JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 17, 787-803 (2001)

Aggressive Scheduling for Memory Accesses of CISC Superscalar Microprocessors

R-MING SHIU, HUI-YUE HWANG AND JEAN JYH-JIUN SHANN
Department of Computer Science and Information Engineering
National Chiao Tung University
Hsinchu, 300 Taiwan
E-mail: jjshann@csie.nctu.edu.tw

For CISC microprocessors, the proportion of memory access instructions is relatively high, and a specific address is likely to be accessed repeatedly in a short period of time because of register-to-memory or memory-to-memory instruction set architectures and limited register sets. As superscalar architectures advance, an aggressive scheduling policy for memory access becomes crucial. In this paper, we examine the scheduling policies of loads/stores on CISC superscalar processors and develop an aggressive scheduling policy called *preload*. The preload scheduling policy allows loads to precede the earlier unsolved pending stores, and delays the checking of conflict and forwarding of data until the data is loaded, thereby allowing greater tolerance of the latency for address generation. Because of its popularity, we focus our attention on the x86 instruction set. Simulation results show that the preload achieves a higher performance in comparison with the traditional scheduling policies such as load bypassing and load forwarding. Furthermore, by reducing the pipeline stages, the preload can achieve even higher performance.

Keywords: CISC, superscalar, memory access ordering, x86 microprocessor, load bypassing, load forwarding

1. INTRODUCTION

For CISC (complex instruction-set computing) microprocessors, such as the x86 [1-3] and VAX [4], the proportion of memory access instructions is relatively high, and a specific address is likely to be accessed repeatedly in a short period of time because of register-to-memory or memory-to-memory instruction set architectures and limited register sets. For those CISC processors that use modern superscalar techniques to achieve higher performance by dynamic scheduling and out-of-order execution of multiple instructions in parallel, exploiting the parallel execution ability of memory access operations becomes crucial.

In this paper, we examine the scheduling policies of loads/stores on CISC superscalar processors and propose a new aggressive scheduling policy suitable especially for CISC. We focus our attention on the x86 instruction set as an example due to its popularity. The x86 superscalar processors, such as Pentium, Pentium Pro, K5, K6, M1 and M2, have dominated the PC market over the past decade, and require more powerful

Received January 6, 2000; revised May 25, 2000; accepted September 26, 2000.
Communicated by Youn-Long Lin.
*This paper presents partial result of a long-term research project financed by both the NSC of R.O.C. under contract no. NSC 86-2622-E-009-009 and the industry.

load/store techniques to achieve higher issue rates for following generations of microprocessors.

For consistency of memory, stores are executed in the original program order. However, loads can be executed speculatively without obeying the original program order. Because of the looser restriction of loads, more parallel execution can be exploited by out-of-order execution of loads, and, thus, forms the scheduling policies of memory accesses such as *load bypassing* and *load forwarding* [5]. 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 the data cache. Otherwise, the load must be held until the conflicted stores are issued. In load forwarding, the data of the conflicted store can be forwarded directly to the pending load. Load bypassing has been used by this generation of x86 microprocessors such as Pentium Pro, K5, and K6.

However, in load bypassing and load forwarding, a load cannot be issued or forwarded if any one of the addresses of its previous stores is unsolved, i.e., has not been generated [6]. In the CISC superscalar microprocessors, an additional pipeline stage must be attached to generate the addresses for loads and stores. The lengthened pipeline will cause the unsolved problem to be even more severe. In this paper, we develop an aggressive scheduling policy called *preload*. In preload, loads can be issued without doing an address conflict check, and therefore, they can precede the previous unsolved pending stores. After the data is read from the cache, these loads will perform the address conflict check and forward data if their addresses violate those of the previous stores. Under the preload policy, we can reduce the scheduling stage of loads in CISC processors and thus achieve higher performance gain.

There are other aggressive techniques, such as address prediction [7-11] and data prediction [12, 13], which are developed to resolve similar problems. However, all these techniques suffer from prediction errors and the cost of recovery mechanisms. Therefore, we focus our research on scheduling policies that need no recovery mechanisms.

To examine the scheduling policies of loads/stores on x86 superscalar microprocessors, we design a load/store unit with a unified memory access buffer (UMAB), and analyze the policies, including load bypassing, forwarding, and preload, in a simulation environment. The simulation results show that the preload achieves higher performance in comparison with the other two traditional scheduling policies.

The organization of this paper is as follows. In the next section, we survey the load/store units of current CISC microprocessors. Then, we propose our design of load/store units with different policies in Section 3. In Section 4, we examine the scheduling policies and other design issues by experiments. Finally, we make conclusions in Section 5.

2. LOAD/STORE UNITS OF CURRENT CSIC MICROPROCESSORS

We choose Intel P6 and AMD K5 to represent current CISC microprocessors and survey their load/store units. We also compare them with the Power PC 604 [14], a processor representing current RISC microprocessors. Features of these three microprocessors are summarized in Table 1. "N/A" means not available.

