World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:4, 2008

Hardware Implementation of Stack-Based Replacement Algorithms

Hassan Ghasemzadeh, Sepideh Mazrouee, Hassan Goldani Moghaddam,
Hamid Shojaei, and Mohammad Reza Kakoee

***Abstract*—**Block replacement algorithms to increase hit ratio have been extensively used in cache memory management. Among basic replacement schemes, LRU and FIFO have been shown to be effective replacement algorithms in terms of hit rates. In this paper, we introduce a flexible stack-based circuit which can be employed in hardware implementation of both LRU and FIFO policies. We propose a simple and efficient architecture such that stack-based replacement algorithms can be implemented without the drawbacks of the traditional architectures. The stack is modular and hence, a set of stack rows can be cascaded depending on the number of blocks in each cache set. Our circuit can be implemented in conjunction with the cache controller and static/dynamic memories to form a cache system. Experimental results exhibit that our proposed circuit provides an average value of 26% improvement in storage bits and its maximum operating frequency is increased by a factor of two

***Keywords*—**Cache Memory, Replacement Algorithms, Least Recently Used Algorithm, First In First Out Algorithm.

I. Introduction

IN modern computer systems the efficiency of the memory hierarchy is one of the most crucial issues. To obtain a high-performance memory system, caching is the best direction in terms of cost and effectiveness. The performance of the cache memories to compensate the speed gap between processors and main memory is determined by the access time and miss rates. Researches have proposed various cache memory replacement schemes to enhance cache performance and reduce its costs. [1-9]

The most cost-effective organization of cache memory called set associative cache holds several blocks per set. This organization is generally employed by cache designers as it offers a reasonable balance between hit ratio and implementation costs. When a cache miss occurs, the cache controller decides which block must be replaced. To make a replacement decision, designers exploit three basic replacement algorithms, *Least Recently Used* (LRU), *First In First Out* (FIFO) and *Random* policies, or employ some variations of these conventional schemes. Random replacement algorithm could be easily implemented using a Pseudo Random Generator circuit. Most efforts have been made to introduce efficient architectures of the true LRU and FIFO algorithms.

This paper presents a flexible architecture which can be used for implementation of both LRU and FIFO algorithms. Aforementioned architecture is an extended version of the Pseudo-FIFO which is previously introduced in [10]. Our architecture needs just some minor changes to move from one algorithm to another because we implement both control circuits in a stack structure. In the LRU algorithm, each cache set is treated as a stack and a reference to a particular block moves that block to the top of the stack. The least recently used block is always at the bottom and on a miss, new data will be loaded into this block and moved to the top of the stack. A block is also moved to the top on a cache hit because it is now the most recently used block. [14]

The rest of this paper is organized as follows. In Section 2, we present some related works. The details of our circuits are given in Section 3. In Section 4, we present the performance evaluation of proposed architecture, and finally we conclude the paper in Section 5.

II. Related Work

Cache memory replacement policies to decrease the miss rates have been widely studied in the past. Many replacement algorithms have been proposed and some of them such as the LRU and FIFO are extensively adopted in caches. Authors in [11] presented a defect-tolerant control circuit for a set associative cache memory. This circuit keeps stack ordering necessary for implementation of the LRU replacement algorithm in a 4-way set associative cache. This method that requires $n(n-1)/2$ bits of memory per set, called Reference Matrix as it revolves around the concept of a reference matrix. This architecture takes advantage of reasonable bits, but it has still low clock rate for practical purposes. The LRU scheme produces good results in terms of low miss rates, but there are some crucial design issues to exploit this algorithm in CPU caches. The most important point is that this policy can quickly consume plenty of memory elements as the associativity grows. Researchers in [1] introduced an idealized circuit of the LRU called Basic Architecture. The drawback with this architecture is that for higher associativities the number of bits grows significantly. The problems with

Manuscript received August 30, 2006. This work was supported in part by the Islamic Azad University. This research was done in the Department of Computer Engineering at the Islamic Azad University, Damavand Branch.

