논문 00-25-11B-13 한국통신학회논문지 '00-11 Vol.25 No.11B

64비트 4-way 수퍼스칼라 마이크로프로세서의 효율적인 분기 예측을 수행하는 프리페치 구조

정회원 문 상 국*, 문 병 인*, 이 용 환**, 이 용 석*

A Prefetch Architecture with Efficient Branch Prediction for a 64-bit 4-way Superscalar Microprocessor

Sangook Moon*, Byung-in Moon*, Yong-hwan Lee**, Yong-surk Lee* ***Regular Members***

요 약

본 논문에서는 명령어의 효율적인 페치를 위해 분기 타겟 주소 전체를 사용하지 않고 캐쉬 메모리(cache memory) 내의 적은 비트 수로 인덱싱 하여 한 클럭 사이클 안에 최대 4 개의 명령어를 다음 파이프라인으로 보내줄 수 있는 방법을 제시한다. 본 프리페치 유닛은 크게 나누어 3 개의 영역으로 나눌 수 있는데, 분기에 관련하여 미리 부분적으로 명령어를 디코드 하는 프리디코드(predecode) 블록, 타겟 주소(NTA : Next Target Address) 테이블 영역을 추가시킨 명령어 캐쉬(instruction cache) 블록, 전체 유닛을 제어하고 가상 주소를 관리하는 프리페치(prefetch) 블록으로 나누어진다. 사용된 명령어들은 SPARC(Scalable Processor ARChitecture) V9에 기준 하였고 구현은 Verilog-HDL(Hardware Description Language)을 사용하여 기능 수준으로 기술되고 검증되었다.

구현된 프리페치 유닛은 명령어 흐름에 분기가 존재하더라도 단일 사이클 안에 4 개 까지의 명령어들을 정확한 예측 하에 다음 파이프라인으로 보내줄 수 있다. 또한 NTA를 사용한 방법은 같은 수의 레지스터 비트를 사용하였을 때 BTB(Branch Target Buffer)를 사용하는 방법과 비교하여 2 배정도 많은 개수의 분기 명령 주소를 저장할 수 있는 장점이 있다.

ABSTRACT

In this paper, in order for efficient instruction prefetch and accurate branch prediction, an NTA(Next Target Address) indexing scheme is used instead of schemes using full branch target address so that up to four instructions can be fetched into the next pipeline in most cases. This prefetch unit can be divided into three blocks; a predecode block which decodes several bits of branch instructions, an instruction cache block with an additional NTA RAM and a prefetch block which controls the whole unit. Instruction set for the simulation is based on the SPARC V9 and the NTA scheme is described and verified with the Verilog-HDL at register transfer level.

Providing up to four instructions per cycle makes the pipeline fully utilized before the execution unit actually needs them. The NTA scheme outperforms the BTB scheme, doubling the number of branch prediction address storage when the same number of register bits is used.

* 연세대학교 전기전자공학과(lizking@dubiki.yonsei.ac.kr)
** 현대전자산업 RAMBUS 설계팀(jaeger@hei.co.kr)
논문번호 : 00241-0629, 접수일자 : 1999년 6월 29일

Ⅰ. 서 론

동시에 여러 개의 명령어를 병렬적으로 처리하는 마이크로프로세서 구조가 발달함에 따라 올바른 주소의 명령어 페치와 보다 정확한 분기 예측이 중요시 되고 있다. 분기 예측이란 분기 명령어가 있을 경우에 그 명령어의 다음 명령어를 예측 하는 것이고 명령어 페치는 현재 수행 중인 명령어 다음 명령어를 메모리 시스템에서부터 불러오는 과정이다[1]. 수퍼스칼라 마이크로프로세서의 파이프라인의 앞 단은 일반적으로 F (Fetch; 명령어들을 명령어 캐쉬로부터 가져와 명령어 버퍼에 저장), A(Align; 명령어 버퍼에 저장되어 있는 명령어들을 정렬), G(Grouping; 명령어들의 디코드를 완결, 유효한 명령어들을 그룹화 하고 디스패치), E(Execution; ALU 결과 처리, 바이패스를 제어), M(Memory access; 메모리 접근) 사이클로 볼 수 있고[2], 본 논문에서도 이러한 가정 하에 프리페치 유닛을 제안하였다. 무조건 분기나 조건 분기 또는 레지스터에 대한 직접 모드나 간접 모드로의 함수 호출이나 복귀 시에 그 명령어들이 디코드 되기 전 까지는 명령어 흐름이 바뀌었는지를 파악할 수 없다. 파이프라인을 보다 효과적으로 이용하기 위해서 프로세서는 일반적으로 가장 최근에 수행된 명령어의 다음 명령어를 페치하게 된다. 만약 디코드 된 명령어가 제어 흐름을 바꾸게 된다면 이전 페치 된 명령어는 무효화 되고 파이프라인은 흐름을 멈추게 된다. 조건 분기에 있어서, 레지스터 간접 모드에 의한 함수 호출이나 복귀에 대한 주소는 E 단계가 끝나기 전 까지는 알 수가 없다. 여기서 분기는 E 단계에서 결정이 되고, M 단계에서는 프로그램 카운터를 갱신하게 된다. 명령어 페치 시 마이크로프로세서는 나중에 확인될 분기가 옳다는 가정 하에 다음 명령어를 선택하게 된다. 만약 분기 예측이 잘못되었다면 얼마간의 파이프라인 정지(stall)와 파이프라인에 존재하는 명령어들의 플러쉬(flush)를 수반한다[3]. 따라서 이렇게 불필요하게 낭비되는 사이클들을 최대한 줄이기 위해서는 효율적인 분기 예측이 필수적이다. 기존에 사용되던 분기 예측 방식은 주로 BTB를 사용한 방식이었다. 이 방식은 캐쉬메모리에서 태그메모리를 인덱스로 사용하는 방식을 따른 것인데 명령어 분기 예측 성능은 우수하지만 분기할 주소의 전체 주소가 필요하기 때문에 하드웨어 자원을 많이 할당해야 한다는 단점이 있다.

