논문 02-27-5C-13 한국통신학회논문지 '02-5 Vol.27 No.5C

SMT 프로세서에 최적화된 명령어 페치 전략에 관한 연구

정회원 홍 인 표*, 문 병 인*, 김 문 경*, 이 용 석**

An Optimal Instruction Fetch Strategy for SMT Processors

In-Pyo Hong*, Byung-In Moon*, Moon-Gyung Kim*, Yong-Surk Lee* ***Regular Members***

요 약

최근에 성능의 한계를 드러내고 있는 수퍼스칼라 RISC를 대체할 새로운 프로세서 구조로서 SMT(Simultaneous Multi-Threading)이 활발히 연구되고 있다. SMT는 하나의 프로세서에 여러 개의 스레드가 하드웨어 자원을 동적으로 공유하며 동시에 수행되는 구조이다. 이러한 환경에서는 프로세서 안에 존재하는 여러 스레드로부터 명령어를 원활하게 공급하여 주는 것이 중요하다. SMT 프로세서는 기존의 프로세서에 비하여 사이클 당 실제 처리되는 명령어 수가 월등히 많기 때문에, 사이클 당 명령어 페치량과 페치된 명령어를 임시 저장하는 페치 큐의 엔트리 수가 신중하게 결정되어야 한다. 또한 사이클마다 페치할 스레드와 각 스레드의 페치량을 결정하는 것이 성능에 큰 영향을 미친다. 따라서 본 논문에서는 이러한 요소들이 프로세서 전체의 성능에 미치는 영향을 분석하고 그 결과를 바탕으로 SMT 프로세서에 최적화된 명령어 페치 전략에 대하여 논한다.

ABSTRACT

Recently, conventional superscalar RISC processors arrive their performance limit, and many researches on the next-generation architecture are concentrated on SMT(Simultaneous Multi-Threading). In SMT processors, multiple threads are executed simultaneously and share hardware resources dynamically. In this case, it is more important to supply instructions from multiple threads to processor core efficiently than ever. Because SMT architecture shows higher IPC(Instructions per cycle) than superscalar architecture, performance is influenced by fetch bandwidth and the size of fetch queue. Moreover, to use TLP(Thread Level Parallelism) efficiently, fetch thread selection algorithm and fetch bandwidth for each selected threads must be carefully designed. Thus, in this paper, the performance values influenced by these factors are analyzed. Based on the results, an optimal instruction fetch strategy for SMT processors is proposed.

Ⅰ. 서 론

오늘날, 고성능 마이크로 프로세서의 대부분은 수퍼스칼라 RISC방식이 주종을 이루고 있다. 수퍼스칼라 RISC는 한 스레드(thread)의 명령어 흐름 안에서 명령어 단위의 병렬성(ILP)을 찾아 동시 수행함으로써 성능을 높이는 방법이다. 그러나, 현재 이 방식의 프로세서는 성능의 한계점에 도달하게 되었고, 이를 해결하기 위한 아키텍처의 향상은 스레드 단위의 병렬성(TLP)을 이용하는 것에 초점이 맞추어져 있다.

TLP를 이용하기 위한 아키텍처의 하나는 멀티프로세서이다. 멀티프로세서는 한 칩에 프로세서 코어를 여러 개 설치한 것으로, 하드웨어 자원의 공유가 원활하지 않아 프로세서의 수보다 스레드의 수가 작은 경우 자원 활용도가 떨어지는 단점이 있다.

Simultaneous Multi-Threading(SMT) 아키텍처는

* 연세대학교 전기전자공학과 프로세서연구실(necross@dubiki.yonsei.ac.kr)
논문번호 : K01219-1105, 접수일자 : 2001년 11월 5일
※ 본 연구는 과학기술부의 국가지정 연구실 사업과 삼성전자에 지원으로 수행되었습니다.

멀티프로세서의 이러한 단점을 극복한 것으로, 여러 스레드가 자원을 동적으로 공유하면서 ILP와 TLP를 효율적으로 이용하도록 되어 있다. SMT 구조는 하나의 프로세서 안에 존재하는 여러 개의 스레드에서 명령어를 선택하여 페치하여야 하므로, 명령어 페치(fetch) 단계의 자원 할당량과 페치 스레드 선택 알고리즘에 따라 프로세서의 성능이 영향을 받는다. 본 논문에서는 프로세서의 페치 자원과 페치 알고리즘에 따른 성능의 변화를 분석하였다. 2장에서는 SMT 프로세서의 전체적인 구조에 대하여 논하고, 3장에서는 명령어 페치 유닛의 구조, 페치 알고리즘들을, 4장에서는 연구 방법에 대하여 설명한다. 5장에서 성능평가 결과를 보이고 6장에서 결론을 맺는다.

Ⅱ. SMT 프로세서의 아키텍처

2.1. 호환 명령어 셋의 선택

SMT 구조의 프로세서는 독자적인 명령어 셋(Instruction Set Architecture, ISA)을 가지는 것으로 설계되지 않고, 기존 프로세서의 명령어 셋과 호환되는 것으로 하였다. 독자적인 명령어 셋을 가지는 프로세서는 컴파일러, 개발환경, OS등을 완전히 새로 만들어야 하므로, 본 연구의 결과물을 활용하는 데에 어려움이 있다.

연구 초기 단계에는 ALPHA, MIPS, ARM의 세 가지의 명령어 셋을 염두에 두고 선택하였다. ALPHA는 고성능 수퍼스칼라 RISC 프로세서로서 SMT 구조로 변환하기에 가장 유리한 구조이다. 그러나 ALPHA는 일부 고성능 서버, 워크스테이션에 사용되는 것으로 국내에서 SMT 구조로의 연구를 완료한다 하여도 시장 진입이 쉽지 않다. MIPS 또한 고성능 RISC 구조로서 SMT로의 이행은 비교적 용이하지만 개발환경이나 연구 결과의 파급효과 면에서 ARM보다 효과적이지 못하다. ARM[1]은 소형 내장형 프로세서로서 상태 레지스터가 존재하고 모든 명령어가 조건부 실행을 지원하여 수퍼스칼라, SMT 구조 등으로 고성능화 하기에 어렵다. 상태 레지스터는 거의 모든 명령어가 목표 레지스터(destination register)로 사용하므로, 명령어간의 종속관계를 예측하는 데에 있어 문제를 복잡하게 만든다. 또한 조건부 실행은 스칼라 RISC 프로세서에서 잦은 분기를 제거하는 효과가 있었으나, SMT나 수퍼스칼라와 같은 다중 명령어 실행 환경에서는 또 다른 명령어간의 종속관계를 만들게 되어 명령어 이슈 시에 종속관계를 해결하기 힘들게 한다. 하지만, ARM은 시장 진입이 용이하고 현재 더 많은 응용 프로그램과 개발환경이 갖추어져 있으며, 가격 경쟁력이 우수하다. 따라서 이러한 장점을 수용하여 SMT 구조로 변환의 어려움에도 불구하고, ARM 명령어 셋을 본 연구의 SMT 프로세서 호환 명령어 셋으로 선택하였다.

