2001년도 컴퓨터소사이어티 추계학술대회 논문집

SMT 마이크로프로세서의 명령어 이슈 구조

°문병인, 김문경, 홍인표, 이용석
연세대학교 전기전자공학과 프로세서 연구실
전화 : (02) 2123-2872 / 팩스 : (02) 312-4584

An Instruction Issue Structure for SMT Microprocessors

°Byung In Moon, Moon Gyung Kim, In Pyo Hong, and Yong Surk Lee
Processor Lab., Dept. of Electrical & Electronic Eng., Yonsei Univ.
E-mail : blue@dubiki.yonsei.ac.kr

Abstract

In this paper, we propose an instruction issue structure for SMT microprocessors. The proposed issue unit issues instructions belonging to the same thread in exact program order. The issue policy greatly reduces the design complexity and hardware cost of our structure, compared with the one adopting out-of-order issue. On the other hand, when the instructions belong to the different threads, the issue order for those instructions may not necessarily be the same with the fetch order. The issue unit issues instructions simultaneously from multiple threads to functional units by exploiting ILP and TLP. That parallel issue notably improves performance and resource utilization.

I. 서론

수퍼스칼라 구조의 근본적인 성능 한계와 반도체 집적 기술의 발달로 인해, 현재 TLP(Thread Level Parallelism)를 이용하여 ILP의 부족을 극복하는 방식의 다중 스레딩(multithreading) 구조가 주목을 받고 있다[1]. 이러한 다중 스레딩 구조 중에서 SMT(Simultaneous MultiThreading)는 다수의 스레드(thread)가 파이프라인(pipeline) 상에 동시에 존재하고, 다수의 스레드들로부터 명령어(instruction)들을 동시에 이슈(issue)하고 수행할 수 있는 구조이다[2-3]. SMT 구조에서는 다수의 스레드들이 프로세서의 자원들을 융통성 있게 공유하기 때문에, 자원 활용도(resource utilization)와 성능이 매우 높다. 이러한 이유로 인해, SMT는 차세대 마이크로프로세서 분야를 이끌어갈 구조로서 연구되고 있다.

마이크로프로세서의 병렬성(parallelism)을 높이기 위한 방법으로서 명령어들을 비순차적으로(out of order)로 이슈하고 실행하는 구조를 채택하는 경우가 있다. 그러나 비순차적 명령어 이슈는 구현이 복잡하고, SMT에서는 각 스레드의 ILP 부족을 TLP로서 보충하기 때문에 각 스레드의 ILP를 증가시키기 위한 비순차적 명령어 이슈에 대한 필요성이 적다.

본 논문은 같은 스레드에 속하는 명령어들을 순차적으로(in order) 이슈하고 실행하는 SMT 마이크로프로세서의 이슈 유닛(issue unit) 구조를 제시한다. 단, 서로 다른 스레드들에 속하는 명령어들 간에는 순서가 존재하지 않기 때문에, 그러한 명령어들 사이에는 이슈 순서에 대한 제한은 없다.

II. 전체적인 SMT 구조

전체적인 SMT 구조는 보통의 수퍼스칼라 구조에 대한 약간의 변경에 의해서 만들어진다. 그림 1은 4개의 스레드를 지원하는 SMT 구조의 전체적인 모습을 보여주는 블록 다이어그램이다. 그림 1의 전체적인 SMT 구조에서 수퍼스칼라 구조와의 차이는 페치 유닛 안에 스레드 선택기(thread selector)가 존재한다는 것과 다수의 PC가 존재한다는 것 및 레지스터 파일 안에 다수의 레지스터 세트가 존재한다는 것이다.

그림 1에서 페치 유닛은 명령어의 페치를 관리하며, 그 안의 스레드 선택기는 명령어를 페치할 스레드들을 선택하는 역할을 수행한다. 명령어 페치의 주소를 저장하고 있는 PC(Program Counter)는 스레드별로 한 개씩 존재한다. 페치된 명령어들은 명령어 이슈 큐(instruction issue queue)에 저장된 후에 디코드되어서

