(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2017-0097626
(43) 공개일자 2017년08월28일

(51) 국제특허분류(Int. Cl.)
G06F 9/30 (2017.01)
(52) CPC특허분류
G06F 9/30036 (2013.01)
G06F 9/30018 (2013.01)
(21) 출원번호 10-2017-7013808
(22) 출원일자(국제) 2015년11월23일
심사청구일자 없음
(85) 번역문제출일자 2017년05월22일
(86) 국제출원번호 PCT/US2015/062057
(87) 국제공개번호 WO 2016/105755
국제공개일자 2016년06월30일
(30) 우선권주장
14/581,289 2014년12월23일 미국(US)

(71) 출원인
인텔 코포레이션
미합중국 캘리포니아 95054 산타클라라 미션 칼리지 블러바드 2200
(72) 발명자
자, 아쉬시
미국 97229 오리건주 포틀랜드 노스웨스트 127번 테라스 5093
발렌타인, 로버트
이스라엘 36054 키리아트 티본 에이치에이 레초브 하드가니오트 33-5
울드-아메드-발, 엘무스타파
미국 85226 아리조나주 챈들러 웨스트 챈들러 블러바드 5000
(74) 대리인
양영준, 김연송, 백만기

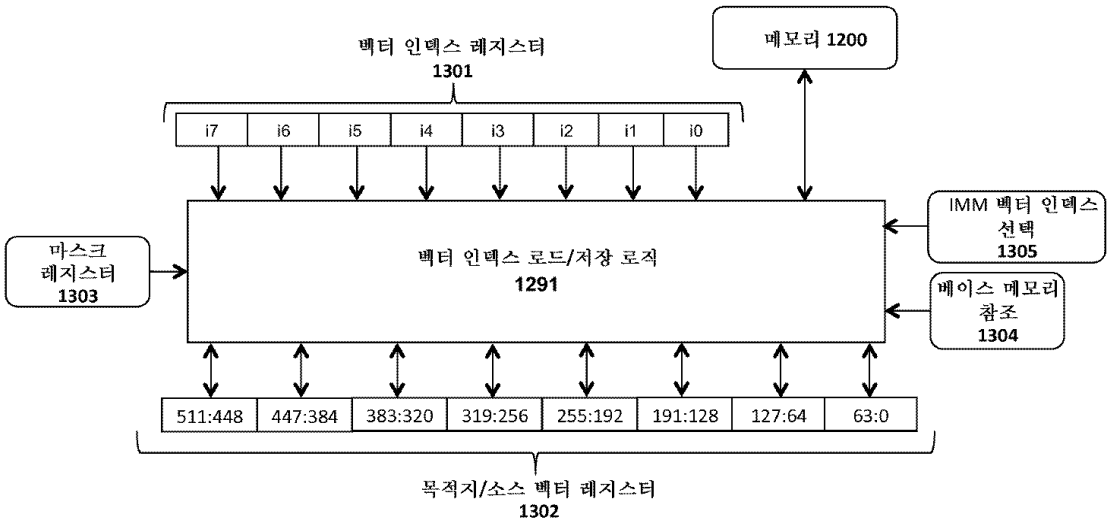
전체 청구항 수 : 총 27 항

(54) 발명의 명칭 **벡터 인덱스 로드 및 저장을 위한 방법 및 장치**

(57) 요 약

벡터 인덱스 로드 및 저장을 수행하기 위한 장치 및 방법. 예를 들어, 프로세서의 일 실시예는: 복수의 인덱스 값을 저장하는 벡터 인덱스 레지스터; 복수의 마스크 비트를 저장하는 마스크 레지스터; 메모리로부터 로드된 복수의 벡터 데이터 요소를 저장하는 벡터 레지스터; 및 로드 연산에 사용될 상기 벡터 인덱스 레지스터에 저장된 인덱스를 즉치 값을 사용하여 식별하고 이에 응답하여 상기 로드 연산을 위한 메모리 어드레스를 결정하기 위해 상기 인덱스를 베이스 메모리 어드레스와 결합하는 벡터 인덱스 로드 로직을 포함하고, 상기 벡터 인덱스 로드 로직은 상기 복수의 마스크 비트에 따라 상기 메모리 어드레스로부터의 벡터 데이터 요소들을 상기 벡터 레지스터로 로드하는 것이다.

대 표 도



(52) CPC특허분류
G06F 9/30101 (2013.01)

명 세 서

청구범위

청구항 1

프로세서로서,

복수의 인덱스 값을 저장하는 벡터 인덱스 레지스터;

복수의 마스크 비트를 저장하는 마스크 레지스터;

메모리로부터 로드된 복수의 벡터 데이터 요소를 저장하는 벡터 레지스터; 및

로드 연산에 사용될 상기 벡터 인덱스 레지스터에 저장된 인덱스를 즉치 값을 사용하여 식별하고 이에 응답하여 상기 로드 연산을 위한 메모리 어드레스를 결정하기 위해 상기 인덱스를 베이스 메모리 어드레스와 결합하는 벡터 인덱스 로드 로직을 포함하고, 상기 벡터 인덱스 로드 로직은 상기 복수의 마스크 비트에 따라 벡터 데이터 요소들을 상기 메모리 어드레스로부터 상기 벡터 레지스터로 로드하는 것인, 프로세서.

청구항 2

제1항에 있어서,

상기 복수의 마스크 비트는 상기 메모리 어드레스로부터 로드될 벡터 데이터 요소들의 수를 지시하는 것인, 프로세서.

청구항 3

제1항에 있어서,

상기 마스크 레지스터는 64 비트 마스크 레지스터를 포함하고 상기 벡터 인덱스 레지스터 및 벡터 레지스터는 8개의 64 비트 값을 갖는 512 비트 벡터 레지스터들을 포함하는, 프로세서.

청구항 4

제1항에 있어서,

상기 즉치 값은 상기 벡터 인덱스 레지스터 내의 인덱스에 대한 인덱스 위치를 식별하는 정수 값을 포함하는, 프로세서.

청구항 5

제1항에 있어서,

상기 인덱스 값들 각각을 계산하고 상기 인덱스 값들을 상기 벡터 인덱스 레지스터 내에 저장하는 메모리 관리 유닛을 추가로 포함하는, 프로세서.

청구항 6

제1항에 있어서,

상기 벡터 데이터 요소들은 배정밀도(double-precision) 부동 소수점 값들을 포함하는, 프로세서.

청구항 7

제1항에 있어서,

상기 벡터 데이터 요소들을 포함하는 데이터 어레이가 메모리에 저장되어야 하고 상기 베이스 메모리 어드레스는 상기 어레이의 베이스를 가리키고, 상기 벡터 인덱스 로드 로직은 상기 어레이 내의 상기 벡터 데이터 요소들 각각을 식별하기 위해 상기 베이스 메모리 어드레스를 상기 인덱스 값들 각각과 결합하는 것인, 프로세서.

청구항 8

제1항에 있어서,

인덱스를 식별하고, 상기 인덱스를 베이스 메모리 어드레스와 결합하고, 상기 벡터 데이터 요소들을 로드하는 것은 벡터 인덱스 로드 명령어의 디코딩 및 실행에 응답하여 수행되는, 프로세서.

청구항 9

제1항에 있어서,

상기 벡터 인덱스 로드 명령어는 복수의 마이크로 연산으로 디코딩되는, 프로세서.

청구항 10

프로세서로서,

복수의 인덱스 값을 저장하는 벡터 인덱스 레지스터;

복수의 마스크 비트를 저장하는 마스크 레지스터;

메모리에 저장될 복수의 벡터 데이터 요소를 저장하는 벡터 레지스터; 및

저장 연산에 사용될 상기 벡터 인덱스 레지스터에 저장된 인덱스를 즉치 값을 사용하여 식별하고 이에 응답하여 상기 저장 연산을 위한 메모리 어드레스를 결정하기 위해 상기 인덱스를 베이스 메모리 어드레스와 결합하는 벡터 인덱스 저장 로직을 포함하고, 상기 벡터 인덱스 로드 로직은 상기 복수의 마스크 비트에 따라 벡터 데이터 요소들을 상기 벡터 레지스터로부터 시스템 메모리 내의 상기 메모리 어드레스로 저장하는 것인, 프로세서.

청구항 11

제10항에 있어서,

상기 복수의 마스크 비트는 상기 메모리 어드레스로 저장될 벡터 데이터 요소들의 수를 지시하는 것인, 프로세서.

청구항 12

제10항에 있어서,

상기 마스크 레지스터는 64 비트 마스크 레지스터를 포함하고 상기 벡터 인덱스 레지스터 및 벡터 레지스터는 8개의 64 비트 값을 갖는 512 비트 벡터 레지스터들을 포함하는, 프로세서.

청구항 13

제10항에 있어서,

상기 즉치 값은 상기 벡터 인덱스 레지스터 내의 인덱스에 대한 인덱스 위치를 식별하는 정수 값을 포함하는, 프로세서.

청구항 14

제10항에 있어서,

상기 인덱스 값들 각각을 계산하고 상기 인덱스 값들을 상기 벡터 인덱스 레지스터 내에 저장하는 메모리 관리 유닛을 추가로 포함하는, 프로세서.

청구항 15

제10항에 있어서,

상기 벡터 데이터 요소들은 배정밀도 부동 소수점 값들을 포함하는, 프로세서.

청구항 16

제10항에 있어서,

상기 벡터 데이터 요소들을 포함하는 데이터 어레이가 메모리에 저장되어야 하고 상기 베이스 메모리 어드레스는 상기 어레이의 베이스를 가리키고, 상기 벡터 인덱스 저장 로직은 상기 어레이 내의 상기 벡터 데이터 요소들 각각을 저장할 위치를 식별하기 위해 상기 베이스 메모리 어드레스를 상기 인덱스 값들 각각과 결합하는 것인, 프로세서.

청구항 17

제10항에 있어서,

인덱스를 식별하고, 상기 인덱스를 베이스 메모리 어드레스와 결합하고, 상기 벡터 데이터 요소들을 저장하는 것은 벡터 인덱스 로드 명령어의 디코딩 및 실행에 응답하여 수행되는, 프로세서.

청구항 18

제17항에 있어서,

상기 벡터 인덱스 로드 명령어는 복수의 마이크로 연산으로 디코딩되는, 프로세서.

청구항 19

방법으로서,

복수의 인덱스 값을 벡터 인덱스 레지스터에 저장하는 단계;

복수의 마스크 비트를 마스크 레지스터에 저장하는 단계;

메모리로부터 로드된 복수의 벡터 데이터 요소를 저장하는 단계; 및

로드 연산에 사용될 상기 벡터 인덱스 레지스터에 저장된 인덱스를 즉치 값을 사용하여 식별하는 단계;

이에 응답하여 상기 로드 연산을 위한 메모리 어드레스를 결정하기 위해 상기 인덱스를 베이스 메모리 어드레스와 결합하는 단계; 및

상기 복수의 마스크 비트에 따라 벡터 데이터 요소들을 상기 메모리 어드레스로부터 상기 벡터 레지스터로 로드하는 단계를 포함하는, 방법.

청구항 20

제19항에 있어서,

상기 복수의 마스크 비트는 상기 메모리 어드레스로부터 로드될 벡터 데이터 요소들의 수를 지시하는 것인, 방법.

청구항 21

제19항에 있어서,

상기 마스크 레지스터는 64 비트 마스크 레지스터를 포함하고 상기 벡터 인덱스 레지스터 및 벡터 레지스터는 8개의 64 비트 값을 갖는 512 비트 벡터 레지스터들을 포함하는, 방법.

청구항 22

제19항에 있어서,

상기 즉치 값은 상기 벡터 인덱스 레지스터 내의 인덱스에 대한 인덱스 위치를 식별하는 정수 값을 포함하는, 방법.

청구항 23

제19항에 있어서,

상기 인덱스 값들 각각을 계산하고 상기 인덱스 값들을 상기 벡터 인덱스 레지스터 내에 저장하는 메모리 관리

유닛을 추가로 포함하는, 방법.

청구항 24

제19항에 있어서,

상기 벡터 데이터 요소들은 배정밀도 부동 소수점 값들을 포함하는, 방법.

청구항 25

제19항에 있어서,

상기 벡터 데이터 요소들을 포함하는 데이터 어레이를 메모리에 저장하는 단계 - 상기 베이스 메모리 어드레스는 상기 어레이의 베이스를 가리킴 -; 및

상기 어레이 내의 상기 벡터 데이터 요소들 각각을 식별하기 위해 상기 베이스 메모리 어드레스를 상기 인덱스 값들 각각과 결합하는 단계를 추가로 포함하는, 방법.

청구항 26

제19항에 있어서,

인덱스를 식별하는 단계, 상기 인덱스를 베이스 메모리 어드레스와 결합하는 단계, 및 상기 벡터 데이터 요소들을 로드하는 단계는 벡터 인덱스 로드 명령어의 디코딩 및 실행에 응답하여 수행되는, 방법.

청구항 27

제26항에 있어서,

상기 벡터 인덱스 로드 명령어는 복수의 마이크로 연산으로 디코딩되는, 방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 일반적으로 컴퓨터 프로세서의 분야에 관한 것이다. 특히, 본 발명은 벡터 인덱스 로드 및 저장을 위한 방법 및 장치에 관한 것이다.

배 경 기 술

[0002] 명령어 세트 또는 명령어 세트 아키텍처(ISA: instruction set architecture)는 고유 데이터 유형들, 명령어들, 레지스터 아키텍처, 어드레싱 모드들, 메모리 아키텍처, 인터럽트 및 예외 처리(exception handling), 및 외부 입력 및 출력(I/O)을 포함하여, 프로그래밍에 관련되는 컴퓨터 아키텍처의 일부이다. 본 명세서에서 "명령어(instruction)"란 용어는 일반적으로 실행을 위해 프로세서에 제공되는 명령어들인 매크로 명령어들을 말하며, 이는 매크로 명령어들을 디코드하는 프로세서의 디코더의 결과인 마이크로 명령어들 또는 마이크로연산들(micro-ops)과는 대조적이라는 점에 유의하여야 한다. 마이크로 명령어들 또는 마이크로연산들은 매크로 명령어와 관련된 로직을 구현하기 위한 연산을 수행하도록 프로세서 상의 실행 유닛에 지시하도록 구성될 수 있다.

[0003] ISA는 마이크로아키텍처와 구별되며, 이는 명령어 세트를 구현하는데 사용되는 프로세서 설계 기술들의 세트이다. 상이한 마이크로아키텍처들을 갖는 프로세서들은 공통의 명령어 세트를 공유할 수 있다. 예를 들어, Intel® Pentium 4 프로세서, Intel® Core™ 프로세서, 및 캘리포니아주 서니베일의 Advanced Micro Devices, Inc.의 프로세서는 (더 새로운 버전에 추가된 일부 확장을 갖는) x86 명령어 세트의 거의 동일한 버전을 구현하지만, 상이한 내부 설계를 갖는다. 예를 들어, 동일한 레지스터 아키텍처의 ISA가 공지된 기술들을 사용하는 상이한 마이크로아키텍처들에서 상이한 방식들로 구현될 수 있으며, 전용 물리적 레지스터들, 레지스터 리네이밍 메커니즘을 사용하는(예를 들어, RAT(Register Alias Table), ROB(Reorder Buffer) 및 리타이어먼트 레지스터 파일의 사용) 하나 이상의 동적으로 할당된 물리적 레지스터들을 포함한다. 달리 특정되지 않는 한, 레지스터 아키텍처, 레지스터 파일 및 레지스터라는 어구들은, 본 명세서에서, 소프트웨어/프로그래머에 가시화되는 것, 및 명령어들이 레지스터들을 지정하는 방식을 지칭하기 위해 이용된다. 구별이 필요한 경우에는, "논리적", "아키텍처의", 또는 "소프트웨어가 볼 수 있는(software visible)과 같은 형용사가 레지스터 아키텍처에서

의 레지스터들/파일들을 나타내는데 사용될 것인 반면, 주어진 마이크로아키텍처에서의 레지스터들을 지정하는데 상이한 형용사들이 사용될 것이다(예를 들어, 물리적 레지스터, 재정렬 버퍼, 리타이어먼트 레지스터, 레지스터 풀).

[0004] 명령어 세트는 하나 이상의 명령어 포맷을 포함한다. 주어진 명령어 포맷은, 다양한 필드들(비트 수, 비트들의 위치)을 정의하여, 다른 것들 중에서, 수행될 연산, 및 그 연산이 수행되어야 하는 피연산자(들)를 지정한다. 일부 명령어 포맷들은 명령어 템플릿들(또는 서브포맷들)의 정의를 통해 추가로 세분된다. 예를 들어, 주어진 명령어 포맷의 명령어 템플릿은 명령어 포맷의 필드의 상이한 서브세트를 갖도록 정의될 수 있고(포함된 필드는 전형적으로 동일 순서에 있지만, 적어도 일부는 더 적은 필드가 포함되기 때문에 상이한 비트 위치를 가짐)/있거나 상이하게 해석되는 주어진 필드를 갖도록 정의될 수 있다. 주어진 명령어는 주어진 명령어 포맷을 이용하여(그리고 정의되는 경우에 그 명령어 포맷의 명령어 템플릿들 중 주어진 명령어 템플릿으로) 표현되고, 연산 및 피연산자들을 지정한다. 명령어 스트림은 명령어들의 특정 시퀀스인데, 이 시퀀스에서의 각각의 명령어는 명령어 포맷의(그리고 정의되는 경우에 그 명령어 포맷의 명령어 템플릿들 중 주어진 명령어 템플릿의) 명령어의 출현이다.

도면의 간단한 설명

[0005] 아래의 도면과 관련된 아래의 상세한 설명으로부터 본 발명의 더 나은 이해가 얻어질 수 있다. 도면에서:

도 1a 및 도 1b는 본 발명의 실시예에 따른 일반적 벡터 친화적 명령어 포맷 및 그의 명령어 템플릿을 나타내는 블록도이다.

도 2a 내지 도 2d는 본 발명의 실시예에 따른 예시적인 특정적 벡터 친화적 명령어 포맷을 도시하는 블록도이다.

도 3은 본 발명의 일 실시예에 따른 레지스터 아키텍처의 블록도이다.

도 4a는 본 발명의 실시예에 따른 예시적인 순차적 페치, 디코드, 리타이어먼트 파이프라인 및 예시적인 레지스터 리네이밍 비순차적 발행/실행 파이프라인 양자를 도시하는 블록도이다.

도 4b는 본 발명의 실시예에 따른, 프로세서에 포함되는 순차적 페치, 디코드, 리타이어먼트 코어의 예시적인 실시예 및 예시적인 레지스터 리네이밍, 비순차적 발생/실행 아키텍처 코어 양자를 도시하는 블록도이다.

도 5a는 단일 프로세서 코어 및 온-다이 인터커넥트 네트워크에 대한 그의 접속의 블록도이다.

도 5b는 본 발명의 실시예에 따른 도 5a의 프로세서 코어의 일부의 확대도를 나타낸다.

도 6은 본 발명의 실시예에 따른 통합 메모리 제어기 및 그래픽을 갖는 단일 코어 프로세서 및 멀티 코어 프로세서의 블록도이다.

도 7은 본 발명의 일 실시예에 따른 시스템의 블록도를 나타낸다.

도 8은 본 발명의 실시예에 따른 제2 시스템의 블록도를 나타낸다.

