공학 연구논문집 제25권 제2호 pp. 211~225, 울산대학교
Journal of Engineering Research Vol. 25, No. 2, pp. 221~225, 1994.
University of Ulsan.

BTB를 이용한 분지명령어처리

구자록
전자계산학과

<요 약>

BTB(Branch Target Buffer)가 파이프라인 프로세서에서 분지예측을 행함으로써 분지명령어의 성능저하를 줄이는데 효율적으로 사용될 수 있음은 여러 논문에서 확인되었다. 이 논문에서는 그러한 단점을 줄이기 위해 상호보완적인 기법을 제시함으로써 BTB를 최적으로 활용할 수 있도록 하였다.

Branch Processing using BTB

Koo, Ja-Rok
Dept. of Computer Science

<Abstract>

A Branch Target Buffer(BTB) can reduce the performance penalty of branches in pipelined processors by predicting the path of the branch and caching informations used by the branch. Currently several branch prediction strategies based on the BTB are proposed, but each strategy has some defects. Thus we propose a combined technique using BTB information.

---

1. 개 요

근래 대부분의 컴퓨터들은 성능을 향상시키기 위해 파이프라이닝을 이용한다. 파이프라이닝은 각각의 연산의 실행을 여러 단계(일명, 파이프라인 단계)로 나눠서 행한다. 그 대표적인 예로 5개의 단계를 갖는 파이프라인을 보면, 명령어 페치, 명령어 해석, 오퍼렌드 페치, 실행, 그리고 결과치 저장으로 되어있다. 파이프라인을 통해 실행되는 시간당 명령어의 수는 가장 느린 파이프라인 단계에 의해서 제한되기때문에 각각의 파이프라인 단계의 실행시간은 서로가 거의 일치해야한다는 사실이 매우 중요하다. 이러한 파이프라인의 최적의 실행상태에서는 5개의 명령어가 동시에 구동된다는 것이다. 이러한 최적의 경우에서는 - 현실적으로 거의 불가능하지만 - 파이프라인을 이용하지 않는 같은 형태의 프로세서보다 5배의 빠른 성능을 산출한다. [CHRIS 93, SMITH 82]

분지명령어는 파이프라이닝의 정상적인 흐름을 방해함으로써 파이프라인의 성능을 떨어뜨린다. 하나의 분지명령어를 실행하기 위해서, 프로세서는 우선 명령어가 분지명령어인지를 인식해야하고, 그 분지명령어가 분지될 것인지를 결정하고, 또한 분지될 명령어의 주소를 계산하여, 그 분지명령어가 분지되면 분지될 명령어의 주소로부터 명령어를 페치해야 한다. 그런데, 파이프라인 프로세서는 분지방향이 결정되기 전에 분지의 방향을 알아야할 필요성(더 많은 명령어를 페치해야함)이 있기때문에, 분지명령어는 파이프라인의 성능을 떨어뜨린다. 그래서, 분지명령어가 나타나면, 프로세서는 두가지 선택을 해야한다. 더 많은 명령어를 페치하기 전에 분지명령어의 실행이 끝날때까지 기다리거나, 경우에 따라서는 잘못된 방향을 택할지라도 계속해서 명령어를 페치하는 경우이다. 물론, 이 두가지 모두 파이프라인의 성능을 저하시킨다. 그런데, 두 번째 경우가 첫번째 경우보다 평균적으로 덜 성능저하를 가져온다. 왜냐하면, 어느 정도의 확률로 페치된 명령어가 올바른 방향에서 올 수 있기 때문이다.

분지타겟버퍼(Branch Target Buffer, BTB, 이하 BTB 사용)는 분지명령어에 관한 정보를 저장하고 분지명령어의 방향을 예측함으로써 분지명령어의 성능저하를 줄여야 하는 경우에 사용될 수 있다.

최근에 발표된 분지명령어처리에 관한 대부분의 논문들은 벤치마크(Benchmark)나 추적데이타(Trace data)를 사용하여 특정 아키텍쳐에 대한 분지처리기법과 성능평가를 기술하고 있다. [LEE 84, MCF 86] 그런데, 이러한 논문들은 어느 특정 아키텍쳐에 관한 성능을 묘사하고 있기때문에 새로운 아키텍쳐의 설계나 기존의 아키텍쳐에서 실증되지 않은 새로운 워크로드(Workload)의 측정에서는 제한적인 사용이 불가피하다.