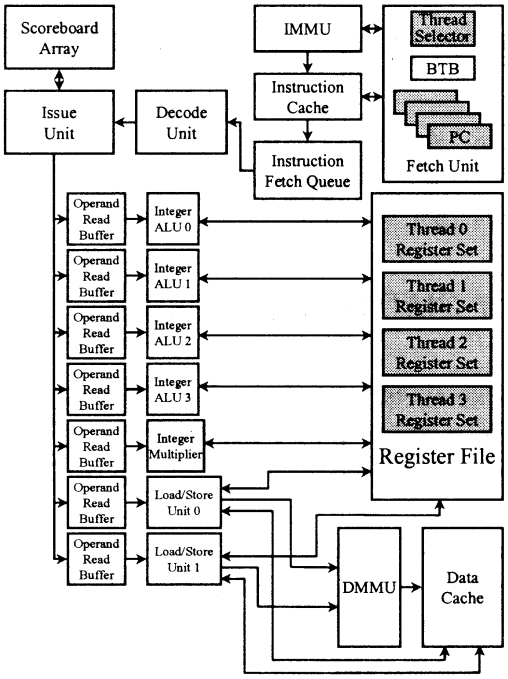

그림 1 전체적인 SMT 구조

Fig. 1 Overall SMT architecture

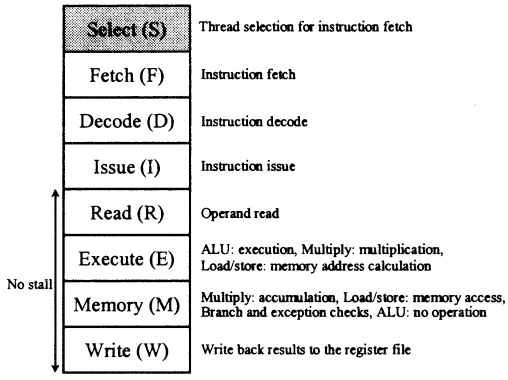

그림 2 SMT의 파이프라인 구조

Fig. 2 SMT pipeline structure

이슈 유닛 안의 명령어 이슈 큐(instruction issue queue)에 저장된다. 이슈 유닛은 명령어 이슈 큐의 명령어들을 기능 유닛들로 이슈한다. 기능 유닛들로 이슈된 명령어들은 레지스터 파일로부터 또는 데이터 바이패싱(bypassing)을 통해서 오퍼랜드를 얻은 후에 기능 유닛에서 명령어의 동작을 수행하고 그 결과를 레지스터 파일에 저장한다. 이러한 일련의 동작은 그림 2에서 보여주는 파이프라인 구조를 형성한다.

III. 이슈 유닛의 구조

이슈 유닛은 그림 2의 파이프라인 단계들 중에서 I(issue) 단계를 담당하는 유닛이다. 이슈 유닛은 명령어 이슈 큐 및 명령어 이슈 제어 로직으로 구성된다. 명령어 이슈 큐는 디코드된 명령어들을 저장하고 있다. 명령어 이슈 제어 로직은 명령어 이슈 큐의 명령어들을 검사하여 기능 유닛들로 이슈한다.

명령어 이슈와 관련해서는 다음과 같은 제한들이 존재한다. 우선, 같은 스레드에 속하는 명령어들에 대해서 명령어의 이슈는 페치 순서, 즉 명령어 이슈 큐 상의 순서대로 이루어진다. 그러나, 서로 다른 스레드들에 속하는 명령어들에 대해서는 서로간에 정해진 이슈 순서가 존재하지 않는다. 명령어를 이슈할 때에는 데이터 의존성(data dependency) 및 자원 충돌(resource conflict)을 검사한다. 데이터 의존성 검사는 해당 레지스터에 대한 스코어보드 엔트리의 내용을 검사함으로써 이루어지며, 기능 유닛에 대한 자원 충돌 검사는 그림 1의 오퍼랜드 읽기 버퍼(operand read buffer)들을 검사함으로써 수행된다.

서로 다른 스레드에 속하는 명령어들에 대한 이슈 순서는 명령어 이슈 큐의 순서와 다를 수 있다. 그리고, 스레드별로 명령어 플러쉬(flush) 동작이 지원된다. 이러한 두 가지 요인으로 인해 명령어 이슈 큐의 중간 엔트리가 비어있게 되는 현상이 발생하는데, 이것은 그림 3의 압축 동작에 의해서 제거된다.

