프로그램 상의 제어 독립성을 이용한 분기 예상 실패 복구 메커니즘

(Branch Misprediction Recovery Mechanism That Exploits Control Independence on Program)

윤 성 룡 † 이 원 모 †† 조 영 일 †††

(Sung-Lyong Yoon) (Won-Mo Lee) (Young-Il Cho)

요 약 제어 독립성은 슈퍼스칼라 프로세서에서 명령어 수준 병렬성을 향상시키기 위한 중요한 요소로 작용하고 있다. 분기 예측기에서 예상이 잘못된 경우에는 예상한 분기 방향의 명령어들을 무효화시키고 올바른 분기 방향의 명령어들을 다시 반입하여 수행해야 한다.

본 논문에서는 컴파일 시 프로파일링을 통한 정적인 방법과 프로그램상의 제어 흐름을 통해 동적으로 제어 독립적인 명령어를 탐지해서 분기 명령어의 잘못된 예상으로 인해 무효화되는 명령어를 효과적으로 감소시켜 프로세서의 성능을 향상시키는 메커니즘을 제안한다.

SPECint95 벤치마크 프로그램에 대해 기존의 방법과 본 논문에서 제안한 방법 사이의 사이클 당 수행된 명령어 수를 분석한 결과, 4-이슈 프로세서에서 2%~7%, 8-이슈 프로세서에서 4%~15%, 16-이슈 프로세서에서 18%~28%의 성능 향상을 보이고 있다.

키워드 : 명령어 수준 병렬성, 제어 독립성, 분기 예상 메커니즘, 슈퍼스칼라 프로세서, 분기 예상 실패 복구 메커니즘

Abstract Control independence has been put forward as a new significant source of instruction-level parallelism for superscalar processors. In branch prediction mechanisms, all instructions after a mispredicted branch have to be squashed and then instructions of a correct path have to be re-fetched and re-executed.

This paper presents a new branch misprediction recovery mechanism to reduce the number of instructions squashed on a misprediction. Detection of control independent instructions is accomplished with the help of the static method using a profiling and the dynamic method using a control flow of program sequences.

We show that the suggested branch misprediction recovery mechanism improves the performance by 2~7% on a 4-issue processor, by 4~15% on an 8-issue processor and by 8~28% on a 16-issue processor.

Key words : Instruction-Level Parallelism, Control Independence, Branch Prediction Mechanism, Superscalar Processor, Branch Misprediction Recovery

1. 서 론

현대의 고성능 마이크로프로세서는 여러 개의 명령어를 동시에 병렬로 수행시켜 성능을 향상시키고 있다. 이런 명령어 수준 병렬성(Instruction Level Parallelism: ILP)을 향상시키기 위해서 슈퍼스칼라 프로세서에서는 반입된 명령어를 유지, 관리하는 명령어 윈도우(Instruction Window)를 가지고 있다. 매 사이클마다 프로세서는 명령어 윈도우로부터 종속 관계가 없이 동시에 실행이 가능한 명령어들을 선택하여 수행시킨다. 따라서, 프로세서의 성능을 향상시키기 위해서는 충분한 명령어 윈도우를 유지해 동시에 수행할 수 있는 명령어

† 비 회 원 : SK텔레텍 S/W개발
slyoon@sktelecom.com
†† 비 회 원 : 경인여자대학 컴퓨터정보학부 교수
wmlec33@hanmail.net
††† 정 회 원 : 수원대학교 정보공학대학 컴퓨터학과 교수
yicho@mail.suwon.ac.kr
논문접수 : 2001년 1월 3일
심사완료 : 2002년 5월 2일

를 많이 찾아낼 수 있어야 한다.

ILP의 성능을 저해시키는 중요한 요소 중 하나로 제어 종속성(Control Dependence)이 있다. 분기 명령어 이후에 반입되어야 할 명령어들은 분기 명령어의 결과가 구해질 때까지 지연되어야 한다. 이러한 제어 종속성을 제거하기 위해서 많은 프로세서에서 분기 예상기법(Branch Prediction Mechanism)을 사용하고 있다[1][2][3][4]. 분기 명령어의 예상된 경로에 있는 명령어를 미리 수행시켜 실제 결과가 구해질 때까지의 지연 시간을 효과적으로 감소시킬 수 있다. 그러나, 모든 분기 명령어를 정확하게 예상할 수는 없다. 따라서, 현재 사용중인 분기 예상기법에서는 분기 명령어의 예상이 틀렸을 경우 분기 명령어 이후에 반입된 명령어들을 모두 무효화시키고 올바른 방향의 명령어들을 재 반입해서 수행시키는 복구 메커니즘(Recovery Mechanism)을 사용한다. 이는 명령어 윈도우의 효율성을 제한하고 하드웨어 자원의 낭비가 심해져 프로세서 성능 감소를 가져온다.

본 논문에서는 분기 예상이 실패했을 경우 잘못 예상된 분기 방향의 명령어의 수행을 무효화시키고 올바른 분기 방향의 명령어를 수행시키는 복구 과정에서 제거되는 명령어를 효과적으로 감소시켜 프로세서 성능을 향상시키는 새로운 분기 예상 실패 복구 메커니즘을 제안한다. 분기 명령어의 결과에 상관없이 잘못된 분기 방향과 올바른 분기 방향에서 모두 수행되는 명령어 즉, 분기가 수렴한 이후의 명령어는 분기 예상이 틀렸을 경우에 명령어들이 무효화된 후 올바른 분기 방향에서 재 반입 및 재 수행하게 된다. 이와 같이 재 사용될 수 있는 명령어는 분기 예상 실패 시 무효화하지 않고 이후에 재 사용함으로써 하드웨어 자원의 낭비를 줄이고 재 반입 및 재 수행에 필요한 사이클을 감소시켜 프로세서의 성능을 향상시킬 수 있다.