도 9는 본 발명의 실시예에 따른 제3 시스템의 블록도를 나타낸다.

도 10은 본 발명의 실시예에 따른 시스템 온 칩(SoC)의 블록도를 나타낸다.

도 11은 본 발명의 실시예에 따른 소스 명령어 세트 내의 바이너리 명령어를 타겟 명령어 세트 내의 바이너리 명령어로 변환하기 위한 소프트웨어 명령어 변환기의 사용을 대조하는 블록도를 나타낸다.

도 12는 본 발명의 실시예가 구현될 수 있는 예시적인 프로세서를 나타낸다.

도 13은 본 발명의 일 실시예에 따른 벡터 인덱스 로드/저장 로직을 나타낸다.

도 14는 예시적인 즉치 및 마스크 값들에 기초하여 예시적인 인덱스들의 세트들을 처리하는 벡터 인덱스 로드/저장 로직을 나타낸다.

도 15는 본 발명의 일 실시예에 따라 벡터 인덱스 로드 연산을 수행하는 방법을 나타낸다.

도 16은 본 발명의 일 실시예에 따라 벡터 인덱스 저장 연산을 수행하는 방법을 나타낸다.

발명을 실시하기 위한 구체적인 내용

[0006] 아래의 설명에서는 설명의 목적으로, 후술하는 본 발명의 실시예들의 충분한 이해를 제공하기 위해 다수의 특정 상세가 설명된다. 그러나, 본 발명의 실시예들은 이러한 특정 상세들 중 일부 없이도 실시될 수 있다는 것이 본 기술분야의 통상의 기술자에게 명백할 것이다. 다른 예들에서는 본 발명의 실시예들의 기본 원리들을 불명확하게 하는 것을 방지하기 위해 공지 구조들 및 디바이스들은 블록도 형태로 도시된다.

[0007] 예시적인 프로세서 아키텍처 및 데이터 유형

[0008] 명령어 세트는 하나 이상의 명령어 포맷을 포함한다. 주어진 명령어 포맷은 다양한 필드들(비트 수, 비트들의 위치)을 정의하여, 다른 것들 중에서, 수행될 연산(opcode), 및 그 연산이 수행되어야 하는 피연산자(들)를 지정한다. 일부 명령어 포맷들은 명령어 템플릿들(또는 서브포맷들)의 정의를 통해 추가로 세분된다. 예를 들어, 주어진 명령어 포맷의 명령어 템플릿은 명령어 포맷의 필드의 상이한 서브세트를 갖도록 정의될 수 있고(포함된 필드는 전형적으로 동일 순서에 있지만, 적어도 일부는 더 적은 필드가 포함되기 때문에 상이한 비트 위치를 가짐)/있거나 상이하게 해석되는 주어진 필드를 갖도록 정의될 수 있다. 따라서, ISA의 각각의 명령어는 주어진 명령어 포맷을 이용하여(정의된 경우, 그 명령어 포맷의 명령어 템플릿들 중 주어진 템플릿에서) 표현되고, 연산 및 피연산자들을 지정하기 위한 필드들을 포함한다. 예를 들어, 예시적인 ADD 명령어는 특정 opcode, 및 그 opcode를 지정하는 opcode 필드 및 피연산자들(소스 1/목적지 및 소스 2)을 선택하는 피연산자 필드들을 포함하는 명령어 포맷을 갖고; 명령어 스트림에서 이러한 ADD 명령어의 출현(occurrence)은 특정 피연산자들을 선택하는 피연산자 필드들에서 특정 콘텐츠를 가질 것이다. 고급 벡터 확장(AVX)(AVX1 및 AVX2)으로서 지칭되고 벡터 확장(VEX) 코딩 스킴을 이용하는 SIMD 확장 세트가 존재했고, 발표되었고/되었거나 공개되었다(예로서, Intel® 64 and IA-32 Architectures Software Developers Manual, October 2011; 및 Intel® Advanced Vector Extensions Programming Reference, June 2011 참조).

[0009] **예시적인 명령어 포맷**

[0010] 본 명세서에 설명되는 명령어(들)의 실시예들은 상이한 포맷들로 구현될 수 있다. 추가적으로, 예시적인 시스템들, 아키텍처들, 및 파이프라인들이 아래에 상세하게 설명된다. 명령어(들)의 실시예들은 그러한 시스템들, 아키텍처들, 및 파이프라인들 상에서 실행될 수 있지만, 이들 상세에 한정되지는 않는다.

[0011] **A. 일반적 벡터 친화적 명령어 포맷**

[0012] 벡터 친화적 명령어 포맷은 벡터 명령어들에 적합한 명령어 포맷이다(예를 들어, 벡터 연산들에 특정적인 소정 필드들이 존재함). 벡터 및 스칼라 연산들 양쪽 모두가 벡터 친화적 명령어 포맷을 통해 지원되는 실시예들이 설명되지만, 대안적인 실시예들은 벡터 친화적 명령어 포맷의 벡터 연산들만을 사용한다.

[0013] 도 1a 및 도 1b는 본 발명의 실시예들에 따른 일반적 벡터 친화적 명령어 포맷 및 이것의 명령어 템플릿들을 도시하는 블록도들이다. 도 1a는 본 발명의 실시예들에 따른 일반적 벡터 친화적 명령어 포맷 및 이것의 클래스 A 명령어 템플릿들을 도시하는 블록도인 한편; 도 1b는 본 발명의 실시예들에 따른 일반적 벡터 친화적 명령어 포맷 및 이것의 클래스 B 명령어 템플릿들을 도시하는 블록도이다. 구체적으로, 일반적 벡터 친화적 명령어 포맷(100)에 대해 클래스 A 및 클래스 B 명령어 템플릿들이 정의되고, 이 양자는 메모리 액세스 없음(no memory access)(105) 명령어 템플릿들 및 메모리 액세스(120) 명령어 템플릿들을 포함한다. 벡터 친화적 명령어 포맷의 상황에서 일반적(generic)이라는 용어는 임의의 특정 명령어 세트에 얽매이지 않는 명령어 포맷을 지칭한다.

[0014] 벡터 친화적 명령어 포맷이 다음의 것을 지원하는 본 발명의 실시예들이 설명될 것이지만: 데이터 요소 폭들(또는 크기들)이 32 비트(4 바이트) 또는 64 비트(8 바이트)인 64 바이트 벡터 피연산자 길이(또는 크기)(따라서, 64 바이트 벡터는 16개의 더블워드-크기의 요소들 또는 대안적으로 8개의 쿼드워드-크기의 요소들로 구성됨); 데이터 요소 폭들(또는 크기들)이 16 비트(2 바이트) 또는 8 비트(1 바이트)인 64 바이트 벡터 피연산자 길이(또는 크기); 데이터 요소 폭들(또는 크기들)이 32 비트(4 바이트), 64 비트(8 바이트), 16 비트(2 바이트) 또는 8 비트(1 바이트)인 32 바이트 벡터 피연산자 길이(또는 크기); 및 데이터 요소 폭들(또는 크기들)이 32 비트(4 바이트), 64 비트(8 바이트), 16 비트(2 바이트) 또는 8 비트(1 바이트)인 16 바이트 벡터 피연산자 길이(또는 크기); 대안적인 실시예들은, 더 크거나, 더 작거나 또는 상이한 데이터 요소 폭들(예를 들어, 128 비트(16 바이트)의 데이터 요소 폭들)을 갖는 더 크거나, 더 작거나 그리고/또는 상이한 벡터 피연산자 크기들(예를 들어, 256 바이트 벡터 피연산자들)을 지원할 수 있다.

[0015] 도 1a의 클래스 A 명령어 템플릿들은: 1) 메모리 액세스 없음(105) 명령어 템플릿들 내에 메모리 액세스 없음, 풀 라운드 제어형 연산(110) 명령어 템플릿 및 메모리 액세스 없음, 데이터 변환형 연산(115) 명령어 템플릿이 도시되어 있고; 2) 메모리 액세스(120) 명령어 템플릿들 내에 메모리 액세스, 일시적(125) 명령어 템플릿 및 메

모리 액세스, 비일시적(130) 명령어 템플릿이 도시되는 것을 포함한다. 도 1b의 클래스 B 명령어 템플릿들은: 1) 메모리 액세스 없음(105) 명령어 템플릿들 내에 메모리 액세스 없음, 기입 마스크 제어, 부분 라운드 제어형 연산(112) 명령어 템플릿 및 메모리 액세스 없음, 기입 마스크 제어, vsize 유형 연산(117) 명령어 템플릿이 도시되어 있고; 2) 메모리 액세스(120) 명령어 템플릿들 내에 메모리 액세스, 기입 마스크 제어(127) 명령어 템플릿이 도시되어 있는 것을 포함한다.

[0016] 일반적 벡터 친화적 명령어 포맷(100)은 도 1a 및 도 1b에 도시된 순서로 아래 나열된 다음의 필드들을 포함한다.

[0017] 포맷 필드(140) - 이 필드 내의 특정 값(명령어 포맷 식별자 값)은 벡터 친화적 명령어 포맷, 및 따라서 명령어 스트림들 내의 벡터 친화적 명령어 포맷에서의 명령어들의 발생들을 고유하게 식별한다. 이와 같이, 이런 필드는 이것이 일반적 벡터 친화적 명령어 포맷만을 갖는 명령어 세트를 필요로 하지 않는다는 점에서 선택적이다.

[0018] 베이스 연산 필드(142) - 그의 내용은 상이한 베이스 연산들을 구별한다.

[0019] 레지스터 인덱스 필드(144) - 그의 내용은, 직접 또는 어드레스 생성을 통해, 그것들이 레지스터들 내에 있든지 메모리 내에 있든지, 소스 및 목적지 피연산자들의 위치들을 지정한다. 이들은 PxQ(예를 들어, 32x512, 16x128, 32x1024, 64x1024) 레지스터 파일로부터 N개의 레지스터를 선택하기에 충분한 비트 수를 포함한다. 일 실시예에서 N은 최대 3개의 소스 및 1개의 목적지 레지스터일 수 있지만, 대안적인 실시예들은 더 많거나 더 적은 소스들 및 목적지 레지스터들을 지원할 수 있다(예를 들어, 이러한 소스들 중 하나가 또한 목적지의 역할을 하는 경우에 최대 2개의 소스까지 지원할 수 있고, 이러한 소스들 중 하나가 또한 목적지의 역할을 하는 경우에 최대 3개의 소스를 지원할 수 있고, 최대 2개의 소스 및 1개의 목적지까지를 지원할 수 있다).

[0020] 변경자 필드(Modifier field)(146) - 그의 내용은 메모리 액세스하지 않는 것들로부터 메모리 액세스를 지정하는 일반적 벡터 명령어 포맷 내의 명령어들의 발생들을 구별하는데, 즉, 메모리 액세스 없음(105) 명령어 템플릿들과 메모리 액세스(120) 명령어 템플릿들 사이에서 구별한다. 메모리 액세스 연산들은(일부 경우에서 레지스터들 내의 값들을 사용하여 소스 및/또는 목적지 어드레스들을 지정하는) 메모리 계층구조에 대해 판독 및/또는 기입하는 반면에, 메모리 액세스 없음 연산들은 그렇게 하지 않는다(예를 들어, 소스 및 목적지들이 레지스터들임). 일 실시예에서 이 필드는 메모리 어드레스 계산들을 수행하는 3가지 상이한 방식들 사이에서 또한 선택하지만, 대안적인 실시예들은 메모리 어드레스 계산들을 수행하는 더 많거나, 더 적거나 또는 상이한 방식들을 지원할 수 있다.

[0021] 증강(Augmentation) 연산 필드(150) - 그의 내용은 베이스 연산 이외에 수행될 다양한 상이한 연산들 중 어느 하나를 구별한다. 이 필드는 상황에 고유하다. 본 발명의 일 실시예에서, 이 필드는 클래스 필드(168), 알파 필드(152), 및 베타 필드(154)로 분할된다. 증강 연산 필드(150)는 연산들의 공통 그룹들이 2, 3, 또는 4개의 명령어보다는 단일 명령어에서 수행될 수 있게 한다.

[0022] 스케일 필드(160) - 그의 내용은 메모리 어드레스 생성을 위한(예를 들어, 2^{scale} * index + base를 이용하는 어드레스 생성을 위한) 인덱스 필드의 내용의 스케일링(scaling)을 허용한다.

[0023] 변위 필드(162A) - 그의 내용은 (예를 들어, 2^{scale} * index + base + displacement를 이용하는 어드레스 생성을 위한) 메모리 어드레스 생성의 부분으로서 이용된다.

[0024] 변위 인자 필드(Displacement Factor Field)(162B)(변위 인자 필드(162B) 바로 위의 변위 필드(162A)의 병치(juxtaposition)는 하나 또는 다른 것이 이용됨을 나타낸다는 것에 주목한다) - 그의 내용은 어드레스 생성의 부분으로서 이용되고, 그것은 메모리 액세스의 크기(N)에 의해 스케일링될 변위 인자를 지정하며, 여기서 N은 (예를 들어, 2^{scale} * index + base + scaled displacement를 이용하는 어드레스 생성을 위한) 메모리 액세스에서의 바이트들의 수이다. 잉여 하위 비트들(redundant low-order bits)은 무시되고, 따라서 변위 인자 필드의 내용은 유효 어드레스를 계산하는데 이용될 최종 변위를 생성하기 위해서 메모리 피연산자 총 크기(N)로 승산된다. N의 값은 풀 opcode 필드(full opcode field; 174)(본 명세서에서 나중에 설명됨) 및 데이터 조작 필드(154C)에 기초하여 실행시간에 프로세서 하드웨어에 의해 결정된다. 변위 필드(162A) 및 변위 인자 필드(162B)는 그것들이 메모리 액세스 없음(105) 명령어 템플릿들을 위해 이용되지 않고 및/또는 상이한 실시예들은 둘 중 하나만 구현하거나 또는 아무것도 구현하지 않을 수 있다는 점에서 선택적이다.

[0025] 데이터 요소 폭 필드(164) - 그의 내용은 이용될 다수의 데이터 요소 폭들 중 하나를 구별한다(일부 실시예에서 모든 명령어들에 대해; 다른 실시예들에서 명령어들 중 일부만에 대해). 이 필드는, 단 하나의 데이터 요소 폭

만이 지원되고/되거나 데이터 요소 폭들이 opcode들의 일부 양태를 이용하여 지원되는 경우에 필요하지 않는다는 점에서 선택적이다.

[0026] 기입 마스크 필드(170) - 그의 내용은, 데이터 요소 위치 기초로, 목적지 벡터 피연산자 내의 그 데이터 요소 위치가 베이스 연산 및 증강 연산의 결과를 반영하는지를 제어한다. 클래스 A 명령어 템플릿들은 병합-기입마스킹(merging-writemasking)을 지원하는 반면에, 클래스 B 명령어 템플릿들은 병합-기입마스킹 및 제로화-기입마스킹(zeroing-writemasking) 양쪽 모두를 지원한다. 병합할 때에, 벡터 마스크들은 목적지 내의 임의의 세트의 요소들이(베이스 연산 및 증강 연산에 의해 특정되는) 임의의 연산의 실행 동안 업데이트들로부터 보호될 수 있게 해주고; 다른 일 실시예에서는, 대응하는 마스크 비트가 0을 갖는 경우에 목적지의 각각의 요소의 이전의 값을 보존할 수 있게 해준다. 이에 반해, 제로화할 때에, 벡터 마스크들은 목적지 내의 임의의 세트의 요소들이(베이스 연산 및 증강 연산에 의해 특정되는) 임의의 연산의 실행 동안 제로화될 수 있게 하고; 일 실시예에서는, 목적지의 요소는 대응하는 마스크 비트가 0 값을 가질 때에 0으로 설정된다. 이러한 기능성의 서브세트는 수행되는 연산의 벡터 길이를 제어하는 능력이지만(즉, 요소들의 범위(span)는 첫 번째 것으로부터 마지막 것까지 변경됨); 변경되는 요소들이 연속적인 것은 필요하지 않는다. 따라서, 기입 마스크 필드(170)는 로드, 저장, 산술, 로직 등을 포함한 부분 벡터 연산들을 허용한다. 기입 마스크 필드(170)의 내용이 이용될 기입 마스크를 포함하는 다수의 기입 마스크 레지스터들 중 하나를 선택하는(및 따라서 기입 마스크 필드(170)의 내용은 수행될 마스킹을 간접적으로 식별하는) 본 발명의 실시예들이 설명되지만, 대안적인 실시예들은 그 대신에 또는 추가적으로 마스크 기입 필드(170)의 내용이 수행될 마스킹을 직접 특정할 수 있게 한다.

[0027] 즉치 필드(172) - 그의 내용은 즉치의 명시(specification)를 허용한다. 이 필드는, 이것이 즉치를 지원하지 않는 일반적 벡터 친화적 포맷의 구현에 존재하지 않으며, 즉치를 이용하지 않는 명령어들에 존재하지 않는다는 점에서 선택적이다.

[0028] 클래스 필드(168) - 그의 내용은 명령어들의 상이한 클래스들 간을 구별한다. 도 1a 및 도 1b를 참조하면, 이 필드의 내용들은 클래스 A 및 클래스 B 명령어들 간을 선택한다. 도 1a 및 도 1b에서, 라운딩된 코너 정사각형들(rounded corner squares)을 이용하여 특정 값이 필드(예를 들어, 도 1a 및 도 1b에서 클래스 필드(168)에 대해 각각 클래스 A(168A) 및 클래스 B(168B))에 존재함을 나타낸다.

[0029] ***클래스 A의 명령어 템플릿***

[0030] 클래스 A의 메모리 액세스 없음(105) 명령어 템플릿들의 경우, 알파 필드(152)는 RS 필드(152A)로서 해석되고, 그 내용은 상이한 증강 연산 유형들 중 어느 것이 수행되어야 하는지를 구별하고(예를 들어, 라운드(152A.1) 및 데이터 변환(152A.2)은 각각 메모리 액세스 없음, 라운드 유형 연산(110) 및 메모리 액세스 없음, 데이터 변환형 연산(115) 명령어 템플릿들에 대해 지정되고), 베타 필드(154)는 지정된 유형의 연산들 중 어느 것이 수행되어야 하는지를 구별한다. 메모리 액세스 없음(105) 명령어 템플릿들에서, 스케일 필드(160), 변위 필드(162A), 및 변위 스케일 필드(162B)는 존재하지 않는다.

[0031] 메모리 액세스 없음 명령어 템플릿 - 풀 라운드 제어형 연산

[0032] 메모리 액세스 없음 풀 라운드 제어형 연산(110) 명령어 템플릿에서, 베타 필드(154)는 라운드 제어 필드(154A)로서 해석되고, 그 내용(들)은 정적 라운딩을 제공한다. 본 발명의 설명된 실시예들에서, 라운드 제어 필드(154A)는 SAE(suppress all floating point exceptions) 필드(156) 및 라운드 연산 제어 필드(158)를 포함하지만, 대안적인 실시예들은 이러한 개념들 양자를 동일한 필드에 인코딩하거나 오직 이러한 개념들/필드들 중 하나 또는 다른 하나만을 갖는 것(예를 들어, 오직 라운드 연산 제어 필드(158)를 가질 수 있다)을 지원할 수 있다.

