SEMANTIC-BASED LOAD/STORE SCHEDULING FOR X86 SUPERSCALAR MICROPROCESSOR

Kuen-Cheng Chiang, R-Ming Shiu, and Jean Jyh-Jiun Shann

Department of Computer Science and Information Engineering,
National Chiao Tung University, Hsinchu, Taiwan 30050, R.O.C.
E-mail: jjshann@csie.nctu.edu.tw

Abstract

For x86 compatible processors, the proportion and latencies of memory accesses are relatively high thus impact the performance of these processors severely. In this paper, we propose a semantic-based load/store scheduling to alleviate these limitations. In x86 architecture, most of the local variables and parameters of a function are stored in stack memory. We find that the addresses of stack accessing operations will be the same if the displacements of these instructions are the same. Therefore, we may track the dependencies and forwarding paths between the stack accessing operations according to the displacement values of the operations. From our simulation results, the speedup of semantic-based load/store scheduling alone can achieve 1.47 compared with the strategy of load bypassing stores with forwarding. While combing this scheduling with selective address/dependency prediction, it can achieve the speedup of 1.70

.

1. Introduction

An x86 compatible processor, the most widely used general-purpose architecture, is a complex-instruction-set- computer (CISC) processor with complex instruction formats, variable instruction sizes, and complex memory addressing modes. Due to the complexity of instruction formats and variety of instruction execution time, most of the modern x86 compatible processors, such as Intel Pentium and Pentium II/III, AMD K5/K6/K7, and Cyrix M1/M2, convert an x86 instruction into one or more primitive operations (POPs) with fixed-length format and equal execution time in order to fit the complex superscalar pipelines and out-of-order mechanisms [1-5].

Load/store operations especially affect the performance of an x86 processor most greatly. Load/store operations appear more frequently in x86 micro-architectures, about 45% to 55% of the total operations in a program, than those appear in RISC architectures. Besides, load/store operations have more execution latencies than other operations because of the increasing gap between processors and memories. As a result, load/store operations influence the performance of an x86 processor much more severely than other operations.

For the consistency of memory, store operations are executed in the original program order. However, load operations can be executed without obeying the original program order, and thus several scheduling policies of memory accesses such as *load bypassing* and *load forwarding* have been developed [2]. In load bypassing, the address of a load is checked with the addresses of its previous stores. If there is no conflict, the load can be issued to data cache. Otherwise, the load must be held until the conflicted stores are issued. This policy has been used by the current generation of the x86 microprocessors. As for load forwarding, the data of the conflicted store can be forwarded directly to the pending load.

However, in these conservative scheduling policies, a load cannot be issued or forwarded if any address of its previous stores is unsolved, i.e. has not

* This paper presents partial result of a long-term research project financed by the NSC of R.O.C. under contract no. NSC 89-2213-E-009-066.

been generated [6]. Such unsolved address problem may cause the performance degradation of a superscalar microprocessor. For an x86 superscalar microprocessor, this problem becomes much severer because the pipeline is lengthen for address calculation.

Many prediction techniques, such as *address prediction* [7], *dependency prediction* [8-10], and *value prediction* [7, 11], have been developed on RISC for resolving the unsolved address problem. These techniques predict the addresses, dependencies, or even data of load operations at the fetch stage. However, when applying to x86 processors, which generally have longer pipelines than RISC microprocessors, all these techniques have to suffer the lengthen penalty of prediction errors and thus may not work effectively. Besides, Smith showed that the predictability of value prediction is very low [7]. Lai has proposed an *address/dependency prediction* in [12] to enhance the x86 prediction for loads by combining the address prediction and the dependency prediction, and improve the dependency prediction by adding the forwarding prediction ability, i.e., refining the predictions with a set of 2-bit counters and filtering out the error-like predictions with another set of 2-bit counters. For further improving the prediction of loads, we develop the *semantic-based load/store scheduling* in this paper to predict the load data according to the plentiful semantics of x86 instruction set.

