pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2013.2.3.111

Soft Error Detection for VLIW Architectures with a Variable Length Execution Set

Jongwon Lee[†] · Doosan Cho[††] · Yunheung Paek[†††]

ABSTRACT

With technology scaling, soft error rate has greatly increased in embedded systems. Due to high performance and low power consumption, VLIW (Very Long Instruction Word) architectures have been widely used in embedded systems and thus many researches have been studied to improve the reliability of a system by duplicating instructions in VLIW architectures. However, existing studies have ignored the feature, called VLES (Variable Length Execution Set), which is adopted in most modern VLIW architectures to reduce code size. In this paper, we propose how to support instruction duplication in VLIW architecture with VLES. Our experimental results demonstrate that a VLIW architecture with VLES shows 64% code size decrement on average at the cost of about 4% additional cell area as compared to the case of a VLIW architecture without VLES when instruction duplication is applied to both architectures. Also, it is shown that the case with VLES does not cause extra execution time compared to the case without VLES.

Keywords : Soft Error, VLIW Architecture, Instruction Duplication, Reliability, VLES, Code Size

Variable Length Execution Set을 지원하는 VLIW 아키텍처를 위한 소프트 에러 검출 기법

이 종 원[†] · 조 두 산[††] · 백 윤 흥[†††]

요 약

공정 기술의 발전으로 인해 내장형 시스템에서 소프트 에러 발생 비율이 크게 증가하고 있다. 고성능, 저전력을 특징으로 하는 VLIW 아키텍처가 내장형 시스템에 널리 사용되어 왔는데, 이러한 VLIW 아키텍처에서 명령어 복제를 통해 소프트 에러를 감지하여 신뢰도를 높이고자 하는 연구가 진행되어 왔다. 하지만 기존 연구는 대부분의 상용 VLIW 아키텍처가 코드 크기 감소를 위해 사용하는 VLES 를 고려하지 않고 이루어졌다. 명령어 복제를 통한 신뢰도 향상을 위한 연구가 실용성 및 적용성을 갖추기 위해서는 VLES 를 지원하는 VLIW 아키텍처에 대해 이루어져야 한다. 이에 본 논문에서는 VLES 를 지원하는 VLIW 아키텍처에서 명령어 복제를 위해 필요한 설계 방법을 논하고 이에 따른 실험 결과를 제시하였다. 실험 결과 VLES 를 지원하지 않을 경우에 비해 약 4% 정도의 추가적인 하드웨어 비용을 들여 평균 64% 정도에 달하는 코드 크기 감소 효과를 얻을 수 있었고, 또한 실행 시간에는 추가적인 손실이 발생하지 않음을 알 수 있었다.

키워드 : 소프트 에러, VLIW 아키텍처, 명령어 복제, 신뢰도, VLES, 코드 크기

1. 서 론

최근 내장형 시스템 설계에 있어서 고성능, 저전력, 저면적 등에 대한 요구뿐만 아니라 신뢰도에 대한 요구 또한 갈수록 높아지고 있다. 이는 공정기술의 발전에 따른 저전압 공급, 칩사이즈 축소, 낮은 노이즈 마진 등으로 인해 소프트 에러의 발생률이 증가하기 때문이다. 소프트 에러란 시스템 내에서의 영구적인 결함을 의미하는 하드 에러에 반대되는 개념으로써 주로 알파 입자나 중성자와 같은 시스템 외부의 에너지 입자들에 의한 일시적인 결함을 의미한다. 이러한 일시적인 결함은 프로그램의 실행 과정의 무결성을 보장할

※ 본 연구는 교육과학기술부/한국과학재단 우수연구센터 육성사업(No. 2012-0000470), 2012년도 정부(교육과학기술부)의 재원으로 한국과학재단의 국가지정연구실사업(No. 0421-2012-0047), (재)스마트 IT 융합 시스템 연구단(글로벌프론티어사업) ((재)스마트 IT 융합 시스템 연구단-0543-20110012), 한국연구재단의 기초연구사업(No.2010-0024529) 및 IDEC의 지원을 받아 수행된 것임.