2.2. SMT ARM 프로세서의 기본 구조

본 논문에 제안된 SMT ARM 프로세서의 구조는 위에서 설명된 ARM 명령어 셋의 단점을 보완하기 위한 단순한 명령어 이슈 및 종속성 예측 방식과 효율적인 하드웨어 자원 공유를 통한 비용절감을 염두에 두고 설계되었다.

기존에 Tullsen[2][3]이 제안한 SMT 구조는 MIPS 구조를 기반으로 하여 register renaming과 out-of-order issue 및 completion을 지원하였다. 그러나, ARM 구조에서는 상태 레지스터와 조건부 실행 지원 등, 명령어 셋의 구조적인 단점으로 인하여 out-of-order 방식의 설계를 하기가 어렵다. 이런 상황에서 out-of-order 방식으로 설계를 하면 하드웨어 복잡도가 증가하고 이로 인하여 프로세서 전체의 지연시간 또한 증가하므로 오히려 성능에 악영향을 미칠 수 있다. 따라서 본 논문에서 제안한 SMT ARM 구조는 간단한 in-order issue 및 completion 방식을 채택하였다. SMT 구조는 기본적으로 ILP와 TLP를 동시에 효과적으로 이용하도록 되어 있으므로, 한 스레드 내에서는 in-order issue 및 completion 방식으로 인하여 ILP를 다소 덜 이용한다고 하여도 이를 TLP로 보충하여 프로세서 전체적인 성능을 높일 수 있다. 따라서 in-order 방식으로 설계하여도 수퍼스칼라인 경우와는 달리 성능 저하 문제가 발생하지 않는다.

SMT ARM 구조는 명령어 큐, 디코드 유닛을 비롯한 대부분의 하드웨어 자원을 스레드들이 효율적으로 공유하여 사용하므로 하드웨어 자원 효율 향상을 통한 비용절감 효과를 볼 수 있다. 이에 반하여 Hirata[4]가 제안한 구조나 Clearwater Networks[5]의 SMT 방식의 프로세서는 각 스레드마다 명령어 큐와 디코드 유닛을 두고 실행 유닛만을 공유하는 형태로 되어 있어, 본 논문의 SMT ARM 구조에 비하여 하드웨어 자원 이용 효율이 낮다.

그림 1은 8개의 스레드를 지원하는 경우에 대하여 SMT ARM 프로세서의 블록도를 보여준다. 그림에서 보듯이 SMT를 지원하기 위하여 PC(Prog-

ram Counter)와 레지스터 셋, 레지스터 셋에 대응하는 스코어보드[6] 어레이가 각 스레드마다 존재하며, 그 외에 캐쉬 읽기 포트, 페치 큐, 디코더 유닛, 레지스터 파일의 읽기/쓰기 포트, 명령어 윈도우, 기능 유닛, 데이터 캐쉬, 결과 버스 등의 하드웨어 자원은 모든 스레드가 동적으로 공유한다.

페치 스레드 셀렉터는 각 스레드의 페치 가능성, 멀티 포트로 설계되는 명령어 캐쉬의 포트 수 등을 고려하여 다음 사이클에 명령어를 페치할 스레드와 그 스레드에 할당될 수 있는 캐쉬 포트 수를 결정한다. 이러한 동작은 페치 단계에서 수행할 수도 있지만, 페치 단계에서 페치할 스레드를 결정하고 명령어 캐쉬를 읽는 동작을 모두 수행하기에는 사이클 시간이 너무 짧다. 따라서 페치 스레드를 선택하는 동작을 하나의 파이프라인 단계로 독립시킨다.

페치유닛은 매 사이클마다 페치 스레드 셀렉터에서 할당된 캐쉬 포트를 이용하여 명령어를 읽어와서 명령어 페치 큐에 넣는다. 이를 위해서 명령어 캐쉬는 non-blocking 캐쉬를 사용하였다. 기존의 수퍼스칼라 구조는 일반적으로 일정 수행 사이클 동안 하나의 스레드만을 처리하므로, 그 스레드에서 캐쉬 미스(Cache miss)가 발생하면 모든 명령어의 페치가 중단된다. 따라서 수퍼스칼라에서는 명령어 캐쉬는 blocking 캐쉬이다. 그러나 SMT에서는 동시에 여러 스레드가 명령어를 페치할 뿐더러 한 스레드에서 캐쉬 미스가 발생하여도 다른 스레드에서는 명령어를 페치할 수 있어야 하므로 다중 포트 non-blocking 캐쉬가 요구된다. 또한 페치 단계에서는 BTB를 이용한 분기 예측을 수행한다. 예측된 분기 결과는 BHB에 분기 명령어의 순서대로 저장되어 실행 단계에서 실제 분기 명령어가 수행되고 정확한 분기 결과가 판별되었을 때, 예측된 분기가 맞았는가를 검사하는 데에 사용된다.

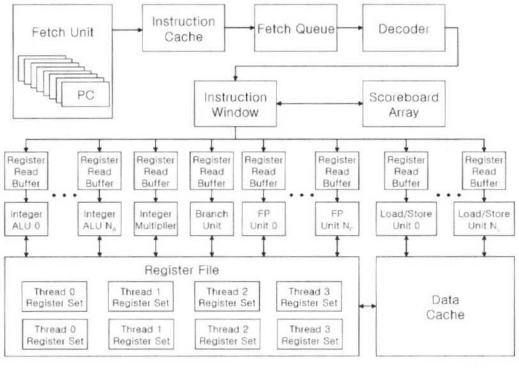

그림 1. 8개의 스레드를 지원하는 SMT 구조

페치 단에서 읽어들인 명령어는 명령어 페치 큐의 끝단부터 저장되며, 디코드 유닛은 페치 큐의 앞단에서부터 명령어를 가져와서 해석한다. 디코드 유닛은 각 명령어의 동작에 따라서 필요한 기능 유닛의 종류와 동작에 관계되는 레지스터 주소 또는 명령어에 포함된 immediate data를 결정한다. 디코드 유닛은 명령어를 해석한 결과는 명령어 창(Instruction Window)에 넣는다.