본 논문의 구성은 다음과 같다. 제 2 장에서는 본 논문의 연구를 위한 연구 배경과 제어 종속성에 의한 성능 감소를 제거하기 위해서 연구되었던 관련 내용을 살펴본다. 제 3 장에서는 분기 예상이 실패했을 경우 본 논문에서 제안한 제어 종속성을 이용한 복구 메커니즘에 대해 소개한다. 제 4 장에서는 본 논문에서 사용하는 실험의 비교 대상을 나열하고 실험에서 사용한 벤치마크 프로그램과 입력 파일을 설명한다. 제 5 장에서는 제어 종속성을 사용하지 않은 기존의 복구 메커니즘과 본 논문에서 제안한 복구 메커니즘을 시뮬레이션을 통해 비교 분석한다. 마지막으로, 제 6 장에서 결론을 맺는다.

2. 관련 연구

2.1 분기 예상 (Branch Prediction)

프로그램의 수행 중 분기 명령어가 반입되면 분기 명령어의 수행이 완료되어 결과가 구해질 때까지 어떤 분기 방향의 명령어를 반입할지의 결정이 지연되어야 한다. 일반적으로 전체 명령어 중 분기 명령어의 비율이 약 20~30% 정도를 차지한다. 현대의 프로세서들은 이슈 폭이 증가함에 따라 매 사이클마다 하나 이상의 분기 명령어가 나타나게 되며, 매 사이클마다 분기 명령어의 결과를 기다리기 위한 사이클 지연이 발생하게 된다. 이런 제어 종속성 문제는 프로세서의 성능에 심각한 영향을 미치게 된다. 이러한 프로세서의 성능 감소를 줄이기 위해 분기 명령어가 반입될 때, 분기 명령어의 결과를 예상함으로써 바로 다음 사이클에 분기 명령어 이후에 수행될 명령어를 반입해서 수행하게 된다. 이를 분기 예상기법이라 한다.

분기 예상기법을 수행하기 위해서는 다음과 같은 작업이 필요하다. 첫째로 분기 명령어의 분기 방향을 예상해서 taken 분기 방향의 명령어를 반입할지 not taken 분기 방향의 명령어를 반입할지를 결정해야 한다. 두 번째로는 분기 방향에 따라서 반입해야 할 명령어의 목적 주소를 예상해야 한다.

분기 방향을 예상하기 위해서는 분기 예상기법을 사용한다[5][6]. 분기 예상기법에 의해 분기 방향이 결정되면 분기 방향에 따른 다음 명령어의 목적 주소를 예상하기 위해서 BTB(Branch Target Buffer)를 사용한다[7][8]. 대부분의 분기 예상 메커니즘은 분기 명령어 예상 시 분기 방향을 예상하기 위한 예상기법과 목적 주소를 예상하는 기법을 함께 사용한다.

2.2 제어 독립성 (Control Independence)

분기 명령어 이후에 나타나는 명령어는 분기 명령어의 결과에 따라 수행할 명령어들이 결정된다. [그림 1]의 블록 B와 C의 명령어처럼 분기 명령어의 결과에 따

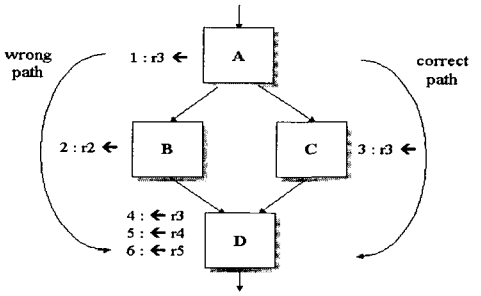

그림 1 제어 독립적인 명령어의 흐름도

라 수행 여부가 결정되는 명령어를 제어 종속적인 명령어라고 한다. 반대로 [그림 1]의 블록 D와 같이 분기 결과에 상관없이 항상 수행되는 명령어를 제어 독립적인 명령어라 한다[9].

기존의 분기 예상 복구 메커니즘에서는 분기 명령어의 예상이 틀렸을 경우 분기 명령어 이후의 반입, 수행되고 있는 모든 명령어를 무효화시키고 올바른 방향의 명령어를 반입, 수행시킨다[10][11]. 그러나, 제어 독립성을 이용하면 무효화되는 명령어를 최소화시켜 프로세서의 성능을 향상시킬 수 있다. [그림 1]의 블록 A에 있는 분기 명령어를 예상하여 블록 B와 D가 수행되었다고 가정하자. 분기 명령어의 실행이 완료되어 예상이 실패한 것으로 판명되면, 블록 B와 D의 명령어를 무효화시킨 다음, 올바른 분기 방향의 블록 C와 D를 반입하여 수행시켜야 한다. 이 때, 블록 D는 잘못 예상된 경로에서 이미 반입되어 수행되었다. 따라서, 블록 B만이 분기 명령어에 제어 종속적인 명령어이므로 블록 B의 명령어들을 무효화시키고 블록 C의 명령어들만 반입시키면 된다. 즉, 분기 명령어에 대해 제어 독립적인 명령어인 블록 D의 명령어들은 다시 반입될 필요가 없다. 이와 같은 블록 D의 명령어들을 재 반입 회피 명령어라 한다. 그러나, 블록 D의 제어 독립적인 명령어를 그대로 사용할 수 있는 것은 아니다. 블록 D의 명령어 중 4번 명령어는 잘못된 분기 방향을 통해서 수행되었을 때에는 1번 명령어의 결과를 사용해 수행하게 된다.

분기 명령어의 예상이 틀린 경우, 올바른 분기 방향의 명령어를 수행해야 하며 이런 경우 4번 명령어는 3번 명령어의 결과를 가지고 재 수행이 이루어져야 한다. 이후의 명령어 중 4번 명령어가 생성한 결과 값을 사용하는 자료 종속(Data Dependence) 관계를 가지고 있는 명령어들도 재 수행이 되어야 한다.

그러나, 5번과 6번 명령어들은 블록 B,C의 제어 종속적인 명령어들과 자료 종속 관계가 없기 때문에 재 수행될 필요가 없는 명령어이다. 이러한 명령어들을 재 수행 회피 명령어라 한다

재 반입 회피와 재 수행 회피의 비교를 위해 4번과 5번 명령어를 비교하면 다음과 같다. 4번 명령어(재 반입 회피)는 제어 독립적인 명령어이기 때문에 재 반입은 회피할 수 있지만 자료 종속 관계로 인해서 재 수행되어야 한다. 그러나, 5번 명령어(재 수행 회피)는 재 반입을 회피할 수 있으면서 동시에 재 수행도 회피할 수 있는 명령어이다.

