논문 00-25-12B-1 한국통신학회논문지 '00-12 Vol.25 No.12B

빠른 명령어 처리가 가능한 EIS 프로세서 구조

정회원 지 승 현*, 전 중 남**, 김 석 일**

EIS Processor Architecture for Enhanced Instruction Processing

Sung-Hyun Jee*, Joong-Nam Jeon**, Suk-Il Kim** *Regular Members*

요 약

본 논문에서는 실행 시에 긴명령어를 구성하는 각 단위 명령어를 독립적으로 스케줄링할 수 있는 EIS 프로세서 구조를 제안하였다. 단위 명령어별 독립적인 수행을 위해서, EIS 프로세서 구조는 여러 개의 연산처리기와 스케줄러의 쌍으로 구성된다. EIS 프로세서 구조내의 모든 스케줄러는 독립적으로 자료종속성이나 자원충돌 여부를 검사하여 단위 명령어를 실행할 지 혹은 다음 파이프라인 사이클동안 실행을 지연시킬지를 결정한다. 또한 EIS 프로세서용 목적코드는 단위 명령어들간 동기화를 위해서 모든 단위 명령어에 종속성정보를 삽입하는 특징을 지닌다. 즉, EIS 프로세서 구조는 긴명령어내의 각 단위 명령어를 독립적으로 실행시킬 수 있으므로 기존의 VLIW 프로세서 구조나 SVLIW 프로세서 구조에서의 실행지연 시간을 제거할 수 있다. 시뮬레이션을 통해서도 EIS 프로세서 구조의 실행사이클이 VLIW 프로세서 구조나 SVLIW 프로세서 구조에서의 경우보다 더 빠름을 입증할 수 있었다. 특히 실수 명령어 분포가 높은 프로그램에서 EIS 프로세서서에서의 실행사이클이 다른 프로세서 구조의 경우에 비하여 현저하게 줄어드는 것을 확인할 수 있었다.

ABSTRACT

This paper proposes the *EIS (Enhanced Individual Scheduling)* processor architecture, which can individually schedule each unit instruction composing a long instruction word at run time. To process that, the EIS processor architecture has a number of functional unit - individual scheduler unit pairs. Every scheduler of the EIS processor individually determines whether to issue a unit instruction or to stall the functional unit due to resource collision or data dependencies. The EIS processor architecture also has an object code which includes dependency information for every unit instruction to allow synchronization between unit instructions. The EIS processor eliminates delayed processing cycles found in VLIW and SVLIW processors because each unit instruction within a given long instruction word is allowed to be processed independently. The result of simulation promise faster execution cycles of the EIS processor than those of a VLIW or a SVLIW processor. Especially, the EIS processor can significantly reduce the execution cycles of programs that have high ratios of floating-point instructions.

I. 서 론

프로세서 구조에 대한 연구방향은 명령어를 순차적으로 실행시키는 프로세서 구조에서 보다 빠른 계산을 하기 위해서 명령어 수준의 병렬성(ILP: Instruction Level Parallelism)을 추출하여 동시에 여러 개의 명령어들을 실행시키는 프로세서 구조로 전환되고 있다. 이들 프로세서에서 명령어를 처리하기 위해서는 응용 프로그램을 구성하는 명령어간의 자료종속성을 분석하는 **종속성분석 단계**, 동시에 실행 가능한 명령어들을 추출하는 **독립성분석 단계** 및 가용한 자원에 명령어를 할당하는 **스케줄링 단**

* 천안외국어대 컴퓨터정보과 ** 충북대학교 컴퓨터과학과
논문번호 : 00311-0803, 접수일자 : 2000년 8월 3일
※ 본 논문은 정보통신부에서 주관한 대학기초연구과제의 일환으로 수행되었음

계의 세 가지 단계를 거쳐야 하며, 이러한 단계를 거친 후에야 비로소 명령어가 의미하는 연산을 실행하게 된다. 여기서 앞의 세 가지 단계가 컴파일과정에서 처리되는지 또는 하드웨어의 도움으로 실행과정에서 처리되는 지에 따라서 프로세서의 구성형태와 이를 지원하는 목적코드의 형태가 달라지게 된다.

수퍼스칼라(superscalar) 프로세서의 경우에는 종속성분석 단계에서 스케쥴링 단계에 이르는 세 가지 단계를 모두 하드웨어에서 처리하므로 프로세서를 구성하는 하드웨어 구조가 복잡하다. 그러나 이 구조를 위하여 생성된 목적코드는 순차코드의 형태와 대동소이하다[1-4]. 이에 반하여 VLIW(Very Long Instruction Word) 프로세서는 세 가지 단계를 모두 컴파일러가 담당하므로 목적코드는 동시에 실행 가능한 명령어들의 묶음인 긴명령어의 실행순서에 의해 구성되어야 한다. 따라서 VLIW 프로세서는 그 구조가 매우 간단한 대신 캐시미스와 같은 예상치 못한 실행상황이 발생할 경우 명령어 실행 형태에 영향을 끼치지 못하도록 명령어 실행 과정을 강제하는 등 제약이 많이 있다[5-8]. 이러한 두 가지 극단적인 프로세서 구조의 장점을 혼합한 프로세서 구조에 대한 연구도 진행되었다. 예를 들어, SVLIW (Superscalar VLIW) 프로세서[9-13]에서는 수퍼스칼라 프로세서 구조에서 하나의 명령어 단위로 자료종속성을 검사하고 스케쥴링하는 것을 긴명령어 단위로 자료종속성을 검사하거나 스케쥴링하도록 한다. 따라서 SVLIW 프로세서는 VLIW 프로세서의 경우에 비하여 목적코드 내에 삽입된 NOP으로만 구성된 긴명령어(LNOP)를 제거할 수 있으므로 작은 크기의 명령어 캐시만으로도 명령어 캐시미스율을 충분히 낮출 수 있는 장점이 있다. 그럼에도 불구하고 SVLIW 프로세서는 VLIW 프로세서와 동일하게 긴명령어 단위로 명령어 스케쥴링을 수행하므로 동일한 긴명령어에 속하는 모든 명령어들의 실행이 종료하기 전에는 다음 번 긴명령어의 실행을 개시할 수 없다.

이러한 문제점을 해결하기 위한 방안으로 제안된 VFPE 프로세서[14-18]는 연산처리기별로 독립된 스케쥴링 로직을 포함하도록 하여 긴명령어를 구성하는 여러 개의 단위명령어들간의 동기 문제를 실시간에 처리하도록 하였다. 그러나 이 구조에서는 긴명령어를 구성하는 과정을 단순화하기 위하여 프로세서를 구성하는 연산처리기별로 수행되는 명령어의 유형을 명확히 구분하고 각 명령어 유형별로 하나의 연산처리기만 하드웨어에 포함되어야 하는 제약이 있으므로 VLIW 프로세서나 SVLIW 프로세서와 같이 연산처리기의 수를 다양하게 할 수 없는 단점이 있다[14,18].

본 논문에서는 VFPE 프로세서 구조의 단점을 해결하기 위하여 종류별로 구분된 연산처리기의 수를 제한하지 않은 프로세서 구조인 ***EIS(Enhanced Individual Scheduling)* 프로세서 구조**를 연구하였다. EIS 프로세서는 프로세서를 구성하는 연산처리기 별로 동적 스케쥴러를 가지고 있으며 각 연산처리기에서 실행되는 명령어간의 동기화를 위한 회로를 포함한다. 즉, EIS 프로세서 내의 각 연산처리기는 자신이 실행할 명령어의 개시 여부를 이웃한 연산처리기에서 수행되어야 하는 명령어와의 동기정보를 이용하여 결정하도록 한다. 따라서 만일 이웃한 연산처리기에서 수행되는 또는 수행될 명령어들과의 동기가 불필요한 명령어인 경우에는 비록 이들이 긴명령어를 구성하고 있는 명령어일지라도 연산을 시작하도록 한다. 그러므로 EIS 프로세서는 SVLIW 프로세서에서 긴명령어를 구성하는 명령어들간에 존재하는 강제적인 동기로 인한 연산처리기의 불필요한 휴지(idling)문제를 해소할 수 있는 장점이 있다.

