레지스터 리네이밍 방법을 사용하는 조건부 실행 비순차적 명령어 이슈 마이크로프로세서에 관한 연구

준회원 최 규 백*, 김 문 경*, 홍 인 표*, 이 용 석**

Research on Conditional Execution Out-of-order Instruction Issue Microprocessor Using Register Renaming Method

Kyu-Baik Choi*, Moon-Gyung Kim*, In-Pyo Hong*, Yong-Surk Lee** *Regular Members*

요 약

본 논문에서는 조건부 실행 비순차적 명령어 이슈 컴퓨터 시스템에서의 레지스터 리네이밍 방법을 제안한다. 레지스터 리네이밍은 읽기 후 쓰기 그리고 쓰기 후 쓰기 의존성을 제거하는 기술이다. 레지스터 리네이밍 방법을 사용하는 조건부 실행 비순차적 명령어 이슈 컴퓨터 시스템을 구현하기 위해서, 우리는 순차적 상태 물리적 레지스터와 미리보기 상태 물리적 레지스터들 양자를 모든 논리적 레지스터들이 공유할 수 있도록 포함하고 있는 레지스터 파일을 사용한다. 또한 본 논문에서 제안된 구조를 구현하기 위해서 순차적 상태 지시기, 리네이밍 상태 지시기, 물리적 레지스터 할당 지시기, 조건 예측 버퍼, 리오더 버퍼들을 구현한다. 이러한 모든 하드웨어를 이용해서, 레지스터 리네이밍 방법을 사용하는 조건부 실행 비순차적 명령어 이슈 컴퓨팅 시스템의 레지스터 리네이밍 및 순차적 상태의 추적을 가능하게 한다. 본 논문에서는 위의 하드웨어를 사용하여 기존 레지스터 리네이밍 방법에 비해서 적은 하드웨어 비용으로 내용 검색(associative lookup)을 제거하고 짧은 복구 시간을 제공하는 개량된 레지스터 리네이밍 방법을 제안한다.

ABSTRACT

In this paper, we present a register renaming method for conditional execution out-of-order instruction issue microprocessors. Register renaming method reduces false data dependencies (write after read(WAR) and write after write(WAW)). To implement a conditional execution out-of-order instruction issue microprocessor using register renaming, we use a register file which includes both in-order state physical registers and look-ahead state physical registers to share all logical registers. And we design an in-order state indicator, a renaming state indicator, a physical register assigning indicator, a condition prediction buffer and a reorder buffer. As we utilize the above hardwares, we can do register renaming and trace the in-order state. In this paper, we present an improved register renaming method using smaller hardware resources than conventional register renaming method. And this method eliminates an associative lookup and provides a short recovery time.

Key Words : register renaming, conditional execution, out-of-order

Ⅰ. 서 론

현재 대부분의 마이크로 프로세서는 성능의 향상을 꾀하기 위해서 비순차 수행을 하고 있으며, 비순차 수행 컴퓨터 시스템을 가지는 프로세서는 병렬

*연세대학교 전기전자공학과 프로세서연구실(wayfarer@dubiki.yonsei.ac.kr)
논문번호 : 030297-0715 접수일자 : 2003년 7월 15일
※ 본 연구는한국과학재단 목적기초연구(R01-2002-000-00310-0)지원으로 수행되었음

적으로 여러 명령어를 이슈함으로서 명령어 단계 병렬성을 이용하여 성능을 향상시킨다. 이것은 한 단일 스레드 안에 어느 정도의 명령어를 병렬적으로 이슈할 수 있는지의 여부에 의해서 제한을 받게 된다. 프로세서가 병렬적으로 이슈할 수 있는 명령어의 수를 증가시킴으로 인한 성능의 향상을 위해서 여러 방법들이 개발되어졌다. 동시에 명령어를 이슈할 수 있도록 하는 기존의 연구로는 리오더 버퍼를 이용하는 방법과 미래 파일이나 명령어 창을 이용하는 방법이 있고[1] 논리 레지스터 세트를 하나의 뱅크 형태로 정의하여 여러 개의 뱅크를 두고 현 상태의 구조적 상태와 순차적 상태에 대한 뱅크 번호를 저장하는 부분을 두어 해결하는 방식[2]이 있으며 마지막으로 프리 리스트와 함께 우선권 인코더와 rename상태 지시기를 이용한 방법[3]이 있다. 위에서 말한 방법들은 다음에서 설명하게 될 단점들을 가지고 있다. 따라서 본 논문에서는 병렬적으로 이슈할 수 있는 명령어의 수를 증가시키기 위한 한 방법인 비순차적 이슈 시스템에서의 리네이밍을 지원하기 위해서 새로운 구조인 프리 리스트와 함께 우선권 인코더와 rename상태 지시기를 이용하는 방법을 개선한 구조를 제안한다.

II. 기존의 관련 연구

1. Reorder Buffer

명령어를 병렬적으로 이슈해서 동시에 수행가능하게 하는 것은 비순차 수행을 하는 슈퍼스칼라 프로세서에 있어서는 중요한 것이다. 여러 명령어의 동시 수행을 제공하기 위해서 레지스터 리네이밍 방법을 사용해서 false data dependency를 제거하는 것이고 레지스터 리네이밍 방법 중에서 가장 많이 사용되는 것이 리오더 버퍼를 이용하는 것이다.

그림 1.은 리오더 버퍼의 블록 다이어그램을 나타낸다. 명령어가 명령어 디코더에서 디코딩이 진행되면 비순차 수행을 가능하게 하기 위해서 의존성을 제거하기 위한 레지스터 리네이밍이 수행된다. 레지스터 리네이밍이라는 것은 비순차 수행을 하는 프로세서가 명령어를 최대한 이슈할 수 있도록 인접한 명령어의 의존성을 제거하기 위해서 논리적 레지스터를 물리적 레지스터에 할당하는 방법이므로 예외 상황이 발생했을 때 sequential order로 전체 명령어의 순서를 바로 잡을 수 있는 방법을 생각해야 한다.