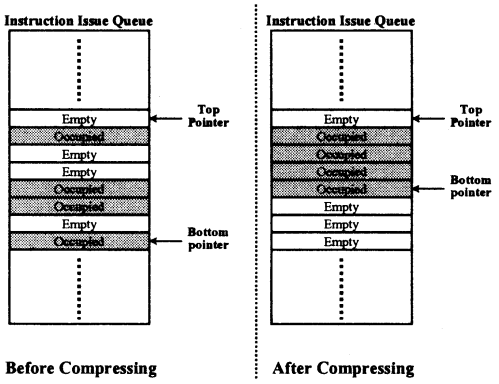

그림 3 명령어 이슈 큐의 압축

Fig. 3 Instruction issue queue compressing

SMT 구조가 수퍼스칼라 구조에 비해서 매우 높은 성능을 보이기는 하지만 설계의 복잡도가 매우 크다. 명령어 이슈 큐가 스레드별로 한 개씩 존재할 때에는 개개의 명령어 이슈 큐는 수퍼스칼라 구조의 명령어 이슈 큐와 같으며, 그림 3의 압축 동작이 필요 없게 된다. 그리고 그림 4와 같이 명령어 이슈 로직과 명령

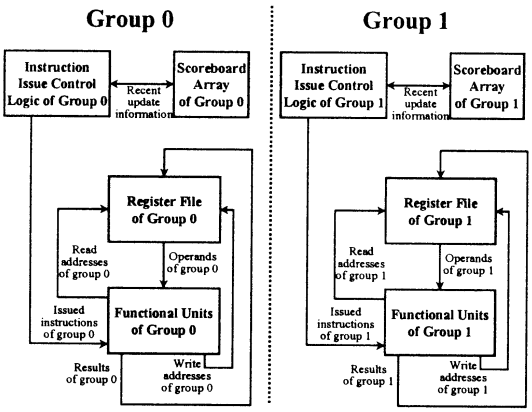

그림 4 명령어 실행 자원의 그룹화

Fig. 4 Grouping of instruction execution resources

어를 수행하기 위한 자원들 및 스레드들을 그룹으로 나누고, 해당 자원과 같은 그룹에 속하는 스레드들만이 그 자원을 공유할 수 있도록 함으로써, SMT 구조의 전체적인 설계 복잡도를 상당히 줄일 수 있다. 그러나 이러한 그룹화는 성능을 낮추는 경향이 있으므로, 그룹화에 의해 너무 많은 그룹으로 나누는 것은 피해야 한다.

표 1 성능 평가를 위한 입력 프로그램들

Table 1 Input programs for performance evaluation

Benchmark suite	Program
SPEC CPU2000	parser
	twolf
	vortex
	mcf
ADS	sorts
	Dhrystone

IV. 성능 평가 및 결과

성능 평가를 위해 이슈 유닛은 C 언어를 사용하여 시뮬레이터 블록으로 기술되었다. 시뮬레이터는 사이클-기본(cycle-based) 및 실행-구동(execution-driven) 방식을 사용하고, ARM 명령어 세트 구조(instruction set architecture)를 채택하여 구현되었다. 시뮬레이션은 페치 유닛, 기능 유닛 등의 SMT 전체 구조의 다른 부분들에 대한 시뮬레이터 블록들을 이슈 유닛의 시뮬레이터 블록과 통합하여 이루어진다. 이슈 유닛의 이슈 성능에 영향을 미치는 것으로는, 명령어 이슈 큐 엔트리의 수, 명령어 이슈 우선권 정책, 그룹화 등 다양하다. 적합한 이슈 구조를 정하기 위해서 이와 같은 요소들을 다양하게 변화시키면서 시뮬레이션을 수행한 후, 그 결과를 바탕으로 적절한 이슈 유닛의 구성을 정하였다. 시뮬레이션 과정에서 명령어 이슈 폭(issue width)과 스레드 수는 8개로 고정하였다. 그리고 표 1의 프로그램들을 ADS(ARM Developer Suite)[4]의 내장 컴파일러를 사용하여 컴파일한 실행 파일들이 시뮬레이터의 입력 벡터들로 사용되었다.