명령어 창에 들어있는 명령어는 각 스레드 안에서는 프로그램 순서대로 이슈되며, 스레드간에는 순서를 짓지 않는다. 즉, 이슈 로직은 각 스레드의 가장 앞에 위치한 명령어에서부터 종속성 문제, bypassing과 실행 유닛의 사용가능성 등을 고려하여 이슈 여부를 결정하게 된다. 앞의 명령어가 종속성이 해결되지 않아 이슈되지 못하면 같은 스레드에 속하는 후속 명령어들은 자동적으로 이슈되지 못한다. 그러나 다른 스레드에 속한 명령어들은 비록 그 명령어가 명령어 창에서 순서가 더 뒤에 위치한다 하여도 종속성이 해결되면 바로 이슈된다. 따라서 명령어 창은 기본적으로는 FIFO 형태를 가지지만, 각 명령어들 사이에 이슈되어 빠져나간 엔트리가 있을 수 있다. 이러한 경우 명령어 창의 효율을 높이기 위하여 중간에 비어있는 엔트리를 없애주는 compaction을 수행한다.

이슈가 결정된 명령어들은 명령어 창에서 나와서 각 실행 유닛의 앞단에 위치한 레지스터 읽기 단계로 들어간다. 즉, 레지스터 읽기 단계는 각 실행 유닛의 하부 블록이다. 레지스터 읽기 단계에서는 디코드 유닛에서 해석한 소스 레지스터 주소를 이용하여 레지스터 값을 읽고, 이를 실행 유닛에 전달한다. 또한 레지스터 읽기 단계에서 다중 사이클 명령어인 경우, 스테이트 머신을 가동하여 그 명령어가 해당 사이클 동안 특정 기능 유닛을 점유하고 사용할 수 있도록 해준다. 다중 사이클 명령어는 레지스터 읽기 단계에서 여러 개의 단일 사이클 동작으로 분해되어 스테이트 머신의 제어에 따라 수행된다. 이러한 레지스터 접근 동작은 기존의 수퍼스칼라에서는 실행 유닛의 실행 단계 전반부에서 수행된다. 그러나, T개의 스레드를 지원하는 SMT는 수퍼스칼라에 비하여 레지스터 파일의 크기가 T배가 되므로 레지스터 파일의 접근 시간이 느려진다. 따라서 이 동작은 하나의 파이프라인 단계로 분리하여야 한다.

이렇게 전달된 동작 정보들을 통해 실행 단계에서는 기능 유닛 안에서 주어진 동작이 실행된다. 명

령어 실행의 결과는 결과 버퍼에 저장되어 '읽기 후 쓰기'문제를 해결해 준다, '쓰기 후 읽기' 문제를 해결하기 위해서 필요하다면 연산 결과를 해당 동작 유닛들로 직접 전달해 줌(bypassing)으로써 동작에 필요한 레지스터 파일의 읽기 포트 수를 줄일 수 있다.

다음으로 메모리 단계에서는 메모리 제어를 통해서 데이터를 읽거나 쓰게 되는데, 메모리 동작을 수행하지 않는 유닛의 경우에는 하나의 버퍼를 더 두어 전체적인 파이프라인 단계를 맞추어 준다. 또한 메모리 단계에서 분기 예측 실패나 예외상황(Exception)에 대하여 복구해 주는 동작이 일어난다.

끝으로, 모든 결과가 끝나 결과버퍼에 기록된 내용들은 각 동작 유닛별로 연결되어 있는 쓰기 포트를 통해 동시에 레지스터 파일로 기록되고, 해당 스코어보드 비트는 클리어 된다.

이와 같은 파이프라인 구조를 도식적으로 나타내면 그림 2와 같으며, 각 파이프라인 단계에서 수행하는 동작을 요약하면 다음과 같다.

- 페치 스레드 선택 단계(S) : 다음 사이클에 명령어를 페치할 수 있는 스레드를 선택하고 캐쉬 포트를 할당한다.
- 페치 단계(F) : 다중 스레드들로부터 명령어들을 페치 하여 페치 큐에 저장한다.
- 디코드 단계(D) : 명령어들이 해석된다.
- 이슈 단계(I) : 종속 문제 또는 자원 충돌 문제가 없는 명령어들이 동작 유닛들로 보내어진다.
- 읽기 단계(R) : 명령어 연산자들을 레지스터 파일에서 읽어 오며, 다중 사이클 명령의 경우에는 스테이트 머신을 통해 동작 유닛의 제어를 시작한다.
- 실행 단계(E) : 동작 유닛에서 명령어들을 실행하고, 메모리 주소 등을 계산하며 그 결과는 결과 버퍼에 기록된다.
- 메모리 단계(M) : 해당 메모리 주소값을 통해 데이터를 읽거나 쓴다.
- 기록 단계(W) : 실행 결과가 순서대로 레지스터 파일에 기록된다.

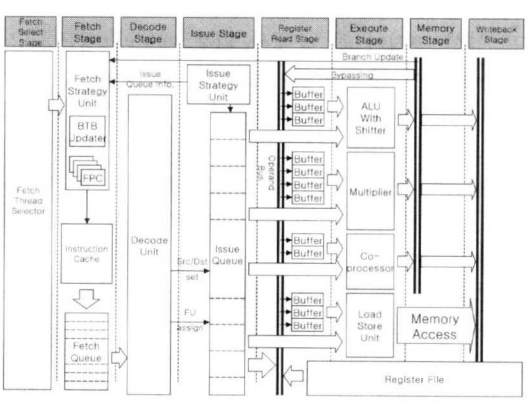

그림 2. 파이프라인 단계 구조도

이와 같은 파이프라인 구조는 수퍼스칼라의 파이프라인 구조를 기반으로 한 것이다. 본 논문에서 제시하는 SMT 구조는 수퍼스칼라와 비교하여 다음과 같은 차이점이 있다.