† 준 회 원 : 서울대학교 전기정보공학부 박사과정
†† 정 회 원 : 순천대학교 전자공학과 조교수
††† 종신회원 : 서울대학교 전기정보공학부 교수

논문접수 : 2012년 12월 11일
수 정 일 : 1차 2013년 1월 29일
심사완료 : 2013년 1월 30일

* Corresponding Author : Doosan Cho(dscho@sunchon.ac.kr)

수 없도록 하여 궁극적으로 프로그램의 실행 결과를 더 이상 신뢰할 수 없게 만든다. 특히 의료, 금융, 보안 분야와 같이 높은 신뢰도가 요구되는 내장형 시스템일 수록 이러한 소프트 에러를 감지하는 기술이 절대적으로 필요하다.

VLIW (Very Long Instruction Word) 아키텍처는 프로세서 내의 명령어 실행 유닛의 수를 늘려서 병렬적으로 실행 가능한 명령어들을 동시에 수행시켜 성능 향상을 꾀하고자 도입된 아키텍처이다[1]. 하드웨어적으로 병렬 수행 가능 명령어를 추출해내는 수퍼스칼라 (superscalar) 와는 달리 컴파일러의 코드 생성에 전적으로 의존하기 때문에 VLIW 아키텍처는 상대적으로 낮은 하드웨어 복잡도를 보이고 이에 따라 전력 소모를 줄일 수 있기 때문에 그 적합성을 인정받아 내장형 시스템에 널리 사용되고 있다[2]. 최근 내장형 시스템의 높아지는 신뢰도 요구에 맞춰 VLIW 아키텍처 에서의 신뢰도 향상을 위한 연구가 이루어져 왔다. 주목할 만한 연구로 컴파일 시간에 명령어 복제를 통해 소프트 에러를 검출하는 방식의 연구가 있다[3]. 이 방식은 복제 명령어의 실행 결과를 원본 명령어의 결과와 비교하여 그 일치 여부에 따라 소프트 에러의 발생을 탐지하는 것이다. 이런 경우 복제 명령어로 인한 코드 사이즈 증가가 부담이 될 수 있다. 하지만 VLIW 코드는 프로그램에 내재하는 병렬성이 적을 경우, 내부에 NOP (No Operation) 명령어를 채우는데, 이들 NOP 을 복제 명령어로 대체하여 복제 명령어로 인한 코드 사이즈 증가를 줄이면서 동시에 신뢰도를 높이게 된다. 즉, VLIW 코드 내부에 NOP 이 많이 나타날 수록 복제 명령어의 오버헤드를 줄인 채 소프트 에러를 검출할 수 있다.

하지만 오늘날 대부분의 VLIW 아키텍처는 애초에 코드 내부에 NOP 이 존재하지 않는 구조를 채택하고 있다. VLIW 코드 내부에 존재하는 NOP 으로 인해 코드 사이즈가 커지게 되고 따라서 큰 프로그램 메모리를 요구하게 되어 내장형 시스템의 면적이 증가할 수 밖에 없는 기존 VLIW 아키텍처가 풀어야 할 문제를 VLES (Variable Length Execution Set) 도입으로 해결했기 때문이다. Fig. 1 은 VLES 의 유무에 따른 VLIW 아키텍처의 동작의 차이를 보여주고 있다. 예시에 사용된 VLIW 아키텍처는 2개의 투입 슬롯 (2-way issue slots) 을 갖는데, 이는 동시에 최대 2개의 명령어가 실행될 수 있음을 의미한다. 프로그램 메모리로부터 매 시간마다 투입 슬롯 개수만큼의 명령어가 읽혀지는데 이를 인출 패킷 (fetch packet) 이라 하고, 이로부터 실행 유닛들로 보내지는 명령어 조합을 실행 패킷 (execute packet) 이라고 부른다. VLES 의 유무에 따른 프로그램 메모리의 명령어 배치 상태를 살펴 보면, VLES 를 지원하는 경우에 NOP 이 존재하지 않음을 알 수 있다. 따라서 인출 패킷은 두 경우에 있어서 서로 다르게 된다. 하지만 두 경우 모두 실행 패킷은 매 시간 마다 동일함을 확인할 수 있다. Fig. 1(b)에서 볼 수 있듯이 시간 t+1 에서의 인출 패킷

