High performance and low design cost TLB for MIPS32 processor

Takahiro SASAKI[1], Gun MUTO[2*], Yuki FUKAZAWA[3] and Toshio KONDO[4]
[1,2,3,4]Department of Information Engineering, Mie University
[*]Currently with Akatsuki Inc.
1577 Kurimamachiya-cho, Tsu city, Mie prefecture, 514-8507, Japan
E-mail : {[1]sasaki, [2]muto, [3]fukazawa, [4]kondo}@arch.info.mie-u.ac.jp

Abstract: TLB (Translation Lookaside Buffer) is one of the key components which affects performance and circuit scale. To improve performance of TLB, it is effective to increment the number of entries. However, because MIPS32 processor adopts CAM based TLB, increasing TLB entry causes serious enlargement of circuit scale. Furthermore, MIPS32 ISA limits the TLB entries up to 64. This paper proposes two methods to break the limitation; one is new approach to implement TLB using RAM for small footprint, and another is adopting a dynamic code analyzer to handle more than 64 TLB entries while keeping binary compatibility. According to the evaluation results, the proposed approach improves hit rate by 20% at the maximum, 6% in average, and reduces area by 29%.

Keywords—**Processor architecture, TLB, RAM, CAM, MIPS32**

1. Introduction

In recent years, superscalar processors that fetch and run multiple instructions at once attracts attention with the need for high-performance embedded processor. In order to research and develop a superscalar processor, it is required that designer can change the processor design implemented by HDL (hardware description language) easily. To meet above demand, FabScalar has been proposed[1]. FabScalar generates HDL design of various superscalar core by passing a parameter file which describes parameters such as fetch width, pipeline length and issue queue size. Although FabScalar supports variety of instruction sets, this study focuses on MIPS32 base processor that is widely used in research fields and embedded devices.

General-purpose processor, including MIPS32, has the high speed buffer called TLB (Translation Lookaside Buffer) in the processor to translate virtual address to physical address promptly to support virtual memory mechanism. In MIPS32 ISA, the TLB is required to be constructed by CAM (Content Addressable Memory). However, CAM has a problem that design cost is very high because design effort of it is very high and circuit scale is large. In contrast, a RAM can be designed easily using memory macros or a memory compiler. Although CAM can be written in register transfer level (RTL) and be synthesized, the area of CAM designed with standard-cell is very large because search key and bit width of TLB are large.

Hence, this study proposes an implementation methodology of a pseudo CAM approach TLB using only standard-cell and RAM macros. This methodology realizes almost same behavior as CAM approach TLB which is used in original MIPS32. However, this methodology compliant to specification of MIPS32. Therefore, it is impossible to achieve better performance than the original MIPS32 TLB. In general, hit rate of TLB raise and memory access performance is improved by increasing TLB entries. Because the proposed pseudo CAM approach is small footprint, TLB entries can be increased with low cost. However, MIPS32 ISA limits the maximum TLB entries to 64, so increasing TLB entries breaks binary compatibility. To achieve better performance than the original MIPS32 TLB, this study also proposes dynamic code analyzer for more than 64 TLB entries. The proposed approach supports more than 64 entries without existing firmware or operating system (OS) modification by analyzing executing code and compensating the difference of behavior between the original MIPS32 TLB and our large TLB. This study implements proposed two approach and evaluates its effectiveness. According to the evaluation results, the proposed approach improves hit rate by 20% at the maximum, 6% in average, and reduces area by 29%.

2. Related Works

Because a CAM is widely used in not only processors but also network routers, encoder/decoder hardware and so on, effective implementation methods of CAM are proposed.

Hong proposes binary-tree-based pseudo CAM design[2]. This approach achieves compact CAM implementation. However, access latency is not so short because the approach adopts binary tree trailing. TLB is required to access in one cycle, so this approach is not suitable for TLB implementation.

Fully CAM implementation methods for FPGA is also proposed[3], [4], [5]. This approach implement a true CAM function. However, circuit scale is in proportion to square of pattern length (that is bit width of virtual address without page offset in a TLB). Therefore, using this approach enlarge circuit scale dramatically. Reference[6] implements superscalar processor using CAM implementation method proposed in [3], [4]. However, the processor does not have a MMU, and no TLB is implemented.