따라서 이 논문은 구현단계의 변수인 파이프라인의 길이와 분지명령어의 유효주소 계산장치의 위치를 고려하면서도 분지기법의 성능을 계산할 수 있는 여러가지의 방법에 대한 수학적인 모델의 유도를 제시하려 한다.

2. 성능평가 모델

이 논문에서 제시하고자하는 분지명령어처리 모델은 파이프라인 프로세서 설계에서 고려되어야할 다른 중요한 요소들의 영향을 단순화시켜, 주로 분지에 따르는 시간지연에 의해서만이 영향을 받을 수 있는 파이프라인의 정상적인 흐름을 가정하였다. 우선 분지명령어처리에서 고려해야할 사항은, 분지명령어 구조, 워크로드, 그리고 파이프라인 구조이다.

2.1 분지명령어 구조

분지명령어에는 무조건분지(Unconditional Branch)와 조건분지(Conditional Branch)가 있다. 이름에서 의미하는 바와 같이 무조건분지는 실행이 이뤄질 때 항상 분지가 일어나는 것이고, 조건분지는 분지조건(참 또는 거짓)에 따라 분지방향을 양자택일하는 것을 의미한다. 이 논문에서는 조건분지와 무조건분지를 다같이 고려하기로 한다. 조건분지에도 좀 더 세부적으로 나누면, 조건(Condition)을 먼저 테스트하여 결과(Condition Code)를 저장하는 명령어와 그 테스트결과를 이용하여 분지의 방향을 결정하는 명령어로 하나의 쌍을 이루는 구조와, 또 다른 하나는 이러한 두 가지 연산을 하나의 명령어에서 처리하는 구조를 들 수 있다.

2.2 워크로드(Workload)

대부분의 성능평가에 관한 자료들은 시뮬레이터(Simulator)나 벤치마크 및 커널(Kernel)을 실행시킬 때 하드웨어 모니터(Hardware Monitor)에 의한 클럭횟수(Clock Count)를 측정함으로써 얻어진다. 이러하 자료들은 프로세서의 설계를 증명하는 데는 효과적이나 프로세서의 설계를 하는 과정에서, 설계자가 워크로드나 파이프라인에서의 변경된 사항에 대한 즉각적인 평가를 행할 시에는 별로 유용하지가 못하다. 그래서 앞에서도 지적했듯이 워크로드를 대변할 수 있는 변수(Parameter)를 사용하는 수학적인 모델을 제시한다. 사용되는 변수에는 다음의 두 변수가 있다.

P_b = 명령어가 분지명령어일 확률
P_{bt} = 실행된 분지명령어가 분지되는 명령어일 확률

Lee[LEE 84]는 실험적으로 수집한 P_b, P_{bt}의 정연된 값을 표-1과 같이 제시하고 있다.

표-1. Fractions of branches, taken(T) and not taken(N) and fraction of branches overall(r).

	IBM CPL	IBM BUS	IBM SCI	IBM SUP	DECC PDP11	CDC 6400	AVERAGE
T	0.640	0.657	0.704	0.540	0.738	0.778	0.676
N	0.360	0.343	0.296	0.460	0.262	0.222	0.324
r	0.317	0.189	0.105	0.376	0.388	0.079	0.242

2.3 파이프라인 구조

파이프라인 아키텍쳐모델에서 고려되어야 할 변수는 분지정보가 시간단위로 알려지는 파이프라인의 단계들이다. 명령어페치 단계인 단계-1에서 시작하여 분지결과가 결정되는 마지막 단계-n에까지 일련번호를 부여하면 다음과 같다:

단계-n" : 명령어가 해석되는 단계,
단계-n′ : 분지될 명령어의 유효주소가 결정되는 단계,
단계-n : 분지결과가 결정되고 조건코드(Condition Code)가 정해지는 단계.

2.4 성능평가 모델

성능평가모델은 예약표(Reservation Table)라 불리우는 시간표(Timing Diagram)를 보여줌으로써 전개된다. 이 예약표는 수평축으로 클럭사이클의 수를, 수직축으로 파이프라인의 단계를 나타낸다. 기본 성능평가의 모델은 해당 분지처리기법을 사용하여 기본블럭에 대한 정상상태의 명령어당 클럭수(CPI:Clocks Per Instruction)를 계산한다. 이 모델은 다음과 같이 수식으로 표현된다[CRAGON 92]:

기법이름 = 1 + P_b[WABD],
WABD = 가중평균분지 지연시간(단위, 클럭사이클수).

