효율적인 분기 예측을 위한 공유 구조의 BTB

이용환*

A Combined BTB Architecture for effective branch prediction

Yong-hwan Lee*

본 논문은 금오공과대학교 학술연구비에 의하여 연구된 논문입니다.

요 약

프로그램의 순차적인 실행 순서를 바꾸는 명령어를 분기 명령어라 하며, 분기는 마이크로프로세서의 파이프라인 정지를 일으켜 성능을 저하시키는 가장 큰 원인이 된다. 이에 따라 분기를 정확히 예측하여 다음 실행될 명령어를 제공한다면 마이크로프로세서의 자연스런 명령어의 실행 흐름은 끊어지지 않게 되고 이로써 높은 성능의 향상을 기대할 수 있게 된다. 분기 예측을 위해서는 분기 타겟 버퍼가 필수적이며, 분기 타겟 버퍼는 분기 예측 결과에 따라 다음에 실행할 명령어의 주소를 제공한다. 본 논문에서는 가상주소를 실제주소로 바꾸어 주는 TLB와 분기 타겟 버퍼가 각각 가지고 있는 태그 메모리를 함께 사용하는 구조를 제안한다. 이러한 공유 태그 구조의 이점은 2개의 태그 메모리를 하나로 공유함으로써 칩 면적의 감소를 꾀하고 더불어 분기 예측 속도를 향상시킬 수 있다는 점이다. 또한, 본 논문에서 제안된 구조는 주소로 사용되는 비트 수가 커지거나 여러 개의 명령어를 동시에 실행할 수 있는 구조에서 그 이점이 더욱 커지기 때문에 향후 개발되는 마이크로프로세서에서 유용하게 사용될 수 있을 것으로 기대된다.

ABSTRACT

Branch instructions which make the sequential instruction flow changed cause pipeline stalls in microprocessor. The pipeline hazard due to branch instructions are the most serious problem that degrades the performance of microprocessors. Branch target buffer predicts whether a branch will be taken or not and supplies the address of the next instruction on the basis of that prediction. If the branch target buffer predicts correctly, the instruction flow will not be stalled. This leads to the better performance of microprocessor. In this paper, the architecture of a tag memory that branch target buffer and TLB can share is presented. Because the two tag memories used for branch target buffer and TLB each is replaced by single combined tag memory, we can expect the smaller chip size and the faster prediction. This shared tag architecture is more advantageous for the microprocessors that uses more bits of address and exploits much more instruction level parallelism.

키워드

branch prediction, branch target buffer, microprocessor

* 금오공과대학교 전자공학부

접수일자 : 2005. 10. 28

Ⅰ. 서 론

마이크로프로세서의 성능은 프로세서의 동작 속도가 빠를수록 그리고 한 사이클에 실행할 수 있는 명령어의 수가 많을수록 높아진다. 한 사이클에 실행할 수 있는 명령어 수를 늘리기 위해서는 명령어 수준 평행성(instruction level parallelism)을 사용하는데 이에는 수퍼스칼라(super-scalar) 방식, 비순차적 실행(out-of-order execution)기법, 다중 쓰레디드(multi-threaded), VLIW(very long instruction word) 등이 이에 속하며, 이들은 서로 의존관계가 없는 명령어들을 동시에 실행하도록 함으로써 속도를 높이게 된다. 동작 속도를 높이기 위해서는 파이프라이닝(pipelining) 기법을 사용한다. 파이프라이닝은 명령어의 실행을 여러 단계로 나누어 행하고 한 시점에 각 단계의 여러 명령어가 동시에 실행될 수 있도록 함으로써 그 동작 속도를 향상시키게 된다.

그러나 이러한 명령어 수준의 평행성을 사용할 때 가장 큰 걸림돌이 되는 것이 분기(branch)이다. 분기 명령어는 순차적으로 실행되는 명령어의 흐름을 바꾸는 명령어로서 이 명령어에 의해 분기가 채택(taken)된다면 파이프라인에서 순차적으로 흐르고 있던 명령어를 무효화시키고 새로운 명령어를 가져와야 한다[1]. 따라서 이러한 과정에서 이미 메모리에서 가져와 실행되고 있는 명령어들의 처리가 되지 않고 사라지며 이에 따라 낭비되는 사이클이 생기게 되므로 프로세서의 성능 저하가 생기게 된다. 이러한 분기에 의한 성능저하는 파이프라인의 단계가 많을수록 더욱 심각하다. 예를 들어 임베디드 프로세서로 많이 사용되는 ARM7이나 ARM9[2] 같은 경우는 파이프라인 단계가 각각 3단계와 5단계로 이때 분기에 의한 영향이 없거나 또는 많아야 1사이클의 낭비가 초래되지만 6단계의 파이프라인을 사용하는 ARM10이나 7단계를 사용하는 XScale[3]의 경우 분기가 채택될 경우 각각 3사이클과 4사이클의 낭비가 생기게 되므로 분기에 의한 성능저하가 크다고 할 수 있다. 또한 한 사이클에 여러 개의 명령어가 동시에 실행되는 구조의 프로세서들의 경우에는 분기가 채택되었을 경우 무효화되는 명령어들이 더욱 많아지므로 분기에 의한 성능 저하가 매우 크다고 할 수 있다.