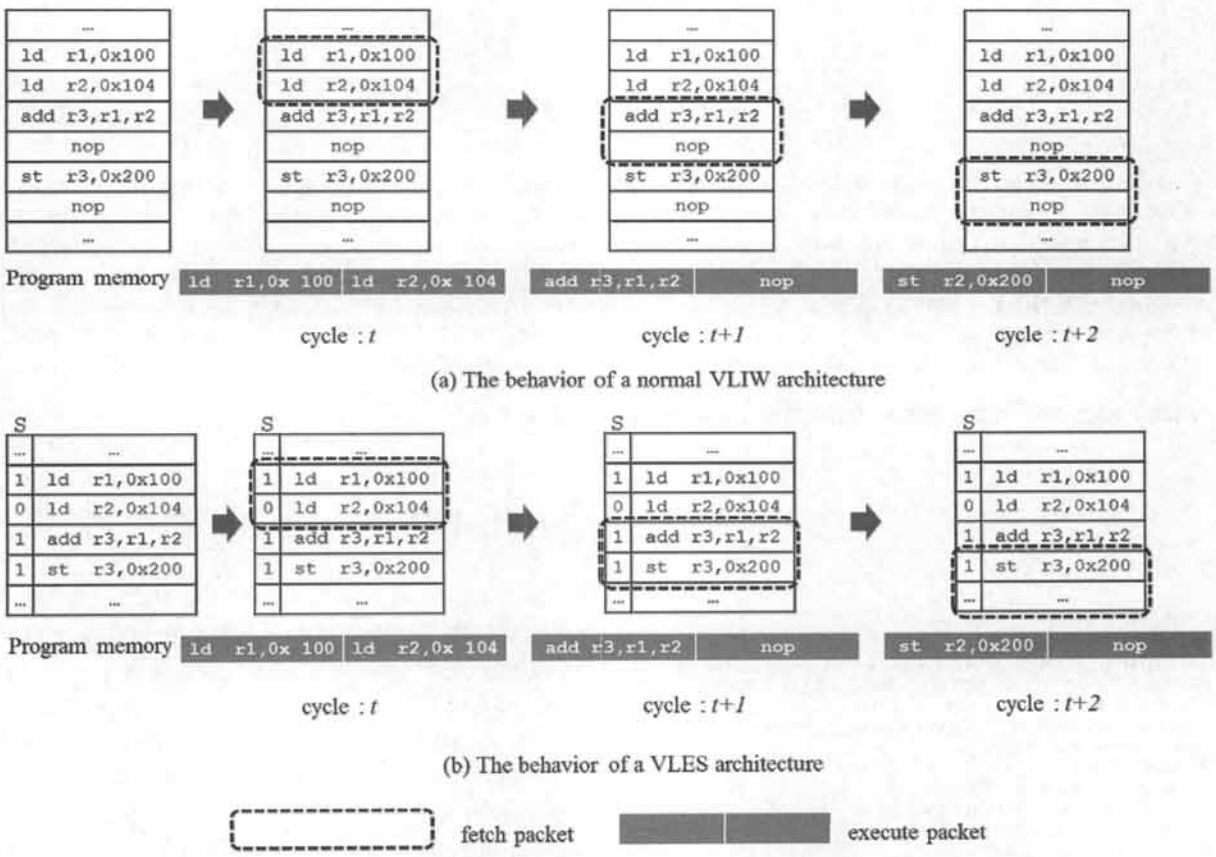

Fig. 1. The differences of behaviors of VLIW architectures according to the existence of VLES

은 add r3,r1,r2 와 st r3, 0x200 으로 이루어져 있으나 두 번째 명령어는 실행 패킷에 나타나지 않게 되는데 이는 두 명령어 사이에 r3 를 통한 자료 의존성 (data dependence) 이 존재하기 때문이다. 다시 말해서, VLES 를 지원하는 VLIW 아키텍처의 경우 VLIW 코드에 NOP 의 도움 없이도 자료 의존성을 해치지 않게끔 실행 패킷을 생성한다.