Table 1. Load/store processing of K5, P6 and PowerPC 604.

Microprocessor	AMD K5	Intel P6	PowerPC 604
Reservation Station	Distributed 4-entry	Centralized 20-entry	Distributed 2-entry
Out-of-order issue	×	✓	×
Number of issues	2 loads or 1 load + 1 store	1 load + 1 store	1 load or 1 store
LSU	2 load/store units	1 load address unit 1 store address unit 1 store data unit	1 load/store unit
Buffer / Size	Store buffer/4-entry	Memory reorder buffer / N/A	Load queue/4-entry Store queue/6-entry
Data Cache	8K bytes 4-way set associative Dual-ported Virtually tagged	8K bytes 4-way set associative Dual-ported Physically tagged Nonblocking	16K bytes 4-way set associative S ingle-ported Physically tagged Nonblocking
Load bypassing	✓	✓	✓
Load forwarding	N/A	N/A	✓

Due to the significant amount of memory access operation, x86 microprocessors provide more load/store ports than RISC microprocessors. For example, the AMD K5 and Intel P6 can issue one load and one store per cycle, while the PowerPC 604 can issue only one load or one store per cycle.

All three microprocessors provide load bypassing. The PowerPC 604 additionally provides load forwarding. However, we cannot find any relevant information in the literature to tell whether or not the AMD K5 and Intel P6 provide load forwarding.

3. SCHEDULING POLICY OF X86 SUPERSCALAR MICROPROCESSORS

In this section, we propose our designs for the load/store units of x86 superscalar microprocessors with different scheduling policies. First, we develop an aggressive scheduling policy – preload. Then, we model the x86 superscalar microprocessor and its load/store unit. Finally, we point out that the number of pipeline stages can be reduced under preload.

3.1 An Aggressive Scheduling Policy – Preload

In preload, loads can be issued without an address conflict check. After the data is loaded back from the cache, these loads will check the addresses with their previous stores. If any conflict is found out, the data of the conflicted stores may be forwarded directly to the pending loads. If not the entire data of the pending load can be obtained from such forwarding, and the load will be re-issued after all its previous stores are com-

pleted. Some data cache accesses may be wasted when conflicts are found. We will describe the detailed actions of preload in Section 3.3.

A load can check its address with that of the previous stores only after all the addresses of these stores have been calculated. If loads can be issued only after the address conflict check, many opportunities for parallel execution may be lost because of the stalling of the loads due to an unfinished linear address calculation. Preload delays the address conflict check after the data cache access, and, thus, loads can be executed without being stalled by unsolved stores.

3.2 A Model of X86 Superscalar Microprocessors

To examine the scheduling policies of loads/stores, we first design a general model of x86 superscalar microprocessors. The pipeline stages are depicted in Fig. 1, and the block diagram of our simulator is depicted in Fig. 2. This model shows the characteristics of x86 processors that need additional stages for translating instructions, dispatching operations, and calculating addresses.

Fig. 1. Pipeline stages of x86 superscalar microprocessors.

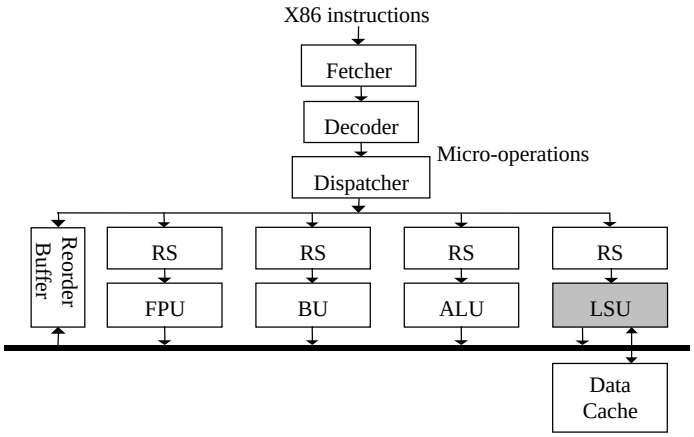


Fig. 2. Block diagram of x86 superscalar microprocessors.

The pipeline is divided into 6 stages: fetch, decode, dispatch, reservation station, execution, and retire. X86 instructions are fetched and decoded into simple, fixed-length micro-operations. For example, memory-to-register or register-to-memory addressing modes are decoded into simple load, store and other related micro-operations. Distributed reservation stations are used in our discussion. Micro-operations are dispatched to different reservation stations in order. The micro-operations with ready operands and available functional units are issued out of order from the reservation stations

to the execution units. The execution latencies are different among different kinds of micro-operations. The micro-operations are completed out-of-order while the in-order state of the program is kept in the reorder buffer. The results of micro-operations are retired in order by the reorder buffer.

3.3 Models of Load/store Units With Different Scheduling Policies

Under the architecture of x86 superscalar microprocessors, we design four models of load/store units: in-order, bypassing, forwarding, and preload. We unify the pipeline stages and the block diagrams of the four models as shown in Fig. 3 and 4, respectively. However, the functions of each stage may be different between the four models.