분기 예측(branch prediction) 기법은 이러한 분기에 의한 성능 저하를 방지하기 위해 고안되었으며 이를 사용하면 90% 이상[4] 분기에 의한 파이프라인 정지를 막을 수 있다. 이러한 분기 예측 기법을 구현하기 위해서는 하드웨어에 의한 동적 분기 예측을 사용하는데 분기 타겟 버퍼 (BTB : Branch Target Buffer)[5]가 그 구현의 대표적인 방식이다.

본 논문에서는 분기 타겟 버퍼의 태그 메모리를 TLB(Translation Look-aside Buffer)와 공유하도록 함으로써 칩 면적의 감소와 분기 예측 속도의 향상을 가져올 수 있는 새로운 구조의 BTB를 제안한다. 이를 위해 2장에서는 기존 구조의 TLB와 BTB에 대하여 논하고, 3장에서는 본 논문에서 제안하는 구조에 대한 설명과 기존 구조와의 비교를 하고, 4장에서 결론으로 끝을 맺는다.

Ⅱ. 기존 구조의 TLB와 BTB

MMU(Memory Management Unit)는 가상메모리(virtual memory) 개념을 지원하기 위하여 가상주소(virtual address)를 실제주소(physical address)로 바꾸어 주는 동작을 수행하며 이 때 MMU는 페이지 테이블(Page Table)을 참조하여 주소 변환을 한다. 그러나 메모리에 위치하고 있는 페이지 테이블을 주소 변환 시마다 매번 참조한다면 주소 변환 속도가 느릴 수밖에 없기 때문에 TLB를 사용하여 주소 변환 동작을 가속화하게 된다. TLB는 일종의 주소 변환에 대한 캐시의 역할을 하며 최근에 실행한 주소 변환에 대한 정보를 빠르게 제공할 수 있기 때문에 주소 변환 시 매번 메모리를 참조하지 않고도 빠른 속도의 주소 변환에 대한 정보를 제공하게 된다.

그림 1.은 가상주소와 실제주소 및 TLB의 구조를 보여주고 있다. 가상주소와 실제주소는 각각 VPN(Virtual Page Number)과 페이지 오프셋 그리고 PPN(Physical Page Number)과 페이지 오프셋으로 나뉘어 진다. 페이징(paging)을 사용하는 MMU는 가상주소와 실제주소 사이의 주소 변환을 페이지 단위로 한다. 따라서 페이지 오프셋 부분의 주소 변환은 필요 없고 VPN을 PPN으로 주소 변환하면 된다.

