論文97-34C-9-5

명령어 버퍼를 이용한 최적화된 수퍼스칼라 명령어 이슈 구조

(An Optimized Superscalar Instruction Issue Architecture Using the Instruction Buffer)

文 秉 仁 *, 李 龍 煥 *, 安 相 俊 *, 李 溶 錫 *

(Byung In Moon, Yong Hwan Lee, Sang Jun An, and Yong Surk Lee)

요 약

수퍼스칼라 구조는 기존의 스칼라 구조보다 좋은 성능을 얻기 위해 여러 명령어들을 동시에 실행한다. 이와 같이 명령어들을 동시에 실행하기 위해서 프로세서 구조는 다수의 기능 유닛들을 보유하고, 그 유닛들에 여러 개의 명령어들을 동시에 이슈할 수 있어야 한다. 그러나 명령어들이 상호 의존성을 지닌 경우가 있으며, 이러한 경우 그 명령어들은 동시에 이슈될 수 없다. 본 논문은 단일 사이클에 최대 4개의 명령어를 이슈할 수 있는 수퍼스칼라 마이크로프로세서의 이슈 유닛(issue unit)의 설계 및 검증에 관하여 기술한다. 이슈 유닛은 프리페치 유닛으로부터 명령어들을 제공받으며, 단일 사이클에 최대 4개의 명령어를 순차적 방식으로 이슈하여 기능 유닛들을 최대한 활용하도록 하였다. 또한 이슈 유닛은 명령어 버퍼를 사용하여 명령어의 페치와 이슈를 분리시킴으로써, 명령어 이슈율(instruction issue rate)을 높였다. 이슈 유닛은 명령어 버퍼(instruction buffer) 및 명령어 해석기(instruction decoder)로 구성되었다. 명령어 버퍼는 프리페치 유닛(prefetch unit)에서 입력된 명령어들을 정렬하여 저장하고 최대 4개의 명령어를 명령어 해석기에 순서대로 제공한다. 명령어 해석기는 명령어를 해석하고, 기능 유닛(functional unit)과 레지스터파일(register file)의 사용 가능성 및 데이터 의존성(data dependency)을 고려하여 실행 유닛으로 이슈하는 기능을 한다. 이슈 유닛은 기능 수준(behavioral level)에서 HDL(Hardware Description Language)을 이용하여 기술되었다. C 프로그램을 사용하여 모의실험(simulation)을 수행한 결과 명령어 버퍼의 크기가 커질수록 명령어 이슈율이 향상됨을 알 수 있었다. 그리고 성능과 하드웨어 비용을 고려하였을 때 12개의 명령어를 저장할 수 있는 명령어 버퍼가 최적의 것임을 알 수 있었다.

Abstract

Processors using the superscalar architecture can achieve high performance by executing multiple instructions in a clock cycle. It is made possible by having multiple functional units and issuing multiple instructions to functional units simultaneously. But instructions can be dependent on one another and these dependencies prevent some instructions from being issued at the same cycle. In this paper, we designed an issue unit of a superscalar RISC microprocessor that can issue four instructions per cycle. The issue unit receives instructions from a prefetch unit, and issues them in order at a rate of as high as four instructions in one cycle for maximum utilization of functional units. By using an instruction buffer, the unit decouples instruction fetch and issue to improve instruction issue rate. The issue unit is composed of an instruction buffer and an instruction decoder. The instruction buffer aligns and stores instructions from the prefetch unit, and sends the earliest four available instructions to the instruction decoder. The instruction decoder decodes instructions, and issues them if they are free from data dependencies and necessary functional units and register file ports are available. The issue unit is described with behavioral level HDL (Hardware Description Language). The result of simulation using C programs shows that instruction issue rate is improved as the instruction buffer size increases, and 12-entry instruction buffer is found to be optimum considering performance and hardware cost of the instruction buffer.

* 正會員, 延世大學校 電子工學科

(Dept. of Electronic Eng., Yonsei University, Korea)

接受日字:1997年4月24日, 수정완료일:1997年9月3日

Ⅰ. 서 론

프로세서 구조 연구 분야에서 최근에 보여준 발전과 반도체 공정 기술의 발달은 고성능의 마이크로프로세서의 개발로 이어졌다. 그러나 정보화 사회가 진전됨에 따라 점점 더 대용량화 되어가는 응용 프로그램과 데이터를 효율적으로 지원하기 위해서 더욱 더 고성능의 마이크로프로세서의 지속적인 개발이 필요하게 되었다. 이러한 요구에 따라 현재 활발히 연구되고 있는 프로세서 구조 중에는 수퍼스칼라 마이크로프로세서[1,2], VLIW(Very Long Instruction Word)[2,3], 멀티쓰레디드(multithreaded) 방식[4], 멀티프로세서[5] 및 네트워크 컴퓨터 등이 있다. 이 중에서 수퍼스칼라 마이크로프로세서는 기존의 응용 프로그램과 호환성을 유지하면서 명령어 수준 평행성(instruction level parallelism)을 이용하여 성능 향상을 꾀할 수 있다는 장점이 있다.