In x86 architectures, for a memory-type source or destination operand, it is referenced by means of a segment selector and an offset as shown in Fig. 1(a). The address computation is made up of one or more of components: base, index, scale factor, and displacement. The offset that results from adding these components is called an effective address. Fig. 1(b) shows all the possible combination of these components for creating an effective address in the selected segment.

Load/store operations can be classified into two categories: stack and data accessing operations. A stack accessing operation, such as push, pop, and BP-based stack operation whose base register is BP, can only access the data in the stack segment specified by segment register SS; whereas a data accessing operation can only access the data in the data segment specified by segment register DS or ES.

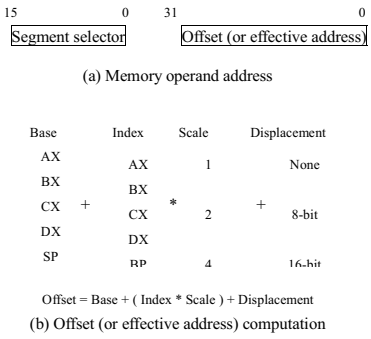

Fig. 1. Memory addressing in x86 architectures.

Because of the lack of registers in x86 architectures, the local variables and function call parameters are always placed in the stack memory. Hence, a large portion of load/store operations is used to access the stack memory. Besides, these variables and parameters are usually kept in the same relative locations in stack memory every time the function is called. The distance between SP pointer and the address of each variable or parameter is always the same and encoded into the displacement of a load/store operation. This feature helps us to predict the address of a stack accessing operation.

The *semantic-based load/store scheduling* tracks the dependencies and forwarding paths of stack operations to improve the accuracy rate of prediction and avoid time-consumed address calculation. We design a *framed-stack buffer* (FSB) to track the update information of all BP-based stack operations according to their displacement values. In Decode stage, we allocate a new stack frame whenever a CALL operation occurred. On the other hand, we free the current stack frame and turn back to the previous one when a RETURN operation occurred.

The organization of this paper is as follows. In Section 2, we propose our design of semantic-based load/store scheduling. Then we evaluate and analysis the performance in different scheduling policies in Section 3. Finally, we make conclusions in Section 4.

2. Models of Semantic-Based Load/Store Scheduling

In this section, we develop the semantic-based load/store scheduling, and combine it with address/dependency scheduling. We construct a

model of x86 superscalar microprocessors. The block diagram and the pipeline stages of this model are depicted in Fig.2.. X86 instructions are fetched and decoded into POPs. Distributed reservation stations (RSs) are used in our discussion. The POPs with ready operands and available functional units are issued out of order from the RSs to the execution units. The POPs are completed out-of-order while the in-order state of the program is kept in the reorder buffer (ROB). The results of POPs are retired orderly by the ROB. For simplicity, the execution units are divided into three categories only: ALU, branch unit (BU) and load/store unit (LSU).

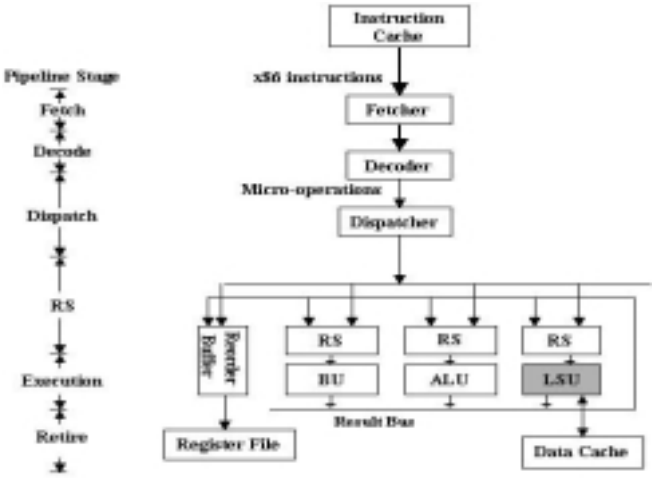

Fig. 2. The block diagram and pipeline stages of x86 superscalar microprocessors.