이와 같이 명령어 복제를 통해 소프트 에러를 탐지하여 신뢰도를 높이고자 하는 기존의 연구는 이러한 VLES 를 지원하지 않는 VLIW 아키텍처에 대해서만 이루어졌다는 점에서 그 실용성과 적용성에 한계를 지니고 있다고 할 수 있다. 따라서 본 논문에서는 VLES 를 지원함과 동시에 명령어 복제를 통해 소프트 에러를 검출할 수 있는 새로운 VLIW 아키텍처를 제안하도록 한다. 2장에서 제안하는 VLIW 아키텍처에 대해 설명하고, 3장에서 실험 환경 및 실험 결과를 제시하고, 4장에서 끝맺도록 하겠다.

2. 제안하는 VLIW 아키텍처

2.1 소프트 에러 감지를 위한 하드웨어 추가

우리는 4개의 투입 슬롯 (4-way issue slots) 을 갖는 VLIW 아키텍처를 사용하였다. 실행 유닛의 구성은 두 개의 Integer ALU, 하나의 Integer Multiplier, 하나의 Load/Store Unit 및 하나의 Branch Unit 으로 이루어져 있고, 16개의 범용 레지스터 (general purpose register) 를 사용하며 5개의 파이프라인 구조를 갖는다 (IF, ID, EX, MEM, WB). 이런 기본적인 구조를 지닌 VLIW 아키텍처에서 출발하여 VLES 및 명령어 복제를 위한 기능을 추가하였다.

Fig. 2는 제안하는 VLIW 아키텍처를 나타낸다. 빗금 친 부분은 새로 추가된 하드웨어 유닛들을 의미한다. 큐 (Queue) 는 각각의 투입 슬롯 마다 존재하여 실행 유닛의 결과값을 저장하는 역할을 한다. 큐0 (Queue 0) 부터 큐 2 (Queue 2) 까지는 원본 명령어의ALU 연산 및 Multiplier 연산 결과값을 저장하고, 큐3 (Queue 3) 은 원본 명령어의 메모리 주소값을 저장한다. 이렇게 저장된 원본 명령어의 결과값들은 이후에 나타나는 복사본 명령어의 결과값과 비교기 (Comparator) 를 통해 비교되어 소프트 에러 발생 여부를 알아낼 수 있다. 원본 명령어와 복사본 명령어의 결과값이 일치하면 비로소 결과값이 최종적으로 레지스터에 쓰이거나 메모리 연산이 수행되고, 결과값이 다른 경우 소프트 에러가 발생한 것으로 간주하여 시스템에게 알린다.

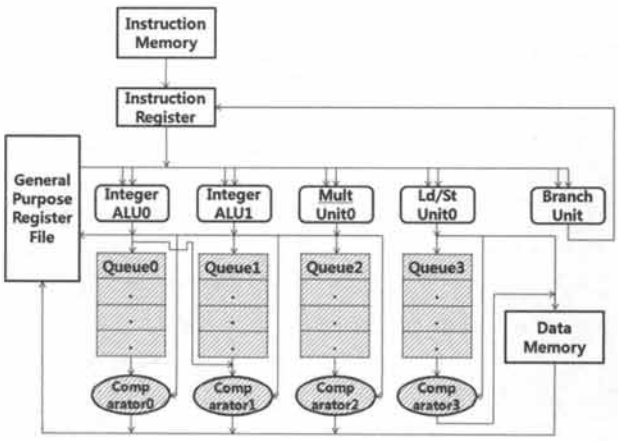

Fig. 2. The proposed VLIW architecture

2.2 명령어 인코딩 방식 추가

이와 같이 원본 명령어와 복사본 명령어의 실행 중의 동작이 서로 다름을 확인할 수 있다. 따라서 하드웨어는 원본 명령어와 복사본 명령어를 실행 중에 구분할 수 있어야 한다. 또한 우리의 아키텍처는 VLES 를 지원할 수 있어야 하기 때문에 하드웨어는 인출 패킷으로부터 실행 패킷을 추출할 수 있어야 한다. 즉, 우리는 VLES 관련 정보와 원본 및 복사본 명령어를 구분할 수 있는 정보를 하드웨어에 제공해야 한다. 이를 위해 명령어의 인코딩 공간 (encoding space) 의 일부를 할당하였다.