이와 같이 모든 제어 독립적인 명령어에 대해서는 재 반입을 위한 페널티를 감소시킬 수 있고, 제어 독립적인 명령어들 중에서 제어 종속적인 명령어와 자료 종속 관계가 없는 명령어들은 재 수행에 드는 페널티도 감소시킬 수 있다.

2.3 동적 제어 독립성(Dynamic Control Independence)

분기 예상기법에 의해 예상이 이루어진 분기 명령어의 수행이 완료되었을 때 해당 분기 명령어의 예상 정확성 여부를 확인할 수 있다. 분기 명령어의 예상이 틀렸을 경우에는 분기 명령어 이후의 이미 수행되었거나 수행중인 모든 명령어를 제거함으로써 무효화시킨다[12]. Smith등은 슈퍼스칼라 프로세서에서 제어 독립성에 대한 기본적인 연구를 통해 잠재적인 성능 향상을 제시하였으며, trace 프로세서에서의 제어 독립성을 이용한 기법도 제안하였다[13].

Shen등은 분기 예상 실패 시 복구하기 위한 메커니즘으로 동적 제어 독립성을 이용하였다[14]. 제어 독립적인 명령어의 위치를 찾기 위해 DCI 버퍼(Dynamic Control Independence Buffer)를 사용한다. DCI 버퍼는 기존의 ROB(Reorder Buffer)와 동일한 구조와 동일한 크기의 하드웨어를 갖는다. DCI 버퍼는 순환 큐(Circular Queue)의 구조로 구성되고 세 개의 포인터 즉, Head 포인터, Mispred Branch 포인터 그리고 Next CI 포인터를 가진다. Head 포인터는 다음 명령어가 삽입될 포인터를 나타낸다. 명령어가 ROB에 할당될 때 동시에 DCI 버퍼에도 동일하게 할당되어 PC(Program Counter)와 태그 및 목적 레지스터의 지시자 등을 입력한다. 명령어의 수행이 완료되어 결과가 생성되었을 때 ROB 엔트리에 결과 값을 입력하는 동시에 DCI 버퍼에도 동일하게 입력된다.

분기 명령어가 정확하게 예상이 이루어졌을 경우에는 ROB와 DCI 버퍼에는 동일한 명령어가 기록된다. 그러나, 분기 예상이 틀렸을 경우에는 분기 명령어 이후의 모든 명령어를 ROB로부터 제거하지만, DCI 버퍼에는 입력된 상태를 그대로 유지시킨다. DCI 버퍼에서 예상이 틀린 분기 명령어를 Mispred Branch 포인터가 지시하게 한다. 올바른 분기로부터 명령어가 반입될 때마다 DCI 버퍼의 Mispred Branch 포인터와 Head 포인터 사이의 모든 명령어와 비교하여 이전의 잘못된 분기 방향에서 나타난 명령어가 올바른 분기 방향에서도 나타나는지 비교한다. 이렇게 이전의 잘못된 분기 방향의 명령어와 올바른 분기 방향에서 사용된 명령어가 동일하다면 해당 명령어가 제어 독립적인 명령어의 첫 번째 명령어가 된다. 이 명령어를 Next CI 포인터가 가리키게 된다. 이와 같은 방법으로 제어 독립적인 명령어를

탐지하여 사용하게 된다.

DCI 버퍼는 동적으로 제어 독립성을 검출함으로써 컴파일러의 도움 없이 하드웨어 자체만으로 구현이 가능하다는 이점이 있다. 그러나, 각 엔트리에 적지 않은 필드를 가지고 있는 ROB와 동일한 하드웨어를 필요로 하는 DCI 버퍼의 사용은 하드웨어 비용 면에서 비효율적이다. 또한, DCI 버퍼의 사용은 매 사이클마다 최대 하나의 분기 명령어만 반입 및 수행되는 슈퍼스칼라 프로세서에서만 가능하다. 왜냐하면 여러 개의 분기 명령어가 동시에 반입되고 수행이 완료되는 경우에는 예상이 틀린 분기 명령어마다 DCI 버퍼를 구성하여야 한다. 이는 파이프라인 이슈 폭이 커지고 있는 현대의 슈퍼스칼라에서는 기하급수적인 하드웨어 자원의 증가를 요구하기 때문에 실제적인 구현이 불가능하다.

3. 분기 예상 실패 복구 메커니즘

프로세서가 잘못 예상된 분기 명령어에 의해 변경된 프로세서의 상태를 복구할 때 제어 독립성을 이용하기 위해서 처리해야 할 두 가지의 작업이 있다. 첫 번째로 잘못 예상된 분기 명령어 이후의 명령어 중 무효화될 필요가 없이 다시 사용되는 제어 독립적인 명령어를 탐지해야 한다. 두 번째로는 탐지된 제어 독립적인 명령어들 중에서 분기 명령어의 결과에 의존하는 제어 종속적인 명령어들의 결과를 사용하는 제어 독립적인 명령어를 찾아내야 한다. 이렇게 분기 명령어에 대해 제어 종속적인 명령어와 자료 종속 관계를 가지고 있는 제어 독립적인 명령어만 올바른 결과를 사용해서 재 수행시키면 되므로, 동일한 명령어의 재 수행에 사용되는 하드웨어 자원의 낭비를 줄여 프로세서의 성능을 향상시킬 수 있다.

3.1 제어 독립적인 명령어의 탐지

본 논문에서 제안한 기법은 프로그램의 구조에서 나타나는 패턴을 이용해 제어 독립적인 명령어를 탐지해 낸다. 컴파일 시 각각의 분기 명령어에 대응하는 제어 독립적인 명령어의 주소 정보를 첨가시킨다. 이렇게 정적으로 구해진 정보를 이용해 프로그램의 수행 시 동적으로 분기 명령어에 대응하는 제어 독립적인 명령어를 탐지한다.