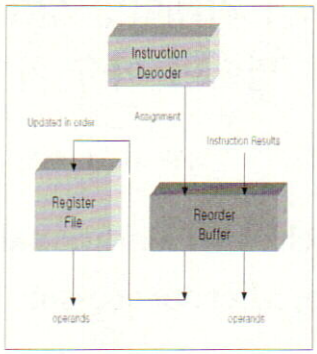

그림 1. Block diagram of reorder buffer

그러한 것을 보장해 주는 하드웨어를 리오더 버퍼를 가지고 구현하는 것이 본 그림에서 나타내는 것이다. 먼저 명령어가 명령어 디코더에서 디코딩이 진행될 때, 리오더 버퍼의 가장 상위 엔트리에 그 명령어가 할당된다. 그리고 그 명령어의 결과는 그 명령어가 functional unit에서 수행이 되고 난 결과를 리오더 버퍼에 쓰게 된다. 그리고 리오더 버퍼는 first-in first out(FIFO) 구조로 이루어져 있으므로 가장 하위 엔트리에 있는 명령어의 완료된 결과 값이 레지스터 파일에 쓰여 지게 된다. 즉 레지스터 파일은 in-order state이고 리오더 버퍼는 lookahead state이다. 만약 명령어가 리오더 버퍼의 최하위 엔트리까지 오는 동안 예외 상황이 발생하지 않는다면 위에서 설명한 과정이 그대로 수행이 된다. 하지만 만약 명령어가 리오더 버퍼의 최하위 엔트리까지 오는 동안 예외상황이 발생한다면 원래 명령어 순서에서 그 이후에 수행되었던 명령어는 flush되고 그 예외 상황 이전에 수행되었던 명령어는 completion이 되어야 한다. 즉 예외 상황이 발생하면 리오더 버퍼의 모든 엔트리의 값들은 버려지게 되는 것이다.

프로세서가 리오더 버퍼 내에 있는 레지스터를 목적 레지스터로 요구하는 경우에 리오더 버퍼는 associative lookup을 해서 목적 레지스터 값을 operands로 내보내야 한다. associative lookup은 목적 레지스터 값을 전부 비교를 해서 값을 내보는

주는 과정이므로 하드웨어의 상당한 부담을 초래하게 된다. 이것은 리오더 버퍼의 커다란 약점으로 지적되는 것이며, 이러한 associative lookup으로 인한 하드웨어의 부담을 제거하기 위해서 미래 파일을 두는 방식의 연구가 생겨나게 되었다.

결론적으로 말해보면, 리오더 버퍼를 사용해서 레지스터 리네이밍을 구현하는 것은 비교적 아키텍처적으로 간단하지만 associative lookup이라는 하드웨어의 부담이 있게 되는 방법이다.

2. Future File

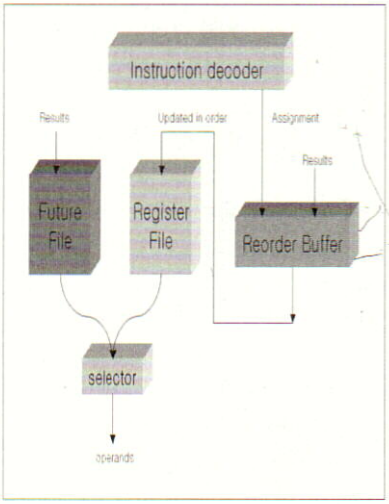

그림 2. Block diagram of future file

앞에서 설명했던 리오더 버퍼의 단점은 associative lookup으로 인한 하드웨어의 부담이 크다는 것이다. 따라서 future file을 이용하는 방법에서는 이를 이용하여 리오더 버퍼에서의 associative lookup에 대한 하드웨어의 부담을 줄이게 된다.

위의 그림 2.는 퓨쳐 파일의 블록 다이어그램을 나타낸다. 이 구조에서는 앞 절에서 설명한 리오더 버퍼만을 두어서 레지스터 리네이밍을 구현하는 것에 비해서 퓨쳐 파일이라는 하드웨어 자원을 더 쓰게 된다. 이 퓨쳐 파일에는 명령어의 수행 결과가 리오더 버퍼에 쓰여 짐과 동시에 퓨쳐 파일에 쓰여 지게 된다. 퓨쳐 파일의 구조는 레지스터 파일과 동일한 구조를 가지게 된다.

먼저 명령어가 디코드 되는 동안 명령어는 리오더 버퍼의 가장 상위 엔트리에 할당이 된다. 그리고 그 해당 명령어가 리오더 버퍼의 최하위 위치에 도달했을 때 까지 예외 상황이 발생하지 않았다면 해당 명령어의 결과 값이 레지스터 파일에 쓰여 지게 된다. 위에서 설명한 리오더 버퍼의 경우와 마찬가지로 예외 상황이 발생된다면 리오더 버퍼의 내용과 퓨쳐 파일의 내용은 버려지게 된다. 따라서 레지스터 파일은 in-order 상태를 유지하게 된다. 그리고 그 해당 명령어의 결과는 레지스터 파일과 동일한 구조를 가지는 퓨쳐 파일에도 쓰여 지게 된다. 따라서 퓨쳐 파일에는 레지스터 파일과 동일한 구조로 되어 있는 하드웨어 구조에 리네이밍 되어진 명령어의 결과가 기록되는 것이다. 이는 위에서 설명했던 리오더 버퍼만을 두었을 때 요구되는 associative lookup이라는 하드웨어의 부담을 줄여주게 된다. 예를 들어 만약 현재 명령어가 리네이밍된 레지스터의 결과 값을 소스로 요구할 경우에, 본 구조에서는 퓨쳐 파일에서 나오는 값과 레지스터 파일에 나오는 두 결과 값 중에서 퓨쳐 파일에 있는 결과를 사용하게 되고 그것은 셀렉터를 통해서 구현이 되는 것이다.

이렇게 하면 associative lookup이라는 리오더 버퍼만을 사용했을 경우에 생기게 되는 하드웨어의 부담은 줄일 수 있게 된다. 하지만 하드웨어 자원의 측면에서 보면 퓨쳐 파일이라는 하드웨어를 추가로 구현해 주어야 하고 퓨쳐 파일을 통해서 나오는 결과와 레지스터 파일을 통해서 나오는 결과 중에 하나를 선택해 주어야 하는 추가 로직이 요구되게 되므로 리오더 버퍼만을 사용한 경우와 비교하여 하드웨어의 부담을 줄였다고는 할 수 없게 된다.