즉, 지연이 일어나지 않는 파이프라인에서는 명령어당 필요로하는 클럭의 수는 한 개이다. 여기서 하나의 분지명령어는 다른 분지명령어들과 적어도 한개 이상의 블럭에 의해서 분리되어있다는 가정을 한다.

2.5 기본 기법(BL: Baseline Strategy)

가장 간단한 기법(그림-1)으로서 분지의 결과가 단계-n에서 알려질 때까지 인라인 명령어열(Inline Instruction Stream)의 처리를 행한다. 그래서, 그 다음 클럭사이클에서 분지될 명령어가 페치되거나 인라인 명령어열의 처리가 계속 진행된다. 이 기법은 다른 분지처리기법의 성능을 비교하는데 사용된다.

t

	1	2	3	4	5	6	7	8
1	B	i	i+1	i+2				
n"=2		B	i	i+1	i+2			
n′=3			B	i	i+1	i+2		
n =4				B	i	i+1	i+2	

(a) P_{bt} = 0

시간지연 = 0

t

	1	2	3	4	5	6	7	8
1	B	i	i+1	i+2	T			
n"=2		B	i	i+1		T		
n′=3			B	i			T	
n =4				B				T

(b) P_{bt} = 1

시간지연 n − 1

t : 시간(Time)
B : 분지명령어(Branch Instruction)
i, i+1, i+2 : 분지명령어 다음의 일련의 명령어들(Instructions)
T : 분지된 명령어(Target Instruction)

그림-1. 기본 기법

분지가 일어나지 않은 경우(P_{bt}=0)에는, 인라인 명령어들을 계속해서 페칭하며, 시간지연은 0이다. 분지가 일어나는 경우(P_{bt}=1)에는, 분지된 명령어(T)를 시간 5에서 페치하며, 시간지연은 n-1이다. 따라서 WABD는 다음과 같다:

$$WABD_{bl} = P_{bt}(n-1) + (1-P_{bt})(0) = P_{bt}(n-1).$$

또한 단위시간당 클럭의 수는 다음과 같다:

$$BL = 1 + P_b[P_{bt}(n-1)].$$

2.6 분지예측 기법(Branch Prediction Strategy)

개요에서도 언급했듯이 사전에 분지방향을 알고있다고 가정하면 분지처리에 있어서 성능을 향상시킬 수 있을 것이다. 분지예측 기법에는 하드웨어상에서 구현되어 더 이상의 변경이 있을 수 없는 정적인 기법(Static Strategies)과 프로그램이 실행됨에 따라 수시로 변경이 가능한 동적인 기법(Dynamic Strategies)이 있다.

2.6.1 일반적인 분지예측 모델(General Prediction Model)

이 모델은 P_b와 P_{bt}, 그리고 F_{pt}의 데이타에 근간을 둔다.

F_{pt} = 분지명령어가 분지되리라고 예측하는 확률.

또한 이 모델에서는, 분지가 일어날 것인지 아닌지를 예측하고, 실제로 분지가 일어났는지 안 일어났는지를 판단할 수 있기때문에 그림-2와 같이 4 가지의 경우(사건)의 확률을 고려한다.

		Actual: Taken	Actual: Not Taken	
Predict	Taken	j($P_{t/t}$)	k($P_{t/nt}$)	$\Sigma=F_{pt}$
	Not Taken	m($P_{nt/t}$)	l($P_{nt/nt}$)	$\Sigma=(1-F_{pt})$
		$\Sigma= P_{bt}$	$\Sigma= (1-P_{bt})$	$\Sigma= A_p$

$P_{t/t}$: 분지예측(Taken Prediction) 뒤에 실제 분지될 확률
$P_{t/nt}$: 분지예측 뒤에 실제 분지가 안될 확률
$P_{nt/t}$: 분지불예측(Not-Taken Prediction) 뒤에 실제 분지될 확률
$P_{nt/nt}$: 분지불예측 뒤에 실제 분지가 안될 확률
A_p : 예측의 정확성(Accuracy of Prediction)
$1 = P_{t/t} + P_{t/nt} + P_{nt/t} + P_{nt/nt}$.

그림-2. 예측 사건 시간지연 및 확률

WABD는 각 사건과 지연 카르노맵(Karnaugh map)의 곱의 합으로 계산되어진다.

$$WABD = jP_{t/t} + kP_{t/nt} + lP_{nt/nt} + mP_{nt/t}.$$

그런데, 분지예측모델에는 두가지로 나눌 수 있다: 예측모델-A(PA)와 예측모델-B(PB).