Similar to our approach, references[7], [8] use RAM macro to implement a CAM. However their target is TCAM (Ternary CAM) and access time is not one cycle, so it is not suitable for TLB.

Some FPGA products[9], [10] have hardware CAM macros. For example Altera APEX20 series has CAM macro in the ESB (Embedded System Block) and enable to implement a hardware CAM. Using these hard macro, TLB can be implemented easily[11]. However, most modern FPGA products does not have hardware CAM macros, and even if the device has hardware CAM, the processor design is dedicated

a specific FPGA products. Our approach can apply to almost all FPGAs and ASIC design flow.

3. Pseudo CAM type TLB using RAM

As mentioned above, a TLB constructed by a CAM is large circuit scale. In order to reduce the circuit scale, we propose a pseudo CAM type TLB using RAM macros[12],[13]. Figure 1 shows the basic idea. This approach is similar to set associative cache system, but we introduce 1)hash function and 2)index table. We implement a TLB using RAM instead of CAM. Like a general set associative cache system, LRU is used to select a bank to replace.

The number of TLB entries denoted as M and the number of ways in the Figure is independent from MIPS32 ISA. If M or the number of ways become large, the TLB hit ratio is close to the original MIPS32 TLB based on CAM. According to our preliminary simulation results, 2-way 64entry TLB can achieve almost same performance to the original MIPS32 TLB. Therefore, in this paper, we use above parameters.

3.1 Hash function

Generally cache memory is accessed using index address which is part of memory address. However, because the number of TLB entries is up to 64 because of MIPS32 ISA limitation, if we adopt index approach, which is same to cache system, into TLB, conflicts which causes large performance degradation occur frequently. So we introduce hash function to disperse TLB entry to the whole TLB.

3.2 Index table

We also introduce an Index Table to access a TLB entry in a similar manner to CAM based TLB. On CAM based TLB, any tuple, which is pair of virtual address and physical address, can be stored in any place. So to replace an entry, index which is sequential address in CAM is used. However, on RAM based TLB, the place to store a tuple is calculated by the equation (1).

$$(\text{virtual address}) mod (\#\text{ of TLB entries}) \qquad (1)$$

So the index of CAM based TLB and that of RAM based TLB have different meaning.

To solve this problem, we use an Index Table to hold relation to the index used in CAM based TLB and the index used in RAM based TLB. In this paper, we call the RAM address which is hash value of VPN and has value from 0 to (M-1) as RAM index, and pseudo-CAM address which has no physical meaning but required for binary compatibility and has value from 0 to 63 as CAM index.

4. Dynamic Code Analyzer

The proposed approach described in Section 3 can achieve almost same performance to CAM based TLB with smaller circuit scale. However, because of limitation of MIPS32 ISA, the proposed approach can not outperform the original CAM based TLB. As mentioned above, original MIPS32 ISA limits the TLB entries.

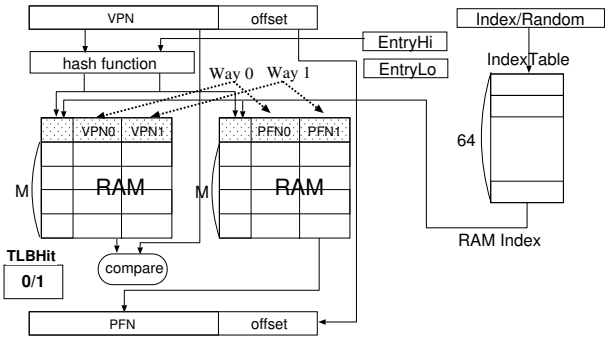

Figure 1. Basic idea of 2-way RAM base TLB.

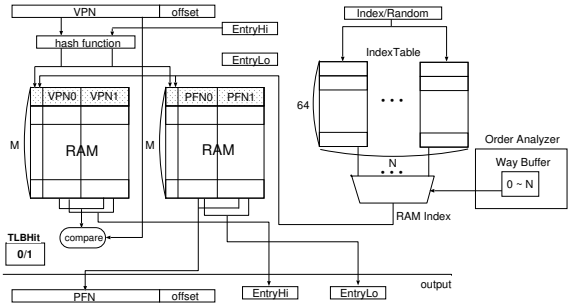

Figure 2. Dynamic code analyzer.