3. Bank-based Register File

리오더 버퍼의 associative lookup이나 미래 파일을 사용함으로 인한 하드웨어의 부담을 줄이기 위한 다음과 같은 방법이 있다. 그림 3.은 이 방법의 블록 다이어그램을 나타낸다. 논리 레지스터 세트를 하나의 뱅크 형태로 정의하여 여러 개의 뱅크를 두고 현 상태의 구조적 상태와 순차적 상태에 대한 뱅크 번호를 저장하는 부분을 두어 해결할 수 있다.

단, 이 경우에는 한 레지스터 영역에는 지정된 논리 레지스터 번호에 해당하는 값만이 기록될 수 있기 때문에 물리 레지스터 저장 영역이 뱅크의 숫자

만큼 증가하게 되어 전체 크기뿐만 아니라 낭비되는 공간 또한 비례해서 커지게 된다.

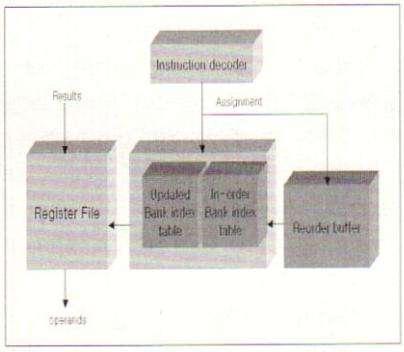

그림 3. Block diagram of Bank Method

또한 뱅크의 숫자가 곧 하나의 레지스터 이름에 대한 최대 리네이밍의 값이 되기 때문에 성능 향상을 위해서는 많은 수의 뱅크를 두는 것이 필요하다.

4. Free list와 함께 우선권 인코더와 rename 상태 지시기를 이용한 방법

위의 3절에서 설명한 것과 같이 리오더 버퍼만을 사용하는 경우에 리오더 버퍼의 associative lookup이 요구되므로 하드웨어의 부담을 가져오게 되고 퓨쳐 파일을 사용할 경우에도 비순차 수행을 위한 레지스터 리네이밍을 가능하게 하기 위해서 레지스터 파일과 동일한 구조의 퓨쳐 파일과 레지스터 파일을 통하는 값과 퓨쳐 파일을 통하는 값 중에서 선택을 하는 로직이 요구되므로 하드웨어에 대한 부담은 여전하다. 이를 해결하기 위해서 논리 레지스터 세트를 하나의 뱅크 형태로 정의하여 여러 개의 뱅크를 두고 현 상태의 구조적 상태와 순차적 상태에 대한 뱅크 번호를 저장하는 부분을 두어 해결하는 방법이 제안되었다. 하지만 이런 경우에도 저장 영역이 뱅크의 숫자만큼 커지게 되어 전체 하드웨어 공간이 커지는 단점을 가지게 된다. 이를 간단하게 한 것이 프리 리스트와 함께 우선권 인코더와 rename상태 지시기를 이용한 방법이다. 우선권 인코더는 위에서 언급한 뱅크와 같은 형태 대신 일반적인 레지스터 파일을 사용하기 위해서 물리 레지스터 할당 지시기와 함께 해당 기록 대상 레지스터의 논리 레지스터 번호를 물리 레지스터 번호로 할당하는 작업을 수행해 준다. 그리고 rename 상태 지시기를 통해서 구조적 상태를 저장하기 때문에 구조적 상태의 레지스터 값을 읽기 위한 리오더 버퍼에서의 내용검색(associative lookup)과 레지스터 값을 저장하기 위한 추가적인 공간이 필요 없다. 그러나 이와 같은 방법에서는 예외상황이 발생한 경우에는 물리 레지스터 주소를 참조하여 프리 리스트와 리네임 상태 지시기의 순차적 상태를 복원해야 하기 때문에 복원 작업을 위해 걸리는 시간도 길고 복잡한 하드웨어를 추가해야 한다.

III. 제안된 구조

본 논문에서 제안하는 구조는 앞장에서 설명했던 4번째 경우인 프리 리스트와 함께 우선권 인코더와 rename상태 지시기를 이용한 방법을 개선한 것이다. 기존의 연구에서의 단점인 순차적 상태를 복원하기 위한 작업을 좀 더 편리하게 하기 위해서 순차적 상태 지시기와 순차적 물리 레지스터 할당 지시기를 통해 해결하는 방법을 제안한다. 순차적 상태 지시기는 순차적인 논리 레지스터의 상태가 어느 물리 레지스터에 기록이 되었는지를 저장하고 순차적 물리 레지스터 할당 지시기는 현재 어느 물리 레지스터에 순차적인 상태가 저장되어 있는지를 저장하여, 복원 작업이 이루어질 경우에 이 두 지시기에 있는 값을 각각 rename 상태 지시기와 물리 레지스터 할당 지시기로 복사해 줄 수 있도록 한다. 복원 작업이 완료되면 이 두 지시기는 reset된다.

본 논문에서 제안하는 것은 레지스터 리네이밍 방법을 사용하는 조건부 실행 비순차적 실행 명령어 이슈 컴퓨팅 시스템에 관한 연구이다. 명령어를 비순차적으로 수행하면서 데이터 의존성을 올바르게 해결하기 위해서는 논리적 목적 레지스터에 대한 쓰기 인스턴스마다 새로운 물리적 레지스터를 할당하고, 논리적 소스 레지스터를 최신의 리네임 상태를 저장하고 있는 물리적 레지스터로 리네이밍하는 것이 필요하다. 또한 분기 예측 오류 및 예외 상황의 발생시에 순차적 상태에서 다시 시작 할 수 있도록 순차적 상태를 추적 및 복구 할 수 있어야 한다.