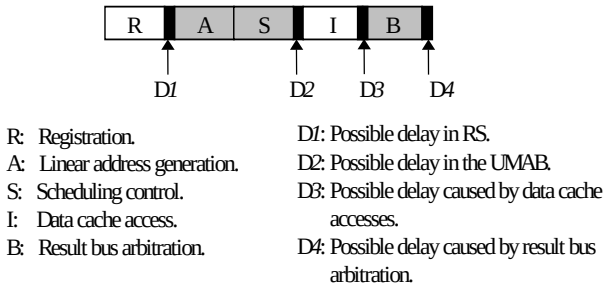


Fig. 3. Pipeline stages of the load/store unit.

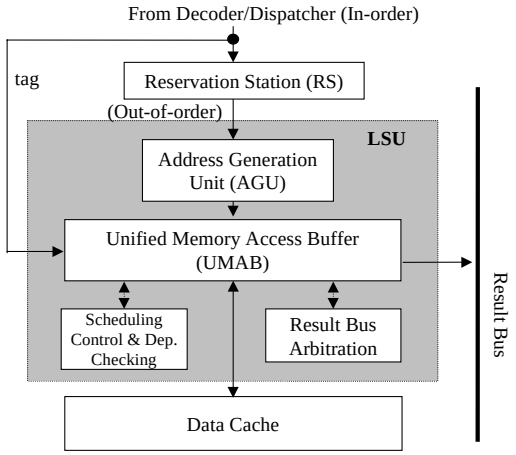


Fig. 4. Block diagram of the load/store unit.

So as to keep the original program order of load/store operations, we have to save the loads/stores that have not yet been retired in buffers. Buffering of loads/stores may be separated or unified. Separated buffering has two buffers, one for loads and the other for stores. Unified buffering, on the other hand, has only one buffer for both loads and stores. The major complexity in separated buffering is the cooperation between the load and store buffers in order to maintain the relations of true data dependencies be-

tween each other. In our design, we choose to maintain a unified buffer called the Unified Memory Access Buffer (UMAB) for loads/stores.

In Fig. 3, the gray blocks (A-stage, S-stage, and B-stage) are the operation latencies caused by the load/store unit. The black sticks (from D*1* to D*4*) denote the execution decoupling, which allows the out-of-order execution of the ready operations. The loads/stores are sequentially dispatched to the reservation station of the load/store unit in the R-stage. Simultaneously, to keep the program order of the loads/stores, the loads/stores are also registered into the UMAB. The loads/stores stored in the reservation station are issued to the load/store unit if all their operands are ready and the address generation unit of the load/store unit is available. The linear addresses of the newly issued loads/stores are then generated in the A-stage.

In the S-stage, the executable loads/stores are issued to the data cache or the data is forwarded to the result bus. There are different issue rules for the loads in the four models. The issue rules are summarized in Table 2. For stores, the issue rules are the same in all four models. A store is executable if it satisfies the following conditions:

1. Its linear address is generated.
2. It reaches to the head of the UMAB.
3. Its retirement permission is received from the reorder buffer.

If the number of the executable loads/stores is larger than the number of ports provided by the data cache, the older loads/stores have higher priorities.

The I-stage is assigned for data cache accesses. One-cycle latency is assumed. For a load, after the data is loaded back from the data cache, the B-stage is responsible for arbitrating the result bus for the data. With preloading, loads must be checked for address conflict, and data forwarding, bus arbitration, and re-issuing must be performed according to the checking result. The functions of the four models in each stage are summarized in Table 3.

Table 2. Issue rules of the loads in the four models.

	Issue rules of the loads
In-order	1. Linear addresses of the loads are generated. 2. There is no unsolved previous store. 3. There is no pending previous store.
Bypassing	1. Linear addresses of the loads are generated. 2. There is no unsolved previous store. 3. There is no true data dependency with the pending previous stores.
Forwarding	1. Linear addresses of the loads are generated. 2. There is no unsolved previous store. 3. There is no true data dependency with the pending previous stores, or there is a true data dependency with a store from which the whole data can be forwarded.
Preload	Linear addresses of the loads are generated.

Table 3. Execution stages in the load/store unit.

	R-stage	A-stage	S-stage	I-stage	B-stage
In-order	Registration	Address generation	In-order scheduling	Data cache access	Result bus arbitration
Bypassing	Registration	Address generation	• Dep. checking • Bypassing scheduling	Data cache access	Result bus arbitration
Forwarding	Registration	Address generation	• Dep. checking • Bypassing scheduling • Data forwarding	Data cache access	Result bus arbitration
Preload	Registration	Address generation	• Preload scheduling	Data cache access	• Dep. checking • Data forwarding • Result bus arbitration /Re-issue to the data cache

The detailed block diagram of an LSU with preload policy is shown in Fig. 5. The dependency check is done by the comparator associated with every UMAB entry. From the diagram, we see that data forwarding and result bus arbitration can be activated in parallel right after the dependency check, and, thus, would not lengthen the latency of the B-stage.