[0033] SAE 필드(156) - 그의 내용은 예외 이벤트 보고를 디스에이블할 것인지 여부를 구별하고; SAE 필드(156)의 내용이 억제가 인에이블됨을 나타낼 때, 주어진 명령어는 임의의 종류의 부동 소수점 예외 플래그를 보고하지 않고, 임의의 부동 소수점 예외 핸들러를 발생시키지 않는다.

[0034] 라운드 연산 제어 필드(158) - 그의 내용은 수행할 라운딩 연산들(예를 들어, 라운드-업, 라운드-다운, 제로를 향해 라운드(Round-towards-zero) 및 근사치로 라운드(Round-to-nearest))의 그룹 중 하나를 구별한다. 따라서, 라운드 연산 제어 필드(158)는 명령어별로 라운딩 모드의 변경을 허용한다. 프로세서가 라운딩 모드들을 지정하기 위한 제어 레지스터를 포함하는 본 발명의 일 실시예에서, 라운드 연산 제어 필드(150)의 내용은 그 레지스터 값을 무효로 한다.

[0035] 메모리 액세스 없음 명령어 템플릿 - 데이터 변환형 연산

[0036] 메모리 액세스 없음 데이터 변환형 연산(115) 명령어 템플릿에서, 베타 필드(154)는 데이터 변환 필드(154B)로서 해석되고, 그 내용은 다수의 데이터 변환(예를 들어, 데이터 변환 없음, 스위즐(swizzle), 브로드캐스트) 중 어느 것이 수행되어야 하는지를 구별한다.

[0037] 클래스 A의 메모리 액세스(120) 명령어 템플릿의 경우에서, 알파 필드(152)는 축출 힌트 필드(152B)로서 해석되고, 그 내용은 이용될 축출 힌트들 중 하나를 구별하지만(도 1a에서, 일시적(152B.1) 및 비일시적(152B.2)이 각각 메모리 액세스, 일시적(125) 명령어 템플릿 및 메모리 액세스, 비일시적(130) 명령어 템플릿에 대해 특정된다), 베타 필드(154)는 데이터 조작 필드(154C)로서 해석되고, 그 내용은 수행될 다수의 데이터 조작 연산들(프리미티브들(primitives)이라고도 알려짐)(예를 들어, 조작 없음, 브로드캐스트, 소스의 상향 변환, 및 목적지의 하향 변환) 중 하나를 구별한다. 메모리 액세스(120) 명령어 템플릿들은 스케일 필드(160), 및 선택적으로 변위 필드(162A) 또는 변위 스케일 필드(162B)를 포함한다.

[0038] 벡터 메모리 명령어들은 변환 지원으로 메모리로부터의 벡터 로드들 및 메모리로의 벡터 스토어들을 수행한다. 정규 벡터 명령어들에서와 같이, 벡터 메모리 명령어들은 데이터 요소-관련 방식으로 메모리로부터/로 데이터를 전달하고, 실제로 전달되는 요소들은 기입 마스크로서 선택되는 벡터 마스크의 내용에 의해 지시된다.

[0039] 메모리 액세스 명령어 템플릿 - 일시적

[0040] 일시적 데이터는 캐싱으로부터 이익을 얻기에 충분할 만큼 빨리 재사용될 가능성이 있는 데이터이다. 그러나 이것은 힌트이며, 상이한 프로세서는 힌트를 완전히 무시하는 것을 포함하는 상이한 방식으로 그것을 구현할 수 있다.

[0041] 메모리 액세스 명령어 템플릿 - 비일시적

[0042] 비일시적 데이터는 제1 레벨 캐시 내의 캐싱으로부터 이익을 얻기에 충분할 만큼 빨리 재사용될 가능성이 없는 데이터이고, 축출에 대한 우선순위가 주어져야 한다. 그러나 이것은 힌트이고, 상이한 프로세서는 힌트를 완전히 무시하는 것을 포함하는 상이한 방식으로 그것을 구현할 수 있다.

[0043] ***클래스 B의 명령어 템플릿***

[0044] 클래스 B의 명령어 템플릿의 경우에, 알파 필드(152)는 기입 마스크 제어(Z) 필드(152C)로서 해석되고, 그 내용은 기입 마스크 필드(170)에 의해 제어된 기입 마스킹이 병합 또는 제로화이어야 하는지를 구별한다.

[0045] 클래스 B의 메모리 액세스 없음(105) 명령어 템플릿들의 경우에, 베타 필드(154)의 부분은 RL 필드(157A)로서 해석되고, 그 내용은 수행될 상이한 증강 연산 유형들 중 하나를 구별하지만(예를 들어, 라운드(157A.1) 및 벡터 길이(VSIZE)(157A.2)는 각각 메모리 액세스 없음, 기입 마스크 제어, 부분 라운드 제어형 연산(112) 명령어 템플릿 및 메모리 액세스 없음, 기입 마스크 제어, VSIZE 유형 연산(117) 명령어 템플릿에 대해 특정된다), 베타 필드(154)의 나머지는 수행될 특정된 유형의 연산들 중 어느 하나를 구별한다. 메모리 액세스 없음(105) 명령어 템플릿들에서, 스케일 필드(160), 변위 필드(162A), 및 변위 스케일 필드(162B)는 존재하지 않는다.

[0046] 메모리 액세스 없음, 기입 마스크 제어, 부분 라운드 제어형 연산(110) 명령어 템플릿에서, 베타 필드(154)의 나머지는 라운드 연산 필드(159A)로서 해석되고, 예외 이벤트 보고는 디스에이블된다(주어진 명령어는 임의의 종류의 부동 소수점 예외 플래그를 보고하지 않고, 임의의 부동 소수점 예외 핸들러를 발생시키지 않는다).

[0047] 라운드 연산 제어 필드(159A) - 단지 라운드 연산 제어 필드(158)로서, 그의 내용은 수행될 라운딩 연산들(예를 들어, 라운드-업, 라운드-다운, 제로를 향해 라운드 및 근사치로 라운드)의 그룹 중 하나를 구별한다. 따라서, 라운드 연산 제어 필드(159A)는 명령어별로 라운딩 모드의 변경을 허용한다. 프로세서가 라운딩 모드들을 지정하기 위한 제어 레지스터를 포함하는 본 발명의 일 실시예에서, 라운드 연산 제어 필드(150)의 내용은 그 레지스터 값을 무효로 한다.

[0048] 메모리 액세스 없음, 기입 마스크 제어, VSIZE 유형 연산(117) 명령어 템플릿에서, 베타 필드(154)의 나머지는 벡터 길이 필드(159B)로서 해석되고, 그 내용은 수행될 다수의 데이터 벡터 길이들(예를 들어, 128, 256, 또는 512 바이트) 중 하나를 구별한다.

[0049] 클래스 B의 메모리 액세스(120) 명령어 템플릿의 경우에, 베타 필드(154)의 부분은 브로드캐스트 필드(157B)로서 해석되고, 그 내용은 브로드캐스트 유형 데이터 조작 연산이 수행될 것인지 여부를 구별하지만, 베타 필드(154)의 나머지는 벡터 길이 필드(159B)로서 해석된다. 메모리 액세스(120) 명령어 템플릿들은 스케일 필드

(160), 및 선택적으로 변위 필드(162A) 또는 변위 스케일 필드(162B)를 포함한다.

[0050] 일반적 벡터 친화적 명령어 포맷(100)과 관련하여, 포맷 필드(140), 베이스 연산 필드(142), 및 데이터 요소 폭 필드(164)를 포함하는 풀 opcode 필드(174)가 도시된다. 풀 opcode 필드(174)가 이들 필드들 전부를 포함하는 일 실시예가 도시되지만, 풀 opcode 필드(174)는 그것들 전부를 지원하지 않는 실시예들에 있어서 이들 필드들 전부보다 적게 포함한다. 풀 opcode 필드(174)는 연산 코드(operation code)(opcode)를 제공한다.

[0051] 증강 연산 필드(150), 데이터 요소 폭 필드(164), 및 기입 마스크 필드(170)는 이러한 특징들이 일반적 벡터 친화적 명령어 포맷에서 명령어별로 특정될 수 있게 한다.

[0052] 기입 마스크 필드와 데이터 요소 폭 필드의 조합들은, 마스크가 상이한 데이터 요소 폭들에 기초하여 적용되는 것을 그것들이 허용한다는 점에서 타이핑된 명령어들(typed instructions)을 생성한다.

[0053] 클래스 A 및 클래스 B 내에서 발견되는 다양한 명령어 템플릿들은 상이한 상황들에서 이롭다. 본 발명의 일부 실시예에서, 상이한 프로세서들 또는 프로세서 내의 상이한 코어들은 클래스 A만을, 클래스 B만을, 또는 양자의 클래스들을 지원할 수 있다. 예를 들어, 범용 컴퓨팅에 대해 의도된 고성능 범용 비순차적 코어는 클래스 B만을 지원할 수 있고, 주로 그래픽 및/또는 과학적(쓰루풋) 컴퓨팅에 대해 의도된 코어는 클래스 A만을 지원할 수 있고, 양쪽 모두를 위해 의도된 코어는 양쪽 모두를 지원할 수 있다(물론, 양자의 클래스들로부터의 명령어들 및 템플릿들의 소정의 혼합을 갖지만 양자의 클래스들로부터의 명령어들 및 템플릿들 전부를 갖지는 않는 코어는 본 발명의 범위 내에 있다). 또한, 단일 프로세서가 복수의 코어를 포함할 수 있는데, 이들 모두는 동일한 클래스를 지원하거나 또는 상이한 코어들이 상이한 클래스를 지원한다. 예를 들어, 별개의 그래픽 및 범용 코어들을 갖는 프로세서에서, 주로 그래픽 및/또는 과학적 컴퓨팅에 대해 의도된 그래픽 코어들 중 하나는 클래스 A만을 지원할 수 있는 반면에, 범용 코어들 중 하나 이상은, 클래스 B만을 지원하는, 범용 컴퓨팅에 대해 의도된 비순차적 실행 및 레지스터 리네이밍을 갖는 고성능 범용 코어들일 수 있다. 별개의 그래픽 코어를 갖지 않는 다른 프로세서는 클래스 A 및 클래스 B 양쪽 모두를 지원하는 하나 이상의 범용 순차적 또는 비순차적 코어를 포함할 수 있다. 물론, 하나의 클래스로부터의 피처들은 본 발명의 상이한 실시예들에 있어서 다른 클래스에서 또한 구현될 수 있다. 하이 레벨 언어로 작성되는 프로그램은 1) 실행을 위한 타겟 프로세서에 의해 지원되는 클래스(들)의 명령어만을 갖는 형태; 또는 2) 모든 클래스의 명령어의 상이한 조합을 이용하여 작성된 대안 루틴을 갖고, 현재 코드를 실행하고 있는 프로세서에 의해 지원되는 명령어에 기초하여 실행할 루틴을 선택하는 제어 흐름 코드를 갖는 형태를 포함하는 다양한 상이한 실행 가능 형태가 될 것이다(예로서, 적시(just in time) 컴파일링 또는 정적 컴파일링될 것이다).

[0054] **B. 예시적인 특정적 벡터 친화적 명령어 포맷**

[0055] 도 2는 본 발명의 실시예에 따른 예시적인 특정적 벡터 친화적 명령어 포맷을 나타내는 블록도이다. 도 2는 필드들의 위치, 크기, 해석 및 순서뿐만 아니라, 이들 필드의 일부에 대한 값들을 지정한다는 점에서 특정적인 특정적 벡터 친화적 명령어 포맷(200)을 도시한다. 특정적 벡터 친화적 명령어 포맷(200)은 x86 명령어 세트를 확장하는 데 이용될 수 있고, 따라서 필드들 중 일부는 기존의 x86 명령어 세트 및 그의 확장(예를 들어, AVX)에서 이용된 것들과 유사하거나 동일하다. 이 포맷은 확장들을 갖는 기존의 x86 명령어 세트의 프리픽스 인코딩 필드, 실제 opcode 바이트 필드(real opcode byte field), MOD R/M 필드, SIB 필드, 변위 필드 및 즉치 필드들과 일관되게 유지된다. 도 2로부터의 필드들이 매핑하는 도 1으로부터의 필드들이 예시된다.

[0056] 본 발명의 실시예들은 예시의 목적으로 일반적 벡터 친화적 명령어 포맷(100)의 문맥에서 특정적 벡터 친화적 명령어 포맷(200)을 참조하여 설명되지만, 본 발명은 청구되는 경우를 제외하고 특정적 벡터 친화적 명령어 포맷(200)으로 한정되지 않는다는 것을 이해해야 한다. 예를 들어, 일반적 벡터 친화적 명령어 포맷(100)은 다양한 필드에 대한 다양한 가능한 크기들을 고려하지만, 특정적 벡터 친화적 명령어 포맷(200)은 특정 크기들의 필드들을 갖는 것으로서 도시된다. 특정 예에 의해, 데이터 요소 폭 필드(164)는 특정적 벡터 친화적 명령어 포맷(200)에서 1 비트 필드로서 도시되지만, 본 발명은 그것으로 한정되지 않는다(즉, 일반적 벡터 친화적 명령어 포맷(100)은 데이터 요소 폭 필드(164)의 다른 크기들을 고려한다).

[0057] 일반적 벡터 친화적 명령어 포맷(100)은 도 2a에 도시된 순서로 아래에 나열된 다음의 필드들을 포함한다.

[0058] EVEX 프리픽스(바이트 0-3)(202) - 4 바이트 형태로 인코딩된다.

[0059] 포맷 필드(140)(EVEX 바이트 0, 비트들 [7:0]) - 제1 바이트(EVEX 바이트 0)는 포맷 필드(140)이고, 그것은 0x62(본 발명의 일 실시예에서 벡터 친화적 명령어 포맷을 구별하는 데 이용되는 고유 값)을 포함한다.

[0060] 제2 내지 제4 바이트(EVEX 바이트 1-3)는 특정 능력을 제공하는 다수의 비트 필드를 포함한다.

[0061] REX 필드(205)(EVEX 바이트 1, 비트 [7-5]) - EVEX.R 비트 필드(EVEX 바이트 1, 비트 [7]-R), EVEX.X 비트 필드(EVEX 바이트 1, 비트 [6]-X), 및 157BEX 바이트 1, 비트 [5]-B로 이루어진다. EVEX.R, EVEX.X 및 EVEX.B 비트 필드들은 대응하는 VEX 비트 필드들과 동일한 기능성을 제공하고, 1의 보수 형태(1s complement form)를 이용하여 인코딩되는데, 즉 ZMM0은 1111B로 인코딩되고, ZMM15는 0000B로 인코딩된다. 명령어들의 다른 필드들은 관련 기술분야에 공지된 바와 같이 레지스터 인덱스들의 하위 3 비트를 인코딩하여(rrr, xxx, 및 bbb), EVEX.R, EVEX.X 및 EVEX.B를 추가함으로써 Rrrr, Xxxx, 및 Bbbb가 형성될 수 있다.

[0062] REX' 필드(110) - 이것은 REX' 필드(110)의 제1 부분이고, 확장된 32개의 레지스터 세트의 상위 16 또는 하위 16을 인코딩하는 데 이용되는 EVEX.R' 비트 필드(EVEX 바이트 1, 비트 [4]-R')이다. 본 발명의 일 실시예에서, 이 비트는, 아래에 표시되는 바와 같은 다른 것들과 함께, (잘 알려진 x86 32-비트 모드에서) BOUND 명령어와 구분하기 위해 비트 반전된 포맷으로 저장되고, 그것의 실제 opcode 바이트는 62이지만, (후술되는) MOD R/M 필드에서 MOD 필드 내의 11의 값을 수락하지 않으며; 본 발명의 대안적인 실시예들은 반전된 포맷으로 이것 및 아래에 표시되는 다른 비트들을 저장하지 않는다. 하위 16개의 레지스터를 인코딩하는 데 1의 값이 이용된다. 다시 말해서, R'Rrrr는 다른 필드들로부터의 EVEX.R', EVEX.R, 및 다른 RRR를 결합시킴으로써 형성된다.

[0063] opcode 맵 필드(215)(EVEX 바이트 1, 비트[3:0] - mmmm) - 그의 내용은 암시적인 선단 opcode 바이트(implied leading opcode byte)(0F, 0F 38 또는 0F 3)를 인코딩한다.

[0064] 데이터 요소 폭 필드(164)(EVEX 바이트 2, 비트 [7]-W) - 표기법 EVEX.W에 의해 표현된다. EVEX.W는 데이터형(32 비트 데이터 요소 또는 64비트 데이터 요소)의 입도(크기)를 정의하는 데 사용된다.

[0065] EVEX.vvvv(220)(EVEX 바이트 2, 비트 [6:3]-vvvv) - EVEX.vvvv의 역할은 다음을 포함할 수 있는데, 즉 1) EVEX.vvvv는 반전된 (1의 보수) 형태로 지정된 제1 소스 레지스터 피연산자를 인코딩하고, 2개 이상의 소스 피연산자를 갖는 명령어에 대해 유효하거나; 2) EVEX.vvvv는 특정 벡터 시프트를 위해 1의 보수 형태로 지정된 목적지 레지스터 피연산자를 인코딩하거나; 3) EVEX.vvvv는 어떠한 피연산자도 인코딩하지 않고, 필드는 예약되고, 1111b를 포함해야 한다. 따라서, EVEX.vvvv 필드(220)는 반전된 (1의 보수) 형태로 저장되는 제1 소스 레지스터 지정자의 4개의 낮은 순서 비트를 인코딩한다. 명령어에 따라, 추가의 상이한 EVEX 비트 필드가 지정자 크기를 32개의 레지스터로 확장하기 위해 이용된다.

[0066] EVEX.U 클래스 필드(168)(EVEX 바이트 2, 비트 [2]-U)-EVEX.U = 0이면, 그것은 클래스 A 또는 EVEX.U0을 나타내고, EVEX.U = 1이면, 그것은 클래스 B 또는 EVEX.U1을 나타낸다.

[0067] 프리픽스 인코딩 필드(225)(EVEX 바이트 2, 비트[1:0]-pp) - 베이스 연산 필드에 대한 추가적인 비트들을 제공한다. EVEX 프리픽스 포맷의 레거시 SSE 명령어들에 대한 지원을 제공하는 것에 외에, 이것은 또한 SIMD 프리픽스를 간소화하는 이득을 갖는다(SIMD 프리픽스를 표현하기 위해 바이트를 요구하는 것이 아니라, EVEX 프리픽스는 2 비트만을 요구함). 일 실시예에서, 레거시 포맷 및 EVEX 프리픽스 포맷 양자에서 SIMD 프리픽스(66H, F2H, F3H)를 이용하는 레거시 SSE 명령어를 지원하기 위해, 이들 레거시 SIMD 프리픽스는 SIMD 프리픽스 인코딩 필드에 인코딩되고; 런타임에서 디코더의 PLA에 제공되기 전에 레거시 SIMD 프리픽스 내로 확장된다(그래서, PLA는 변경 없이 레거시와, 이들 레거시 명령어의 EVEX 포맷 양자를 실행할 수 있다). 더 새로운 명령어들이 opcode 확장으로서 직접 EVEX 프리픽스 인코딩 필드의 내용을 이용할 수 있지만, 소정 실시예들은 일관성을 위해 유사한 방식으로 확장되고, 오히려 상이한 의미들이 이들 레거시 SIMD 프리픽스들에 의해 특정되는 것을 허용한다. 대안적인 실시예는 2 비트 SIMD 프리픽스 인코딩들을 지원하도록 PLA를 재설계할 수 있고, 따라서 확장을 요구하지 않는다.