본 논문의 구성은 다음과 같다. 제 2장에서는 기존의 명령어 수준 프로세서 구조의 특징을 네 가지 병렬처리 모델로 구분하고 각 모델별 특징을 개괄하였다. 제 3장에서는 프로세서내 각 연산처리기가 독립적으로 스케쥴링을 수행하는 EIS 프로세서 구조를 설계하였다. 그리고 제 4장에서는 EIS 프로세서의 성능을 VLIW 프로세서 및 SVLIW 프로세서의 경우와 비교하기 위하여 각각의 시뮬레이션 시스템을 구현하고 여러 가지 자료종속성을 갖는 종속성 그래프를 대상으로 실행 사이클을 측정하여 EIS 프로세서 구조가 기존의 프로세서 구조에 비하여 성능이 우수함을 확인하였다. 마지막으로 제 5장에서는 결론을 내리고 향후 계획을 기술하였다.

Ⅱ. 병렬처리 모델 분석

2.1 병렬처리 모델 분류

순차 프로그램에서 병렬처리를 수행하기 위해서는 1) 명령어간의 종속성 관계를 파악하는 종속성분석 단계, 2) 동시에 실행가능한 명령어들을 추출하는 독립성분석 단계, 3) 유용한 자원의 범위 내에서 명령어들을 스케쥴링하는 스케쥴링 단계가 순서대로

수행되어야 한다. 이때 모든 단계가 컴파일시간에 수행된다면 컴파일 동안에 얻어진 병렬정보들을 목적코드 내에 포함시켜야 하므로 목적코드의 형태는 크고 복잡해지는 반면, 프로세서 구조는 단순해질 것이다. 그러나 모든 단계가 실시간에 수행된다면 목적코드의 형태는 단순한 반면, 프로세서 구조는 복잡해질 것이다. 이와 같이 병렬처리에 필요한 각 단계의 수행시기에 따라서 목적코드의 구성형태와 이를 수행하는 프로세서 구조가 달라진다[18].

수퍼스칼라 프로세서용 목적코드의 형태는 순차 프로세서용으로 생성된 목적코드의 구성형태와 동일하다. 즉, 목적코드를 구성하는 명령어의 나열 순서는 프로세서에서 명령어를 인출하는 순서를 결정하며, 연산처리기에서 실행되는 명령어 이외에 별도의 프로세서를 위한 특별한 코드가 추가되지 않는다. 따라서 수퍼스칼라 프로세서를 위하여 생성된 목적코드는 순차 프로세서에서 그대로 사용될 수 있다. 다만, 이들 목적코드는 포함하고 있는 연산처리기의 수나 특별한 하드웨어를 지원하기 위하여 컴파일러가 코드 최적화 과정을 거치면서 보다 빠른 계산이 가능하도록 명령어를 재배치할 뿐이다. 따라서 본 논문에서는 이러한 형태의 목적코드를 수행하는 프로세서 모델을 그림 1에 보인바와 같이 *SEQ* 모델이라고 분류하였다. 즉, SEQ 모델을 적용하는 프로세서용 목적코드에는 실제로 수행되는 명령어 외에 이들 명령어간의 자료종속성 정보 또는 명령어간의 동기를 위한 정보가 추가되지 않는다고 간주한다.

VLIW 프로세서용 목적코드는 순차코드에서 프로세서를 구성하고 있는 연산처리기 수만큼의 동시에 실행가능한 명령어를 하나의 긴명령어로 구성하는 절차를 거쳐서 생성된다. 이 과정에서 하나의 긴명령어를 구성할 수 있는 동시에 수행가능한 명령어가 연산처리기의 수보다 적은 경우에는 긴명령어의 나머지 부분을 NOP (No OPeration)으로 채워야 한다. 따라서 VLIW 프로세서용 목적코드를 생성하는 컴파일러는 연산처리기가 매 사이클마다 수행하여야 하는 실행 패턴에 맞추어 긴명령어를 구성하여야 한다. 본 논문에서는 이러한 형태의 목적코드를 수행하는 프로세서 모델을 *PICS* 모델이라고 부르기로 한다. 즉, PICS 모델은 긴명령어 단위로 명령어가 인출되고 실행되므로 긴명령어를 구성하는 명령어들 가운데 실행이 종료된 명령어들이 있다고 하더라도 실행중인 다른 명령어가 실행을 종료할 때까지 대기중인 다음 번 긴명령어를 실행할 수 없다.

이와는 달리 SVLIW 프로세서[9-13]는 VLIW 프로세서에서 긴명령어 단위의 실시간 스케줄링이 가능하도록 확장한 구조이다. 즉, SVLIW 프로세서 구조는 긴명령어 간의 자료종속성 및 자원 충돌 여부가 실시간에 확인되고 이 결과에 의하여 긴명령어가 스케줄링될 수 있다. 그러므로 SVLIW 모델의 경우에는 PICS 모델에서 연산처리기의 실행 패턴을 강제하기 위하여 삽입되었던 NOP으로만 구성된 긴명령어(LNOP)가 필요없다. 따라서 이 모델의 목적코드는 PICS 모델에 비하여 짧아지는 장점이 있다. 본 논문에서는 이러한 형태의 목적코드를 수행하는 프로세서 모델을 *PILS*모델이라고 부르기로 한다.

PILS 모델에서도 긴명령어 단위로 인출, 디코드, 스케줄링 및 연산이 이루어지므로 PICS 모델과 마찬가지로 긴명령어를 구성하는 명령어들 가운데 실행이 종료된 명령어들이 있더라도 실행중인 다른 명령어가 실행을 종료할 때까지 기다려야 한다. 이와 같은 PILS 모델의 목적코드 실행 패턴은 PICS 모델의 실행 패턴과 대동소이하다. 그러나 만약 긴명령어 단위로 인출, 디코드가 된다고 하더라도 스케줄링 단계에서 긴명령어를 구성하는 각각의 명령어가 독립적으로 스케줄링 및 연산을 수행한다면 불필요한 실행지연 문제점을 개선할 수 있다. 예를 들어 VFPE 프로세서[14,18]에서는 PILS 모델과 같이 긴명령어 단위로 인출 및 디코딩이 이루어지되 스케줄링은 명령어 단위로 이루어지도록 하고 있다.