2.1 Semantic-Based Load/Store Scheduling

The block diagram of a superscalar processor with semantic-based scheduling is shown in Fig.3. When a micro-operation is decoded, the type of the operation is checked. If the operation is a BP-based operation, it will be sent to the Framed-Stack Buffer (FSB) and the Prediction Valid Buffer (PVB). According to the displacement of the BP-based operation, the data, dependency and address of the operation is predicted and stored into the PVB. If the data of the operation is predicted, the data will be sent to the result bus. After the operation is entered the LSU, the predicted information will be checked. If the prediction is wrong and the predicted data is sent to the result bus, the operation must be recovered and the predictor must be updated. The behavior of the semantic-based load/store scheduling will be discussed more detailedly in the following paragraphs.

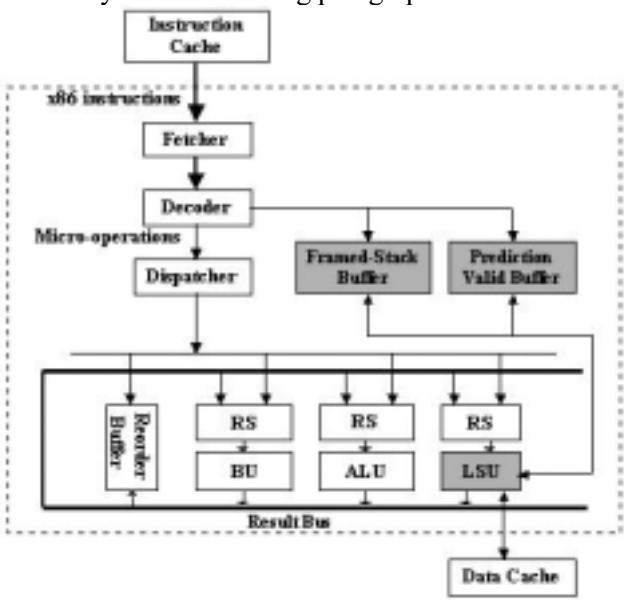

Fig. 3. The block diagram of a superscalar processor with semantic-based load/storescheduling.

While a program is compiled to x86 instructions, the local variables and parameters of a function are allocated in stack memory, and the BP pointer register is pointed to the top of the stack. Hence, the distance between the address of a local variable or a parameter and BP is constant. Compiler will encode the distance into the displacement of an operation. According to this feature, we design a mechanism, called framed-stack buffer (FSB), to track the dependencies and forwarding paths of BP-based operations.

As shown in Fig. 4, the FSB is designed as multiple frames, called stack frames, and selected by a frame selector. The frame selector points to the frame that is currently used. When the program enters into a subroutine, a new stack frame will be allocated by incrementing the value of the frame selector. The frame selector is decremented when the program exits the subroutine and returns back to the previous stack frame.

As shown in Fig. 5, an entry of a stack frame contains five fields: *valid* (*v*), *tag*, *depend micro-PC* (*dep μPC*), *address* (*addr*), and *data*. The *v* and *tag* fields are used for indicating whether the entry is valid or not and what the displacement stored in the entry is, respectively. The *dep μPC* field is used to indicate which operation that stores data in the entry is. And the *addr* field stores the address of the

memory access of the operation that stores data into the *data* field. The *data* field is used for storing data of the dependent load operation or store operation when the data is ready. On the other hand, the PVB is used to store the predicted information and is implemented as a first-in-first-out queue. Each entry in PVB contains five fields: *valid* (*v*), *micro-PC* (*μPC*), *depend micro-PC* (*dep μPC*), *address* (*addr*), and *state*. The *v*, *dep μPC* and *addr* fields are the same as the fields in the stack frame. The *μPC* field is the micro program counter that indicates which BP-based operation the entry belongs to.