수퍼스칼라 마이크로프로세서는 여러 개의 기능 유닛들을 보유하고 있으며 각 기능 유닛은 각기 명령어를 실행함으로써 여러 명령어가 동시에 실행된다. 그러나 여러 개의 기능 유닛을 보유하고 있다고 해서 단일 사이클에 여러 개의 명령어를 동시에 실행할 수 있는 것은 아니다. 단일 사이클에 여러 명령어의 입력 오퍼랜드를 제공하고 결과를 저장하기 위해서 레지스터파일의 포트 수를 확장해야 하며 병렬적인 파이프라인(pipeline) 구조를 갖추어야 한다. 그리고 기능 유닛들이 실행할 충분한 명령어들을 기능 유닛들에 공급하여야 한다. 즉 기능 유닛들에 대한 명령어 공급이 부족할 때에는 아무 동작도 하지 않는 기능 유닛의 수가 늘어나게 되고 그만큼 프로세서 성능은 낮아진다. 따라서 기능 유닛과 그 외의 자원들을 충분히 활용하면서 프로세서 전체의 성능을 향상시키는데 가장 크게 영향을 미치는 것은 기능 유닛들에 실행할 명령어들을 이슈하는 방법 및 구조이다. 명령어 이슈에 있어서 가장 크게 장애가 되는 것은 명령어들 사이에 존재하는 데이터 의존성 및 분기 명령으로 인한 명령어 흐름의 변경이다. 데이터 의존성이 있는 경우라도 명령어 실행을 계속하여 성능을 향상시키기 위한 비순차적 명령어 이슈(out-of-order instruction issue)[6,7] 방법에 대한 연구가 계속 진행되어 오고 있다. 그러나 비순차적 명령어 이슈 방법은 추가적인 데이터 의존성[8]을 검출해야 하는 어려움이 있거나 명령어 창(instruction window)[2,9] 또는 레지스터 리네이밍(register renaming)[2] 등이 필요하여 하드웨어가 매우 복잡해질 뿐 아니라 프로세서의 면적에도 큰 문제를 제기한다. 이에 반하여 순차적 이슈(in-order instruction issue) 방법은 이슈 구조의 설계는 간단하지만 성능이 낮아진다는 단점이 있다.

본 논문에서는 순차적 이슈 방법의 간단함을 이용하면서 성능을 높이기 위해서 명령어 버퍼를 사용하였다. 기존의 이슈 방법에서는 명령어 페치(instruction fetch) 다음 사이클에 명령어를 이슈하는 방법을 취한다. 특히 RISC[10] 방식을 이용하는 프로세서는 명령어 해석이 간단하기 때문에 이와 같은 방법을 많이 사용한다. 그러나 명령어 캐쉬(instruction cache)에서 태그 미스(tag miss)가 발생하거나 분기 명령어의 목적 명령어(branch target instruction) 주소가 정렬(align)되지 않은 경우 이슈할 명령어를 충분히 공급하지 못하게 된다. 명령어 버퍼는 명령어 페치와 이슈를 분리시키고 명령어들을 정렬 및 병합(merge)[2]하여 이슈할 수 있게 함으로써 명령어 이슈율을 높인다. 본 논문에서는 명령어 버퍼의 사용에 의한 성능 향상을 HDL을 이용한 모의실험에 의해 보여준다.

본 논문의 나머지 부분은 다음과 같다. Ⅱ장에서는 명령어 버퍼를 사용하여 명령어들을 정렬하여 이슈하는 방법에 대해 자세히 다룬다. Ⅲ장에서는 이슈 유닛 설계의 기반이 되는 전체 마이크로프로세서의 구조 및 이슈 규칙에 대해서 설명한다. Ⅳ장에서는 명령어 이슈 구조를 설명하고, Ⅴ장에서는 이슈 유닛에 대한 모의실험 및 성능 평가를 통해 명령어 버퍼를 이용한 이슈 구조의 개선된 성능을 보여준다. 마지막으로 결론에서 본 이슈 구조의 적당성에 대해서 재검토한다.

Ⅱ. 명령어 버퍼에 의한 명령어 정렬 및 병합

프로세서 전체의 성능은 명령어 이슈율에 크게 의존하며, 명령어 이슈율은 명령어들 사이에 존재하는 데이터 의존성 및 명령어 페치의 효율성에 크게 의존한다. 특히 수퍼스칼라 마이크로프로세서와 같이 단일 사이클에 여러 명령어를 동시에 이슈하고 실행하는 구조는 명령어 페치의 효율성에 따라 성능이 달라진다. 따라서 수퍼스칼라 구조에서는 단일 사이클에 여러 개의 명령어를 동시에 페치하여 명령어 해석기에 공급한

다. 예를 들어 4-way 수퍼스칼라 구조는 단일 사이클에 4개 정도의 명령어를 동시에 페치해야 좋은 성능을 가질 수 있다.

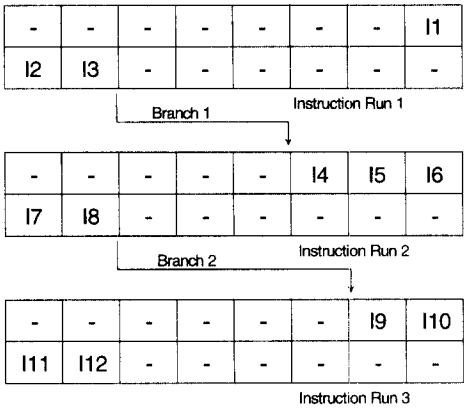

그림 1. 정확히 정렬되지 못한 명령어 열의 예
Fig. 1. Example of Misaligned Instruction Runs.

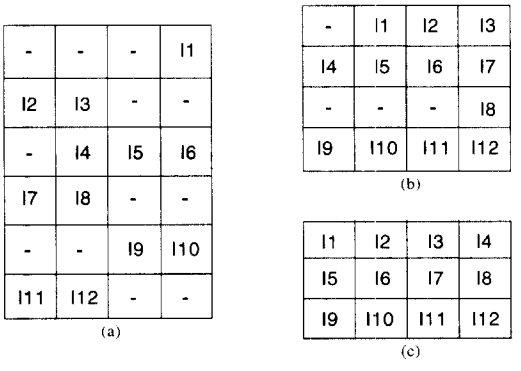

그림 2. 그림 1에 있는 명령어들의 이슈 사이클
(a) 정렬 및 병합을 사용하지 않은 경우; (b) 정렬을 사용하는 경우; (c) 정렬 및 병합을 사용하는 경우

Fig. 2. Issue Cycles of Instructions in Fig. 1. (a) without Aligning or Merging; (b) with Aligning; (c) with Aligning and Merging