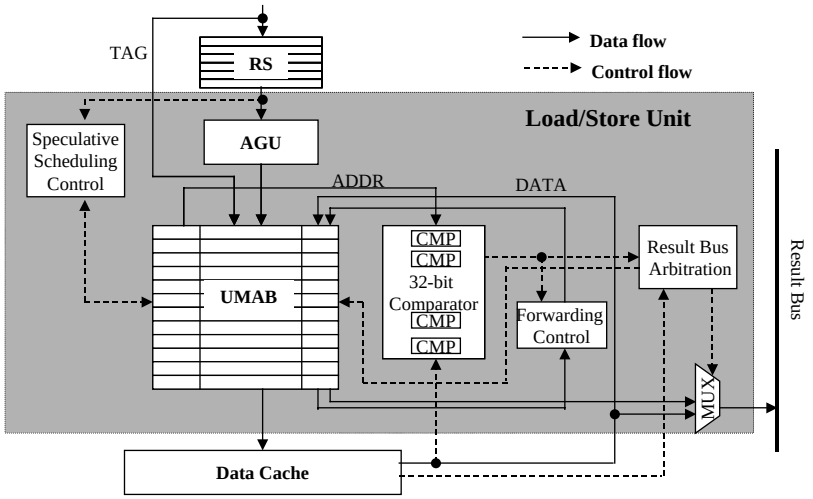
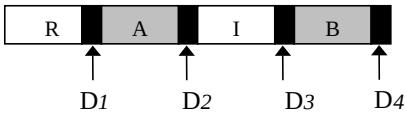


Fig. 5. Detailed block diagram of a load/store unit with preload policy.

3.4 Preload With Reduced Pipeline

With preload, the functions of the S-stage may be merged into that of the A-stage, and, thus, the S-stage can be eliminated. A load that has finished its address calculation can be issued to the data cache immediately. The pipeline stages and the block diagram of this new model are shown in Figs. 6 and 7, respectively. Compared to other models, the S-stage of the reduced pipeline is eliminated, and the scheduling control can work at the same time as that of the address calculation.



R: Registration.
A: Linear address generation and scheduling control.
I: Data cache issue.
B: Result bus arbitration.

D*1*: Possible delay in RS.
D*2*: Possible delay in UMAB.
D*3*: Possible delay caused by data cache misses.
D*4*: Possible delay caused by result bus arbitration.

Fig. 6. Stages of the reduced pipeline for the load/store unit with preload.

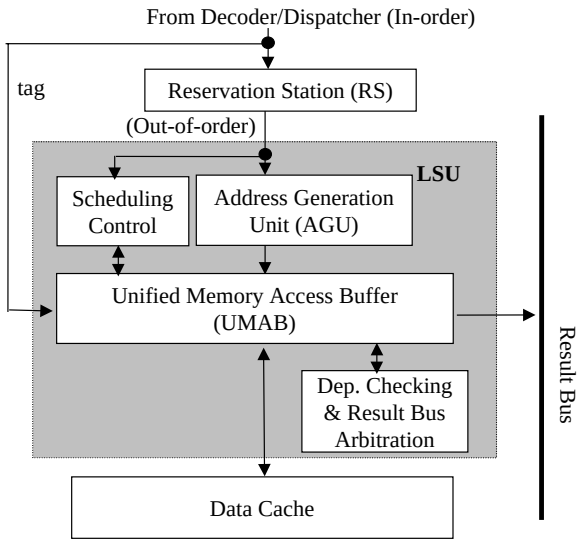


Fig. 7. Block diagram of the load/store unit with reduced pipeline.

4. PERFORMANCE EVALUATION AND ANALYSIS

In this section, we use simulations to examine scheduling policies. First, we present the simulation environment and the simulation models. Then, we examine the proportion of loads/stores in benchmarks to show their importance. Finally, we simulate different scheduling policies under different sets of parameters.

4.1 Simulation Environment

In order to examine the scheduling policies of loads/stores, we build a trace-driven simulator. The source code of SPEC95 is compiled with the gcc compiler. The executable benchmark along with the input files are executed under Linux OS on x86

compatible PCs. The traces are retrieved by the system call "ptrace()" and are read into our simulator to evaluate the performance of the simulation models we proposed in Section 3. The flow chart of the simulation environment is shown in Fig. 8.

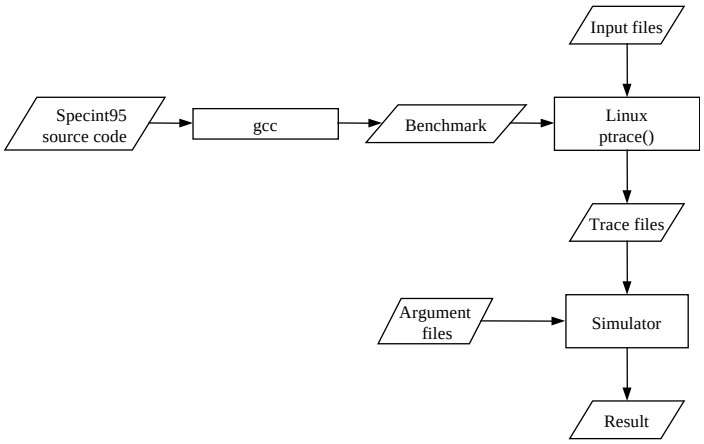


Fig. 8. Flow chart of the simulation environment.