그림 5는 명령어 이슈 큐의 엔트리 수를 변화시키면서 수행한 시뮬레이션 결과이다. 결과에서 보여주듯이 명령어 이슈 큐의 엔트리 수는 32개가 가장 적당한 것으로 나왔다. 단 스레드별로 명령어 이슈 큐가 존재할 경우에는 전체 엔트리 수가 64개, 즉 큐별로 8개의 엔트리를 가지는 것이 가장 적절한 구성이다. 표 2는 명령어 이슈 우선권 정책에 따른 성능을 보여준다. OLDEST는 명령어 이슈 큐 중에서 가장 오래된 명령어들을 우선적으로 이슈하는 정책이고, ICNT_FU는 기능 유닛에 존재하는 명령어들이 가장 적은 스레드에, ICNT_MISS는 데이터 캐쉬 미스가 가장 적은 스레드에 이슈 우선권을 주는 정책이다. 세 가지의 우선권 정책 모두 비슷한 성능을 보이지만, 그 중에서 OLDEST가 약간이나마 성능이 더 좋고 구현이 가장 간단하다. 그룹화와 관련해서는 표 3에 나와 있듯이, 2개의 그룹으로 나누었을 경우는 그룹화를 적용하지 않았을 경우와 비슷한 성능을 보이면서 설계 복잡도를 줄일 수 있다. 반면에 4개의 그룹으로 나눌 경우에는 성능이 많이 나빠지는 것을 볼 수 있다. 전체적으로 SMT의 이슈 유닛은 5 이상의 이슈율(issue rate)을 보이는데, 이것은 수퍼스칼라 구조에서는 달성할 수 없는 성능이다.

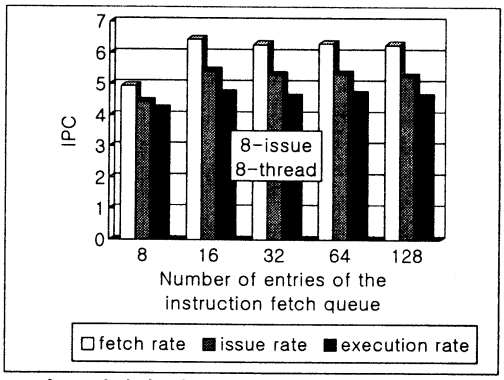

그림 5 명령어 이슈 큐 엔트리 수에 따른 성능

Fig. 5 Performance as a function of the number of the instruction issue queue

표 2 이슈 우선권 정책에 따른 성능 변화
Table 2 Performance vs. issue priority policy

이슈 우선권 정책	이슈율	실행률
OLDEST	5.3211	4.6680
ICNT_FU	5.2031	4.6174
ICNT_MISS	5.2185	4.5083

표 3 그룹화에 따른 성능 변화
Table 3 Performance as a function of grouping

그룹 수	이슈율	실행률
1	5.3211	4.6680
2	5.1242	4.5197
4	4.5117	4.0372

V. 결론

본 논문에서는 ILP와 TLP를 이용하여 프로세서의 성능을 높이는 SMT 마이크로프로세서의 이슈 유닛에 대한 구조를 연구하였다. 이슈 유닛은 같은 스레드에 속하는 명령어들은 순차적으로 이슈하며, 서로 다른 스레드들에 속하는 명령어들은 순서에 구애받지 않고 이슈한다. 시뮬레이션 결과 명령어 이슈 큐는 32개의 엔트리를 갖는 것이 적당하다. 그런데 명령어 이슈 큐가 스레드별로 존재할 경우에는 각 큐가 8개의 엔트리를 가지는 것이 비용과 성능을 고려했을 때 효율적이다. 그리고 명령어 수행 자원들을 2개의 그룹으로 나누는 그룹화는 성능을 유지하면서 설계 복잡도를 줄일 수 있는 방법이다.

감사의 글

본 연구는 한국과학기술기획평가원의 국가지정연구실사업(과제번호 : 2000-N-NL-01-C-246)에 의해 수행되었습니다.

참고문헌

[1] Peter Song, "Multithreading Comes of Age", Microprocessor Report, Vol. 11, No. 9, July 14, 1997.

[2] D.M. Tullsen, S.J. Eggers, J.S. Emer, and H.M. Levy, "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor", Proceedings of 23rd Annual International Symposium on Computer Architecture, pp. 191-202, Philadelphia, Pennsylvania, May 1996.

[3] Keith Diefendorff, "Compaq Chooses SMT for Alpha", Microprocessor Report, Vol. 13, No. 16, December 6, 1999.

[4] "ARM Developer Suite: Compiler, Linker, and Utilities Guide", ARM Limited, 2000.