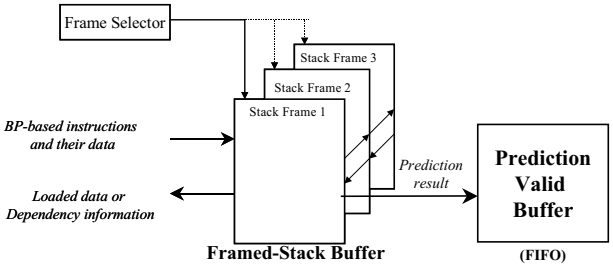

Fig. 4. The mechanism for semantic-based load/store scheduling.

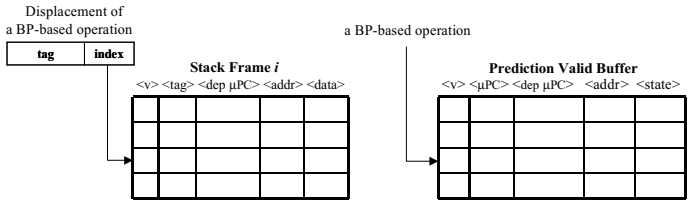

Fig. 5. The block diagram of a stack frame and the prediction valid buffer.

When a BP-based operation enters Decode stage, its displacement is used to index an entry in the current stack frame. The BP-based operation is processed in two cases. In the first case, the BP-based operation (the micro-PC is op_1) is not found in the stack frame or is a store operation. The operation will update the *tag* and *dep μPC* fields of the indexed entry in the stack frame as shown in Fig. 6. The entry indicates that it is depends upon the operation whose micro-PC is op_1, and the data and address of this operation will be sent to this entry according to the *dep μPC* field. The BP-based operation is also sent to the PVB and an entry is allocated for this operation. The *μPC* and *state* fields of this PVB entry are set as "op_1" and "initial", respectively. In the second case, the BP-based operation (the micro-PC is op_2) is found in stack frame and is a load operation, the contents of the stack frame and PVB are shown in Fig. 6. If the *data* field of the indexed entry of the stack frame contains the data of micro operation op_1, the data will be sent to the result bus. At the same time, the PVB allocates a new entry for this operation and stores the prediction information in it. The entry in PVB indicates that the micro operation op_2 depends upon micro operation op_1. Besides, the *state* field of this entry is set as "predict". The contents of the stack frame and PVB are shown in Fig. 7.

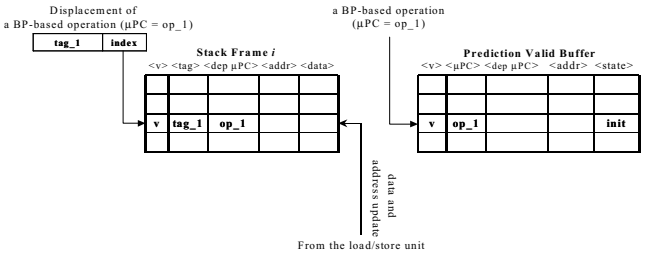

Fig. 6. The actions of the stack frame and the PVB when the BP-based operation is not found in the stack frame or is a store operation.

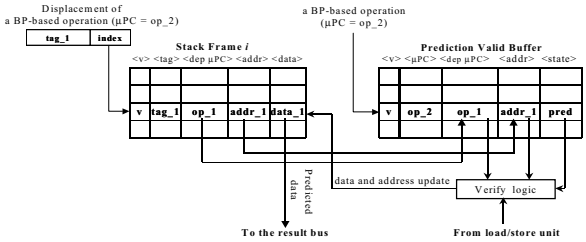

Fig. 7. The actions of the stack frame and the PVB when the BP-based load operation is found in the stack frame.

After a BP-based operation enters Execution stage, it will check the *state* field of its corresponding entry in PVB and store the data and address of the operation into the entry of the stack frame if the *state* field of the entry is "initial". On the other hand, the calculated address of the BP-based operation and the *addr* field of the corresponding entry in PVB will be sent to the verify logic, if the *state* field of the entry is "predict". If the comparison of these two addresses in the verify logic is not matched, the predicted load operation need to be recovered and the entry within the stack frame will be updated. Otherwise, the load operation is retired without accessing the data cache.