본 논문에서는 명령어의 보다 효율적인 페치를 위해 분기 명령어의 실행 시에 BTB에서처럼 분기 타겟 주소 전체를 사용하지 않고 기존 캐쉬 메모리에 NTA 테이블[4]을 별도로 두고 보다 적은 비트수로 인덱싱 하는 방법을 사용하여 한 클럭 사이클 안에 최대 4 개의 명령어를 보내줄 수 있는 구조를 제안하였다. 기준으로 삼은 마이크로프로세서에서의 명령어 수행 형태는 in-order-issue, out-of-order-completion 명령어 캐쉬의 구조는 16k-byte 2-way set-associative 이고 프리페치 블록에서는 다음 명령어의 주소를 예측하기 위하여 동적 분기 예측(dynamic branch prediction)을 사용한다[5]. 명령어들은 SPARC V9에 기준 하였고 Verilog-HDL을 사용하여 기능 수준으로 기술되고 검증되었다.

Ⅱ. 프리페치 유닛의 구조

일반적인 파이프라인 구조에서의 가장 큰 약점은 보통의 프로그램에서 수행된 명령어의 15-30% 정도를 구성하는 분기 명령어들에 의한 파이프라인 정지와 명령어 플러쉬로 인하여 생기는 손실이다[6]. 이러한 문제의 해결은 보통 최적화 컴파일러의 도움을 받는 소프트웨어적인 방법과 자주 분기되는 주소를 미리 저장해 두었다가 택하는 하드웨어적인 방법이 있다[7]. 본 논문에서는 동적 분기 예측을 사용하여 분기 타겟을 예측하는 하드웨어적인 방법을 채택하여 보다 빠르고 정확한 프리페치를 수행하도록 한다.

프리페치 유닛은 파이프라인의 F단계를 수행한다. 그림 1에서 processor execution unit을 제외한 부분은 프로세서와 상호작용을 이루는 프리페치 유닛의 블록도이다. 프리페치 유닛은 크게 보아 3 개의 블록으로 나누어지는데 이는 각각 다음과 같다.

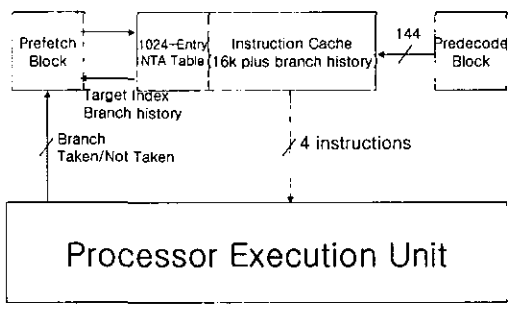


그림 1. 프리페치 유닛의 블록도

- 프리디코드 블록
- 명령어 캐쉬
- 프리페치 제어 블록

1. 프리디코드 블록

4 개의 명령어들을 디코드하고 이들을 적절하게 묶어서(grouping) 기능 유닛으로 이슈(issue)하는 과정은 많은 지연 시간을 요하기 때문에 보통 이를 여러 사이클에 걸쳐서 처리해야 한다. 따라서 본 논문에서는 캐쉬 미스(cache miss)시 라인 채우기 동작을 할 때 분기에 관한 명령어들을 부분적으로 디코드 하고 또한, 명령어 버퍼로 보내질 때 부분적으로 디코드하여 최종적으로 그룹화 단계에서 명령어를 디코드 할 수 있게 함으로써 디코드 시간에 의한 부담을 덜 수 있도록 한다.

외부 캐쉬에서 요청된 데이터가 들어올 때, 버스의 폭(width)이 128비트이기 때문에 캐쉬 라인에 채워지는 명령어 8 개를 보내주기 위해 두 사이클을 소요하게 된다. 이때 NTA 테이블을 참조하기 위해서는 단일 사이클에 4 개씩 들어오는 명령어들을 어느 정도 미리 디코드(predecode) 하여 디코드 부담을 덜 수 있도록 한다. 이에 따라 프리디코드 블록에서는 한 사이클에 명령어 4 개씩 분기 명령어 여부를 판별하고 분기의 종류를 부분적으로 디코드 하여 각 명령어마다 3비트로 구성된 프리디코드 영역을 가지게 하였다.

