논문 2008-45SD-11-9

# 마이크로프로세서 전력소모 절감을 위한 명령어 큐 구조

## ( Instruction Queue Architecture for Low Power Microprocessors )

최 민*, 맹 승 렬*

( Min Choi and Seungryoul Maeng )

요 약

현대 마이크로프로세서는 적정수준의 전력소모에 고성능의 애플리케이션성능을 요구한다. 전력소모와 성능향상의 상호보정 측면에서 볼때, 명령어 윈도우(Instruction window)는 특별히 중요한 구성요소이다. 이는 명령어 윈도우의 크기를 확장하면 성능향상을 가능하도록 하지만, 기존의 명령어 구조를 그대로 이용하여 크기만 늘리는 것은 전력소모와 복잡도 측면에서 불리하기 때문이다. 본 연구에서는 전력소모를 감소하기 위해서 직접 검색 테이블(Direct table lookup :DTL)을 사용하여 명령어 윈도우에서 발생하는 연관 검색을 최소화한다. 이를 위해 비트 벡터(bit-vector) 기반의 태그 변환 기법을 제안하여 데이터 종속성 및 자원 충돌 현상을 효과적으로 해결한다. 본 논문에서는 SPEC2000 벤치마크를 활용하여 성능평가를 수행하여 제안된 기법이 기존 방법 대비 24.45%의 전력소모 개선 효과를 나타냄을 확인하였다.

Abstract

Modern microprocessors must deliver high application performance, while the design process should not subordinate power. In terms of performance and power tradeoff, the instructions window is particularly important. This is because a large instruction window leads to achieve high performance. However, naive scaling conventional instruction window can severely affect the complexity and power consumption. This paper explores an architecture level approach to reduce power dissipation. We propose a low power issue logic with an efficient tag translation. The direct lookup table (DTL) issue logic eliminates the associative wake-up of conventional instruction window. The tag translation scheme deals with data dependencies and resource conflicts by using bit-vector based structure. Experimental results show that, for SPEC2000 benchmarks, the proposed design reduces power consumption by 24.45% on average over conventional approach.

**Keywords:** 저전력 마이크로아키텍쳐 (low-power microarchitecture), 수퍼스칼라 프로세서(superscalar processor), 명령어 윈도우 (instruction window)

## I. 서 론

현대 마이크로 프로세서에서 파워 및 성능 효율성 측면에서 중요한 구성요소는 명령어 윈도우(Instruction window)이다. 명령어 윈도우는 디스패치한 명령어들 중에서 독립적으로 동시 실행 가능한 명령어들을 찾아내어 실행 장치로 투입하는 구성요소이다. 명령어 윈도우의 크기를 확장하면 더 많은 명령어 수준의 병렬성(Instruction level parallelism: ILP)을 노출하는 효과를 나타낸다. 하지만, 기존의 명령어 구조를 그대로 이용하여 크기만 늘리는 것은 복잡도와 전력소모에 나쁜 영향을 미칠 수 있다. 실제로, Folegnani와 Gonzalez는 동적 스케줄링 프로세서에서 명령어 윈도우를 구성하는 명령어 이슈 큐(issue queue)가 가장 복잡하고 전력소모가 심한 구성요소임을 보였다. 이러한 주된 원인은 명령어 윈도우가 내용 기반 검색 메모리(Content addressable memory: CAM)에 기반한 구조를 가지기 때문이다. 명령어 윈도우의 많은 비교 회로는 각 소스 연산자의 태그 값을 방송된 태그값과 매번 비교하기 때문에 전력소모가 크다. 따라서, 본 연구에서는 전력소모를 최소화하면서도 확장성있는 명령어 윈도우 구조를 설계하는 방법을 제안한다. 이를 위해, 직접 테이블 변환(Direct lookup table: DTL) 구조를 적용한 명령어 윈도우를 활용한다. 또한, DTL 윈도우 자원를 효과적으로 활용하기 위해서 레지스터 재명명 기법을 수정하였다. 본 연

* 학생회원, ** 평생회원, 한국과학기술원 전자전산학과 (KAIST)

접수일자: 2008년8월23일, 수정완료일: 2008년10월30일

구에서는 프로그램 특성을 분석한 결과를 바탕으로 명령어 이슈로직에서 전력소모를 최적화하기 위한 태그 변환 장치(Tag translation unit: TTU) 구조를 제안한다. TTU는 DTL 명령어 윈도우 내에서 이미 다른 명령어에 의해 같은 명령어 슬롯이 점유되어 있는 경우 발생하는 자원 충돌 현상을 해결하기 위한 구성요소이다. 제안된 기법의 성능을 평가하기 위해, Sim-Panalyzer[5] 시뮬레이터에서 구현하고 SPEC CPU 2000 벤치마크[6]를 실행하였다.