- 일반적인 고성능 수퍼스칼라와 달리 in-order 기반의 이슈와 completion 방식을 가진다. 이는 ARM 명령어 셋의 단점으로 인하여 이슈와 completion 로직이 SMT 구조에서 지나치게 복잡해질 수 있기 때문에 하드웨어 구현 가능성을 고려하여 단순화한 것이다. 이러한 단순화가 수퍼스칼라에서는 성능저하를 야기할 수 있지만, SMT 상에서는 TLP를 이용하므로 성능저하 현상이 극복될 수 있다.
- SMT 구조에서는 각 명령어의 소속 스레드 번호가 각 파이프라인 단계를 따라서 흘러가게 된다. 이 스레드 번호는 각 명령어 사이의 종속성 관계를 파악하는 데에 사용되고, 분기 예측 실패나 익셉션, 인터럽트가 발생하였을 때, 해당 스레드의 후속 명령어들을 무효화하는 등의 동작을 위하여 필수적이다. 또한, 레지스터 파일에서 해당 스레드의 지정된 레지스터를 찾아 접근하는 데에도 사용된다.
- SMT는 수퍼스칼라에 비하여 각 파이프라인에서 수행할 동작이 다소 복잡하므로, SMT 동작을 수행하는 데에 필요한 페치 스레드 선택, 레지스터 읽기 등의 파이프라인 단계를 추가하였다.
- 명령어 캐쉬는 non-blocking 구조로 설계된다. 이는 한 스레드의 캐쉬 미스가 다른 스레드의 명령어 페치를 방해하는 일을 막기 위함이다.
- 스레드 수만큼의 프로그램 카운터와 레지스터 파일 그리고 이에 따르는 스코어보드 엔트리들이 존재한다. 특히 레지스터 파일은 크기가 증가함에 따라 접근 시간이 느려지므로 레지스터 접근을 위한 파이프라인 단계가 추가된다.

본 연구에서는 위의 파이프라인 구조에 따라 8개의 스레드를 지원하도록 SMT 프로세서 구조를 설계하였다. 사이클 당 이슈할 수 있는 최대 명령어 수는 8개이며, 명령어 캐쉬와 데이터 캐쉬는 32KB 씩 가지도록 설계되었다.

Ⅲ. 명령어 페치 유닛의 구조

명령어의 페치는 명령어 캐쉬에서 다음 수행할 명령어들을 프로세서 코어로 가져오는 동작으로서, SMT ARM 구조에서는 페치 스레드 선택, 명령어 페치의 두 단계로 이루어진다. 다음 그림은 페치 유닛의 대략적인 구조를 나타낸다.

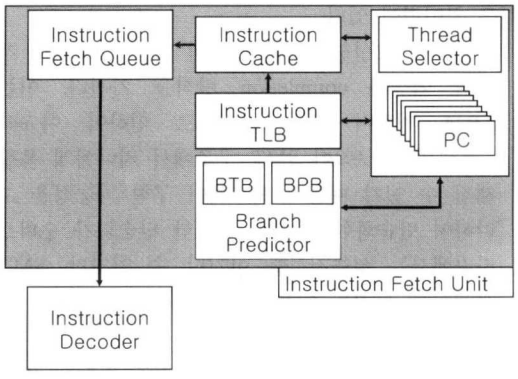

그림 3. Fetch unit의 구조

3.1 페치 스레드 선택

SMT 구조는 하나의 프로세서에서 여러 스레드가 활성화되어 존재하며, 매 사이클마다 여러 스레드의 명령어가 동적으로 자원을 공유하며 수행된다. SMT의 장점을 살려 ILP의 한계를 극복하고 TLP를 충분히 이용하려면, 여러 개의 스레드에서 명령어를 원활하게 공급하여 주는 것이 중요하다.

한 사이클에 프로세서에 활성화되어 있는 모든 스레드에서 명령어를 페치해 오는 것은 불가능하다. 명령어 캐쉬의 포트 수가 모자라는 경우도 있고, 특정 스레드의 예외상황이나, 분기 예측 실패로 인하여 명령어 페치가 불가능한 경우도 있다. 따라서 SMT 구조에서는 실제 명령어를 명령어 캐쉬에서 가져오기 전에 다음 사이클에 페치가 가능한 스레드와 그 스레드가 사용할 수 있는 캐쉬 포트의 양을 결정해야 한다. 페치 스레드 셀렉터가 이러한 역할을 수행한다.

페치 스레드 셀렉터는 각 스레드들을 선택사항으로 지정된 우선 순위 알고리즘에 따라 순서를 정하여 우선 순위 리스트를 작성한다. 작성된 우선 순위 리스트의 스레드 중에서 페치가 불가능한 것들을 선택하여 제거하고, 남은 스레드에 대하여 우선 순위에 따라 명령어 캐쉬 포트를 할당한다.

페치가 가능한 스레드 모두에 균등하게 명령어 캐쉬 포트를 할당하는 것보다 현재 프로세서의 상황을 검색하여 프로세서의 파이프라인을 정지시킬 확률이 가장 작은 스레드를 우선적으로 페치함으로써 프로세서의 성능을 높일 수 있다. 이에 따라 시뮬레이션 시에 선택할 수 있는 우선 순위 알고리즘은 다음과 같다.

- ICOUNT_IFQ

명령어 페치 큐에 가장 적은 수의 명령어가 들어있는 스레드에 우선권을 준다.

- ICOUNT_Q

명령어 페치 큐와 명령어 이슈 큐에 가장 적은 수의 명령어가 들어있는 스레드에 우선권을 준다.

- ICOUNT_ALL

명령어 페치 큐, 이슈 큐와 모든 파이프라인 단계에 가장 적은 수의 명령어가 들어있는 스레드에 우선권을 준다.

- ICOUNT_BHB

BHB에 가장 적은 수의 명령어가 들어있는 스레드에 우선권을 준다. BHB는 파이프라인 상의 분기 명령에 대한 기록을 저장해놓은 것으로서, BHB에 명령어가 많이 들어있다는 것은 현재 파이프라인에 해당 스레드의 분기 명령이 많아 분기 예측 오류가 일어날 가능성이 크다는 것이다. 분기 예측 오류가 발생하면 분기 명령 후속의 명령어들은 무효화되므로, 현 사이클에 페치한 명령어도 무효화될 가능성이 있다. 따라서 BHB에 명령어가 많이 저장된 스레드의 페치는 우선권을 낮추어 준다.

- ICOUNT_LB

Load Buffer(LB)에 가장 적은 수의 명령어가 들어있는 스레드에 우선권을 준다. LB는 외부 메모리에서의 데이터 load를 수행하는 명령어가 저장되는 곳이다. 따라서 LB에 명령어가 많이 들어있다는 것은 앞선 명령어 들 중에서 외부 메모리 접근을 시도하는 것이 많다는 것이고, 이는 큰 지연시간을 갖는다. 이 스레드에 대하여는 현 사이클에 페치를 하여도 페치된 명령어들은 이슈되지 못하고 큐에 쌓일 확률이 크므로, LB에 많은 명령어가 저장된 스레드는 페치 우선 순위를 낮추어 준다.