H. Ghasemzadeh, H. Shojaei and M. R. Kakoee are with the Department of Computer Engineering, Damavand Branch, Islamic Azad University, Damavand, Tehran, Iran (e-mail: h.ghasemzadeh@ece.ut.ac.ir and {shojai , kakoee}@cad.ece.ut.ac.ir).

S. Mazrouee was with the Islamic Azad University. She is now with the Oil Pension Fund Investment Co., Tehran, Iran (e-mail: s.mazrouee@gmail.com).

implementation of LRU led cache designers to employ an approximation of this scheme instead of its full structure. Authors in [12] designed a chip to implement LRU approximation scheme. In [13], authors presented a systolic LRU processor which requires $(3n\log_2^n + n)$ storage bits per set. Many new Intel processors like Pentium use a binary tree structure called Pseudo-LRU for implementing an approximation of the LRU in a 4-way set associative cache. Instead of 5 bits in Basic Architecture, Pseudo-LRU needs only 3 bits of memory elements per cache line. In [14], authors proposed an approximation to the LRU replacement scheme which has been implemented for video compression purposes. Also, authors in [15] reported an approximation of the LRU called Quasi-LRU in a 4-way set associative, 1MB and dual ported data cache memory. This circuit is the same Pseudo-LRU used by Intel that was pointed to before. We note that these architectures are very complex so that they could not be implemented and/or cascaded withal having more miss rates for approximation structures than true ones.

III. System Architecture

The original idea behind our architecture is to implement stack-based algorithms such as the LRU and FIFO in a stack structure. This stack contains *n* entries, each of which represents a block. We recall from Section 1 that the replaced block (actually, its address) is always stored at the bottom of the stack. In our architecture, there are $n\log_2^n$ bits ordered in *n* rows, each row consists $\log_2^n$ bits, altogether form a control stack. Fig. 1 exhibits a top level block diagram of system architecture for a 4-way set associative cache. Each row in stack circuit refers to corresponding block in a cache set. The power-up state of the stack is significant since initial values indicate stack order at the startup time. A *precharge* signal which should be activated at the startup is applied to the circuit to convey the memory elements to an initial state.

When the CPU makes a memory call and the cache receives it, the tag field of CPU address is associatively searched in *tag-RAM*. The *tag-RAM* has some output signals that determine the address on which a hit occurred, or a miss signal becomes high to indicate that the referred block is not in the cache. Therefore, in a 4-way set associative cache, five bits are issued due to cache controller output as the inputs to control circuit to show the status of current accessed block. The four first bits, *match<0>* thru *match<3>* correspond to a row of the stack on a hit. When a miss occurs, the *match* signals are high impedance and the *miss* is used to update the stack.

A. LRU Control Circuit

The stack circuit is updated during each clock cycle to maintain blocks order in corresponding cache set. In an LRU circuit, three different situations may happen during a memory access:

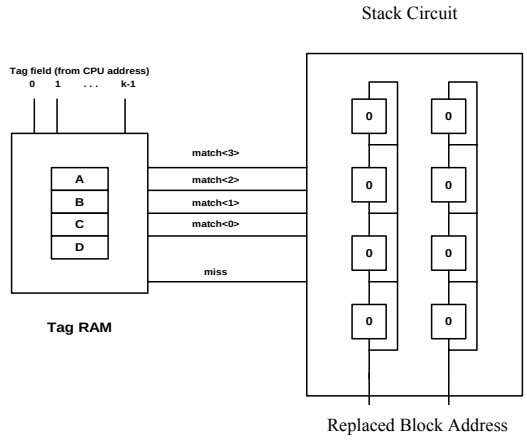

Fig. 1 Top Level Block Schematic of System Architecture

- When an access to most recently used block occurs, no change in stack order should happen. The MRU block is always at the top location of the stack. Therefore, on an access to most recently used block, the stack is not needed to be updated.
- If requested block is not found in the cache, a cache miss has occurred. The *miss* signal becomes high through the *tag-RAM* output. In this situation, LRU block whose address is at the bottom is sent to the cache controller. Consequently, the requested block is brought from main memory and copied to appropriate space in the cache. Now, the address at the bottom refers to the most recently accessed block. Therefore, on a cache miss, the address at the bottom of the stack should be moved to the top of the stack indicating most recently used block, and the other addresses should be shifted down once in the stack.
- If requested block is found in every place in the cache set except the top of the stack, a hit has occurred and the stack should be updated. In this case, the row which contains the address of accessed block should be moved to the top and all the upper rows will be shifted down.

