ELectronics EXpress

IEICE Electronics Express, Vol.5, No.22, 927–931

A low-cost recovery scheme for dynamically scheduled processors

Min Choi[a)] **and Seungryoul Maeng**

Department of Electrical Engineering and Computer Science, KAIST

Guseong-dong, Yuseong-gu, Daejeon 305–701, Korea

a) *min@kaist.ac.kr*

Abstract: Although today's branch predictors show high accuracy, the branch misprediction penalty is getting larger due to aggressive speculation and deeper pipelining. In order to reduce the miss penalty, we propose a fast and low-cost branch recovery scheme using the incremental register renaming (IRR) and the bit-vector based rename map table (BVMT). The IRR enforces the destination register number of the instruction stream to appear in non-decreasing order. With this incremental property of the IRR, the BVMT recovery scheme completely eliminates the roll-back overhead on branch misprediction. Thus, the instruction fetcher does not stop and it fetches instructions from the correct path immediately after the misprediction detected. The goal of our scheme is to prevent a processor from flushing the pipeline, even under branch misprediction. Consequently, the BVMT instantly reconstructs the map table to any mispredicted branch and it outperforms the conventional approach by an average of 10.93%.

Keywords: processor architecture, branch misprediction recovery, bit-vector based rename map table

Classification: Integrated circuits

References

[1] H. Akkary, R. Rajwar, and S. Srinivasan, "Checkpoint Processing and Recovery: Towards Scalable Large Instruction Window Processors," *IEEE International Symposium on Microarchitecture (MICRO)*, pp. 423–434, Dec. 2003.

[2] P. Ranganathan, V. Pai, and S. Adve, "Using Speculative Retirement and Larger Instruction Windows to Narrow the Performance Gap between Memory Consistency Models," *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pp. 199–210, June 1997.

[3] P. Zhou, S. Onder, and S. Carr, "Fast Branch Misprediction Recovery in Out-of-order Superscalar Processor," *Proceedings of the ACM International Conference on Supercomputing (ICS)*, June 2005.

[4] Synopsys Design Compiler 2007, [online] http://www.synopsys.com/

[5] J. Martinez, J. Renau, M. Huang, M. Prvulovic, and J. Torrelas, "Cherry: Checkpointed Early Resource Recycling in Out-of-order Microprocessors," *Proceedings of the IEEE/ACM International Symposium on Microarchi-*

© IEICE 2008

DOI: 10.1587/elex.5.927
Received September 08, 2008
Accepted October 08, 2008
Published November 25, 2008

tecture (MICRO), 2002.

1 Introduction

Branch misprediction is one of the significant performance degradation contributors in high-frequency deep pipeline superscalar processors. Improving the accuracy of branch prediction has been a holy grail of the processor designer. Yet, it is also important to reduce the misprediction recovery latency, because this incurs "stalls" to repair the architecture state. The most critical part of the architecture state is the register renaming map table, used for register renaming to increase instruction level parallelism by removing false data dependency. Several techniques have been proposed for efficient recovery such as checkpoint processing and recovery [1], history buffer recovery [2], eager misprediction recovery [3], etc. The key idea of these schemes is to maintain an extra data structure for fast recovery, e.g., a history buffer. However, in the worst case, these schemes require processing all the instances in the data structure superseding the mispredicted branch. Thus, the worst case recovery complexity of these schemes is O(n) where n is the number of such instances. As pipelines are stretched, the recovery processing latency linearly increases, causing non-trivial performance loss. The goal of this paper is to design a fast, low-cost misprediction recovery mechanism. To this end, we propose a register renaming policy called incremental register renaming (IRR). Then, we propose a bit-vector based rename map table (BVMT) where each row represents a mapping bit vector of an architectural register to the physical registers. On a misprediction, a process can recover the rename map table by simply resetting all the columns used by the instructions superseding a mispredicted branch. Thus, we can achieve O(1) recovery complexity.

2 Incremental Register Renaming

We propose the IRR, a novel register renaming policy. The IRR is distinct from conventional register renaming because it uses a different reclaiming policy to rename registers. Although a recently freed physical register used by a preceding instruction is available, the IRR does not reuse the register to map the destination register of the next instruction. Instead, the IRR maps architectural registers of each instruction to a rename register "sequentially" as shown in Fig. 1. For example, given that the last mapped physical register number is say 8, the destination register of a new instruction will be mapped to the 9th physical register. In this way, the IRR enforces the destination register numbers of an instruction stream to appear in the non-decreasing order. As we will see, this incremental property of the IRR has a great potential of eliminating the roll-back overhead on branch misprediction.