본 논문은 다음과 같이 구성된다. Ⅱ장은 명령어 윈도우 구조에 대한 기존 관련연구를 간략히 소개한다. Ⅲ장은 본 논문의 연구동기에 대해 서술하고, Ⅳ장은 본 논문에서 제시하는 DTL 명령어 윈도우 구조에 대한 설계를 기술한다. 마지막으로, Ⅴ장에서 성능 및 전력소모에 대해 실험 및 평가한다.

## Ⅱ. 관련연구

Ehrhart[3]와 Ernst[4]는 명령어 윈도우 논리회로의 복잡도를 줄이기 위해서, 내용 기반 검색 메모리를 사용하는 명령어 탐색구조를 제거하는 기법을 제안하였다. 이 방법은 명령어의 실행 시간 및 이슈 지연시간을 예측하여 잘못 예측된 명령어들을 선택적으로 재실행한다. 그러나 이와 같이 예측에 기반한 방법은 경우에 따라 부정확한 결과를 가져올 수 있다. 예를들면, 캐시 미스와 같이 수행시간을 예측하기 힘든 경우가 이에 해당한다. Weiss[8], Weinraub[9], 그리고 Canal[2]은 명령어 윈도우에서 연관 태그 탐색(Associative tag search)을 회피하는 방법으로 파워 소모를 절감하는 방법을 제안하였다. 이 방법은 명령어 큐를 구현하는 데 사용되는 CAM 대신 일반 RAM 구조의 명령어 큐와 연관 매핑 테이블(Associative mapping table)을 사용한다. 그림 1은 CAM 기반 명령어 큐와 DTS 기반 명령어 큐를 비교한 것이다. 이 때, 연관 매핑 테이블은 RAM 기반 명령어 큐에서 자원의 충돌 현상이 발생하는 경우에만 활용된다. RAM 기반 명령어 큐는 해당 번지에 한 개의 특정 태그 정보만 저장할 수 있기 때문이다. 하지만, 일반적으로 프로그램에 내재된 명령어 종속성으로 인해 매핑 테이블의 크기를 작게 유지하기 힘들다.

따라서 이러한 접근방법 역시 논리회로의 복잡성을 획기적으로 개선하지는 못하였다. Ponomarev[7]과 Buyuktosunoglu[1]는 명령어 윈도우의 사용 형태를 주기적으로 샘플링 함에 의해서 그 크기를 동적으로 조정하는 방법을 제안한다. 그러나 이러한 방법도 특히 비활성화 된 명령어 큐의 파티션을 다시 활성화 시키려고 하는 경우 오버헤드가 존재한다. 또한, 부가적인 하드웨어와 제어회로가 명령어 큐의 활용정도를 모니터링 하는데 요구된다.

여기서는 본 논문의 연구와 비교적 더 관련 깊은 Weinraub[9]과 Canal[2]의 접근방법을 보다 세부적으로 비교분석 하고자 한다. Weinraub은 그림 2와 같은 저전력 명령어 윈도우 구조를 제안한다.

이 방법은 CAM 기반의 명령어 큐를 RAM으로 대체하는 것으로 본 논문에서는 성능 비교를 위한 실험 대조군(baseline) 으로 사용한다. 명령어 큐에 대한 주소를 보관하는 테이블 구조와 완전 연관 버퍼(Fully associative buffer)의 두 가지 추가적인 구조가 활용된다. 전자는 IQ 포인터(IQ Pointer: IQP)의 역할을 하고, 후자는 태그 연관 버퍼(Tag Associative Buffer: TAB)로서 동작한다. IQP는 태그를 테이블에 대한 인덱스(index)로 사용함에 따라 연관 검색을 제거하는 효과가 있다. 태그연관 버퍼는 작은 완전 연관 버퍼 구조로서,

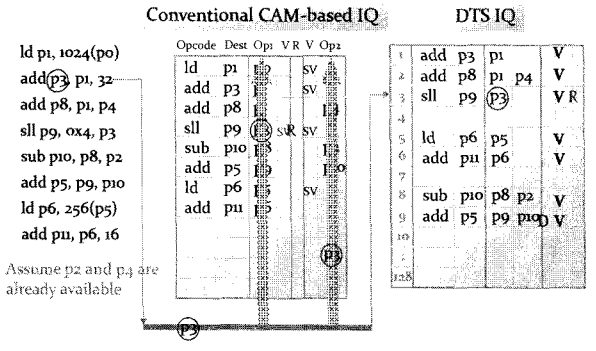


그림 1. CAM 기반 명령어 큐와 DTS 명령어 큐
Fig. 1. Comparison of conventional instruction queue and DTS based one.