[0068] 알파 필드(152)(EVEX 바이트 3, 비트[7] - EH; EVEX.EH, EVEX.rs, EVEX.RL, EVEX.기입 마스크 제어, 및 EVEX.N으로도 알려짐; 또한 α로 예시됨) - 앞서 설명된 바와 같이, 이 필드는 상황에 고유하다.

[0069] 베타 필드(154)(EVEX 바이트 3, 비트들[6:4]-SSS, EVEX.s_{2-0}, EVEX.r_{2-0}, EVEX.rr1, EVEX.LL0, EVEX.LLB라고도 함; 또한 βββ로 예시되어 있음) - 앞서 설명된 바와 같이, 이 필드는 상황에 고유하다.

[0070] REX' 필드(110) - 이것은 REX' 필드의 나머지이고, 확장된 32 레지스터 세트의 상위 16 또는 하위 16 중 어느 하나를 인코딩하는 데 이용될 수 있는 EVEX.V' 비트 필드(EVEX 바이트 3, 비트 [3]-V')이다. 이 비트는 비트 반전된 포맷으로 저장된다. 하위 16개의 레지스터를 인코딩하기 위해 1의 값이 이용된다. 다시 말해서, V'VVVV는 EVEX.V', EVEX.vvvv를 결합함으로써 형성된다.

[0071] 기입 마스크 필드(170)(EVEX 바이트 3, 비트들 [2:0]-kkk) - 그의 내용은 전술한 바와 같은 기입 마스크 레지스터들에 레지스터의 인덱스를 지정한다. 본 발명의 일 실시예에서, 특정 값 EVEX.kkk=000은 특정 명령어에 대해 어떤 기입 마스크도 이용되지 않음을 암시하는 특정한 거동을 갖는다(이것은 모든 것들에 하드와이어드된 기입 마스크 또는 마스킹 하드웨어를 바이패스하는 하드웨어의 이용을 포함하는 각종 방식들로 구현될 수 있음).

[0072] 실제 opcode 필드(230)(바이트 4)는 또한 opcode 바이트로 알려진다. opcode의 일부는 이 필드에서 특정된다.

[0073] MOD R/M 필드(240)(바이트 5)는 MOD 필드(242), Reg 필드(244), 및 R/M 필드(246)를 포함한다. 전술한 바와 같이, MOD 필드(242)의 내용은 메모리 액세스와 메모리 액세스 없음 연산들 사이를 구별한다. Reg 필드(244)의 역할은 두 가지 상황으로 요약될 수 있는데, 즉 목적지 레지스터 피연산자 또는 소스 레지스터 피연산자를 인코딩하거나, opcode 확장으로서 간주되고, 임의의 명령어 피연산자를 인코딩하는 데 사용되지 않는다. R/M 필드(246)의 역할은 메모리 어드레스를 참조하는 명령어 피연산자를 인코딩하거나, 목적지 레지스터 피연산자 또는 소스 레지스터 피연산자를 인코딩하는 것을 포함할 수 있다.

[0074] SIB(Scale, Index, Base) 바이트(바이트 6) - 전술한 바와 같이, 스케일 필드(150)의 내용은 메모리 어드레스 생성을 위해 이용된다. SIB.xxx(254) 및 SIB.bbb(256) - 이 필드들의 내용들은 레지스터 인덱스들 Xxxx 및 Bbbb과 관련하여 앞서 언급하였다.

[0075] 변위 필드(162A)(바이트들 7-10) - MOD 필드(242)가 10을 포함할 때, 바이트들 7-10은 변위 필드(162A)이고, 그것은 레거시 32-비트 변위(disp32)와 동일하게 작용하고, 바이트 입도에서 작용한다.

[0076] 변위 인자 필드(162B)(바이트 7) - MOD 필드(242)가 01을 포함할 때, 바이트 7은 변위 인자 필드(162B)이다. 이 필드의 위치는 바이트 입도로 작용하는 레거시 x86 명령어 세트 8 비트 변위(disp8)의 위치와 동일하다. disp8이 부호 확장되기(sign extended) 때문에, 이것은 단지 -128과 127바이트 오프셋들 사이를 어드레싱할 수 있고; 64바이트 캐시 라인들에 관하여, disp8은 4개의 실제 유용한 값들인 -128, -64, 0, 64로만 설정될 수 있는 8 비트를 이용하며; 더 큰 범위가 종종 필요하기 때문에, disp32가 이용되지만; disp32는 4바이트를 요구한다. disp8 및 disp32와 반대로, 변위 인자 필드(162B)는 disp8의 재해석이고; 변위 인자 필드(162B)를 이용할 때, 실제 변위는 메모리 피연산자 액세스의 크기(N)로 곱해진 변위 인자 필드의 내용에 의해 결정된다. 이러한 유형의 변위는 disp8*N으로 지칭된다. 이것은 평균 명령어 길이를 감소시킨다(단일 바이트가 그 변위에 사용되지만 훨씬 더 큰 범위를 갖는다). 이러한 압축된 변위는, 유효 변위가 메모리 액세스의 입도의 배수이고, 그에 따라 어드레스 오프셋의 잉여 하위 비트들이 인코딩될 필요가 없다는 가정에 기초한다. 다시 말해, 변위 인자 필드(162B)는 레거시 x86 명령어 세트 8-비트 변위를 대체한다. 따라서, 변위 인자 필드(162B)는 disp8이 disp8*N으로 오버로드된다는 것만 제외하고 x86 명령어 세트 8 비트 변위와 동일한 방식으로 인코딩된다(그래서 ModRM/SIB 인코딩 규칙들에서 어떠한 것도 변하지 않는다). 다시 말하면, 인코딩 규칙들 또는 인코딩 길이들에서 어떤 변경도 존재하지 않지만, (바이트-관련 어드레스 오프셋(byte-wise address offset)을 획득하기 위해 메모리 피연산자의 크기에 의해 변위를 스케일링할 필요가 있는) 하드웨어에 의한 변위 값의 해석에서만 변경이 존재한다.

[0077] 즉치 필드(172)는 전술한 바와 같이 동작한다.

[0078] ***풀 opcode 필드***

[0079] 도 2b는 본 발명의 일 실시예에 따른 풀 opcode 필드(174)를 구성하는 특정적 벡터 친화적 명령어 포맷(200)의 필드들을 도시하는 블록도이다. 구체적으로, 풀 opcode 필드(174)는 포맷 필드(140), 베이스 연산 필드(142), 및 데이터 요소 폭(W) 필드(164)를 포함한다. 베이스 연산 필드(142)는 프리픽스 인코딩 필드(225), opcode 맵 필드(215), 및 실제 opcode 필드(230)를 포함한다.

[0080] ***레지스터 인덱스 필드***

[0081] 도 2c는 본 발명의 일 실시예에 따른 레지스터 인덱스 필드(144)를 구성하는 특정적 벡터 친화적 명령어 포맷(200)의 필드들을 도시하는 블록도이다. 구체적으로, 레지스터 인덱스 필드(144)는 REX 필드(205), REX' 필드(210), MODR/M.reg 필드(244), MODR/M.r/m 필드(246), VVVV 필드(220), xxx 필드(254), 및 bbb 필드(256)를 포함한다.

[0082] ***증강 연산 필드***

[0083] 도 2d는 본 발명의 일 실시예에 따른 증강 연산 필드(150)를 구성하는 특정적 벡터 친화적 명령어 포맷(200)의 필드들을 도시하는 블록도이다. 클래스(U) 필드(168)가 0을 포함할 때, 그것은 EVEX.U0(클래스 A(168A))을 의

미하고; 그것이 1을 포함할 때, 그것은 EVEX.U1(클래스 B(168B))를 의미한다. U=0이고 MOD 필드(242)가 11을 포함할 때(메모리 액세스 없음 연산을 의미함), 알파 필드(152)(EVEX 바이트 3, 비트 [7]-EH)는 rs 필드(152A)로서 해석된다. rs 필드(152A)가 1을 포함할 때(라운드 152A.1), 베타 필드(154)(EVEX 바이트 3, 비트들 [6:4]- SSS)는 라운드 제어 필드(154A)로서 해석된다. 라운드 제어 필드(154A)는 1 비트 SAE 필드(156) 및 2 비트 라운드 연산 필드(158)를 포함한다. rs 필드(152A)가 0을 포함할 때(데이터 변환 152A.2), 베타 필드(154)(EVEX 바이트 3, 비트들 [6:4]- SSS)는 3 비트 데이터 변환 필드(154B)로서 해석된다. U=0이고 MOD 필드(242)가 00, 01, 또는 10을 포함할 때(메모리 액세스 연산을 의미함), 알파 필드(152)(EVEX 바이트 3, 비트 [7]-EH)는 축출 힌트(EH) 필드(152B)로서 해석되고, 베타 필드(154)(EVEX 바이트 3, 비트들 [6:4]- SSS)는 3 비트 데이터 조작 필드(154C)로서 해석된다.

[0084] U=1일 때, 알파 필드(152)(EVEX 바이트 3, 비트 [7]-EH)는 기입 마스크 제어(Z) 필드(152C)로서 해석된다. U=1이고 MOD 필드(242)가 11을 포함할 때(메모리 액세스 없음 연산을 의미함), 베타 필드(154)의 부분(EVEX 바이트 3, 비트 [4]- S_0)은 RL 필드(157A)로서 해석되고; 그것이 1을 포함할 때(라운드 157A.1), 베타 필드(154)의 나머지(EVEX 바이트 3, 비트 [6-5]-S_{2-1})는 라운드 연산 필드(159A)로서 해석되고, RL 필드(157A)가 0을 포함할 때(VSIZE 157.A2), 베타 필드(154)의 나머지(EVEX 바이트 3, 비트 [6-5]-S_{2-1})는 벡터 길이 필드(159B)(EVEX 바이트 3, 비트 [6-5]-L_{1-0})로서 해석된다. U=1이고 MOD 필드(242)가 00, 01, 또는 10을 포함할 때(메모리 액세스 연산을 의미함), 베타 필드(154)(EVEX 바이트 3, 비트들 [6:4]-SSS)는 벡터 길이 필드(159B)(EVEX 바이트 3, 비트 [6-5]-L_{1-0}) 및 브로드캐스트 필드(157B)(EVEX 바이트 3, 비트 [4]-B)로서 해석된다.

[0085] C. 예시적인 레지스터 아키텍처

[0086] 도 3은 본 발명의 일 실시예에 따른 레지스터 아키텍처(300)의 블록도이다. 도시된 실시예에서, 폭이 512 비트인 32개의 벡터 레지스터들(310)이 존재하고; 이들 레지스터들은 zmm0 내지 zmm31로서 참조된다. 하위 16개의 zmm 레지스터들의 하위 256 비트는 레지스터들 ymm0-16에 오버레이된다. 하위 16개의 zmm 레지스터들의 하위 128 비트(ymm 레지스터들의 하위 128 비트)는 레지스터들 xmm0-15에 오버레이된다. 특정적 벡터 친화적 명령어 포맷(200)은 아래 표에 예시된 바와 같이 이들 오버레이된 레지스터 파일에 대해 동작한다.

[0087]

조정가능 벡터 길이	클래스	연산	레지스터
벡터 길이 필드(159B)를 포함하지 않는 명령어 템플릿	A (도 1a; U=0)	110, 115, 125, 130	zmm 레지스터(벡터 길이는 64 바이트)
	B (도 1b; U=1)	112	zmm 레지스터(벡터 길이는 64 바이트)
벡터 길이 필드(159B)를 포함하는 명령어 템플릿	B (도 1b; U=1)	117, 127	벡터 길이 필드(159B)에 따라, zmm, ymm 또는 xmm 레지스터 (벡터 길이는 64 바이트, 32 바이트 또는 16 바이트)

[0088] 다시 말해, 벡터 길이 필드(159B)는 최대 길이와 하나 이상의 다른 더 짧은 길이들 사이에서 선택하고, 각각의 그러한 더 짧은 길이는 선행 길이의 절반 길이이고; 벡터 길이 필드(159B)를 갖지 않는 명령어 템플릿들은 최대 벡터 길이에 대해 동작한다. 또한, 일 실시예에서, 특정적 벡터 친화적 명령어 포맷(200)의 클래스 B 명령어 템플릿들은 패킹 또는 스칼라 단/배 정밀도 부동 소수점 데이터 및 패킹 또는 스칼라 정수 데이터에 대해 동작한다. 스칼라 연산들은 zmm/ymm/xmm 레지스터 내의 최하위 데이터 요소 위치에서 수행되는 연산들이고; 상위 데이터 요소 위치들은 실시예에 따라 명령어 이전에 이들이 있었던 것과 동일하게 남겨지거나 또는 제로화된다.

[0089] 기입 마스크 레지스터들(315) - 도시된 실시예에서, 각각 64 비트 크기인 8개의 기입 마스크 레지스터(k0 내지 k7)가 존재한다. 대안적인 실시예에서, 기입 마스크 레지스터들(315)은 16 비트 크기이다. 전술한 바와 같이, 본 발명의 일 실시예에서, 벡터 마스크 레지스터 k0은 기입 마스크로서 이용될 수 없고; 통상적으로 k0을 나타내는 인코딩이 기입 마스크에 이용될 때, 이것은 0xFFFF의 하드와이어드 기입 마스크(hardwired write mask)를 선택하여, 그 명령어에 대한 기입 마스킹을 효과적으로 디스에이블한다.

[0090] 범용 레지스터들(325) - 예시된 실시예에서, 메모리 피연산자들을 어드레싱하기 위해 기존의 x86 어드레싱 모드들과 함께 이용되는 16개의 64-비트 범용 레지스터들이 존재한다. 이들 레지스터들은 RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP 및 R8 내지 R15라는 이름들로 참조된다.

[0091] MMX 패킹 정수 플랫 레지스터 파일(350)이 에일리어싱되는 스칼라 부동 소수점 스택 레지스터 파일(x87 스택)(345)-예시된 실시예에서, x87 스택은 x87 명령어 세트 확장을 이용하여 32/64/80-비트 부동 소수점 데이터에 대해 스칼라 부동 소수점 연산들을 수행하는 데 이용된 8-요소 스택이고; MMX 레지스터들을 이용하여 64-비트 패킹 정수 데이터에 대해 연산들을 수행하고, 또한 MMX 및 XMM 레지스터들 사이에서 수행되는 일부 연산들에 대한 피연산자들을 유지한다.

[0092] 본 발명의 대안적인 실시예들은 더 넓거나 더 좁은 레지스터들을 이용할 수 있다. 추가적으로, 본 발명의 대안적인 실시예들은 더 많거나, 더 적거나 또는 상이한 레지스터 파일들 및 레지스터들을 이용할 수 있다.

[0093] **D. 예시적인 코어 아키텍처, 프로세서 및 컴퓨터 아키텍처**

[0094] 프로세서 코어는 상이한 방식으로, 상이한 목적을 위해, 상이한 프로세서에서 구현될 수 있다. 예를 들어, 그러한 코어의 구현은 1) 범용 컴퓨팅을 위해 의도된 범용 순차적 코어; 2) 범용 컴퓨팅을 위해 의도된 고성능 범용 비순차적 코어; 3) 주로 그래픽 및/또는 과학적(쓰루풋) 컴퓨팅을 위해 의도된 특수 목적 코어를 포함할 수 있다. 상이한 프로세서의 구현은: 1) 범용 컴퓨팅을 위해 의도된 하나 이상의 범용 순차적 코어 및/또는 범용 컴퓨팅을 위해 의도된 하나 이상의 범용 비순차적 코어를 포함하는 CPU; 및 2) 주로 그래픽 및/또는 과학적(쓰루풋) 컴퓨팅을 위해 의도된 하나 이상의 특수 목적 코어를 포함하는 코프로세서를 포함할 수 있다. 이러한 상이한 프로세서들은 상이한 컴퓨터 시스템 아키텍처들을 초래하고, 이들 아키텍처들은, 1) CPU와는 별개의 칩 상의 코프로세서; 2) CPU와 동일한 패키지에서의 별개의 다이 상의 코프로세서; 3) CPU와 동일한 다이 상의 코프로세서(이 경우, 이러한 코프로세서는 때때로 통합 그래픽 및/또는 과학적(쓰루풋) 로직과 같은 특수 목적 로직 또는 특수 목적 코어로 지칭됨); 및 4) 추가적인 기능성, 전술한 코프로세서 및 설명된 CPU(때때로 애플리케이션 코어(들) 또는 애플리케이션 프로세서(들)로 지칭됨)를 동일한 다이 상에 포함할 수 있는 시스템 온 칩을 포함할 수 있다. 예시적인 코어 아키텍처들이 다음에 설명되고, 예시적인 프로세서들 및 컴퓨터 아키텍처들의 설명들이 후속된다.

[0095] 도 4a는 본 발명의 실시예들에 따른 예시적인 순차적 파이프라인과 예시적인 레지스터 리네이밍, 비순차적 발행/실행 파이프라인 양자를 도시하는 블록도이다. 도 4b는 본 발명의 실시예들에 따른 프로세서에 포함되는 순차적 아키텍처 코어와 예시적인 레지스터 리네이밍, 비순차적 발행/실행 아키텍처 코어의 예시적인 실시예 양자를 도시하는 블록도이다. 도 4a-b의 실선 상자들은 순차적 파이프라인 및 순차적 코어를 도시하고, 점선 상자들의 선택적인 추가는 레지스터 리네이밍, 비순차적 발행/실행 파이프라인 및 코어를 도시한다. 순차적 양태가 비순차적 양태의 서브세트인 것을 고려하여, 비순차적 양태가 설명될 것이다.

[0096] 도 4a에서, 프로세서 파이프라인(400)은 페치 스테이지(402), 길이 디코드 스테이지(404), 디코드 스테이지(406), 할당 스테이지(408), 리네이밍 스테이지(410), 스케줄링(디스패치 또는 발행이라고도 알려짐) 스테이지(412), 레지스터 판독/메모리 판독 스테이지(414), 실행 스테이지(416), 라이트백(write back)/메모리 기입 스테이지(418), 예외 처리 스테이지(422), 및 커밋 스테이지(424)를 포함한다.

[0097] 도 4b는 실행 엔진 유닛(450)에 결합된 프런트엔드 유닛(430)을 포함하는 프로세서 코어(490)를 도시하고, 양자가 메모리 유닛(470)에 결합되어 있다. 코어(490)는 RISC(reduced instruction set computing) 코어, CISC(complex instruction set computing) 코어, VLIW(very long instruction word) 코어, 또는 하이브리드 또는 대안적인 코어 유형일 수 있다. 또 다른 선택 사항으로서, 코어(490)는 예를 들어, 네트워크 또는 통신 코어, 압축 엔진, 코프로세서 코어, 범용 컴퓨팅 그래픽 프로세싱 유닛(general purpose computing graphics processing unit)(GPGPU) 코어, 그래픽 코어 등과 같은 특수 목적 코어일 수 있다.