우리가 사용하는 VLIW 아키텍처는 각각의 명령어가 32 비트의 인코딩 공간을 사용하는 구조이다. 명령어 집합은 기본적으로 RISC (Reduced Instruction Set Computer) 명령어 집합을 따르고 있기 때문에 추가적인 정보를 인코딩할 수 있는 여지가 충분하다[4]. 우리는 VLES 정보를 위해 1 비트, 그리고 원본 및 복사본 명령어의 구분을 위해 2비트를 사용해서 총 3 비트를 32 비트 명령어 인코딩 공간에 할당하였다.

VLES 를 위한 1 비트는 정지 비트 (stop bit) 로 사용된다. VLES 를 지원하는 VLIW 아키텍처는 매 시간 마다 읽어 들이는 인출 패킷에서 이를 이루는 명령어들의 정지 비트를 순차적으로 조사하여 실행 패킷을 추출해낸다. 정지 비트가 0 인 경우는 병렬적으로 실행이 가능한 명령어이고, 정지 비트가 1인 경우는 병렬적으로 실행이 불가능한 명령어를 의미한다. (단, 인출 패킷의 첫번째 명령어의 정지비트는 항상 1이고 무조건 실행 패킷에 포함시킨다.) 따라서 읽어 들인 인출 패킷에서 정지 비트가 1이 나타날 때까지 그 이전 명령어들을 실행 패킷으로 만들어 실행 유닛으로 보내게 된다. 예를 들어, Fig. 1(b)의 시간 t 에서의 인출 패킷의 두 번째 명령어 (ld r2, 0x104) 의 정지 비트는 0 이기 때문에 첫 번째 명령어 (ld r1, 0x100) 와 함께 실행 패킷을 이루게 된다. 하지만 시간 t+1 에서의 인출 패킷의 두 번째 명령어 (st r3, 0x200) 의 정지 비트는 1 이기 때문에 첫 번째 명령어 (add r3,r1,r2) 와 함께 실행 패킷을 이루지 못하고 다음 시간의 실행 패킷에 포함된다.

다음으로 원본 명령어와 복사본 명령어의 구분에 사용되는 2 비트에 대해 살펴보자. Table 1은 이들 비트에 대한 간단한 설명을 나타낸다. 두 개의 비트는 각각 B1 과 B2 비트로 표기되는데, B1 비트는 명령어가 원본인지 복사본인지 구분하는 역할을 하고, B2 비트는 큐를 사용하는지 여부를 나타낸다. 구체적으로 설명하면, B2 비트가 0 인 경우는 원본과 복사본 명령어 모두 큐를 사용하지 않는다. 원본 명령어와 해당 명령어의 복사본이 같은 시간에 수행될 때, B2

비트가 0 인 경우에 해당한다. 이 때는 두 연산이 동시에 수행되기 때문에 별도로 큐에 저장할 필요 없이 연산 결과를 곧장 비교기에 보내서 결과를 비교한다. 하지만 원본 명령어와 복사본 명령어가 서로 다른 시간에 수행되는 경우는 먼저 수행되는 원본 명령어의 결과가 큐에 저장되고, 이후에 해당 명령어의 복사본이 수행될 때 그 결과를 큐에 저장된 원본 결과물과 비교한다. 이런 경우에는 B2 비트가 1 로 설정된다.

