it to be implemented using high-speed circuitry. This combination makes the issue queue structure very power hungry. For example, Wilcox *et al.* showed that the issue logic could account for 46% of the total power dissipation in future out-of-order processors that support speculation [15]. Similarly we estimate that half of this power dissipation comes from the issue queue itself [15].

From the previous discussion we see that there are two inefficiencies with issue queue designs. First, the queue has to have enough entries to hold both pre- and post-issue instructions and meet the tight timing constraints for a single-cycle bid/grant loop when on average less than half are likely to ever bid. This wastes power and aggravates critical paths. Second, the post-issue instructions may be limiting the exposure of available ILP. In this paper we address both inefficiencies. Specifically, we propose a more cost-effective *Dual Issue Queue* structure that separates the post-issue instructions from the rest of the pending pre-issued instructions in the queue. Using this design, we estimate 18% of the issue queue power can be saved without degrading performance.

2. RELATED WORK

Several works have addressed reducing the complexity and power of the issue queue structure [13], [1], [7], [14]. Of the more relevant works, in [1], Bahar *et al.* propose a scheme to dynamically adjust the processor’s issue width in order to save power without losing performance. Folegnani *et al.* [7] proposed to dynamically resize the instruction queue according to performance requirements in order to save power. A static scheme to save power was proposed in [14], that divided instructions among a pair of in-order and out-of-order issue queues according to instruction criticality. We propose a reduced complexity, power aware issue queue structure; however, unlike the above mentioned works, we also evaluate the specific effect of speculation and deeper pipelines on the issue queue structure.

Lebeck *et al.* [11] proposed an alternative scheme for dealing with instructions dependent on loads that miss. Following a load miss, the chain of its dependent instructions is moved from the issue window to a *waiting instruction buffer* (WIB). Once the load completes, its dependent instructions are moved back to the issue window. The WIB structure requires a larger number of entries and a separate mechanism for detecting dependencies between instructions. In addition, instructions need to be moved back and forth to and from the two structures. In construct, our proposed scheme does not have these limitations. Morancho *et al.* [12] proposed a structure called the *recovery buffer* which stores issued instructions dependent on mispredicted loads. This scheme assumes the verification delay is known at the time of issue. Although issue logic in the recovery buffer is simplified, it does not scale well with longer delays, and the authors do not include any evaluation of power.

3. LOAD HIT SPECULATION MODEL

The simulator used in this study is a modified version of the SIMPLESCALAR [3] tool suite. The configuration of the processor models a future generation out-of-order micro-architecture with an 8 instruction wide pipeline. In addition,

Table 1: Baseline processor configuration.

Parameter	Configuration
Inst. Window	64-entry LSQ, 256-entry ROB 64-entry ISQ
Machine Width	8-wide fetch,issue, commit
Fetch Queue	16-entry
Number of FUs Latency in ()	8 Int add (1), 2 Int mult/div (3/20) 4 Load/Store (3), 8 FP add (2) 2 FP mult/div/sqrt (4/12/24)
L1 Icache	16KB 2-way; 32B line; 1 cycle
L1 Dcache	64KB 2-way; 32B line; 3 cycle
L2 Cache	128KB 4-way; 64B line; 12 cycle
Memory	100-cycle latency; 16 bit-wide
Branch Pred.	4k 2lev + 4k bimodal + 4k meta 6 cycle mispred. penalty
BTB	1K entry 4-way set assoc.
RAS	32 entry queue
ITLB	64 entry fully assoc.
DTLB	64 entry fully assoc.
issue→exec. latency	variable from 1 to 9 cycles
feedback delay	variable from 1 to 9 cycles

we implemented a separate reorder buffer (ROB) and issue queue (ISQ), which is common in today’s microprocessor designs. Our simulator models a single unified queue for integer and floating point instructions, and we assume that issued instructions are kept in the issue queue until it is known that they will not need to be re-issued. Table 1 shows the complete configuration of the processor.

Modifications to SIMPLESCALAR also include a variable, 1–9 cycle delay between the issue and added execute stage, in order to increase the pipeline depth specifically between the issue and writeback stages. Also, a variable wire latency was added for the feedback of hit/miss information from the cache to the consuming, post-issue instructions. Consuming instructions may be issued such that the producers’ results will be ready by the time execution begins. Load instructions are speculated to be ready after the time it takes to access the level 1 data cache. Once it is discovered that a load missed in the cache, instructions that were issued in a mispredicted load’s speculative window and are dependent, directly or indirectly, on the load are replayed.¹ Therefore, issue queue entries are released only when instructions reach writeback, for all instructions.