[그림 2]는 반복문과 같이 프로그램 순서 상에서 앞쪽으로 분기하는 후향 분기 명령어(Backward Branch)의 패턴을 보여주고 있다. 이 경우 분기 명령어의 결과에 상관없이 항상 수행되는 명령어, 즉 제어 독립적인 명령어는 후향 분기 명령어의 다음 명령어가 된다.

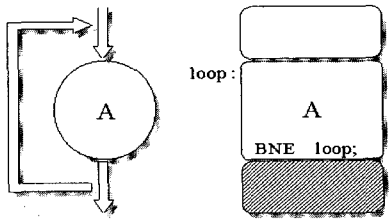

그림 2 후향 분기 명령어의 프로그램 패턴

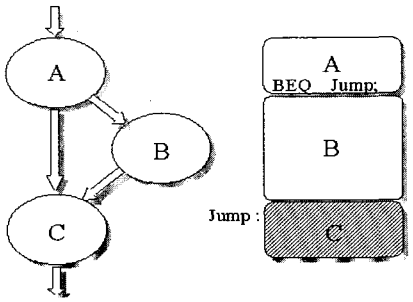

그림 3 단방향 전향 분기 명령어의 패턴

[그림 3]은 프로그램의 뒤쪽으로 분기를 하는 단방향 전향 분기 명령어(Forward Branch)의 기본 블록과 하위 레벨의 형태로 변환된 프로그램의 간략한 도식을 보여주고 있다. 그림에서 보는 바와 같이 기본 블록 A의 마지막에는 조건 분기 명령어가 위치하여 분기 명령어의 결과에 따라서 기본 블록 B와 C의 명령어를 수행할지 아니면 블록 C의 명령어만 실행할지가 결정된다. 따라서, 분기 명령어의 결과에 상관없이 항상 수행되는 제어 독립적인 명령어는 기본 블록 C가 된다. 이 때 분기 명령어에 대한 첫 번째 제어 독립적인 명령어는 조건 분기 명령어의 목적 명령어의 주소가 된다.

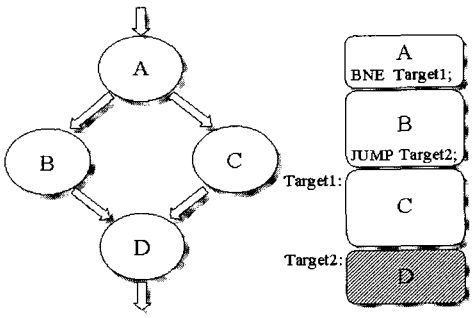

그림 4 양방향 전향 분기 명령어의 패턴

[그림 4]는 분기 명령어의 목적 명령어들이 뒤쪽에 존재하면서 분기 명령어의 결과에 따라서 양방향으로 분기하는 전향 분기 명령어 형태를 보여주고 있다. 이 유형의 하위 레벨의 프로그램을 보면 기본 블록 A의 마지막에 조건 분기 명령어가 존재하며, 조건 분기 명령어의 목적 명령어(Target1) 바로 앞의 명령어가 제어 독립적인 명령어로 분기하는 무조건 분기 명령어가 존재한다. 따라서, 양방향으로 분기하는 전향 조건 분기 명령어의 제어 독립적인 명령어는 목적 명령어 바로 이전에 존재하는 무조건 분기 명령어의 목적 명령어(Target2)가 된다.

위와 같은 방법을 통해 각 분기 명령어에 대한 제어 독립적인 명령어의 정보를 구해서 프로그램이 실제로 수행하는 동안에 동적으로 제어 독립적인 명령어를 탐지할 수 있다.

3.2 제어 독립적인 명령어의 주소 정보

각 분기 명령어에 제어 독립적인 명령어의 주소 정보를 첨가하는데 있어서 다양한 변화를 고찰할 수 있다. 첨가되는 제어 독립적인 명령어의 주소 정보를 전체 주소로 제공하는 방법과 해당 분기 명령어로부터 제어 독립적인 명령어로의 오프셋을 제공하는 방법이 있다.

전체 주소로 제어 독립적인 명령어의 주소 정보를 제공하는 방법은 분기 명령어에 대한 제어 독립적인 명령어의 모든 주소 정보를 제공할 수 있지만, 주소 정보를 위해 명령어 집합 구조를 변경해야 하며 많은 비트 수를 요구하는 단점이 있다. 전체 주소의 단점을 보완하기 위한 방법은 분기 명령어에 첨가되는 제어 독립적인 주소 정보를 n비트의 오프셋으로 제공하는 방법이 있다. 이 방법은 적은 비트 수로 제어 독립적인 명령어의 주소 정보를 첨가할 수 있고 명령어 집합 구조를 변경하지 않아도 된다. 하지만, 전체 주소를 첨가하는 방법에 비해 처리될 수 있는 제어 독립적인 명령어의 점유율이 감소할 수 있다는 단점이 있다.

[그림 5]는 본 논문의 실험을 위해 사용한 명령어 형식을 보여주고 있다[15]. [그림 5]와 같이 분기 명령어를 위해 사용되는 명령어 형식에는 6-비트의 사용되지 않는

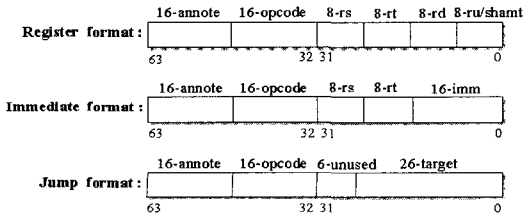

그림 5 SimpleScalar 구조의 명령어 형식

필드가 존재한다. 이 6-비트의 오프셋을 통해 제어 독립적인 명령어를 명시할 수 있는 비율이 많다면 명령어 집합 구조를 수정하지 않고도 사용되지 않는 6-비트 필드를 사용할 수 있다. 본 논문에서 사용한 명령어 집합 구조 이외에도 다른 구조에서도 역시 분기 명령어에 해당하는 명령어 형식에는 사용하지 않는 비트들을 제공하고 있다. 따라서, 본 논문에서는 분기 명령어에 대한 제어 독립적인 명령어를 명시하기 위해 6-비트의 오프셋을 사용한 방법과 제어 독립적인 명령어의 전체 주소를 사용하는 방법을 모두 사용하여 실험을 수행하였다.