- IQOL

명령어 이슈 큐에서 후미(tail) 쪽에 명령어가 존재하는 스레드부터 우선권을 준다.

- RR

Round-robin 알고리즘에 따라 각 스레드의 우선 순위를 결정짓는다.

- 위의 알고리즘을 수행한 결과가 동일한 경우에는 스레드 ID가 작은 스레드가 우선권을 갖는다.

위의 우선 순위에 따라 순서가 지어진 스레드들에 대하여 페치 가능성을 체크한다. 각 스레드의 선행 명령어가 익셉션을 발생시켰는지 혹은 페치 큐에 빈 공간이 없는지를 검사한다. 명령어 페치 큐는 grouping 2에 의하여 그룹 지어질 수 있기 때문에 특정 그룹에 속하는 스레드들에 대하여서는 명령어 페치 큐에 빈 공간이 존재하지 않을 수 있다.(Grouping 2의 그룹수가 1인 경우에는 하나의 페치 큐를 모든 스레드가 공유하며, 이러한 경우 페치 큐에 빈 공간이 없으면 모든 스레드의 페치가 중단된다.) 이러한 스레드들에 대하여는 다음 사이클에 바로 페치가 불가능하므로 캐쉬 포트 할당 대상에서 제외한다. 페치 선택에서 스레드가 제외되는 조건은 다음과 같다.

- 명령어 페치 큐가 가득참
- 해당 스레드의 선행 명령어의 익셉션 발생
- 명령어 TLB miss 처리 중
- 명령어 cache miss 처리 중

페치 가능성을 검사하고 이를 통과한 스레드들은 명령어 캐쉬 포트를 할당받는다. 명령어 캐쉬 포트는 우선 순위가 높은 스레드부터 하나씩 할당받게 되고, 이 과정이 완료된 후, 남는 캐쉬 포트가 있을 때에는 남는 포트를 우선 순위가 높은 스레드에 하나씩 더 할당한다. 자주 발생하지 않는 일이지만, 위의 두 번의 할당 과정을 거쳐도 포트가 남을 경우에는 다시 우선 순위가 높은 것부터 2개씩의 포트를 할당한다. 이는 캐쉬 포트가 1, 2, 4개 단위로 스레드에 할당될 수 있도록 하기 위함이다.

이러한 과정을 거쳐 페치할 스레드와 포트 수를 결정한 후, 다음 사이클에 실제 명령어 페치가 수행된다. 이는 페치 스레드 셀렉트 단계와 실제 페치 단계가 각각 독립적인 파이프라인 단계이기 때문이다. 따라서 현 사이클에 페치되고 있는 명령어는 전 사이클에 선택된 스레드에서 나온 명령어이다.

3.2 명령어 페치

페치 단계에서는 페치 스레드 선택 단계에서 할당받은 명령어 캐쉬 포트를 사용하여 명령어를 읽어와서 명령어 페치 큐에 넣는다. 이 때, 분기 예측을 위하여 BTB도 함께 읽혀지며, 논리적 메모리 주소를 물리적 메모리 주소로 변환하기 위한 TLB 접근도 동시에 수행된다.

명령어 캐쉬는 32KB의 용량을 가지며, 8-way set assocaitive 방식이다. SMT 구조에서는 스레드가 캐쉬를 공유함으로 인하여 thrashing 현상이 발생할 수 있으므로, 명령어 캐쉬의 associativity를 수퍼스칼라에 비하여 크게 설정한다. 다중 포트로 설계되는 명령어 캐쉬는 하나의 포트가 두개의 명령어를 페치할 수 있도록 설계되어 있다. 따라서 사이클 당 최대 페치 가능한 명령어 수가 8이라는 것은 캐쉬의 포트수가 네 개 존재한다는 것을 의미한다. 이들 명령어 캐쉬 포트는 스레트 선택 단계의 결과에 따라서 하나의 스레드에 하나씩 할당될 수도 있고, 여러개의 포트가 하나의 스레드에 할당될 수도 있다.

명령어가 페치되는 도중에 명령어 페치 큐가 가득 찰 경우에는 현재 페치된 것까지만 처리된 것으로 하고 PC를 조정한다. 그리고 같은 페치 큐를 공유하는(같은 그룹에 속하는) 후속 스레드의 페치가 중단되고 해당 스레드는 페치 스레드 선택에서 제외된다.

분기 예측을 위한 BTB는 캐쉬의 한 라인 당 하나의 엔트리가 존재하며, 그 엔트리에는 캐쉬에 들어있는 두 명령어에 하나씩의 분기 예측 정보를 제공한다. 즉, 캐쉬 라인에 속하는 명령어들을 두 명령어씩 묶어서 그 안에 분기 명령이 있는가와 그 분기 명령의 예측된 목적 주소가 어디인지에 관한 정보를 제공하는 것이다. 페치 단계에서 이 분기 예측 정보는 캐쉬의 명령어와 함께 읽혀지며, 페치를 수행할 때 페치된 명령어에 분기할 것으로 예측된 분기 명령이 존재하는 경우에는 그 명령까지만 페치 큐에 삽입하고 페치를 중단한다. 그리고 다음 사이클에 예측된 목적 주소로부터 페치를 수행하기 위하여 PC를 예측된 목적 주소로 설정한다.

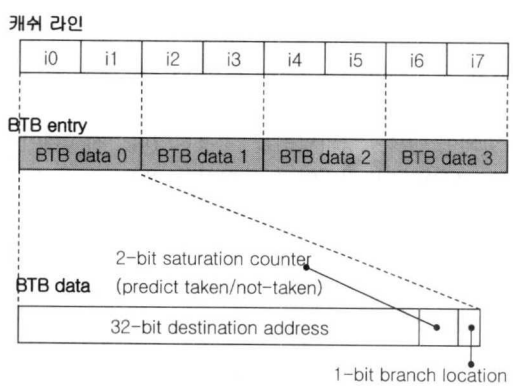

그림 4. BTB의 구조

페치를 수행할 때는 BTB뿐만 아니라, 명령어 TLB와 명령어 캐쉬가 읽혀진다. 페치 도중에 명령어 TLB나 명령어 캐쉬의 miss가 발생하는 경우에는 해당 스레드의 현재 사이클의 페치를 중단한다. 본 SMT 구조의 명령어 캐쉬는 non-blocking 캐쉬이므로 캐쉬 miss인 경우에는 pending cache miss 목록에 해당 캐쉬 라인의 주소가 삽입되고, TLB miss인 경우는 해당 스레드의 선행 명령어들이 모두 파이프라인에서 완료되기를 기다린 후, TLB miss 처리를 시작한다. 물론 캐쉬 miss나 TLB miss가 처리되는 동안에는 페치와 페치 스레드 선택과정에서 그 스레드는 제외된다.