© IEICE 2008
DOI: 10.1587/elex.5.927
Received September 08, 2008
Accepted October 08, 2008
Published November 25, 2008

3 The BVMT Structure

To handle recovery from branch mispredictions quickly and efficiently, we modified the conventional rename map table structure to the BVMT. The BVMT has head and tail pointers like a circular queue. The tail index is incremented for a mapping in each instruction and similarly, the head index is incremented when the instruction retires. Both indexes simply move to the next column in the round-robin order driven by IRR, and thus a processor will use a renamed register that corresponds to the index number of the tail pointer in the BVMT. This is because a processor writes new register mappings sequentially to the tail of the BVMT. Fig. 1 demonstrates the update

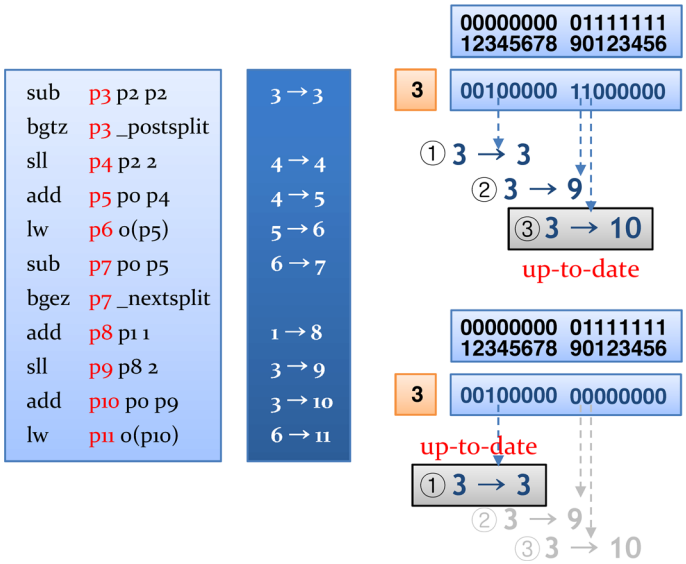

Fig. 1. An example of incremental register renaming and the usage of bit-vector based RMT

and recovery of the BVMT. In the BVMT structure, each row represents the architectural register number and the bits in a column are translated to the renamed register number. For the mapping of 1→8, for instance, we set a bit on the 8th column of the first row. Let us say that the misprediction occurs at "bgez," then the processor just need to clean all bits after the 8th column, since IRR guarantees that those mapping are only incorrect. The right-hand side of Fig. 1 shows the third row of the BVMT. It represents the fact that the architectural register 3 is mapped to the renamed register 3, initially. After that, two subsequent instructions that need architectural register 3 use the renamed register 9 and 10, respectively. The mapping to the 10 is the latest mapping. The figure shows the state after the branch recovery of the mispredicted branch, "bgtz". Since the most up-to-date state of the mapping is from 3 to 3, last two subsequent mappings are squashed during the recovery process. The BVMT can be simply realized using a priority decoder that can translate the location of a bit to a physical register number. Here, we use the general four 8 bit and one 4 bit priority decoders. We measured the additional cost of the priority decoder in terms of power dissipation, area

© IEICE 2008
DOI: 10.1587/elex.5.927
Received September 08, 2008
Accepted October 08, 2008
Published November 25, 2008

estimation, and access delay as shown in Table I. The supplementary logic has been implemented with Verilog HDL and synthesized with the Synopsys Design Compiler [4] targeted towards a 0.18 μm TSMC library.

4 Result

The BVMT offers the following key advantages: update history preservation and low-cost recovery. First, the update history in the BVMT is entirely maintained within minimal storage overhead. We exploit the IRR to store the new register mappings incrementally without replacing the previous mapping information. Thus, a processor can rollback to any register map table state corresponding to a certain branch. Second, the BVMT minimizes the reconstruction overhead of register map table; i.e., recovering the machine state to the last consistent point of an exception or a branch misprediction. The recovery of a BVMT can be completed in a short time by squashing the column bit vectors of speculative instructions as of the mispredicted branch. The squashing operation can be done instantly using inexpensive custom circuitry, such as gang-cleaning [5]. We perform extensive simulations using the Spec CPU2000 benchmark suite on Simplescalar. Fig. 2 shows the performance in terms of instructions per cycle (IPC) for integer and floating applications respectively. From the figure, we see that the BVMT outper-