2.2 Classified Semantic-Based Load/Store Scheduling

In this subsection, we propose a mechanism that combines the techniques of semantic-base scheduling and address/dependency scheduling, called the classified semantic-based load/store scheduling (CSB). As shown in Fig. 8, after the load/store operations have been decoded, they will be sent to the classifier in the classified semantic-based load/store predictor. The classifier will identify the operations and then send the BP-based load/store operations to the semantic-based predictor and the other load operations to the address/dependency predictor. As for those store operations that are not BP-based, the classifier will ignore them. Therefore, for a predicted load/store operation, either the semantic-based predictor or the address/dependency predictor will make a prediction and save the predicted result into the shared PVB. If the data of a load operation is ready, it will be sent to the result bus speculatively. On the other hand, if the address of a load operation is ready, the address will be sent to the data cache for executing the load operation speculatively.

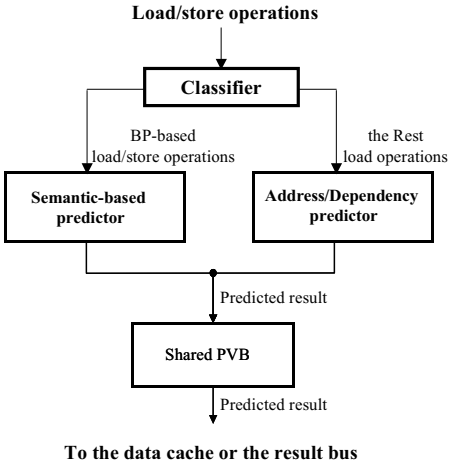

Fig. 8. The mechanism of the classified semantic-based load/store scheduling.

3. Performance Evaluation and Analysis

In this section, we evaluate the semantic-based (SB) and classified semantic-based (CSB) load/store scheduling models developed by us with simulation results.

3.1 Simulation Environment

In this paper, we use a trace driven simulator to evaluate the performance of various scheduling models. The flow chart of the simulation environment is shown in Fig. 9. The source codes of SPECint95 benchmarks [13] are compiled with GCC compiler. After that, the executable benchmarks along with the input files are executed on Linux OS in x86 compatible PCs. Then, the traces of the benchmarks are retrieved by the system call "ptrace()". Finally, the traces retrieved are read into our simulator to evaluate the performance of the simulation models proposed by us.

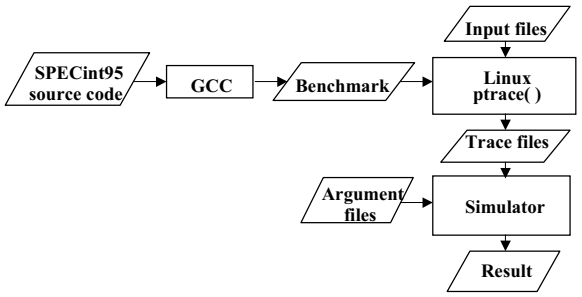

Fig. 9. Flow chart of the simulation environment.

To focus our discussion on the design issues of the load/store units and ignore the constraints of other parts in the microprocessors, we make some assumptions in our simulation as follows:

1. The issue way of superscalar pipeline is 8.
2. The accuracy of branch prediction is 100%.
3. The number of ALUs and BUs are unlimited and their latencies are 1 cycle.
4. The reservation station of each functional unit is 64-entry.
5. A single 16-issue load/store unit is provided.
6. The level one cache is 16-KB 4-way set-associative with 16 access ports. The latency is 1 cycle, if the access is hit. Otherwise, the latency is 10 cycles.

3.2 Evaluation of Semantic-Based Load/Store Scheduling

As shown in Fig. 10. The proportion of BP-based operations in load/store micro-operations., the proportion of BP-based operations is greater than 60% in all load/store operations. According to the features of x86 instruction sets, these BP-based operations appear because of the use of the local variables and function call parameters. The dependencies between these operations exist most of the time, and become

the bottleneck of address prediction. Besides, because of the dependency, even the address is predicted correctly, the preloaded data of this address may also be unusable. In our proposed mechanism, we use the framed-stack buffer to track the dependencies between these load/store operations and keep the sequence order of these operations in prediction valid buffer (PVB) to avoid the anti-dependencies and output dependencies. As a result, this mechanism can only be implemented in Decode stage because that it should keep the order of load/store operations and check whether a CALL or RET operation is occurred in order to switch the frame currently used.