표 1은 3비트로 저장되는 프리디코드 결과를 나타낸다. 프리디코드 블록은 명령어들을 6가지로 구분하는데 이에는 무조건 분기 명령어, 조건 분기 명령어, 일반 JMPL 명령어, 귀환 JMPL 명령어, 저장 명령어와 분기가 아닌 일반 명령어가 속한다. 이 중 JMPL 명령어는 DCT(Delayed Control Transfer) 동작이므로 이를 빠르게 처리할 필요가 있어 구분

표 1. 3비트 프리디코드 영역

predecode_out	type
000	unconditional branch
001	conditional branch
010	other instructions
011	store instruction
100	normal JMPL
101	return JMPL

되며 저장 명령어를 구분하는 이유는 저장시 이미 페치된 명령어를 플러쉬하지 않은 명령어도 포함되기 때문이다. 프리디코드된 코드는 명령어와 함께 명령어 캐쉬에 저장되며 다음 단계의 지연을 최소화 할 수 있도록 한다.

2. 명령어 캐쉬의 구조

명령어 캐쉬는 실제 주소로 인덱스 되고 실제 태그와 비교된다. 라인의 크기는 32 바이트이고 16-kbyte 의 가상 2-way set associative 방식을 사용하였다. 기준으로 사용되는 마이크로프로세서에서의 페이지 크기가 8 kbyte 이므로 가상 주소의 하위 13 비트는 MMU에서 번역될 필요가 없고 실제 주소처럼 쓰일 수 있다. 캐쉬 메모리가 가상적인 2-way 방식을 사용하여 하위 비트만으로는 인덱스를 제대로 할 수 없기 때문에 한 비트로 이루어진 NTA 테이블의 셋 예측 비트와 캐쉬 라인마다 존재하는 셋 예측 비트가 사용되어 인덱스 될 셋을 결정하게 된다. 셋을 예측하여 인덱스 하는 방법은 다음과 같다.

① 분기 명령어의 경우: NTA 테이블의 셋 예측 비트 사용
② 비분기 명령어의 경우: 이전 캐쉬 라인의 셋 비트를 그대로 사용

명령어 8 개로 이루어진 캐쉬 라인에서 최대 4 개까지의 명령어가 명령어 버퍼로 나가게 된다. 본 논문에서는 하드웨어를 간단하게 하기 위하여 페치하고자 하는 명령어 그룹이 두 개의 캐쉬 라인에 걸쳐 있으면 두 번째 라인의 명령어들은 페치 되지 않는다. 만일 인덱스가 캐쉬 라인의 끝에서 3 개 중 하나의 명령어를 가리킨다면 캐쉬는 그 명령어부터 그 라인 끝까지만 페치 한다. 명령어 캐쉬는 크게 4 개의 영역으로 나누어져 있다. 각 캐쉬 라인이 유효한가를 나타내는 라인 유효 비트를 포함한 태그 램 영역, 명령어들을 담고 있는 데이터 램 영역, 미리 부분적으로 디코드 한 정보를 담고 있는 프리디코드 램 영역, 마지막으로 다음 페치할 주소를 가리키는 분기 타겟과 그에 관계된 분기 예측 비트 및 LRU (Least Recently Used) 비트, 셋 예측 비트를 담고 있는 NTA 테이블 램 영역으로 구성된다(그림 2)

하드웨어로 분기 예측을 구현하기 위한 일반적인 방법은 BTB를 사용하여 가장 최근에 수행된 분기에 대한 정보를 저장하는 것이다. 보통 BTB는 명령어의 주소로 액세스 되는데 만약 명령어가 분기 명령어라면, BTB는 예측비트와 타겟 주소를 내보내

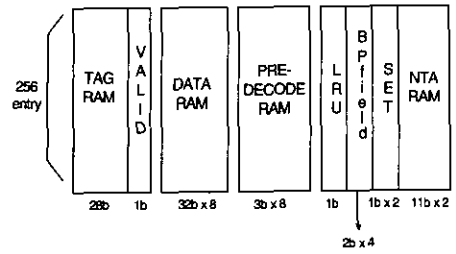
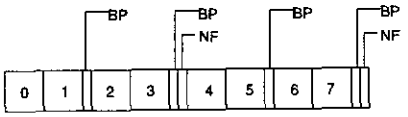


그림 2. 명령어 캐쉬 램 영역(way 0)

게 된다. BTB가 캐쉬와 다른 점은 주소 태그에 대해 엔트리가 하나밖에 없고 분기 명령어들은 공간적 지역성(spatial locality)이 거의 없기 때문에 예측도를 높이기 위해서는 더욱 더 큰 크기의 BTB가 요구된다는 것이다[4]. 또한 BTB의 엔트리를 늘린다 해도 예측 실패(misprediction)의 영향 때문에 효율은 크게 높아지지 않는다. 이러한 이유로 본 논문에서는 캐쉬 메모리에 8 개의 명령어로 이루어진 라인 한 개당 2 개씩에 해당하는 개수의 엔트리를 가진 NTA 테이블을 두어 분기 예측 정보를 포함시킨다(그림 3). 캐쉬의 크기가 16kbyte 이므로 1024 개의 엔트리가 할당되고 각 엔트리는 명령어의 하위 주소로 인덱스 된다. 이렇게 함으로써 페치 제어 블록이 올바른 정보를 가지고 있는 캐쉬 블록을 페치할 때마다 명령어 디코더나 실행부에서의 처리를 기다리지 않고 다음 페치 할 블록을 쉽게 가져올 수 있게 된다.



