논문 2008-45SD-12-10

해석적 모델을 이용한 분산된 리오더 버퍼 슈퍼스칼라 프로세서의 성능분석

(The Performance Analysis of Distributed Reorder Buffer in Superscalar Processor using Analytical Model)

윤 완 오*, 신 광 식*, 김 경 섭*, 이 윤 섭*, 최 상 방**

(Wan-oh Yoon, Kwang Sik Shin, Kyeong-seob Kim, Yun Sub Lee, and Sang-bang Choi)

요 약

슈퍼스칼라 프로세서에서 리오더 버퍼의 복잡도를 줄이는 여러 가지 방법이 제시 되었다. 그 중에서 리오더 버퍼의 포트를 가장 단순하게 하는 방법은 하나로 되어 있는 리오더 버퍼의 구조를 실행 유닛의 개수만큼 여러 개로 나누어 분산된 리오더 버퍼로 구현하는 것이다. 각각의 분산된 리오더 버퍼는 실행 유닛의 작업 부하에 따라 그 크기를 달리 할 수 있다. 하지만 분산된 리오더 버퍼의 크기에 따라 성능의 변화가 크다. 지금까지의 분산된 리오더 버퍼로 나누는 연구는 적절한 크기를 결정하기 위해 시뮬레이션 결과에 기반 하여 직관적으로 유추하였다. 본 논문은 분산된 리오더 버퍼에 M/M/1 큐잉 이론을 이용한 수학적 모델을 적용하여 최적의 크기를 결정하고 CPU2000 벤치마크 프로그램을 수행하여 성능을 측정하고 평가하였으며 기존 슈퍼스칼라 프로세서 성능의 99.2%를 보여주는 분산된 리오더 버퍼의 최적 크기를 정할 수 있었다. 기존의 리오더 버퍼와 본 논문에서 제시한 분산된 리오더 버퍼를 HDL로 구현하였을 때 포트에서 82%의 하드웨어 자원과 30%이상의 지연시간을 줄였다.

Abstract

There are several approaches for reducing the ROB(Reorder Buffer) complexity in processors. The one technique that makes the simplest ROB ports relies on a distributed implementation that spreads the centralized ROB structure across the functional units(FUs). Each distributed buffers are decided on the size of them by workload of the functional units. The performance of the processor depends on the size of distributed ROB. However, most of previous works have depended on the simulation results to decide the optimsize of distributed ROB. In this paper, we use an analytical model based on the M/M/1 Queuing theory to determine the optimum size of each distributed ROB. Our schemes are evaluated by using the simulation performed by the CPU2000 benchmarks. We are able to choose the optimum size of distributed ROB showing the 99.2% performance compared with existing superscalar processors. We can save 82% hardware resources in ports and reduce more than 30% of delay when ROB and distributed ROB proposed in this paper are designed by HDL.

Keywords: Reorder Buffer, Superscolor, Queuing theory, Processor, HDL

Ⅰ. 서 론

오늘날의 프로세서는 VLSI제조 기술의 진보와 컴퓨터 관련 기술의 발달로 급격한 성능향상을 이루었다. 최근 거의 모든 프로세서는 슈퍼스칼라 프로세서로 구현되고 있다. 슈퍼스칼라 프로세서는 프로세서의 성능을 높이기 위해 프로그램 내의 명령어 병렬성(ILP : Instruction Level Parallelism)을 사용하여 여러 명령어들을 동시에 수행한다. 프로그램 내의 병렬성을 높이기 위해 주로 리오더 버퍼(reorder buffer)와 명령어 윈도우를 사용하여 비순차(out-of-order)로 명령어를 수행한다. 또한 리오더 버퍼를 사용하면 거짓 데이터 의존성(false data dependency)을 없애기 위한 레지스터 리네이밍(register renaming)을 사용 할 수 있다. 프로세서는 리오더 버퍼를 이용해 비순차로 연산을 하고 순차(in-order)로 종료(commit) 하기 때문에 인터럽트나 분기 예측 실패 후에 정확한 상태에서 다시 시작할 수 있

* 학생회원, ** 평생회원 인하대학교 전자공학과
(Dept. of Electronic Engineering, Inha University)
접수일자: 2008년3월20일, 수정완료일: 2008년11월27일

다. 일반적으로 리오더 버퍼는 많은 포트를 갖는 레지스터 파일로 구성된다. 리오더 버퍼를 갖는 W-way 슈퍼스칼라 프로세서는 리오더 버퍼에서 다음과 같은 포트를 필요로 한다.

1. 매 클럭당 W개의 명령어가 동시에 디스패치/이슈(dispatch/ issue)될 때 리오더 버퍼로부터 소스 오퍼랜드의 값을 읽기 위해 적어도 2W개의 읽기 포트가 필요하다.

2. 매 클럭당 실행단에서 연산된 W개의 결과를 동시에 리오더 버퍼에 완료할 때 W개의 쓰기 포트가 필요하다.

이와 같이 리오더 버퍼에는 많은 입출력 포트가 존재하기 때문에 리오더 버퍼의 구조가 복잡해진다. 따라서 칩에서 차지하는 면적도 많아지고 전력 소모도 많아진다. 그리고 이를 이용하는 로직에 임계경로(critical path-delay)가 생겨 결국 프로세서의 클럭을 높일 수가 없게 된다. 이를 해결하기 위해 리오더 버퍼를 단순화 시키는 방법이 여러 가지 제시 되었다. 그 중에 한 가지 방법은 리오더 버퍼를 나누어 각각의 실행 유닛에 할당하는 방법이다. 즉 하나의 리오더 버퍼를 실행 유닛의 개수만큼 나눈 분산된 리오더 버퍼(DROB : Distributed ROB)를 각각의 실행 유닛에 할당하는 것이다. 이렇게 했을 때 각 실행 유닛의 부하에 따라 분산된 리오더 버퍼의 크기를 조절 할 수 있다. 즉 자주 사용하는 실행 유닛에는 분산된 리오더 버퍼의 크기를 크게 하고 반대의 경우에는 작게 할당할 수 있다. 또 평균 6%미만의 명령어가 디스패치시에 리오더 버퍼로부터 소스 오퍼랜드를 읽어 오기 때문에 이 포트도 제거함으로써 리오더 버퍼를 더 간단하게 할 수 있다[1~2]. 즉 하나의 분산된 리오더 버퍼에서 명령어 완료시 결과 값을 완료(writeback)할 때 필요한 1개의 쓰기 포트와 종료할 때 필요한 1개의 읽기 포트로 구성하게 된다. 결과적으로 미미한 성능 저하를 초래할 수 있지만 하드웨어 자원과 전력소모를 많이 줄일 수 있다. 리오더 버퍼를 분산된 리오더 버퍼로 나누는 목적은 리오더 버퍼의 복잡도를 낮추기 위해서이다. 하지만 분산된 리오더 버퍼의 크기에 따라 성능 변화가 매우 크기 때문에 분산된 리오더 버퍼의 적절한 크기를 결정하는 것은 매우 어렵고 중요하다. 그러나 지금까지의 분산된 리오더 버퍼로 나누는 연구는 적절한 크기를 결정하기 위해 시뮬레이션 결과에 기반 하여 직관적으로 유추하는 수준에 머물러 왔다. 이에 본 논문에서는 분산된 리오더 버퍼에 M/M/1 큐잉 이론을 이용한 수학적 모델을 적용하여 최적의 크기를 결정하는 것을 제안하고 제안한 모델을 벤치마크 프로그램을 수행하여 성능을 측정하였다. 기존의 리오더 버퍼와 본 논문에서 제시한 분산된 리오더 버퍼를 HDL로 구현하였을 때 포트에서 82%의 하드웨어 자원과 30%이상의 지연시간을 줄였다.