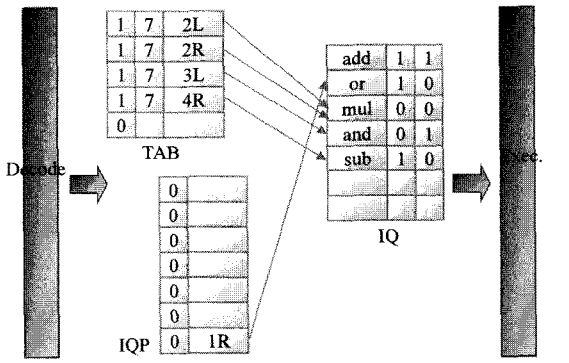


그림 2. Weinraub의 저전력 명령어 윈도우 구조
Fig. 2. Weinraub's low power instruction window architecture.

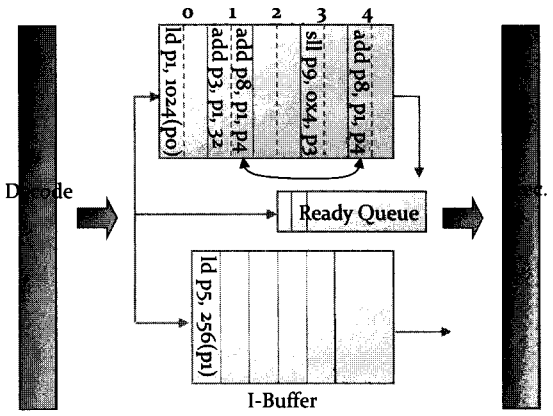


그림 3. Canal의 저복잡도 명령어 이슈 큐 구조
Fig. 3. Canal's low complexity instruction issue queue architecture.

IQP의 자원 충돌시 이를 해결하기 위해 활용된다. 예를 들어, IQP의 특정 엔트리가 이미 사용 중인 경우, 다음 번 새 명령어의 소스 연산자가 이와 동일할 때, 태그연관버퍼에 태그와 실제 해당 명령어가 위치하는 슬롯의 인덱스를 기록함으로써 자원 충돌현상을 해결한다. 이러한 기법은 완전 연관 검색 시 발생하는 태그 불일치(Tag mismatch) 횟수를 감소시키며, 결과적으로 CAM 구조에서 발생하는 전력 소모를 줄인다. 그럼에도 불구하고, 이 방법은 여전히 TAB에서 완전 연관 검색 구조를 활용하고 있으며 또한, TAB와 IQP 자체에서 발생하는 전력소모에 대해서는 고려하지 않은 단점이 있다.

그림 3은 Canal[2]의 N-use 방법을 나타낸다. 이는 DTS 방법을 직접적으로 재구성한 연구로서, 연관 메모리를 사용해서 DTS기반 명령어큐의 단일 개 슬롯에 최대 N개(설정가능)까지 할당할 수 있도록 하여 DTS의 단점을 개선한 것이다. 그러나 이 방법은 N-use 테이블을 구성하기 위해서 여전히 집합 연관 메모리(Set associative memory)를 사용하는데다가 자원 충돌 시 이를 해결하기 위한 방안으로 CAM 버퍼를 사용하기 때문에 여전히 높은 전력소모를 갖는다. 또한, 이 논문에서는 명령어 이슈로직의 복잡도를 개선하는 것이 주요 목적이기에 전력소모 관점에서의 실험 및 평가 결과가 제공되지 않는다.

## III. 연구 동기

이 장에서는 전력 소모 절감으로부터 얻어지는 장점을 확인하기 위해서 일반적인 프로그램의 실행 행태를 분석한다. 그림 4에서는 하나 또는 둘 이상의 소스 오

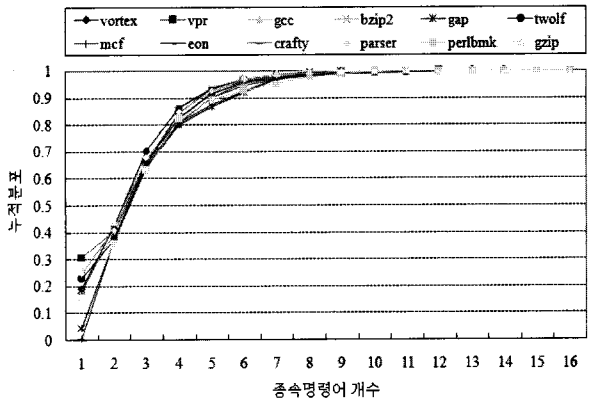


그림 4. 종속 명령어 개수에 따른 누적분포
Fig. 4. Cummulative distribution of the number of dependent instructions.