[0098] 프런트엔드 유닛(430)은 명령어 캐시 유닛(434)에 결합된 분기 예측 유닛(432)을 포함하고, 명령어 캐시 유닛(434)은 명령어 변환 색인 버퍼(TLB)(436)에 결합되고, 명령어 변환 색인 버퍼(TLB)(436)는 명령어 페치 유닛(438)에 결합되고, 명령어 페치 유닛(438)은 디코드 유닛(440)에 결합된다. 디코드 유닛(440)(또는 디코더)은 명령어들을 디코딩하고, 오리지널 명령어들로부터 디코딩되거나, 다른 방식으로 오리지널 명령어들을 반영하거나, 오리지널 명령어들로부터 도출되는, 하나 이상의 마이크로연산들, 마이크로-코드 엔트리 포인트들, 마이크로명령어들, 다른 명령어들, 또는 다른 제어 신호들을 출력으로서 생성할 수 있다. 디코드 유닛(440)은 다양한 상이한 메커니즘들을 이용하여 구현될 수 있다. 적절한 메커니즘의 예는 탐색표, 하드웨어 구현, 프로그램 가

능 로직 어레이(PLA), 마이크로코드 판독 전용 메모리(ROM) 등을 포함하지만 이에 한정되지 않는다. 일 실시예에서, 코어(490)는 (예를 들어, 디코드 유닛(440)에 또는 그렇지 않으면 프런트엔드 유닛(430) 내에) 특정 매크로 명령어들에 대한 마이크로코드를 저장하는 마이크로코드 ROM 또는 다른 매체를 포함한다. 디코드 유닛(440)은 실행 엔진 유닛(450)의 리네이밍/할당기 유닛(452)에 결합된다.

[0099] 실행 엔진 유닛(450)은 리타이어먼트 유닛(454) 및 하나 이상의 스케줄러 유닛(들)(456)의 세트에 결합되는 리네이밍/할당기 유닛(452)을 포함한다. 스케줄러 유닛(들)(456)은 예비 스테이션들, 중앙 명령어 윈도우 등을 포함하는 임의의 수의 상이한 스케줄러들을 나타낸다. 스케줄러 유닛(들)(456)은 물리적 레지스터 파일(들) 유닛(들)(458)에 결합된다. 물리적 레지스터 파일(들) 유닛(들)(458) 각각은 하나 이상의 물리적 레지스터 파일들을 나타내고, 이들 중 상이한 것들이 스칼라 정수, 스칼라 부동 소수점, 패킹 정수, 패킹 부동 소수점, 벡터 정수, 벡터 부동 소수점, 상태(예를 들어, 실행될 다음 명령어의 어드레스인 명령어 포인터) 등과 같은 하나 이상의 상이한 데이터 유형들을 저장한다. 일 실시예에서, 물리적 레지스터 파일(들) 유닛(458)은 벡터 레지스터 유닛, 기입 마스크 레지스터 유닛, 및 스칼라 레지스터 유닛을 포함한다. 이들 레지스터 유닛들은 아키텍처 벡터 레지스터들, 벡터 마스크 레지스터들 및 범용 레지스터들을 제공할 수 있다. 물리적 레지스터 파일(들) 유닛(들)(458)은(예를 들어, 재정렬 버퍼(들) 및 리타이어먼트 레지스터 파일(들)을 이용하여; 미래의 파일(들), 히스토리 버퍼(들), 및 리타이어먼트 레지스터 파일(들)을 이용하여; 레지스터 맵 및 레지스터들의 풀(pool)을 이용하여; 등등) 레지스터 리네이밍 및 비순차적 실행이 구현될 수 있는 다양한 방식을 예시하기 위해 리타이어먼트 유닛(454)에 의해 오버랩된다. 리타이어먼트 유닛(454)과 물리적 레지스터 파일(들) 유닛(들)(458)은 실행 클러스터(들)(460)에 결합된다. 실행 클러스터(들)(460)는 하나 이상의 실행 유닛들(462)의 세트 및 하나 이상의 메모리 액세스 유닛들(464)의 세트를 포함한다. 실행 유닛들(462)은 다양한 유형의 데이터(예를 들어, 스칼라 부동 소수점, 패킹 정수, 패킹 부동 소수점, 벡터 정수, 벡터 부동 소수점)에 대해 다양한 연산들(예를 들어, 시프트, 덧셈, 뺄셈, 곱셈)을 수행할 수 있다. 일부 실시예는 특정 기능들이나 기능들의 세트들에 전용의 복수의 실행 유닛들을 포함할 수 있지만, 다른 실시예들은 단 하나의 실행 유닛, 또는 모두가 모든 기능들을 수행하는 복수의 실행 유닛을 포함할 수 있다. 스케줄러 유닛(들)(456), 물리적 레지스터 파일(들) 유닛(들)(458), 및 실행 클러스터(들)(460)는 가능하게는 복수 개인 것으로 도시되는데, 그것은 특정 실시예들이 특정 유형의 데이터/연산들에 대해 별개의 파이프라인들(예를 들어, 각각이 그들 자신의 스케줄러 유닛, 물리적 레지스터 파일(들) 유닛, 및/또는 실행 클러스터를 갖는 스칼라 정수 파이프라인, 스칼라 부동 소수점/패킹 정수/패킹 부동 소수점/벡터 정수/벡터 부동 소수점 파이프라인, 및/또는 메모리 액세스 파이프라인-별개의 메모리 액세스 파이프라인의 경우에, 이 파이프라인의 실행 클러스터만이 메모리 액세스 유닛(들)(464)을 갖는 특정 실시예들이 구현됨)을 생성하기 때문이다. 개별 파이프라인들이 사용되는 경우, 이들 파이프라인 중 하나 이상은 비순차적 발행/실행일 수 있고 나머지는 순차적일 수 있다는 점도 이해해야 한다.

[0100] 메모리 액세스 유닛들(464)의 세트는 레벨 2(L2) 캐시 유닛(476)에 결합된 데이터 캐시 유닛(474)에 결합된 데이터 TLB 유닛(472)을 포함하는 메모리 유닛(470)에 결합된다. 일 예시적인 실시예에서, 메모리 액세스 유닛들(464)은 로드 유닛, 저장 어드레스 유닛, 및 저장 데이터 유닛을 포함할 수 있고, 이들 각각은 메모리 유닛(470)의 데이터 TLB 유닛(472)에 결합된다. 명령어 캐시 유닛(434)은 또한 메모리 유닛(470)의 레벨 2(L2) 캐시 유닛(476)에 결합된다. L2 캐시 유닛(476)은 하나 이상의 다른 레벨의 캐시 및 최종적으로 메인 메모리에 결합된다.

[0101] 예로서, 예시적인 레지스터 리네이밍, 비순차적 발행/실행 코어 아키텍처는 다음과 같이 파이프라인(400)을 구현할 수 있는데: 1) 명령어 페치(438)가 페치 및 길이 디코딩 스테이지들(402 및 404)을 수행하고; 2) 디코드 유닛(440)이 디코드 스테이지(406)를 수행하고; 3) 리네임/할당기 유닛(452)이 할당 스테이지(408) 및 리네이밍 스테이지(410)를 수행하고; 4) 스케줄러 유닛(들)(456)이 스케줄 스테이지(412)를 수행하고; 5) 물리 레지스터 파일(들) 유닛(들)(458) 및 메모리 유닛(470)이 레지스터 판독/메모리 판독 스테이지(414)를 수행하고; 실행 클러스터(460)가 실행 스테이지(416)를 수행하고; 6) 메모리 유닛(470) 및 물리 레지스터 파일(들) 유닛(들)(458)이 라이트백/메모리 기입 스테이지(418)를 수행하고; 7) 다양한 유닛들이 예외 처리 스테이지(422)에 수반될 수 있고; 8) 리타이어먼트 유닛(454) 및 물리 레지스터 파일(들) 유닛(들)(458)이 커밋 스테이지(424)를 수행한다.

[0102] 코어(490)는 본 명세서에 설명된 명령어(들)를 포함한 하나 이상의 명령어 세트들(예를 들어, x86 명령어 세트(및 더 새로운 버전들이 추가된 그의 일부 확장들); 캘리포니아주 서니베일의 MIPS Technologies의 MIPS 명령어 세트; 캘리포니아주 서니베일의 ARM Holdings의 ARM 명령어 세트(및 NEON과 같은 선택적인 추가적인 확장들))을 지원할 수 있다. 일 실시예에서, 코어(490)는 패킹 데이터 명령어 세트 확장(예를 들어, AVX1, AVX2)을 지원하

는 로직을 포함함으로써, 많은 멀티미디어 애플리케이션에 의해 이용되는 연산들이 패킹 데이터를 이용하여 수행될 수 있게 한다.

[0103] 코어가 (연산들 또는 쓰레드들의 2개 이상의 병렬 세트를 실행하는) 멀티쓰레딩을 지원할 수 있고, 시간 슬라이싱된 멀티쓰레딩, 동시 멀티쓰레딩을 포함하는 다양한 방식으로(이 경우 단일 물리적 코어는 물리적 코어가 동시에 멀티쓰레딩하는 각각의 쓰레드에 대한 논리적 코어를 제공함), 또는 이들의 조합(예를 들어, Intel® 하이퍼쓰레딩 기술에서와 같은 시간 슬라이싱된 페치 및 디코딩 및 그 후의 동시 멀티쓰레딩)으로 지원할 수 있음을 이해해야 한다.

[0104] 레지스터 리네이밍이 비순차적 실행의 맥락에서 설명되었지만, 레지스터 리네이밍은 순차적 아키텍처에서 사용될 수도 있다는 점을 이해해야 한다. 프로세서의 예시된 실시예가 또한 개별적인 명령어 및 데이터 캐시 유닛들(434/474)과 공유 L2 캐시 유닛(476)을 포함하고 있지만, 대안의 실시예들은, 예를 들어, 레벨 1(L1) 내부 캐시, 또는 다수의 레벨의 내부 캐시와 같은, 명령어들 및 데이터 둘 다에 대한 단일 내부 캐시를 가질 수 있다. 일부 실시예에서, 시스템은 내부 캐시와, 코어 및/또는 프로세서에 대해 외부에 있는 외부 캐시의 조합을 포함할 수 있다. 대안적으로, 캐시 모두가 코어 및/또는 프로세서에 대해 외부적일 수 있다.

[0105] 도 5a-b는 더 특정한 예시적인 순차적 코어 아키텍처의 블록도를 도시하며, 이 코어는 칩 내의(동일한 유형 및/또는 상이한 유형들의 다른 코어들을 포함하는) 여러 개의 로직 블록 중 하나이다. 로직 블록들은 애플리케이션에 따라, 일부 고정 기능 로직, 메모리 I/O 인터페이스들, 및 다른 필요한 I/O 로직을 갖는 고 대역폭 인터커넥트 네트워크(예를 들어, 링 네트워크)를 통해 통신한다.

[0106] 도 5a는 본 발명의 실시예들에 따른 온-다이 인터커넥트 네트워크(502)에 대한 접속 및 레벨 2(L2) 캐시(504)의 로컬 서브세트와 함께, 단일 프로세서 코어의 블록도이다. 일 실시예에서, 명령어 디코더(500)는 패킹 데이터 명령어 세트 확장을 갖는 x86 명령어 세트를 지원한다. L1 캐시(506)는 스칼라 유닛 및 벡터 유닛에 대한 캐시 메모리로의 낮은 레이턴시 액세스들을 허용한다. (설계를 단순화하기 위해) 일 실시예에서 스칼라 유닛(508) 및 벡터 유닛(510)은 별개의 레지스터 세트들(각각 스칼라 레지스터들(512) 및 벡터 레지스터들(514))을 이용하고, 이들 사이에 전달되는 데이터는 메모리에 기입된 다음, 레벨 1(L1) 캐시(506)로부터 다시 판독되지만, 본 발명의 대안적인 실시예들은 상이한 접근법을 이용할 수 있다(예를 들어, 단일의 레지스터 세트를 이용하거나, 또는 기입 및 다시 판독되지 않고 2개의 레지스터 파일 사이에서 데이터가 전달되는 것을 허용하는 통신 경로를 포함함).

[0107] L2 캐시(504)의 로컬 서브세트는 프로세서 코어당 하나씩, 별개의 로컬 서브세트들로 분할되는 글로벌 L2 캐시의 부분이다. 각 프로세서 코어는 L2 캐시(504)의 그 자신의 로컬 서브세트에 대한 직접 액세스 경로를 갖는다. 프로세서 코어에 의해 판독된 데이터는 그의 L2 캐시 서브세트(504)에 저장되고, 그 자신의 로컬 L2 캐시 서브세트들에 액세스하는 다른 프로세서 코어들과 병렬로, 신속히 액세스될 수 있다. 프로세서 코어에 의해 기입된 데이터는 그 자신의 L2 캐시 서브세트(504)에 저장되고, 필요한 경우 다른 서브세트들로부터 플러싱된다. 링 네트워크는 공유 데이터에 대한 코히런시(coherency)를 보장한다. 링 네트워크는 양방향성이어서, 프로세서 코어들, L2 캐시들 및 다른 로직 블록들과 같은 에이전트들이 칩 내에서 상호 통신하는 것을 허용한다. 각각의 링 데이터-경로는 방향당 1012 비트 폭이다.

[0108] 도 5b는 본 발명의 실시예들에 따른 도 5a의 프로세서 코어의 부분의 확대도이다. 도 5b는 벡터 유닛(510) 및 벡터 레지스터들(514)에 관한 추가적인 상세뿐만 아니라, L1 캐시(504)의 L1 데이터 캐시(506A) 부분을 포함한다. 구체적으로, 벡터 유닛(510)은 16-폭 벡터 프로세싱 유닛(VPU)(16-폭 ALU(528) 참조)이고, 이것은 정수, 단정밀도 부동, 및 배정밀도 부동 명령어들 중 하나 이상을 실행한다. VPU는 스위즐 유닛(520)에 의한 레지스터 입력들의 스위즐링, 수치적 변환 유닛들(522A-B)에 의한 수치적 변환, 및 메모리 입력에 대한 복제 유닛(524)에 의한 복제를 지원한다. 기입 마스크 레지스터들(526)은 결과적인 벡터 기입들의 서술을 허용한다.

[0109] 도 6은 본 발명의 실시예들에 따라, 2개 이상의 코어를 가질 수 있고, 통합 메모리 제어기를 가질 수 있고, 통합 그래픽을 가질 수 있는 프로세서(600)의 블록도다. 도 6의 실선 박스들은 싱글 코어(602A), 시스템 에이전트(610), 하나 이상의 버스 제어기 유닛(616)의 세트를 갖는 프로세서(600)를 도시하는 한편, 옵션인 점선 박스들의 추가는 다수의 코어들(602A-N), 시스템 에이전트 유닛(610) 내의 하나 이상의 통합 메모리 제어기 유닛(들)(614)의 세트, 및 특수 목적 로직(608)을 갖는 대안적인 프로세서(600)를 도시한다.

[0110] 따라서, 프로세서(600)의 다양한 구현들은 다음을 포함할 수 있다: 1)(하나 이상의 코어를 포함할 수 있는) 통합 그래픽 및/또는 과학적(쓰루풋) 로직인 특수 목적 로직(608), 및 하나 이상의 범용 코어인 코어들(602A-

N)(예를 들어, 범용 순차적 코어들, 범용 비순차적 코어들, 이 둘의 조합)을 갖는 CPU; 2) 그래픽 및/또는 과학적(쓰루풋)을 위해 주로 의도된 다수의 특수 목적 코어들인 코어들(602A-N)을 갖는 코프로세서; 및 3) 다수의 범용 순차적 코어들인 코어들(602A-N)을 갖는 코프로세서. 따라서, 프로세서(600)는 범용 프로세서, 코프로세서, 또는, 예를 들어, 네트워크 또는 통신 프로세서, 압축 엔진, 그래픽 프로세서, GPGPU(general purpose graphics processing unit), 하이 쓰루풋 MIC(many integrated core) 코프로세서(30개 이상의 코어를 포함함), 임베디드 프로세서 등과 같은 특수 목적 프로세서일 수 있다. 프로세서는 하나 이상의 칩 상에 구현될 수 있다. 프로세서(600)는, 예를 들어, BiCMOS, CMOS, 또는 NMOS와 같은 다수의 프로세스 기술들 중 임의의 것을 사용하여 하나 이상의 기판의 일부가될 수 있고 및/또는 이들 기판 상에 구현될 수 있다.

[0111] 메모리 계층구조는 코어들 내의 하나 이상의 레벨들의 캐시, 세트 또는 하나 이상의 공유 캐시 유닛들(606), 및 통합 메모리 제어기 유닛들(614)의 세트에 연결되는 외부 메모리(도시되지 않음)를 포함한다. 공유 캐시 유닛들(606)의 세트는 레벨 2(L2), 레벨 3(L3), 레벨 4(L4), 또는 기타 레벨들의 캐시와 같은 하나 이상의 중간 레벨 캐시들, LLC(last level cache), 및/또는 이들의 조합을 포함할 수 있다. 일 실시예에서 링 기반 인터커넥트 유닛(612)은 통합 그래픽 로직(608), 공유 캐시 유닛들(606)의 세트, 및 시스템 에이전트 유닛(610)/통합 메모리 제어기 유닛(들)(614)을 상호접속하지만, 대안적인 실시예들은 이러한 유닛들을 상호접속하기 위한 잘 알려진 기술들 중 임의의 것을 사용할 수 있다. 일 실시예에서는, 하나 이상의 캐시 유닛들(606)과 코어들(602A-N) 사이에 코히런시가 유지된다.

[0112] 일부 실시예들에서, 코어들(602A-N) 중 하나 이상은 멀티쓰레딩을할 수 있다. 시스템 에이전트(610)는 코어들(602A-N)을 조정하고 동작시키는 컴포넌트들을 포함한다. 시스템 에이전트 유닛(610)은 예를 들어 PCU(power control unit) 및 디스플레이 유닛을 포함할 수 있다. PCU는 코어들(602A-N) 및 통합 그래픽 로직(608)의 전원 상태를 조절하기 위해 필요한 로직과 컴포넌트들일 수 있거나 이들을 포함할 수 있다. 디스플레이 유닛은 하나 이상의 외부 접속된 디스플레이들을 구동하기 위한 것이다.

[0113] 코어들(602A-N)은 아키텍처 명령어 세트와 관련하여 동종 또는 이종일 수 있다; 즉, 코어들(602A-N) 중 2개 이상은 동일 명령어 세트를 실행할 수 있는 반면, 다른 것들은 그 명령어 세트의 서브세트 또는 상이한 명령어 세트만을 실행할 수 있다.

[0114] 도 7 내지 도 10은 예시적인 컴퓨터 아키텍처들의 블록도들이다. 랩톱들, 데스크톱들, 핸드헬드 PC들, 퍼스널 디지털 어시스턴트들, 엔지니어링 워크스테이션들, 서버들, 네트워크 디바이스들, 네트워크 허브들, 스위치들, 임베디드 프로세서들, DSP(digital signal processor)들, 그래픽 디바이스들, 비디오 게임 디바이스들, 셋톱 박스들, 마이크로컨트롤러들, 휴대 전화들, 휴대용 미디어 플레이어들, 핸드헬드 디바이스들, 및 다양한 다른 전자 디바이스들에 대해 본 기술분야에 알려진 다른 시스템 설계들 및 구성들도 적합하다. 일반적으로, 본 명세서에 개시되는 바와 같은 프로세서 및/또는 다른 실행 로직을 통합할 수 있는 매우 다양한 시스템들 또는 전자 디바이스들이 일반적으로 적합하다.