To break the limitation, we introduce dynamic code analyzer to increase the number of TLB entries with keeping binary compatibility. Figure 2 shows block diagram of the pseudo CAM type TLB using 2-way RAMs with proposed dynamic code analyzer. We increase the number of Index Table, and add dynamic code analyzer (labeled Order Analyzer in the Figure2).

Generally, MIPS32 programs replace a TLB entry by following two manners; one is using a set of TLBP and TLBWI instructions, and the other is using a TLBWR instruction. In the former case, before replacing a TLB entry, TLB probe instruction (TLBP) is executed to find the replace target entry. Succeeding to the TLBP instruction, TLB write index (TLBWI) is execute to replace the TLB entry. In the latter case, instead of specifying the position to replace, write random TLB entry (TLBWR) instruction, which writes to a random TLB entry, is used.

4.1 TLBP and TLBWI instructions

In the original MIPS32 processor, TLBP instruction probes TLB and the result of TLBP which is CAM index, is stored in the privileged register named Index register. Succeeding TLBWI instruction uses the Index register as CAM index of TLB entry to replace. However, our approach does not use a CAM, so we need to generate pseudo-CAM index. The Index Table described above holds relation between CAM index and RAM index. However, to find CAM index, we need to retrieve whole TLB entries. To find it quickly, we also introduce CAM Index Table which holds mapping from RAM index to CAM index.

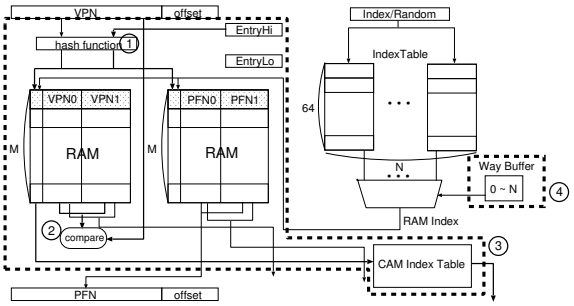

Figure 3. TLB probe instruction.

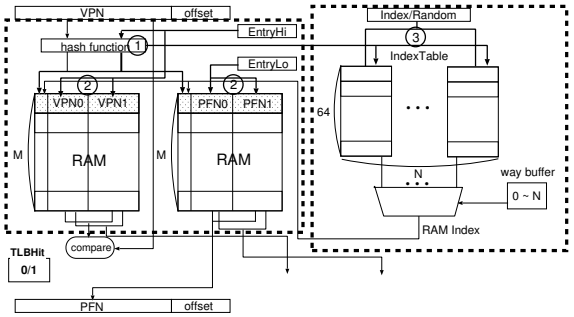

Figure 4. TLB write index instruction.

In addition, our approach holds the way to the Way Buffer register shown in the Figure 2, and uses it when replacing a TLB entry.

Figure 3 shows the behavior of TLBP instruction. As first, (1)hash value of virtual page number (VPN) which is stored in the EntryHi register is calculated. Then, (2)VPN-RAM is read using the hash value as a RAM index, and compare VPN and both VPN0 (Way 0) and VPN1 (Way 1). If VPN0 or VPN1 matches to VPN, TLB hits and (3)translates RAM index to CAM index using CAM Index Table. In addition, (4)TLBP stores the bank number in which the TLB entry found to Way Buffer.

Succeeding TLBWI replaces the TLB entry. As first, (1)hash value is calculated using VPN in the EntryHi register to calculate RAM index. Next, (2)to select the way to replace LRU is used and replace a TLB entry. At the same time, (3)both of RAM Index Table and CAM Index Table are updated.

4.2 TLBWR instruction

TLBWR is similar to TLBWI instruction. Instead of Index register, TLBWR uses Random register as CAM index. Other mechanism is same to the TLBWI.

4.3 TLB flush

Firmware or OS sometimes flush whole TLB. Generally, these system software invalidates all 64 TLB entries using TLBWI. However, our approach has more than 64 TLB entries, so these code does not work correctly. In order to solve the problem, we also add two special registers, named LastIndex and UpdateNum, to detect TLB flush. When TLBWI is executed, the value of Index register is compared to LastIndex+1. If the values match, UpdateNum is incremented. Otherwise, UpdateNum is cleared. If UpdateNum becomes 64, it means system software try to flush whole TLB entries, so whole TLB entries is cleared.

In this way, our approach detects TLB flush code, and behaves suitably.