In Fig. 2, we show the control circuit of the LRU replacement scheme for a 4-way set associative cache. The circuit has *n+1* inputs and $\log_2^n$ outputs, where *n* is associativity. The outputs indicate the block which has been used least recently. The inputs include *n* lines called *match* and one line called *miss*. The *match* lines are compared with different rows of the stack. The outputs produced by one comparator show the equality of *match* inputs and stack rows as the stack contains all variations of *match* lines. The *match* inputs are equivalent to just one row in the stack. The comparator outputs force the *enable* inputs of the flip-flops to be activated.

On a hit, the rows above the row where the hit occurred must be shifted down to update stack order. We have used cascaded OR gates to activate *load enable* of the flip-flops. When the *miss* input becomes high, in addition to move the bottom cells to the top, all upper rows are transferred down. Therefore, the *miss* signal activates the *enable* input of the last row and propagates through the OR gates to activate the *load enable* of the upper rows.

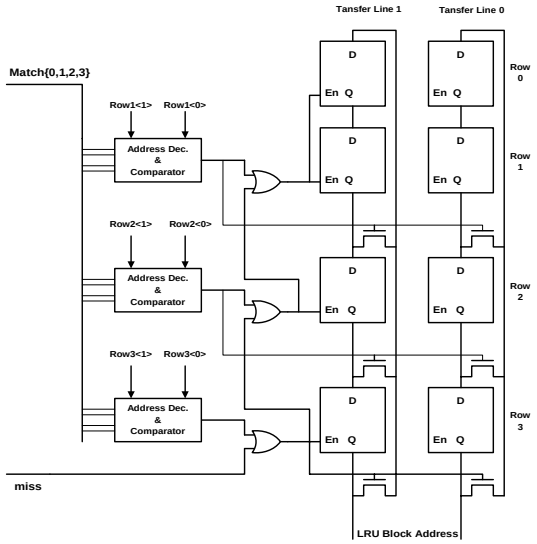

Fig. 2 LRU control circuit for a 4-way set associative cache memory

In this architecture, there are $\log_2^n$ transfer lines that move row content to the top. The row is selected through comparator outputs and there are some nMOS switches in each line that move stack content to the transfer lines. On a row match, the load enables of the matched row and all upper rows become high and the content of the matched row appears on the transfer lines. When system clock arrives, the transmigration is done and the stack is updated. Now, the stack is stable, MRU block address is at the top and LRU block address could be found at the bottom. The same scenario occurs at the next memory access.

The memory cells are normal D-flip-flops with *enable* inputs. In Fig. 2, it is apparent that the lines to *En* on the flip-flops on the left are meant to continue to the adjacent flip-flop on the right. At the power-up state, *precharge* signal sets flip-flop outputs to appropriate values to form initial order of the LRU stack. As we saw before, each LRU circuit consists of some flip-flops and associated combinational logic circuits which update status of the flip-flops.

B. FIFO Control Circuit

To update stack order in a FIFO structure, there are two different situations when a memory call occurs:

- On a cache miss, the address at the bottom of the stack refers to the first block brought to the corresponding cache set. Therefore, the address at the bottom should be moved to the top of the stack indicating first arrived block, and the other addresses should be shifted down once in the stack.
- On a cache hit, the stack order should be kept as it has been before a memory reference. Actually, on a hit, the first arrived block dose not change.

Fig. 3 shows the control circuit of the FIFO algorithm for a 4-way set associative cache. This circuit has only 1 input (*miss* signal), while it has, like LRU, $\log_2^n$ outputs. The outputs indicate the first arrived block which should be replaced with the new block.