본 논문은 다음과 같은 구성으로 되어 있다. Ⅱ장에서는 분산된 리오더 버퍼의 구조를 소개한다. Ⅲ장에서는 본 논문에서 제시한 M/M/1 큐잉 모델을 설명하고 분산된 리오더 버퍼의 해석적 모델을 제시하며 M/M/1 큐잉 이론을 적용하여 분산된 리오더 버퍼의 최적화된 크기를 결정할 수 있는 수식을 설명한다. Ⅳ장에서는 리오더 버퍼와 본 논문에서 제시한 분산된 리오더 버퍼를 HDL(Hardware Description Language)로 구현하여 하드웨어 자원의 사용 비율과 지연시간을 비교하고 성능을 측정 하였다. 그리고 해석적 모델을 이용하여 얻은 분산된 리오더 버퍼의 크기로 시뮬레이션 하여 분산된 리오더 버퍼의 크기에 따른 성능의 변화를 비교해 본다. 마지막 Ⅴ장에서는 본 논문의 결론을 맺는다.

Ⅱ. 관련 연구

2.1 분산된 리오더 버퍼

명령어가 디코드 될 때 대부분의 소스 오퍼랜드는 레지스터 파일이나 바이패스 네트워크(bypass network)에서 가져오고 평균 6% 미만의 소스 오퍼랜드만이 리오더 버퍼로부터 읽혀온다[1]. 따라서 리오더 버퍼에서 소스 오퍼랜드를 읽기 위한 포트를 완전히 제거해도 전체 성능의 저하가 그리 크지 않게 된다. 리오더 버퍼에서 소스 오퍼랜드를 읽기 위한 포트를 완전히 제거한다는 것은 읽기 포트를 줄이는 것과는 근본적으로 다르다. 포트를 없애는 것은, 첫째 리오더 버퍼에서 임계경로를 제거 하는 것이 되고 두 번째로 리오더 버퍼 설계상에서 복잡도가 많이 줄어드는 것을 의미한다. 비록 리오더 버퍼에서 소스 오퍼랜드를 읽기 위한 포트가 없어졌다 해도 W-way 슈퍼스칼라 프로세서에서 실행 유닛에서 연산된 결과 값을 저장하는 W개의 쓰기 포트와 결과값을 레지스터나 메모리에 갱신할 때 필요한 W개의 읽기 포트는 제거 할 수 없다. 이처럼 리오더 버퍼를 각 실행 유닛의 개수만큼 나누어 분산된 리오더 버퍼를 만들면 포트의 수를 많이 줄일 수 있으며 리오더 버퍼는 다음과 같은 방법으로 더 간단해 질 수 있다.

방법 1. 각 분산된 리오더 버퍼는 실행 유닛의 부하에 따라 적절한 크기로 만들 수 있다. 즉, 연산을 많이

하는 덧셈, 뺄셈 실행 유닛에는 분산된 리오더 버퍼의 크기를 크게 할당하고 비교적 적은 연산을 하는 곱셈, 나눗셈 실행 유닛에는 분산된 리오더 버퍼의 크기를 작게 할당한다.

방법 2. 각 실행 유닛은 한 싸이클에 한 개의 결과만 저장을 하므로 분산된 리오더 버퍼에는 한 개의 쓰기 포트만으로 충분하다. 즉, 하나의 리오더 버퍼를 사용할 때처럼 여러 개의 결과를 동시에 쓰지 않기 때문에 이를 중재하는 추가 로직이나 실행 유닛에서 연산된 결과를 잠시 저장하는 버퍼가 필요하지 않다.

방법 3. 종료시에 분산된 리오더 버퍼에 있는 결과를 종료하여 레지스터를 갱신할 때 한 개의 읽기 포트만 필요하다. 정수 덧셈, 뺄셈 실행유닛을 예로 들면, 이 실행 유닛에 할당된 분산된 리오더 버퍼들이 명령어 결과값을 완료할 때는 단순히 라운드 로빈 방식으로 값을 쓰기 때문에 종료할 때 동시에 같은 분산된 리오더 버퍼에서 연속된 두 개의 값들을 종료할 가능성은 극히 희박하다. 따라서 종료를 위한 하나의 읽기 포트만으로 충분하다. 하지만 곱셈, 나눗셈 실행 유닛에서는 드물지만 하나의 분산된 리오더 버퍼에서 연속된 값을 동시에 종료해야 할 경우 최소 두 클럭이 필요하다.

2.2 분산된 리오더 버퍼의 구조

그림 1은 각 실행 유닛에 하나의 분산된 리오더 버퍼를 할당한 블록 다이어그램이다. 그림 1에서 리오더 버퍼는 분산된 리오더 버퍼의 포인터를 갖고 있어서 분산된 리오더 버퍼에 있는 연산 결과들이 프로그램 순서대로 종료될 수 있도록 해 주고 예외상황이나 인터럽트의 발생시 프로그램이 다시 실행될 수 있게 해 준다[6]. 그림 1의 분산된 구조에서 리오더 버퍼는 W-way 슈퍼스칼라 프로세서에서 명령어 디스패치 시의 프로그램 순서대로 분산된 리오더 버퍼의 포인터를 만드는데 필요한 W개의 쓰기 포트와 분산된 리오더 버퍼의 연산결과를 종료하여 레지스터에 갱신할 때 필요한 포인터를 가져올 W개의 읽기 포트만으로 충분하다. 또는 1개의 읽기, 쓰기 포트로 구성하기 위해서 그림 1의 리오더 버퍼는 한번에 W개의 포인터를 저장할 수 있는 크기를 가지면 된다. 그림 1의 리오더 버퍼에서 각 포인터는 실행 유닛의 이름과 분산된 리오더 버퍼의 주소를 저장하게 된다. 명령어가 종료되는 경우 분산된 리오더 버퍼의 주소를 읽고 그 주소에 해당하는 값을 레지스터에 쓴다.