Table 1. The meaning of two bits for distinguishing original and duplicate instructions

B1	*B2*	*의미*
0	0	복사본, 큐에 값을 쓰지 않음
0	1	복사본, 큐에 저장된 원본 결과물과 비교하여 일치하면 결과값을 레지스터에 저장 혹은 메모리 참조, 일치하지 않으면 소프트 에러 발생 알림
1	0	원본, 큐에 값을 쓰지 않음
1	1	원본, 연산 결과물을 큐에 저장

지금까지 설명한 내용의 이해를 돕기 위한 예제가 Fig. 3 에 나타나 있다. Fig. 3(a)는 입력으로 사용되는 코드의 프로그램 메모리 내에서의 배치 모습을 나타낸다. 32 비트 명령어 인코딩 공간 중 앞에서부터의 세 비트를 차례대로 정지 비트 (S), B1 비트 (B1), B2 비트 (B2) 의 용도로 할당하였다. 편의상 복사본 명령어는 회색으로 나타내었다. 회색으로 표시된 명령어는 복사본이므로 모두 B1 비트값이 0 임을 알 수 있고, 반대로 흰색은 모두 원본이므로 B1 비트값이 1 임을 확인할 수 있다. 우리가 제안하는 아키텍처는 VLES 를 지원하므로 코드 내에 NOP 이 나타나지 않음을 확인할 수 있다. Fig. 3(b)는 입력 코드의 수행 중의 모습을 나타낸다. 시간의 흐름에 따라 각각의 명령어의 투입 슬롯 배치 상태를 확인할 수 있다. 실행 중의 상태를 보면 빈 슬롯이 나타나는 것을 확인할 수 있는데 이는 VLES 를 지원하지 않을 경우 모두 NOP 으로 코드에 나타나야 하는 부분들이다. (Fig. 3(c)). 우리는 VLES 를 지원하지 않기 때문에 이들 NOP 을 코드에 나타낼 필요가 없다. 매 시간 마다 가장 좌측에 나타나는 명령어의 정지 비트가 1임을 확인할 수 있는데, 이 정지 비트를 통해 실행 패킷이 각각의 시간에 맞게 생성되어 투입 슬롯에 배치됨을 알 수 있다. 또한 시간 0 과 시간 2 의 경우 모두 Add 연산의 원본과 복사본이 같은 시간에 수행됨을 볼 수 있다. 따라서 이들의 B2 비트가 모두 0 인 것을 알 수 있다. 반대로 Mul, Load, Store 명령의 경우는 실행 가능한 투입 슬롯이 한 개뿐이므로 원본과 복사본이 각각 다른 시간에 수행되어야 한다. 따라서 이들 명령어의 B2 비트는 모두 1임을 알 수 있다.

3. 실험 결과

본 논문에서 새롭게 제안한 VLIW 아키텍처를 구현하기 위해서 Synopsys사의 Processor Designer 를 사용하였다 [5]. Synopsys 에서 제공하는 LISA 2.0언어를 사용하여 아키텍처를 구성하는 모듈들과 이들의 동작을 기술하여 어셈블러, 링커, 시뮬레이터를 생성하여 코드 사이즈 및 실행 시간을 측정하였다. 또한 Processor Designer 는 HDL (Hardware Description Language) 코드를 생성해주는데, 이를 Design Compiler 에 입력으로 제공하여 기술된 아키텍처의 하드웨어 면적을 측정하였다. 우리는 제안된 VLIW 아키텍처와 기존의 VLES 를 지원하지 않는 VLIW 아키텍처의 연구를 비교하였다[3]. 실험에는 DSPstone 벤치마크의 코드를 사용하였다[6].