2.6.2 분지예측 모델-A

PA는 다음 그림-3(a)에 나타나 있듯이, 단계-n"에서 해석되고, 단계-n'에서 분지될 명령어의 유효주소가 계산된 후에 분지될 명령어가 시간-4에서 페치된다. 이때 인라인 명령어인 i와 i+1은 포기되어진다. 따라서 분지명령어가 분지될 경우, 시간지연은 j=(n'-1)이 된다. PA 기법의 WABD는 다음과 같다:

$$WABD_{pa} = P_{t/t}(n'-1) + (P_{t/nt} + P_{nt/t})(n-1).$$

명령어당 클럭의 수는 다음과 같다:

$$PA = 1+P_b[P_{t/t}(n'-1) + (P_{t/nt} + P_{nt/t})(n-1)].$$

t

	1	2	3	4	5	6	7	8
1	B	i	i+1	T				
n"=2		B	i		T			
n'=3			B			T		
n =4				B			T	

(a) $P_{t/t}$

시간지연-j=n'-1

t

	1	2	3	4	5	6	7	8
1	B	i	i+1	T	i			
n"=2		B	i			i		
n'=3			B				i	
n =4				B				i

(b) $P_{t/nt}$

시간지연 -k=n-1

t

	1	2	3	4	5	6	7	8
1	B	i	i+1	i+2	T			
n"=2		B	i	i+1		T		
n'=3			B	i			T	
n =4				B				T

(c) $P_{nt/t}$

시간지연 -m=n-1

t

	1	2	3	4	5	6	7	8
1	B	i	i+1	i+2				
n"=2		B	i	i+1	i+2			
n′=3			B	i	i+1	i+2		
n =4				B	i	i+1	i+2	

(d) $P_{nt/nt}$

시간지연-l=0

그림-3. 분지예측 모델-A

2.6.3 분지예측 모델-B

PB의 시간지연도 비슷한 방법으로 결정된다. 이 분지예측 기법의 예약표는 그림-4에 나타나 있다. 이 기법에서는 분지명령어가 해석되고 유효주소가 계산되면, 인라인 명령어들은 결코 포기되어지지 않는다. 그들은 분지의 결과가 알려지고 나서야 포기되어진다. 보는 바와 같이, 명령어당 실행 클럭 수를 줄임으로써 성능향상을 꾀한다.

이 기법의 WABD는 다음과 같다:

$$WABD_{pb} = P_{t/t}(n'-1) + P_{t/nt}(n-n') + P_{nt/t}(n-1).$$

명령어당 클럭의 수는 다음과 같다:

$$PA = 1+P_b[P_{t/t}(n'-1) + P_{t/nt}(n-n') + P_{nt/t}(n-1)].$$

t

	1	2	3	4	5	6	7	8
1	B	i	i+1	T				
n"=2		B	i	i+1	T			
n′=3			B	i		T		
n =4				B			T	

(a) $P_{t/t}$

시간지연-j=n′-1

t

	1	2	3	4	5	6	7	8
1	B	i	i+1	T	i+2			
n"=2		B	i	i+1		i+2		
n′=3			B	i	i+1		i+2	
n =4				B	i	i+1		i+2

(b) $P_{t/nt}$

시간지연 -k=n-n′

t

	1	2	3	4	5	6	7	8
1	B	i	i+1	i+2	T			
n"=2		B	i	i+1		T		
n'=3			B	i			T	
n =4				B				T

(c) $P_{nt/t}$

시간지연-m=n-1

t

	1	2	3	4	5	6	7	8
1	B	i	i+1	i+2				
n"=2		B	i	i+1	i+2			
n'=3			B	i	i+1	i+2		
n =4				B	i	i+1	i+2	

(d) $P_{nt/nt}$

시간지연 -l=0

그림-4. 분지예측 모델-B

3. BTB를 이용한 동적인 분지예측

3.1 BTB의 일반적인 구조

앞에서도 언급되었듯이, 분지될 명령어를 좀더 빨리 페치하기 위해서, BTB를 이용한 예측을 페치단계에서 실행하는데, 여기에는 분지될 명령어의 주소를 BTB에 저장하는 방법과 분지될 명령어 자체를 BTB에 저장하는 방법이 있다.