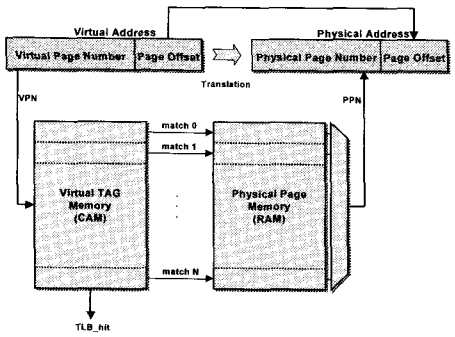

그림 1. TLB를 사용한 주소 변환
Fig. 1. Address Translation Using TLB

TLB는 가상주소를 담고 있는 Virtual TAG 메모리 부분과 이에 대응하는 실제 주소를 담고 있는 Physical Page 메모리의 두 부분으로 나뉘어 있으며 이들은 각각 CAM(Content Addressable Memory)와 SRAM으로 구성되어 있다. 주소 변환이 필요한 가상주소는 Virtual TAG를 참조하며 Virtual TAG이 CAM으로 만들어져 있기 때문에 동시에 모든 엔트리(entry) 중 이 VPN과 같은 엔트리가 있는지를 비교할 수 있다. 이 비교 결과로 얻어지는 신호를 TLB 히트라고 하고, 히트일 경우에는 이에 대응하는 matchx 신호로 Physical Page Memory를 읽어 PPN을 얻고 이를 실제주소로 사용하면 된다. 반대로 TLB 미스인 경우에는 메모리에 위치하는 페이지 테이블을 읽어 주소 변환 정보를 얻은 후 실제주소를 발생시키고 또한 후에 있을 수 있는 주소 변환을 대비하기 위해 이 주소 변환 정보를 TLB에 저장하게 되는데 이 과정을 테이블 워크(table walk)라 한다.

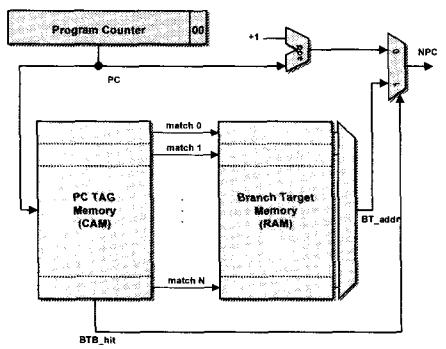

그림 2. 기존 BTB의 구조
Fig. 2. Conventional BTB Architecture

기존의 BTB는 그림 2.와 같은 구성으로 되어 있다. PC(Program Counter)는 현재 파이프라인 단계에서 페치(fetch)할 명령어의 주소이고 페치할 명령어가 32비트라고 하면 주소 중 하위 2비트는 바이트 오프셋으로 사용되지 않는다. 프로세서는 PC를 사용하여 이 주소에 대한 명령어를 명령어 캐시로부터 페치해 오면서 동시에 BTB를 참조하게 된다.

BTB에서 PC 주소는 CAM으로 구성된 PC TAG 메모리의 각 엔트리들과 동시에 비교되고 이 결과에 따라 BTB 히트 신호가 결정된다. BTB 히트가 발생하였다는 것은 이 PC 주소에 의해 페치해 올 명령어가 분기 명령어이기 때문에 이전에 BTB에 저장되었다는 것을 의미하며, 히트 시 이에 해당하는 matchx 신호에 따라 SRAM으로 구성된 Branch Target Memory의 엔트리를 읽는다.

Branch Target 메모리의 각 엔트리는 분기 타겟 주소와 분기 예측 비트(prediction bit) 및 유효 비트(valid bits) 등으로 구성되어 있다. 분기 예측 비트는 이 PC에 해당하는 명령어에 의해 분기가 채택(taken)될 가능성이 어느 정도 인가를 예측하는데 사용하며 채택 가능성이 높다고 판단되고 그것의 유효 비트가 세트되어 있다면 순차적으로 증가된 주소인 PC(=PC+1)를 사용하지 않고 Branch Target 메모리에서 얻어진 분기 타겟 주소를 다음 명령어의 페치 주소로 사용하게 된다.

BTB에 의하여 프로세서는 다음 명령어를 대부분의 경우 정확하게 예측하여 가져올 수 있고, 이에 따라 파이프라인이 정지하는 경우를 최대한 방지할 수 있으므로 BTB 사용 시 프로세서의 성능은 매우 높아지게 되므로 대부분의 프로세서에서는 동적 분기 예측을 지원하기 위한 BTB를 채택하고 있다[6].

Ⅲ. TLB 태그 공유 구조의 BTB

TLB의 Virtual TAG 공유 구조의 BTB는 같은 정보가 TLB의 Virtual TAG 메모리와 BTB의 PC TAG 메모리에 동시에 저장되고 있다는 점에 착안하여 이를 공유할 수 있도록 만든 구조이다.

그림 3.은 본 논문에서 제안하는 TLB 태그 공유 구조의 BTB 구조이다. Virtual TAG 메모리에는 기존의 TLB 경우와 같이 변환될 가상주소인 VPN이 저장되어

있고 Physical Page Memory에도 기존 TLB와 같이 PPN이 담겨 있다. 따라서 PC의 주소 변환은 기존의 TLB와 마찬가지로 동작하게 된다.