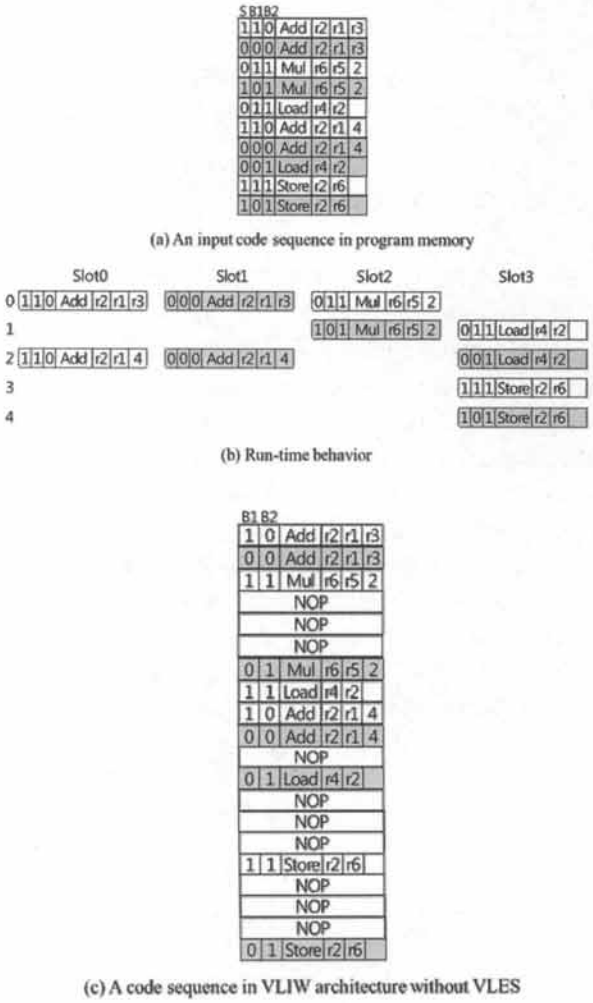

Fig. 3. An example code for the behavior of the proposed VLIW architecture

3.1 코드 크기

Fig. 4는 VLES 유무에 따른 두 가지 VLIW 아키텍처에 대해 복제 가능한 명령어들을 모두 복제한 경우의 코드 크기 측정 결과를 나타낸다. 그림에서 볼 수 있듯이 VLES 를 지원하는 VLIW 아키텍처의 경우에 더 작은 코드 크기를 보임을 확인할 수 있다. VLES 를 지원하지 않는 경우의 VLIW 아키텍처는 NOP 을 복제 명령어로 대체하더라도 여

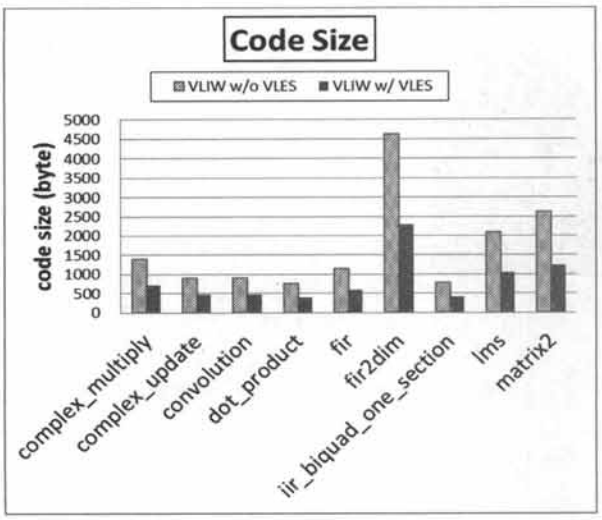

Fig. 4. Code size

전히 상당 수의 NOP 을 코드에 포함시키는데 반해, VLES 를 지원하는 VLIW 아키텍처는 NOP 이 전혀 나타나지 않기 때문에 이로 인한 코드 크기 감소 효과를 보이게 된다. 실험에서 얻은 코드 크기 감소 정도는 평균 64% 에 달했다.

3.2 실행 시간

실행 시간은 두 가지 VLIW 아키텍처에 대해서 벤치마크별로 동일한 결과를 보였다. 이러한 결과는 VLES 유무는 복제 명령어로 인한 추가적인 수행 시간의 증가에 영향을 미치지 않는다는 것을 의미한다. 실행 측면에서 복제 명령어는 프로그램 내에 존재하는 병렬성의 부족으로 인해 발생하는 유휴 실행 유닛을 활용하게 되는데 이는 VLES 유무와는 관계가 없기 때문이다. 즉, 우리는 VLES 를 지원함으로써 명시적인 NOP 을 코드에서 제거할 수는 있지만 유휴 실행 유닛의 발생까지는 막을 수는 없는 것이다. 따라서 복제 명령어가 활용할 수 있는 유휴 실행 유닛은 VLES 와 독립적이기 때문에 두 가지 VLIW 아키텍처에 대해서 동일한 실행 시간 결과가 나타나는 것이다.