동시에 여러 개의 명령어를 페치함으로써 명령어 이슈율을 높일 수 있지만 분기 명령어로 인해 발생하는 명령어 흐름 의존성(procedural dependency)[11] 때문에 명령어 페치의 속도가 느려진다. 이 문제는 분기 예측(branch prediction)을 사용하여 해결이 가능하다. 그러나 분기 예측을 사용하더라도 그림 1과 같이 캐쉬 라인(cache line)에서 명령어 열(instruction run)이 정확히 정렬되어 있지 않으면 그림 2-(a)와 같이 명령어 자원을 크게 낭비하게 된다. 보통의 응용 프로그램이 평균 5개 또는 6개의 명령어마다 1개의 분기 명령어를 가지고 있다는 것을 고려한다면 명령어들이 정확히 정렬되지 못한 것으로 인한 성능의 저하는 매우 크다. 이 문제는 프로그램을 컴파일할 때 분기 명령어의 위치와 분기 목적 명령어의 위치를 조정함으로써 부분적으로는 해결이 가능하지만 완전한 해결책은 되지 못하다. 그러나 하드웨어가 동적으로 명령어들을 정렬하여 명령어 해석기에 보내줄 수 있고 명령어들 사이에 상호 의존성이 없으면 그림 2-(b)와 같이 명령어 이슈율을 높일 수 있다. 그런데 명령어들을 정렬만 하여 이슈할 경우에는 분기 명령어들 사이의 명령어 열에 한하여 효율적인 재배치 동작이 이루어지므로 작지만 낭비되는 이슈 자원이 존재한다. 따라서 정렬된 명령어 열들을 병합(merge)하여 이슈할 수 있다면 그림2-(c)와 같이 더욱 효율적인 명령어 이슈를 할 수 있다.

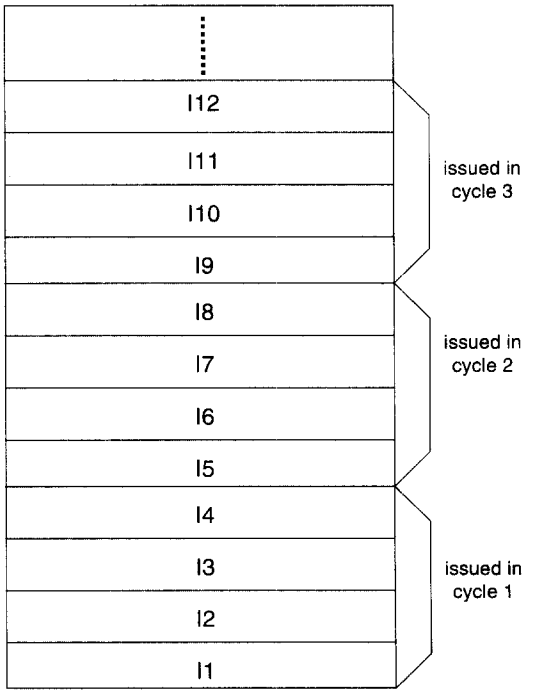

그림 3. 명령어 버퍼를 이용한 명령어의 정렬 및 병합
Fig. 3. Aligning and Merging Instructions Using Instruction Buffer.

본 논문에서는 명령어를 정렬 및 병합하기 위한 방

법으로서 명령어 버퍼를 사용하였다. 명령어를 프리페치(prefetch)하여 명령어 버퍼에 미리 저장하고, 순서대로 저장된 명령어를 명령어 해석기에 보냄으로써 그림 3과 같이 정렬 및 병합된 상태로 명령어들을 해석하여 이슈할 수 있다. 명령어 버퍼는 FIFO 형식의 버퍼이어야 하며, 이것은 버퍼 내에 천이(shift) 기능을 포함시키거나 2개의 포인터(pointer)를 사용하여 구현할 수 있다.

명령어 버퍼가 없는 경우에는 명령어 페치와 이슈 중에서 어느 한 동작이 정지되면 다른 동작도 정지된다. 그러나 버퍼를 사용함으로써 이슈가 정지되더라도 명령어 페치는 계속될 수 있으며, 명령어 페치가 정지되어도 명령어 버퍼에 있는 명령어들을 사용하여 명령어 이슈를 계속할 수 있다. 명령어 버퍼는 이와 같이 명령어 정렬 및 병합 그리고 명령어 페치와 이슈의 분리(decoupling) 기능을 제공함으로써 명령어 페치 및 이슈 성능을 크게 향상시킨다.

Ⅲ. 이슈 규칙의 결정

이슈 구조를 결정하기 위해서는 우선 전체 마이크로프로세서의 구조 및 그 구조에 맞는 이슈 규칙을 결정해야 한다. 본 논문에서는 이슈 유닛이 사용될 수퍼스칼라 마이크로프로세서의 구성 요소 및 구조적 특징을 설계의 복잡성 및 성능을 고려해서 결정하고 이 구조에서의 최적화된 이슈 규칙을 결정하였다. 그리고 정해진 마이크로프로세서 구조 및 이슈 규칙에서 최적화된 성능을 갖는 이슈 구조를 설계하였다.

1. 수퍼스칼라 마이크로프로세서

본 논문의 마이크로프로세서는 SPARC version 9 [12]을 명령어 세트(instruction set)로 하는 64 비트 4-way 수퍼스칼라 마이크로프로세서이며, 성능과 하드웨어의 비용을 고려해서 다음과 같이 구성 요소의 종류 및 기능을 결정하였다.

● 프리페치 유닛 - 단일 사이클에 최대 4개의 유효한 명령어를 이슈 유닛에 제공한다. 명령어 캐쉬를 포함하고 있으며 분기 예측 기능을 가지고 있다.

■ 정수 레지스터파일(integer register file) - 2개의 정수 명령어와 1개의 로드/스토어(load/store) 명령어를 동시에 이슈할 수 있도록 7개의 읽기 포트(read port)와 3개의 쓰기 포트(write port)를 가지고 있다.

■ 부동소수점 레지스터파일(floating-point register file) - 5개의 읽기 포트와 3개의 쓰기 포트를 제공함으로써 2개의 부동소수점 명령어와 1개의 부동소수점 로드/스토어 명령어를 동시에 이슈할 수 있도록 하였다.