그러나 BTB의 PC TAG 메모리에는 PC 주소 부분 중 VPN과 겹치지 않는 부분인 페이지 오프셋(page offset)만을 저장하게 하고, 나머지 주소 부분은 VPN이 아닌 TLB의 Virtual TAG 메모리 엔트리들 중 하나를 가리키는 인덱스를 만들어 이를 대신 저장하도록 한다. 인덱스를 대신 저장함으로써 발생하는 이점은 VPN의 대부분을 저장하는 것보다 저장해야할 많은 수의 비트를 감소시키게 되어 칩 크기를 크게 줄일 수 있게 된다는 것이다. 인덱스를 사용한 분기 타겟 버퍼의 사용은 기존의 방식과는 약간 다르게 동작하게 되는데 그 자세한 동작은 다음과 같다.

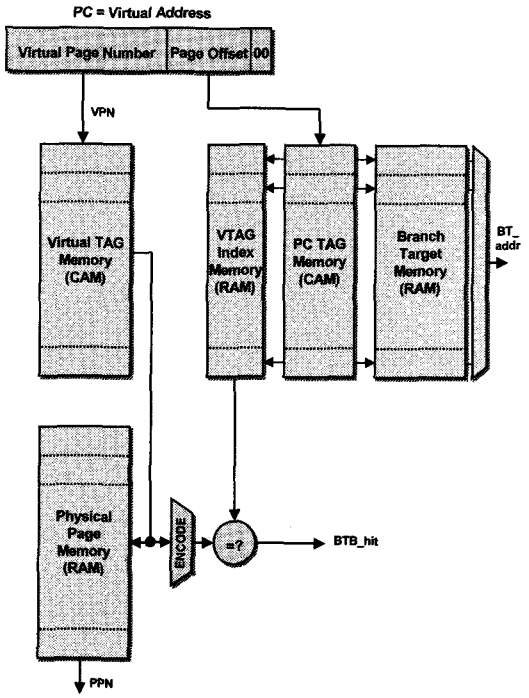

그림 3. TLB 태그 공유 구조의 BTB
Fig. 3. BTB Architecture with Combined TLB Tag

① PC 주소 중 VPN은 Virtual TAG 메모리에 그리고 페이지 오프셋 부분은 PC TAG 메모리에 각각 참조시켜 히트 여부를 판정한다.

② VPN과 일치하는 엔트리가 Virtual TAG 메모리에 있어 히트가 발생하면 이 엔트리의 번호를 인코더(encoder)로 인코딩 한다.

③ 페이지 오프셋 주소와 같은 엔트리가 PC TAG 메모리에 있어 히트가 발생하면 이 엔트리에 해당하는 VTAG Index 메모리와 Branch Target 메모리의 엔트리를 읽는다.

④ ②에서 인코딩된 값과 ③의 VTAG Index 메모리를 읽은 인덱스가 서로 같다면 BTB 히트가 발생한 경우이므로, Branch Target 메모리에서 읽은 분기 예측 비트에 따라 분기 채택 가능성을 예측하고 이 결과에 따라 분기 타겟 주소를 발생시킨다.

공유 태그 구조의 BTB는 기존의 BTB에 비해 메모리를 훨씬 적게 사용한다. Virtual TAG, Physical TAG, Branch Target 메모리들은 기존 구조와 그 크기와 동작이 유사하나 PC TAG의 경우 그 크기가 50%이하로 줄어든다는 장점을 갖게 된다.

예를 들어 XScale의 경우와 같이 32-entry의 TLB, 128-entry BTB, 4Kbytes pages, 32-bit address을 사용한다고 하고 RAM인 경우 단위 셀 당 6개의 트랜지스터, CAM인 경우 단위 셀 당 9개의 트랜지스터를 사용한다고 가정하자. CAM의 경우 SRAM보다 메모리 단위 셀의 트랜지스터 개수가 많이 소요되는 것은 물론이고 이에 필요한 로직과 센스 앰프 등의 개수도 더욱 많기 때문에, 실제로 CAM의 경우 같은 비트 수의 SRAM보다 메모리 셀의 트랜지스터 개수 차이 이상으로 커지게 된다. 따라서 기존 BTB에 사용되는 메모리의 크기가 실제로는 더욱 크지만 여기서는 비교를 단순화하기 위해 메모리 단위 셀의 트랜지스터 개수만을 고려한다.