특정 스레드의 페치가 중단될 수 있는 조건을 나열하면 다음과 같다.

- 명령어 캐쉬 혹은 TLB miss
- 명령어 페치 큐가 가득참
- 익셉션 발생
- 현 사이클에 선행 분기 명령어의 분기 예측 실패
- Unbuffered store 동작 발생

Ⅳ. 연구 방법

본 논문에서 제안된 SMT ARM 구조는 cycle-based, execution driven 시뮬레이터로 구현된다. 시뮬레이터는 C 언어를 사용하여 작성되었다.

시뮬레이터는 SMT ARM 구조 전체를 모델링하도록 설계되었지만, 특별히 명령어 페치에 관련된 부분에 대하여는 여러 가지 선택 사항을 두어, 이를 변화시키면서 전체 프로세서의 성능에 페치가 미치는 영향을 볼 수 있도록 하였다. 이 선택 사항은 명령어 페치 큐의 엔트리 수, 페치 스레드 선택 알고리즘, 명령어 캐쉬의 포트 수 등을 포함한다.

시뮬레이터에는 실제로 ARM 프로세서에서 수행될 수 있는 응용 프로그램을 입력으로 가한다. ARM 프로세서용의 응용 프로그램의 실행 파일을 얻기 위해서는 ARM Development Suit(ADS)[7]에 포함되어 있는 컴파일러를 사용하였다. 시뮬레이터에는 ADS에서 컴파일된 독립적인 응용 프로그램들을 여러 개 입력으로 가하여 다중 스레드에서의 SMT ARM 구조의 성능 평가를 실시하였다. 이 때 시뮬레이터에서 수행된 프로그램이 올바른 결과를 출력하는가를 검증하기 위하여 동일한 프로그램을 ADS의 일부분인 ARMulator에서도 수행하여 비교 대상으로 삼았다.

ADS에서 생성되는 실행 파일의 형식은 ELF이다. 본 시뮬레이터도 ELF 형식의 실행 파일에서 프로그램 명령어들을 뽑아내어 시뮬레이터의 메모리에 적재하는 loader 부분을 포함한다.

ADS는 별도의 OS가 존재하지 않고, ARMulator라는 프로세서 emulator에 software interrupt(SWI)를 처리하는 부분이 있어, 화면 출력, console 입력, 파일 입출력 등의 몇 가지 필수적인 OS 기능을 응용 프로그램에 제공하도록 되어 있다. 본 시뮬레이터도 같은 응용 프로그램을 사용해야 하므로 ARMulator와 호환이 되는 SWI 명령을 처리할 수 있도록 software interrupt routine을 작성하여 OS를 대신할 수 있도록 하였다.

성능 평가에 사용된 workload는 500개의 무작위 순서의 문자열을 순서대로 배열하는 sort 프로그램과 Dhrystone benchmark 프로그램, SPEC benchmark의 일부를 사용하였다. Sort는 insertion sort, quick sort, shell sort의 세 가지 sort 알고리즘을 500개의 엔트리에 반복 적용하는 프로그램이며, Dhrystone benchmark는 프로세서의 성능을 측정하도록 고안된 작은 명령어 sequence인 Dhrystone 유닛을 3000회 반복하는 프로그램이다.

본 연구에서는 페치 스레드 선택 알고리즘, 명령어 페치 큐의 엔트리 수, 사이클 당 최대로 페치할 수 있는 명령어 수(fetch bandwidth)가 프로세서 성능에 주는 영향을 분석하도록 한다. 성능 평가를 위한 기준으로는 사이클 당 페치된 명령어 수(Fetched Instruction Per Cycle)와 프로세서 전체의 성능을 나타내는 IPC(Issued Instructions Per Cycle)를 사용한다. 각 성능 값은 두 workload를 사용하여 나오는 결과의 평균치이다.

Ⅴ. 성능 평가

5.1 페치 bandwidth

그림 5는 사이클당 최대로 페치 가능한 명령어 수에 따른 프로세서 성능의 변화를 나타낸다. Fetch bandwidth는 명령어 캐쉬의 포트 수에 따른 성능의 변화로 보아도 무방하다. 명령어 페치 큐의 영향과 스레드 선택 알고리즘의 영향을 최소화한 상태에서 성능 수치를 산출하기 위하여 명령어 페치 큐의 엔트리 수는 64개로 fetch bandwidth에 비하여 크게 주었고, 스레드 선택 알고리즘은 가장 단순한 RR으로 고정하였다.

그림에서 나타나듯이 한 사이클에는 8개 ~ 16개의 명령어를 페치하는 것이 적당하다. 최대 페치량

을 2개 ~ 4개의 명령어로 제한하는 경우는 명령어 페치량이 이슈량을 따라가지 못하여 프로세서 전체의 성능을 저하시키게 된다. 이러한 현상은 2개 ~ 4개의 페치량을 가진 성능 결과에서 fetch rate와 IPC 수치가 거의 유사한 것으로 확인할 수 있다. 즉, 이슈 단계가 사이클 당 실행 유닛으로 명령어를 보낼 수 있는 능력에 비하여 페치된 명령어가 작으므로, 이슈 단계에서도 페치된 만큼의 명령어만을 실행 유닛에 줄 수 있게 되는 것이다.

페치량을 32개로 크게 하는 경우는 반대로 이슈 부분이 성능을 제한하게 된다. 페치를 이슈량보다 크게 하는 것은 결국 명령어 페치 큐와 이슈 큐 상에 이슈되지 못한 명령어가 쌓이는 것을 의미하고, 이렇게 큐 상에 존재하는 명령어들은 분기 예측 실패나, 예외상황, 인터럽트 등에 의하여 소거될 확률이 크게 된다. 이것은 그림의 BW16 ~ BW32에서 fetch rate와 IPC가 크게 차이나는 부분에서 확인할 수 있다.

위의 결과에서는 이슈량보다 높은 페치량을 가지는 BW 16의 경우가 가장 성능이 우수하지만, 16개의 명령어를 한 사이클에 페치하려면 명령어 캐쉬의 포트가 8개가 되어야 한다. 이러한 캐쉬의 설계는 물리적으로 상당한 부담이 되며 캐쉬의 동작주파수를 떨어뜨릴 위험이 있다.

