行政院國家科學委員會補助專題研究計畫成果報告

※※※※※※※※※※※※※※※※※※※※※※※※※
※　　　　　　　　　　　　　　　　　　　　　　　※
※　X86 超純量處理器之預測性記憶體存取單元設計　※
※　　　　　　　　　　　　　　　　　　　　　　　※
※※※※※※※※※※※※※※※※※※※※※※※※※

計畫類別：[✓ 個別型計畫]　　□整合型計畫
計畫編號：NSC89－2213－E－009－205
執行期間：89 年 8 月 1 日至 90 年 7 月 31 日

計畫主持人：單智君　博士
共同主持人：鍾崇斌　博士

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
□出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

執行單位：國立交通大學資訊工程學系

中　華　民　國　89　年　10　月　31　日

X86 超純量處理器之預測性記憶體存取單元設計

Design a speculative memory access unit of x86 superscalar processor

計畫編號：NSC-89-2213-E-009-205

執行期限：89 年 8 月 1 日至 90 年 7 月 31 日

主持人：單智君 博士

共同主持人：鍾崇斌 博士

計畫參與人員：徐日明、邱日清、蔣昆成、劉嘉修

一、中文摘要

x86 指令集有複雜的記憶體定址模式，為了配合高時脈的要求，必須克服其冗長的位址轉換計算。除此之外，其記憶體存取的頻率很高，特別需要開發高效能的記憶體存取技術，來提供足夠的記憶體存取頻寬，以配合多重執行單元的運算能力。本計畫研究各種記憶體存取位址與相依性的預測策略，來克服 x86 指令集冗長的位址計算等待延遲，設計出適合下一代高派發率 x86 超純量處理器的預測性記憶體存取單元。並提出一個測試原型，以驗證設計的效率與可行性。

我們提出的預測性記憶體存取，主要是將不需等待運算元的預測機制與傳統的有效位址計算機制並行處理。為了增加預測正確率，我們發展新的位址及相依性預測。加上回送預測能力、2 位元計數器細緻化預測、及以 2 位元計數器過濾預測等方法，加強相依性預測。為了減小預測錯誤處罰，我們考慮預測的時機與讀出資料的送回策略等問題。實驗結果顯示，透過增加預測正確率及減小預測錯誤處罰，我們提出的預測性排程可以有效增加效能。

關鍵詞：x86 **指令集、超純量架構、記憶體資料存取、預測執行、位址預測、相依性預測**

Abstract

X86 instruction set has complex memory addressing modes and address calculations, and thus become difficult to achieve high clock rate. Moreover, with high memory access frequency, x86 superscalar processors especially need parallel memory access techniques to support enough data bandwidth. In this project, we design the address and dependence prediction strategies for memory accessing of high-issue-rate x86 superscalar processors, and build a prototype of our speculative memory access unit to evaluate the performance and feasibility of the design.

Our speculative memory access strategies handle the address and dependency prediction mechanism in parallel with the traditional address calculation mechanism. To increasing the prediction accuracy, we develop new address and dependency prediction policies. We improve the dependency prediction by adding forwarding prediction ability, refining the predictions with 2-bit counter, and filtering out the error-like predictions with another 2-bit counter. To reduce the miss-penalty, we consider the prediction stage and the strategies for handling loaded data. Experiment results show that, by reducing the miss-penalty and increasing the prediction accuracy, the predictive scheduling proposed in this work can significantly improve the performance.

Keywords：x86 instruction set, superscalar, memory access, speculative execution, address prediction, dependency prediction

二、緣由與目的

For x86 program, the proportion of memory access instructions is relatively high and a specific address is likely to be accessed repeatedly in a short period because of their register-to-memory or memory-to-memory instruction set architectures and limited register sets. For the consistency of memory, stores are executed in the original program order. However, loads 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 [1]. However, in these conservative scheduling policies, a load cannot be issued or forwarded if any addresses of its previous stores is unsolved, i.e. has not been generated. This problem becomes much severer in an x86 superscalar microprocessor because the pipeline is lengthen for address calculation.

Many prediction techniques, such as address prediction, dependency prediction, and value prediction, have been developed on RISC for resolving the unsolved address problem [2][3]. These techniques predict the addresses, dependencies, or even data of loads 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 cannot work effectively. Therefore, we enhance these prediction techniques by increasing the prediction accuracy and reducing the miss-penalty.

To increasing the prediction accuracy, we develop new address and dependency prediction policies. We choose the 2-stride scheme in [3] as our address prediction, and choose the store-load pair scheme in [5] to develop our dependency prediction. To reduce the miss-penalty, we consider the prediction stage and the strategies for handling loaded data.

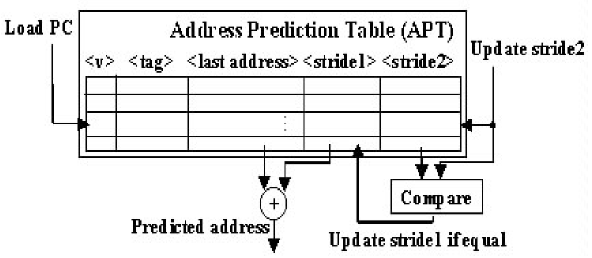
三、結果與討論

3.1 Prediction Policies of Loads

We develop the prediction policies, including address prediction, pre-load, and three dependence/forwarding predictions, to increase the prediction accuracy.