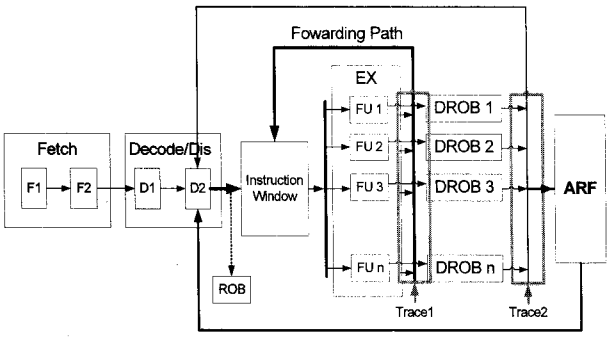


그림 1. 분산된 리오더 버퍼를 갖는 슈퍼스칼라 프로세서의 구조

Fig. 1. Superscalar data-path with distributed reorder buffer.

그림 1에서 분산된 리오더 버퍼는 실행 유닛의 연산결과를 쓰기 위한 하나의 쓰기 포트와 분산된 리오더 버퍼의 결과값을 종료하여 레지스터를 갱신할 때 필요한 읽기 포트로 구성되어 있다. 분산된 리오더 버퍼가 가득 차지 않는 이상 실행 유닛에서 연산된 결과 값은 바로 분산된 리오더 버퍼로 완료된다. 한 클럭에 분산된 리오더 버퍼의 개수만큼 동시에 완료될 수 있다. 그림 1의 구조에서는 많은 결과 값이 한 클럭에 생성 된다고 해도 동시에 최대 W개 밖에 완료 할 수 없다. 명령어를 디스패치 할 때 만약 소스 오퍼랜드가 분산된 리오더 버퍼에 있고 분산된 리오더 버퍼가 종료하지 않는다면 읽기 포트를 이용해 명령어 수행에 필요한 소스 오퍼랜드를 읽어 온다. 하지만 분산된 리오더 버퍼가 종료를 하고 있다면 명령어를 디스패치 하지 못하고 기다려야 한다. 만약 디스패치 시에 필요한 소스 오퍼랜드까지 연속으로 종료하면 레지스터에서 그 값을 읽어 오게 된다. 즉, 종료하는 작업이 디스패치 시에 소스 오퍼랜드를 읽는 작업보다 우선시 된다. 왜냐하면 분산된 리오더 버퍼는 여러 개 이므로 하나의 분산된 리오더 버퍼가 계속해서 종료할 확률은 매우 작기 때문이다. 여러 개의 덧셈, 뺄셈 실행 유닛에서 라운드 로빈 방식으로 실행하고 그 결과값을 각 분산된 리오더 버퍼에 종료하므로 하나의 분산된 리오더 버퍼가 연속해서 완료할 확률은 더욱 더 낮아진다. 그리고 소스 오퍼랜드를 리오더 버퍼에서 읽어올 확률이 6%이내 이므로 종료 하는 것 때문에 디스패치를 못하는 경우는 거의 희박하다. 따라서 본 논문에서는 해석적 모델을 이용하여 분산된 리오더 버퍼의 크기를 최적화 하는 것을 제안 하였으며 HDL을 이용해 96개의 엔트리를 갖는 리오더 버퍼와 본 논문에서 제시한 분산된 리오더 버퍼를 구현하여 성능 비교를 하였다.

Ⅲ. 분산된 리오더 버퍼의 해석적 모델

3.1 M/M/1 큐

M/M/1 큐잉 모델은 많은 분야에서 시스템의 성능을 예측하는데 자주 사용되는 모델링 방법이다. 그림 2와 같은 하나의 서버가 대기 큐(waiting queue)에 있는 입력을 처리하는 M/M/1 큐잉 모델을 가정한다. 여기서 입력 시간 간격(customer interarrival)과 서비스 타임은 지수 분포(exponentially distributed)이어야 한다[9]. 그리고 큐는 무한히 크다고 가정한다. 즉, 큐로 들어오는 모든 입력은 서비스 될 때 까지 대기한다. λ는 일정한 상수값을 갖는 생성계수로써 평균 도착율(average arrival rate)을 나타내고, μ는 일정한 상수값을 갖는 소멸계수로써 평균 서비스율(average service rate)를 나타낸다.

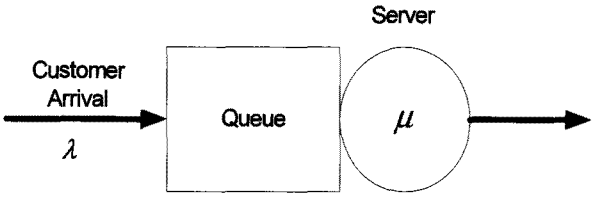


그림 2. M/M/1 큐
Fig. 2. M/M/1 Queue.

다음은 포아송 도착 과정(Poisson arrival process)을 갖는 시스템의 특성을 보여준다[9].

1. 일정 시간 사이에 발생한 사건의 수는 다른 시간대의 발생한 사건의 수와 독립이다. 즉, 포아송 과정은 무기억(memoryless)이다.

2. 매우 짧은 시간에 하나의 사건이 발생할 확률은 시간에 비례한다.

3. 짧은 시간동안 한개 이상의 사건이 발생할 확률은 무시할 수 있다.

3.2 분산된 리오더 버퍼의 해석적 모델

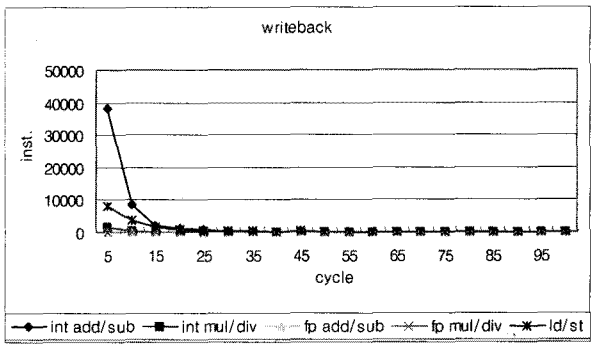
하나의 분산된 리오더 버퍼를 놓고 볼 때 분산된 리오더 버퍼는 하나의 쓰기 포트와 하나의 읽기 포트를 갖고 있다. 여기서 예외상황과 인터럽트가 발생하지 않고 완벽한 분기 예측기(branch predictor)를 갖고 있다고 가정하면, 디스패치할 때 리오더 버퍼의 위치가 정해지고 연산된 결과는 디스패치할 때의 순서대로 종료하므로 분산된 리오더 버퍼는 FIFO(First In First Out) 큐로 간주될 수 있다. 분산된 리오더 버퍼에는 하나의 읽기 포트와 하나의 쓰기 포트가 있고 매우 짧은 시간을 클럭의 주기라고 한다면 다음과 같은 정의를 얻을 수 있다.