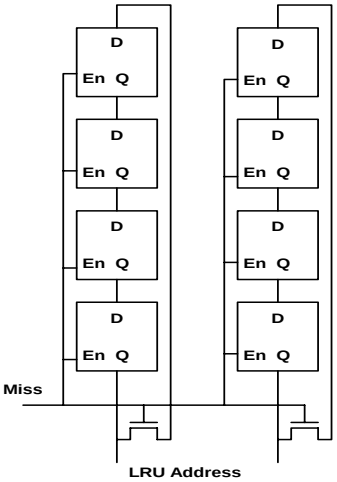

Fig. 3 FIFO control circuit for a 4-way set associative cache memory

IV. Performance Evaluation

A. Miss Rates Analysis

We employ trace-driven simulations with several kinds of workloads to evaluate LRU and FIFO algorithms and compare them with other common schemes such as Random and Pseudo-LRU. Hence, the cache simulator is developed to support all aforementioned replacement algorithms. Workloads used in this trace-driven simulation include fifteen different types of programs (*dec0, fora, forf, fsxzz, lvex, lisp, macr, memxx, mul2, mul3, pasc, ue02, spice, gcc, tex)* including more then 3,000,000 references. Both data and instruction references are collected and used for simulation. We used 32-byte block size in a 32 KB unified cache and evaluated the replacement schemes in terms of miss rates. Fig. 4 demonstrates the resulting performance with different associativity values. In this diagram, each data-point corresponds to the average taken over all aforementioned benchmarks.

It is shown that LRU algorithm provides average values of 13.49% and 8.46% improvement over Random and Pseudo-LRU algorithms. Also, FIFO algorithm has an average 4.31% performance improvement over Random algorithm. Although, Pseudo-LRU provides a small improvement of 1.24% over FIFO, it has more hardware complexity compared to FIFO scheme. Therefore, there is still a considerable performance gap between LRU/FIFO and Random/Pseudo-LRU replacement schemes that leads us to implement new architectures of LRU and FIFO.

B. Time Complexity Analysis

We consider operating frequency as an important parameters in evaluating timing analysis of our architecture. A comparison is made between Reference Matrix circuit [11,16] and our LRU circuit in terms of operating frequency. We do not include Basic Architecture of the LRU algorithm in this evaluation because this architecture is indeed an idealized structure of LRU and mainly impossible to be implemented

for practical purposes.

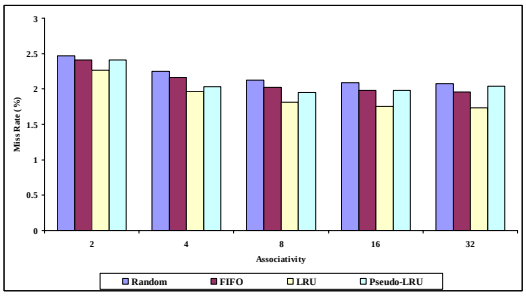

Fig. 4 Miss rates versus associativity for different algorithms in a 32KB cache memory

We use 2-bit Pseudo-nMOS comparators which have only one output to be high when two inputs are equal. The circuit is very fast and small [17-18]. These advantages become very important in highly associative caches. Fig. 5 shows the circuits we employed for performance evaluation.

To calculate the maximum operating frequency of the LRU circuit, we need to measure time intervals t_1, t_2 which are described as follows.

- Since one of OR gate inputs is connected to the *enable* of the flip-flop from the lower row, whenever En_{i+1} is set to 1, it propagates through the OR gate and sets En_i to 1. Meanwhile, this new value of En_i sets En_{i-1} to 1 and so on. The upper bound of the time interval for each change is t_1.

The generation of LRU address is triggered by system clock. At the rising edge of the clock, the flip-flop is activated and updated. After updating flip-flops, the LRU index will be presented at the bottom of the stack. The clock can be applied as soon as the *enable* inputs become valid. The delay between activating *enable* signals and updating stack order is referred to as the time interval t_2.