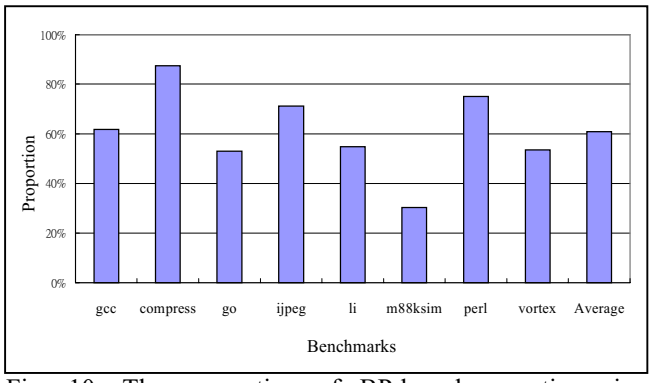

Fig. 10. The proportion of BP-based operations in load/store micro-operations.

3.2.1 Semantic-Based Scheduling

In this subsection, we discuss the semantic-based scheduling in two models. The first one is the *semantic-based* (SB) scheduling model, which uses the semantic-base predictor to predict the address and the value of a data. As we mentioned in Section 2, the SB predictor can only work for the BP-based operations. In other words, about 40% of the load/store operations cannot be handled in this model. Hence, we combine the semantic-based predictor and the selective address/dependency predictor to be the second model, called the *classified semantic-based* (CSB) *scheduling*. In this model, the semantic-based predictor is in charge of BP-based operations and the selective address/dependency predictor is in charge of the other load operations.

The performance of SB and CSB scheduling models with unlimited semantic-based predictor is shown in Fig. 11. we see that the performance of SB and CSB can achieve the speedup of 1.52 and 1.77, respectively, with respect to the strategy of load bypassing stores with forwarding. Comparing with the selective address/dependency prediction (SADP) proposed by Lai, which has the best performance gain in traditional predictions and achieves the speedup of 1.33, SB and CSB have higher performance.

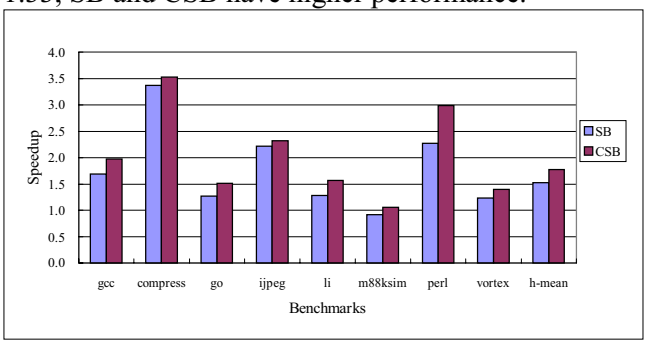

Fig. 11. The performance of semantic-based scheduling models.

3.2.2 Semantic-Based Scheduling with Limited Frame-Stack Buffer

In this subsection, we discuss the impact of performance caused by the size of the stack frames in the frames-stack buffer. We simulate the relationship between the frame size and the performance of a processor. In the simulation models, there are eight frames in the framed-stack buffers and the size of a frame is from 16 to 512. As shown in Fig. 12, the performances of SB and CSB with frame size that is bigger than 128 entries are saturated. Therefore, we suggest that 128 entries for each frame are enough.