정의 1. 매 클럭당 분산된 리오더 버퍼에 값을 쓰거나 읽는 것은 이전의 읽기 쓰기와는 독립이다.

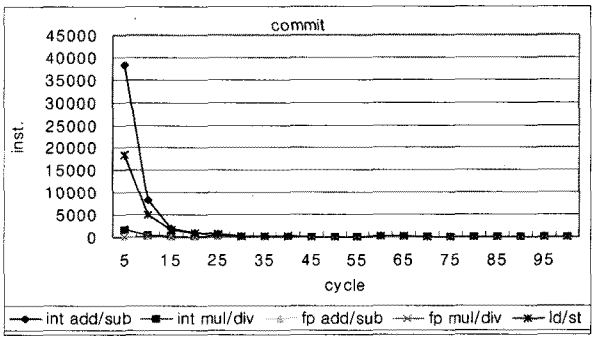
정의 2. 매 클럭당 한 번의 읽기나 쓰기를 할 확률은 클럭의 수에 비례한다.

정의 3. 분산된 리오더 버퍼에서 한 클럭에 최대 한 번의 읽기와 쓰기 밖에 할 수 없다.

따라서 분산된 리오더 버퍼에 연산 결과 값을 읽고 쓰는 것은 포아송 도착 과정이고 M/M/1 큐잉시스템으로 모델링 할 수 있다. 그림 3은 분산된 리오더 버퍼의 쓰기 포트와 읽기 포트에서 측정한 단위 시간당 명령어를 읽고 쓰는 명령어의 개수를 보여준다. 정수 연산 프로그램인 bzip2를 사용했기 때문에 비교적 정수 덧셈, 뺄셈과 메모리 접근에서 포아송 도착 과정에 가까운 것을 볼 수 있다. 가로축은 클럭 수를 나타내고 세로축은 명령어 결과 값의 개수를 나타낸다. 그림 3의 (a)는 명령어 완료시에 분산된 리오더 버퍼의 쓰기 포트를 이용



(a) 명령어 완료시 분포
(a) distribution of instruction writeback



(b) 명령어 종료시 분포
(b) distribution of instruction commit

그림 3 분산된 리오더 버퍼에서 명령어 분포
Fig. 3. Distribution of instructions in DROB.

해 명령어 결과 값을 쓰는 M/M/1 큐에서 도착율의 분포를 나타내고 그림 3의 (b)는 분산된 리오더 버퍼에 있는 결과 값을 레지스터로 갱신하는 서비스율의 분포이다. 단위 클럭당 실행 유닛에서 연산을 마치고 분산된 리오더 버퍼에 완료하거나 분산된 리오더 버퍼에서 종료되어 레지스터나 메모리를 갱신하는 것은 M/M/1 큐잉 시스템에서 그림 2의 도착과정과 서비스로 볼 수 있다.

그리고 분산된 리오더 버퍼에서 한 클럭에 최대 한 개의 명령어 결과를 레지스터에 종료하기 때문에 하나의 서버를 갖는다고 볼 수 있다.

단위 시간(클럭)당 실행 유닛에서 분산된 리오더 버퍼로 값이 완료되는 평균 완료율(λ)과 분산된 리오더 버퍼에서 레지스터로 종료되는 평균 종료율(μ)을 안다면 분산된 리오더 버퍼에서 서비스율(ρ)과 분산된 리오더 버퍼에 있는 값들의 평균 개수(N_s)를 알 수 있다[9].

$$\rho = \frac{\lambda}{\mu} \tag{1}$$

$$N_s = \frac{\rho}{1-\rho} \tag{2}$$

식(2)에서 N_s는 분산된 리오더 버퍼에서 종료되기를 기다리는 값들의 평균 개수이다. 즉, 분산된 리오더 버퍼의 크기는 최소한 N_s보다 커야 분산된 리오더 버퍼가 가득차지 않을 수 있고 성능저하가 최소화 될 수 있다. 분산된 리오더 버퍼의 크기를 크게 할수록 성능 저하는 줄어들게 된다. 하지만 하드웨어 비용은 늘어나게 된다. 성능 저하는 거의 없으면서 분산된 리오더 버퍼의 크기는 최소가 되도록 정해야 한다. 따라서 분산된 리오더 버퍼에 써진 값들의 개수(N_s)가 분산된 리오더 버퍼의 크기(x)보다 크거나 같을 확률은 수식(3)과 같다[10].

$$P\{N_s \geq x\} = \rho^x \tag{3}$$

분산된 리오더 버퍼가 가득찰 확률을 10% 이내로 하기 위해서는 분산된 리오더 버퍼의 크기가 다음과 같은 값을 가져야 한다.

$$\rho^x \leq 0.1$$

$$x > \frac{\log(10^{-1})}{\log(\rho)} = 1/\log(\frac{1}{\rho}) \tag{4}$$

분산된 리오더 버퍼의 최적크기를 결정하는데 필요한 서비스율(ρ)을 보다 객관적인 수치로 나타내기 위해 여러 가지 응용 프로그램을 수행하고 이것을 추적하여 분산된 리오더 버퍼에서 실험적으로 평균 완료율(λ)과 종료율(μ)을 구한다. (4)번 수식을 이용하여 분산된 리오더 버퍼의 크기를 구하고 여기서 구한 분산된 리오더 버퍼의 크기로 시뮬레이션 하여 기존의 하드웨어와 성능을 비교하고 전체 분산된 리오더 버퍼의 크기에 따른 성능 변화를 확인하였다. 리오더 버퍼와 분산된 리오더 버퍼를 HDL로 구현할 때 리오더 버퍼의 지연시간이 더 길게 되므로 리오더 버퍼의 접근 시간을 2클럭, 분산된 리오더 버퍼의 접근시간을 1클럭으로 설정 하였다.