■ 정수 기능 유닛(integer functional unit) - 2개의 정수 명령어를 동시에 실행할 수 있도록 2개의 ALU을 가지고 있다.

■ 부동소수점 기능 유닛(floating-point functional unit) - 부동소수점 덧셈기, 곱셈기 나눗셈기/제곱근기를 각기 따로 가지고 있으며, 따라서 서로 다른 종류의 부동소수점 명령어를 동시에 실행할 수 있다.

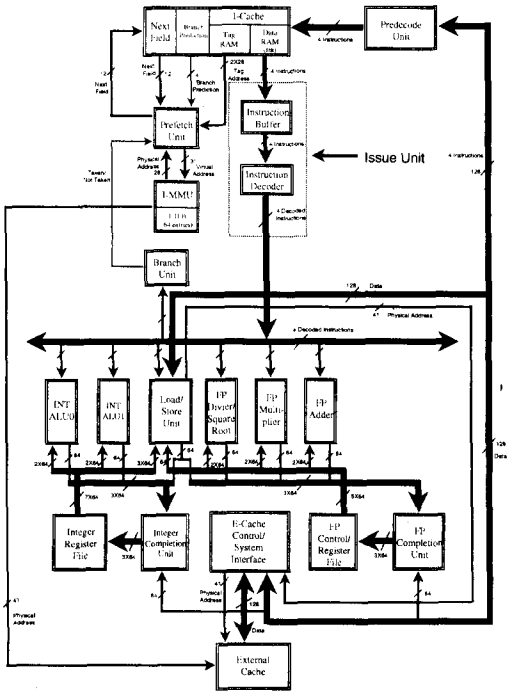

그림 4. 수퍼스칼라 마이크로프로세서의 블럭다이어그램

Fig. 4. Block Diagram of Superscalar Micropro cessor.

■ 로드/스토어 유닛(load/store unit) - 정수 명령어의 실행과 독립적으로 로드/스토어 명령어를 실행할 수 있도록 하기 위해서 이 유닛이 따로 존재하며, 메모리 주소 계산을 위한 덧셈기를 포함한다. 데이터 캐쉬(data cache)에서 태그 미스가 발생하더라도 명령어 이슈를 계속할 수 있도록 하기 위해서 로드 버퍼

(load buffer)와 스토어 버퍼(store buffer)를 가지고 있다.

그리고 신속한 메모리 접근(memory access)을 위해 내부 메모리 관리 유닛인 MMU(Memory Management Unit)를 포함하고 있으며 그림 4와 같은 전체 구조를 가진다.

2. 이슈 규칙

프로세서의 성능은 명령어 이슈율에 의해 크게 좌우되며, 수퍼스칼라 마이크로프로세서는 단일 사이클에 여러 개의 명령어를 동시에 이슈함으로써 성능을 향상시킨다. 그런데 데이터 의존성 때문에 명령어를 이슈할 수 없는 경우가 자주 발생하며, 이러한 경우는 명령어 이슈율 향상의 가장 큰 장애가 된다. 수퍼스칼라 마이크로프로세서와 같이 동시에 여러 개의 명령어를 이슈하는 구조에서는 데이터 의존성으로 인한 이슈 자원 낭비가 더욱 크며 데이터 의존성 검출 및 해결을 위한 로직이 매우 복잡하다. 데이터 의존성이 존재할 때에도 명령어 이슈를 계속하기 위한 비순차적 이슈 방법이 많이 연구되었으나, 이러한 방법들은 설계가 복잡할 뿐 아니라 하드웨어 비용도 크다. 따라서 본 이슈 구조에서는 순차적 이슈 방법을 사용하며, 성능 향상을 위해 최적화된 이슈 규칙을 사용하였다. 이슈 규칙은 다음과 같은 특징을 가지고 있다.

1) 단일 사이클에 최대 4개의 명령어를 이슈할 수 있다.

2) 순차적 방식의 명령어 이슈 방법을 사용한다.

3) 대부분의 명령어는 순차적으로 실행을 완료하지만 지연 시간이 매우 긴 부동소수점 나눗셈/제곱근 명령어와 로드/스토어 명령어는 비순차적 방식으로 실행이 완료된다.

4) R-A-W(read-after-write) 데이터 의존성 - 이슈할 명령어가 아직 결과를 출력하지 못한 이전 명령어의 결과 데이터를 사용할 경우 그 명령어는 이슈되지 못한다.

5) W-A-W(write-after-write) 데이터 의존성 - 명령어 해석기에 있는 명령어를 이슈할 경우 같은 레지스터를 목적 레지스터로 하는 이전의 명령어보다 먼저 레지스터에 데이터를 쓰거나 또는 같은 사이클에 쓰게 될 경우 그 명령어는 이슈되지 못한다.

6) 명령어 정렬 및 병합의 장점을 살리기 위해 분기 명령어 이전의 명령어와 이후의 명령어는 분기 명령어와 같이 이슈될 수 있도록 한다.

7) 로드/스토어 명령어로 인해 데이터 캐쉬에서 태그 미스가 발생하더라도 로드 버퍼 또는 스토어 버퍼에 로드/스토어 명령어를 일시적으로 저장하고 후에 미스가 해결되었을 때 동작을 다시 시작하도록 하여 데이터 캐쉬의 태그 미스로 인해 명령어 이슈와 실행이 중단되지 않도록 한다.

8) 스토어 명령어에 의해 메모리에 쓰여질 데이터가 이전의 명령어에 의해 아직 계산되고 있어도 스토어는 이슈될 수 있도록 하였다. 이것은 스토어 버퍼에 스토어 명령어를 일시적으로 저장하고 후에 데이터가 준비되면 그 때에 메모리에 쓰면 되기 때문이다.

이슈 규칙 중에서 4)와 5)는 순차적 명령어 이슈 방법을 사용하기 때문에 발생하는 이슈 제한이다. 그러나 3), 6), 7) 및 8)과 같이 순차적 명령어 이슈의 단점을 보완하여 성능을 최대한 향상시키도록 이슈 구조를 개선하였다.