위의 시뮬레이션 결과와 하드웨어적인 고찰을 종합하면 사이클 당 페치량은 프로세서의 최대 이슈 명령어 수와 같도록 설정하는 것이 자원 효율과 성능 면에서 유리하다.

5.2 명령어 페치 큐의 엔트리 수

그림 6은 명령어 페치 큐의 엔트리 수에 따른 성능 변화를 나타낸다. 여기서 사이클 당 페치량은 5.1절의 결과에 따라 8로 고정하였고, 스레드 선택 알고리즘은 RR으로 하였다.

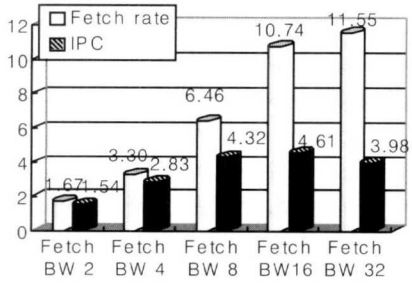

그림 5. Fetch Bandwidth에 따른 성능 변화

그림의 시뮬레이션 결과는 페치 큐의 엔트리 수가 최대 페치량보다 크기만 하면 일정한 성능을 보임을 나타낸다. 페치 유닛은 분기 예측에 따라 분기가 될 것으로 예상되는 명령이 페치된 경우에는 뒤의 명령의 페치를 중단하게 되고, 또한 해당 스레드의 예외상황이 발생하였을 경우에도 페치를 중단하게 된다. 따라서 매 사이클마다 실제로 페치되는 명령어는 최대 페치량에 못 미치게 된다. 실제로 5.1절의 결과에 따르면 사이클 당 최대 페치량이 8인 경우에 사이클 당 페치 되는 실제 명령어 수는 평균 6.46개에 머물러 있는 것을 볼 수 있다. 따라서 평균적인 명령어 페치량을 감당할 수 있는 페치 큐 엔트리 수는 8이 된다.

그러나 이슈 단계에서 종속성이 해결되지 않아 특정 스레드의 이슈가 중단되면, 결국 이슈 큐가 가득차게 되고, 디코딩이 중단되어 명령어 페치 큐에 존재하는 유효한 명령어 수가 최대 페치량 이상으로 증가할 수 있다. 따라서 페치 큐를 지나치게 작게 설정하였을 때에는 이러한 일시적인 명령어 수 증가를 감당할 수 없으므로 성능 저하 현상이 생기게 된다.

따라서 페치 큐의 엔트리 수는 이슈 중단으로 인한 페치 큐 상의 명령어 수 증가와 평균적인 실제 페치 명령어 수를 감안하였을 때, 사이클 당 최대 페치 가능한 명령어 수의 두 배로 하는 것이 타당하다.

5.3 페치 스레드 선택 알고리즘

그림 7은 페치 스레드 선택 알고리즘에 따른 성능 변화를 나타낸다. 여기서 사이클 당 최대 가능한 페치 수는 8로, 페치 큐의 엔트리 수는 64개로 고정하였다.

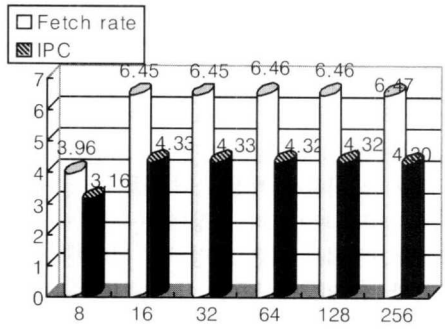

그림 6. Fetch Queue entry 수에 따른 성능 변화

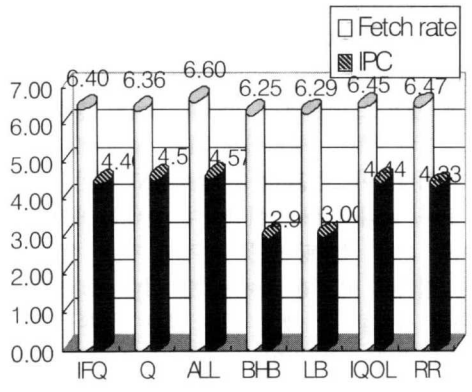

그림 7. 페치 스레드 선택 알고리즘에 따른 성능 변화

페치 스레드 선택 알고리즘은 앞서 기술한 바와 같이 파이프라인을 정지시킬 확률이 가장 적은 스레드에서 명령어를 우선적으로 페치하여 이슈율을 높이고 프로세서 전체의 IPC를 향상시키는 데에 있다. 그림에서 보는 바와 같이 페치 스레드 선택 알고리즘은 사이클 당 실제 페치된 명령어 수에는 별다른 영향을 주지 않는다. 선택된 스레드의 종류에 상관없이 페치 유닛은 최대 페치량 설정과 분기 예측에 따라 실제 페치량을 조절하므로 fetch rate는 스레드 선택 알고리즘에 무관하다.

그러나 프로세서 성능의 척도인 IPC는 ICOUNT_BHB와 ICOUNT_LB 알고리즘을 사용한 성능 평가에서 다른 경우와 비교하여 현저히 낮게 나타났다. 이들은 파이프라인 상에 존재하는 분기 명령의 수와 메모리 접근 명령의 수에 근거하여 페치 스레드를 선택하는 알고리즘이다. 그런데, 분기 명령이나 메모리 접근 명령의 수는 매 사이클마다 동적으로 변하는 것이 아니고, 일정 사이클 동안 일정한 수가 유지된다. 또한 경우에 따라서는 분기명령이나 메모리 접근 명령이 존재하지 않는 사이클이 상당기간 유지될 수 있다. 결과적으로 이 사이클 동안은 특정 스레드만이 계속 선택되는 결과를 낳는다. SMT 구조에서 부족한 ILP를 TLP로 보충하려면 파이프라인 상에 여러 스레드의 명령어가 고루 섞여 있어야 한다. 하지만 이렇게 특정 스레드들의 명령어 만이 계속 페치되면 파이프라인 상에 존재하는 명령어가 특정 스레드들에서 나온 명령어로 획일화되어 TLP를 이용할 수 없다. 이러한 이유로 ICOUNT_BHB와 ICOUNT_LB는 성능 향상에 도움을 주지 못한다.