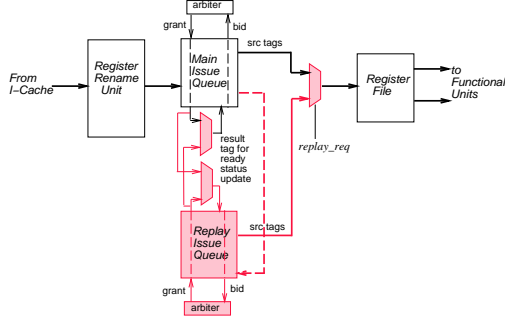
Simulations are executed on a subset of the SPEC95 and SPEC2000 integer and floating point benchmarks [8], [9] (a total of 32 benchmarks). DL1 miss rates for these benchmarks varied from 0–23%.

4. DUAL ISSUE QUEUE SCHEME

In order to reduce the power dissipation of the issue queue, we could try to decrease its size. However, we found that reducing the size of the issue queue by as little as 25%, to a 48-entry issue queue decreased performance up to 20% for some benchmarks. A 48-entry issue queue is not sufficient, because the smaller issue queue becomes cluttered with post-issue instructions, not allowing new instructions to enter. If post-issue instructions were removed from the issue queue (and placed elsewhere), we could potentially reduce the size of the issue queue, saving power in the queue while at the same time increasing available ILP and thereby retaining performance. We now investigate this approach.

¹Without load hit speculation, instructions can be removed from the issue queue as soon as they issue and resolve their dependencies. With load hit speculation instructions cannot be removed until they are guaranteed to not be replayed.

Figure 1: Proposed Dual Issue Queue Scheme.

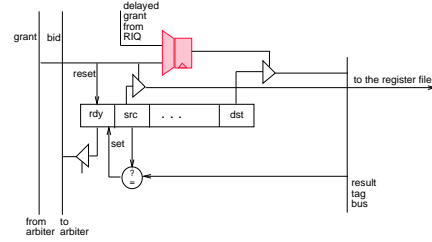


Our new issue queue design consists of two parts: the main issue queue (MIQ), and the replay issue queue (RIQ). The RIQ is effectively a temporary place holder for issued instructions that may need to be replayed. The MIQ and RIQ are similar in structure. The main difference is that since the RIQ needs to be searched for ready instructions only after a load hit misprediction, it is not on the critical path of the processor pipeline. This allows us to make the RIQ’s bid/grant loop slower, with a 2 cycle latency. The pipeline for our new issue queue scheme is shown in Figure 1. The added logic is shown in gray. Initially, dispatched instructions are placed in the MIQ. The MIQ is searched for ready instructions, and these are given a chance to bid for an issue slot. This part of the issue queue is similar to a standard out-of-order issue queue. Instructions that have already issued and updated the ready status of their dependent instructions are then moved from the MIQ to the RIQ if there are empty slots available. If the RIQ is full, the issued instructions can remain in the MIQ. After instructions are issued, chances are that they will not be dealt with again, since most load instructions hit in the cache, and their dependent instructions will not be re-issued.

During the issue stage, instructions may be selected for issue either from the RIQ or the MIQ, but not both. To facilitate this, instructions from both queues are allowed to update their ready status every cycle, but only one queue is allowed to bid for issue resources at a time. In our simulation model, the processor gives priority in selecting instructions to be issued to the RIQ; however, the only time instructions in the RIQ can bid and be granted an issue slot is after a load hit misprediction. The RIQ holds ready instructions only after a missed load reaches the writeback stage. Therefore, we do not need to check the RIQ for ready instructions every cycle.

Once an instruction is granted an issue slot, the ready status of its dependent instructions need to be updated. If the issued instruction comes from the RIQ, its dependent instructions may reside in either the RIQ or MIQ. The dependent instructions may be in the MIQ due to lack of space in the RIQ or simply because they are pre-issue instructions. If the issued instruction resides in the MIQ, its dependent instructions will most likely reside in the MIQ as well. These updates are part of the bid/grant critical path loop and will not be delayed an extra cycle. However, if dependent instructions from a load miss reside in both the RIQ and MIQ (due to space limitations in the RIQ) then, since the bid/grant loop of the RIQ is slower, broadcasting the grant signals from the MIQ to the RIQ will incur a longer latency.