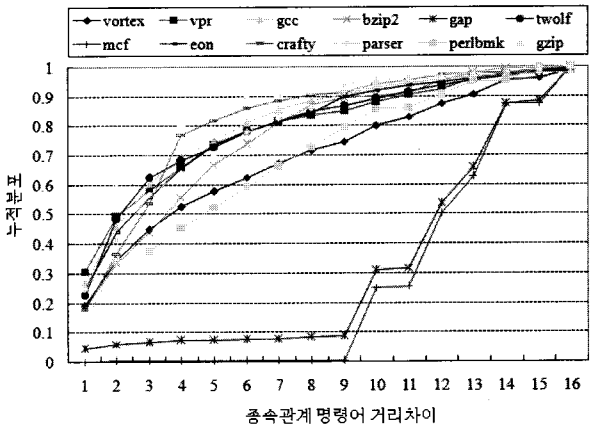


그림 5. 종속 명령어 거리 차이
Fig. 5. Dependency displacement of dependent instructions.

퍼랜드가 실행되기 위해서는 대부분의 경우에 다른 명령어와 데이터 종속관계에 있다는 사실을 나타낸다. 약 80%의 명령어들이 평균적으로 2-3개 명령어들과 종속관계를 가진다는 사실을 나타낸다. 이러한 사실은 본 논문의 DTL 기법을 최소한의 오버헤드로 실현할 수 있음을 증명한다. 이는 전체 명령어 중 50% 가량이 한 개 또는 그 이하의 종속관계를 갖는 명령어를 갖기 때문이다. 그림 5는 명령어들의 종속관계가 해결되는 거리차이를 보여준다.

즉, 어떤 명령어가 연산을 위해 필요로 하는 레지스터가 있을 때, 그 값을 생성하는 명령어가 어느 정도 위치에 떨어져 있는지를 나타낸다. 데이터 종속성의 60% 이상이 최대 6-7개 명령어 이내에서 해결된다는 사실은 대부분의 명령어 종속관계가 근거리에서 해결된다는 뜻이다. 즉, 기본 블록(Basic block) 내부에서 대부분의 종속관계가 해결된다는 의미이다. 종속관계가 이는 SPEC 2000 벤치마크 애플리케이션을 통해 실험적으로 얻은

결과이다. 물론, 예외적으로 mcf와 gap은 조금 다른 특성을 보이는데, 이들 애플리케이션 들은 메모리 로드 및 저장 연산을 많이 수행하는 특성이 있어서 종속 명령어들 간의 거리에 있어 큰 값을 보인다.

## Ⅳ. DTL 명령어 윈도우

### 1. DTL 구조 (DTL Architecture)

이 장에서는 기존 Alpha 21264 파이프라인 구조를 기반으로 기존 명령어 큐를 DTL 기반의 구조로 대체한 아키텍처를 소개한다. DTL 아키텍쳐는 레지스터 배열의 데이터 접근이 연관 태그 검색을 수행하는 것보다 훨씬 빠르고 저전력 특성을 갖는다는 사실에 기반한다. DTL 기법은 위와 같은 특징을 활용하기 위해서 명령어 윈도우 구조를 레지스터의 다수의 배열 형태로 구성한다. 매번 명령어가 수행될 때에, 결과 레지스터 태그는 테이블 기반의 DTL 명령어 큐를 검색하기 위한 인덱스로 사용된다.