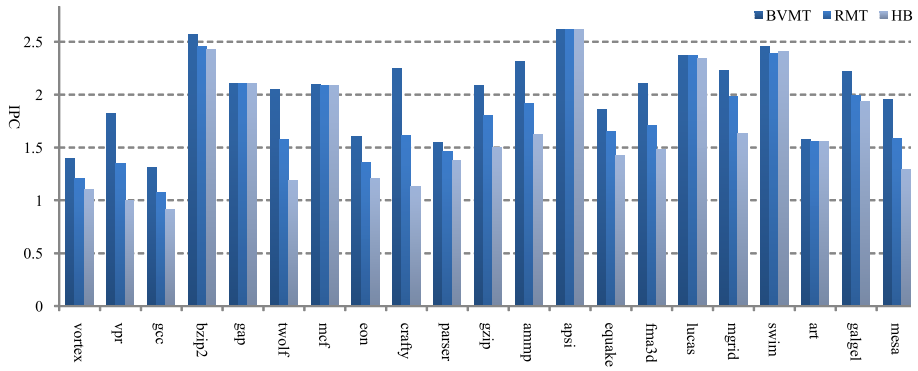

Fig. 2. IPC performance

forms the conventional rename map table (RMT) across all benchmarks by 10.93% on average. The RMT shows consistently better performance than the history based (HB) method. The performance enhancement comes from the fact that the BVMT method saves clock cycles to repair the map tables, whereas the RMT and the HB require additional cycles for the map table recovery ranging from a few cycles to tens of cycles. Moreover, the BVMT is more effective for integer applications than for floating point applications because floating-point applications have relatively better branch prediction accuracies in most cases, and thus, branch misprediction occurs more frequently in integer applications. In Table I, we provide complexity analysis for map table operations. For a write operation at the commit stage, the HB and the RMT have O(2) complexity because they both have to update not

© IEICE 2008
DOI: 10.1587/elex.5.927
Received September 08, 2008
Accepted October 08, 2008
Published November 25, 2008

only the rename map table, but also the history buffer and the retirement map table, respectively. However, the BVMT only has to clear only one bit, resulting in O(1). For a recovery operation, the HB method pops all mappings from the HB and updates into the rename map table in reverse order. Thus, in the worst case scenario all of the elements in the map table need to be updated and the time complexity of the HB method is O(n). In the RMT method, the retirement map table is copied to the frontend map table when the mispredicted branch reaches the retire point. Therefore, the asymptotic time complexity of RMT method is O(n). However, in the BVMT, the processor recovers the state by the gang-clear operation; i.e., all bits after the position of the mispredicted branch are cleared at once, and thus, the complexity is O(1).

Table I. Complexity Analysis and The BVMT Decoding Overhead

	BVMT	Checkpoint	HB	RMT
Write at dispatch	O(1)	-	O(1)	O(1)
Write at commit	O(1)	-	O(2)	O(2)
Checkpoint	O(1)	O(n)	-	-
Recovery	O(1)	O(n)	O(n)	O(n)

Area		Power		Timing	
Num. of ports	38	Dynamic	909.6226 μW	Min	0.74 ns
Num. of nets	88	Leakage	2.5549 nW	Max	1.49 ns
Num. of cells	56				

5 Conclusion

In this paper, we prevent a processor from flushing the pipeline under branch misprediction by allowing the instruction fetcher to work continuously. To this end, we propose a fast and low-cost branch recovery scheme using incremental register renaming and the BVMT. The proposed renaming method can reconstructs the map table corresponding to any branch with the minimum overhead of O(1) complexity. The overhead includes only the squashing operation which can be done instantly using inexpensive custom circuitry. We do not require extra overhead such as identifying speculative state. Thus, the front-end can resume immediately after the recovery process with low cost. Consequently, our approach enables the front-end to fetch instructions from the correct path immediately after the misprediction detected.

Acknowledgments

This research was supported by the MKE(Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute for Information Technology Advancement) (IITA-2008-C1090-0801-0045).

© IEICE 2008
DOI: 10.1587/elex.5.927
Received September 08, 2008
Accepted October 08, 2008
Published November 25, 2008