The worst case delay is used to calculate the maximum operating frequency. This case may happen when a miss occurred. In this situation, the *miss* signal becomes high and shall propagate through OR gates to set enable inputs of flip-flops to 1. The total number of OR gates in this sequence is *n-1* where *n* is set associativity. Thus, in the worst case, the circuit needs $(n-1)\times t_1$ to update the *enable* signals. We recall that the system clock triggers the generation of the LRU address which results in updating memory cell from its last value to its current value i.e. the last row now contains the index of the LRU block, so the maximum delay is the maximum value between $(n-1)\times t_1$ and t_2. The following equation describes the relation between *T*, time interval, and *n*, associativity in LRU circuit:

$$T_{LRU} = \text{Max}[(n-1)\times t_{OR} + t_{DFF}] \quad (1)$$

In Fig. 6, we illustrate the difference between our LRU and Reference Matrix circuits in terms of operating frequency. We employed CMOS transistor models for Spice simulations, but the results could similarly be generalized. To compare our architecture against Reference Matrix, we consider four different sized of cache set (4, 8, 16, 32). Overall, the average maximum operating frequency of our LRU circuit is 573 MHz whereas this value is 220 MHz for Reference Matrix architecture.

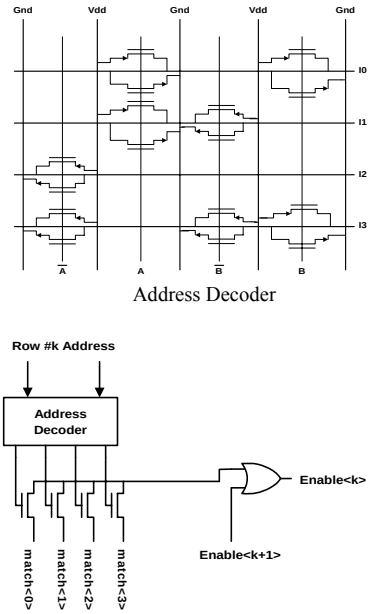

Fig. 5 A sample 2-bit Pseudo-nMOS comparator

C. Discussion on Hardware Complexity

To get insights into the superiority of our circuit over Reference Matrix in terms of hardware complexity, we notice that memory cells are the main area consuming elements. As we illustrated in Section 3, the number of storage bits per set in implementation of the LRU and FIFO algorithms is equal to $n\log_2 n$ for an n-way set associative cache. Hence, the number of memory elements per set in 4, 8, 16 and 32-way set associative cache corresponds to 8, 24, 64 and 160 respectively. As the memory cells are added, the chip area increases and so the overall space of the stack which would be fixed to the chip is in $n\log_2 n$ order. It can be inferred that the area complexity of our architecture would be $O(n\log_2 n)$ where *n* is the number of blocks per cache set. On the other hand, a Reference Matrix circuit as an important structure of true LRU scheme requires $n(n-1)/2$ memory cells to keep the $n!$ possible stack formations. Therefore, the area complexity of Reference Matrix circuit is $O(n^2)$. To test the effectiveness of our technique over the Basic Architecture, the most important point is that although the Basic Architecture consumes small number of memory elements, it is nearly impossible to be implemented due to very complex circuit.

It is revealed that our presented circuit outperforms former architectures. In particular, our architecture provides an average value of 26% performance improvement in storage bits over Reference Matrix circuit. This improvement would be much noticeable especially when the number of blocks per set increases. One of the most significant features of our technique in implementation of the LRU and FIFO algorithms is expandability for high associativities which makes hardware

implementation much easier that former circuits proposed in literatures.

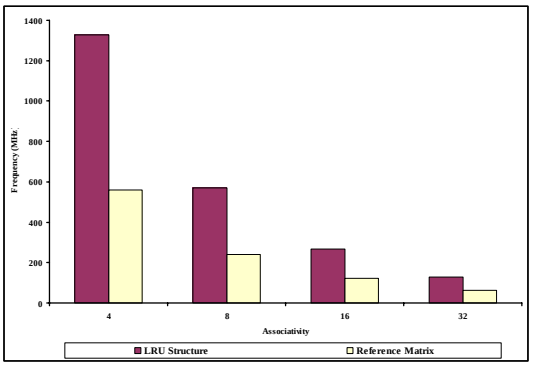

Fig. 6 Operating frequency vs. associativity for our circuit and reference-matrix

V. Conclusion

In this paper, we introduced an extended version of Pseudo-FIFO organization of LRU replacement policy. Our flexible architecture can be readily employed in VLSI implementation of both LRU and FIFO schemes. Results achieved using Spice simulator show that our circuit can provide significant performance improvements with respect to traditional architectures such as Reference Matrix and Basic Architecture.