BTB의 구조는 그림-5에 나타나 있는 바와 같이, 명령어열의 페치를 위한 Cache의 접근과 동시에 BTB에 접근한다. BTB의 명령어들의 주소는 해싱(Hashing)을 이용하거나 내용연상검색(Associative Searching)을 이용하여 찾아낸다.

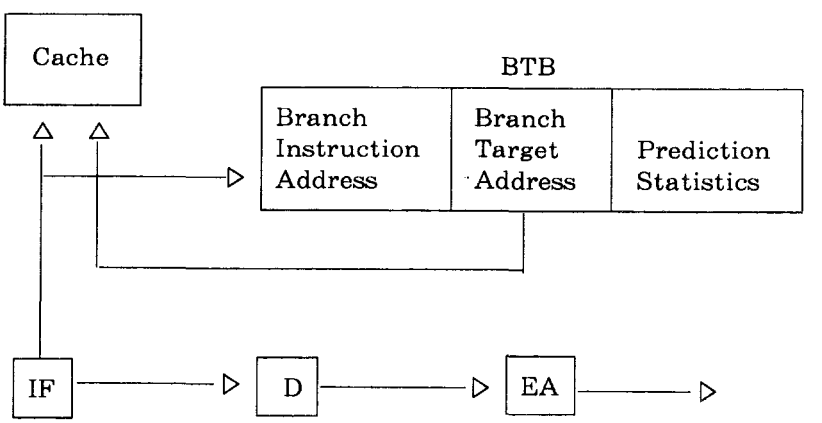

그림-5. BTB(Branch Target Buffer)

3.2 분지될 명령어 주소를 저장하는 방법

이 기법은 BTB에 분지될 명령어의 주소를 저장하는 기법으로 크게 6가지의 경우가 발생할 수 있다.

1) BTB 미스 : 이전에 분지명령어가 실행된 적이 없음.
2) BTB 히트 : 분지예측, 실제 분지, 정확한 주소.
3) BTB 히트 : 분지예측, 실제 분지, 비정확한 주소.
4) BTB 히트 : 분지예측, 실제 미분지.
5) BTB 미스 : 분지미예측, 실제 분지.
6) BTB 미스 : 분지미예측, 실제 미분지.

이러한 경우, 분지명령어의 과거실행결과에 기초한 예측기법이 사용되면, 분지명령어의 결과가 나타난 후에 BTB의 과거 실행결과필드를 갱신하는 시간이 필요하다. 먼저 BTB 미스인 경우를 생각하면, 분지명령어에 대한 첫 접근이 일어날 때는 BTB에서의 미스(Miss)가 발생하게되는데, 정적인 분지예측 모델-A를 이용한다. BTB 미스가 실행된 후에는 방금 실행된 분지명령어의 정보를 BTB에 저장해야 한다. 그래서, 미스의 경우 갱신사이클(Update Cycle: u)이 WABD모델에 가산되어져야 한다.

t

	1	2	3	4	5	6	7	8	9
1	B	i	i+1	T	U	T+1			
n"=2		B	i		T	U	T+1		
n′=3			B			T	U	T+1	
n =4				B			T	U	T+2

시간지연-j=n′-1+u

(a) $P_{t/t} = P_{bt}$

t

	1	2	3	4	5	6	7	8	9
1	B	i	i+1	T	i	U			
n"=2		B	i			i	U		
n′=3			B				i	U	
n =4				B				i	U

시간지연-k=n-1+u

(b) $P_{t/nt} = 1 - P_{bt}$

그림-6. BTB 미스의 경우

그래서, BTB미스의 경우 WABD는 다음과 같다:

$$WABD = n-1-P_{bt}(n-n')+u.$$

BTB의 히트(Hit)인 경우에는, 하나의 분지명령어가 실행이 이루어지면, BTB에는 3개의 요소가 존재한다: 1) 분지명령어의 주소, 2) 분지될 명령어의 유효주소, 그리고 3) 예측에 이용할 과거실행 정보.