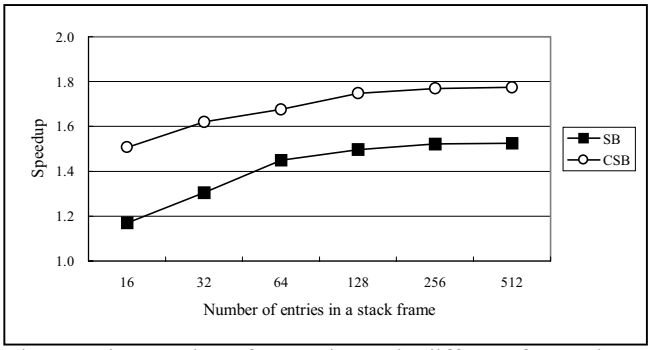

Fig. 12. The speedup of SB and CSB in different frame size. Assume that there are 8 stack frames in each model.

4. Conclusions

In this paper, we propose a *semantic-based load/store scheduling* to reduce the latencies of load operations. This scheduling scheme is focus on how to track the dependencies and forwarding paths between load/store instructions. Unlike other

strategies, the semantic-based load/store scheduling not only forwards data from store operations to load operations but also reuses the data between load operations.

According to our simulation result, we suggest that two 128-entry stack frames is a good choice for the tradeoff between hardware cost and performance of a processor in either *semantic-based* (SB) scheduling or *classified semantic-based* (CSB) scheduling. The speedup of these SB and CSB can achieve 1.47 and 1.70, respectively, compared with the strategy of load bypassing stores with forwarding scheduling. Using a single stack frame only in the semantic-based predictor is also an option for reducing cost and complexity in hardware and control.

The prediction of semantic-based scheduling focuses on how the reusing or forwarding does within a function call. According to the feature of x86 instruction set, the parameters of a function are pushed into the stack memory before the function is called. But the tracking of parameters between the caller and the callee is limited in the value of stack pointer (SP) register. Therefore, in the future, we will focus our attention on resolving the value of the stack pointer to explore the potential performance caused by passing parameters.

References

[1] D. A. Patterson and J. L. Hennessy, Computer Architecture: A Quantitative Approach, 2nd Ed., Morgan Kaufmann, 1996.

[2] Mike Johnson, Superscalar Microprocessor Design, Prentice Hall, 1991.

[3] L. Gwennap, "Intel's P6 Uses Decoupled Superscalar Design," Microprocessor Report, Vol. 9, No. 2, Feb. 16, 1995.

[4] M. Slater, "AMD's K5 Designed to Outrun Pentium," Microprocessor Report, Vol. 8, No. 14, Oct. 24, 1994.

[5] L. Gwennap, "Cyrix Describes Pentium Competitor," Microprocessor Report, Vol. 7, No. 14, Oct. 25, 1993.

[6] M. Franklin and G. S. Sohi, "ARB: A Hardware Mechanism for Dynamic Reordering of Memory References," IEEE Transaction on Computers, vol. 45, no.5, pp. 552-571, 1996.

[7] Y. Sazeides and J. E. Smith, "The Predictability of Data Values," In the Proceeding of Micro-30, December 1997.

[8] H.Y. Hwang, An X86 Load/store Unit with Aggressive Scheduling of Load/store Operations, Master Thesis, Department of Computer Science and Information Engineering, College of Electrical Engineering and Computer Science, National Chiao Tung University, 1998.

[9] A. Moshovos, S. E. Breach, T. N. Vijaykumar, and G. S. Sohi, "Dynamic Speculation and Synchronization of Data Dependences," In Proc. of the 24th Annual International Symposium on Computer Architecture, 1997.

[10] G. Z. Chrysos and J. S. Emer, "Memory Dependency Prediction using Store Sets," In Proc. of the 25th Annual International Symposium on Computer Architecture, 1998.

[11] M. H. Lipasti, C. B. Wilkerson, and P. S. John, "Value Locality and Load Value Prediction," In ASPLOS-VII, October, 1996.

[12] S.J. Lai, Speculative Load/Store Scheduling for x86 Superscalar Microprocessors, Master Thesis, Department of Computer Science and Information Engineering, College of Electrical Engineering and Computer Science, National Chiao Tung University, 1999.

[13] S. E. Chang and Y. R. Chang, "A Study of SPEC CPU95 Benchmarks," Technical Report, Department of Information and Computer Engineering, Chung Yuan University, 1996.