Acknowledgment

We highly thank Dr. M. Safari, the dean of Damavand Branch, for his support. We thank Dr. H. Hossein, Mr. M. Ganji and Mr. M. Ranjbari as well for providing us helps in doing this research in the Department of Computer Engineering at the Islamic Azad University, Damavand Branch.

References

[1] J. Handy, The Cache Memory Book, Academic Press, San Diego, pp. 47-67, 1993.
[2] H. Ghasemzadeh, S. Mazrouee and M. R. Kakoee, Modified Pseudo LRU Replacement Algorithm, 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS), pp. 368-376, March 27-30 2006.
[3] R. Pendse, Pipeline LRU Block Replacement Algorithm, Proc. 43rd Midwest Symp. On Circuits and Systems, Lansing MI, Aug. 8-11, 2002.
[4] J. Jeong and M. Dubios, Cost-Sensitive Cache Replacement Algorithms, 9th International Symposium on High-Performance Computer Architecture (HPCA-9'03), 2002.
[5] S. Jiang, X. Zhang, Making LRU Friendly to Weak Locality Workloads : A Novel Replacement Algorithm to Improve Buffer Cache Performance, IEEE Transactions on Computers, Vol. 54, No. 8, Aug. 2005.
[6] R. Pendse, Pipeline LRU Block Replacement Algorithm, Proc. 43rd Midwest Symp. On Circuits and Systems, Lansing MI, Aug. 8-11, 2002.
[7] D. Lee, et. Al., LRFU : a spectrum of policies that subsumes the least recently used and least frequently used policies, IEEE Transaction on Computers, vol. 50, no. 12, pp. 1352-1361, 2001.
[8] A.W. Wayne and J.L. Baer, Modified LRU Policies for Improving Second-Level Behavior, proceeding 6th International Symposium on High-Performance Computer Architecture, pp. 49-60, 2000.
[9] Yoon, J., Min, S.L., and Cho, Y., Buffer cache management: predicting the future from the past, International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'02), pp. 92-97, 2002.
[10] H. Ghasemzadeh and S. O. Fatemi, Pseudo-FIFO Architecture of LRU Replacement Algorithm, Proc. 9th IEEE International Multi Topic Conference (INMIC), December 23-25, 2005.
[11] D. Lamet and J.F. Frenzel, Defect-Tolerant Cache Memory Design, IEEE Transactions on Computers, April 1993.
[12] B. Beridegard, B. Nilsson and L. Philipson, VLSI Implementation of A Virtual Memory Paging Algorithm, VLSI: Algorithms and Architectures, Elsevier Science Publishers B.V., pp. 167-174, 1985.
[13] W. Luk and G. Brown, A Systolic LRU Processor and Its Top-Down Development, Science of Computer Programming, Vol. 15, pp. 217-233, 1990.
[14] O. Fatemi, F. Idris and S. Panchanathan, FPGA Implementation of the LRU Algorithm for Video Compression, IEEE 1994 International Conference on Acoustics, Speech, and Signal Processing, pp. 337-344, June 1994.
[15] K.A. Hurd, A 600MHZ 64b PA-RISC Microprocessor, ISSCC Digest of Technical Papers, pp 94-95, February 2000.
[16] K. Maruyama, mLRU Replacement Algorithm in terms of the Reference Matrix, IBM Technical Disclosur Bulletin, pp. 3101-3103, March 1975.
[17] J.M. Rabaey, Digital Integrated Circuits, A Design Perspective, Prentice-Hall Electronics and VLSI Series, pp. 332-381, 1996.
[18] Weste N.H.E. and Eshraghian K., Principles of CMOS VLSI Design, A System Perspective, Second Edition, Addison-Wesely Publishers Company, pp. 513-620, 1994.

Hassan Ghasemzadeh received the BSc degree in Computer Engineering from the Sharif University of Technology, Tehran, Iran, and the MSc degree in Electrical and Computer Engineering from the University of Tehran, Tehran, Iran, in 1998 and 2001 respectively. Currently, he is a faculty member and director of undergraduate studies in the Department of Computer Engineering at the Islamic Azad University, Damavand Branch.