3.3 제어 독립적인 명령어 중 재 수행할 명령어의 탐지

제어 독립적인 명령어라고 하더라도 잘못된 분기 예상이 복구된 후에 모든 제어 독립적인 명령어가 이전의 결과 값을 가지고 그대로 재 사용될 수 있는 것은 아니다. 물론 제어 독립적인 명령어는 ROB로부터 제거되어 다시 명령어 캐쉬로부터 반입되는 단계를 제거시켜 소비되는 사이클을 감소시킬 수 있다. 그러나, 본 논문의 궁극적인 목적은 잘못 예상된 분기 방향을 통해서 수행한 제어 독립적인 명령어가 올바른 분기 방향을 통해서도 같은 상태를 가지고 수행된다면 이러한 제어 독립적인 명령어는 재 수행하지 않고 이미 잘못된 분기 방향에서 수행된 상태를 그대로 재 사용할 수 있다는 점이다[16].

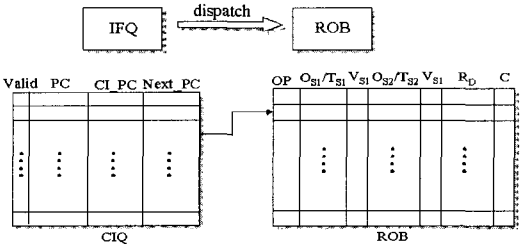

그림 6 제안한 복구 메커니즘의 하드웨어 구조

[그림 6]에서와 같이 분기 명령어의 반입 단계에서 분기 예상을 위해 분기 예측기를 접근함과 동시에 컴파일 과정에서 구해진 분기 명령어와 제어 독립적인 명령어까지의 오프셋을 계산한다. 계산되어진 제어 독립적인 명령어의 전체 주소는 CIQ(Control Independent Queue)에 입력된다. CIQ의 엔트리에서 PC는 분기 명령어의 주소를 저장하며 CI_PC 필드에는 해당 분기 명령어의 제어 독립적인 명령어의 주소를 입력한다. 분기 명령어의 수행이 완료되어 분기 명령어의 예상이 틀렸다고 판단되었을 경우 Valid 비트를 세트시켜 예상이 틀린 분기 명령어임을 명시하고, 이후 예상이 틀린 분기

명령어의 복구 시 제어 종속적인 명령어만을 무효화시킨다. 즉, 분기 명령어 이후의 모든 명령어를 무효화시키지 않고 CIQ에 저장되어 있는 정보를 이용해 해당 분기 명령어에서부터 제어 독립적인 명령어 바로 이전의 제어 종속적인 명령어만을 무효화시킨다.

[그림 6]의 ROB는 기존의 ROB와 동일한 구조에 C(Completion) 비트만을 추가한 형태이다. C 비트는 명령어의 수행이 완료될 때 세트시켜 해당 명령어의 수행이 완료되었는지의 상태를 나타낸다. 제거되는 제어 종속적인 명령어가 생성한 결과 값을 이후의 제어 독립적인 명령어가 입력 값으로 사용하고 있다면, 즉 자료 종속 관계가 존재한다면 해당 제어 독립적인 명령어는 잘못된 입력 값을 가지고 수행된 명령어가 된다. 이와 같은 제어 독립적인 명령어의 C 비트는 리셋시킨다. 해당 제어 독립적인 명령어가 이미 수행이 완료되었다면 세트되어 있는 C비트를 리셋시킴으로써 이후에 제어 독립적인 명령어의 수행이 시작할 때 최근의 올바른 입력 값을 가지고 다시 수행하게 하는 역할을 한다.

이와 같이 분기 명령어의 예상이 실패했을 경우, 이후에 다시 사용될 수 있는 명령어를 ROB에서 제거하지 않고 유지함으로써 재거되는 명령어를 감소시켜 제어 독립적인 명령어의 재 수행 시간을 줄일 수 있고 하드웨어 자원의 낭비를 줄임으로써 프로세서의 성능을 향상시킬 수 있다.

4. 실험 환경

본 절에서는 분기 명령어의 예상이 틀렸을 경우에 이후의 모든 명령어를 무효화시키는 기존의 기법과 본 논문에서 제안한 제어 독립적인 명령어를 이용한 복구 메커니즘을 비교 분석하였다. 기존에 제안된 DCI 버퍼를 사용한 기법은 여러 개의 분기 명령어를 반입 및 수행하는 슈퍼스칼라 프로세서에서는 사용이 부적합한 기법이기 때문에 본 논문에서의 비교 대상에서 제외시켰으며, 본 논문에서 제안한 기법 중 6-비트 오프셋을 사용한 방법과 전체 주소를 사용한 방법을 구분하여 실험하였다.

본 논문에서 사용한 시뮬레이터는 MIPS-4 명령어 집합 구조를 기반으로 한 SimpleScalar/ PISA 3.0 툴 셋을 사용하였으며, 분기 예상 실패의 페널티를 모델링한 실행 구동 시뮬레이터이다[17]. 실험은 4, 8, 16 개의 이슈 폭을 갖는 비 순서적(out-of-order) 이슈 및 실행 프로세서를 사용하였으며 ROB 대신 레지스터 재명명(Register Rename)과 명령어 재순서화(Instruction Reordering)를 동시에 제공하는 RUU(Register Update Unit)를 사용하였다[18]. [표 1]은 본 논문에서 사용한 시뮬레이션의 환경을 보여주고 있다.