Figure 2: The bid/grant critical path, modified to support the Dual Issue Queue.



4.1 Power and Complexity

The complexity of designing and implementing any out-of-order issue queue is a function of the issue width of the processor and the number of entries in the issue queue [1], [13]. As illustrated in Figure 1, updating dependent instructions in the MIQ requires adding a multiplexor along the bid/grant loop since the issued instruction can come from either the MIQ or RIQ. Although this multiplexor does not come for free, the delay and power impact may be minimized by incorporating the MUX function into the latch used to control the driving of the result tags, as shown in Figure 2. Furthermore, the select lines for the MUX are not on the critical path since they are set according to the actions occurring in the previous cycle. The extra MUX added to the source tag output bus path also adds some delay and power; however, combining it with a pipeline latch will make this extra logic unlikely to have a significant effect on power or the issue to execute latency. In any case, this dual issue queue scheme allows us to use an RIQ and MIQ each with fewer entries compared to a unified issue queue design. Fewer entries means reduced capacitive load on the big/grant lines as well as a simplified arbiter. Also, since a single issue queue entry is comprised of $2 \times W$ tag comparators, where W is the issue width of the machine, the saving from eliminating these comparators can more than compensate for the added power dissipation of the MUXes.

After simulating the dual issue queue configuration with different sized queues, we found that a 48-entry MIQ and a 24-entry RIQ is sufficient in terms of performance. To estimate the power savings of using this dual issue queue scheme over a standard 64-entry issue queue, we extrapolated from available Alpha 21264 power estimates [15], as well as previous work [1], [13], [14]. It was shown in [15] that the 8-way issue Alpha 21464 was expected to have 23% of its power dissipation resulting from the issue queue logic. We also assume that the issue queue logic consists of the register scoreboard (consuming 25% of this power), the request logic (15%), and arbiters (60%) [1]. Our dual issue queue scheme saves power by using smaller queues, one of them slower and inactive for the major part of the execution.² The MIQ and RIQ consume $\frac{48}{64}$ and $\frac{24}{64}$ of the power of a standard queue respectively (this is approximated for the arbitration logic [13]). The slower RIQ may be designed using slower circuit styles, higher threshold transistors, and reduced transistors sizes[2]. We assume that it will reduce dynamic power dissipation by 20% and leakage power by 50% [14]. Assuming dynamic power accounts for 90% of total power and leakage 10%, the slower RIQ dissipates 77% of the power of a standard queue of the same size. The inac-

²The RIQ is active only 0–4% of the time.

Table 2: Fraction of the power consumption of a standard issue queue.

	reg scoreboard (25%)	req logic (15%)	arbitration (60%)	total
inactive RIQ (24-entry)	(0.77)($\frac{24}{64}$)	0	0	0.07
active MIQ (48-entry)	$\frac{48}{64}$	$\frac{48}{64}$	$\frac{48}{64}$	0.75

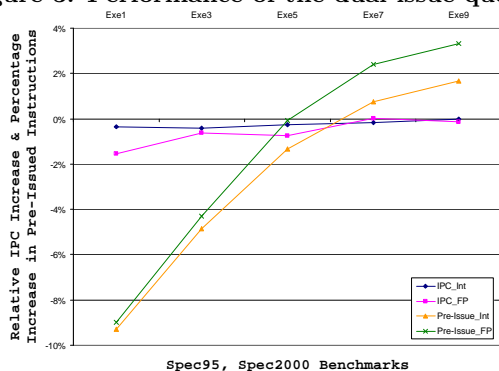
tive RIQ has its arbiters and request logic disabled, but still updates its scoreboard. Table 2 summarizes the analysis. Summing up, our scheme dissipates only 82% of the power of a standard issue queue.