그림-7에 나타나 있는 바와 같이 t_1에서 분지명령어가 페치되면, 동시에 BTB가 접

근되어진다. 그래서, 분지명령어가 존재하면 분지될 명령어의 주소가 분지될 명령어를 페치하기 위해서 메모리시스템에 전달되어 진다. 그런데, 메모리시스템 설계상의 문제점때문에 분지될 명령어가 파이프라인의 페치 단계에 오기전에 약간의 클럭사이클(d)이 요구되어진다.

t

	1	2	3	4	5	6	7	8	9
1	B	i	T	T+1	U	T+2			
n"=2		B	i	T	T+1	U	T+2		
n′=3			B	i	T	T+1	U	T+2	
n =4				B		T	T+1	U	T+2

(a) $P_{t/t}$

시간지연-j=d+u

t

	1	2	3	4	5	6	7	8	9
1	B	i	T	T′	U	T′+1			
n"=2		B	i		T′	U	T′+1		
n′=3			B			T′	U	T′+1	
n =4				B			T′	U	T′+1

(a-a) $P_{t/t}$

시간지연-j=n′-1+u

t

	1	2	3	4	5	6	7	8	9
1	B	i	T	T+1	U	i+1			
n"=2		B	i	T		U	i+1		
n′=3			B	i		T	U	i+1	
n =4				B	i			U	i+1

(b) $P_{t/nt}$

시간지연-k=n-1-d+u

t

	1	2	3	4	5	6	7	8	9
1	B	i	i+1	i+2	U	T			
n"=2		B	i	i+1		U	T		
n′=3			B	i			U	T	
n =4				B				U	T

(c) $P_{nt/t}$

시간지연-m=n-1+u

t

	1	2	3	4	5	6	7	8	9
1	B	i	i+1	i+2	U	i+3			
n"=2		B	i	i+1	i+2	U	i+3		
n′=3			B	i	i+1	i+2	U	i+3	
n =4				B	i	i+1	i+2	U	i+3

시간지연-l=u

(d) $P_{nt/nt}$

그림-7. BTB 히트의 경우

BTB 히트의 경우 WABD는 다음과 같다:

$$WABD=u+P_{t/t}\{d+P_{ac}(n'-d-1)\}+P_{t/nt}(n-d-1)+P_{nt/t}(n-1).$$

따라서, BTB 히트 및 미스를 동시에 고려하면, 이 기법의 WABD는 다음과 같다:

$$WABD_{ba}=u+P_{mc}\{n-1-P_{bt}(n-n')\}+(1-P_{mc})[P_{t/t}\{d+P_{ac}(n'-d-1)\}+P_{t/nt}(n-d-1)+P_{nt/t}(n-1)].$$

3.3 분지될 명령어 자체를 저장하는 방법

이 기법은 위의 기법과는 달리 BTB에 분지될 명령어의 주소를 저장하는 것이 아니라 분지될 명령어 자체를 저장하는 것이다. 일명, BTC(Branch Target Cache)라고도 한다. 이 모델은 BTC에서 히트(Hit)가 일어나면, 분지명령어의 페칭사이클 뒤에 인라인 명령어나 분지될 명령어가 곧이어 페칭됨을 가정한다. 이 기법은 현재까지의 분지명령어의 실행결과 정보에 근간을 둔 예측을 행한다. 분지명령어에 대한 첫 접근이 일어날 때는 BTC에서의 미스(Miss)가 발생하게되는데, 이 경우는 위 기법의 미스에 해당한다.

그래서, BTC미스의 경우 WABD는 다음과 같다:

$$WABD=n-1-P_{bt}(n-n')+u.$$

BTC의 히트(Hit)인 경우에는, 예약표는 그림-8과 같다.

t

	1	2	3	4	5	6	7	8	9
1	B	T	T+1	T+2	U	T+3			
n"=2		B	T	T+1	T+2	U	T+3		
n′=3			B	T	T+1	T+2	U	T+3	
n =4				B	T	T+1	T+2	U	T+3

시간지연-j=u

(a) $P_{t/t}$

t

	1	2	3	4	5	6	7	8	9
1	B	T	T+1	T+2	U	i			
n"=2		B	T	T+1		U	i		
n′=3			B	T			U	i	
n =4				B				U	i

(b) $P_{t/nt}$

시간지연-k=n-1+u

t

	1	2	3	4	5	6	7	8	9
1	B	i	i+1	i+2	U	T			
n"=2		B	i	i+1		U	T		
n′=3			B	i			U	T	
n =4				B				U	T

(c) $P_{nt/t}$

시간지연-m=n-1+u

t

	1	2	3	4	5	6	7	8	9
1	B	i	i+1	i+2	U	i+3			
n"=2		B	i	i+1	i+2	U	i+3		
n′=3			B	i	i+1	i+2	U	i+3	
n =4				B	i	i+1	i+2	U	i+3