[표 2]는 실험을 위해서 6개의 SPEC95int 벤치마크 프로그램과 입력 파일에 대한 실험 정보를 보여주고 있다. 각 벤치마크 프로그램은 -O3의 최적화 옵션을 가진

표 1 시뮬레이션 환경

실험 변수	실험 데이타		
fetch/issue-width	4-issue	8-issue	16-issue
RUU/LSQ/CIQ size	64 / 32 / 16	128 / 64 / 32	256 / 128 / 64
Branch Predictor	8-bit BHR + 2-bit 4k-entry PHT를 가진 Two-Level Adaptive Training Branch Predictor 2K, 4-way set associative BTB		
Return Addr. Stack	32	64	128
Multiple Branch #	1	1	2
Functional Unit	8 int ALU 2 int mul/div 4 FP-adder 1 FP mul/div	16 int ALU 4 int mul/div 8 FP-adder 2 FP mul/div	32 int ALU 8 int mul/div 16 FP-adder 4 FP mul/div
I-cache	32KB	64KB	128KB
	2-way set associative, 32-byte cache block 1-cycle hit time, 6-cycle miss penalty		
D-cache L1	64KB	128KB	256KB
	4-way set associative, 32-byte cache block 1-cycle hit time, 6-cycle miss penalty		
D-cache L2	256KB	512KB	1MB
	4-way set associative, 64-byte cache block 6-cycle hit time, 18-cycle miss penalty		
Memory access latency	First chunk : 18-cycle Inter chunk : 2-cycle 16-bytes memory access bus width		

표 2 벤치마크 프로그램의 구성

벤치마크	입력 파일	수행된 전체 명령어 수	수행된 분기 명령어 수	분기 예상 정확도
compress	test.in	36M	6.2M	90.9%
gcc	jump.i	40M	8.3M	91.3%
go	2stone9.in	100M	14.8M	80.7%
li	queen6.lsp	42M	9.9M	96.5%
m88ksim	tiny.in	100M	22.8M	96.5%
vortex	vortex.tiny.in	65M	10.2M	95.7%

GNU GCC(ver 2.6.3)를 사용해 컴파일 했으며 실험은 각 벤치마크 프로그램의 수행된 명령어 수를 최대 100M까지 제한적으로 수행하였다.

'수행된 전체 명령어의 수'는 동적으로 수행이 완료된 명령어의 수를 나타내고 있으며, '수행된 분기 명령어 수'는 수행이 완료된 명령어 중 분기 명령어 수를 나타내며, 4~6 명령어 당 1개의 분기 명령어가 수행되고 있음을 알 수 있다. '분기 예상 정확도'는 8-이슈 폭을 가진 프로세서에 대해 분기 예상기법을 적용한 결과, 예상이 수행된 명령어 중에서 정확한 예상이 이루어진 명령어의 비율을 보여주고 있다.

5. 실험 결과

5.1 무효화되는 명령어 중 제어 독립적인 명령어의 비율

본 논문에서 제안한 기법에서는 예상이 실패했을 경우, 이후에 모험적으로 수행된 모든 명령어를 무효화시키지 않고 제어 종속적인 명령어만을 무효화시켜 제어 독립적인 명령어의 재 반입을 제거하였다.

[그림 7]의 그래프는 분기 예상 실패로 인해 무효화되는 명령어를 보여준다. 'CD'는 무효화되는 명령어 중에서 제어 종속적인 명령어의 비율을 나타내며, CI는 제어 독립적인 명령어의 비율을 나타낸다. CI에 해당하는 제어 독립적인 명령어 중에서 'CI with data'는 제어 종속적인 명령어와 자료 종속관계를 갖는 비율을 나타내고, 'CI without data'는 제어 종속적인 명령어와 자료 종속관계가 없는 명령어의 비율을 나타낸다. 즉, CI에 해당하는 비율만큼 명령어가 다시 반입되는 페널티를 줄일 수 있다. 추가로 전체 제어 독립적인 명령어 중에서도 제어 종속적인 명령어와 자료 종속관계가 없어서 이전의 수행 결과 값을 재 사용할 수 있는 비율이 'CI without data'에 나타낸 부분이 된다.

6개의 벤치마크 프로그램의 실험 결과, 8-이슈 폭을 갖는 프로세서에서 평균 53%의 명령어의 재 반입 사이클을 줄일 수 있고, 재 반입을 회피하는 명령어 중 평균 28%에 해당하는 명령어는 재 수행하지 않고 이전 결과 값을 그대로 사용함으로써 재 수행 사이클을 감소시킬 수 있다.

5.2 제어 독립 명령어 명시를 위한 변화

본 절에서는 각 분기 명령어에 제어 독립적인 명령어의 주소 정보를 첨가하는데 있어서 오프셋을 위해 사용되는 필드의 다양한 크기 변화를 고찰해 본다. 또한, 첨가되는 제어 독립적인 명령어의 주소 정보를 전체 주소로 제공하는 방법과 해당 분기 명령어로부터 제어 독립적인 명령어 사이의 오프셋으로 제공하는 방법을 구분하여 실험하기 위해 각각 어느 정도의 제어 독립적인 명령어를 처리할 수 있는지 알아보았다.

대부분의 프로그램 상에서 분기 명령어와 목적 명령어 사이의 거리는 적은 오프셋으로 표현이 가능하다. 따라서, 분기 명령어에 대한 제어 종속적인 명령어까지의 거리 또한 적은 오프셋으로 사용이 가능하다. [그림 8]은 각 오프셋의 비트를 변화시키면서 처리될 수 있는 제어 독립적인 명령어의 비율을 실험한 결과를 보여주고 있다.

실험 결과, 모든 벤치마크 프로그램에서 8-비트의 오

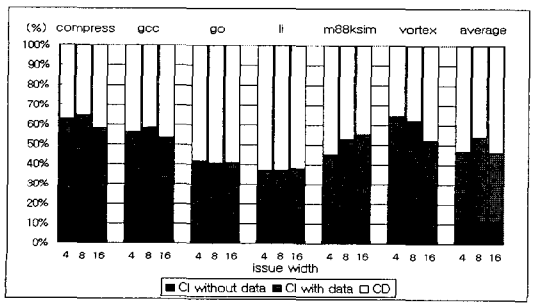

그림 7 무효화되는 명령어 중 제어 독립적인 명령어의 비율

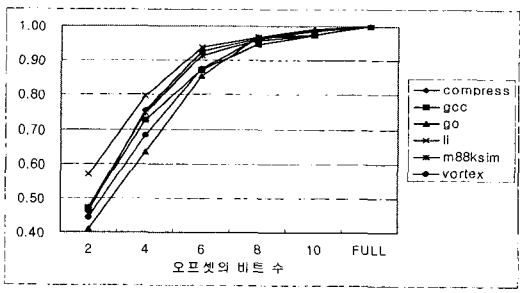

그림 8 제어 독립적인 명령어 명시를 위한 오프셋 변화

프셋을 사용한 실험 결과는 할 경우에 약 95%의 제어 독립적인 명령어를 처리할 수 있음을 보여주고 있다. 그러나, 8-비트의 오프셋과 전체 주소를 사용하기 위해서는 명령어 집합 구조를 변경해야 하는 문제가 발생하게 된다. 본 논문의 실험에 사용되고 있는 MIPS-4 명령어 형식은 분기 명령어의 경우, 6-비트의 사용되지 않는 비트를 가지고 있다. 따라서, 분기 예상이 틀렸을 경우 평균 90%의 제어 독립적인 명령어를 처리할 수 있는 6-비트의 오프셋을 사용한 구조와 전체 주소를 사용해 모든 독립적인 명령어를 처리하는 구조로 구분하여 실험하였다.