조건부 실행을 특징으로 하는 명령어 세트 구조를 채택한 비순차적 시스템에서는, 조건부 실행 명령어의 실행 여부가 후속하는 명령어의 레지스터 리네이밍에 미치는 영향 때문에 기존의 방법으로는 레지스터 리네이밍 및 순차적 상태의 추적을 수행할 수가 없다. 따라서 본 논문에서는 기존의 연구인 우선권 인코더와 함께 프리 리스트와 rename상태 지시를 이용하는 방법의 단점을 극복하기 위한 연구를 하였고 여러 하드웨어 자원을 이용해서 이를 해결하는 방법을 설명한다.

1. 전체 블록 및 흐름 설명

그림 4.은 본 논문의 레지스터 리네이밍 방법을 사용하여 비순차적으로 명령어를 이슈하고 수행하는 조건부 실행 컴퓨터 시스템의 전체적인 구성을 보여준다.

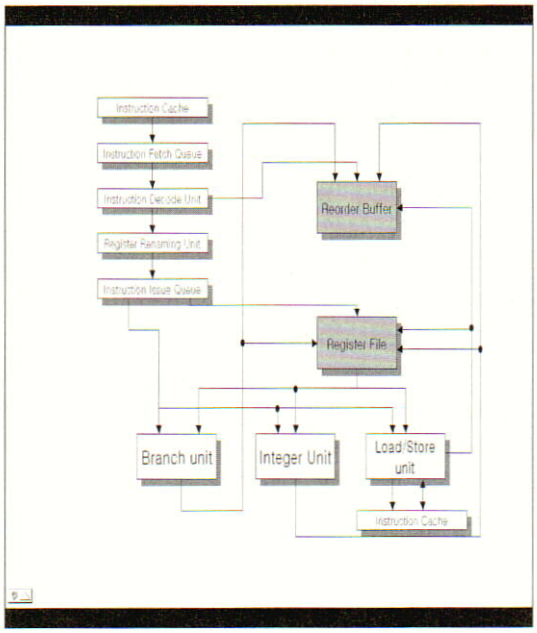

그림 4. Block Diagram of Conditional Execution Out-of-order Computing System Architecture

명령어들은 명령어 캐쉬(instruction cache)로부터 페치(fetch)되어서 명령어 주소(instruction address)와 같이 선입선출 방식의 명령어 페치 큐(instruction fetch queue)에 순서대로 저장된다. 수퍼스칼라 컴퓨터 시스템에서는 시스템의 사양에 따라서 단일 사이클에 페치되는 명령어의 수가 달라진다. 명령어 페치 과정에서는 분기 예측(branch prediction)이 이루어진다.

명령어 페치 큐에 저장된 명령어 중에서 가장 오래된 명령어들은 명령어 디코드 유닛(instruction decode unit)에 의해서 디코드되고, 디코드된 명령어들은 명령어 페치 큐에서 제거된다. 명령어를 디코드하는 과정에서는 명령어를 내부적으로 사용하는 오피 코드(op code)와 소스 및 목적 레지스터 주소들로 변환된다. 여기에서 단일 명령어가 가질 수 있는 목적 레지스터 수는 한 개이며, 소스 레지스터 수는 두 개이다. 명령어 디코드 과정에서는 명령어의 실제 실행 여부를 예측하는 작업도 이루어진다. 이 과정에서 조건부 실행 명령어의 경우는 명령어 주소와 조건 예측 버퍼를 이용해서 실제 수행 여부를 예측한다. 명령어 디코드에 의해서 변환된 명령어들은 레지스터 리네이밍 유닛(register renaming unit)으로 전달되며, 레지스터 리네이밍 유닛은 데이터 의존성을 해결해 주는 역할을 한다. 레지스터 리네이밍 유닛에 의해서 논리적 목적 레지스터에 대해서 물리적 목적 레지스터를 할당하며, 논리적 소스 레지스터에 대해서는 실제로 대응되는 물리적 소스 레지스터(physical source register)로 변환하여 준다. 또한, 조건 레지스터를 갱신하는 명령어나 조건 레지스터의 값을 읽어서 명령어 수행에 사용하는 명령어에 대해서도, 조건 레지스터에 대한 물리적 조건 레지스터 할당이나 조건 레지스터의 읽기에 대한 물리적 조건 레지스터로의 리네이밍이 이루어진다. 다만, 실제로 실행되지 않을 것으로 예측된 명령어에 대해서는 이와 같은 리네이밍 작업의 대부분이 이루어지지 않는다. 상기 명령어 페치, 디코드 및 레지스터 리네이밍은 명령어들의 순차적인 순서대로 이루어진다. 레지스터 리네이밍이 이루어진 명령어는 명령어 이슈 큐로 전달되어 명령어 이슈 큐의 엔트리 하나를 차지한다. 또한 리오더 버퍼에도 목적 레지스터 주소와 물리적 조건 레지스터 주소에 대한 정보가 저장된다. 명령어 이슈 큐는 최하위 포인터(bottom pointer)와 최상위 포인터(top pointer)에 의해 순환적으로 관리된다. 즉 최하위 포인터는 명령어 이슈 큐에 존재하는 명령어 중에서 가장 오래된 명령어 엔트리를 가리키며, 최상위 포인터는 다음에 명령어를 받아들일 엔트리를 가리킨다. 명령어가 명령어 이슈 큐로 전달될 때에 해당 엔트리는 명령어의 실제 실행에 대한 예측 비트(이하 실행 예측 비트라고 함), 오피 코드, 물리적 목적 레지스터 주소, 물리적 소스 레지스터 주소들, 목적 레지스터로서의 물리적 조건 레지스터(이하 물