BP : Branch Prediction(2-bits wide)
NTA : Next Target Address(11-bits wide)
0-7 : Instructions(32-bits wide)

그림 3. 캐쉬 라인 구조

명령어 주소의 인덱스가 명령어 캐쉬의 태그와 데이터에 액세스 될 동안 상위 주소는 명령어 TLB에 동시에 액세스 된다. 명령어 캐쉬의 히트의 경우, NTA 테이블의 인덱스는 곧바로 다음 사이클에 명령어 캐쉬로 귀환(feed back)되고 미스의 경우에는 외부 캐쉬로의 라인 필을 요청한다. 외부 캐쉬와 연결되어 있는 버스의 폭이 128비트이기 때문에 256비트로 구성된 한 캐쉬 라인을 가져오기 위하여 라인 필 방식은 data requested first 방식을 따른다. 이 방식은 요청된 데이터가 들어있는 블록을 먼저 내부 캐쉬로 보내는 방법이다[9]. 명령어 흐름에 있어 분기 명령어를 만나면 NTA 테이블을 액세스하고 프리페치 제어 블록은 계산된 분기 타겟의 인덱스를 갱신 시키고 다른 명령어라면 순차적인 다음 주소를 수행한다. 이렇게 페치되어 나온 4 개까지의 명령어들은 부가적인 정보와 함께 G단계에서 디코드 되기 위해 명령어 버퍼에 순서대로 쌓이게 된다.

3. 프리페치 제어 블록의 구조

프리페치 제어 블록에서는 동적 분기 예측과 NTA 테이블의 갱신을 수행하면서 프리페치 동작을 제어한다. 프리페치 제어 유닛은 F, A, G, E, M 각 단계의 가상 주소와 분기 예측 비트를 담는 레지스터 블록과 내부 제어 회로, 리턴 주소 스택(return address stack)으로 구성되어 있다. 제어 유닛은 캐쉬 라인 필 동작이 수행될 때마다 분기 예측 비트와 프리디코드 영역 비트를 참조하여 타겟 주소를 계산하고 NTA 테이블을 갱신한다. 동적 분기 예측은 2 비트의 예측 비트가 나타내는 4가지 상태에 따라 최근에 수행된 분기 주소를 가지고 다음 분기를 예측한다. 동적 분기 예측을 구현하기 위해 2 개의 명령어마다 하나의 2 비트 분기 예측 비트를 할당하는데 그 이유는 보통 분기 명령어가 지연 명령어(delayed instruction)를 수반한다는 특성 때문이다. 따라서 분기 명령어 다음에 바로 분기 명령어가 올 확률은 매우 작다고 생각할 수 있다. 따라서 이러한 가정 하에 2 개의 명령어마다 한 개씩의 분기 예측 필드를 할당하면 명령어 캐쉬 전체의 가능한 분기 여부를 예측할 수 있게 된다.

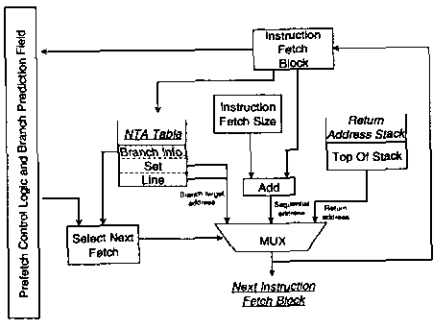
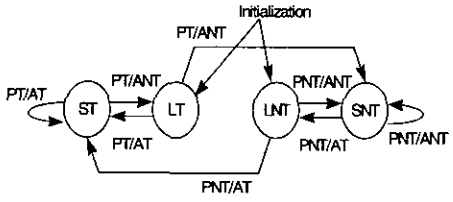


그림 4. 타겟 주소 예측 제어 블록도

그림 4는 프리페치 제어 유닛에서 보낼 주소를 제어하는 것을 도식화 한 것이다. 프리페치 제어 로

직은 분기 예측 비트와 프리디코드 비트를 참조하여 다음 주소를 결정하게 된다. 분기가 예측되지 않는다면 이전 주소에 페치된 명령어의 크기를 더한 순차적 주소를 다음 주소로 사용한다. 분기가 예측될 경우 return 명령어가 아니라면 최근에 분기했던 주소를 다음 주소로 사용하면 되지만 return에 의한 분기라면 이 분기타겟 주소는 자주 바뀌게 되므로 귀환 주소 스택으로부터 나오는 주소를 다음 주소로 사용하게 된다.