5.3 제어 독립성을 이용한 복구 메커니즘의 성능 향상

본 절에서는 분기 예상이 틀렸을 경우에 분기 예상 이전의 상태로 복구함에 있어서 잘못된 분기 방향에서 나타난 명령어들이 올바른 분기 방향에서 반복적으로 나타날 때, 제어 독립적인 명령어를 재 사용하기 위한 복구 메커니즘을 실험하였다.

실험 대상은 분기 예상이 실패했을 경우에 예상된 분기 명령어 이후의 모든 명령어를 무효화시키고 다시 올바른 분기 방향으로부터 명령어를 반입해 수행하는 기본 구조와 본 논문에서 제안한 복구 메커니즘 중 제어 독립적인 명령어를 명시하기 위해 6-비트의 오프셋을 사용하는 구조, 그리고 전체 주소를 사용하여 제어 독립적인 명령어를 명시하는 구조를 비교 실험하였다.

[그림 9]는 6개의 SPECint95 벤치마크 프로그램을 사용해 4, 8, 16 개의 이슈 폭을 갖는 프로세서에서 사이클 당 수행된 명령어 수를 측정함으로써 제어 독립을 이용한 복구 메커니즘의 성능 향상을 보여주고 있다. 모든 벤치마크 프로그램에서 기존의 방법에 비해 6-비트 오프셋을 사용한 경우와 전체 주소를 사용한 경우에서 성능이 향상되고 있다. 이는 분기 명령어의 예상이 틀려서 분기 명령어 이후의 명령어들이 무효화될 때 제어 독립적인 명령어가 항상 존재하기 때문이다.

분기 예상 정확도는 제어 독립성을 이용한 복구 메커니즘의 성능에 많은 영향을 미치고 있다. 가장 분기 예상 정확도가 낮은 go의 경우, 다른 벤치마크 프로그램보다 월등한 성능 향상을 보이고 있다. 이는 분기 예상이 자주 틀리는 경우, 복구하기 위해 무효화되는 명령어가 증가하게 되고, 무효화되는 명령어 중에서 성능 향상에 기여할 수 있는 제어 독립적인 명령어가 많이 발생하기 때문이다.

16-이슈 폭을 갖는 프로세서는 4 또는 8-이슈 폭을 갖는 프로세서보다 많은 명령어가 반입, 수행된다. 분기 예상이 정확하다면 동시에 많은 명령어를 수행시킬 수 있으므로 효과적인 성능을 발휘할 수 있지만, 분기 예상이 틀렸을 경우에는 많은 명령어가 무효화돼야 하므로 이런 무효화되는 명령어 중 제어 독립적인 명령어가 차지하는 비율이 증가한다.

제어 독립적인 명령어를 나타내기 위해 6-비트의 오프셋 정보를 사용한 경우, 전체 주소를 사용한 경우에 근접한 성능 향상을 보이고 있다. 5.2 절에서의 실험에서 6-비트의 오프셋 정보만으로 약 90%의 제어 독립적인 명령어의 정보를 처리할 수 있기 때문에 본 논문에서는 명령어 집합 구조를 수정하지 않고 오프셋 정보를 사용한 복구 메커니즘의 구현을 제안한다.

[그림 9]에서 보는 바와 같이 4-이슈 폭의 프로세서는 2%~7%, 8-이슈 폭의 프로세서는 4%~15%, 그리고 16-이슈 폭의 프로세서에는 8%~28%의 상대적인 성능 향상을 보이고 있다.

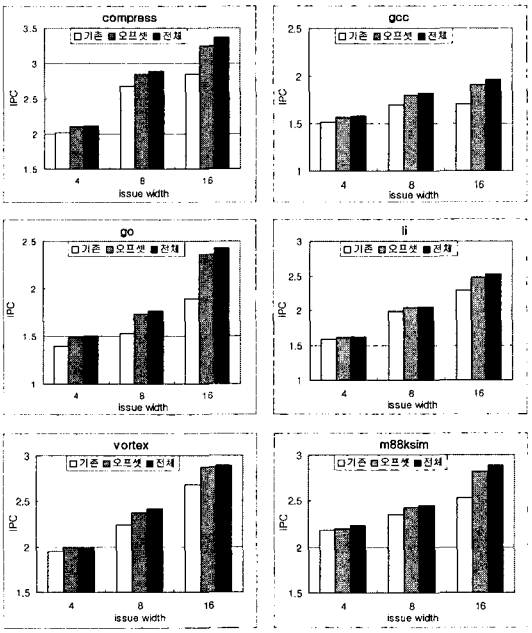

그림 9 제어 독립성을 이용한 복구 메커니즘의 성능 향상

6. 결 론

슈퍼스칼라 프로세서에서 동시에 다수의 명령어를 수행시키는 명령어 수준 병렬성을 향상시키기 위해서는 분기 예상 기법이 필수적이다. 그러나, 모든 분기 명령어를 정확하게 예상하기란 불가능하며, 분기 예상 실패시 발생하는 페널티는 프로세서의 성능에 심각한 영향

을 준다.