리적 목적 조건 레지스터(physical destination condition register)라고 함)의 주소, 소스 레지스터로서의 물리적 조건 레지스터(이하 물리적 소스 조건 레지스터(physical source condition register)라고 함)의 주소 및 해당 명령어에 대응되는 리오더 버퍼의 엔트리를 가리키는 포인터를 저장한다. 리오더 버퍼는 각 명령어의 논리적 목적 레지스터 주소 및 물리적 목적 레지스터 주소 양자, 물리적 목적 조건 레지스터 주소, 명령어 주소, 조건 레지스터를 갱신하는 명령어임을 알려주는 비트(이하 조건 갱신 비트라고 함), 조건부 실행 명령어인지를 알려주는 비트(이하 조건부 비트라고 함), 실행 예측 비트 및 명령어 수행 결과로서 실제로 실행되는 명령어인지를 알려주는 비트(이하 실행 결과 비트라고 함)를 명령어들의 순차적인 순서대로 저장하고 있어서, 연속적으로 완료된 명령어 열에 대한 순차적 상태를 추적하기 위해서 사용된다. 리오더 버퍼는 명령어 이슈 큐와 같이 최하위 포인터와 최상위 포인터에 의해서 순환적으로 관리되며, 또한 선입선출 구조를 가진다. 명령어 이슈 큐에 존재하는 명령어에 대해서 소스 레지스터들 및 소스 조건 레지스터의 데이터가 모두 준비되고 사용하고자 하는 기능 유닛 중에서 사용할 수 있는 것이 존재할 경우에, 각 명령어는 기능 유닛으로 이슈된다. 사용하는 기능 유닛의 종류는 명령어의 종류에 따라서 결정되는데, 분기 명령어(branch instruction)는 분기 유닛(branch unit)으로, 정수 연산 명령어(integer arithmetic instruction)는 정수 연산 유닛(integer arithmetic unit)으로, 메모리(memory)를 엑세스(access)하는 명령어는 로드/스토어 유닛(load/store unit)으로 이슈된다. 명령어 이슈는 비순차적으로 이루어질 수 있기 때문에 선행 명령어가 이슈되지 않은 상태로 명령어 이슈 큐에 존재하더라도 후행하는 명령어가 먼저 이슈되고, 명령어 이슈 큐에서 빠져나갈 수 있다. 명령어가 이슈된 후에 해당 엔트리는 비어있게 되고, 명령어 이슈는 비순차적으로 이루어지기 때문에 명령어 이슈 후에는 중간에 비워 있는 엔트리들을 이슈되지 않은 엔트리들을 사용하여서 압축(compressing)할 필요가 있다. 예를 들어서 명령어 이슈 후에 최상위 포인터와 최하위 포인터 값의 차이가 7이고, 최상위 엔트리와 최하위 엔트리 사이에 유효하게 명령어를 가지고 있는 엔트리가 세 개일 때에는 유효한 엔트리들을 최상위 엔트리쪽으로 최대한 천이(shift)시키고, 최하위 포인터 값을 조정하여, 최상위 포인터와 최하위 포인터 값이 차이가 3이 되도록 하며, 최상위 엔트리와 최하위 엔트리 사이에 존재하는 모든 엔트리들은 유효하게 명령어를 포함하도록 한다. 기능 유닛으로 이슈된 명령어들은 명령어 수행을 완료한 후에는 그 결과와 조건 레지스터 결과를 각각 물리적 목적 레지스터와 물리적 조건 레지스터에 쓰면서, 리오더 버퍼에게 명령어가 완료되었음을 알려준다. 후에 해당 엔트리가 리오더 버퍼의 최하위 엔트리가 되었을 때에는 그 엔트리는 커미트되고, 대응하는 명령어의 수행 결과를 순차적 상태로 만든다.

2. 구성 유닛

본 논문에서는 일반 레지스터의 순차적 상태 및 최신의 리네임 상태를 저장하는 물리적 레지스터들을 모두 포함하는 물리적 레지스터 어레이(array)로 구성된 레지스터 파일(register file), 순차적 상태 지시기(in-order state indicator), 리네임 상태 지시기(rename state indicator), 물리적 레지스터 할당 지시기(physical register allocation indicator), 조건 레지스터의 순차적 상태 및 최신의 리네임 상태를 저장하는 물리적 조건 레지스터(physical condition register)들을 모두 포함하는 조건 레지스터 어레이(condition register array), 순차적 조건 상태 지시기(in-order condition state indicator), 리네임 조건 상태 지시기(rename condition state indicator), 물리적 조건 레지스터 할당 지시기(physical condition register allocation indicator) 및 물리적 조건 레지스터 할당 수(count of allocated physical condition registers) 등을 사용하여, 명령어의 조건부 실행을 특징으로 하는 명령어 세트 구조를 채택한 비순차적 컴퓨터 시스템에서 효율적이면서 저 비용으로 일반 레지스터와 조건 레지스터에 대한 물리적 레지스터의 할당과 리네이밍 및 순차적 상태의 추적과 복구를 관리한다.

본 논문의 컴퓨터 시스템은 순차적 상태를 나타내는 물리적 레지스터와 미리보기 상태를 나타내는 물리적 레지스터들 양자를 모든 논리적 레지스터들이 공유할 수 있도록 포함하고 있는 레지스터 파일, 레지스터 파일 내의 물리적 레지스터 중에서 어느 것들이 각각의 논리적 레지스터의 순차적 상태를 나타내는지를 지시하는 순차적 상태 지시기, 레지스터 파일 내의 물리적 레지스터 중에서 어느 것들이 각각의 논리적 레지스터의 구조적 상태(최신의 리네임 인스턴스)를 나타내는지를 지시하는 리네임 상태

지시기, 물리적 레지스터 중에서 어느 것들이 논리적 레지스터들에 할당되고, 어느 것들이 프리(free)해서 새로운 물리적 레지스터의 할당에 사용될 수 있는지를 알려주는 물리적 레지스터 할당 지시기, 조건 레지스터의 상이한 리네임 인스턴스들을 나타내는 물리적 조건 레지스터들을 포함하는 조건 레지스터 어레이, 조건 레지스터 어레이 중에서 어떤 물리적 조건 레지스터가 논리적 조건 레지스터의 순차적 상태를 저장하고 있는지를 지시하는 순차적 조건 상태 지시기, 논리적 조건 레지스터의 구조적 상태(최신의 리네임 인스턴스)를 저장하고 있는 물리적 조건 레지스터를 알려주는 리네임 조건 상태 지시기, 논리적 조건 레지스터 갱신에 할당할 물리적 조건 레지스터를 지시하는 물리적 조건 레지스터 할당 지시기, 할당된 물리적 조건 레지스터의 수를 지시하는 물리적 조건 레지스터 할당 수, 조건부 실행 명령어의 실행 비실행 여부를 예측하기 위한 조건 예측 버퍼(condition prediction buffer) 및 미리보기 상태의 물리적 레지스터 주소 값들을 선입선출 형식으로 보유하고 있는 리오더 버퍼를 가지는 것을 특징으로 한다. 이러한 구성 요소들에 의해서 본 발명은 명령어의 조건부 실행을 특징으로 하는 명령어 세트 구조를 채택한 비순차적 명령어 이슈 컴퓨터 시스템(이하 조건부 실행 비순차적 시스템이라는 명칭 사용)에서의 레지스터 리네이밍, 물리적 레지스터의 할당과 퇴거 및 순차적 상태 추적을 관리한다.