프로세서가 분기 명령어를 만나면, 무조건 분기일 경우에는 상관이 없지만 그것이 조건 분기일 경우 어떤 주소의 명령어 블록을 미리 페치해 놓아야 할지 결정해야 한다. 이런 경우에 정확한 분기 예측이 필요하다. 실제 분기가 결정될 때 까지 미리 페치된 명령어들은 순서대로 실행되어 간다. 분기 예측이 정확했다면 프로그램의 흐름은 정상적으로 진행되어 가지만 분기 예측이 실패했다면 그 전까지의 페치된 명령어들은 플러쉬 되고 파이프라인의 손실(penalty)이 발생한다. 이러한 경우에 E 단계에서 실행된 결과의 정수 조건 코드가 파이프라인의 M 단계에서 결정되기 때문에 4 사이클의 손실이 일어나게 된다.



PT : Predicted Taken	ST : Strongly Taken
PNT : Predicted Not Taken	LT : Likely Taken
AT : Actual Taken	SNT : Strongly Not Taken
ANT : Actual Not Taken	LNT : Likely Not Taken

그림 5. 분기 상태 천이도

분기 예측 상태를 표시하기 위해 2 비트로 구성된 4 가지 상태를 가진 분기 예측 비트를 사용한다. 2 비트를 사용한 분기 예측은 프로그램 상의 루프 수행의 구현에 적합하다. 1 비트, 즉 2 가지 상태(taken, not_taken)만으로 분기 예측을 구현한다면 루프를 수행할 때마다 분기 예측을 실패하게 되는데 비해 2 비트를 사용하여 예측을 한다면 루프를 빠져 나올 때 한 번만 분기 예측에 실패를 하기 때문이다. 그림 5은 분기 예측 상태 천이도이다. 각 상태는 LT(Likely Taken), LNT(Likely Not Taken), ST(Strongly Taken), SNT(Strongly Not Taken)로 표시한다. 외부 캐쉬로부터 명령어들이 라인 필 될 때마다 분기 예측 비트 영역을 초기화 해야 하는데, SPARC-V9에는 최적화 컴파일러가 프로그램을 컴파일 할 때 분기명령어에 정적 예측 비트를 설정할 수 있게 하였다. 이 비트들에 따라 예측 영역의 초기화 시에 LT또는 LNT로 초기화 시키고 이러한 비트가 설정되지 않은 명령어들은 LNT로 초기화 시키도록 하였다. 이는 왜냐하면 분기가 선택(LT)되었다고 초기화를 시킨다면 선택된 분기의 타겟 주소를 계산하는 추가된 사이클이 필요하기 때문이다. 이렇게 초기화된 분기 예측 영역은 M 단계의 분기 유닛(branch unit)에서 결정된 분기의 선택 유무에 따라서 분기가 결정된다. 만약 분기 예측 실패라면 올바른 주소를 캐쉬 메모리에 갱신 시키고 분기 명령어 이후에 수행되고 있던 명령어들과 명령어 버퍼에 들어있던 명령어들을 모두 플러쉬 시키고 이후 파이프라인 단계에서부터 올바른 명령어들을 페치할 수 있도록 하였다.

CALL이나 JMPL 명령어에 의한 서브루틴으로부터의 빠른 복귀를 위하여 4 개의 엔트리로 이루어져 귀환 주소를 저장할 수 있는 리턴 주소 스택을 따로 할당한다. CALL 명령이나 JMPL 명령이 확인될 때마다 그 명령어의 가상 주소에 해당하는 프로그램 카운터의 값이 스택에 저장된다. 서브루틴에서의 리턴이 수행될 때마다 리턴 주소는 스택의 맨 위에서 취해지고 스택은 팝(pop) 된다. 본 논문에서의 마이크로프로세서는 *rs1* = %o7 의 값을 가지는 jmpl 명령어는 일반적인 서브루틴에서의 복귀로 보고 *rs1* = %i7 의 값을 가지는 jmpl 명령어는 최종 루틴(leaf subroutine)에서의 복귀로 간주한다. 리턴 명령이 들어올 때마다 NTA 테이블의 주소도 같이 스택의 맨 위 값과 비교된다. 만약 비교해서 같지 않으면, 리턴 주소 스택에 우선권이 주어지게 된다.

분기 예측을 구현하기 위해 프로세서는 수행된 순서대로 예측된 분기 명령어들의 목록을 가지고 있어야 한다. 이는 FIFO(First-In First-Out) 형태의 버퍼에 저장되고 F, A, g, e, M 단계의 분기 정보를 가지고 있어야 하므로 엔트리는 5개를 할당하였다. 그림 6에는 분기 정보 저장 버퍼의 구조가 나와 있다. 각 버퍼 엔트리에는 해당하는 분기 명령어의 주소와 그에 대한 전체 타겟 주소와 그 분기에 대한 분기 예측 정보가 담겨 있고 이는 분기 결정 유닛에서 보낸 주소와 비교된다.