The benchmark programs we simulated are of SPECint95. We briefly describe them in Table 4 [15].

Table 4. Short descriptions of the SPECint95 benchmarks.

Benchmark	Remarks
Go	Artificial intelligence; plays the game of "Go"
m88ksim	Motorola 88000 chip simulator; run test program
Gcc	New version of GCC; build SPARC code
Compress	Compress and decompress file in memory
Li	LISP interpreter
Ijpeg	Graphic compression and decompression
Perl	Manipulates strings and prime numbers in Perl
Vortex	A database program

4.2 Simulation Models

The simulator in our experiments simulates the models mentioned in Sections 3.2 and 3.3. To focus the discussion on the design issues of load/store units, we make the following assumptions.

1. The accuracy of branch prediction is 100%. When a branch is taken, its succeeding instructions if fetched at that cycle are discarded.
2. No data cache miss occurs, and the access latency is 1 cycle.
3. The latency of ALU and BU is 1 cycle.

To clear up the parameters of our simulator, we set three sets of restrictions for the simulator and summarize them in Table 5. Restriction 1 restricts the UMAB size and let other parameters be unlimited so that we can analyze the influence of the UMAB size and ignore the influences of other parameters. For similar reason, Restriction 2 limits the UMAB size and the number of LSU ports, and let other parameters be unlimited. In order to analyze the simulation results under the specific set of parameters, Restriction 3 limits all the parameters listed.

Table 5. Three sets of restrictions for the components in our simulator.

Restriction	Fetcher	Decoder	Dispatcher	Reservation Station	LSU ports	UMAB Size
1	Unlimited	Unlimited	Unlimited	Unlimited	Unlimited	Variable
2	Unlimited	Unlimited	Unlimited	Unlimited	Variable	32 entries
3	Unlimited	8 micro-operations	8 micro-operations	16 entries	3 ports	32 entries

The four basic models of the load/store unit mentioned in Section 3 (in-order, load bypassing, load forwarding, and preload) are simulated under these three sets of restrictions. The simulation results described in Sections 4.4, 4.6, and 4.7 are collected under Restrictions 1, 2, and 3, respectively.

4.3 The Proportion of Loads/stores in SPECint95 Benchmarks

Before evaluating the performances of different scheduling policies, we examine the proportion of loads/stores in the benchmarks of SPECint95 to show the importance of these operations in CISC superscalar microprocessors. In our model, the CISC x86 instructions are translated into simple, fixed-length micro-operations. As shown in Fig. 9, the proportions of loads/stores in most of the benchmarks approximate 50%. In comparison with RISC microprocessors, which comprise only about 25%~30% loads/stores, the design of the load/store unit is much more important in CISC microprocessors.

4.4 Performance Evaluations Under the Limitation of the UMAB Size

Under only the UMAB size restriction, i.e., Restriction 1 in Section 4.2, the performance variations between different UMAB sizes of in-order, bypassing, forwarding,

and preload models are depicted in Fig. 10. The performance of the in-order model with 4-entry UMAB is regarded as the base case for comparison in this paper. As shown in Fig. 10, the preload model has the best performance gain, while the in-order model is the worst.

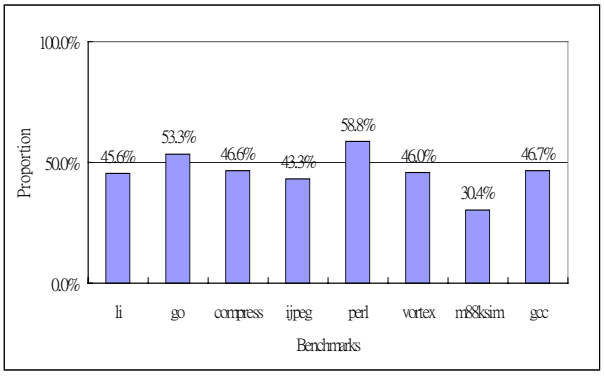


Fig. 9. Proportions of load/store micro-operations in the translated SPECint95.

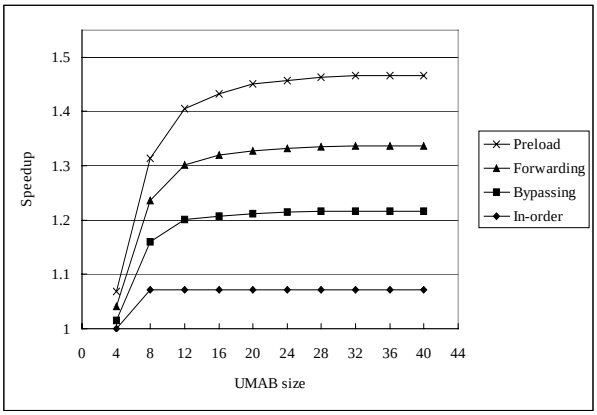


Fig. 10. Speedup for different scheduling policies under various UMAB sizes.