Ⅳ. 이슈 유닛의 설계

이슈 유닛은 명령어 버퍼와 명령어 해석기로 구성되어 있다. 프리페치 유닛으로부터 입력된 명령어들은 명령어 버퍼에 순차적으로 저장되며, 명령어 해석기는 명령어 버퍼에 저장된 명령어들을 순차적으로 해석해서 이슈한다.

1. 명령어 버퍼

명령어 버퍼는 프리페치 유닛에서 입력된 유효한 명령어들을 순서대로 정렬하여 내부 저장 소자(storage element)에 저장하고 저장된 명령어들을 순차적으로 명령어 해석기에 제공함으로써 명령어가 이슈되도록 한다. 내부 저장 소자에 저장할 수 있는 명령어의 수는 제한되었기 때문에 내부 저장 소자가 모두 채워진 경우에는 명령어 페치를 중단하며 명령어들이 이슈되어서 빈 자리가 생길 때에 명령어 페치가 다시 시작된다. 그리고 입력되는 명령어 중에는 유효하지 않은 명령어가 있기 때문에 입력되는 명령어의 유효성(validness)을 결정하기 위해서 입력 명령어와 같이 입력되는 유효 비트(valid bit)들을 사용한다. 명령어 버퍼는 2개의 포인터를 이용하여 FIFO 형식의 버퍼로 구현되었는데, 그림 5와 같이 명령어 버퍼에서 입력되는 명령어가 들어갈 위치는 tail이라는 포인터를 사용하여

나타내며, 명령어 해석기로 내보낼 명령어들의 위치는 head라는 포인터를 사용하여 알려준다.

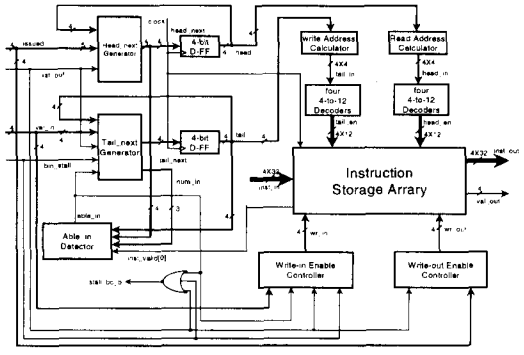

그림 5. 명령어 버퍼의 블럭다이어그램
Fig. 5. Block Diagram of Instruction Buffer.

명령어 버퍼의 크기는 간단하게 정할 수 있는 것이 아니다. 최적의 크기를 결정하기 위해서 명령어 버퍼의 크기를 변화시키면서 모의실험을 수행하여 성능을 비교 평가한 결과 12개의 명령어를 저장할 수 있는 명령어 버퍼가 성능과 하드웨어 비용을 고려해서 최적의 것임을 알 수 있었다. 따라서 본 이슈 유닛은 12개의 명령어를 저장할 수 있는 명령어 버퍼를 사용하였다.

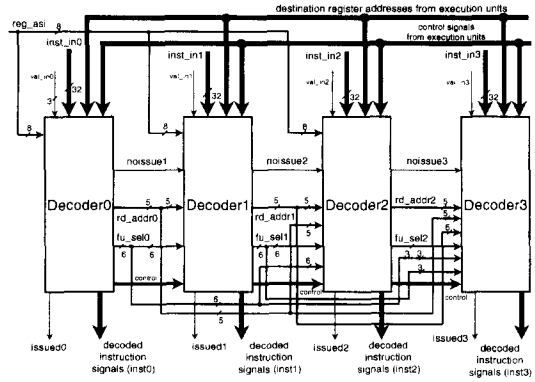

그림 6. 명령어 해석기의 블럭다이어그램
Fig. 6. Block Diagram of Instruction Decoder.

2. 명령어 해석기

명령어 해석기는 명령어 버퍼에서 입력되는 명령어들을 단일 사이클에 최대 4개의 명령어까지 각 기능 유닛에 이슈한다. 명령어 이슈는 III-2에 나온 이슈 규칙에 따라 이루어진다. 그 외에도 인터락(interlock)을 검출하기 위해서 레지스터 주소들을 비교하며, 필요한 경우 파이프라인을 정지(stall)시킨다.

명령어 해석기는 그림 6과 같이 4개의 하위 블럭으로 이루어졌다. 각 블럭은 명령어 버퍼에서 입력되는 4개의 명령어 중에서 할당된 1개의 명령어를 이슈 규칙에 따라 이슈한다. 그리고 이전의 명령어는 다음 명령어의 이슈에 영향을 미치기 때문에 다음 명령어의 이슈를 위해 필요한 제어 신호(control signal)들을 보내준다. 그리고 다음 명령어의 데이터 의존성 검사를 위한 목적 레지스터의 주소와 기능 유닛 충돌의 검출을 위한 기능 유닛의 선택 신호를 다음 명령어를 이슈하는 블럭으로 보내준다. 내부적으로는 입력받은 명령어를 해석하는 부분, 이전의 명령어와의 데이터 의존성을 검사하는 로직, 명령어가 사용할 기능 유닛을 결정하는 부분 및 이전 명령어에 대한 기능 유닛 충돌을 검사하는 로직을 가지고 있다. 그리고 해석 및 검사를 통해 얻은 정보를 이용하여 최종적으로 이슈를 결정한다.

V. 결과 및 고찰

본 논문의 이슈 유닛의 동작을 확인하기 위해서 이슈 유닛을 HDL을 사용하여 기능 수준에서 기술하고 모의실험을 수행하였다. 모의실험은 우선 명령어 버퍼와 명령어 해석기로 나누어 수행한 다음 두 모듈(module)을 통합하여 전체 이슈 유닛을 구성하여 전체적인 동작의 적당성을 확인하였다.