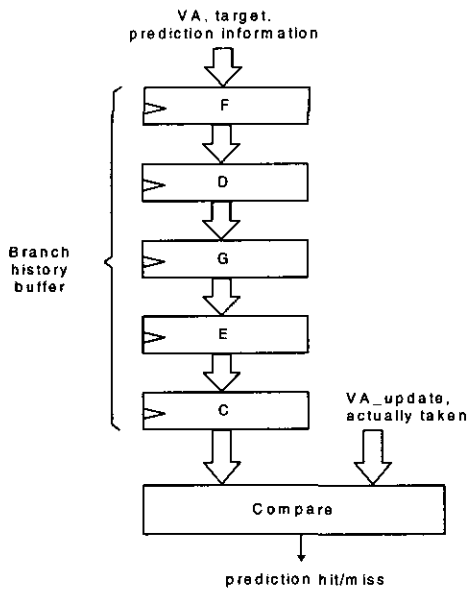


그림 6. 분기 정보 저장 버퍼의 구조

프로세서의 명령어 수행 형태가 in-order-issue, out-of-completion 이기 때문에 분기 명령도 실행되는 프로그램의 흐름을 따라 이슈 된 순서대로 수행된다. 분기 명령어가 실제로 실행되었을 때 프로세서는 버퍼에서 처음으로 나올(맨 처음에 수행된) 분기 저장 정보와 실제로 실행된 분기 명령어의 정보를 비교한다. 분기 예측의 성공 여부는 다음 조건에 의해 판별된다.

① 실행되고 있는 분기 명령어에 대한 타겟의 캐쉬 엔트리 안에서의 위치는 분기 정보 저장 버퍼의 마지막에서 두 번째 엔트리에 해당하는 분기 명령어의 주소와 일치하여야 한다.
② 실행되는 분기 명령어에 대한 분기 예측 정보와 정보 저장 버퍼의 마지막 엔트리에 포함된 분기 예측 정보가 서로 같아야 한다.
③ 실행되는 분기의 위치와 정보 저장 버퍼의 가장 오래된 엔트리의 명령어의 위치가 같다면 예측된 타겟 주소는 분기 결정 유닛에서 결정된 다음 명령어의 주소와 일치해야 한다. 만일 일치하지 않는다면 이는 실행된 분기가 선택 된다고 예측 되었지만 실제로는 선택이 안된 경우이다.

세 번째 조건의 경우는 예측된 분기 명령은 순차적이 아닌 계산된 타겟 주소를 가지고 있고 실제 수행된 분기 명령어는 순차적인 주소를 가지고 있는 경우이다. 이 경우에 예측된 타겟 주소는 타겟 인덱스와 다음 블록의 태그로 구성되어 있기 때문에 이들의 비교는 원래의 캐쉬 태그가 바뀌었다는 정보를 포함하게 된다. 이는 캐쉬 미스를 의미하고 이 때 외부 캐쉬로의 라인 필을 요청한다. 레지스터 간접 모드인 분기의 경우 참조 하는 레지스터의 값이 실행 단계에서 바뀌었을 수도 있기 때문에 최종적으로 타겟을 비교해 보아야 실제 분기가 올바르게 예측되었는지 알 수가 있다.

위 경우를 모두 만족시키지 못한다면 프리페치 제어 블록은 분기 예측 실패라고 보고 분기 유닛에서는 실행 유닛에서 생성된 올바른 분기 타겟 주소를 제어 블록으로 보내주어 명령어 캐쉬를 갱신한다.

Ⅲ. 시뮬레이션 및 결과 고찰

본 논문에서는 효율적인 분기 예측을 수행하는 4-way 수퍼스칼라 마이크로프로세서에서 사용하기에 적합한 프리페치 유닛의 구조를 제시하고 동작을 확인하기 위해 Verilog-HDL로 기술한 후 테스트 벡터를 입력으로 가한 뒤 회로의 동작을 기능 수준(behavioral level)으로 시뮬레이션을 수행하였다. 회로 설계의 복잡성을 줄이기 위해 그림 1에서 보인 프리페치 유닛의 구조를 3개의 기능별 모듈(module)로 나누고 이를 계층적 구조에 의해 상호 연결하여 시뮬레이션을 진행하였다. 최상위 모듈은 아래의 3 개 하위 모듈로 구성된다.

- 프리디코드 모듈
- 페치 제어 모듈
- 명령어 캐쉬 모듈

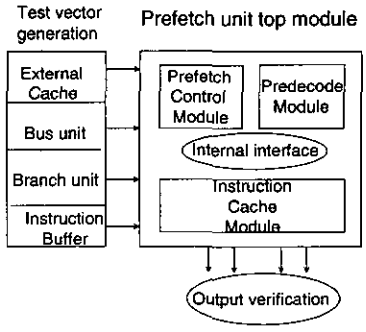


그림 7. 프리페치 유닛의 시뮬레이션 환경

프리페치 유닛의 시뮬레이션 환경은 그림 7과 같다. 기능 검증은 다음과 같은 방법으로 하였다. 먼저 각 유닛 사이에서 입출력 신호의 타이밍(timing)

을 정하고 외부 유닛에서 입력되는 신호를 테스트 벡터에서 가정으로 만들어 내고 프리페치 유닛에서 나오는 출력 신호가 올바른 값을 갖는지 확인한다. 또한 내부 신호의 흐름도 올바른 값과 올바른 타이밍을 갖는지 확인한다.