The in-order model has the worst performance because the loads are always stalled by the pending stores. The bypassing model has better performance because the loads can bypass those previous pending stores which have no true data dependencies with the loads. The forwarding model achieves higher performance because the loads may obtain the data forwarded from the nearest stores. Recall that, no matter in load bypassing or forwarding, a load cannot be issued or forwarded if any address of its previous stores is unsolved. Therefore, the preload model has the best performance because the loads can bypass all previous stores, pending or unsolved.

In the in-order, bypassing, forwarding, and preload models, the performance is saturated when the UMAB sizes reach 8, 12, 16, and 20 entries, respectively. The higher

performance a model can achieve, the larger UMAB of the model for saturated performance is required. This means that a more aggressive scheduling policy will use more UMAB entries effectively.

The data cache traffic will increase a little in the preload model. The average forwarding rates in the forwarding model and the preload model are both 5.4%. However, the preload model has loaded the data from cache before forwarding, thus increasing the traffic. Compared with the in-order and the bypassing models, re-issue loads increase the traffic. As shown in Fig. 11, the re-issuing rates are always below 1%. The low traffic increment along with high performance gain makes the preload model practical.

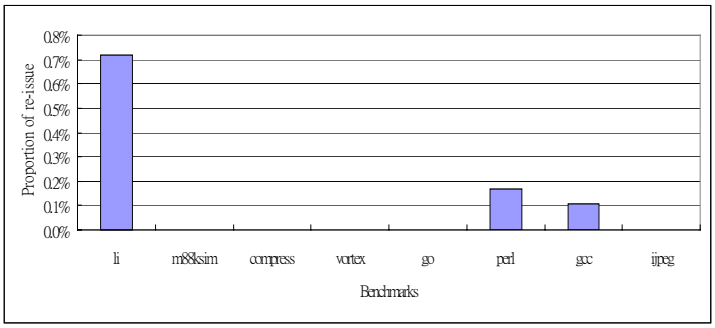


Fig. 11. Re-issued rate of each benchmark in the preload model.

4.5 Performance Evaluation of the Speculative Load/store Unit with Reduced Pipeline

As mentioned in Section 3.4, the S-stage in the pipeline of the preload model can be removed. The performance improvement by reducing the pipeline is shown in Fig. 12. It is obvious that the reduction of just one pipeline stage can improve the performance significantly.

4.6 Performance Evaluations Considering the Limitation of the Number of Ports of the Load/store Unit

In this subsection, we simulate the scheduling policies under Restriction 2 mentioned in Section 4.2 in order to estimate the effect of different numbers of load/store unit ports. According to Restriction 2, the UMAB size is set to 32, the number of LSU ports is variable, and the other parameters are not limited. The simulation results are depicted in Fig. 13. It is obvious that the performance improvement from one port to two is significant. The performance rises gradually from two ports to three. However, there is only a small change in performance for more than three ports. Therefore, we suggest that providing the load/store unit with three ports is best for performance consideration.

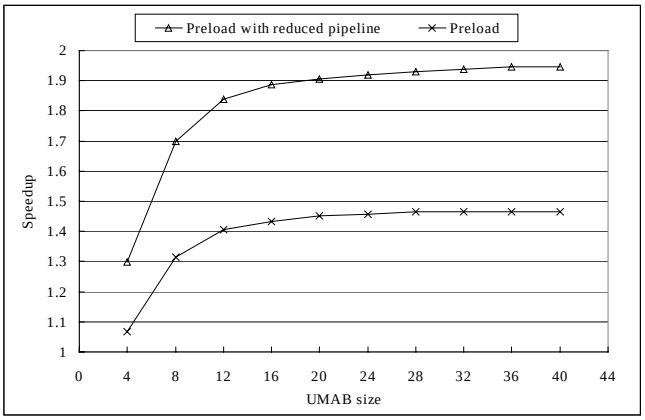


Fig. 12. Speedup improvement of the preload by reducing pipeline.

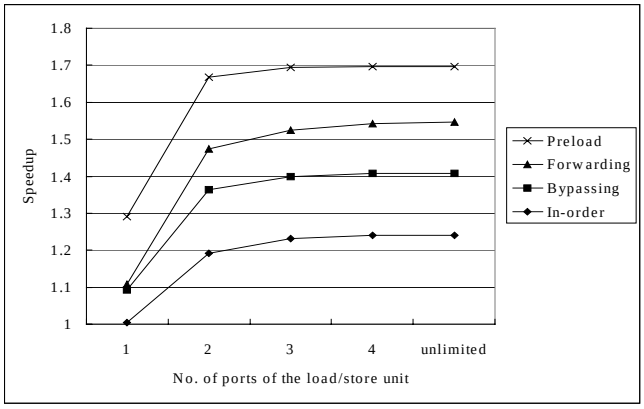


Fig. 13. Speedup of scheduling polices under various numbers of LSU ports.