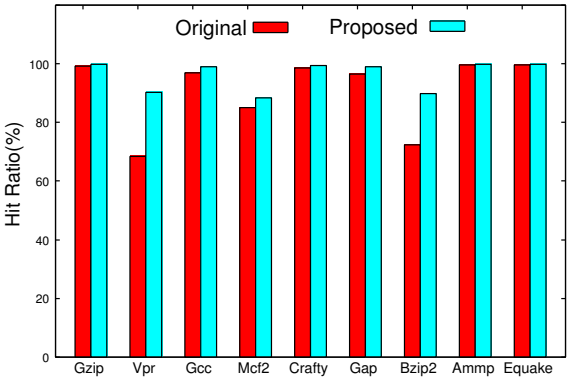

Figure 5. Hit ratio of TLB.

5. Evaluation

To evaluate the performance of the proposed approach, we develop a trace driven software simulator. We use nine benchmark programs from SPEC2000 benchmark suits. Figure 5 shows the evaluation results. The proposed TLB can achieve better hit rate than the original MIPS32 TLB.

We also design the TLB using SystemVerilog, and evaluate circuit scale, and confirm that the circuit scale of our proposed TLB is enough small.

6. Conclusion

We propose a compact and high performance TLB implementation method. In original MIPS32 processor, TLB is implemented using CAM which require large circuit scale. Our proposed method can implement TLB using standard cell library and general RAM macros. Furthermore, our approach can achieve 20% at the maximum and 6% in average better performance than the original MIPS32 processor without breaking binary compatibility.

As our future works, we will implement VLSI chip and evaluate performance and power consumption.

Acknowledgment

This work is supported by the VLSI Design and Education Center (VDEC) of the University of Tokyo in collaboration with Synopsys Inc., Cadence Design Systems Inc., Rohm Corporation., and Toppan Printing Corporation. This work is also supported by JSPS KAKENHI Grant Number 24700047 and 15K00074.

References

[1] N. K. Choudhary, et. al: “FabScalar: Composing Synthesizable RTL Designs of Arbitrary Cores within a

Canonical Superscalar Template", Proc. of the ISCA-38, pp. 11–22, June 2011.
[2] Hoang Le, Weirong Jiang, V.K. Prasanna, "Scalable high-throughput SRAM-based architecture for IP-lookup using FPGA", Proc. of the International Conference on Field Programmable Logic and Applications, pp.137–142, 2008.
[3] Jean-Louis Brelet: Using Block RAM for High Performance Read/Write CAMs, Xilinx application note, XAPP204, 2000.
[4] Kyle Locke: Parameterizable Content-Addressable Memory, Xilinx application note, XAPP1151, 2011.
[5] A.M.S. Abdelhadi, G.G.F. Lemieux: "Deep and narrow binary content-addressable memories using FPGA-based BRAMs", Proc. of the International Conference on Field-Programmable Technology, pp.318–321, 2014.
[6] Brandon H. Dwiel, Niket K. Choudhary and Eric Rotenberg: FPGA Modeling of Diverse Superscalar Processors, Proceedings of the 2012 IEEE International Symposium on Performance Analysis of Systems & Software, pp.188–199, 2012.
[7] S. Lekshmipriya, Suby Varghese: "FPGA Based Architecture for High Performance SRAM Based TCAM for Search Operations", International Journal of Science and Research, Vol.4, No.2, pp.1862–1867, 2013.
[8] Prajitha P. B.: "SRAM-Based TCAM Architecture for ATM", International Journal for Science and Advance Research In Technology, Vol.1, No.4, pp.64–67, 2015.
[9] Altera: "Implementing High-Speed Search Applications with Altera CAM", Altera Application Note 119, 2001.
[10] Xilinx: "Content-Addressable Memory", Product Specification DS253, 2008.
[11] Altera: "APEX 20KC Programable Logic Device Datasheet", Altera, 2001.
[12] Gun MUTO, Takahiro SASAKI, Yuki FUKAZAWA and Toshio KONDO: "Implementation method of CAM based TLB for standard cell base design," IEICE technical report, Vol.115, No.174, pp.1–6, 2015 (Japanese).
[13] Gun MUTO, Takahiro SASAKI, Yuki FUKAZAWA and Toshio KONDO: "Improvement of TLB performance of MIPS-based processor," IEICE technical report, Vol.115, No.374, pp.13-18, 2015 (Japanese).