1. 프로그램에 의한 기능 검증

HDL로 설계된 프리페치 유닛의 성능 평가를 위하여 여러 가지 기능이 포함된 어셈블리 프로그램을 이용하였다. 검증 방법은 4 개의 어셈블리 언어로 작성된 테스트 프로그램을 gcc로 컴파일 하여 실행 파일을 생성시키고 이 파일을 실제 SPARC- V9 아키텍처로 구현된 UltraSPARC workstation에서 실행시켜 주소의 trace를 생성해 내었다. 그 후 이 trace에서 분기가 실제로 채택되었는가에 대한 정보를 추출할 수 있도록 하여 이를 HDL 시뮬레이션 시 사용할 수 있도록 하였다. 이는 분기 결정 유닛이 없기 때문에 분기가 채택되었는지를 알 수 없기 때문이다. 한편 컴파일된 실행 파일은 HDL 상의 메모리에 올려지고 이를 시뮬레이션 하면서 실제 분기 채택 여부는 앞에서 추출되어 있는 정보를 사용하게 된다. 시뮬레이션에 사용한 프로그램은 bubble sort, matrix multiply, calculation, Hanoi tower 등이다. 이들의 시뮬레이션 결과로 평균 분기 예측 성공율과 사이클 당 명령어 페치율이 표 2에 나타내었다. 표에서 보이는 바와 같이 분기 예측 성공율은 기존에 사용된 방식과 비슷한 성능을 보인다.

2. 성능 향상도

표 3은 본 논문에서 설계한 효율적인 분기 예측을 수행하는 프리페치 유닛과 BTB를 사용하여 설계한 프리페치 유닛을 비교한 내용이다. 4 개의 테스트 프로그램을 수행한 본 논문의 프리페치 유닛의 실행 결과와 자료를 참조한 BTB에 대한 결과를 비교하였다[4].

표 2. 평균 분기 예측 성공율 및 페치율

수행 프로그램	bubble	matrix	calcul	hanoy	평균
명령어 페치율	3.33	3.00	3.67	3.50	3.37개
분기 성공율	91%	95%	88%	91%	91.25%

본 논문에서는 효과적인 분기 예측을 위하여 NTA 테이블 영역을 명령어 캐쉬와 따로 두어 인덱스로 사용한다. 명령어 캐쉬의 엔트리가 256 개씩 2 세트가 존재하므로 거기에 해당하는 NTA 필드 램은 1024 개의 엔트리가 필요하다. 캐쉬 메모리의 크기가 16-kbyte이므로 명령어의 개수는 4096 개로 볼 수 있다. 이렇게 볼 때 본 논문의 프리페치 유닛은 명령어 4 개당 하나의 분기를 파이프라인 멈춤 없이 수행할 수 있다. 이러한 결과는 256 개의 엔트리를 사용한 경우의 BTB와 비슷한 수행율을 보인다.

BTB에는 분기가 선택되었을 때 분기 명령어의 주소와 분기 예측 비트 영역과 분기 타겟의 주소가 포함되어 있다. 본 논문의 실제 주소 영역이 41비트이고 가상 주소 영역이 44비트이므로 256 엔트리를 가진 BTB 구조의 프리페치 유닛에서 차지하는 램 영역은 실제 주소의 태그 28비트와 가상 주소의 타겟 64비트를 계산하면 256 x (28 + 64) = 23552 비트가 요구되는 데 비해 본 논문에서 NTA 테이블을 사용하여 구현한 프리페치 유닛에서 차지하는 램 영역은 분기 예측 영역이 같다고 가정하였을 경우 11 x 1024 = 11264 비트를 차지한다. 이는 동일한 면적의 하드웨어를 사용할 경우 거의 2배로 많이 분기 예측 엔트리를 할당할 수 있다는 것을 의미한다.

표 3. 성능 평가 비교[4]

	128 BTB	256 BTB	설계된 프리페치 유닛
Misfetch penalty	0.46	0.4	0.41
Register bits used	11776	23552	11264
면적대 성능비	1.80	1.00	2.06

Misfetch penalty는 다음의 (식1)에 의해 구하였고 면적대 성능비는 (식2)를 사용하고 256 엔트리의 BTB를 기준으로 하였다.

$$Misfetchpenalty = \frac{\text{미스된}NTA\text{테이블}}{\text{사용된분기의}NTA\text{테이블}} \quad (1)$$

*주소계산사이클

$$\text{면적대성능비} = \frac{1 - misfetchpenalty}{registerbitsused} \quad (2)$$

이에 따르면 본 논문에서 구현한 프리페치 유닛의 페널티는 주소 계산 사이클이 한 사이클이므로 0.41로 계산된다. 이는 256 엔트리의 BTB를 사용하였을 경우와 비슷한 수행율을 보이지만 위에서 보인 바와 같이 하드웨어를 절감할 수 있다.