3.1.1 Address Prediction

We choose the 2-stride scheme in [4] as our address prediction (AP) because its outstanding accuracy and reasonable implement cost. AP predicts the data addresses of loads using a 2-stride address predictor shown in Figure 1, whose address prediction table (APT) stores the information generated by previous loads as a set-associative cache.


Load PC
Address Prediction Table (APT)
<v> <tag> <last address> <stride1> <stride2>
Update stride2
+
Compare
Predicted address
Update stride1 if equal
Figure 1. Block diagram of 2-stride address predictor

3.1.2 Pre-Load

Pre-load (PL) is the simplest dependency prediction policy that predicts every load as non-dependent. In PL, loads can be issued to the data cache without the address conflict check once its data address is calculated or predicted by AP. PL delays the address conflict checking after the data cache access, and thus loads can be executed without being stalled by unsolved stores.

3.1.3 Dependency/Forwarding Prediction

We choose the store-load pair scheme in [5] as the base to develop our dependency prediction for its accuracy and reasonable implement cost. By adding the forwarding prediction scheme to predict if the value can be forwarded from the unified memory access buffer (UMAB), the dependency prediction becomes the dependency/forwarding prediction (DP). DP predicts the dependency using a store-load pair predictor shown in Figure 2, whose dependency/forwarding prediction table (DPT) stores the information generated by previous loads as a set-associative cache. DPT is indexed by the PC of the encountered load.

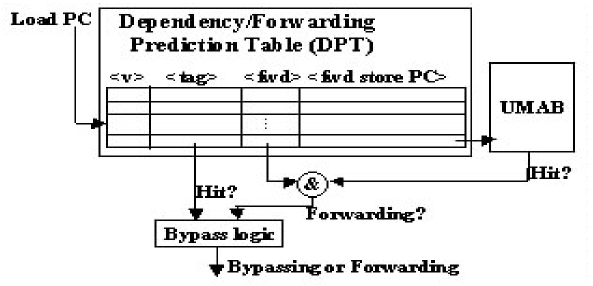

Load PC
Dependency/Forwarding Prediction Table (DPT)
<v> <tag> <fwd> <fwd store PC>
UMAB
Hit?
&
Hit?
Forwarding?
Bypass logic
Bypassing or Forwarding

Figure 2 The block diagram of store-load pair predictor.

3.1.4 Counter-based Dependency /Forwarding Prediction

To improve the accuracy of DP, we refine DP to become the counter-based dependency/forwarding prediction (CDP). CDP predicts the dependency using a counter-based store-load pair predictor shown in Figure 3 whose counter-based DPT (CDPT) is modified from DPT by adding a classify counter field. The classify counter field is a 2-bit saturation counter to keep the tendency of independence of a load.

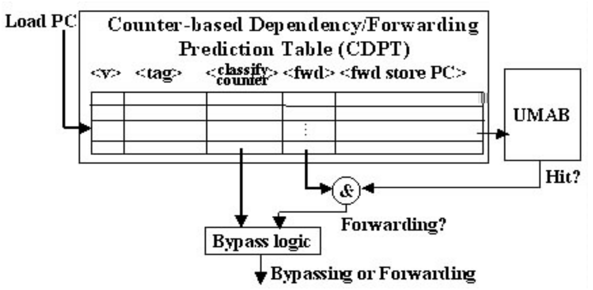

Load PC
Counter-based Dependency/Forwarding Prediction Table (CDPT)
<v> <tag> <classify counter> <fwd> <fwd store PC>
UMAB
&
Hit?
Forwarding?
Bypass logic
Bypassing or Forwarding

Figure 3. The block diagram of counter-based store-load pair predictor.

3.1.5 Selective Dependency/Forwarding Prediction

To further improve the accuracy of CDP, we refine CDP to become the *selective dependency/forwarding prediction* (SDP). The empirical observations of [4] notify that relatively few loads cause most of the miss-predictions, and filter out these loads will increase prediction accuracy. SDP predicts the dependency using a counter-based store-load pair predictor with filter shown in Figure 4 whose selective DPT (SDPT) is modified from CDPT by adding a *filter counter* field. The *filter counter* is a 2-bit saturation counter to keep the tendency of miss prediction.

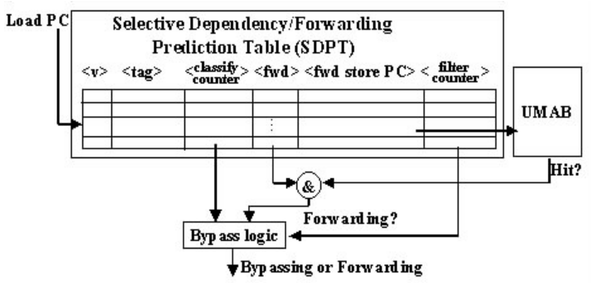

Load PC
Selective Dependency/Forwarding Prediction Table (SDPT)
<v> <tag> <classify counter> <fwd> <fwd store PC> <filter counter>
UMAB
Hit?
&
Forwarding?
Bypass logic
Bypassing or Forwarding

Figure 4. Block diagram of the counter-based store-load pair predictor with filter.