3.3 하드웨어 면적

Table 2는 VLES 유무에 따른 두 가지 VLIW 아키텍처의 하드웨어 셀 면적 (cell area) 측정 결과를 나타낸다. VLES 를 지원하는 경우에 이를 위한 추가적인 회로 및 모듈들이 필요하기 때문에 이들로 인한 셀 면적의 증가가 나타남을 볼 수 있다. 하지만 그 정도는 미약하여 약 4% 정도에 그침을 확인할 수 있다.

Table 2. Cell area of two VLIW architectures (unit : μm^2)

	VLIW w/o VLES	VLIW w/ VLES
Combinational area	426635.84	450145.96
Non-combinational area	136997.53	137324.94
Total cell area	563633.37	587470.90

4. 결 론

내장형 시스템에 사용되는 VLIW 아키텍처의 신뢰도를 높이기 위한 연구는 그 동안 일반적인 VLIW 아키텍처에서 지원하는 VLES 를 고려하지 않은 채 이루어 졌다. 이에 본 논문에서는 이러한 방면의 연구의 실용성과 적용성을 높이기 위해 VLES 를 고려한 VLIW 아키텍처에서 명령어 복제를 통한 신뢰도 향상을 위해 필요한 아키텍처 설계 방법을 논하고 이에 따른 실험 결과를 제시하였다. 실험 결과 VLES 를 지원하지 않을 경우에 비해 약 4% 정도의 추가적인 하드웨어 비용을 들여 평균 64% 정도에 달하는 코드 크기 감소 효과를 얻을 수 있었고, 또한 실행 시간에는 추가적인 손실이 발생하지 않음을 알 수 있었다.

참 고 문 헌

[1] Rau, B.R., Fisher, J.A.: Instruction-level parallel processing: History, overview, and perspective. Journal of Supercomputing 7(1-2), 9 - 50 (1993).

[2] ZHONG, H., FAN, K., MAHLKE, S., AND SCHLANSKER, M. 2005. A distributed control path architecture for vliw processors. In Parallel Architectures and Compilation Techniques, 2005. PACT 2005. 14th International Conference on. IEEE, pp.197-206.

[3] HU, J., LI, F., DEGALAHAL, V., KANDEMIR, M., VIJAYKRISHNAN, N., AND IRWIN, M. 2009. Compiler-assisted soft error detection under performance and energy constraints in embedded systems. ACM Transactions on Embedded Computing Systems (TECS) 8, 4, 27.

[4] LEE, J., YOUN, J., LEE, J., AHN, M., AND PAEK, Y. 2012. Dynamic operands insertion for VLIW architecture with a reduced bit-width instruction set. In Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International. pp.119-130.

[5] Synopsys inc., http://www.synonpsys.com

[6] ZIVOJNOVIC, V., VELARDE, J., SCHLAGER, C., AND MEYR, H. 1994. DSPstone: A DSP-oriented benchmarking methodology. In Proc. of the Intern. Conf. on Signal Processing and Technology.

이 종 원

e-mail : jwlee@sor.snu.ac.kr

2007년 서울대학교 전기공학부(학사)

2007년~현 재 서울대학교 전기정보공학부 박사과정

관심분야 : 컴파일러, 임베디드 시스템, VLIW, MPSoC, CGRA, soft error

조 두 산

e-mail : dscho@sunchon.ac.kr
2001년 한국외국어대학교 전자제어공학과 (학사)
2003년 고려대학교 전기전자공학과(석사)
2009년 서울대학교 전기컴퓨터공학부 (박사)
2008년~2010년 ㈜ 리코어스 이사
2010년~현 재 순천대학교 전자공학과 조교수
관심분야 : 임베디드 시스템 설계/최적화, 메모리 시스템 최적화, 컴파일러, 병렬분산처리

백 윤 흥

e-mail : ypaek@snu.ac.kr
1988년 서울대학교 컴퓨터공학과(학사)
1990년 서울대학교 컴퓨터공학과(석사)
1997년 UIUC 전산과학(박사)
1997년~1999년 NJIT 조교수
1999년~2003년 KAIST 전자전산학교 부교수
2003년~현 재 서울대학교 전기정보공학부 교수
관심분야 : 임베디드 소프트웨어, 컴파일러, MPSoC, CGRA