[0115] 이제 도 7을 참조하면, 본 발명의 일 실시예에 따른 시스템(700)의 블록도가 도시된다. 시스템(700)은 하나 이상 프로세서들(710, 715)을 포함할 수 있고, 이는 제어기 허브(720)에 연결된다. 일 실시예에서, 제어기 허브(720)는, GMCH(graphics memory controller Hub)(790) 및 IOH(Input/Output Hub)(750)(개별 칩들 상에 있을 수 있음)를 포함하고; GMCH(790)는 메모리 및 메모리(740)와 코프로세서(745)에 연결되는 그래픽 제어기들을 포함하고; IOH(750)는 GMCH(790)에 입력/출력(I/O) 디바이스들(760)을 연결한다. 대안적으로, 메모리 및 그래픽 제어기들 중 하나 또는 둘 다는(본 명세서에서 설명되는 바와 같이) 프로세서 내에 통합되고, 메모리(740) 및 코프로세서(745)는 프로세서(710) 및 IOH(750)와 단일 칩에 있는 제어기 허브(720)에 직접 연결된다.

[0116] 추가 프로세서들(715)의 옵션 특성은 도 7에서 파선으로 표기된다. 각각의 프로세서(710, 715)는, 본 명세서에 설명되는 처리 코어들 중 하나 이상을 포함할 수 있고, 프로세서(600)의 일부 버전일 수 있다.

[0117] 메모리(740)는, 예를 들어, DRAM(dynamic random access memory), PCM(phase change memory), 또는 이 둘의 조합일 수 있다. 적어도 하나의 실시예에 대해, 제어기 허브(720)는, FSB(frontside bus)와 같은 멀티-드롭 버스, QPI(QuickPath Interconnect)와 같은 포인트 투 포인트 인터페이스, 또는 유사한 접속(795)을 통해 프로세서(들)(710, 715)과 통신한다.

[0118] 일 실시예에서, 코프로세서(745)는, 예를 들어, 하이 쓰루풋 MIC 프로세서, 네트워크 또는 통신 프로세서, 압축 엔진, 그래픽 프로세서, GPGPU, 임베디드 프로세서 등과 같은 특수 목적 프로세서이다. 일 실시예에서, 제어기 허브(720)는 통합 그래픽 가속기를 포함할 수 있다.

[0119] 아키텍처 특성, 마이크로아키텍처 특성, 열적 특성, 전력 소비 특성 등을 포함하는 장점 기준들의 스펙트럼과 관련하여 물리적 리소스들(710, 715) 사이에는 다양한 상이함이 존재할 수 있다.

[0120] 일 실시예에서, 프로세서(710)는 일반적인 유형의 데이터 처리 연산들을 제어하는 명령어들을 실행한다. 명령어들 내에는 코프로세서 명령어들이 임베딩될 수 있다. 프로세서(710)는 이러한 코프로세서 명령어들을 부속된 코프로세서(745)에 의해 실행되어야 하는 유형의 것으로 인식한다. 따라서, 프로세서(710)는 이러한 코프로세서 명령어들(또는 코프로세서 명령어들을 나타내는 제어 신호들)을 코프로세서 버스 또는 다른 인터커넥트 상에서 코프로세서(745)에 발행한다. 코프로세서(들)(745)는 수신된 코프로세서 명령어들을 수락 및 실행한다.

[0121] 이제 도 8을 참조하면, 본 발명의 일 실시예에 따른 제1의 더 구체적인 예시적인 시스템(800)의 블록도가 도시된다. 도 8에 도시된 바와 같이, 멀티프로세서 시스템(800)은 포인트 투 포인트 인터커넥트 시스템이고, 포인트 투 포인트 인터커넥트(850)를 통해 연결되는 제1 프로세서(870) 및 제2 프로세서(880)를 포함한다. 프로세서들(870 및 880) 각각은 프로세서(600)의 일부 버전일 수 있다. 본 발명의 일 실시예에서, 프로세서들(870 및 880)은 각각 프로세서들(710 및 715)이고, 코프로세서(838)은 코프로세서(745)이다. 또 다른 실시예에서, 프로세서들(870 및 880)은 각각 프로세서(710) 및 코프로세서(745)이다.

[0122] 프로세서들(870 및 880)은 각각 IMC(integrated memory controller) 유닛들(872 및 882)을 포함하는 것으로 도시된다. 프로세서(870)는 또한 자신의 버스 제어기 유닛들의 일부로서 포인트 투 포인트(P-P) 인터페이스들(876, 878)을 포함하고; 유사하게, 제2 프로세서(880)는 P-P 인터페이스들(886, 888)을 포함한다. 프로세서들(870, 880)은 P-P 인터페이스 회로들(878, 888)을 사용하는 포인트 투 포인트(P-P) 인터페이스(850)를 통해 정보를 교환할 수 있다. 도 8에 도시된 바와 같이, IMC들(872 및 882)은 프로세서들을 각각의 메모리, 즉 메모리(832) 및 메모리(834)에 연결하며, 이는 각각의 프로세서들에 로컬로 부속되는 메인 메모리의 일부들일 수 있다.

[0123] 프로세서들(870, 880)은 각각 포인트 투 포인트 인터페이스 회로들(876, 894, 886, 898)을 사용하여 개별 P-P 인터페이스들(852, 854)을 통해 칩셋(890)과 정보를 교환할 수 있다. 칩셋(890)은 옵션으로 고성능 인터페이스(839)를 통해 코프로세서(838)와 정보를 교환할 수 있다. 일 실시예에서, 코프로세서(838)는, 예를 들어, 하이 쓰루풋 MIC 프로세서, 네트워크 또는 통신 프로세서, 압축 엔진, 그래픽 프로세서, GPGPU, 임베디드 프로세서 등과 같은 특수 목적 프로세서이다.

[0124] 공유 캐시(도시되지 않음)는 어느 하나의 프로세서에 포함되거나, 둘 다의 프로세서의 외부이지만 여전히 P-P 인터커넥트를 통해 프로세서들과 접속될 수 있어서, 프로세서가 저전력 모드에 놓이는 경우 어느 하나 또는 둘 다의 프로세서의 로컬 캐시 정보가 공유된 캐시에 저장될 수 있다.

[0125] 칩셋(890)은 인터페이스(896)를 통해 제1 버스(816)에 연결될 수 있다. 일 실시예에서, 제1 버스(816)는 PCI(Peripheral Component Interconnect) 버스이거나, 또는 PCI Express 버스 또는 또 다른 제3 세대 I/O 인터커넥트 버스와 같은 버스일 수 있지만, 본 발명의 범위는 이에 제한되지는 않는다.

[0126] 도 8에 도시된 바와 같이, 다양한 I/O 디바이스들(814)이 제1 버스(816)를 제2 버스(820)에 연결하는 버스 브리지(818)와 함께 제1 버스(816)에 연결될 수 있다. 일 실시예에서, 코프로세서, 하이 스루풋 MIC 프로세서, GPGPU, 가속기(예를 들어, 그래픽 가속기 또는 디지털 신호 처리(DSP) 유닛 등), 필드 프로그래머블 게이트 어레이 또는 임의의 다른 프로세서와 같은 하나 이상의 추가적인 프로세서(들)(815)가 제1 버스(816)에 연결된다. 일 실시예에서, 제2 버스(820)는 LPC(low pin count) 버스일 수 있다. 일 실시예에서는, 예를 들어, 키보드 및/또는 마우스(822), 통신 디바이스들(827) 및 명령어들/코드 및 데이터(830)를 포함할 수 있는 디스크 드라이브 또는 기타 대용량 저장 디바이스와 같은 저장 유닛(828)을 포함하는 다양한 디바이스들이 제2 버스(820)에 연결될 수 있다. 또한, 오디오 I/O(824)가 제2 버스(820)에 연결될 수 있다. 다른 아키텍처들도 가능하다는 점에 유의한다. 예를 들어, 도 8의 포인트 투 포인트 아키텍처 대신에, 시스템은 멀티드롭 버스 또는 다른 이러한 아키텍처를 구현할 수 있다.

[0127] 이제 도 9를 참조하면, 본 발명의 실시예에 따른 제2의 더 구체적인 예시적인 시스템(900)의 블록도가 도시된다. 도 8 및 도 9에서 동일한 요소들은 동일한 참조 번호들을 가지며, 도 8의 특정 양태들은 도 9의 다른 양태들을 불명료하게 하는 것을 회피하기 위해 도 9로부터 생략되었다.

[0128] 도 9는 프로세서들(870, 880)이 통합 메모리 및 I/O 제어 로직("CL")(872 및 882)를 각각 포함할 수 있다는 점을 도시한다. 따라서, CL(872, 882)는 통합 메모리 제어기 유닛들 및 I/O 제어 로직을 포함한다. 도 9는 CL(872, 882)에 메모리들(832, 834)만이 연결되는 것이 아니라, 제어 로직(872, 882)에 I/O 디바이스들(914)도

연결된다는 점을 도시한다. 레거시 I/O 디바이스들(915)은 칩셋(890)에 연결된다.

[0129] 이제, 도 10을 참조하면, 본 발명의 일 실시예에 따른 SoC(1000)의 블록도가 도시된다. 도 6에서의 유사한 요소들은 동일한 참조 번호를 갖는다. 또한, 파선 박스들은 더 진보된 SoC들에 대한 옵션 특징들이다. 도 10에서, 인터커넥트 유닛(들)(1002)은 다음에 연결된다: 하나 이상의 코어(202A-N)의 세트 및 공유 캐시 유닛(들)(606)을 포함하는 애플리케이션 프로세서(1010); 시스템 에이전트 유닛(610); 버스 제어기 유닛(들)(616); 통합 메모리 제어기 유닛(들)(614); 통합 그래픽 로직, 이미지 프로세서, 오디오 프로세서, 및 비디오 프로세서를 포함할 수 있는 하나 이상의 코프로세서들(1020) 또는 그 세트; SRAM(static random access memory) 유닛(1030); DMA(direct memory access) 유닛(1032); 및 하나 이상의 외부 디스플레이에 연결하기 위한 디스플레이 유닛(1040). 일 실시예에서, 코프로세서(들)(1020)는, 예를 들어, 네트워크 또는 통신 프로세서, 압축 엔진, GPGPU, 하이 쓰루풋 MIC 프로세서, 임베디드 프로세서와 같은 특수 목적 프로세서를 포함한다.

[0130] 본 명세서에 개시된 메커니즘들의 실시예들은 하드웨어, 소프트웨어, 펌웨어, 또는 이러한 구현 접근법들의 조합으로 구현될 수 있다. 본 발명의 실시예들은, 적어도 하나의 프로세서, 저장 시스템(휘발성 및 비휘발성 메모리 및/또는 저장 요소들을 포함함), 적어도 하나의 입력 디바이스 및 적어도 하나의 출력 디바이스를 포함하는 프로그램가능 시스템들 상에서 실행되는 컴퓨터 프로그램들 또는 프로그램 코드로서 구현될 수 있다.

[0131] 도 8에 도시된 코드(830)과 같은 프로그램 코드는 본 명세서에 설명되는 기능들을 수행하고 출력 정보를 생성하도록 입력 명령어들에 적용될 수 있다. 출력 정보는 공지된 방식으로 하나 이상의 출력 디바이스에 적용될 수 있다. 이 애플리케이션을 위해, 처리 시스템은, 예를 들어, 디지털 신호 프로세서(DSP), 마이크로컨트롤러, 주문형 집적 회로(ASIC) 또는 마이크로프로세서와 같은 프로세서를 갖는 임의의 시스템을 포함한다.

[0132] 프로그램 코드는 처리 시스템과 통신하기 위해 하이 레벨 절차형 또는 객체 지향형 프로그래밍 언어로 구현될 수 있다. 또한, 프로그램 코드는 요구되는 경우에 어셈블리 또는 기계 언어로 구현될 수 있다. 사실상, 본 명세서에 설명된 메커니즘들은 임의의 특정 프로그래밍 언어로 범위가 제한되지는 않는다. 임의의 경우에, 이 언어는 컴파일형 또는 해석형 언어일 수 있다.

[0133] 적어도 하나의 실시예의 하나 이상의 양태는, 머신에 의해 판독될 때에 이 머신으로 하여금 본 명세서에 설명된 기술들을 수행하는 로직을 제조하게 하는, 프로세서 내의 다양한 로직을 나타내는 머신 판독가능 매체 상에 저장된 대표적인 명령어들에 의해 구현될 수 있다. "IP 코어들"로서 알려진 이러한 표현들은 유형(tangible) 머신 판독가능 매체 상에 저장되고, 다양한 고객들 또는 제조 설비들에 공급되어, 로직 또는 프로세서를 실제로 제조하는 제조 머신들로 로딩될 수 있다.

[0134] 이러한 머신 판독가능 저장 매체는, 하드 디스크들, 플로피 디스크들, 광학 디스크들, CD-ROM들(compact disk read-only memories), CD-RW들(compact disk rewritable's) 및 광자기 디스크들을 포함하는 임의의 다른 유형의 디스크, ROM들(read-only memories), RAM들(random access memories), 예를 들어 DRAM들(dynamic random access memories), SRAM들(static random access memories), EPROM들(erasable programmable read-only memories), 플래시 메모리들, EEPROM들(electrically erasable programmable read-only memories), 상변화 메모리(PCM)와 같은 반도체 디바이스들, 자기 또는 광학 카드들, 또는 전자 명령어들을 저장하기에 적합한 임의의 다른 유형의 매체와 같은 저장 매체를 비롯하여, 머신 또는 디바이스에 의해 제조되거나 형성되는 제조물들의 비일시적인 유형 배열들을 포함할 수 있지만, 이에 제한되지는 않는다.

[0135] 따라서, 본 발명의 실시예들은, 명령어들을 포함하거나, 또는 본 명세서에 설명된 구조들, 회로들, 장치들, 프로세서들 및/또는 시스템 피처들을 정의하는 HDL(Hardware Description Language)과 같은 설계 데이터를 포함하는 비일시적인 유형의 머신 판독가능 매체를 또한 포함한다. 이러한 실시예들은 프로그램 제품들로 또한 언급될 수 있다.

[0136] 일부 경우에, 소스 명령어 세트로부터 타깃 명령어 세트로 명령어를 변환하기 위해 명령어 변환기가 이용될 수 있다. 예를 들어, 명령어 변환기는 명령어를 코어에 의해 처리될 하나 이상의 다른 명령어로(예를 들어, 정적 바이너리 번역, 동적 번역(dynamic compilation)을 포함하는 동적 바이너리 번역을 이용하여) 번역하거나, 모핑하거나, 에뮬레이트하거나, 또는 다른 방식으로 변환할 수 있다. 명령어 변환기는 소프트웨어, 하드웨어, 펌웨어 또는 이들의 조합으로 구현될 수 있다. 명령어 변환기는 온 프로세서(on processor), 오프 프로세서(off processor), 또는 부분 온 및 부분 오프 프로세서(part on and part off processor)일 수 있다.

[0137] 도 11은 본 발명의 실시예들에 따라 소스 명령어 세트 내의 바이너리 명령어들을 타겟 명령어 세트 내의 바이너리 명령어들로 변환하는 소프트웨어 명령어 변환기의 사용을 대조하는 블록도다. 도시된 실시예에서, 명령어

변환기는 소프트웨어 명령어 변환기이지만, 대안적으로 명령어 변환기는 소프트웨어, 펌웨어, 하드웨어 또는 이들의 다양한 조합으로 구현될 수 있다. 도 11은 하이 레벨 언어(1102)의 프로그램을 x86 컴파일러(1104)를 사용하여 컴파일하여, 적어도 하나의 x86 명령어 세트 코어를 갖는 프로세서(1116)에 의해 원천적으로 실행될 수 있는 x86 바이너리 코드(1106)를 생성할 수 있다는 것을 도시한다. 적어도 하나의 x86 명령어 세트 코어를 갖는 프로세서(1116)는, 적어도 하나의 x86 명령어 세트 코어를 갖는 Intel 프로세서와 실질적으로 동일한 결과를 달성하기 위해,(1) Intel x86 명령어 세트 코어의 명령어 세트의 상당 부분 또는(2) 적어도 하나의 x86 명령어 세트 코어를 갖는 Intel 프로세서 상에서 실행되도록 되어 있는 애플리케이션들 또는 다른 소프트웨어의 오브젝트 코드 버전들을, 호환가능하게 실행하거나 다른 방식으로 처리함으로써 적어도 하나의 x86 명령어 세트 코어를 갖는 Intel 프로세서와 실질적으로 동일한 기능들을 수행할 수 있는 임의의 프로세서를 나타낸다. x86 컴파일러(1104)는 추가적인 링크 처리 유무와 무관하게 적어도 하나의 x86 명령어 세트 코어를 갖는 프로세서(1116)에서 실행될 수 있는 x86 바이너리 코드(1106)(예를 들어, 오브젝트 코드)를 생성하도록 동작될 수 있는 컴파일러를 나타낸다. 유사하게, 도 11은, 하이 레벨 언어(1102)에서의 프로그램이 대안 명령어 세트 컴파일러(1108)를 사용하여 컴파일되어, 적어도 하나의 x86 명령어 세트 코어가 없는 프로세서(1114)(예를 들어, 캘리포니아주 서니베일의 MIPS Technologies의 MIPS 명령어 세트를 실행하는 및/또는 캘리포니아주 서니베일의 ARM Holdings의 ARM 명령어 세트를 실행하는 코어들이 없는 프로세서)에 의해 원천적으로 실행될 수 있는 대안 명령어 세트 바이너리 코드(1110)을 생성할 수 있다는 점을 보여준다. 명령어 변환기(1112)는 x86 바이너리 코드(1106)를 x86 명령어 세트 코어가 없는 프로세서(1114)에 의해 원천적으로 실행될 수 있는 코드로 변환하는데 사용된다. 이러한 변환된 코드는 대안 명령어 세트 바이너리 코드(1110)와 동일할 가능성이 없는데, 그 이유는 이를 행할 수 있는 명령어 변환기를 제조하기 어렵기 때문이다; 그러나, 변환된 코드는 일반적인 연산을 달성할 것이며, 대안 명령어 세트로부터의 명령어들로 이루어질 것이다. 따라서, 명령어 변환기(1112)는, 에뮬레이션, 시뮬레이션 또는 임의의 다른 프로세스를 통해, x86 명령어 세트 프로세서 또는 코어를 갖지 않는 프로세서 또는 기타 전자 디바이스가 x86 바이너리 코드(1106)를 실행할 수 있게 해주는 소프트웨어, 펌웨어, 하드웨어, 또는 이들의 조합을 나타낸다.

[0138] 벡터 인덱스 로드 및 저장을 위한 방법 및 장치

[0139] 고성능 코드(HPC)의 주요 특징은 상당한 수의 메모리 참조를 가지며, 그 중 대부분은 동일한 메모리이지만 상이한 인덱스를 갖는 메모리를 가리킨다는 것이다. 현재의 구현에서들, 이들 메모리 인덱스 각각은 "개별" 범용 레지스터(예를 들어, x86 아키텍처에서의 GPR x86-64)를 필요로 한다. 제한된 레지스터 세트(예를 들어, 8개의 x86 및 16개의 x64 레지스터)로 그리고 다수의 메모리 참조로, GPR 레지스터 세트는 곧 채워져서 스택에서 스필-앤-필(spill-and-fill)을 초래하고 비대하고 비효율적인 코드를 생성한다. 또한, 이들 인덱스는 가산, 승산 또는 감산과 같은 추가 계산을 필요로 하고, 이들 모두가 GPR 스칼라 계산을 요구하여, GPR 레지스터 스택을 더 소비하고 그 계산들이 모든 인덱스들에 걸쳐 동일하기 때문에 불필요한 중복 코드를 생성한다.