Ⅳ. 결 론

본 논문에서는 한 클럭 사이클에 4 개까지의 명령어를 수행할 수 있는 수퍼스칼라 마이크로프로세서에서 효율적인 분기 예측을 수행하는 프리페치 유닛을 설계하였다. 프리페치 유닛은 프리디코드 블록, 명령어 캐쉬, 프리페치 제어 블록을 포함하고 있다.

명령어 흐름에 있어 정확한 분기 예측은 무엇보다도 중요하다. 본 논문에서는 정확한 분기 예측을 위하여 명령어 2 개당 하나의 분기 예측 영역과 명령어 4 개당 하나씩의 NTA 필드 영역을 두어 효과적인 분기 예측과 타겟 주소로의 분기를 가능하게 하였다.

프리페치 유닛의 아키텍쳐 검증 및 성능 평가를 위하여 HDL로 기술된 마이크로프로세서 모델을 사용하였고 기능 수준과 로직 수준의 시뮬레이션을 수행하였다. 기능 검증을 위하여 프리페치 유닛의 여러 동작을 인위적으로 유발 시키는 테스트 벡터를 작성하여 수행하였고 이의 성능 평가를 위하여 어셈블리 언어로 작성된 4 개의 테스트 프로그램이 사용되었다. 성능 평가의 결과 분기 예측은 약 91%의 히트율을 보였고 사이클 당 명령어 페치율은 3.37 개를 기록하였다.

이 결과는 본 프리페치 유닛이 4-way 수퍼스칼라 마이크로프로세서에서 명령어를 사이클 당 4 개씩 이슈 하는데 적합함을 보여준다. 설계된 프리페치 유닛은 기존에 존재하는 동일 성능의 BTB 구조를 사용하는 프리페치 구조에 비해 하드웨어의 약 50%를 절약할 수 있다.

참 고 문 헌

[1] John L. Hennessy, David A. Patterson, *Computer Organization and Design : The Hardware/Software Interface*, Morgan Kaufmann Publishers Inc., pp364-426, 1994

[2] Mike Johnson, *Superscalar Microprocessor Design*, Prentice Hall, New Jersey, pp1-7, pp9-30, 1991

[3] B.Calder and D. Grunwald, "Next Cache Line and Set Prediction", *Proc. 22nd Ann. Int'l Symp. Computer Architecture, IEEE CS Press*, 1995, pp. 287-297

[4] Johnny K. F., Lee Alan Jay Smith, "Branch Prediction Strategies and Branch Target Buffer Design", *IEEE Computer*, pp6-21, Jan. 1984

[5] Scott McFarling, John Hennessy *Reducing the Cost of Branches*, IEEE Computer, pp396-403, 1986

[6] 안상준, "수퍼스칼라 마이크로프로세서용 프리페치 유닛에 관한 연구", *연세대학교 대학원 전자공학과 석사학위 논문*, 1995

[7] Marc Tremblay and J. Michael O'Connor, "UltraSPARC I : A Four-Issue Processor Supporting Multimedia", *IEEE Micro* April, 1996, pp.42-49

[8] Greenley, el al, *UltraSPARC : The Next Generation Superscalar 64-bit SPARC, SPARC Technology* Business-Sun Microsystems, Inc., 1063-6390, 1995, IEEE

[9] Yong S. Lee, "A Secondary Cache Controller Design for a High-End Microprocessor", *IEEE Journal of Solid-State Circuits*, Vol.27, No.8, pp1141-1146, August 1992

문 상 국(Moon Sangook) 정회원

1995년 2월 : 연세대학교 전자공학과 졸업
1997년 2월 : 연세대학교 전자공학과 석사
1997년 3월~현재 : 연세대학교 전자공학과 박사과정

<주관심 분야> VLSI 설계, 암호용 프로세서 설계

문 병 인(Moon Byung-In) 정회원

1995년 2월 : 연세대학교 전자공학과 졸업
1997년 2월 : 연세대학교 전자공학과 석사
1997년 3월~현재 : 연세대학교 전자공학과 박사과정

<주관심 분야> 마이크로프로세서 설계, SMT

이 용 환(Lee Young-Hwan) 정회원

1993년 2월 : 연세대학교 전자공학과 졸업
1995년 2월 : 연세대학교 전자공학과 석사
1999년 2월 : 연세대학교 전자공학과 박사
1999년 3월~현재 : 현대전자산업 RAMBUS 설계팀 선임연구원 근무중
<주관심 분야> 캐쉬 메모리 설계, 메모리 아키텍처

이 용 석(Lee Yong-Surk) 정회원

1973년 5월 : 연세대학교 전기공학과 졸업
1977년 2월 : University of Michigan Ann Arbor 석사
1981년 3월 : University of Michigan Ann Arbor 박사
1993년~현재 : 연세대학교전기전자공학과 교수 재임중
<주관심 분야> 마이크로프로세서 설계