IV. 시뮬레이션과 성능분석

4.1 시뮬레이션 환경

프로세서의 성능을 테스트하기 위해 널리 사용되는 Simplescalar simulator를 이용하였다[11]. 표 1에 제시한 하드웨어의 디스패치, 이슈로직, 완료 그리고 종료단계를 그림 1의 동작을 하도록 설정 하였다. 자일링스(Xilinx)사의 ISE 7.1 버전을 이용하고 Vertex4 lx200 FPGA를 타겟으로 하여 리오더 버퍼와 분산된 리오더 버퍼를 구현하고 이때 사용된 게이트 수와 최대 지연시간을 조사 하였다. 그리고 여기에서 조사한 지연시간에 기반 하여 리오더 버퍼의 접근 시간을 2클럭, 분산된 리오더 버퍼는 1클럭으로 설정하고 각각의 성능을 테스트 하였다. CPU2000에서 사용하는 테스트 코드를 수행하여 성능을 측정 하였다. 실험에서 8-2-4-2-18로 표기된 것은 정수 덧셈 뺄셈기, 정수 곱셈 나눗셈기, 실수 덧셈 뺄셈기, 실수 곱셈 나눗셈기, 메모리 읽기쓰기 실행 유닛에 할당된 각 분산된 리오더 버퍼의 크기를 나타낸다. 4개의 정수연산 코드(bzip2, gcc, parser, vpr)와 4개의 실수연산 코드(wupwise, art, equake, mesa)를 이용하였다. 시뮬레이션 환경은 표 2와 같다. 분산된 리오더 버퍼에 읽기 포트가 2개인 경우(디스패치시 분산된 리오더 버퍼 내에 있는 소스 오퍼랜드를 읽기 위한 포트와 결과값을 종료할 때 사용되는 읽기 포트)와 본 논문에서 제시한 1개일 경우(읽기 포트 하나로 소스 오퍼랜드 읽을 때와 종료할 때 모두 사용된 경우)와 성능 차이를 비교하였다. 그리고 M/M/1큐가 안정된 상태에서 값을 측정하기 위하여 5억 개의 명령어를 실행한 뒤 2억 개의 명령어를 측정하여 실제 분산된 리오더 버

표 1. 슈퍼스칼라 프로세서의 시뮬레이터 구성
Table 1. Features of superscalar processor simulator.

특 징		설 명
명령어 패치		한 클럭에 최대 4개 패치
명령어 이슈		한 클럭에 최대 4개 이슈
명령어 종료		한 클럭에 최대 4개 종료
명령어 윈도우		32개의 엔트리
ALU	정수 덧셈 뺄셈 유닛	4개 수행시간 : 1클럭
	정수 곱셈 나눗셈 유닛	1개 곱셈 수행시간 : 3클럭 나눗셈 수행시간 : 20클럭
	실수 덧셈 뺄셈 유닛	4개 수행시간 : 2클럭
	실수 곱셈 나눗셈 유닛	1개 곱셈 수행시간 : 4클럭 나눗셈 수행시간 : 24클럭
	메모리 읽기 쓰기 유닛	2개 수행시간 : 2클럭
리오더 버퍼		96개의 엔트리

퍼에 있는 값들의 평균 개수와 분산된 리오더 버퍼의 크기를 결정하기 위한 ρ의 값을 구하였다. 여기서 구한 ρ의 값을 식(4)에 대입해 분산된 리오더 버퍼의 최적 크기를 결정하였다. 결정된 값을 분산된 리오더 버퍼에서 평균 존재하는 값들의 개수와 비교하였으며 마지막으로 분산된 리오더 버퍼의 크기에 따른 성능의 변화를 확인 하였다.

4.2 시뮬레이션 분석

첫 번째 시뮬레이션은 분산된 리오더 버퍼의 읽기 포트가 2개인 경우와 본 논문에서 제시한 1개일 경우의 성능의 차이를 비교하였다. 읽기 포트가 2개인 경우 한 개는 분산된 리오더 버퍼에 있는 결과 값을 종료하기 위하여 사용되고 나머지는 디스패치시에 소스 오퍼랜드를 읽기 위해 사용된다.

표 2. 시뮬레이션 환경
Table 2. Simulation Environment.

특 징			설 명
Simplescalar 버전			3.0
SPEC CPU 2000 버전			1.0
SPEC CPU 2000	CINT 2000	bzip2	압축 프로그램
		gcc	C 언어 컴파일러
		parser	워드 프로세서
		vpr	FPGA 회로 Placement & Routing
	CFP 2000	wupwise	양자 크로모 역학
		art	영상 인식
		equake	지진파 전파 시뮬레이션
		mesa	3D 그래픽
시뮬레이션 OS			Linux Kernel Ver 2.4.20-8smp
시뮬레이션 H/W			Intel Pentium 4 CPU 2.80GHz Main Memory 512 Mbyte

리오더 버퍼와 분산된 리오더 버퍼를 HDL로 구현하여 합성결과를 비교하고 여기서 구한 최대 지연 시간을 참고하여 리오더 버퍼와 분산된 리오더 버퍼의 접근 시간을 다르게 설정하고 벤치마크 프로그램을 실행하여 성능을 비교 하였다.

두 번째 시뮬레이션은 벤치마크 프로그램을 실행하여 분산된 리오더 버퍼에 있는 연산 결과 값들 개수의 평균과 최대값을 기록하고 그림 1의 Trace1에서 평균 완료율(λ)과 Trace2에서 평균 종료율(μ)을 구한다. 평균 완료율(λ)과 종료율(μ)을 이용하여 얻은 값(ρ)을 식(4)에 대입하면 각 분산된 리오더 버퍼의 크기 x값을 구할 수 있다. 그리고 그렇게 결정한 값과 분산된 리오더 버퍼에서 평균 존재하는 값들의 개수와 비교해 본다. 마지막으로 분산된 리오더 버퍼의 크기에 따른 성능의 변화를 확인하였다.

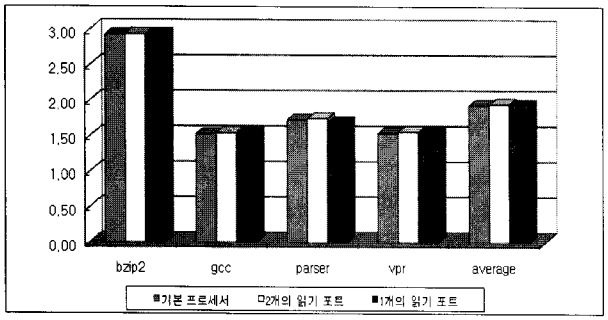
4.2.1 분산된 리오더 버퍼에서 읽기 포트의 수와 성능과의 관계

분산된 리오더 버퍼에서 읽기 포트의 수가 1개 일 때와 두 개일 때의 성능을 비교한다. 리오더 버퍼와 1개의 읽기 포트를 갖는 분산된 리오더 버퍼를 HDL로 구현하여 Vertex4 lx200 칩을 타겟으로 합성한 결과 포트 구성에서 하드웨어 자원 사용 비율에 많은 차이가 생겼으며 지연시간 또한 큰 차이를 보여주었다.

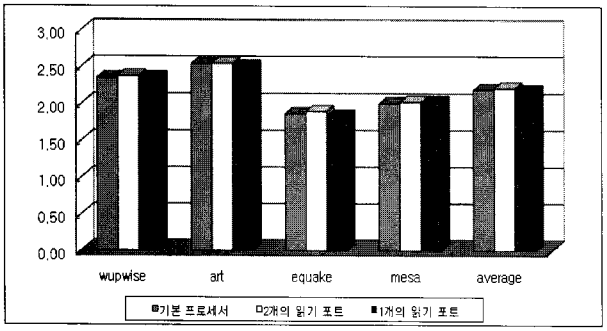
표 3. HDL 합성 결과
Table 3. HDL Synthesis Report.