1. 모의실험 환경

입력 벡터를 만들기 위해서 우선 C 프로그램을 gcc [13]로 컴파일(compile)하여 만든 실행 파일(binary file)을 변환하여 16진수(hexadecimal)로 된 텍스트(text) 형식의 프로그램 파일을 만들었다. 성능 평가를 위해서는 널리 사용되는 benchmark 프로그램을 사용하는 것이 가장 좋으나 성능 평가 과정을 간략하게 하기 위해서 많이 쓰이는 C 프로그램들을 사용하였다. 사용한 C 프로그램의 종류 및 총 수행 명령어 수가 표 1에 나와 있다. 이렇게 해서 만들어진 프로그램은 많은 분기 명령어들을 포함하고 있고 분기 명령어로 인해 명령어의 흐름이 바뀐다. 모의실험은 분기 처리 유닛(branch unit) 없이 수행되므로 모의실험을 수행하기 전에 명령어 열의 흐름을 분석할 필요가 있다. 이러한 이유 때문에 hexadecimal 파일로 된 프로그램의 명령어 열 흐름을 분석하기 위해서 검증된 마이

크로프로세서의 HDL 모델을 이용한 모의실험을 수행하였다. 본 논문에서는 SPARC 명령어 세트를 사용하며, 32 비트 스칼라 구조를 갖는SMPC-96[14]의 HDL을 명령어 열 흐름의 분석을 위해서 사용하였다. 이렇게 분석된 명령어 열의 흐름에 근거하여 프로그램 명령어들의 순서를 재조정하여 최종적인 입력 벡터를 만들었다. 입력 벡터 발생 과정을 그림 7에서 요약하였다.

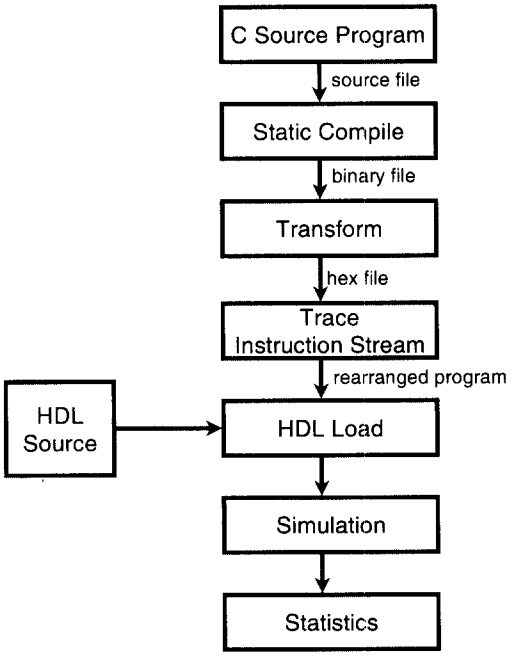

그림 7. 입력 벡터 발생 및 모의실험의 흐름도
Fig. 7. Flow of Input Vector Generation and Simulation.

표 1. 입력 벡터로 사용되는 프로그램들의 크기
Table 1. Sizes of Programs Used as Input Vectors.

프로그램	총 수행 명령어 수
shell_sort_c	120,553
matrix_c	6,099
qsort_c	2,460
sieve.c	114,532
hanoi_tower.c	19,164
merge_sort.c	4,535

광범위한 성능 평가를 효율적으로 하기 위해서는 기능 유닛들의 HDL 모델이 필요하다. 그러나 본 논문의 이슈 유닛에 맞는 기능 유닛들의 HDL 모델이 없기 때문에 의사 기능 유닛 모델(pseudo functional unit model)을 만들었다. 의사 기능 유닛 모델은 정수 연산, 부동소수점 연산 및 로드/스토어 동작을 포함하여 실제로 명령어를 수행하는 기능 유닛의 기능을 최소화시켜 간략하게 만든 것으로서 이슈 유닛과 상호 데이터 및 제어 신호를 교환함으로써 모의실험을 정확하게 한다. 프리페치 유닛의 명령어 캐쉬는 8-Kbyte direct mapped cache를 사용하였다. 그러나 명령어 캐쉬의 크기는 최적의 명령어 버퍼 크기에 영향을 많이 미치기 때문에 8-Kbyte 이하의 크기를 갖는 명령어 캐쉬를 사용한 모의실험을 수행하여 명령어 캐쉬의 크기 감소에 따른 성능 저하를 측정하였다. 성능 평가는 그림 8과 같이 이슈 유닛과 의사 기능 유닛 및 명령어 캐쉬로 이루어진 모델을 사용하여 수행하였다.

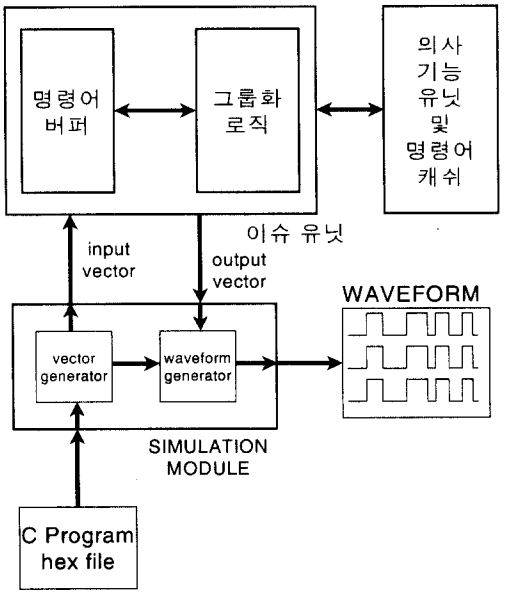

그림 8. 모의실험 환경
Fig. 8. Simulation Environment.

