行政院國家科學委員會補助專題研究計畫成果報告

※※※※※※※※※※※※※※※※※※※※※※※※※
※　　　　　　　　　　　　　　　　　　　　　　　　※
※　　　具資料流引擎之 x86 微處理機設計　　　　※
※　　　　　　　　　　　　　　　　　　　　　　　　※
※※※※※※※※※※※※※※※※※※※※※※※※※

計畫類別：☑**個別型計畫**　　□**整合型計畫**
計畫編號：NSC89－2213－E－009－066
執行期間： 88 **年** 8 **月** 1 **日至** 89 **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**　25 **日**

具資料流引擎之 x86 微處理機設計

Design of an x86 microprocessor with data-flow engine

計畫編號：NSC89-2213-E-009-066
執行期限：88 年 8 月 1 日~89 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：邱日清、徐日明、陳志龍、林育得
E-MAIL ：jjshann@csie.nctu.edu.tw

一、中文摘要

本計畫的目標在於完成具資料流引擎之 x86 微處理器（DF86）結構設計相關問題的研究與探討。X86 指令集的格式、語意及定址模式都非常複雜。X86 微處理器發展至目前的超純量架構，以種種特殊的設計來解決這些問題；但是指令擷取和資料相依性檢查的頻寬，還是無法配合上指令自然的平行度和多重執行單元的運算能力，而成為效率的瓶頸。依指令自然的平行度執行是資料流計算機結構的特性，本計畫將資料流（data-flow）架構引入 x86 微處理機中的構想，稱之為 DF86 結構。將資料流圖建立、儲存於微處理器內，以重複使用來解決這些瓶頸。整個架構的設計重點包括前端解碼、資料流引擎、及資料存取等三部份。其中前端解碼部份的研究主題包括：建立 data-flow graph 的方法、control flow 與 data flow 的切換、及將控制相依性轉為資料相依性的方法。資料流引擎部份的研究主題包括：data-flow graph 的暫存方法與空間管理、data-flow graph 的使用與維護。至於資料存取部份則在提出適合資料流特色的高頻寬資料存取機構。

DF86 結構是結合 Control flow 與 Data flow 的計算方法。將可使得需依靠高指令擷取度來提高效率的超純量方式，因加入內存資料流圖的計算結構，而降低指令擷取寬度，並達成指令自然的平行度執行，使得計算效能獲得充分提昇。

我們針對此機制中的參數進行模擬並選定合理值，然後用此設定值跟其他微處理器的派發率作比較。模擬結果顯示我們的機制的效能的確比其他微處理器高。在硬體花費/效能的考量下, 我們的機制平均每個週期可以派發二・０七個 X86 指令。

關鍵詞：x86 指令集、微處理機、資料流架構、超純量架構、資料流圖、高頻寬資料存取

Abstract

The objective of this project is to study and design an x86 microprocessor with data-flow engine – DF86. We will design the micro-architecture of the microprocessor using a data-flow kernel, and verify the design by simulation and emulation. The instruction formats, semantics and addressing modes of the x86 instruction set are extremely complex. When the x86 microprocessors progress to become superscalar, many complex designs are added to resolve these bottlenecks. However, the bandwidths of instruction fetch and dependence check are still too small to exploit the inherent parallelism of programs and to match high execution rate of multiple execution units. In this project, we solve these bottlenecks by building a data-flow engine, saving the data-flow graph in the processor, and reuse the graph. The micro-architecture can be partitioned to three parts: front-end decoder, data-flow engine, and data access. The research topics include: building, storing and maintaining the data-flow graph, switching between control flow and data flow, transferring control dependence to become data dependence and providing high-bandwidth data access.

We do the simulation to decide the parameters of our mechanism and compare the issue rate of our mechanism to that of

other microprocessors. Simulation results show that the performance of our mechanism is better than that of other microprocessors. Under performance/cost consideration, our mechanism can issue 2.07 X86 instructions per cycle.

Keywords：x86 Instruction Set, Microprocessor, Data-flow, Superscalar, Data-flow Graph, High-bandwidth Data Access.

二、緣由與目的

Researchers have discussed the use of ILP (instruction-level parallelism) to accelerate performance for more than 20 years. In order to achieve high ILP, many kinds of microprocessors have been proposed, such as VLIW, superscalar and pipeline processors. However, the complex X86 instruction set makes it hard to improve the performance of X86 processors. For examples, checking instruction boundary limits the exploitation of the ILP of superscalar processors, and increases hardware cost. Complex instruction semantics makes it necessary to translate X86 instructions to RISC-like instructions, and increases the burden of data dependence checking. Besides, the complex addressing modes make it hard to raise the clock rate due to inefficient pipeline stage design. Even the first Intel IA-64 processor, Merced [2], translates X86 instructions to VLIW instructions by a special hardware. However, the clock rate of Merced is low because Merced increases the stage delay due to more hardware complexity. Subsequently, Intel publicizes second IA-64 processor—Itanium — to translate X86 instructions to VLIW instructions by the compiler since Merced wastes much time to do instruction translation. But this processor needs excellent compiler and recompiles all original applications. It is inconvenient for users to do it.