리소스	DROB	ROB
LUT2	32	183
LUT3	8	219
LUT3_L	1664	19200
LUT4	74	9762
MUXF5	704	9216
MUXF6	256	4608
MUXF7	64	2304
MUXF8	0	1152
Slices	2686(3%)	15442(17%)
Slice Flip Flops	3456(1%)	3456(1%)
4 input LUTs	1778(0%)	29364(16%)
bonded IOBs	876(90%)	641(66%)
GCLKs	12(37%)	1(3%)
Minimum period	2.681ns	3.809ns
Maximum Frequency	372.995MHz	262.536MHz

표 3(괄호 안의 값은 타겟 칩에서 자원의 사용 비율이다.)의 합성 결과를 보면 전체적으로 리오더 버퍼가 분산된 리오더 버퍼보다 하드웨어 자원을 훨씬 더 많이 사용하고 지연 시간 또한 30%이상 더 길어진 것을 확인할 수 있다. 표 3의 값을 참고 하여 시뮬레이션에서 리오더 버퍼의 값을 읽거나 쓰는데 2클럭이 소요되고 분산된 리오더 버퍼에서는 1클럭이 소요된다고 설정하고 성능을 비교 하였다. 그림 4는 벤치마크 프로그램을 시뮬레이션 하였을 때의 성능을 IPC(Instruction Per Cycle)로 나타낸 것이다. 각 그래프에서 x축은 벤치마크 프로그램을 나타내고 y축은 IPC를 나타낸다. 여기서 표 1에 사용된 프로세서를 "기본 프로세서"라고 하고 "1개의 읽기 포트"와 "2개의 읽기 포트"는 분산된 리오더 버퍼에서 읽기 포트가 각각 1개와 2개인 경우이다. 그리고 분산된 리오더 버퍼의 크기는 8-4-6-4-16의 크기를 갖는다. 기본 프로세서의 리오더 버퍼의 크기가 96엔트리 이고 분산된 리오더 버퍼도 96개의 엔트리를 갖는다. 분산된 리오더 버퍼의 총 크기나 기본 프로세서의 리오더 버퍼의 크기는 같지만 분산된 리오더 버퍼에는 제어로직이나 포트가 많이 줄었기 때문에 표 4에서 보듯이 회로는 많이 간단해지고 성능은 약 0.5% 저하된 것을 알 수 있다.



(a) 정수 연산 벤치마크
(a) Fixed point benchmark



(b) 실수 연산 벤치마크
(b) Float point benchmark

그림 4. 분산된 리오더 버퍼에서 읽기 포트와 성능과의 관계
Fig. 4. Performance comparison of the DROB's port configuration.

그림 4에서 parser와 equake 그리고 art에서 1개의 읽기 포트를 가진 프로세서의 성능이 기본프로세서 보다 조금 떨어지는 것을 확인할 수 있다. 이것은 여기서 사용된 많은 명령어들이 소스 오퍼랜드를 분산된 리오더 버퍼에서 읽어 왔기 때문이다.

4.2.2 분산된 리오더 버퍼의 크기와 성능과의 관계

각 분산된 리오더 버퍼가 매우 크다고 가정하고 4개의 정수연산 코드(bzip2, gcc, parser, vpr)와 4개의 실수연산 코드(wupwise, art, equake, mesa)를 이용하여 분산된 리오더 버퍼에 있는 연산 결과값들 개수의 평균과 최대값을 기록하고 그림1의 Trace1에서 평균 완료율(λ)과 Trace2에서 평균 종료율(μ)을 구한다. 평균 완료율(λ)과 종료율(μ)을 이용하여 얻은 값(ρ)을 식(4)에 대입하면 각 분산된 리오더 버퍼의 크기 x값을 구할 수 있다. 구한 결과 값은 표 4와 같다. x값을 이용해 2의 배수에 가장 가까운 값으로 실험 하였다. 실행 유닛에 할당되는 각 분산된 리오더 버퍼의 크기가

표 4. 각 분산된 리오더 버퍼에서 사용된 레지스터 개수의 평균, 최대값 그리고 계산값.
Table 4. Maximum, Average and Calculated Number of Entries Used within Each DROB(contin.)
(a) 정수 실행 유닛의 분산된 리오더 버퍼
(a) Fixed point FU's DROB

	Int ADD/SUB				Int MUL/DIV			
	ρ	N_s	max	ave	ρ	N_s	max	ave
bzip2	0.88	7.15	18	10.3	0.00	0	0	0
gcc	0.67	2.02	20	2.78	0.42	0.72	5	0.05
parser	0.77	3.27	13	4	0.62	1.64	3	0.25
vpr	0.69	2.23	17	2.75	0.56	1.26	6	0.24
wupwise	0.84	5.23	12	5.25	0.00	0	0	0
art	0.78	3.5	11	4.51	0.00	0	0	0
equake	0.81	4.38	12	5.91	0.00	0	4	0.06
mesa	0.67	2.06	11	2.94	0.65	1.83	9	0.49
average	0.76	3.73	14.25	4.81	0.28	0.68	3.38	0.14
$x=1/\log\frac{1}{\rho}$	8.53				1.81			

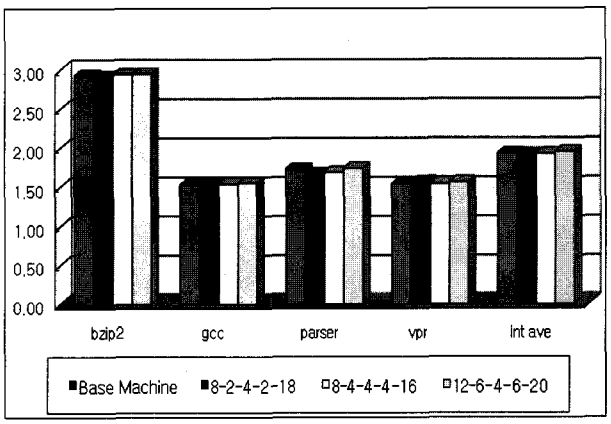
(b) 실수 실행 유닛의 분산된 리오더 버퍼
(b) Float point FU's DROB