이를 위해, 새롭게 디코딩되어 들어오는 명령어들을 결과 태그가 가리키는 값을 기반으로 DTL 슬롯에 배치한다. 이것이 가능한 이유는, 일반적으로 명령어 수행에 의해 생성된 결과 값은 대부분의 경우 최대 한번만 사용되는 경향을 가지기 때문이다. 물론, 빈번하게 발생하지는 않지만, 앞서 이슈된 명령어의 의해 이미 할당되어 있는 슬롯에 또 다른 명령어가 들어오는 경우, 충동상황을 해결하기 위해서 TTU 구조를 제안한다. 본 논문은 저전력 명령어 윈도우를 위한 기본구조로서 DTL 명령어큐와 함께 태그 변환 장치(Tag translation unit: TTU)를 적용한다. TTU는 공통 데이터 버스(Common data bus: CDB)상에서 버스를 지나는 데이터를 버스 스누핑(Snooping)으로 감시한다. 본 논문에서 제안하는 DTL 방법은 기존 명령어 윈도우 구조에서 사용되는 연관 검색을 가급적 배제한다. 이를 위해서 CAM 기반의 명령어 윈도우를 단순한 테이블 형태의 명령어 구조로 대체한다. 그리고 값을 생성하는 명령어와 소비하는 명령어를 연결하는 방식으로 레지스터 종속성을 유지 및 관리 한다. 그림 7은 DTL 명령어 큐 구조를 보여주는데, 각 슬롯에는 세 가지 다른 비트 플래그(bit flag)들이 있다. 각각, 유효(valid), 준비(ready), 그리고 더블(double) 오퍼런드에 대한 정보를 표현한다.

유효 비트는 명령어 큐의 각 슬롯에서 유효한 정보를 담고 있는지를 나타낸다. 준비 비트는 명령어 실행에 필요한 근원 레지스터들이 모두 활용가능한지 여부를 나타낸다. 만약, 준비 비트가 설정된 경우, 해당 명령어는 실행유닛으로 이슈될 자격을 얻는다. 더블 비트는 명령어가 필요로 하는 근원 오퍼런드의 개수가 두개 인 경우를 나타낸다. DTL 명령어 윈도우는 특성상 한 명령어 슬롯에 한 개 명령어 종속성만 표현하도록 설계되었기 때문에 두 개의 근원 오퍼런드를 필요로 하는 경우를 위한 대비책이다. 이러한 경우에 과거 N-use 방법에서 같은 명령어 정보를 두 군데 명령어 슬롯에 중복 저장하는데 비해 본 연구에서는 오직 한 개 슬롯에만 정보를 유지하므로(그림 7의 ②번 경우) 전체 명령어 윈도우의 자원 활용률 측면에서 상당히 개선된 방법이다. 단지, 슬롯당 한 개 비트씩 추가로 도입하여 두 개의 근원 오퍼랜드를 요구하고 있음을 알려준다. 이후부터는 태그 변환을 통해 TTU가 데이터 종속성을 해결한다. 즉, 더블 비트를 가지는 명령어가 필요로 하는 근원지 오퍼런드에 대한 결과값이 CDB상에 발생했을 때, 그 결과값을 더블 비트가 켜져있는 명령어 슬롯으로도 들어가도록 변환해주는 방법이다. 실제로, 그림 7

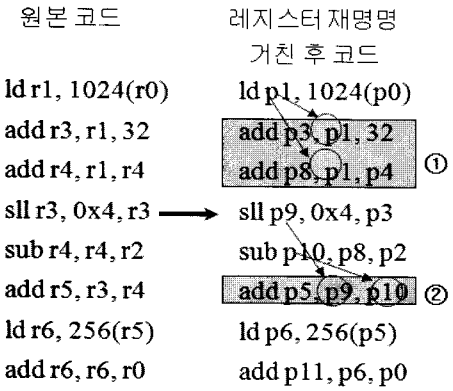


그림 6. 레지스터 종속성
Fig. 6. Register dependencies.

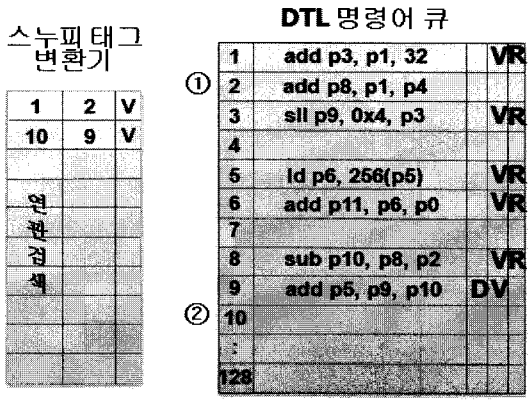


그림 7. DTL 명령어큐 구조와 명령어 할당
Fig. 7. DTL instruction queue architecture and instruction allocation.

에서 태그 10번에 대한 결과값이 발생했을 때, TTU에서 10->9 변환이 이루어져 추가적으로 9번 슬롯에 있는 명령어에도 해당 결과값이 전송된다.