(d) $P_{nt/nt}$

시간지연-l=u

그림-8. BTB 히트의 경우

BTC 히트의 경우, WABD는 다음과 같다:

$$WABD=u+(P_{nt/t}+P_{t/nt})(n-1).$$

따라서, BTC 히트 및 미스를 동시에 고려하면, 이 기법의 WABD는 다음과 같다:

$$WABD_{bi}=u+P_{mc}\{n-1-P_{bt}(n-n')\}+(1-P_{mc})(P_{nt/t}+P_{t/nt})(n-1).$$

3.4 분지될 명령어의 주소 및 그 자체를 저장하는 방법

위에서 설명한 기법인 BTB에 분지될 명령어의 주소를 저장하는 것이나 분지될 명령어 자체를 저장하는 것은, BTB내의 분지명령어에 관한 정보(즉, 분지명령어의 주소, 예측정보, 분지될 명령어의 주소, 그리고 분지될 명령어 그 자체) 및 그에 따른 분지명령어의 분지성능저하(단위, 사이클횟수)에 의해[CHRIS 93], 분지될 명령어의 주소 및 명령어 그 자체를 모두 저장하는 기법보다 성능이 떨어짐을 확인할 수 있다. 또한 위의 기법중 하나인 분지명령어 자체를 저장하는 것은 분지명령어 주소를 유도할 수 없기때문에 사실상 불가능한 기법이다. 이 논문에서 제안하고자하는 모델의 WABD를 유도함에 있어서, 우선 BTB 미스인 경우에는 위의 기법에서 사용한 정적

인 분지예측 모델-A 대신, 정적인 분지예측 모델-B를 사용함으로써 분지성능저하를 줄인다. (그림-9)

t

	1	2	3	4	5	6	7	8	9
1	B	i	i+T	T	U	T+1			
n"=2		B	i	i+1	T	U	T+1		
n′=3			B	i		T	U	T+1	
n =4				B			T	U	T+1

시간지연-j=n-1+u

(a) $P_{t/t} = P_{bt}$

t

	1	2	3	4	5	6	7	8	9
1	B	i	i+1	T	i+2	U			
n"=2		B	i	i+1		i+2	U		
n′=3			B	i	i+1		i+2	U	
n =4				B	i	i+1		1+2	U

시간지연-k=n-n′+u

(b) $P_{t/nt} = 1-P_{bt}$

그림-9. BTB 미스의 경우

그래서, BTB미스의 경우 WABD는 다음과 같다:

$$WABD=n-n'+u+P_{bt}(2n'-n-1).$$

BTC의 히트(Hit)인 경우에는, 예약표는 위의 기법 그림-7과 같다.

BTB 히트의 경우 WABD는 다음과 같다:

$$WABD=u+(P_{nt/t}+P_{t/nt})(n-1).$$

따라서, BTB 히트 및 미스를 동시에 고려하면, 이 기법의 WABD는 다음과 같다:

$$WABD_{bai}=u+P_{mc}\{n-n'+P_{bt}(2n'-n-1)\}+(1-P_{mc})(P_{nt/t}+P_{t/nt})(n-1).$$

4. 분지예측 기법들의 성능 비교

여기서는 위에서 언급한 여러 분지처리 예측기법들간의 비교를 행한다. 우선, 정확한 데이타를 구하기 힘든 경우 추정치를 산출하여 사용해야한다. 예를 들면 위의 그림-2에 의해서, $P_{t/t} = P_{bt}A_p$로 대신할 수 있다. 그래서 추정가능식을 나열한다:

$$P_{t/t} = P_{bt}A_p,$$
$$P_{nt/t} = P_{bt}(1-A_p),$$
$$P_{t/nt} = (1-A_p)(1-P_{bt}),$$
$$P_{t/t} = A_p(1-P_{bt}).$$

따라서, 다시 한번 위의 식들을 정리하면 다음과 같다:

$$WABD_{bl} = P_{bt}(n-1),$$
$$WABD_{pa} = P_{bt}A_p(n'-1)+\{(1-A_p)(1-P_{bt})+P_{bt}(1-A_p)\}(n-1),$$

$WABD_{pb} = P_{bt}A_p(n'-1)+(1-A_p)(1-P_{bt})(n-n')+P_{bt}(1-A_p)(n-1)$,

$WABD_{ba} = u+P_{mc}\{n-1-P_{bt}(n-n')\}+(1-P_{mc})\{A_pP_{bt}\{d+P_{ac}(n'-d-1)\}+(1-A_p)\{n-1-d(1-P_{bt})\}$,

$WABD_{bi} = u+P_{mc}\{n-1-P_{bt}(n-n')\}+(1-P_{mc})(1-A_p)(n-1)$,

$WABD_{bai} = u+P_{mc}\{n-n'+P_{bt}(2n'-n-1)\}+(1-P_{mc})(1-A_p)(n-1)$.