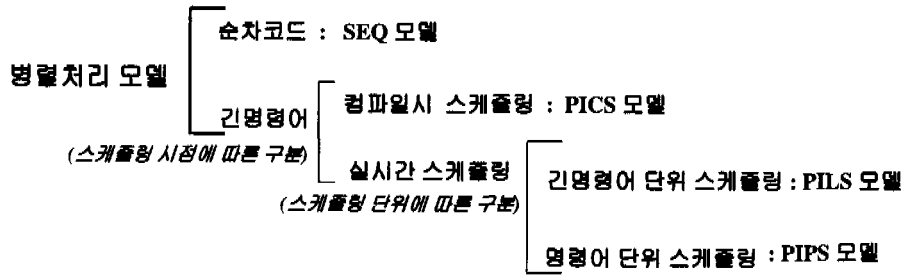


그림 1. 병렬처리 모델 분류

이를 위하여 VFPE 프로세서 구조에서는 서로 다른 연산처리기에서 수행되는 명령어간의 동기를 위한 정보가 목적코드에 추가되어 있다. 본 논문에서는 이러한 형태의 목적코드를 수행하는 프로세서 모델을 *PIPS*모델이라고 부르기로 한다. 여기서 PIPS 모델의 프로세서는 기존의 VFPE 프로세서에서 실행할 명령어 유형에 따라서 특정 연산처리기를 사용하도록 명령어를 할당하는 제약은 없는 것으로 간주하였다. 이를 위해서 기존의 VFPE 프로세서용 목적코드와 비교하면, 본 논문에서 제안하는 PIPS 모델은 명령어간의 동기 정보가 더 많이 필요할 수도 있으나 이 부분은 실험 부분에서 보다 자세하게 설명할 것이다.

2.2 모델별 명령어 실행 흐름

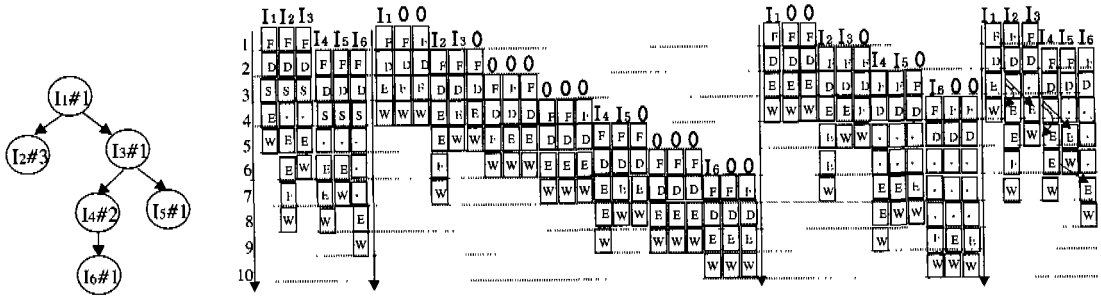
본 절에서는 앞에서 제안한 네 가지 모델별로 주어진 명령어 그래프를 실행하는 패턴을 살펴보기로 한다. 그림 2(a)은 6개의 명령어로 구성된 명령어 그래프이다. 그림 2(a)에서 동그라미 안의 라벨은 명령어의 번호, #(구분선), 그리고 각 명령어의 연산에 필요한 연산 사이클의 수를 의미한다. 즉, 명령어 I_1은 1사이클이 필요하며 I_2는 3사이클이 필요하다. 또한, 모델별 각 프로세서는 세 개의 연산처리기로 구성되며 명령어 파이프라인은 인출(F), 분석(D), 실행(E) 및 쓰기(W)의 네 단계라고 가정하나 SEQ 모델은 스케줄링 단계(S)가 포함되므로 다섯 단계의 명령어 파이프라인을 가진다고 가정하였다.

SEQ 모델의 명령어실행 흐름은 그림 2(b)와 같다. 예를 들어, 명령어 I_3와 명령어 I_4, I_5간은 자료종속성 관계이므로 I_3의 실행이 종료될 때까지 I_4와 I_5는 '·'으로 표현된 부분인 5 번째 사이클에서 실행을 기다린다. SEQ 모델에서 그림 2(a)를 실행하는 데는 총 9 사이클이 필요하다.

그림 2(c)에 보인 PICS 모델의 명령어실행 흐름에서 명령어 I_3와 명령어 I_4, I_5간에는 자료종속성 관계가 존재하므로 긴명령어 $(I_2\ I_3\ 0)$와 $(I_4\ I_5\ 0)$ 사이에 두 개의 LNOP을 삽입하여 $(I_2\ I_3\ 0)$의 실행이 종료된 후에 $(I_4\ I_5\ 0)$가 실행되어야 한다. 따라서 PICS모델의 경우에는 총 10 사이클이 필요하다. PILS 모델의 명령어 실행 흐름은 그림 2(d)와 같다. 즉, 긴명령어 $(I_4\ I_5\ 0)$와 앞서 실행중인 긴명령어 $(I_2\ I_3\ 0)$간의 자료 종속성을 제거하기 위해서, $(I_4\ I_5\ 0)$는 $(I_2\ I_3\ 0)$의 실행이 종료될 때까지 '·'으로 표현된 부분과 같이 5 번째 사이클과 6 번째 사이클의 두사이클 동안 대기하였다가 실행 사이클로 진입하게된다. 따라서 이 모델의 경우에도 PICS 모델과 같이 10 사이클이 필요하다.

PIPS 모델의 경우를 보면, 그림 2(e)와 같이 자료종속성 관계가 있는 명령어들이 긴명령어를 구성할 수 있다고 가정하자. 즉, 명령어 $(I_1\ I_2\ I_3)$와 $(I_4\ I_5\ I_6)$이 각각 명령어 동기를 위한 정보를 포함한다고 간주하자. 따라서 긴명령어 $(I_1\ I_2\ I_3)$가 인출되어 분석 단계가 완료되면 명령어 I_1은 실행 단계로 진입하여 I_1의 실행이 완료된 후에야 다음 사이클에서 명령어 I_2와 I_3가 실행단계로 진입할 수 있다. 그림 2(e)의 화살표가 이를 보여준다. 마찬가지로 명령어 I_4와 I_5도 명령어 I_3의 실행이 종료될 때까지 대기(그림 2에서 ·으로 표현한 부분)하였다가 실행단계로 진입해야 한다. 이와 같이 동작하는 PIPS 모델의 경우에 그림 2(a)에 보인 그래프를 실행하는데 필요한 시간은 총 8 사이클이다.

이상의 관찰을 토대로 PIPS 모델이 기존의 PICS 모델이나 PILS 모델에 비하여 적은 실행 사이클을



(a)명령어 그래프 (b) SEQ (c) PICS (d)PILS (e) PIPS

그림 2. 프로세서 구조별 명령어실행 흐름

요구함을 알 수 있다. 그 이유는 자료종속성이 있는 명령어들을 하나의 긴명령어로 구성하여 명령어인출 단계에서 동시에 인출함으로 인해서 다른 모델에 비하여 프로그램 실행 시에 소요되는 명령어인출 사이클을 줄일 수 있기 때문이다. 또한, PIPS 모델에서는 각 연산처리기가 명령어의 실행을 종료하자마자 같은 긴명령어를 구성하였던 다른 명령어들의 실행여부에 관계없이 곧바로 다음 명령어를 실행할 수 있다. 그에 비해, PICS 모델과 PILS 모델에서는 어떤 연산처리기가 먼저 명령어의 실행을 종료하더라도 같은 긴명령어를 구성하는 명령어들의 실행이 종료될 때까지 다음 긴명령어를 구성하는 명령어를 할당받아 실행할 수 없다. 따라서 PIPS 모델을 적용한 프로세서 구조가 전술한 PICS 모델이나 PILS 모델에 비하여 우수함을 알 수 있다. 본 논문에서는 PIPS 모델을 적용한 프로세서 구조를 제안하였으며, 이를 EIS(Enhanced Independent Scheduling) 프로세서라고 명명하였다.