3.2 Predictive Models of Loads

To reduce the miss-penalty of prediction, we consider different prediction stage and send –back strategies.

3.2.1 Prediction at Different Stages

Traditionally, the predictions are made at fetch cycle as show in Figure 5(a) because it is the first cycle the PC of a load can be obtained. However, in the lengthen pipeline of x86, the miss-penalty is large enough to cancel out the advantage from prediction early. A delayed prediction as show in Figure 5(b) is developed by delaying the prediction until an instruction has been decoded and dispatched to reduce the miss-penalty. When predicting at front-end, all the predicted loads must stores in prediction validation buffer (PVB). By delaying the prediction after instruction dispatch, these predicted loads be stored in UMAB thus saving the hardware cost. The delayed prediction also let the prediction work only on loads and the prediction information can be validated in time thus slightly increases the prediction accuracy.

3.2.2 Send-back Strategies of Loaded Data

In general case, data loaded by a predicted load may be used immediately without verification by the succeeding operations. Then, once a miss prediction is detected, the miss-predicted load and all its succeeding operations must be recovered. We call this an aggressive send-back strategy of loaded data (ASB). However, we found that no verification of loaded data is too aggressive for some aggressive prediction policies. Thus, we develop a conservative send-back strategy (CSB) to check the accuracy of load prediction before the loaded data is used in order to eliminate the recovery penalty.

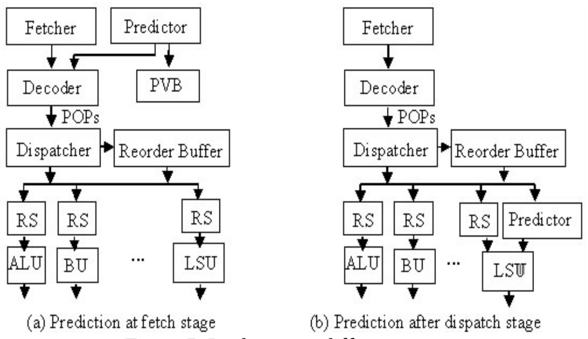


Figure 5. Prediction at different stages.

3.3 Performance Analysis

The prediction policies PL, DP, CDP, and SDP can all combine with AP and become new policies PL_AP, DP_AP, CDP_AP, and SDP_AP. The prediction stages, front end (F) or delayed (D), and send-back strategies, aggressive (A) or conservative (C), can be combined to form four predictive models: F_A, D_A, F_C, and D_C. The average speedups of the prediction policies and the predictive models over the load forwarding policy are shown in Figure 6. The average speedups are the harmonic means of the speedups of the eight SPECint95 benchmarks. The prediction policies are distinguished by the predictive model. The APT, DPT, and SDPT are all 4K-entry and 4-way associative, which are chosen for performance saturation.

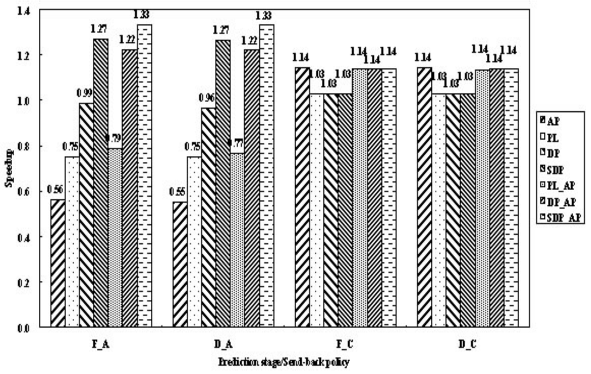


Figure 6. Performance comparison of various prediction policies and predictive models.

四、計畫成果自評

We have examined the address and the dependency prediction policies, prediction stage, and send-back strategies for memory accesses in x86 superscalar processors. The traditional prediction techniques developed on RISC cannot work effectively on x86 because of the lengthen penalty of prediction misses. However, combine the address and the dependency prediction can achieve good performance. Furthermore, we develop CDP and SDP to increase the prediction accuracy, and develop the delayed prediction and CSB to reduce the miss-penalty. The delayed prediction would not decrease the performance but can reduce hardware cost. CSB eliminates the penalties of miss-predicted loads and the recovery mechanism; thus let AP become a cost-effective selection. While a carefully designed SDP with ASB can achieve the highest performance. Simulation results show that SDP_AP can achieve 1.33 speedup over the traditional load-forwarding policy under commercial programs and next generation designs.

五、參考文獻

[1] M. Johnson, Superscalar Microprocessor Design, Prentice Hall, 1991.

[2] R. J. Eickemeyer and S. Vassiliadis: ‘A load instruction unit for pipeline processors,’ *IBM Journal of Research and Development*, vol. 37, 1993, pp.547-564.

[3] G. Z. Chrysos and J. S. Emer, "Memory Dependency Prediction using Store Sets," ISCA-25, 1998.

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

[5] 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.