다음으로 본 논문의 구조에서 특징되는 물리 레지스터 할당 지시기(physical register allocation indicator)와 우선권 인코더 그리고 순차적 포인터 어레이와 순차적 물리 레지스터 할당 지시기에 대해서 설명하도록 하겠다. 먼저 물리 레지스터 할당 지시기와 우선권 인코더가 있다. 물리 레지스터 할당 지시기는 그림 5.의 상위의 블록에 보여 지며 모든 물리 레지스터에 대한 free-bit을 포함한다. 그리고 그것은 명령어가 디코딩될 때 물리 레지스터 주소들이 target logical register들로 할당 되도록 검색되어진다. free-bit의 초기값은 1이고 그것은 관련되는 physical register가 아직 할당되어지지 않았다는 것을 의미한다. target register에 대해서 새로운 물리 레지스터 주소가 할당될 때 우선권 인코더는 물리 레지스터 할당 지시기로부터 free한 물리 레지스터 주소들을 선택하고 선택된 free-bit는 0으로 clear 된다.

두 번째로 그림 4.의 아래쪽 블록에 renaming state pointer array(RSPA)가 있다. 그것은 물리 레지스터가 architectural state value를 가지는 현재의 논리 레지스터를 나타내는 정보를 제공한다. 명령어가 새롭게 디코드될 때 소스 레지스터들의 논리 레지스터 주소들은 RSPA를 사용해서 물리 레지스터 주소들로 변환된다. 결국 모든 명령어에 대해서 타겟 레지스터들의 논리 레지스터 주소들은 PRAI를 사용해서 물리 레지스터 주소들로 변환되며 그러한 주소들이 RSPA를 update하는데 사용되어진다.

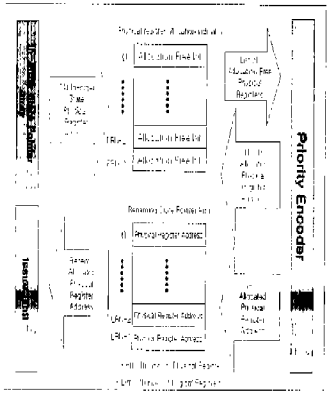

그림 5. Physical register allocation indicator

본 논문에서 사용되는 리오더 버퍼는 보통의 비순차 이슈 마이크로 프로세서에서 사용되는 리오더 버퍼를 간단하게 한 것이다. 이슈 큐는 디코드된 명령어를 저장하며 명령어간의 의존성이 없는 이슈 큐안의 명령어들이 이슈 되어진다. 리오더 버퍼는 디코드된 명령어의 레지스터 정보를 순서대로 저장한다. 가장 최근에 디코드된 명령어에 대한 정보는 리오더 버퍼의 가장 상위 엔트리에 할당되어 진다. 리오더 버퍼내의 엔트리는 complete-bit, 논리 레지스터 주소 공간 그리고 target 레지스터의 물리 레지스터 주소 공간으로 구성되어 있다. 수행이 완료된 명령어는 물리 레지스터 주소를 가진 레지스터 파일에 결과를 저장한다. 그리고 리오더 버퍼내에 관련된 엔트리의 complete-bit는 명령어가 끝난 것을 나타내도록 셋팅되어 진다. 만약 리오더 버퍼 내

에 최하위 엔트리의 완료 비트가 셋팅이 된다면 그 엔트리는 retire된다.

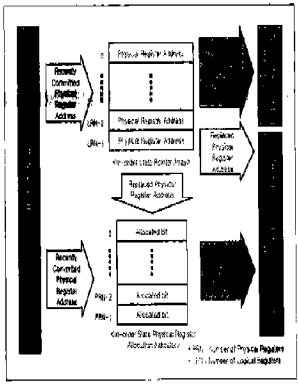

그림 6. In-order state pointer array and an in-order state physical register allocation indicator

그림 6.에 보여 지는 것은 in-order state pointer array(ISPA) 그리고 in-order state physical register allocation indicator(IPRAI)이다. 그것은 물리 레지스터들이 순차적 상태를 가지는 논리 레지스터들에 할당되어지는 것에 대한 정보를 제공한다. 리오더 버퍼 엔트리가 retire될 때 그 엔트리에 논리 레지스터 주소는 그것의 물리 레지스터 주소를 ISPA에 저장하는데 사용되어진다. 그리고 그 엔트리내의 물리 레지스터 주소는 IPRAI내에 관련된 allocation-bit를 셋팅하는 데 사용된다. 이전 in-order state를 가지는 이전 물리 레지스터 주소가 새로운 것으로 대치될 때 그 대치된 주소는 재할당하기 위한 PRAI내에 관련되는 free-bit을 셋팅하는 데 사용되어지고 물리 레지스터가 더 이상 in-order state를 가지지 않는다는 것을 나타내기 위해서 IPRAI내의 관련된 allocation-bit을 clear하는데 사용된다.