	FP ADD/SUB				FP MUL/DIV			
	ρ	N_s	max	ave	ρ	N_s	max	ave
bzip2	0.00	0	0	0	0.00	0	0	0
gcc	0.44	0.8	1	0	0.00	0	2	0
parser	0.00	0	0	0	0.00	0	0	0
vpr	0.66	1.9	3	0.1	0.53	1.12	10	0.6
wupwise	0.62	1.6	3	0.58	0.65	1.84	4	1.18
art	0.74	2.83	8	0.49	0.67	2.01	16	1.38
equake	0.56	1.26	0	0.57	0.00	0	0	0
mesa	0.75	2.97	6	0.75	0.67	2.06	11	1.1
average	0.47	1.42	2.63	0.31	0.31	0.88	5.38	0.53
$x=1/\log\frac{1}{\rho}$	3.05				1.99			

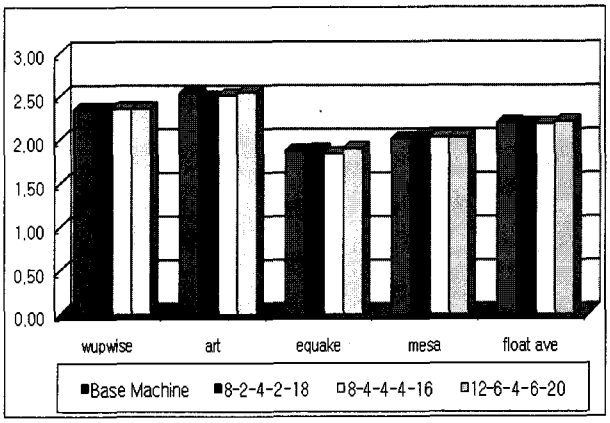
(c) 메모리 접근 실행 유닛의 분산된 리오더 버퍼
(c) Load and Store Unit's DROB

	LOAD			
	ρ	N_s	max	ave
bzip2	0.89	8.14	27	9.93
gcc	0.83	5.02	30	5.4
parser	0.90	9.16	29	10.2
vpr	0.87	6.84	28	7.95
wupwise	0.93	12.6	22	5.75
art	0.88	7.31	26	8.31
equake	0.85	5.84	24	6.88
mesa	0.88	7.1	27	7.6
average	0.88	7.94	26.63	7.75
$x=1/\log\frac{1}{\rho}$	17.9			

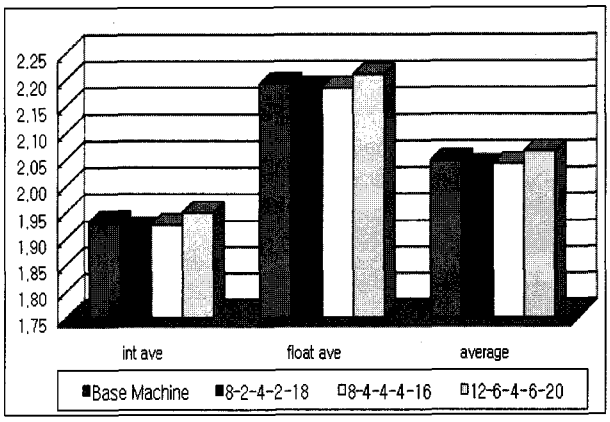
8-2-4-2-16, 8-4-4-4-16, 12-6-4-6-20일 때 각 시스템의 성능 IPC를 기본 슈퍼스칼라 프로세서와 비교한 결과는 그림 5와 같다. 여기서 리오더 버퍼의 읽기 쓰기 접근은 2클럭이 소요되고 분산된 리오더 버퍼는 1클럭이 소요 된다고 설정하고 시뮬레이션 하였다. 분산된 리오더 버퍼의 값이 커질수록 성능이 좋아 지지만 어느 정도 이상이 되면 성능 향상에 크게 도움을 주지 못한다. 따라서 하드웨어도 줄이고 성능저하도 거의 없는 적절한 분산된 리오더 버퍼의 크기를 결정해야 한다.



(a) 정수 연산 벤치마크
(a) Fixed point benchmark



(b) 실수 연산 벤치마크
(b) Float point benchmark



(c) 벤치마크 평균
(c) Average Benchmark

그림 5. 분산된 리오더 버퍼의 크기와 성능(IPC)
Fig. 5. IPCs of several configurations of DROBs.

실험에서 보듯이 각 분산된 리오더 버퍼크기는 8-2-4-2-18일 때 성능과 하드웨어 자원을 효율적으로 할 수 있다.

표 4의 (a)에서 bzip2나 parser에서 정수 덧셈 뺄셈 유닛에 할당된 분산된 리오더 버퍼의 사용률이 높다. 그 결

과 그림 5의 (a)에서 처럼 bzip2와 parser에서 성능의 저하가 생긴다. 표 4의 (b)에서 art의 경우도 마찬가지 이다. 이것은 분산된 리오더 버퍼의 크기가 작기 때문이 아니라 소스 오퍼랜드를 분산된 리오더 버퍼에서 읽어 오기 때문이다. 이 세 가지 벤치마크를 제외한 나머지 벤치마크 프로그램에서는 기본 수퍼 스칼라 프로세서와 비슷한 성능을 보여 주고 있다.

그림 5의 분산된 리오더 버퍼의 크기가 12-4-6-4-20일 경우에서 보듯이 리오더 버퍼의 크기가 클수록 성능이 좋아진다. 왜냐하면 종료하지 못한 명령어가 있어도 계속해서 명령어를 수행할 수 있기 때문이다. 하지만 리오더 버퍼의 크기와 성능이 정확히 비례하는 것은 아니다. 여기에 여러 가지 제약조건이 있기 때문이다. 예를 들면 명령어 윈도우의 크기가 충분하지 않다면 서로 독립된 명령어를 많이 찾을 수 없으므로 명령어간의 병렬성이 떨어져 최적의 성능을 내지 못한다. 본 논문에서의 실험은 이러한 명령어 윈도우의 크기를 32 엔트리로 정해 놓았다. 그림 5의 (c)에서 보듯이 분산된 리오더 버퍼에 하나의 읽기 포트만 갖고 그 크기가 8-2-4-2-18인 경우 기본 프로세서에 비해 0.8%의 성능 저하를 보여 주고 있다. 하지만 기본 프로세서의 리오더 버퍼는 96개의 엔트리를 갖는 반면에 분산된 리오더 버퍼는 총 88개의 엔트리를 갖고 있다. 단순히 리오더 버퍼의 크기만 줄어든 것이 아니라 리오더 버퍼에 연결되는 포트 또한 줄었기 때문에 표 3의 결과와 같이 많은 하드웨어 자원이 줄어든 것을 알 수 있다. 하드웨어 자원을 줄였다는 것은 전력소모 또한 줄일 수 있다는 것을 의미하며 지연시간 또한 줄어들어서 프로세서의 동작 속도를 더 빠르게 할 수 있다.