Ⅲ. EIS 프로세서 구조 설계

3.1 EIS 프로세서와 목적코드

본 논문에서 제안하는 EIS 프로세서 구조내의 각 연산처리기가 실행 중에 다른 연산처리기와 상관없이 독립적으로 실행되기 위해서는 실시간에 자료종속성 정보를 이용하여 명령어간 동기를 제어할 수 있어야 한다. 즉, 긴명령어를 구성하는 명령어들간의 동기를 위해서는 명령어간의 자료종속성 관계가 표현된 목적코드를 생성하여야 한다. 목적코드에는 명령어 별로 **선행종속성**(pre-dependency) 정보와 **후행종속성**(post-dependency) 정보가 추가된다. 여기서 선행종속성 정보란 해당 명령어에 앞서서 먼저 수행되어야 하는 명령어를 처리하는 연산처리기에 관한 정보를 의미하며, 후행종속성 정보는 해당 명령어가 수행된 이후에 수행되어야 하는 명령어들이 처리되는 연산처리기에 관한 정보를 나타낸다. 그림 3은 EIS 프로세서에서 사용될 긴명령어의 구성형태이다.

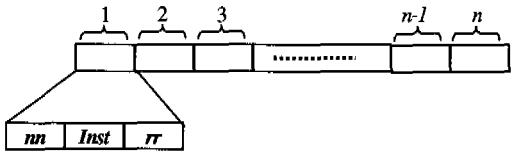


그림 3. 긴명령어의 구성형태

그림 3에서 긴명령어를 구성하는 단위 명령어의 수는 n이며, 각 단위 명령어는 nn로 표시된 선행종속성 정보, $Inst$으로 표시한 명령어 및 rr로 나타낸 후행종속성의 세 영역으로 구성된다.

이상과 같이 목적코드 내의 각 명령어는 명령어에 포함된 두 가지 종속성 정보를 토대로 이웃한 연산처리기에서 수행되는 명령어들간의 동기가 가능하다. 즉, 주어진 명령어가 연산처리기에서 실행되기 위해서는 주어진 명령어와 선행종속성 관계에 있는 명령어가 실행되는(또는 실행될) 연산처리기로부터 해당 명령어의 실행이 완료되었음을 통보받은 후에야 실행된다. 또한, 주어진 명령어가 실행된 후에는 후행종속성 관계에 있는 명령어를 실행할 연산처리기로 실행이 종료되었음을 통보하여 해당 명령어가 실행될 수 있도록 한다. 이때 이들 종속성 정보는 각각 연산처리기 만큼의 비트 스트림(bit stream)으로 표현할 수 있으므로 각 명령어당 추가되어야 하는 비트 스트림 길이는 연산처리기 수의 두 배가 된다.

그림 4는 EIS(Enhanced Independent Scheduling) 프로세서 구조이다. EIS 프로세서는 연산처리기(FU: Fetch Unit), 명령어 큐(IQ: Instruction Queue), 동적 스케줄러(DS: Dynamic Scheduler), 레지스터 파일(register file), 명령어 캐시(instruction cache), 데이터 캐시(data cache) 및 BTB (Branch target buffer)로 구성된다 여기서 명령어 큐는 EIS 프로세서에서 인출된 긴명령어를 단위 명령어 별로 구분하여 각 연산처리기의 동적 스케줄러에 공급하는 역할을 담당한다. 동적 스케줄러는 실행중인 명령어와 실행하려는 명령어간에 자원 충돌이나 자료 종속성이 발생하는지 여부를 검사하고 문제점이 발생하면 문제점이 제거될 때까지 실행을 지연시킨 후에 실행할 명령어를 유용한 자원에게 할당한다. 또한 EIS 프로세서 구조는 BTB 구조를 적용한 분기예측기를 이용하여 분기명령어의 인출 시에 BTB 구조에 저장된 분기명령어의 분기예측 정보를 이용하여 곧바로 다음 사이클에 타겟명령어를 인출하도록 한다

EIS 프로세서는 명령어의 실행 과정이 그림 5와 같이 네 단계의 파이프라인을 거친다. 이때 명령어 실행 과정은 명령어 유형에 따라서 점유하는 파이프라인 단계가 서로 다르다. EIS 프로세서는 파이프라인 단계를 크게 산술연산 명령어, 메모리참조 명령어 및 분기명령어의 세 가지 명령어 유형으로 구분하고, 각 유형의 명령어 파이프라인은 실행단계

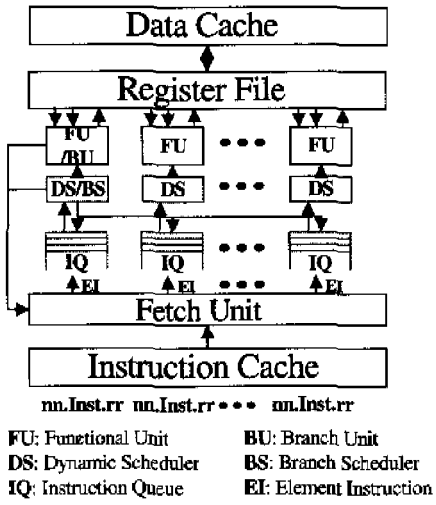


그림 4. EIS 프로세서 구조

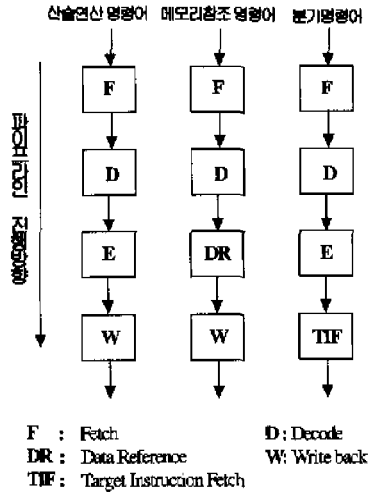


그림 5. 명령어 파이프라인

외에는 매 단계를 한 사이클에 수행한다.

그림 6는 EIS 프로세서 구조에서의 명령어파이프라인(그림 5)의 실행알고리즘을 나타낸 것이다. 인출(F: fetch) 단계는 매 사이클마다 캐시로부터 하나 외 긴명령어를 인출하고 인출한 긴명령어를 명령어로 분할하여 각 연산처리기의 명령어 큐에 적재한다. 분석(D: decode) 단계는 각 연산처리기내 명령어 큐의 최상위(front)에 적재된 명령어를 가져와 분석하여 명령어 실행에 필요한 정보를 알아낸다. 또한, 분석 단계에서는 명령어에 포함된 자료종속성을 분석하여 해당 명령어의 스케줄링이 가능한지 여부를 검사한다. 즉, 명령어 내에 포함되어 있는 선행종속성 정보와 후행종속성 정보를 토대로 실행하고자 하는 명령어가 현재 다른 연산처리기에서 실행중인 명령어와 자원 충돌을 일으키는지 또는 자료종속성이 발생하는 지를 분석하여 만약 이러한 문제가 발생하지 않으면 해당 연산처리기의 실행 단계로 명령어를 진입하도록 한다. 만약 실행중인 명령어와의 사이에서 자원 충돌이나 자료 종속성이 존재하면 이러한 문제가 해결될 때까지 이 명령어가 실행 단계로 돌입하지 못한다. 이러한 과정을 자원 충돌이나 자료 종속성이 해결될 때까지 반복해서 수행한다. EIS 프로세서내 각 연산처리기는 이와 같은 분석 단계의 수행절차를 통하여 다른 연산처리기와 상관없이 독립적으로 수행할 수 있다. 실행(E: execute) 단계에서는 분석 단계로부터 전달된 명령어들을 연산처리기에 할당하여 실행을 수행한다. 실행중인 명령어는 마지막 실행 사이클 동안에 자신의 후행종속성 정보를 이용하여 앞으로 실행할 자료종속성 관계가 있는 명령어의 선행종속성 정보를 리세트함으로써 선행종속성으로 인하여 실행단계에 진입하지 못하는 명령어들이 다음 사이클에서 실행단계로 진입할 수 있도록 한다. 예를 들어, 여러 사이클이 소요되는 실수명령어의 경우에는 이 명령어가 실행 단계의 마지막 사이클이 수행되는 과정에서 그 명령어의 후행종속성 정보가 지시하는 연산처리기의 종속성 정보를 리세트함으로써 선행종속성으로 인하여 실행을 기다리고 있는 명령어들이 다음 사이클에서 실행 단계로 진입하도록 한다. 이러한 과정은 실행 사이클이 길고 비정형적인 실수명령어가 실행중 임의의 시점에서 후행종속성 정보를 잘못 변경하는 것을 방지하기 위해서 요구되며 명령어의 마지막 실행단계에서 이와 같이 분석단계를 제어하도록 한다. 분기명령어의 경우에 실행단계에서는 분기예측이 적중했는가를 판단하여 만일 적중했으면 다음 단계에서 BTB 구조내의 분기예측 정보를 갱신하는 작업을 수행하고 만약 적중하지 않았으면 현재 파이프라인에서 인출, 분석, 실행하고 있는 명령어들을 모두 제거하는 작업을 수행한다. 쓰기(W: write back) 단계에서는 실행된 결과값을 레지스터에 저장하는 작업을 수행한다.