4.2 Performance Results

The performance impact of our scheme compared to a unified single cycle 64-entry issue queue was measured in terms of instructions per cycle (IPC). In Figure 3 we show overall averages for floating point and integer benchmarks in lines **IPC.Int** and **IPC.FP** respectively. Performance does not suffer despite the fact that the main issue queue is smaller. For the deepest pipeline simulated (Exe9), average performance degradation was close to 0%, with a range of -5% to +3.5% change in performance. These performance results do not take into account the fact that the clock rate may be increased since the issue queue is smaller (assuming its speed determines the global clock rate). The largest performance degradation is seen for simulations with a 1 cycle issue to execute latency (Exe1), but since our scheme targets deeper pipelines, this is not so much a concern for us. For shallow pipelines the number of post-issue instructions is smaller than for deeper pipelines. The additional RIQ structure is thus less utilized, and the performance suffers from the smaller MIQ. Lines **Pre-Issue.Int** and **Pre-Issue.FP** show the percentage increase of pre-issued instructions in the issue queue with the dual issue queue scheme, with respect to the standard 64-entry issue queue. As can be seen, the dual issue queue scheme allows the number of pre-issued instructions to increase with pipeline depth, and as a result so does the exposure of available ILP.

We found that largest performance improvements for the dual issue queue scheme came from *swim* (2.3%) and *wup-wise* (3.5%) under the deepest pipeline. The better performing benchmarks are floating point benchmarks, characterized by having more parallel streams of dependent instructions. They also tend to benefit the most from having a larger issue queue. These benchmarks have a relatively large increase in the number of pre-issued instructions in the issue queue, and benefit from their ability to expose more

Figure 3: Performance of the dual issue queue.



ILP with the dual issue queue scheme. This boost in performance comes on top of the 18% savings in power dissipation of the issue queue.

5. CONCLUSION

Speculation is an important method for increasing performance by enabling more instruction-level parallelism. However, speculation increases the percentage of post-issue instructions in the issue queue, causing designers to use larger queues to compensate. These queues are power hungry and put additional pressure on the critical path. We propose a new issue queue scheme that addresses the utilization concerns while saving 18% of the issue queue power dissipation and without compromising performance. Future work will examine the impact on the issue queue when multiple forms of speculation such as load address, value or memory dependence prediction are also employed.

Acknowledgments

This work was supported in part by an NSF-CAREER grant number MIP-9734247 and an equipment grant from Sun Microsystems.

6. REFERENCES

- [1] R. I. Bahar and S. Manne. Power and energy reduction via pipeline balancing. In *ISCA*, July 2001.
- [2] S. Borkar. Design challenges of technology scaling. *IEEE Micro*, 19(4), August 1999.
- [3] D. Burger and T. Austin. The simplescalar tool set. Technical report, University of Wisconsin, Madison, 1999. Version 3.0.
- [4] B. Calder and G. Reinman. A comparative survey of load speculation architectures. In *Journal of Instruction-Level Parallelism*, May 2000.
- [5] G. Chrysos and J. Emer. Memory dependence prediction using store sets. In *ISCA*, June 1998.
- [6] Compaq Computer Corporation. *Alpha 21264 Microprocessor Hardware Reference Manual*, July 1999.
- [7] D. Folegnani and A. Gonzalez. Energy-effective issue logic. In *ISCA*, July 2001.
- [8] J. Gee, M. Hill, D. Pnevmatikatos, and A. J. Smith. Cache performance of the spec benchmark suite. *IEEE Micro*, 13(4), August 1993.
- [9] J. L. Henning. Spec cpu2000: Measuring cpu performance in the new millennium. *IEEE Computer*, 33(7), July 2000.
- [10] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel. The microarchitecture of the pentium 4 processor. *Intel Technology Journal*, Q1 2001.
- [11] A. R. Lebeck, J. Koppanalil, T. Li, J. Patwardhan, and E. Rotenberg. A large, fast instruction window for tolerating cache misses. In *ISCA*, May 2002.
- [12] E. Morancho, J. M. Llberia, and A. Olive. Recovery mechanism for latency misprediction. In *Int. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, September 2001.
- [13] S. Palacharla, N. Jouppi, and J. Smith. Complexity-effective superscalar processors. In *ISCA*, June 1997.
- [14] J. S. Seng, E. S. Tune, and D. M. Tullsen. Reducing power with dynamic critical path information. In *MICRO-34*, December 2001.
- [15] K. Wilcox and S. Manne. Alpha processors: A history of power issues and a look to the future. In *Cool-Chips Tutorial*, November 1999. Held in conjunction with *MICRO-32*.
- [16] A. Yoaz, M. Erez, R. Ronen, and S. Jourdan. Speculation techniques for improving load related instruction scheduling. In *ISCA*, May 1999.