### 2. STT와 STT 비트 벡터(Bit-Vector)

STT는 연관 검색방식의 래치 구조를 가지며, CDB로부터 발생하는 트랜잭션을 스누핑할 수 있다. 따라서, CDB상의 각 명령어의 결과 태크가 STT 내 엔트리의 각 필드와 일치하는 경우 이에 대응하는 또다른 결과 태그를 생성한다. 예를들어, 한 명령어에 대해서 그에 뒤따르는 또 다른 명령어가 서로 동일한 소스 오퍼런드를 가지는 경우 이미 앞선 명령어의 오퍼런드가 명령어 큐에 할당되어 있을 것이다. 이 때, 레지스터 재명명 로직은 그림 7의 경우 ①과 같이, 뒤 따르는 명령어를 명령어 큐 상의 임의의 위치에 할당한다. 그러는 동안에 STT는 소스 오퍼런드와 실제로 할당된 인덱스 값의 매핑을 보관하다가, 향후에 CDB상에 소스 오퍼런드 값에 대한 태그가 발생할 때, 이러한 정보를 바탕으로 태그 변환을 수행한다. 이와 같은 하드웨어적 지원을 통해 복잡한 연관 태그 매칭 연산이 단순한 테이블 검색으로 변경될 수 있다. 이를 효과적으로 구현하기 위해서, 본 논문에서는 그림 8과 같이 비트 벡터(bit-vectors)를 활용한 태그변환 로직을 제안한다.

STT 비트 벡터는 STT에 비해서 속도가 빠르면서도 적은 전력을 소모하는 구조로 되어 있다. 결과 태그가 CDB상에 뜰 때, STT 비트 벡터가 체그된다. 만약, 해당 결과 태그에 해당하는 비트가 설정되어 있다면, 또 다른 결과 태그가 TTU에 의해서 생성된다. 본 연구에서는 12비트 크기의 STT 비트 벡터를 활용하며, 이는

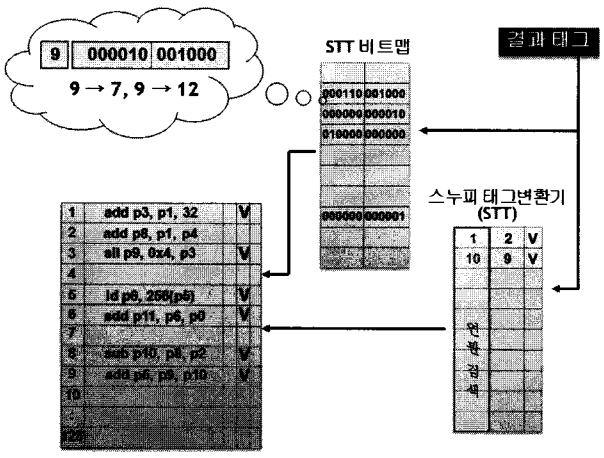


그림 8. STT 비트 벡터와 스누핑 구조
Fig. 8. STT bitmap and bus snooping.

각각 6bit씩 나뉘어 중간부터 앞/뒤로 사용된다. 즉, 처음 6비트는 정방향으로 명령어 종속성을 나타내며, 두 번째 6비트는 역방향으로 명령어의 종속성을 나타낸다.

본 연구에서 STT 비트 벡터의 폭을 6으로 설정한 것은 대부분의 명령어 종속성이 기본 블록(basic block) 내에 존재한다는 특성으로부터 비롯하는데, 일반적으로 기본 블록의 크기는 6~7 명령어 정도로 알려져 있다[패터슨 책]. STT 비트 벡터의 각 비트는 명령어 종속성의 상대적 거리를 나타낸다. 예를들어, STT-BV내 첫 번째 엔트리가 000000:100000 값을 가지는 경우, 해당 명령어로부터 첫번째 다음에 나타나는 명령어가 이 명령어에 대해 명령어 종속성을 가지는 상황을 표현한다. 즉, 태그 1에 대항하는 오퍼런드 결과값을 동일하게 필요로 한다. 또 다른 경우는 그림 7의 ②번 경우와 같이 첫번째 이전 명령어가 역시 태그 10에 해당하는 같은 결과값을 필요로 하는 경우이다. 물리적으로 레지스터 태그의 개수가 32개이라고 가정하면, STT 비트 벡터는 총 384개(32 X 12)까지의 명령어 종속성을 표현할 수 있다. 이와는 다르게 STT 구조만 활용하는 경우 STT 구조의 크기(capacity)만큼에 해당하는 32개의 명령어 종속성을 표현할 수 있다. 디스패치 되어 들어오는 명령어들에 대해서 명령어 큐에 엔트리를 할당할 때, DLT 기법은 해당 명령어를 비교적 생성자 명령어로부터 가까운 위치에 위치시키도록 노력한다. 실제로, 가능하면 생성자 명령어로부터 앞뒤로 6개 명령어 슬롯 안에 위치시킨다.