3.2 명령어실행 특성

그림 7은 9개의 명령어로 구성된 예제 프로그램을 기존의 PICS 모델을 적용한 VLIW 프로세서와 PILS 모델을 적용한 SVLIW 프로세서, 그리고

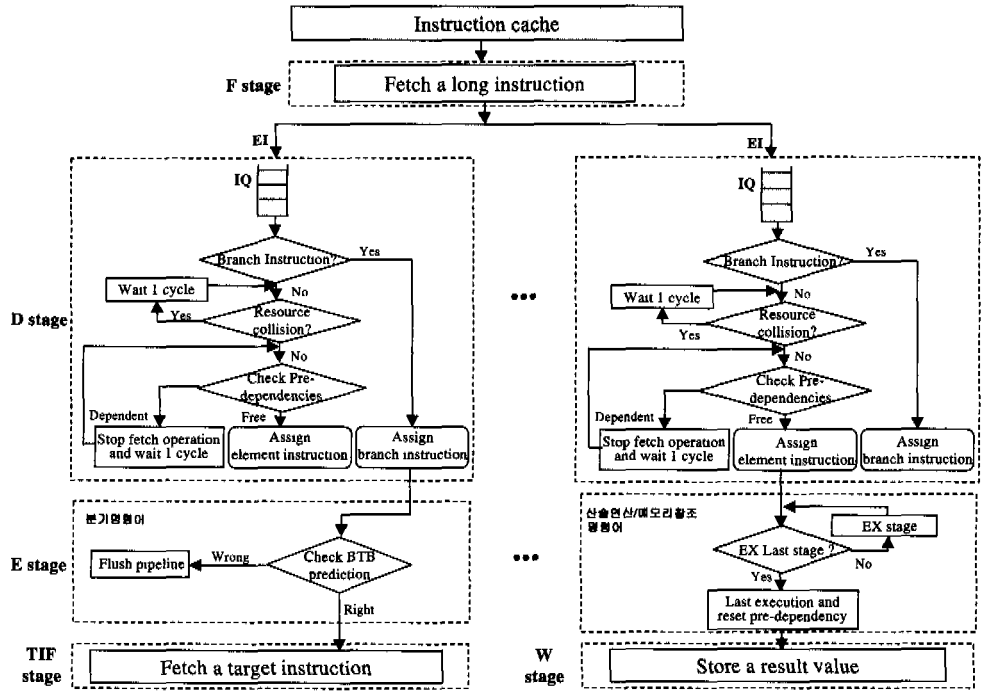


그림 6. 명령어 파이프라인 과정

PIPS 모델을 적용한 EIS 프로세서에서 실행할 경우의 명령어실행 패턴을 보여준다. 여기서 메모리참조 명령어 *LD*와 *STOR*, 덧셈명령어 *ADD* 및 뺄셈명령어 *SUB*의 실행 사이클은 1, 곱셈명령어 *MUL*과 나눗셈 명령어 *DIV*의 실행 사이클은 각각 2와 3이라고 가정한다. 또한, 이 논문에서 다룰 모든 프로세서 구조는 인출(F), 분석(D), 실행(E), 쓰기(W)인 네 단계의 명령어 파이프라인을 가지며 실행단계의 계산 결과를 다음 명령어의 실행단계에서 사용하도록 by-pass 선로[8]가 존재한다고 가정한다. 그림 7(a-1)의 예제 프로그램에 대한 명령어 그래프는 그림 7(a-2)이다.

그림 7(b-1)은 그림 7(a-1)로부터 생성된 VLIW 프로세서용 목적코드이다. 그림 7(a-1)에서 5, 6번째 명령어 *DIV*, *SUB*와 7번째 명령어 *ADD*는 자료종속성 관계이므로 서로 다른 긴명령어 I_6, I_9으로 생성된다. 이때 긴명령어 I_6와 I_9는 자료종속성 관계이므로 이를 제거하기 위해 두 개의 LNOP 명령어인 I_7, I_8이 삽입된다. 그림 7(c-1)은 그림 7(b-1)에 대한 명령어실행 패턴이다. 이때 VLIW 프로세서는 메모리에 적재된 긴명령어를 매 사이클마다 순서대로 읽어서 긴명령어 단위로 스케줄링을 수행한다. 그림 7(a-1)를 VLIW 프로세서에서 수행하기 위해서는 15 사이클이 필요하다.

그림 7(b-2)는 그림 7(a-1)로부터 생성된 SVLIW 프로세서용 목적코드이다. 그림 7(a-1)에서 5, 6번째 명령어 *DIV*, *SUB*와 7번째 명령어 *ADD*는 자료종속성 관계이므로 서로 다른 긴명령어 I_4, I_5로 생성된다. 이때 긴명령어 I_4와 I_5간에 존재하는 자료종속성은 I_4의 실행이 종료될 때까지 I_5의 실행을 지연시킴으로써(·으로 나타냄) 실시간에 제거되므로 목적코드에 별도의 LNOP을 삽입할 필요가 없다. SVLIW 프로세서는 VLIW 프로세서의 실행방식과 동일하게 매 사이클마다 긴명령어 단위로 명령어 스케줄링을 수행한다. 그림 7(a-1)를 SVLIW 프로세서에서 수행하기 위해서는 15 사이클이 필요하다.