2. 성능 평가

명령어 버퍼의 사용으로 이슈 유닛의 성능이 향상되기는 하지만 버퍼 내에 저장할 수 있는 최대 명령어 수에 따라 그 성능이 달라진다. 명령어 버퍼의 최대

저장 명령어 수(N_Entry)를 변화시키면서 모의실험을 수행한 결과 명령어 이슈율은 표 2와 같이 변화하였다. 그리고 스칼라 구조인 SMPC-96의 명령어 이슈율을 1로 하였을 때 각 프로그램에 대한 명령어 이슈율 향상 정도는 그림 9와 같이 나타난다. 그런데 표 2와 그림 9는 8-Kbyte 명령어 캐쉬를 사용한 결과이다. 명령어 캐쉬의 크기가 명령어 이슈율에 미치는 영향은 매우 크므로 명령어 캐쉬의 크기를 변화시키면서 그 영향을 측정하였다. 명령어 캐쉬의 크기 변화에 따르는 명령어 이슈율의 변화가 표 3에 나와 있으며, 그림 10에 SMPC-96의 이슈율을 1로 하였을 때의 명령어 이슈율 향상이 나와 있다.

표 2. 다양한 명령어 버퍼 크기를 갖는 이슈 유닛들의 성능 비교

Table 2. Performance Comparison of Issue Units with Different Instruction Buffer Sizes.

N_Entry	명령어 이슈율 (CPI)
SMPC-96	0.819
0	1.202
4	1.583
8	1.853
12	1.986
16	2.016

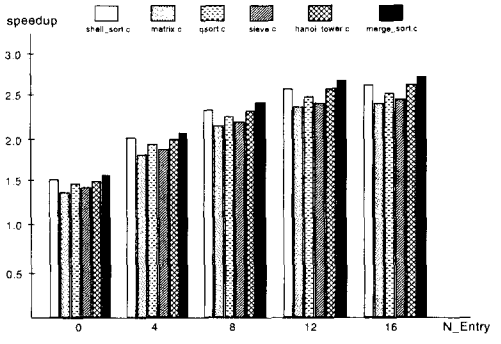

그림 9. 다양한 명령어 버퍼 크기를 갖는 이슈 유닛의 성능 향상 (8-Kbyte 명령어 캐쉬)

Fig. 9. Performance Increases of Issue Units with Different Instruction Buffer Sizes (8-Kbyte I-Cache).

표 3. 다양한 명령어 캐쉬 크기를 갖는 이슈 유닛의 성능 비교

Table 3. Performance Comparison of Issue Units with Different Cache Sizes.

명령어 캐쉬 크기 / 명령어 버퍼 크기	1-Kbyte	2-Kbyte	4-Kbyte	8-Kbyte
8 entries	1.332	1.541	1.701	1.853
12 entries	1.612	1.789	1.863	1.986

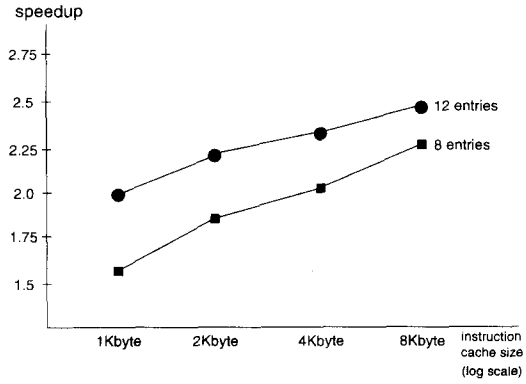

그림 10. 다양한 명령어 캐쉬 크기를 갖는 이슈 유닛의 성능 향상

Fig. 10. Performance Increases of Issue Units with Different Instruction Cache Sizes.

표 2와 그림 9를 보면 알 수 있듯이 수퍼스칼라 구조라고 하더라도 명령어 버퍼를 사용하지 않는 경우에는 성능이 크게 개선되지 않는다. 그러나 명령어 버퍼를 사용하는 경우에는 최대로 저장할 수 있는 명령어 수가 증가함에 따라 성능이 크게 향상된다. 그림 9에서 알 수 있듯이 최대 저장 명령어 수가 12개 이상이 되면 더 이상 성능이 향상되지 않는다. 이것은 명령어를 12개 이상 저장할 수 있는 경우 이슈 유닛에 공급할 명령어들을 충분히 공급할 수 있기 때문이다. 실제로 명령어 캐쉬에서 태그 미스가 발생하는 경우 여러 사이클 동안 명령어를 페치하지 못하며, 이러한 상황에서도 명령어 이슈를 계속하기 위해서는 12개 이상의 명령어를 명령어 버퍼에 저장할 수 있어야 한다. 그런데 이 결과는 명령어 캐쉬의 크기가 8-Kbyte인 경우에 대한 것이다. 명령어 캐쉬의 크기가 더 커 질 때에는 캐쉬 태그 미스가 더욱 줄어들기 때문에 그 경우에도 12-entry 명령어 버퍼가 최적의 성능을 가진다. 명령어 캐쉬의 크기가 8-Kbyte보다 작은 경우에도

12-entry 명령어 버퍼를 사용하면 표 3과 그림 10에서 알 수 있듯이 성능이 크게 낮아지지 않는다. 8-entry 명령어 버퍼를 사용할 경우에는 1-Kbyte 명령어 캐쉬에 대한 성능이 8-Kbyte 명령어 캐쉬에 대한 성능보다 28%나 낮아지는 반면 12-entry 명령어 버퍼를 사용할 경우에는 그 성능이 19%의 성능 하락만을 보인다. 이와 같이 12-entry 명령어 버퍼는 명령어 캐쉬의 크기에 많은 영향을 받지 않으면서 충분한 명령어를 저장한다. 더욱이 요즈음의 고성능 마이크로프로세서는 8-Kbyte 이상의 내부 명령어 캐쉬를 제공한다는 점을 고려할 때 명령어 버퍼가 저장할 수 있는 최대 명령어 수는 12개일 때 최적의 성능을 유지할 수 있음을 알 수 있다.