## V. 실 험

본 연구에서 제안하는 방법의 성능 평가를 위해 Sim-Panalyzer에서 SPEC 2000 CPU 벤치마크를 수행하여 실험하였다. Sim-Panalyzer는 cycle 단위의 정확성을 갖는 아키텍쳐 수준의 파워 시뮬레이터이다. Sim-Pabalyzer는 SimpleScalar를 기반으로 개발된 전력소모 분석 툴이다. Sim-Panalyzer에서는 명령어 이슈 큐와 순서재배치 버퍼(reorder buffer) 그리고 레지스터(physical register)를 레지스터 업데이트 유닛(Register Update Unit: RUU)으로 통합하여 제공한다. 본 연구에서는 현대 마이크로프로세서 구조를 보다 정확하게 모델링하기 위해서, 순서 재배치 버퍼와 명령어 큐, 그리고 레지스터를 먼저 분리하였다. SPEC 벤치마크 수행시 실행 초기부분을 결과에서 제외하기 위해 10억개 명령어를 건너 뛰어 실행하도록 했다. 이는 응용 프로그

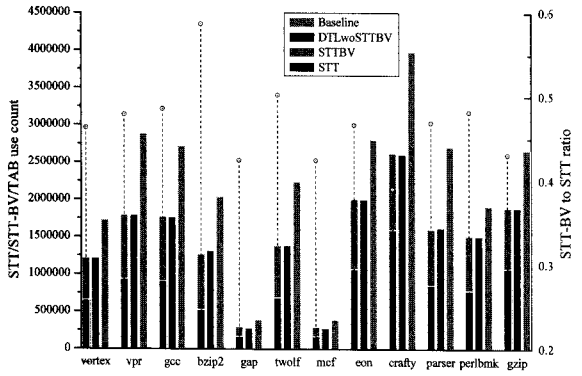


그림 9. STT/STT 비트벡터 TAB 사용량 비교
Fig. 9. STT/STT bitmap and TAB usage.

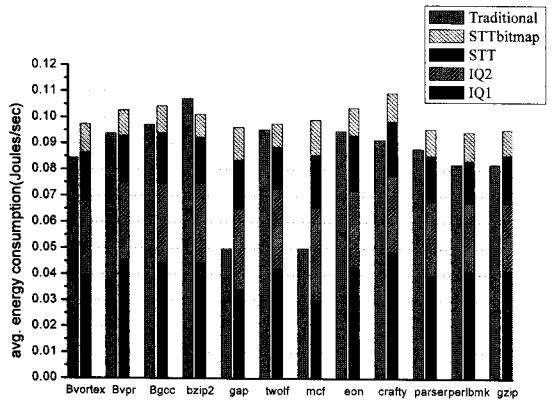


그림 10. 윈도우 크기가 32일때의 파워 소모 비교
Fig. 10. Power consumption on 32 Instruction queue.

램 실행의 초기 부분이 나머지와 매우 다른 실행 형태를 보이기 때문이다. 결과는 벤치마크의 각 응용프로그램 당 100만개 명령어씩 실행하면서 성능을 측정하였다. 성능 비교대상은 본 논문의 DLT 명령어 윈도우 방법과, 기존 CAM 기반의 명령어 윈도우 방법, 그리고 Weinraub[9] 방법이다.

그림 9는 STT-BV/STT의 사용량 비교를 통한 DLT 명령어 큐의 충돌정도를 나타낸다. 명령어 큐에서 충돌은 전력 소모에 큰 영향을 미친다.

왜냐하면 명령어 윈도우의 할당방법은 TTU를 통해서 충돌을 해결하는데 이처럼 STT에 접근하는 것은 STT-BV에 비해서 비교적 큰 전력소모를 요구하기 때문이다. 본 연구에서는 49.77%의 STT-BV 사용정도을 보였다. 다시 말하면, 전체 명령어들 중 절반에 해당하는 경우에 전력소모가 적은 STT-BV를 활용하여 태그 변환이 이루어졌다는 것이다.