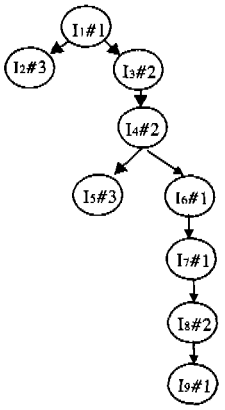
그림 7(b-3)은 그림 7(a-1)로부터 생성된 EIS 프로세서용 목적코드이다. 그림 7(a-1)에서 5, 6번째 명령어 *DIV*, *SUB*와 7번째 명령어 *ADD*는 자료종속성 관계이므로 긴명령어 I_2를 구성하는 명령어 *DIV*, *SUB*의 후행종속성 정보는 4(비트패턴 100)이고 긴명령어 I_3를 구성하는 명령어 *ADD*의 선행종속성 정보는 3(비트패턴 011)이다. EIS 프로세서는 그림 7(c-3)와 같이 자료 종속성 관계가 있는 명령어들을 동시에 인출하며 서로 다른 사이클에서

I 1
I 2
I 3
I 4
I 5
I 6
I 7
I 8
I 9

명령어 번호

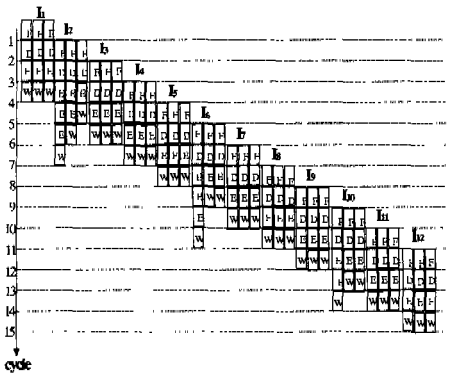
LD	R_1,	A	
DIV	R_2,	R_1,	R_{20}
MUL	R_3,	R_1,	R_{21}
ADD	R_4,	R_1,	R_3
DIV	R_5,	R_4,	R_{22}
SUB	R_6,	R_4,	R_{23}
ADD	R_7,	R_5,	R_6
MUL	R_8,	R_7,	R_1
STOR	R_8,	B	

(a-1) 예제 프로그램

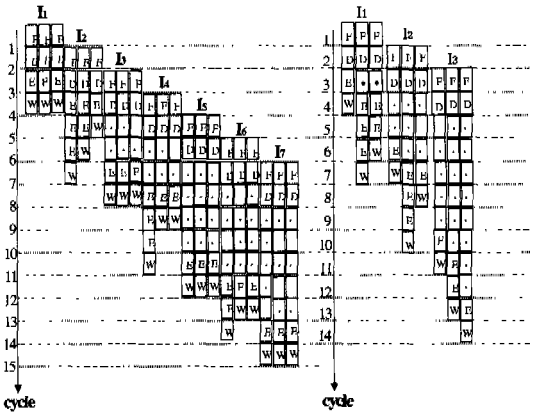


(a-2) 명령어그래프

긴명령어 번호	(b-1) VLIW			(b-2) SVLIW			(b-3) EIS		
I1	LD	NOP	NOP	LD	NOP	NOP	0.LD.3	4.DIV.0	4.MUL.4
I2	DIV	MUL	NOP	DIV	MUL	NOP	1.ADD.3	4.DIV.4	4.SUB.4
I3	NOP	NOP	NOP	ADD	NOP	NOP	3.ADD.2	4.MUL.1	2.STOR.0
I4	NOP	NOP	NOP	DIV	SUB	NOP			
I5	ADD	NOP	NOP	ADD	NOP	NOP			
I6	DIV	SUB	NOP	MIL	NOP	NOP			
I7	NOP	NOP	NOP	STOR	NOP	NOP			
I8	NOP	NOP	NOP						
I9	ADD	NOP	NOP						
I10	MIL	NOP	NOP						
I11	NOP	NOP	NOP						
I12	STOR	NOP	NOP						



(c-1) VLIW



(c-2) SVLIW (c-3) EIS

그림 7 프로세서 구조별 명령어실행 과정

인출된 명령어 사이에도 자료종속성이나 자원 충돌이 발생하지 않으면 동시에 실행할 수 있다. 즉, 6번째 사이클에서 자료종속성이 없는 I_1의 두 번째 명령어 *DIV*와 I_2의 첫 번째 명령어 *ADD*가 동시에 실행하는 것을 볼 수 있다. 명령어 파이프라인 실행 중에 나타나는 ·은 실행하고자 하는 명령어와 현재 실행중인 명령어들 간에 자원 충돌이나 자료 종속성이 발생하여 실행하고자 하는 명령어의 실행이 해당 사이클동안 지연되는 것을 의미한다. 이러한 방법으로 EIS 프로세서용 목적코드를 실행하는데는 그림 7(c-3)과 같이 14 사이클이 필요하다. 이는 VLIW 프로세서나 SVLIW 프로세서에서 실행할 경우에 각각 15 사이클이 소요되는 것과 비교할 때, 1 사이클이 줄어든 것이다.

Ⅳ. 실험 및 고찰

본 실험에서는 기존의 VLIW 프로세서, SVLIW 프로세서 및 EIS 프로세서 구조의 성능을 평가하기 위해서, 다양한 명령어 그래프로부터 생성한 목적코드를 실행하였을 경우에 각 프로세서 구조별로 요구하는 실행 사이클을 측정하여 프로세서간 성능을 비교하였다.

그림 8은 시뮬레이션 시스템 구조이다. 그림 8에서 그래프 생성기(graph generator)는 다양한 입력 값을 받아서 자료종속성을 가진 명령어 그래프를 임의로 생성한다. 이때 입력 값은 기본 블럭의 수, 기본 블럭을 구성하는 명령어 수, 정수명령어/실수명령어분포 비율, 실수명령어의 실행 사이클, 자료종속성 비율, 루프의 실행횟수이다. 다양한 형태의 명령어 그래프를 구성하기 위해서, 정수명령어/실수명령어의 분포 비율과 명령어간에 존재하는 자료종속성 비율은 각각 100/0, 80/20, 60/40, 40/60, 20/80와 20%, 40%, 60%, 80%으로 변화시키면서 주어진다. 병렬화 어셈블러(parallelizing assembler)는 생성된 명령어 그래프를 입력으로 받아서 긴명령어 단위로 목적코드를 구성한다. 프로세서 시뮬레이터(processor simulator)는 프로세서 구조별로 생성된 목적코드를 실행하는 명령어 파이프라인을 모의한다. 병렬화 어셈블러로부터 생성된 EIS용 목적코드는 자료종속성 정보를 포함하고 있더라도 VLIW용 목적코드와 SVLIW용 목적코드 내에 포함되는 LNOP이나 NOP으로 인한 목적코드의 증가보다 낮다[15,16].

그림 9는 명령어 그래프내의 자료종속성 비율 τ을 변화시키면서 생성한 프로세서 구조별 목적코드를 각각의 프로세서에서 실행하였을 때 성능비(speedup)를 측정한 것이다. 프로세서간 성능비 ϕ를 동일한 프로그램에 대한 실행사이클의 비로 정의하면 식과 같다.

$$\phi = \frac{t_1}{t_n} \tag{1}$$

식 (1)에서 t_1은 순차프로세서에서의 실행사이클을 의미하고 t_n은 연산처리기가 n개인 프로세서에서의 실행사이클을 의미한다. 이 실험에서는 프로세서 구조가 네 개의 연산처리기로 구성된다고 가정하고 명령어 그래프를 구성하기 위한 입력 값은 기본 블럭의 수 500, 기본 블록내 명령어 수 15이 주어진다.