3. 제안된 구조의 동작 흐름

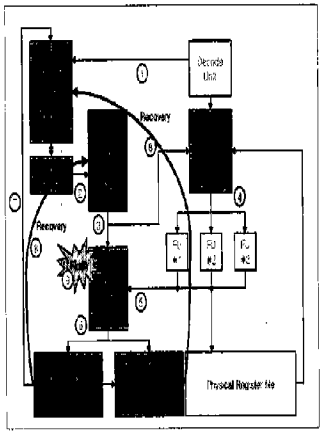

그림 7. Block diagram of the architecture suggested in this paper

그림 7.은 본 논문에서 제안하는 구조의 전체 블록 다이어그램을 보여준다. 그림에서 보여 지는 번호를 본 절에서는 설명 앞에 붙여서 전체적인 동작을 설명하도록 하겠다. ① 만약 하나 이상의 디코드된 명령어들이 있다면 그들 사이에 외존성을 검사하고 만약 의존성이 있다면 리네임된 값이 그 명령어에 주어진다. 그리고 만약 의존성이 없다면 PRAI(physical register allocation indicator)는 타겟 명령어의 논리 레지스터 주소를 사용함으로서 새로운 물리 레지스터 주소를 제공하며 ②RSPA(Renaming state pointer array)로 새로운 물리 레지스터 주소를 보낸다. ③ 리네임된 레지스터 주소들은 이슈 유닛과 리오더 버퍼로 보내진다. ④ 의존성이 없는 명령어들은 그 명령어에 해당하는 기능 유닛으로 이슈가 되고 그러한 결과는 물리 레지스터 파일에 저장이 된다. ⑤ 명령어가 완료가 될 때 리오더 버퍼 내에 관련된 엔트리의 complete-bit이 셋팅된다. ⑥ 리오더 버퍼 내에 최하위 엔트리의 complete-bit가 셋팅될 때 그 엔트리는 retire되고 그것의 정보는 순차적 상태 정보를 저장하기 위해서 ISPA(in-order state

pointer array)와 IPRAI(in-order state physical register allocation indicator)로 보내진다. ⑦ 새롭게 update되는 정보가 ISPA에 저장이 될 때 예전 정보는 그 정보에 의해서 밀리게 되고 이전에 순차적 상태를 가졌던 물리 레지스터를 free하기 위해서 PRAI로 보내진다. ⑧ exception이 발생한다면, 그 예외 상황은 리오더 버퍼 내에 감지가 되어진다. 예외 상황이 감지된 후에 ISPA의 복록들은 RSPA로 복사되고 IPRAI내의 allocation-bit는 PRAI로 복사가 된다. 그 후에 ISPA와 IPRAI의 모든 엔트리들은 리셋된다. ⑨ 결국 잘못된 정보를 가지는 리오더 버퍼는 flush된다.

4. 조건 예측 실행의 정확성

본 논문에서는 앞 절에서 설명한 하드웨어 자원을 이용해서 레지스터 리네이밍을 수행하게 된다. 레지스터 리네이밍 방법은 비순차적 컴퓨터 시스템의 구현을 위해서 필수적인 것이다. 하지만 이전의 레지스터 리네이밍 방법들은 구조상으로 조건부 실행을 특징으로 하는 컴퓨팅 시스템에는 적용될 수가 없는 것이었기 때문에 조건부 실행을 특징으로 하는 컴퓨터 시스템을 비순차적으로 시스템으로 구성하는 것이 불가능했다. 그러나 본 논문에서 제안하는 구조는 조건부 실행을 특징으로 하는 비순차적 컴퓨터 시스템의 구현을 가능하게 하였고 조건 실행 예측의 정확성이 기반이 될 경우에 컴퓨터 시스템의 전체 성능향상을 꾀할 수 있으므로 본 논문에서 제안하는 구조의 유용성을 알 수 있게 된다.

비순차적 명령어 수행을 하는 컴퓨팅 시스템에 조건부 실행 명령어를 지원하는 것을 가능하게 하기 위해서 조건을 예측해서 실행을 하게 된다.

조건 예측을 수행할 경우의 정확성에 대해서 ARMTM architecture version 5를 target ISA(Instruction Set Architecture)로 사용하였다. ARM ISA에서 거의 모든 명령어들은 status register의 조건 flags에 따라서 조건부로 수행되어질 수 있다. 본 시뮬레이션에서는 workloads로서 ARM Development SuiteTM의 Dhrystone, Sort와 SPEC benchmark 2000의 Parser, Vortex, Twlf, mcf를 가지고 trace-driven 방식으로 시뮬레이션을 수행하였다. 그림 8.은 전체 시뮬레이션 흐름을 보여준다.

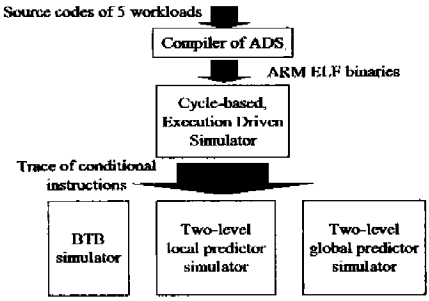

그림 8. Simulation flow

본 시뮬레이션은 조건을 예측해서 수행하는 경우에 어느 정도의 정확성이 타나나는지에 대한 것이고 시뮬레이션 결과를 통해서 조건 예측의 정확성이 높다는 것을 확인할 수 있다.

표1. 조건 예측 수행된 경우의 정확성

	Predict Taken	Bimodal (512 entry)	Global (10 history bits)	Local (256 entry and 12 history bits)
Result	69.274%	89.908%	92.374%	93.067%

표에서 보는 바와 같이 predict taken, bimodal, global, local방식에 대해서 각각 시뮬레이션을 수행하였으며 각각의 accuracy를 나타내었다.

Predict taken으로 수행된 경우 정확성이 69.2%이고 512 entry를 가진 bimodal인 경우는 89.9%이고 10개의 history bit을 가지고 있는 경우는 92.3%이고 256개의 entry와 12개의 history bit을 가지는 경우에는 정확도가 93%가 된다. 즉 예측의 정확성은 거의 90%가 된다. 이와 같이 조건 예측 수행을 하는 경우의 정확성이 증가하므로 비순차적 명령어 수행 컴퓨팅 시스템에 조건 예측 수행을 하게 되면 성능이 증가함을 알 수 있다. 따라서 조건 예측의 정확성이 기반이 될 경우에 전체 시스템의 성능이 증가하게 되고 본 논문의 제안된 구조가 비순차적 컴퓨팅 시스템에서의 조건부 실행 명령어를 지원하므로 전체 시스템의 성능 향상에 기여하게 될 것이다.