V. 결 론

일반적으로 리오더 버퍼는 많은 포트를 가진 레지스터 파일로 구성된다. W-way 슈퍼스칼라인 경우 리오더 버퍼에는 적어도 3W개의 읽기 포트와 W개의 쓰기 포트가 필요하다.

리오더 버퍼에 많은 입출력 포트가 존재하기 때문에 리오더 버퍼구조가 복잡해진다. 따라서 칩에서 차지하는 면적도 많아지고 전력 소모도 많아진다. 그리고 이를 이용하는 로직에 임계경로가 생겨 결국 프로세서의 성능이 저하 된다. 이를 해결하기 위해 본 논문에서는 하나의 리오더 버퍼를 실행 유닛의 개수만큼 나눈 분산된 리오더 버퍼를 각각의 실행 유닛에 할당 하고 매우 적은 수의 명령어가 디스패치시에 리오더 버퍼로부터 소스 오퍼랜드를 읽어 오기 때문에 이 포트도 제거함으로써 리오더 버퍼를 더 간단하게 하였다. 하나의 분산된 리오더 버퍼에서 명령어 완료시 결과 값을 완료할 때 필요한 1개의 쓰기 포트와 종료할 때 필요한 1개의 읽기 포트로 구성하게 된다. 분산된 리오더 버퍼 (8-4-6-4-16)를 갖는 프로세서와 기본 프로세서를 비교하면 0.5% 이내에 드는 비슷한 성능을 보여 주지만 포트에서 하드웨어 자원을 82%이상 줄일 수 있었다.

분산된 리오더 버퍼를 갖는 슈퍼스칼라 프로세서의 성능을 분석하고 분산된 리오더 버퍼의 값을 찾기 위하여 M/M/1 큐잉 모델을 이용하였다. 분산된 리오더 버퍼의 크기를 결정하는데 분산된 리오더 버퍼가 가득 찰 확률을 10% 이내로 하고 크기를 결정하였다.

리오더 버퍼를 분산된 리오더 버퍼로 나누는 목적은 성능 저하를 최소화 하면서 리오더 버퍼의 복잡도를 낮추는 것이다. 분산된 리오더 버퍼의 크기와 성능과의 관계를 본 논문에서 제시한 수학적 모델을 통해 분석하고 시뮬레이션 하였다. 기본 프로세서의 리오더 버퍼는 96개의 엔트리를 갖는 반면에 분산된 리오더 버퍼 8-2-4-2-18의 경우 총 88개의 엔트리를 갖고 있다. 96개의 엔트리를 갖고 4W개의 포트를 갖고 있는 리오더 버퍼를 2:1 멀티플렉서로 구성할 경우 멀티플렉서가 404개 필요한 반면 각각의 한 개의 읽기 포트를 갖는 분산된 리오더 버퍼의 경우 총 88개의 엔트리를 갖고 2:1 멀티플렉서가 74개 필요하다. 반면에 성능의 저하는 0.8%에 불과하다. 이처럼 분산된 리오더 버퍼의 경우 원하는 범위내의 성능에서 하드웨어 자원이 많이 줄어든 것을 확인할 수 있었다.

참 고 문 헌

[1] G. Kucuk, D. Ponomarev, O. Ergin, and K. Ghose "Complexity-Effective Reorder Buffer Designs for Superscalar Processors," IEEE Transaction on Computers, pp. 653-665, June 2004.

[2] G. Kucuk, D. Ponomarev, and K. Ghose, "Low-Complexity Reorder Buffer Architecture," Proc. Int'l Conf. Supercomputing(ICS '02), pp. 57-66, June 2002.

[3] S. Wallace, N. Dagli, and N. Bagherzadeh, "Design and Implementation of a 100MHz Centralized Instruction Window for a Superscalar Microprocessor," 1995 International Conference on Computer Design, pp. 1-6, October 1995.

[4] P. Dubey, G. Adams III, and M. Flynn, "Instruction window size trade-offs and characterization of program parallelism," IEEE Trans. Computers, pp. 431-442, April 1994.
[5] S. Gurindar, and S. Vajapeyam, "Instruction Issue Logic for High-performance, Interruptable Multiple Functional unit, Pipelined Computer," IEEE Transaction on Computers, pp. 349-359, March 1990.
[6] J. Hennessy, and D. Patterson, Computer Architecture : A Quantitative Approach, 3rd edition, Morgan Kaufmann, pp.171-209, 2002.
[7] G. Intrater, and R. Talmudi, "A Superscalar Microprocessor," Proceeding of the 17th Convention of Electrical and Electronics Engineers in Israel, pp. 267-270, March 1991.
[8] J. Smith, and G. Sohi, "The Microarchitecture of Superscalar Processors," Proceedings of the IEEE, Vol.83, No.12, pp.1609-1624, December 1995.
[9] D. Bertsekas, and R. Gallager, Data Networks, Prentice Hall, pp. 149-173. 2002.
[10] M. Hassan, and R. Jain, High Performance TCP/IP Networking, Prentice Hall, pp. 335-337. 2000.
[11] D. Burger, and T. Austin, "The SimpleScalar tool set : Version 2.0," Tech. Report, Dept. of CS, Univ. of Wisconsin-Madison, June 1997.

저 자 소 개

윤 완 오(학생회원)
2000년 경기대학교 전자공학과 학사 졸업.
2002년 인하대학교 전자공학과 석사 졸업.
2002년~현재 인하대학교 전자공학과 박사 과정.
<주관심분야 : 분산 처리 시스템, 병렬프로그래밍, 컴퓨터 아키텍처>

김 경 섭(학생회원)
2002년 한남대학교 전자공학과 학사 졸업.
2002년~현재 인하대학교 전자공학과 석사 과정.
<주관심분야 : 컴퓨터 아키텍처, SoC & 임베디드 시스템 디자인>

최 상 방(평생회원)
1981년 한양대학교 전자공학과 학사 졸업.
1981년~1986년 LG 정보통신(주)
1988년 University of Washington 석사 졸업.
1988년 University of Washington 박사 졸업.
1991년~현재 인하대학교 전자공학과 교수
<주관심분야 : 컴퓨터 구조, 컴퓨터 네트워크, 무선 통신, 병렬 및 분산 처리 시스템, antFault-tolerant computing>

신 광 식(학생회원)
2001년 인하대학교 전자공학과 학사 졸업.
2003년 인하대학교 전자공학과 석사 졸업.
2008년 인하대학교 전자공학과 박사 졸업.
<주관심분야 : 멀티미디어 데이터 통신, 컴퓨터 네트워크, 무선통신>

이 윤 섭(학생회원)
2006년 경희사이버대학교 정보통신학과 학사 졸업.
2008년 인하대학교 전자공학과 석사 졸업.
2008년~현재 인하대학교 전자공학과 박사 과정.
<주관심분야 : 컴퓨터 아키텍처, SoC & 임베디드 시스템 디자인>