The conceptions of Tomasulo's algorithm and scoreboarding were applied to implement the mechanism of out-of-order execution in early 1960. These conceptions were used in high performance computer architectures, and were the pioneers of data-driven computation. Although out-of-order execution is only implemented in function units, the performance it enhances is good. This application is very suitable in the computer architectures of traditional von Neumann machines. Because of the demand of large and native ILP, the design of data-driven computer should be constructed based on the execution with data flow graphs.

三、結果與討論

3.1 The DF86 Microarchitecture

By the data-driven concepts, the microarchitecture is designed following three directions, which are the Loop Unrolling, the Macro-Node Function, and the Data Superscaling. This architecture is named as the DF86 architecture. Figure 1 is used to illustrate the DF86 architecture.

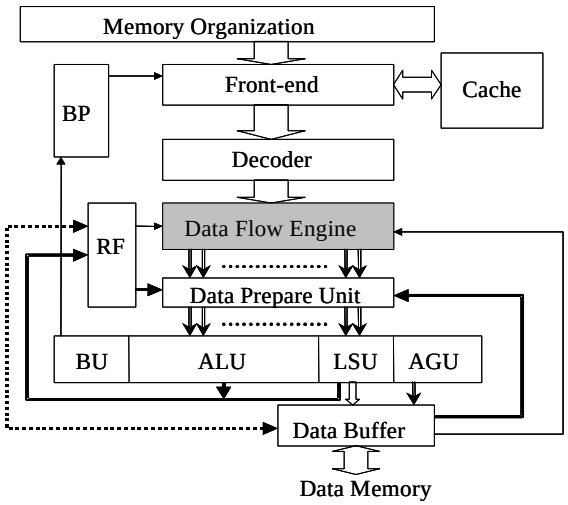

Figure 1. The DF86 Architecture

The Front-end unit is an instruction fetch unit that supports the execution traces as the traditional control flow. The Decoder decodes an instruction to an operator node and token-like data that represent the source and destination operands of an instruction. The Data-flow Engine maintains data-flow graphs and firing rules. The Data Buffer

manages the data accesses with the Data Memory. The Register File (RF) is used to construct a register token. The Data Prepare Unit combines the operator node and the ready data that the operator node needs to form an executable instruction. Then we send the executable instruction to the ALU for execution.

The Loop Unrolling is based on the dynamic loop semantic analysis to build the data tag. By the data driven computation method and the data superscalar mechanism, the operators of loop match their partner in data flow engine to make massive parallel processing. Following the dynamic semantic analysis, the function call is handled as the methods of macro node in the data flow computer. The Macro-Node Function method is alternative with traditional function invoking methods. With the dependence relations, between caller‘s parameter and callee’s arguments, and between return values and results, the inter-function data transfer mechanism can be reduced to speedup function-invoked time. The data superscalar can pump data from memory to support structure data processing, which can be analyzed by the dynamic semantic analysis. The pumping data unit can fetch multiple data from a set of data lines and tag them for partner metching to achieve massive parallel instruction execution.

3.2 The Loop Unrolling Architecture

When the *Loop Semantic Analyzer* detects a loop, it will send the parameters of requested data to the *Pumping-Data Unit* and set up loop unrolling environment at the *Match Unit*. Once the data enters the Match Unit in parallel and is matched with its partner, we will put the corresponding instructions of the data flow graph to data prepare unit to wait for execution. Then the functional unit will send the result data back to the pumping-data unit via Register File, in order to continuously precede the operations that are dependent with the result data. Figure 2 is used to illustrate the Loop Unrolling architecture.

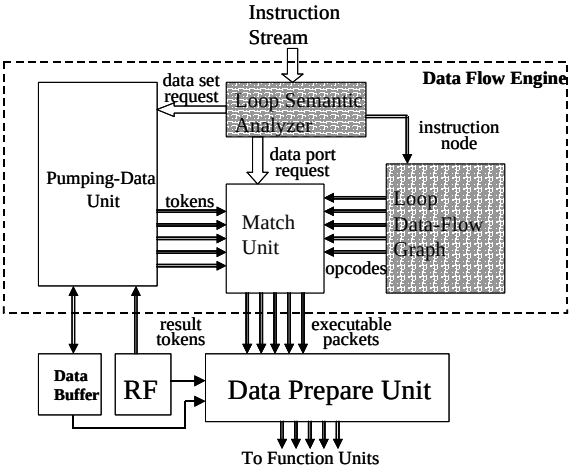

Figure 2. The Loop Unrolling Architecture

3.3 The Macro-Node Function Architecture

Figure 3 is used to illustrate the macro-node function architecture. The Function-invoked Semantic Analyzer analyzes the executed instruction streams and constructs the dependence relations, between caller‘s parameter and callee’s arguments, and between return values and results. It manages the data-flow graph frame for each function and makes the function’s data input/output descriptions. By the data driven nature, the function can be invoked by data dependences.