결론적으로 본 논문에서 제안된 구조는 비순차적 명령어 수행 컴퓨팅 시스템에 조건부 수행 명령어

의 추가를 가능하게 함으로서 성능을 향상시키게 된다. 그리고 시뮬레이션 결과를 통하여 prediction이 수행될 경우에 accuracy가 정확함을 알 수 있다. 따라서 조건 예측 수행이 정확하기 때문에 조건부 실행 명령어의 지원을 가능하게 하는 본 논문에서의 구조가 전체 시스템의 성능 향상에 영향을 미치게 될 것이다.

5. 제안된 구조와 기존 방법의 비교

레지스터 리네이밍, 물리적 레지스터의 할당과 회거 및 순차적 상태 추적을 관리하는 것에 관하여 본 논문의 기반이 되는 방법은 다음과 같이 기존 방법들의 단점들을 제거한다.

본 논문의 기반이 되는 방법은 순차적 상태를 나타내는 레지스터들로 lookahead상태를 나타내는 레지스터들의 내용을 전송하는 과정을 제거한다. 그리고 associative lookup을 피하도록 하였으며, 레지스터 리네이밍 과정과 레지스터 읽기를 분리할 수 있도록 하여, 각각은 별도의 파이프 라인 단계에서 이루어지게 하여 클럭 주파수를 높일 수 있도록 한다. 그리고 보통의 단일 레지스터 어레이가 lookahead 상태의 물리적 레지스터와 순차적 상태의 물리적 레지스터를 구분하지 않고 보유하도록 하여, 레지스터 값을 읽고 쓰는 과정을 순차적 이슈 시스템의 것으로 단순화 한다. 또한 레지스터 어레이 내의 모든 물리적 레지스터는 어떤 논리적 레지스터에도 할당될 수 있도록 하였다. 이와 같이 논리적 레지스터들 간에 모든 물리적 레지스터들을 공유함으로서 레지스터 파일의 크기를 최소화하고 물리적 레지스터들의 활용도와 시스템의 성능을 높이도록 한다.

Ⅳ. 결 론

본 논문에서 제안된 구조는 위와 같은 장점들을 가진 것을 기반으로 해서 명령어의 조건부 실행을 특징으로 하는 명령어 세트 구조를 채택한 비순차적 명령어 이슈 컴퓨터 시스템에서 레지스터 리네이밍 방법과 물리적 레지스터의 할당과 회거 및 순차적 상태 추적을 효율적으로 관리하는 방법을 제공한다. 기존의 레지스터 리네이밍을 지원하는 방법들이 명령어의 조건부 실행을 특징으로 하는 명령어 세트 구조를 채택한 비순차적 컴퓨터 시스템에 적용될 수 없다는 점을 고려하면 본 논문에서 제안된 구조가 우수함을 알 수 있으며 시뮬레이션을 통해서 살펴 본 조건 실행 예측의 정확성이 기반이 될 경우에는 컴퓨터 시스템의 전체 성능이 크게 향상된다는 사실은 제안된 구조가 비순차적 명령어 수행 컴퓨팅 시스템에서의 조건부 실행 명령어를 지원한다는 것이 컴퓨터 전체 시스템의 성능 향상에 영향을 미친 다는 것을 알 수 있다. 그리고 기존 레지스터 리네이밍 방법에 비해서 적은 하드웨어 비용으로 associative lookup을 제거하고 짧은 복구 시간을 가지는 구조를 제안하였다.

참 고 문 헌

[1] Mike Johnson, Superscalar Microprosseor Design, prentice hall, Inc, 1991

[2] S.J. Nam, I.C. Park, and C.M. Kyung, "Fast Precise Interrupt Handling without associative Searching in Multiple Out-Of-Order Issue Processor", *IEICE Trans. On Information and Systems*, vol.E82-D, no3, pp.645-563, Mar. 1999.

[3] Stephan Jourdan, Ronny Ronen, Michael Bekerman, Bishara Shomar, and Adi Yoaz, "A novel Renaming Scheme to Exploit Value Temporal Locality through Physical Register Reuse and Unification", *MICRO-31. Proceedings. 31st Annual ACM/IEEE International Symposium on 1998*, pp.216-225, 1998.

[4] J.E. smith and A.R. Pleszkun, "Implementation of Precise Interrupts in Pipelined Processors", *IEEE Transactions on Computers*, vol.37, no.5, pp.562-573, May 1988.

[5] D. sima, "The design space of register renaming techniques", *Micro, IEEE*, vol:20 Issue:5, september - october, 2000.

[6] M. Moudgill, K. Pingali, S. Vassiliadis, "Register Renaming and Dynamic Speculation: an Alternative Approach", *Microarchitecture, 1993. Proceedings of the 26th Annual International Symposium on*, pp.202-213, 1-3 dec. 1993.

최 규 백(Kyu-Baik Choi) 준회원

2001년 8월 : 중앙대학교 전자전기공학부 졸업
2001년 9월~현재 : 연세대학교 전기전자공학과 석사과정

<주관심분야> 마이크로 프로세서, 네트워크 프로세서, ASIC

심 문 경(Moon-Gyung Kim) 준회원

1997년 2월 : 연세대학교 전자공학과 졸업
1999년 2월 : 연세대학교 전자공학과 석사
1999년 3월~현재 : 연세대학교 전기전자공학과 박사과정

<주관심분야> 마이크로프로세서, 네트워크프로세서

홍 인 표(In-Pyo Hong) 준회원

1999년 2월 : 연세대학교 전자공학과 졸업
2001년 2월 : 연세대학교 전기전자공학과 석사
2001년 3월~현재 : 연세대학교 전기전자공학과 박사과정

<주관심분야> 마이크로프로세서, 네트워크프로세서

이 용 석(Yong-Surk Lee) 정회원

1973년 2월 : 연세대학교 전기공학과 졸업
1977년 2월 : Univ. of Michigan, Ann Arbor 석사
1981년 2월 : Univ. of Michigan, Ann Arbor 박사
1993년 3월~현재 : 연세대학교 전기전자공학과 교수

<주관심분야> 마이크로프로세서, 네트워크프로세서