기존 구조의 BTB에서 PC TAG 메모리 셀의 트랜지스터 수는 30 비트 × 128 엔트리 × 9 트랜지스터로 34560 개만큼의 트랜지스터를 사용해야 하는 반면 공유 구조에서는 PC TAG 메모리 셀로 10 비트 × 128 엔트리 × 9 트랜지스터를 사용하고 VTAG Index 메모리 셀로 5 비트 × 128 엔트리 × 6 트랜지스터만큼을 사용하여 총 15360개만큼의 트랜지스터를 사용하게 된다. 이를 몇 가지 경우의 TLB 엔트리 수 및 BTB 엔트리 수로 계산한 결과가 표 1.에 나타나 있으며 이 결과를 볼 때 메모리 코어로 사용되는 트랜지스터의 숫자는

기존 구조의 절반 이하로 볼 수 있다.

TLB 동작 방식은 기존과 같으므로 TLB의 속도 또한 기존의 구조와 똑같다. 그러나 공유 구조의 BTB는 기존의 BTB와는 다른 방식으로 동작하기 때문에 그 속도에 차이가 있다. 우선 앞에서 설명한 공유 구조의 BTB의 동작 과정 중 ②와 ③은 실제로 동시에 이루어지는 동작이며 Virtual TAG를 엑세스하고 인코딩하는 동작보다는 PC TAG를 읽고 이에 해당되는 VTAG Index 엔트리를 읽는 속도가 더 느리다. 따라서 최대 지연 경로는 이 경로에 인코딩된 값과 비교하여 BTB 히트 신호를 만드는 과정이라 할 수 있다. 이에 비해 기존 BTB 구조의 최대 지연 경로는 PC TAG 메모리를 읽고 Branch Target 메모리의 엔트리를 읽는 경로라 할 수 있다. 이 둘을 비교하면 공유 BTB 구조의 PC TAG 메모리 엔트리 당 비트 수가 훨씬 적고 또한 공유 구조의 VTAG Index 메모리 비트 수가 기존 구조의 Branch Target 메모리 비트 수보다 훨씬 적기 때문에 여기에 5비트를 비교하는 게이트의 속도를 더해도 기존 구조보다 빠르다고 할 수 있다.

표 1. BTB의 메모리 셀 크기 비교
Table 1. Comparison of the size of BTB memories

TLB entries	BTB entries	Conventional BTB (number of tr)	Shared BTB (number of tr)	Percent (%)
32	128	34560	15360	44
32	256	69120	30720	44
64	128	34560	16128	47
64	256	69120	32256	47

Ⅳ. 결 론

본 논문에서는 TLB와 BTB에 저장되는 중복된 정보를 공유하도록 함으로써 BTB의 크기를 줄이고 또한 속도도 빠르게 할 수 있는 구조를 제안하였다. BTB의 분기 적중률은 마이크로프로세서들의 파이프라인 단계 수가 더욱 늘어나면서 그 성능에 있어서 전보다 더욱 중요한 역할을 하고 있으며, 본 논문의 구조를 사용한다면 크기는 많이 커지지 않으면서도 BTB의 엔트리 수를 늘려 분기 적중률을 향상시킬 수 있는 구조를 만들 수 있을 것이다. 또한 최근의 마이크로프로세서는 32비트에서 64비트로 발전하는 경향이 있으며 본 논문의 공유 구조의 BTB는 32비트 보다는 64비트를 사용할 때 이점이 더욱 크다.

※ 반도체설계교육센터(IDEC)의 CAD Tool 지원에 감사드립니다.

참고문헌

[1] K. Thangarajan, and W. Mahmoud, "Survey of Branch Prediction Schemes for Pipelined Processors", IEEE Computer, Vol. 17, No. 3, 2002

[2] Steve Furber, *ARM system-on-chip Architecture Second Edition*, Addison- Wesley

[3] Intel, Intel XScale Microacrchitecture, 2001

[4] C. Lee, I. Chen, and T. Mudge, "The bi-mode branch predictor", Proceedings of MICRO-30, Dec 1997

[5] J. Lee, and A. Smith, "Branch Prediction Strategies and Branch Target Buffer Design", IEEE Computer, Vol. 6, No. 2, 1984

[6] Calder, Brad, Dirk and Emer, "A System Level Perspective on Branch Prediction Architecture Performance", Proceedings of the 28 Intl. Symposium on Microarchitecture, pp. 199-206, 1995

저자소개

이용환(Yong-hwan Lee)

1993. 2. 연세대학교 전자공학과 (공학사)
1999. 2. 연세대학교 전자공학과 (Ph.D.)
1999~2002 하이닉스반도체
2003~2004 삼성전자
2004~ 금오공과대학교 전자공학부
※관심분야 : SoC, 임베디드 시스템 및 소프트웨어, 마이크로프로세서 구조