[0140] 본 발명의 실시예들은 각각의 로드/저장 연산에 대해 인덱스 레지스터로서 SIMD 벡터 레지스터를 사용함으로써 로드 및 저장의 성능을 향상시키는 벡터 인덱싱된 로드/저장 명령을 포함한다. 이러한 방식으로 벡터 레지스터를 사용하면 인덱스의 커뮤테이션(commutation)이 가속화되고, 이는 GPR 스칼라 계산이 아닌 SIMD 벡터 계산을 사용하여 달성될 수 있어, 결과적으로 더 작고 효율적인 코드가 생성된다. 본 발명의 실시예들은 또한 자연적으로 벡터화된 루프 내에서 유동한다. 모든 인덱스 계산이 이제 SIMD 벡터 레지스터에서 수행되고 스택에 스필된 다음 로드/저장을 수행하기 위해 다시 판독될 필요가 없기 때문에, HPC 코드 루프의 더 효율적인 벡터화가 달성될 수 있다. 또한, 이러한 실시예들은 컴파일러가 벡터화하는 것이 상당히 쉽다(예를 들어, GPR 스택의 관리가 더 적기 때문에).

[0141] 본 발명의 일 실시예는 3-소스 벡터 인덱스 로드 명령(VILD) 및 3-소스 벡터 인덱스 저장 명령(VIST)을 포함하는데, 이는 다음의 형태들을 취한다:

[0142] 벡터 인덱스 로드: VILD[B/W/D/Q] DST_SIMD_REG {k1}{z}, [MEM+VectIndex], ImmSelectForVectIndex; 및

[0143] 벡터 인덱스 저장: VIST[B/W/D/Q] [MEM+VectIndex] {k1}{z}, SRC_SIMD_REG, ImmSelectForVectIndex, 여기서:

[0144] → MEM은 메모리 베이스 참조이고,

[0145] → VectIndex는 SIMD 벡터 레지스터이고,

[0146] → ImmSelectForVectIndex는 VectorIndex 레지스터로부터 대응하는 인덱스를 선택하는 데 사용되고,

[0147] → B/W/D/Q는 명령어가 바이트, 워드, 더블워드 또는 쿼드워드 값을 사용하여 수행되는지를 나타낸다.

[0148] 도 12는 본 발명의 실시예가 구현될 수 있는 예시적인 프로세서(1255)를 도시한다. 일 실시예에서, 프로세서(1255)의 각각의 코어 0-N은 본 명세서에서 설명된 벡터 인덱스 로드/저장 동작을 수행하기 위한 벡터 인덱스 로드/저장 로직(1291)을 갖는 메모리 관리 유닛(1290)을 포함한다. 또한, 각각의 코어 0-N은 범용 레지스터(GPR)들(1205)의 세트, 벡터 레지스터들(1206)의 세트, 및 마스크 레지스터들(1207)의 세트를 포함한다. 일 실시예에서, 다수의 벡터 데이터 요소들은 2개의 256 비트 값, 4개의 128 비트 값, 8개의 64 비트 값, 16개의 32 비트 값 등을 저장하기 위한 512 비트 폭을 가질 수 있는 각각의 벡터 레지스터(1206) 내에 패킹된다. 그러나, 본 발명의 기본 원리들은 임의의 특정 크기/유형의 벡터 데이터로 제한되지 않는다. 일 실시예에서, 마스크 레지스터들(1207)은 벡터 레지스터들(1206)에 저장된 값들에 대해 비트 마스킹 연산들을 수행하는 데 사용되는 8개의 64 비트 피연산자 마스크 레지스터들을 포함한다(예를 들어, 전술한 마스크 레지스터들 k0 내지 k7로서 구현됨). 그러나, 본 발명의 기본 원리들은 특정 마스크 레지스터 크기/유형에 제한되지 않는다.

[0149] 단순화를 위해 도 12에는 단일 프로세서 코어("코어 0")의 세부 사항이 도시되어 있다. 그러나, 도 12에 도시된 각각의 코어는 코어 0과 동일한 로직 세트를 가질 수 있음을 이해할 것이다. 예를 들어, 각각의 코어는 특정된 캐시 관리 정책에 따라 명령어들 및 데이터를 캐싱하기 위한 전용 레벨 1(L1) 캐시(1212) 및 레벨 2(L2) 캐시(1211)를 포함할 수 있다. L1 캐시(1212)는 명령어들을 저장하기 위한 개별 명령어 캐시(1220) 및 데이터를 저장하기 위한 개별 데이터 캐시(1221)를 포함한다. 다양한 프로세서 캐시들 내에 저장된 명령어들 및 데이터는 고정 크기(예를 들어, 길이가 64, 128, 512 바이트)일 수 있는 캐시 라인의 세분성으로 관리된다. 이 예시적인 실시예의 각각의 코어는 메인 메모리(1200) 및/또는 공유 레벨 3(L3) 캐시(1216)로부터 명령어들을 페치하기 위한 명령어 페치 유닛(1210); 명령어들을 디코딩하기 위한(예를 들어, 프로그램 명령어들을 마이크로-연산들 또는 "uop들"로 디코딩하기 위한) 디코딩 유닛(1220); 명령어들을 실행하기 위한 실행 유닛(1240); 및 명령어들을 리타이어하고 그 결과들을 라이트 백(write back)하기 위한 라이트백(writeback) 유닛(1250)을 포함한다.

[0150] 명령어 페치 유닛(1210)은 메모리(1200)(또는 캐시들 중 하나)로부터 페치될 다음 명령어의 어드레스를 저장하기 위한 다음 명령어 포인터(1203); 어드레스 변환의 속도를 향상시키기 위해 최근에 사용된 가상-물리적 명령어 어드레스들의 맵을 저장하기 위한 명령어 변환 색인 버퍼(ITLB)(1204); 명령어 분기 어드레스를 추측적으로 예측하기 위한 분기 예측 유닛(1202); 분기 어드레스 및 타겟 어드레스를 저장하기 위한 BTB(branch target buffer)들(1201)을 포함하는 다양한 잘 알려진 컴포넌트들을 포함한다. 일단 패치되면, 명령어들은 그 후 디코딩 유닛(1230), 실행 유닛(1240), 및 라이트백 유닛(1250)을 포함하는 명령어 파이프라인의 나머지 스테이지들로 스트리밍된다. 이들 유닛들의 각각의 구조 및 기능은 본 기술분야의 통상의 기술자에 의해 잘 이해되며 본 발명의 다른 실시예들의 관련 양태들을 모호하게 하는 것을 피하기 위해 본 명세서에서 상세히 설명하지 않을 것이다.

[0151] 도 13은 소스 인덱스 값 i0-i7을 저장하기 위한 벡터 인덱스 레지스터(1301) 및 벡터 인덱스 로드 명령어들에 응답하여 메모리(1200)로부터 로드된 벡터를 저장하거나 벡터 인덱스 저장 명령어에 응답하여 메모리(1200)에 저장하기 위한 목적지/소스 벡터 레지스터(1302)를 이용하는 벡터 인덱스 로드/저장 로직(1291)의 일 실시예를 도시한다. 일 실시예에서, 목적지/소스 벡터 레지스터(1302)는 8개의 64 비트 벡터 데이터 요소(63:0, 127:64, 191:128 등에 위치함)를 갖는 512 비트 벡터 레지스터를 포함하고, 벡터 인덱스 레지스터(1301)는 8개의 64 비트 인덱스 값을 갖는 512 비트 벡터 레지스터를 포함한다. 마스크 레지스터(1303)는 본 명세서에서 설명된 바와 같이 기입 마스킹 연산을 수행하기 위한 마스크 값을 저장한다. 일 실시예에서, 마스크 레지스터(1303)는(예를 들어, 전술한 k0-k7 레지스터들과 같은) 64 비트 레지스터이다. 메모리 어드레스를 계산하기 위해 벡터 인덱스 레지스터(1301)로부터의 인덱스와 결합하기 위해 베이스 메모리 참조(1304)가 제공된다. 또한, 최종 메모리 어드레스를 계산하기 위해 베이스 메모리 참조(1304)와 결합될 벡터 인덱스 레지스터(1301)로부터의 특정 벡터 인덱스를 식별하기 위해 즉치 선택 값(1305)이 제공된다. 다양한 예시적인 레지스터 크기 및 데이터 유형이 설명의 목적으로 위에 언급되었지만, 본 발명의 기본 원리는 임의의 특정 크기/유형의 인덱스, 마스크, 즉치 값 또는 데이터 요소로 제한되지 않는다는 것에 유의해야 한다.

[0152] 동작 시, 복수의 상이한 인덱스 i0-i7이 벡터 인덱스 레지스터(1301) 내에 저장된다. 벡터 인덱스 로드/저장 로직(1291)은 즉치 값(1305)으로부터 현재 로드/저장 연산에 사용될 인덱스를 식별한다. 일 실시예에서, 즉치 값(1305)은 벡터 인덱스 레지스터(1301) 내에 저장되는 N개의 상이한 인덱스 값들을 식별하기 위해 0과 N-1 사이의 값을 포함한다(도 13에서 N=8). 벡터 인덱스 로드/저장 로직(1291)이 적절한 인덱스를 식별하면, 그것은 그 인덱스를 베이스 메모리 참조(1304)와 결합하여 로드/저장 연산을 위한 메모리 어드레스를 계산한다. 그 후

그것은 목적지/소스 벡터 레지스터(1302)를 사용하여 로드/저장을 수행하고, 메모리로부터 값들을 벡터 레지스터(1302) 내의 데이터 요소 위치들로 로딩하거나 벡터 레지스터(1302)의 데이터 요소 위치들로부터의 값들을 메모리에 저장한다. 일 실시예에서, 마스크 레지스터(1303)로부터의 마스크 값은 주어진 메모리 어드레스로부터/로 로드/저장될 요소의 수를 특정하는 데 사용된다.

[0153] 도 14는 배정밀도 부동 소수점 값들(각각 64 비트)이 메모리로부터 로드되는 특정 예를 도시하며, 여기서 Struct Atom [double x, y, z] 및 Atom AtomArray [10000]이다. 이 예에서 MEM은 어레이의 베이스, 즉 & AtomArray [0]을 가리킨다. VectIndex는 메모리 인덱스들(DWORD 값)로서 8개의 값을 포함하는 ZMM1 레지스터(1301)이다: ZMM1 = {7000, 6000, 4000, 3000, 900, 500, 2000, 1000}. 마스크 k1(1303) = 이진수 00000111은 요소들 x, y 및 z가 주어진 인덱스에서 로딩되어야 함을 지시한다(LSB 위치들에서 3개의 세트 비트 111에 의해 지시된 바와 같이). 도시된 예에서, 인덱스 2000은 ZMM1 레지스터(요소 1)의 제2 요소에 저장되는, 로드될 데이터를 식별한다고 가정한다. 따라서, AtomArray의 인덱스 2000으로부터 Atom 요소들 x, y, z를 로드하기 위해, 명령어 VILD ZMM2 {k1}, [MEM + ZMM1], 1이 실행될 수 있다. 로드 연산의 결과는 목적지 벡터 레지스터(1302), ZMM2 = {0,0,0,0,0, AtomArray[2000].z, AtomArray[2000].y, AtomArray[2000].x}이다.

[0154] 본 발명의 일 실시예에 따른 방법이 도 15에 도시되어 있다. 이 방법은 전술한 아키텍처의 컨텍스트 내에서 실행될 수 있지만, 임의의 특정 시스템 아키텍처로 제한되지는 않는다.

[0155] 1501에서, 벡터 인덱스 로드 명령어가 메모리로부터 페치되거나 캐시(예를 들어, L1, L2 또는 L3 캐시)로부터 판독된다. 1502에서, 인덱스 값들이 소스 벡터 인덱스 레지스터에 저장된다. 1503에서, 베이스 메모리 어드레스와 결합될 인덱스는 벡터 인덱스 로드 명령어와 함께 포함된 즉치 값을 사용하여 벡터 인덱스 레지스터 내에서 식별된다. 1504에서, 인덱스를 베이스 메모리 참조와 결합하여 로드 연산을 위한 메모리 어드레스를 결정한다. 마지막으로, 1505에서, 메모리 어드레스에 의해 식별된 데이터가 메모리로부터 로드된다(예를 들어, 전술한 어레이 데이터와 같은).

[0156] 본 발명의 일 실시예에 따른 방법이 도 16에 도시되어 있다. 이 방법은 전술한 아키텍처의 컨텍스트 내에서 실행될 수 있지만, 임의의 특정 시스템 아키텍처로 제한되지는 않는다.

[0157] 1601에서, 벡터 인덱스 저장 명령어가 메모리로부터 페치되거나 캐시(예컨대, L1, L2 또는 L3 캐시)로부터 판독된다. 1602에서, 인덱스 값들이 소스 벡터 인덱스 레지스터에 저장된다. 1603에서, 베이스 메모리 어드레스와 결합될 인덱스는 벡터 인덱스 저장 명령어와 함께 포함된 즉치 값을 사용하여 벡터 인덱스 레지스터 내에서 식별된다. 1604에서, 인덱스를 베이스 메모리 참조와 결합하여 저장 연산을 위한 메모리 어드레스를 결정한다. 마지막으로, 1605에서, 데이터가 메모리 어드레스의 메모리에 저장된다.

[0158] 전술한 바와 같이, 고성능 코드(HPC)는 상당한 수의 메모리 참조를 가지며, 여기서 베이스 메모리 어드레스는 동일하지만 인덱스들은 상이하다. 또한, 인덱스들은 일반적으로 계산되고 그 값들은 SIMD 벡터 레지스터들에 저장되고 유지된다. 본 발명의 실시예들은 SIMD 벡터 레지스터들 내의 인덱스들을 유지 및 재사용하는 것을 이용하여 스필-투-메모리(spill-to-mem) 및/또는 필-투-GPR 레지스터 연산들을 수행하지 않고 로드 및 저장을 수행한다.

[0159] 예로서, 배정밀도 데이터 세트에 대해 동작하는 HPC 애플리케이션으로부터의 C 코드 시퀀스의 애플리케이션 사용 사례, 스냅샷을 고려한다:

```
if (delr2 >= lowest_efs_u) { //FP 배정밀도 데이트 세트
        ind = (int)(dens_efs * delr2);
        ind = ind << 2;        // ! 4 * ind
        df = cgi_cgj * (f_tbl[ind] + du * f_tbl[1 + ind] + du2 * f_tbl[2 + ind] + du3 *
        f_tbl[3 + ind]);
        ...
} else {
        delr = sqrt(delr2);
        delrinv = 1.0 / delr;
        x = (*dxdr) * (delr);
        ind = (int)(*eedtbdns_stk * x);
        dx = x - (double)(ind) * (*del);
        ind = ind<<2;// ! 4 * ind
        e3dx = dx * eed_cub[2 + ind];
        e4dx2 = dx * dx * eed_cub[3 + ind];
        Switch = eed_cub[ind] + dx * (eed_cub[1 + ind] + half * (e3dx + third *
        e4dx2));
        ...
}
```

[0160]

[0161] 상기 프로그램 코드는 "if" 또는 "else" 조건이 존재하는지에 따라 각각 f_tbl로부터 또는 eed_cub로부터 4개의 요소를 로드한다. 또한, 인덱스(즉, "ind")의 계산은 루프가 벡터화됨에 따라 벡터 SIMD 레지스터 내에서 자연스럽게 나온다.

[0162] 간단한 경우를 들어, 벡터화된 루프에 대해 "if" 조건이 참이라고 가정한다. 그 경우, 8개의 인덱스 "ind"는 모두 SIMD 벡터 레지스터에 있다. 주어진 인덱스에서 4개의 요소만 로드되어야 하기 때문에 마스크는 0x00001111로 설정된다. f_tbl을 가리키는 마스크 값들 및 베이스 어드레스 레지스터는 이 8개의 로드 모두에 대해 일정하다는 것에 유의한다. 따라서, 유일한 차이는 인덱스들이다.

[0163] 대조적으로, 현재의 ISA 구현들에서, "f_tbl" 어레이의 8개의 벡터 로드를 수행하기 위해, SIMD 레지스터는 메모리에 스필되어야 하고 거기에서부터 인덱스 값들이 판독되어야 한다. 이는 비대한 코드를 초래하여, 결과적으로 범용 레지스터들(예를 들어, x86-64 GPR 레지스터들)이 다른 곳에서 사용되고 있으면 그 레지스터들의 추가 스필-필을 초래하여, 코드가 느려진다.

[0164] 벡터 인덱스 로드/저장을 수행하기 위한 본 발명의 실시예들은 로드/저장을 수행하기 위해 SIMD 레지스터 내의 인덱스 값들을 그대로 재사용함으로써 이 문제를 해결한다. 이로 인해 훨씬 더 읽기 쉽고 효율적이고 컴팩트한 SIMD 코드가 생성된다. 이 특징은 또한 수집 및 분산에 사용되는 SIMD 인덱스 레지스터와 유사하지만 이들 바닐라 로드 및 저장으로 확장된다.

[0165] 다음의 예는 본 명세서에서 설명된 본 발명의 실시예들의 이점을 강조하는 데 도움이 될 것이다:

[0166] Struct Atom [double x, y, z];

[0167] Atom AtomArray [10000];

[0168] 1. 메모리는 다음과 같이 정의된다:

[0169] Struct Table [double x, y, z,w];

[0170] Table f_tbl [10000];

[0171] 2. 계산된 인덱스들은 아래의 표 1에 나타낸 바와 같이, 본 발명의 일 실시예에 따라 ZMM 레지스터에 보유되고,

대응하는 값들은 x86-64 GPR 레지스터를 사용하는 구현을 위해 예시된다.

표 1

[0172]

GPR	r15	r14	r13	r12	r11	r10	r9	r8
ZMM	7000	6000	4000	3000	9000	5000	2000	1000

[0173] 4개의 요소 x, y, z 및 w를 로드하기 위해 마스크 K1이 0x0F로 설정된다고 가정한다.

[0174] 현재 ISA 구현들을 사용하여, 8개의 벡터 로드가 다음과 같이 레지스터들 ZMM1-8에서 수행된다:

```
VMOVDQA ZMM1 {k1}, [MEM+r8]
VMOVDQA ZMM2 {k1}, [MEM+r9]
VMOVDQA ZMM3 {k1}, [MEM+r10]
VMOVDQA ZMM4 {k1}, [MEM+r11]
VMOVDQA ZMM5 {k1}, [MEM+r12]
VMOVDQA ZMM6 {k1}, [MEM+r13]
VMOVDQA ZMM7 {k1}, [MEM+r14]
VMOVDQA ZMM8 {k1}, [MEM+r15]
```

[0175]

[0176] 벡터-인덱싱된 로드를 사용하는 본 발명의 실시예에서, 인덱스는 하나 이상의 ZMM 레지스터(예를 들어, 아래의 예에서 ZMM15)에 저장될 수 있다. 전술한 명령어 포맷을 사용하여, 대응하는 벡터 인덱싱된 로드는 다음의 명령어 시퀀스(쿼드워드(Q) 값을 가정함)로 수행될 수 있다:

```
VILDQ ZMM1 {k1}, [MEM+ZMM15], 0
VILDQ ZMM2 {k1}, [MEM+ZMM15], 1
VILDQ ZMM3 {k1}, [MEM+ZMM15], 2
VILDQ ZMM4 {k1}, [MEM+ZMM15], 3
VILDQ ZMM5 {k1}, [MEM+ZMM15], 4
VILDQ ZMM6 {k1}, [MEM+ZMM15], 5
VILDQ ZMM7 {k1}, [MEM+ZMM15], 6
VILDQ ZMM8 {k1}, [MEM+ZMM15], 7
```

[0177]

[0178] 유사한 명령어 시퀀스가 VIST를 사용하는 저장 연산을 위해 수행될 수 있다. 상기 VILD 명령어(및 VIST 명령어)의 경우, 인덱스 값을 검색할 때 GPR 레지스터 필요성이 수반되지 않는다.