ICOUNT_IFQ, ICOUNT_Q, ICOUNT_ALL은 각각 현재 프로세서의 페치 큐, 이슈 큐, 파이프라인 전체에 존재하는 명령어 수를 근거로 스레드를 선택한다. 즉, 각 큐에 존재하는 명령어 수를 스레드 별로 세어서 가장 작은 수의 명령어가 존재하는 스레드에 우선권을 준다. 따라서 앞선 사이클에서 우선권을 부여받았던 스레드들은 다음 사이클에는 큐에 많은 명령어가 존재하게 되고, 다음 사이클에는 우선 순위가 떨어지고 전 사이클에 캐쉬 포트를 할당받지 못한 스레드들의 우선 순위가 높아지게 된다. 이러한 방식으로 프로세서 안에 다양한 스레드로 부터의 명령어가 섞이게 되므로 TLP를 충분히 이용할 수 있게 된다. RR도 사이클 마다 우선순위를 rotate하므로 위의 장점을 그대로 사용할 수 있다. IQOL은 이슈 큐 상에서 이슈 순서가 가장 늦은 스레드를 우선적으로 페치하게 되는데, 이 또한 다양한 스레드의 명령어들을 사이클 별로 다양하게 섞어서 페치하는 결과를 보인다.

이러한 알고리즘들은 대부분이 프로세서에 존재하는 다양한 큐의 명령어 수를 세는 것으로서, 하드웨어로 구현하기에 그리 큰 부담이 되지는 않는다. 이 중에서도 RR이 가장 간단하지만 ICOUNT_IFQ, ICOUNT_Q, ICOUNT_ALL 등에 비하여 약간의 성능 저하가 있으므로, ICOUNT_BHB와 ICOUNT_LB를 제외한 다른 알고리즘 중에서 하드웨어 크기와 구현 가능성을 고려하여 회로 설계 단계에서 선택하여 줄 수 있다

Ⅵ. 결 론

SMT 프로세서는 기존의 수퍼스칼라 프로세서에 비하여 높은 실제 페치량을 가지며, 여러 스레드에서 동적으로 명령어를 선택하여 페치하므로 페치를 위한 하드웨어 자원의 양과 스레드 선택 알고리즘이 프로세서 전반의 성능에 큰 영향을 주게된다. 따라서 본 논문에서는 SMT 프로세서에서 명령어 페치 유닛의 구조에 관한 기준을 제시하였다.

사이클 당 페치 가능한 명령어 수는 최대 이슈 가능한 명령어 수와 동일하게 유지하는 것이 자원 효율과 성능 향상 두 가지를 모두 만족시킬 수 있다. 페치 명령어 수를 크게 하는 것은 약간의 성능 향상이 기대되지만, 이것은 결국 명령어 캐쉬의 포트 수 증가를 가져오므로 바람직하지 않다. 페치 큐의 엔트리 수는 사이클 당 최대 페치량 이상이 되어야 하며, 일시적인 페치 명령어의 누적 현상을 고려하면 최대 페치량의 두 배 정도가 적당할 것이다.

그러나, 그 이상의 엔트리는 페치 성능과 프로세서 전체 성능 향상에 도움을 주지 못한다.

위의 두 가지 요소는 위에 서술한 기준을 초과하는 하드웨어 자원에 대하여는 투입한 만큼의 효과를 거둘 수 없으므로, 명령어 캐쉬 포트와 페치 큐의 엔트리는 기준을 만족하는 최소치로 설정하는 것이 최적이다.

페치 스레드 선택 알고리즘은 각 스레드의 우선순위가 매 사이클 마다 동적으로 변화할 수 있는 것으로 채택한다. 여러 사이클에 걸쳐 일정한 우선순위가 유지되는 스레드 선택 알고리즘은 SMT에서 TLP 이용을 제한하므로 성능 저하를 야기할 수 있다.

참 고 문 헌

[1] ARM, *ARM Architecture Reference Manual, Part A CPU Architecture*, 1996

[2] Susan J. Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo, Rebecca L. Stamm, Dean M. Tullsen, "SIMULTANEOUS MULTITHREADING: A Platform for Next-Generation Processors", September/October 1997 *IEEE Micro*, p.p. 12~19

[3] D.M. Tullsen, S.J. Eggers, J.S. Emer, H.M. Levy, J.L. Lo, and R.L. Stamm, "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," *Proc. 23rd Annual International Symposium on Computer Architecture*, pp.191-202, May 1996.

[4] H. Hirata, K. Kimura, S. Nagamine, Y. Mochizuki, A. Nishimura, Y. Nakase, and T. Nishizawa, "An Elementary Processor Architecture with Simultaneous Instruction Issuing from Multiple Threads," *Proc. 19th Annual International Symposium on Computer Architecture*, pp.136-145, May 1992.

[5] *Clearwater Networks, Inc.* "Introducing the CNP810 Family of Network Services Processors", http://www.clearwaternetworks.com/clearwater_overview.pdf, June 2001

[6] Mike Johnson, *Superscalar Microprocessor Design*, pp.107-108, Prentice-Hall, Inc. 1991

[7] *ARM, ARM Developer Suit version 1.1, Compiler, Linker and Utilities Guide*, 1999

홍 인 표(In-Pyo Hong) 정회원

1999년 2월 : 연세대학교 전자공학과 졸업
2001년 2월 : 연세대학교 전기전자공학과 석사
2001년 3월 ~ 현재 : 연세대학교 전기전자공학과 박사과정

<주관심 분야> 마이크로프로세서, 네트워크프로세서

문 병 인 (Byung-In Moon) 정회원

1995년 2월 : 연세대학교 전자공학과 졸업
1997년 2월 : 연세대학교 전자공학과 석사
2002년 2월 : 연세대학교 전기전자공학과 박사
2002년 3월~현재 : 하이닉스 반도체
<주관심 분야> 마이크로프로세서, 네트워크프로세서

김 문 경(Moon-Gyung Kim) 정회원

1997년 2월 : 연세대학교 전자공학과 졸업
1999년 2월 : 연세대학교 전자공학과 석사
1999년 3월~현재 : 연세대학교 전기전자공학과 박사과정
<주관심 분야> 마이크로프로세서, 네트워크프로세서

이 용 석(Yong-Surk Lee) 정회원

1973년 2월 : 연세대학교 전기공학과 졸업
1977년 2월 : Univ. of Michigan, Ann Arbor 석사
1981년 2월 : Univ. of Michigan, Ann Arbor 박사
1993년 3월~현재 : 연세대학교 전기전자공학과 교수
<주관심 분야> 마이크로프로세서, 네트워크프로세서