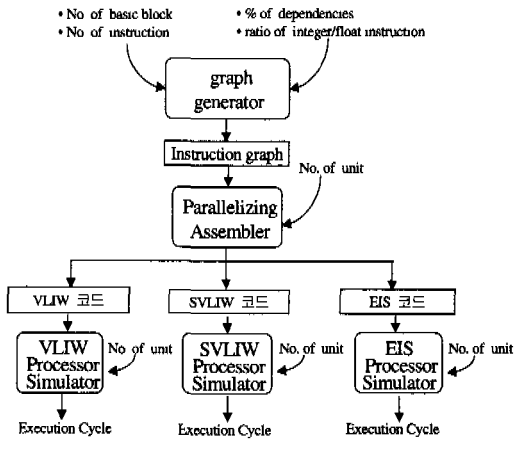
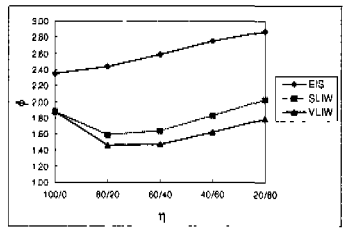
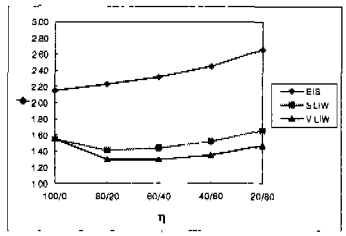


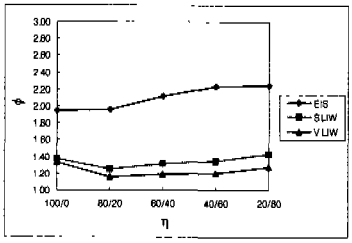
그림 8. 시뮬레이션 시스템



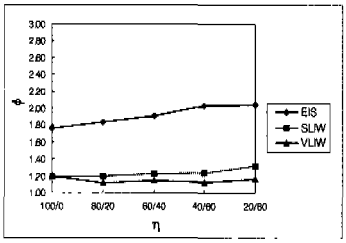
(a) τ=20%



(b) τ=40%



(c) τ=60%



(d) τ=80%

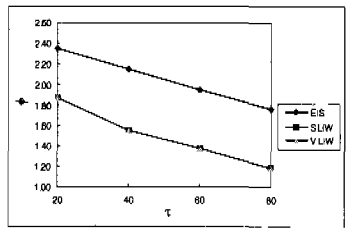
η: 정수명령어/실수명령어분포 비율,
τ: 자료종속성 비율

그림 9. 자료종속성 비율의 변화에 따른 성능 비교

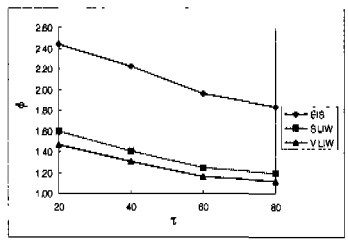
그림 9의 실험을 통하여 EIS 프로세서가 VLIW 프로세서와 SVLIW 프로세서와 비교할 때 자료종속성 비율의 변화에 관계없이 항상 높은 성능비를 가짐을 알 수 있다. 이는 EIS 프로세서가 명령어 단위로 독립적으로 스케줄링을 수행하는 것과 달리, VLIW 프로세서와 SVLIW 프로세서는 긴명령어 단위로 순차적으로 스케줄링을 수행하기 때문이다. 즉 긴명령어 단위의 스케줄링 방식을 이용하면, 실행하려는 긴명령어는 먼저 실행 중인 모든 명령어들의 실행이 종료된 후에야 실행을 시작할 수 있으므로 목적코드 내에 실행 사이클이 길고 비정형적인 실수명령어가 많을수록 또는 자료종속성 비율이 높을수록 더 많은 실행 사이클을 요구하게 된다. 그러므로 실수명령어가 많이 포함된 목적코드를 실행할 경우에는 명령어별로 독립적인 스케줄링을 수행하는 EIS 프로세서가 긴명령어 단위로 스케줄링을 수행하는 VLIW 프로세서나 SVLIW 프로세서와 비교할 때 상대적으로 높은 성능비를 가진다.

그림 9에서 VLIW 프로세서와 SVLIW 프로세서의 성능비가 포물선형을 나타내는 것은 목적코드 내의 명령어분포 비율 때문이다. 즉, 목적코드 내의 명령어분포 비율이 100/0, 20/80, 80/20와 같이 정수명령어 혹은 실수명령어인 한 유형의 명령어들로 편중되어 있으면 실행사이클의 격차가 작은 명령어들로 긴명령어를 구성하는데, 이렇게 구성된 긴명령어의 성능효율은 명령어분포 비율이 60/40, 40/60와 같이 다른 유형의 실행 사이클 격차가 큰 명령어들로 구성된 긴명령어의 경우에 비해 우수하기 때문이다.

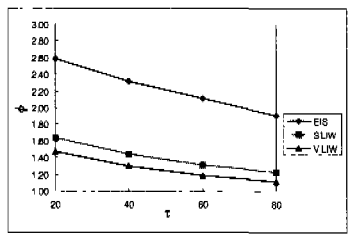
그림 10은 명령어 그래프내의 정수명령어/실수명령어분포 비율을 변화시키면서 생성한 프로세서별 목적코드를 VLIW 프로세서, SVLIW 프로세서, 및 EIS 프로세서에서 실행하였을 때 성능비 ϕ를 측정한 것이다. 실험에서는 그림 9와 동일한 입력 값을 이용하여 명령어 그래프를 구성하였다.



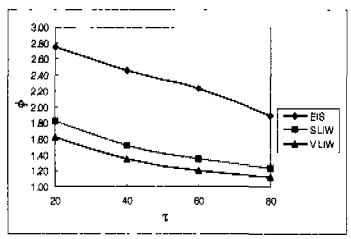
(a) η=100/0



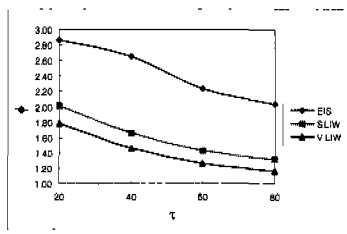
(b) η=80/20



(c) η=60/40



(d) η=40/60



(e) η=20/80

η: 정수명령어/실수명령어분포 비율,
τ: 자료종속성 비율

그림 10. 명령어 유형의 분포 변화에 따른 성능 비교

그림 10의 실험 결과를 통해서도 EIS 프로세서의 성능비가 VLIW 프로세서나 SVLIW 프로세서의 성능비와 비교할 때 우수함을 알 수 있다. EIS 프로세서가 우수한 이유는 목적코드의 길이가 자료종속성 비율에 상관없이 항상 일정할 뿐 아니라, 긴명령어를 구성하는 각 명령어가 다른 명령어들과 독립적으로 수행하기 때문이다. 그에 비하여, VLIW 프로세서나 SVLIW 프로세서는 자료종속성 비율이 높아질수록 많은 양의 NOP과 LNOP을 목적코드에 삽입하므로 목적코드에서 유용한 명령어가 차지하는 범위가 그만큼 줄어든다. 또한 VLIW 프로세서나 SVLIW 프로세서는 긴명령어 단위의 스케줄링 방식으로 인하여 다음에 실행할 긴명령어는 실행중인 모든 명령어들의 실행이 종료된 후에야 실행을 시작할 수 있으므로 자료종속성 비율이 높아질수록 성능비도 점차적으로 낮아진다.

그림 10에서 세 프로세서 구조의 성능비가 모두 자료종속성 비율이 증가할수록 점차적으로 저하됨을 알 수 있다. 이것은 자료종속성 비율이 낮을수록 병렬수행으로 인한 성능향상이 높은 반면, 자료종속성 비율이 높아질수록 병렬수행을 할 명령어들이 그에 반비례하여 적어지므로 병렬수행으로 인한 성능향상이 낮기 때문이다. 또한, VLIW 프로세서와 SVLIW 프로세서는 정수명령어 혹은 실수명령어 가운데 한 유형의 분포 비율이 높은 경우에 성능비가 높다가 두 명령어 유형의 분포 비율이 비슷한 시점에서 성능비가 낮아지는 것을 알 수 있다. 이것은 VLIW 프로세서와 SVLIW 프로세서에서 목적코드 내의 명령어분포 비율이 정수명령어 혹은 실수명령어인 한 유형의 명령어들로 편중되어 있으면 실행 사이클의 격차가 작은 명령어들로 긴명령어를 구성하는데, 이렇게 구성된 긴명령어의 성능효율은 서로 다른 유형의 실행 사이클 격차가 큰 명령어들로 구성된 긴명령어의 성능효율에 비하여 우수하기 때문이다.