4.7 Performance Evaluations of the Load/store Unit Under the Limited Front End

In the previous sections, we evaluated the performance of the load/store unit under the unlimited front end, which provides unlimited decoder bandwidth, dispatching bandwidth, and reservation station buffer size. In this section, we restrict the decoder bandwidth to at most eight micro-operations. We first examine the size of the reservation stations and the numbers of ALUs and branch units (BUs). The base case is the performance under 4-entry reservation stations, one ALU, one BU, and one load/store unit (LSU). As shown in Fig. 14, there is almost no performance improvement when the number of the ALU increases from two to three. In the case of three ALUs, there is little performance improvement when the size of the reservation station is larger than 16 entries.

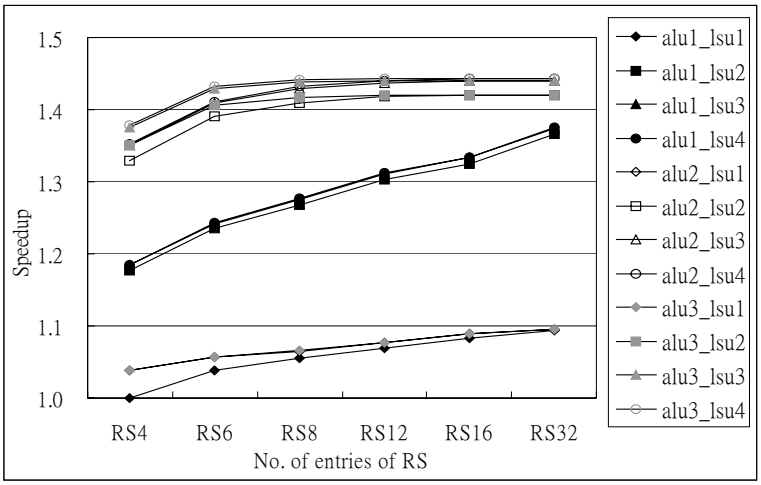


Fig. 14. Performance variations between different sizes of RS and numbers of ALUs and LSUs.

We then restrict the size of the reservation stations to 16, the number of ALU to three, the number of BU to one, and the number of load/store units to three. The base case of the following simulation is the performance of in-order model with 4-entry UMAB and unlimited front end. The performance differences between the limited and unlimited front end are shown Fig. 15.

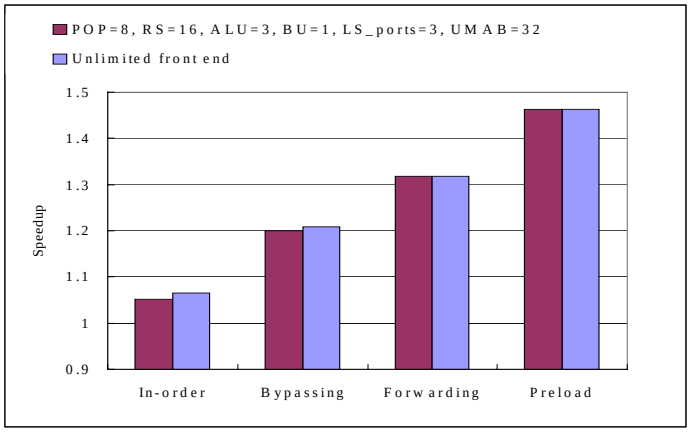


Fig. 15. Comparison between unlimited and limited front end.

As shown in Fig. 15, the preload model still has the best performance under the limited front end. Additionally, there is only a sprinkling of performance degradation between the limited and unlimited front ends in the forwarding and preload models. As a result, under the limited front end, the capability of the scheduling policies is important. A load/store unit with load bypassing and load forwarding is necessary, and a load/store unit with preload is strongly recommended.

5. CONCLUSIONS

In this paper, we develop the preload scheduling policy, design the models of the load/store units with different scheduling policies, and examine these scheduling policies. We simulate four models, which are in-order, load bypassing, load forwarding, and preload. Furthermore, we realize that the preload can reduce the pipeline stage for scheduling to decrease the load latency.

Almost half of the micro-operations in the translated x86 instructions are loads/stores. Therefore, the capability of load/store units is very important in x86 microprocessors. In comparison to other scheduling policies, the simulation results show that the preload achieves the best performance gain. The preload model enlarges the execution window of load/store instructions the most and gains the most profit as the window size grows. In addition, the pipeline stages of the preload model can be further reduced to gain more performance improvement.

As our simulation results show, the load/store unit with preload can gain as much performance improvement under the limited front-end superscalar microprocessor as that of infinite-way superscalar microprocessors. Hence, in the next generation of the CISC processors, memory access is still one of the most important design issues as the superscalar degree increases.