다음과 같이 위 식들의 중요한 변수들의 값을 가정한 뒤, 그래프를 그린다:

$u = d = 1$,
$P_{mc} = P_{ac} = 0.02$,
$P_{bt} = 0.676$,
$n = 4$,
$n' = 3$.

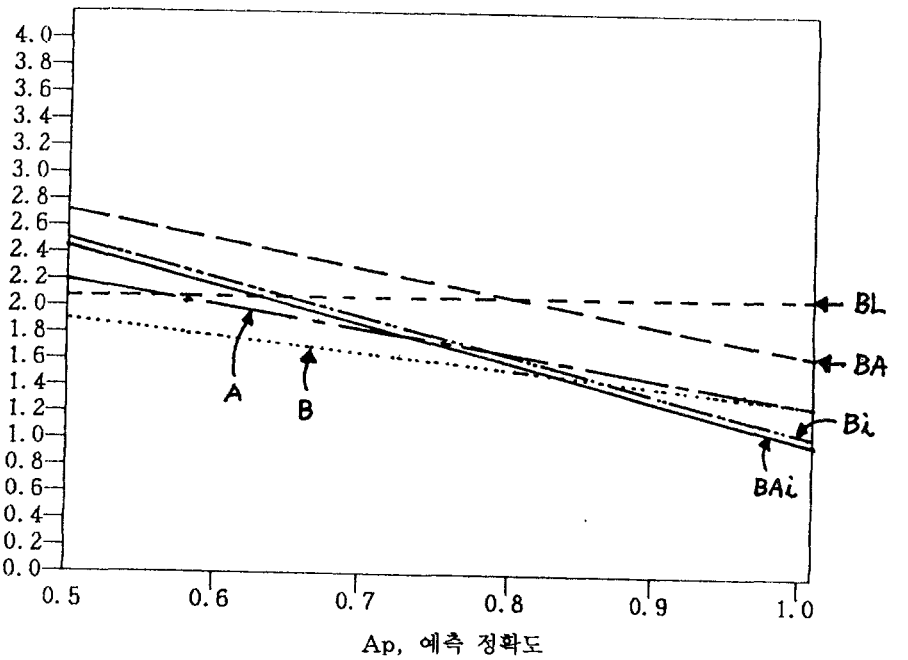

그림-10. 분지예측기법들의 성능 비교

5. 결 론

BTB(Branch Target Buffer)가 파이프라인 프로세서에서 분지예측을 행함으로써 분지명령어의 성능저하를 줄이는데 효율적으로 사용될 수 있는 상호보완적인 기법을 제시함으로써 BTB를 최적으로 활용할 수 있도록 한다. 분지명령어의 주소, 분지정보, 분지될 명령어의 주소, 분지될 명령어 그 자체를 동시에 BTB에 저장함으로써 분지명령어 처리시간지연을 최대한 극소화시켰다. 물론 BTB의 크기에 대한 비교는 분지명령어 처리시간과의 상호 관계에 밀접한 연관을 갖고 있는 고로 다음 과제로 남긴다.

참고문헌

[BRIAN 91] Brian K. Bray, M. J. Flynn, "Strategies for Branch Target Buffers", in the Proceedings of the 24th Annual International Symposium on Microarchitecture, Nov. 1991, pp. 42-50.

[CHRIS 93] Chris H. Perleberg, Alan Jay Smith, "Branch Target Buffer Design and Optimization", IEEE Transaction on Computer, vol. 42, no. 4, April 1993, pp. 396-412.

[CRAGON 92] Harvey G. Cragon, Branch Strategy Taxonomy and Performance Models, IEEE Computer Society Press, 1992.

[SMITH 82] Alan J. Smith, "Cache Memories", Computing Surveys, vol. 14, no. 3, Sept. 1982, pp. 473-530.

[LEE 84] Lee, J. K. F. and Smith, A. J., "Branch Prediction Strategies and Branch Target Buffer Design", IEEE Computer, vol. 21, no. 7, Jan. 1984, pp. 6-22.

[MCF 86] McFarling, S. and Hennessy, J., "Reducing the Cost of Branches", in the 13th Annual International Symposium on Computer Architecture, 1986, pp. 396-403.