V. 결 론

명령어를 병렬처리하기 위해서는 명령어간의 종속성 관계를 파악하는 종속성분석 단계, 동시에 실행 가능한 명령어들을 추출하는 독립성분석 단계 그리고 명령어들을 유용한 자원에 할당하는 스케줄링 단계로 구성되는 일련의 과정이 수행되어야 한다. 이때 각 단계는 컴파일시간이나 실시간 중에 수행되어야 하며 각 단계의 수행시기에 따라 프로세서 구조와 목적코드 형태가 달라진다. 본 논문에서는 프로세서의 구조를 SEQ 모델, PICS 모델, PILS 모델 및 PIPS 모델의 네 가지로 구분하고 각 모델의 장·단점을 분석하였다. 그 결과, PIPS 모델이 긴명령어 내의 각 명령어들을 독립적으로 스케줄링 할 수 있으므로 비록 명령어간의 동기를 위하여 자료종속성 정보가 목적코드에 포함되어야 하는 단점이 있으나 다른 세 가지 모델에 비하여 동일한 응용 프로그램을 수행하는데 필요한 명령어 실행 사이클이 가장 짧은 장점이 있다. 이에 따라 본 논문에서는 PIPS 모델에 근거한 EIS 프로세서를 설계하였다.

제안한 EIS 프로세서 구조는 각각의 연산처리기에 동적 스케줄러를 포함시켜 하나의 쌍으로 구성하여 각각의 연산처리기가 다른 연산처리기들의 명령어 실행여부와 상관없이 독립적으로 실행할 수 있도록 구성하였다. 또한, 목적코드는 각 명령어간 자료종속성 정보를 포함하여 긴명령어를 구성하므로써 연산처리기가 이를 이용하여 다른 연산처리기와 동기 제어가 가능하도록 구성하였다.

EIS 프로세서 구조의 성능을 입증하기 위하여 네 가지 프로세서 모델을 모의하는 시뮬레이션 시스템을 구현하여 여러 가지 명령어 그래프를 토대로 실험하였다. 즉, 목적코드를 구성하는 명령어들의 정수명령어와 실수명령어의 분포 비율과 명령어간의 자료종속성 비율을 변화시키면서 명령어 그래프를 구성하고, 이를 각각의 구조별 목적코드로 변환하여 총 실행 사이클의 수를 측정하여 본 결과, EIS 프로세서가 기존의 세가지 모델을 구현한 VLIW, SVLIW 등의 프로세서에 비하여 총 실행 사이클의 수가 가장 적었음이 입증되었다.

참 고 문 헌

[1] Pohua P. Chang, Daniel M. Lavery, Scott A. Mahlke, William Y. Chen, and wen-mei W. Hwu, "The Importance of Prepass Code Scheduling for Superscalar and Superpipelined Processors," *IEEE Transactions on Computers,* Vol. 44, No. 3, March 1995.

[2] Shyh-Kwei Chen, W. Kent Fuchs and Wen-Mei W. Hwn, "An analytical approach to scheduling code for superscalar and VLIW architectures," *Proc. Inter. Conf. Para. Pro.,* pp. I258-I292, 1994.

[3] Jong-bok Lee and Wonyong Sung, "Perform-

ance Modeling of Superscalar Processors using Multiple Branch Prediction," *Journal of KISS,* Vol. 24, No. 5, May 1997.

[4] Dezso Sima, "Superscalar instruction issue," *IEEE Micro,* Vol. 17, No. 5, Sept/Oct 1997.

[5] Robert P. Colwell, Robert P. Nix, and etc "A VLIW architecture for a trace scheduling compiler," *Proc. Trans. Comp.,* Vol. 37, No. 8, pp. 967-979, August 1988.

[6] Thomas M. Conte and Sumedh W. Sathaye, "Dynamic Rescheduling: A technique for object code compatibility in VLIW architecture," *Proc. 28th Inter. Symp. Micro.,* 1995.

[7] Arthur Abnous, Roni Potasman and Alex Nicolau, "A percolation based VLIW architecture, "*Proc. Inter. Corf. Para. Pro.,* pp. I144-I148, 1991.

[8] Arthur Abnous and Nader Bagherzadeh, "Pipelining and bypassing in a VLIW processor," *Trans. Para. Dist. Sys.,* Vol. 5, No. 6, pp. 658-664, June 1994.

[9] Boyoun Jeong, Joongnam Jeon and Sukil Kim, "Performance Improvement of SVLIW Architectures by Removing LNOPs from an Object Code," *Journal of KIPS,* Vol. 4, No. 9, August 1997.

[10] Boyoun Jeong, *A Design of SVLIW Processor with Dynamic Resource Collision Remove Unit,* 忠北大學校 碩士學位 論文, 1997.

[11] Boyoun Jeong, Joongnam Jeon and Sukil Kim, "Design of VLIW architectures minimizing dynamic resource Collisions," *Journal KISS,* Vol. 24, No. 4, pp. 357-368, April 1997.

[12] SungHyun Jee, No-Kwang Park and Sukil Kim, "Performance analysis of caching instructions on SVLIW processor and VLIW processor," *Journal IEEE Korea Council,* Vol. 1, No. 1, December 1997.

[13] Sung-Hyun Jee, No-Kwang Park and Sukil Kim, "Performance evaluation of data dependence removable instruction pipelines," *Journal KISS,* Vol 26, No. 3, March 1999.

[14] Shusuke Okamoto and Masahiro Sowa, "Hybrid processor based on VLIW and PN-Superscalar," *PDPTA'96 Inter. Conf.,* pp. 623-632, 1996.

[15] NoKwang Park, Sunghyun jee and Sukil Kim, "Instruction Execution Performance Analysis for PASC Processor," Proceedings of the 10th KIPS., pp. 306-309, Dec. 1998.

[16] NoKwang Park, *A Design and Performance Evaluation of PASC Processor,* 忠北大學校 碩士學位 論文, 1999.

[17] Sung-Hyun Jee, No-Kwang Park, Joong-Nam Jeon and Sukil Kim, "PASC Processor Architecture for Enhanced Loop Execution," *Journal KIPS,* Vol 6, No. 5, pp1225-1240, May 1999.

[18] Sung-Hyun Jee, A Design of A Processor Architecture for Codes With Explicit Data Dependencies, 忠北大學校 博士學位 論文, 2000.

지 승 현(Sung-Hyun Jee)
천안외국어대 컴퓨터정보과 전임강사
(Dept. of Computer Science, Chungbuk National University)
<주관심 분야> 병렬처리 컴퓨터 구조, 병렬처리 알고리즘

전 중 남(Joong-Nam Jeon)
충북대학교 컴퓨터과학과 교수
(Dept. of Computer Science, Chungbuk National University)
<주관심 분야> 컴퓨터구조, 공장자동화, 병렬처리 컴퓨터 구조, 병렬처리 알고리즘

김 석 일(Suk-Il Kim)
충북대학교 컴퓨터과학과 교수
(Dept. of Computer Science, Chungbuk National University)
<주관심 분야> 병렬처리 컴퓨터 구조, 슈퍼컴퓨팅, 병렬처리 알고리즘, 시각 장애인 사용자 인터페이스 등임