그림 10과 11은 윈도우 크기가 각각 32와 64인 경우

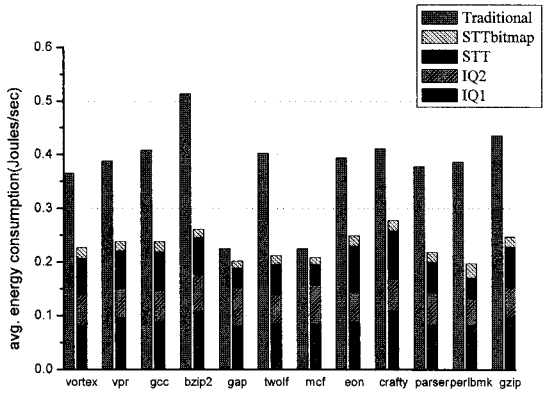


그림 11. 윈도우 크기가 32일때의 파워 소모 비교
Fig. 11. Power consumption on 64 instruction queue.

측정한 파워소모량의 비교이다. 기존 방법에 비해서 본 논문에서 제안하는 STT와 STT 비트벡트를 적용한 방법이 전력소모를 감소시키는 효과가 나타난다. 특히, 윈도우 크기가 커짐에 따라 보다 많은 파워절감효과를 보이는 것을 확인하였다. 이는 윈도우 크기를 확장하는 경우 완전연관 메모리 구조인 CAM의 태그 검색 로직의 확장성 결여로 인해 전력소모가 급격히 증가하는 경향을 띄기 때문이다.

## VI. 결 론

본 논문에서는 일반적인 프로그램의 실행특성을 분석함으로써, 새로운 명령어 윈도우의 구조를 제안며 태그 변환 기법을 제안하였다. DLT 기법은 기존 방법에서 필수불가결한 것으로 여겨졌던 명령어 윈도우 상의 연관 검색기법을 최대한 지양하고, 결과태그를 직접 검색하는 방식으로 동작한다. 또한, 속도가 빠르면서도 적은 전력을 소모하는 STT 비트 벡터 구조를 도입하여 전체 명령어들 중 절반가량을 이곳에서 태그변환 함으로써 전력소모 정도를 줄인다. 이 방법은 STT에서 발생하는 연관 검색을 감소시키는 효과를 유발한다. 이를 통해 본 논문에서는 기존 직접 태그 검색 기법 대비 24.45% 전력소모 향상효과를 얻었다.

## 참 고 문 헌

[1] A. Buyuktosunoglu, D. Albonesi, S. Schuster, D. Brooks, P. Bose and P. Cook. A Circuit Level Implementation of an Adaptive Issue Queue for Power-Aware Microprocessors, In the Proceedings of the GLSVLSI, 2001.

[2] R. Canal. Reducing The Complexity of The Issue Logic, In Proceedings of the ACM International Conference of Supercomputing, 2001.
[3] T. Ehrhart. Reducing the Scheduling Critical Cycle using Wakeup Prediction, In the Proceedings of the International Symposium on High-Performance Computer Architecture, 2004.
[4] D. Ernst. Cyclone: A Low-Complexity Broadcast-Free Dynamic Instruction Scheduler, In Proceedings of the IEEE International Symposium on Computer Architecture, 2003.
[5] Sim-Panalyzer, v2.0.2, http://www.eecs.umich.edu/ panalyzer/
[6] SPEC CPU2000, http://www.spec.org/cpu/
[7] D. Ponomarev, G. Kucuk, and K. Ghose, Reducing Power Requirements of Instruction Scheduling Through Dynamic Allocation of Multiple Datapath Resources, In the Proceedings of International Symposium on Microarchitecture, Dec. 2001.
[8] S. Weiss, J. Smith, Instruction Issue Logic in Pipelined Supercomputers, IEEE Transactions on Computers, vol.39, no.3, 1990.
[9] Y. Weinraub. Power-Aware Out-of-Order Issue Logic in High-Performance Microprocessors, Microprocessors and Microsystems, 30(7): 457-467, 2006.

## 저 자 소 개

최 민(학생회원)
2001년 광운대학교 전자계산학과 학사 졸업.
2003년 한국과학기술원 전자전산학과 석사 졸업.
2008년 현재 한국과학기술원 전자전산학과 박사 과정.
<주관심분야 : 컴퓨터구조, 임베디드 시스템>

맹 승 렬(평생회원)
1977년 서울대학교 전자공학과 학사 졸업.
1979년 한국과학기술원 전산학과 석사 졸업.
1984년 한국과학기술원 전산학과 박사 졸업.
1988년~1989년 University of Pennsylvania, Visiting Professor
1994년~1995년 University of Texas, Visiting Professor
2008년 현재 한국과학기술원 전자전산학과 전산학전공 교수
<주관심분야 : 컴퓨터구조, 병렬처리, 임베디드시스템>