Ⅳ. 결 론

본 논문에서는 명령어 버퍼를 이용한 4-way 수퍼스칼라 마이크로프로세서의 이슈 유닛을 설계하고 HDL을 이용하여 기능 수준에서 검증 및 성능 평가를 수행하였다. 명령어 버퍼는 페치된 명령어들을 정렬 및 병합하여 이슈할 수 있도록 하여 명령어 이슈율을 향상시켰다. 또한 명령어 페치와 명령어 이슈를 분리하는 역할을 하여 명령어 부족으로 인한 명령어 이슈의 정지를 가능한 한 최대로 줄일 수 있었다. 명령어 이슈 구조는 명령어 버퍼와 명령어 해석기로 이루어졌다. 명령어 버퍼는 프리페치 유닛으로부터 받은 명령어들을 정렬하여 순서대로 저장하고, 명령어 해석기는 명령어 버퍼에 저장된 명령어들을 해석하여 순서대로 각 기능 유닛들에 이슈한다. 그리고 이슈 유닛은 순차적 명령어 이슈의 단점을 보안하기 위해서 명령어 의존성이 없는 경우에는 명령어들을 계속 이슈할 수 있도록 하여 성능을 향상시켰다.

이슈 유닛의 성능 평가 결과 명령어 버퍼를 사용하지 않을 경우에는 스칼라 구조에 비해 성능이 크게 개선되지 않음을 알 수 있었다. 그러나 명령어 버퍼를 사용함으로써 명령어 이슈율이 크게 향상되었다. 8-Kbyte 명령어 캐쉬를 사용할 경우 12-entry 명령어 버퍼가 가장 적당하다는 것을 알 수 있었으며, 이 경우에 스칼라 구조의 32 비트 마이크로프로세서인 SMPC-96에 대해서 2.5배의 성능 향상을 보였다. 8-Kbyte 이하의 명령어 캐쉬를 사용할 경우에도 12-entry 명령어 버퍼를 사용하면 성능 하락이 작으며, 수퍼스칼라 구조의 고성능 마이크로프로세서들은 8-Kbyte 이상의 내부 명령어 캐쉬를 사용한다는 것을 고려하면 12-entry 명령어 버퍼를 이용한 이슈 유닛이 가장 적당하다.

감사의 글

※ 본 연구는 서울대학교 반도체공동연구소의 교육부 반도체분야 학술연구조성비(과제번호 : ISRC 96-E-2017)에 의해 수행되었습니다.

참 고 문 헌

[1] Kai Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, pp. 308-322, 1993

[2] Mike Johnson, *Superscalar Microprocessor Design*, Prentice-Hall, pp. 9-175, 1991

[3] Michael J. Flynn, *Computer Architecture: Pipelined And Parallel Processor Design*, Jones and Bartlett Publishers, pp. 456-498, 1995

[4] Robert A. Iannucci, Guang R. Gao, Robert H. Halstead, Jr., Burton Smith, *Multithreaded Computer Architecture: A Summary of The State of The Art*, Kluwer Academic Publishers, pp. 1-60, 1994

[5] Ben Catanzaro, *Multiprocessor System Architectures*, Sun Microsystems, Inc., pp. 4-46, 1994

[6] Weiss and James E. Smith, "Instruction Issue Logic in Pipelined Supercomputers", *IEEE Transactions on Computers*, vol. c-33, pp. 1013-1022, Nov. 1984

[7] Gurindar S. Sohi, "Instruction Issue Logic for High-Performance, Interruptible, Multiple Functional Unit, Pipelined Computers", *IEEE Transactions on Computers*, vol. 39, pp. 349-359, Mar. 1990

[8] Garold S. Tjaden and Michael J. Flynn, "Detection And Parallel Execution of Independent Instructions", *IEEE Tran-*

sactions on Computers, vol. c-19, pp. 889-895, Oct. 1970

[9] Ramon D. Acosta, Jacob Kjelstrup, and H. C. Torng, "An Instruction Issuing Approach to Enhancing Performance in Multiple Functional Unit Processors", *IEEE Transactions on Computers*, vol. c-35, pp. 815-828, Sep. 1986

[10] Manolis G. H. Katevenis, *Reduced Instruction Set Computer Architectures for VLSI*, The MIT Press, pp. 42-85, 1985

[11] Lee, J. K. F., and A. J. Smith. "Branch Prediction Strategies and Branch Target Buffer Design.", *IEEE Computer*, vol. 17, pp. 6-22, Jan. 1984

[12] David L. Weaver and Tom Germond, *The SPARC Architecture Manual*, Prentice-Hall, Inc., version 9, 1994

[13] Sun Microsystems, Inc., *C-compiler(Sun Release 4.1) on-line manual*, 1988

[14] 안종근, *내부코드를 이용한 32 비트 SPARC RISC 정수연산유닛 제어부의 VLSI 설계 및 검증*, 연세대학교 대학원 전자공학과 석사학위 논문, 1996년 8월.

저 자 소 개

文 秉 仁(正會員)

1972년 1월 24일생. 1995년 2월 연세대학교 전자공학과 졸업(공학사). 1997년 2월 연세대학교 전자공학과 석사 과정 졸업(공학석사). 1997년 3월 현재 연세대학교 전자공학과 박사과정. 주관심 분야는 마이크로프로세서 및 DSP 설계임

安 相 俊(正會員)

1969년 9월 5일생. 1993년 8월 연세대학교 전자공학과 졸업(공학사). 1995년 8월 연세대학교 전자공학과 석사 과정 졸업(공학석사). 1995년 9월 현재 연세대학교 전자공학과 박사과정. 주관심 분야는 마이크로프로세서 및 DSP 설계임

李 龍 煥(正會員)

1970년 1월 29일생. 1993년 2월 연세대학교 전자공학과 졸업(공학사). 1995년 2월 연세대학교 전자공학과 석사 과정 졸업(공학석사). 1995년 3월 현재 연세대학교 전자공학과 박사과정. 주관심 분야는 마이크로프로세서 및 VLSI 설계, CAD 등임

李 溶 錫(正會員)

1950년 10월 23일생. 1973년 2월 연세대학교 전기공학과 졸업(공학사). 1977년 2월 University of Michigan, An Arbor 전자공학과 석사 과정 졸업(공학석사). 1981년 2월 University of Michigan, Ann Arbor 전자공학과 박사 과정 졸업(공학박사). 현재 연세대학교 전자공학과 교수. 주관심 분야는 마이크로프로세서, SRAM, 캐쉬 및 DSP 설계임