The CISC instruction sets have good application compatibility and code density. Due to the recent advances in silicon technology, sophisticated load/store units can be implemented to handle the complex addressing modes and the heavy memory traffic. As the issue rates of superscalar microprocessors are increasing, careful design of the load/store units becomes more important.

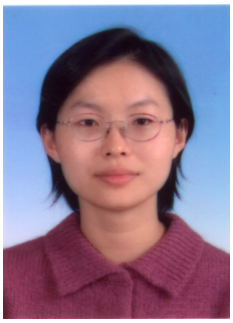
In the future, we plan to study software scheduling of loads/stores for help in improving the efficiency of hardware scheduling. Furthermore, the scheduling of loads/stores in this paper does not allow the stores to be executed speculatively. Relaxing the restriction on in-order executed stores may exploit greater parallel execution of loads/stores. Of course, some mechanism should be provided to maintain the consistency of system memory. We are also interested in the performance evaluation and recovery mechanism of speculative stores.

REFERENCES

1. L. Gwennap, “Intel’s P6 uses decoupled superscalar designing,” *Microprocessor Report*, Vol. 9, No. 2, 1995, pp. 5-16.
2. AMD Inc, *AMD 5K86 Processor Technical Reference Manual*, 1996.
3. M. Slater, “AMD’s K5 designed to outrun Pentium,” *Microprocessor Report*, Vol. 8, No. 14, 1994, pp. 1-11.
4. J. E. Wilson, S. W. Melvin, M. C. Shebanow, W. M. Hwu, and Y. N. Patt, “On tuning the microarchitecture of an HPS implementation of the VAX,” in *Proceedings of the 20th Annual International Symposium on Microarchitecture*, 1987, pp. 162-167.
5. M. Johnson, *Superscalar Microprocessor Design*, Prentice Hall, 1991.
6. M. Franklin and G. S. Sohi, “ARB: a hardware mechanism for dynamic reordering of memory references,” *IEEE Transactions on Computers*, Vol. 45, No. 5, 1996, pp. 552-571.

7. R. J. Eickemeyer and S. Vassiliadis, "A load-instruction unit for pipelined processors," *IBM Journal of Research and Developing*, Vol. 37, No. 4, 1993, pp. 547-564.
8. T. M. Austin and G. S. Sohi, "Zero-cycle loads: microarchitecture support for reducing load latency," in *Proceedings of the 28th International Symposium on Microarchitecture*, 1995, pp. 208-218.
9. Y. Sazeides, S. Vassiiadis, and J. E. Smith, "The performance potential of data dependence speculation and collapsing," in *Proceedings of the 29th International Symposium on Microarchitecture*, 1996, pp. 238-247.
10. A. Moshovos, S. E. Breach, T. N. Vijaykumar, and G. S. Sohi, "Dynamic speculation and synchronization of data dependences," in *Proceedings of the 24th Annual International Symposium on Computer Architecture*, 1997, pp. 181-193.
11. G. Z. Chrysos and J. S. Emer, "Memory dependence prediction using store sets," in *Proceedings of the 25th Annual International Symposium on Computer Architecture*, 1998, pp. 142-153.
12. M. H. Lipasti, C. B. Wilkerson, and J. P. Shen, "Value locality and data speculation," in *Proceedings of the 7th International Conference on Architecture Support for Programming Languages and Operating Systems*, 1996, pp.138-147.
13. Y. Sazeides and J. E. Smith, "The predictability of data values," in *Proceedings of the 30th International Symposium on Microarchitecture*, 1997, pp. 248-258.
14. Motorola Inc., *PowerPC 604 RISC Microprocessor User's Manual*, 1994.
15. B. Case, "SPEC95 retires SPEC92," *Microprocessor Report*, 1995, pp. 11-14.

R-Ming Shiu (徐日明) received the B.S. degree from Fu Jen Catholic University, Taipei, Taiwan, R.O.C, in 1993, and the Ph.D. degree from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 2000, all in Computer Science and Information Engineering. He is currently performing military service. His research interests include computer architecture and parallel processing.



Hui-Yue Hwang (黃慧瑜) received her B.S. and M.S. degrees in Computer Science and Information Engineering from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1996 and 1998, respectively. After graduation, she joined the PowerVision Technologies, INC., in Science-Based Industrial Park, Hsinchu, and is now a senior engineer of Logic Product Department. Her research interests include queuing theorem and scheduling mechanisms.

Jean Jyh-Jiun Shann (單智君) received the B.S. degree in Electronic Engineering from Feng-Chia University, Taichung, Taiwan, Republic of China in 1981. She attended the University of Texas at Austin from 1982 to 1985, where she received the M.S.E. degree in Electrical and Computer Engineering in 1984. She was a lecturer in Department of Computer Science and Information Engineering, National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., while working towards the Ph.D. degree. She received the degree in 1994 and is currently an Associate Professor in the department. Her research interests include computer architecture, parallel processing, and information retrieval.