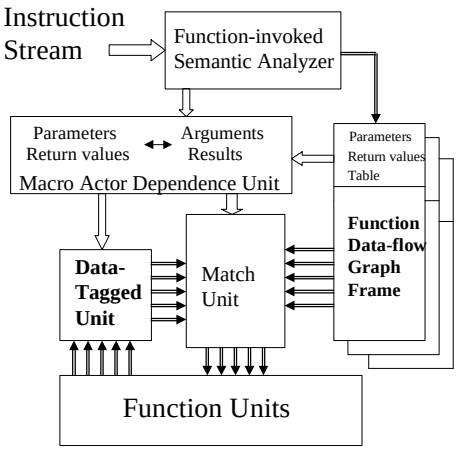

Figure 3. The Macro-Node Function Architecture

3.4. The Data Superscaling

The data superscalar can pump data from memory to support structure data

access to achieve massive parallel instruction execution. Figure 4 is used to illustrate the pumping data architecture.

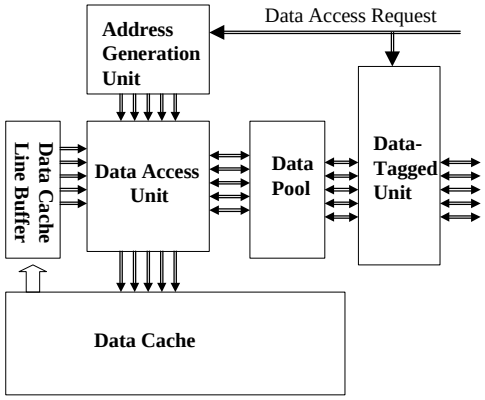

Figure 4. The pumping data architecture

3.5 Performance Analysis

We compare the issue rate of our mechanism to that of other microprocessors. The results are shown in Figure 5. In this figure, we apply the semantic-based loop unrolling mechanism to the two simulations, noted as *our choice* and *Ideal*. In *our choice*, we assume it has 7 issue degree, 3 unrolling degree, 64-entry inst-queue, and 3 loop frames within a function frame. On the other hand, the *Ideal* means unlimited issue degree, unlimited unrolling degree, unlimited-entry inst-queue, and unlimited loop frames within a function frame.

From this figure, we find that the performance of *Ideal* is excellent and its issue rate is 2.3 instructions per cycle. Besides, the issue rate of *our choice* also surpasses all other microprocessors and achieve at 2.07 instructions per cycle.

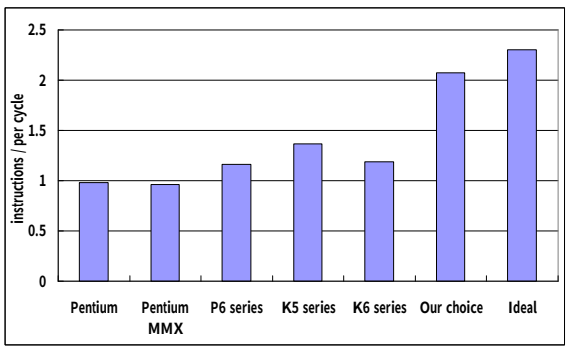

Figure 5 Issue rate comparisons.

四、計畫成果自評

In this project, we have designed an x86 architecture with a data-flow engine, called the DF86 microarchitecture, construct a simulator to evaluate the parameters of our mechanisms, and have excellent results. These show that the DF86 microarchitecture is better than other current microprocessors'. This project is fully matched with the proposal requirements. In the future, we will use the microarchitecture to solve following problems: the precise interrupt, the data structure handling, the data pipeline execution structure, and the multiple instruction analyzer. Finally, we will build a prototype with the high-density FPGA modules.

五、參考文獻

[1] B.R. Rau and J.A. Fisher, "Instruction-Level Parallel Processing: History, Overview and Perspective," in J. Supercomputing, Vol. 7, No. 1/2, pp. 9-50, 1993

[2] Intel Corporation,"Intel(R) Itanium(TM) processor microarchitecture overview," October 5-6, 1999

[3] AMD Corporation, "AMD Athlon(TM) Processor Architecture," August 23, 1999

[4] J.E. Thornton, "Parallel Operation in the Control Data 6600," in Proceedings of the Fall Joint Computers Conference, pp. 33-40, 1961

[5] E.J. Lerner, "Data-flow Architecture," in IEEE Spectrum, pp.57-62,April 1984

[6] J.W. Davidson and S. Jinturkar, "Improvinginstruction-level parallelism by loop unrolling and dynamic memory disambiguation," in Proceedings of the 28th Annual International Symposium on Microarchitecture, pp. 125 –132, 1995

[7] N. Bellas, I. Hajj, C. Polychronopoulos, and G. Stamoulis, "Energy and Performance Improvements in Microprocessor Design using a Loop Cache," in ICCD, 1999