본 논문에서는 제어 독립성의 중요성을 언급함과 동시에 분기 예상이 실패한 경우, 분기 명령어 이후의 명령어들을 복구하는데 필요한 페널티를 감소시키기 위해 제어 독립적인 명령어를 재 사용하는 새로운 복구 메커니즘을 제안하였다. 제어 독립성을 사용하기 위해서 컴파일 시 프로파일링을 통한 정적인 방법과 프로그램상의 제어 흐름을 통해 동적으로 제어 독립적인 명령어를 탐지하는 방법을 이용하여 분기 명령어의 잘못된 예상으로 인해 무효화되는 명령어를 효과적으로 감소시켜 프로세서의 성능을 향상시켰다.

잘못된 분기 예상의 결과에 의해 수행된 명령어 중에서 분기 명령어 이전의 상태로 복구하기 위해 무효화되는 명령어 중 제어 독립적인 명령어의 비율을 측정함으로써 제어 독립성으로 인한 잠재적인 성능 향상의 가능성을 제시하였으며, 분기 예상 정확도 및 동시에 다수의 명령어를 예상하는 큰 이슈 폭을 갖는 프로세서 구조가 제어 독립성에 미치는 영향을 실험하였다. 또한, 기존의 명령어 집합 구조를 변경하지 않고 적은 하드웨어의 추가로 기존의 분기 예상 실패 복구 메커니즘을 향상시키는 기법을 제안하였다.

비 순서적 슈퍼스칼라 프로세서에서 실험하였으며, 6개의 SPECint95 벤치마크 프로그램을 통해 성능 향상을 평가한 결과, 4-이슈 폭의 프로세서에서 평균 3.8%와 8-이슈 폭의 프로세서에서 평균 7.8%의 성능 향상을 보였다. 동시에 2개의 명령어를 예상하는 다중 분기 예상기법을 사용한 16-이슈 폭의 프로세서는 평균 15.6%의 성능 향상을 보였다.

향후 연구과제로는 재 반입 회피를 통한 성능향상 정도와 재 수행 회피를 통한 성능향상 정도의 비율 분석을 통해 동적으로 각각에 대해 더 효과적인 기법을 사용하도록 하는 연구가 이루어져야 할 것이다.

참 고 문 헌

[1] S. McFarling, "Combinig branch predictors," Technical Report TN-36, Digital Western Research Lab., June 1993.

[2] C. C. Lee, I. C. Chen and T. Mudge, "The Bi-Mode Branch Predictor," in Proc. of 30th MICRO, Dec. 1997.

[3] S. T. Pan, K. So and J. T. Rahmeh, "Improving the accuracy of dynamic branch prediction using branch correlation," In 5th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 76-84, October 1992.

[4] E. Sprangle, R. Chappell, M. Alsup and Y. Patt, "The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference," in Proc. of 24th ISCA, May 1997.

[5] E. Jacobsen, E. Rotenberg and J. E. Smith, "Assigning Confidence to Conditional Branch Predictors," in Proc. of 29th MICRO, pp 142-131, 1998.

[6] D. I. August, W. W. Hwu and S. A. Mahlke, "A Framework for Balancing Control Flow and Prediction," in Proc. of 30th MICRO, pp 92-103, 1997.

[7] J. F. Lee and A. J. Smith, "Branch prediction strategies and branch target buffer design," IEEE Computer, pp. 6-22, January 1984.

[8] C. Perleberg and A. J. Smith, "Branch target buffer design and optimization," IEEE Transactions on Computers, pp. 396-412, April 1993.

[9] E. Rotenberg, Q. Jacobson, and J. Smith, "A Study of Control Independence in Superscalar Processors," in Proc. of 5th HPCA, 1999.

[10] T. Y. Yeh and Y. N. Patt, "A comparison of dynamic branch predictors that use two levels of branch history," In 20th ISCA, May 1993.

[11] K. Lick and G. Tyson, "Hybrid Branch Prediction Using Limited Dual Path Execution," Technical Report UCR-CS-96-7, Univ. of California, July 1996.

[12] A. Sodani and G. S. Sohi, "Understanding the Differences Between Value Prediction and Instruction Reuse," in Proc. of 31st Ann. Int. Symp. on Microarchitecture, 1998.

[13] E. Rotenberg and J. E. Smith, "Control Independence in Trace Processors," Journal of Instruction-Level Parallelism, May 2000.

[14] Y. Chou, J. Fung and J. Shen, "Reducing branch misprediction penalties via dynamic control independence detection," Intl. Conf. on Supercomputing, June 1999.

[15] Charles Price, "MIPS IV Instruction Set, revision 3.1," MIPS Technologies, Inc., Mountain View, CA, January 1995.

[16] A. Sodani and G. S. Sohi, "Dynamic Instruction Reuse," in Proc. of 24th ISCA, June 1997.

[17] T. M. Austin and D. Burger, "The SimpleScalar Tool Set, Version 3.0," University of Wisconsin-Madison, 1998.

[18] G. S. Sohi, "Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers," IEEE Trans. Comput., 1990.

윤 성 룡

1999년 수원대학교 전자계산학과 이학사, 2001년 수원대학교 전자계산학과 이학석사, 2001년 1월 ~ 현재 SK텔레텍 S/W 개발 근무, 관심분야는 ILP 프로세서의 분기예상 메커니즘 및 결과값 예상 메커니즘, CDMA2000 1xEV-DO, IMT-2000

이 원 모

1993년 수원대학교 전자계산학과 이학사, 1996년 수원대학교 전자계산학과 이학석사, 2001년 8월 수원대학교 전자계산학과 박사과정 수료, 2001년 8월 ~ 현재 경인여자대학 컴퓨터정보학부 겸임교수, 관심분야는 최적화 컴파일러 설계, ILP 프로세서의 분기예상 메커니즘 및 결과값 예상 메커니즘

조 영 일

1980년 한양대학교 전자공학과 공학사. 1982년 한양대학교 전자공학과 공학석사. 1985년 한양대학교 전자공학과 공학박사. 1986년 3월 ~ 현재 수원대학교 정보공학대학 컴퓨터학과 교수. 관심분야는 최적화 컴파일러 설계, ILP 프로세서의 분기예상 메커니즘 및 결과값 예상 메커니즘, ILP프로세서의 정적, 동적 명령어 스케줄링, VoIP