[0179] 전술한 명세서에서는, 본 발명의 실시예들이 그의 특정한 예시적인 실시예들을 참조하여 설명되었다. 그러나, 첨부된 청구항들에서 제시된 바와 같은 본 발명의 더 넓은 사상 및 범위를 벗어나지 않고 그에 대해 다양한 수정 및 변경들이 행해질 수 있다는 것은 명백할 것이다. 따라서, 명세서 및 도면은 제한적인 의미가 아닌 예시적인 의미로 간주되어야 한다.

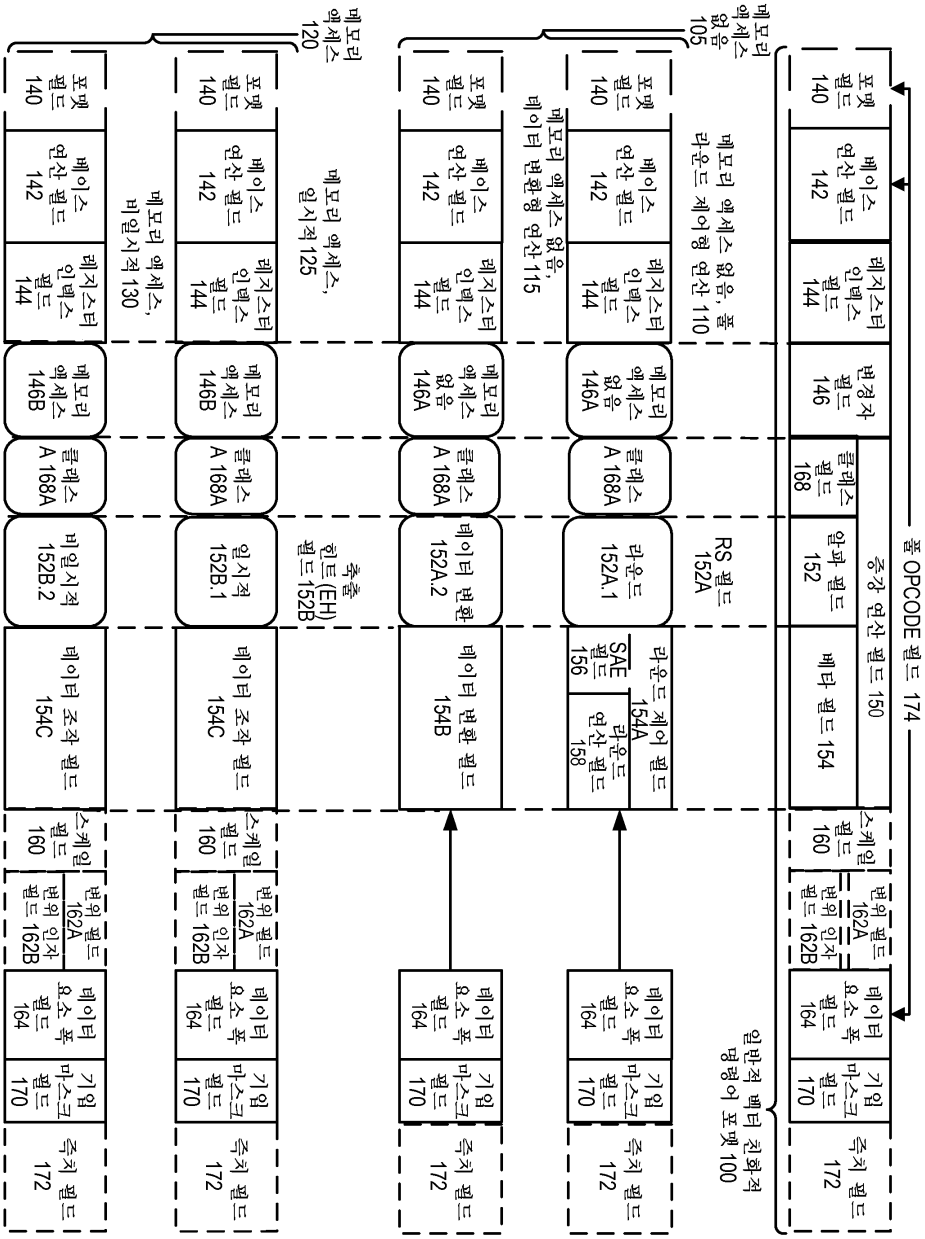
[0180] 본 발명의 실시예들은 전술한 다양한 단계들을 포함할 수 있다. 이들 단계들은, 범용 또는 특수 목적 프로세서가 이들 단계들을 수행하게 하는 데 사용될 수 있는 머신 실행가능 명령어들로 구현될 수 있다. 대안적으로, 이들 단계들은, 그 단계들을 수행하기 위한 하드와이어드 로직을 포함하는 특정 하드웨어 컴포넌트들에 의해, 또는 프로그램된 컴퓨터 컴포넌트들과 사용자 지정 하드웨어 컴포넌트들의 임의의 조합에 의해 수행될 수 있다.

[0181] 본 명세서에서 설명된 바와 같은, 명령어들은 비일시적인 컴퓨터 판독가능 매체로 구현된 메모리에 저장되는 소프트웨어 명령어들, 또는 미리 결정된 기능성을 갖거나 특정 연산들을 수행하도록 구성된 주문형 집적 회로(ASIC)들과 같은 하드웨어의 특정 구성들을 언급할 수 있다. 따라서, 도면들에 도시된 기법들은, 하나 이상의

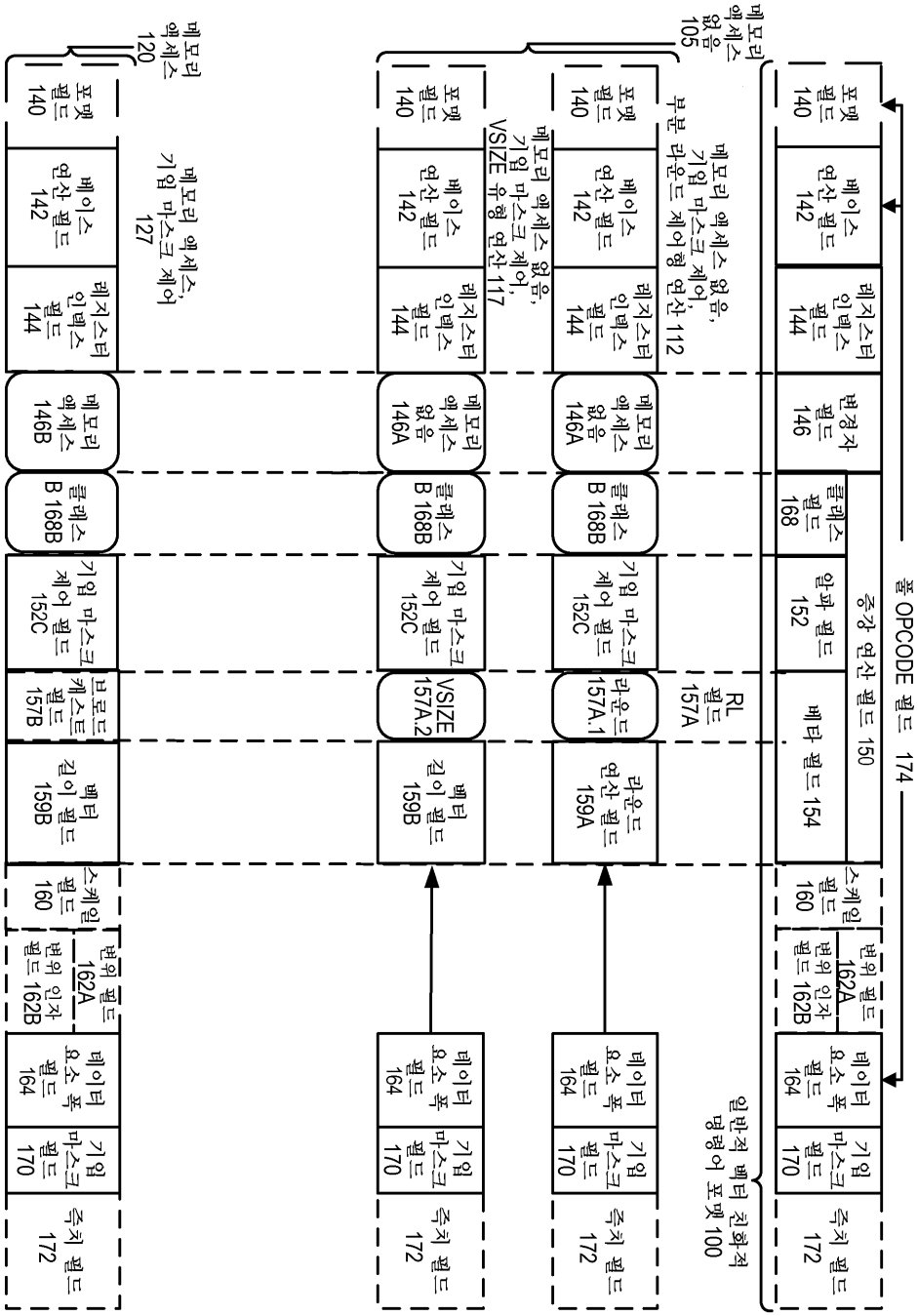
전자 디바이스(예를 들어, 종단국, 네트워크 요소 등) 상에 저장되어 실행되는 데이터 및 코드를 사용하여 구현될 수 있다. 이러한 전자 디바이스들은, 비일시적인 컴퓨터 머신 판독가능 저장 매체(예를 들어, 자기 디스크; 광학 디스크; 랜덤 액세스 메모리; 판독 전용 메모리; 플래시 메모리 디바이스; 상변화 메모리) 및 일시적인 컴퓨터 머신 판독가능 통신 매체(예를 들어, 전기, 광학, 음향 또는 다른 형태의 전파 신호 - 예를 들어, 반송파, 적외선 신호, 디지털 신호 등)와 같은 컴퓨터 머신 판독가능 매체를 사용하여 코드 및 데이터를 저장 및 통신한다(내부적으로 그리고/또는 네트워크를 통해 다른 전자 디바이스들과). 또한, 이러한 전자 디바이스들은, 하나 이상의 저장 디바이스(비일시적인 머신 판독가능 저장 매체), 사용자 입력/출력 디바이스(예를 들어, 키보드, 터치스크린 및/또는 디스플레이) 및 네트워크 접속과 같은 하나 이상의 다른 컴포넌트에 연결된 하나 이상의 프로세서의 세트를 전형적으로 포함한다. 프로세서들의 세트와 다른 컴포넌트들의 연결은 전형적으로 하나 이상의 버스 및 브리지(버스 제어기라고도 불림)를 통해 이루어진다. 저장 디바이스, 및 네트워크 트래픽을 반송하는 신호들은 하나 이상의 머신 판독가능 저장 매체 및 머신 판독가능 통신 매체를 각각 나타낸다. 따라서, 주어진 전자 디바이스의 저장 디바이스는 전형적으로 그 전자 디바이스의 하나 이상의 프로세서의 세트 상에서 실행될 코드 및/또는 데이터를 저장한다. 물론, 본 발명의 실시예의 하나 이상의 부분은 소프트웨어, 펌웨어 및/또는 하드웨어의 상이한 조합들을 사용하여 구현될 수 있다. 이 상세한 설명 전체에 걸쳐, 설명의 목적으로, 본 발명의 철저한 이해를 제공하기 위해서 다수의 특정 상세가 제시되었다. 그러나, 본 발명은 이들 특정 상세 중 일부가 없이도 실시될 수 있다는 것이 본 기술분야의 통상의 기술자에게 명백할 것이다. 특정 경우에, 본 발명의 주제를 모호하게 하는 것을 회피하기 위해서 잘 알려진 구조들 및 기능들은 상세하게 설명되지 않았다. 따라서, 본 발명의 범위 및 사상은 이하의 청구항들에 관하여 판단되어야 한다.

도면

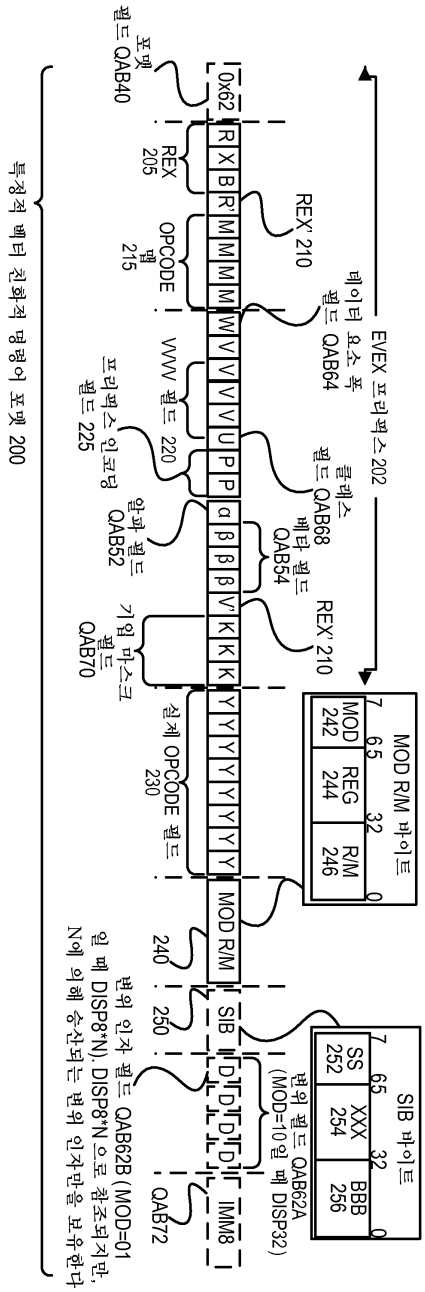
도면1a


메모리 액세스 없음 105
메모리 액세스 없음, 폴 라운드 제어형 연산 110
메모리 액세스 없음, 데이터 변환형 연산 115
메모리 액세스 120
메모리 액세스, 일시적 125
메모리 액세스, 비일시적 130
일반적 벡터 친화적 명령어 포맷 100
폴 OPCODE 필드 174
증강 연산 필드 150
포맷 필드 140
베이스 연산 필드 142
레지스터 인덱스 필드 144
변경자 필드 146
메모리 액세스 없음 146A
메모리 액세스 146B
클래스 필드 168
클래스 A 168A
알파 필드 152
베타 필드 154
RS 필드 152A
라운드 152A.1
데이터 변환 152A.2
일시적 152B.1
비일시적 152B.2
힌트(EH) 필드 152B
라운드 제어 필드 154A
SAE 필드 156
라운드 연산 필드 158
데이터 변환 필드 154B
데이터 조작 필드 154C
스케일 필드 160
변위 필드 162A
변위 인자 필드 162B
데이터 요소 폭 필드 164
기입 마스크 필드 170
즉치 필드 172

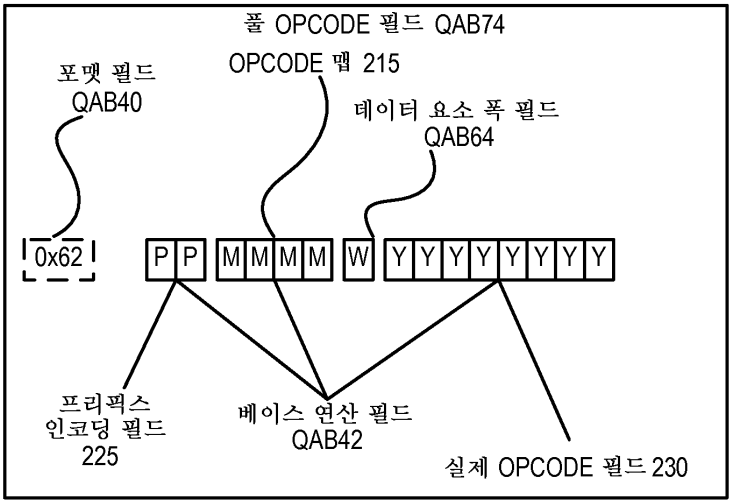
도면1b


메모리 액세스 없음 105
메모리 액세스 없음, 기입 마스크 제어형 연산 112
부분 라운드
메모리 액세스 없음, 기입 마스크 제어, VSIZE 유형 연산 117
메모리 액세스 120
메모리 액세스, 기입 마스크 제어 127
포맷 필드 140
베이스 연산 필드 142
레지스터 인덱스 필드 144
변경자 필드 146
메모리 액세스 없음 146A
메모리 액세스 146B
클래스 B 168B
클래스 필드 168
증강 연산 필드 150
알파 필드 152
기입 마스크 제어 필드 152C
베타 필드 154
RL 필드 157A
라운드 연산 필드 159A
VSIZE 157A.2
라운드 157A.1
브로드캐스트 필드 157B
벡터 길이 필드 159B
스케일 필드 160
변위 필드 162A
변위 인자 필드 162B
데이터 요소 폭 필드 164
기입 마스크 필드 170
즉치 필드 172
풀 OPCODE 필드 174
일반적 벡터 친화적 명령어 포맷 100

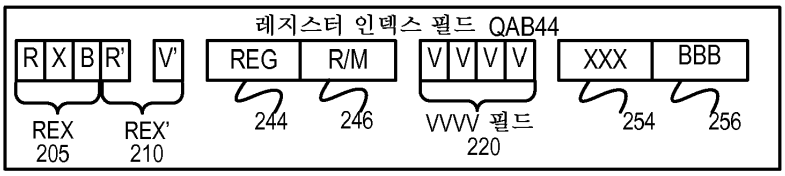
도면2a


특정적 벡터 친화적 명령어 포맷 200
EVEX 프리픽스 202
0x62
R X B R' M M M M W V V V V U P P
α β β β V' K K K
Y Y Y Y Y Y Y Y
MOD R/M
SIB
D D D D IMM8
REX 205
OPCODE 맵 215
REX' 210
데이터 요소 폭 QAB64
VVVV 필드 220
클래스 QAB68
베타 필드 QAB54
REX' 210
프리픽스 인코딩 필드 225
알파 필드 QAB52
기입 마스크 필드 QAB70
실제 OPCODE 필드 230
240
250
QAB72
포맷 필드 QAB40
MOD R/M 바이트
MOD 242
REG 244
R/M 246
7 6 5 3 2 0
SIB 바이트
SS 252
XXX 254
BBB 256
변위 필드 QAB62A (MOD=10일 때 DISP32)
일 때 DISP8*N), DISP8*N으로 참조되지만, N에 의해 승산되는 변위 인자만을 보유한다
변위 인자 필드 QAB62B (MOD=01

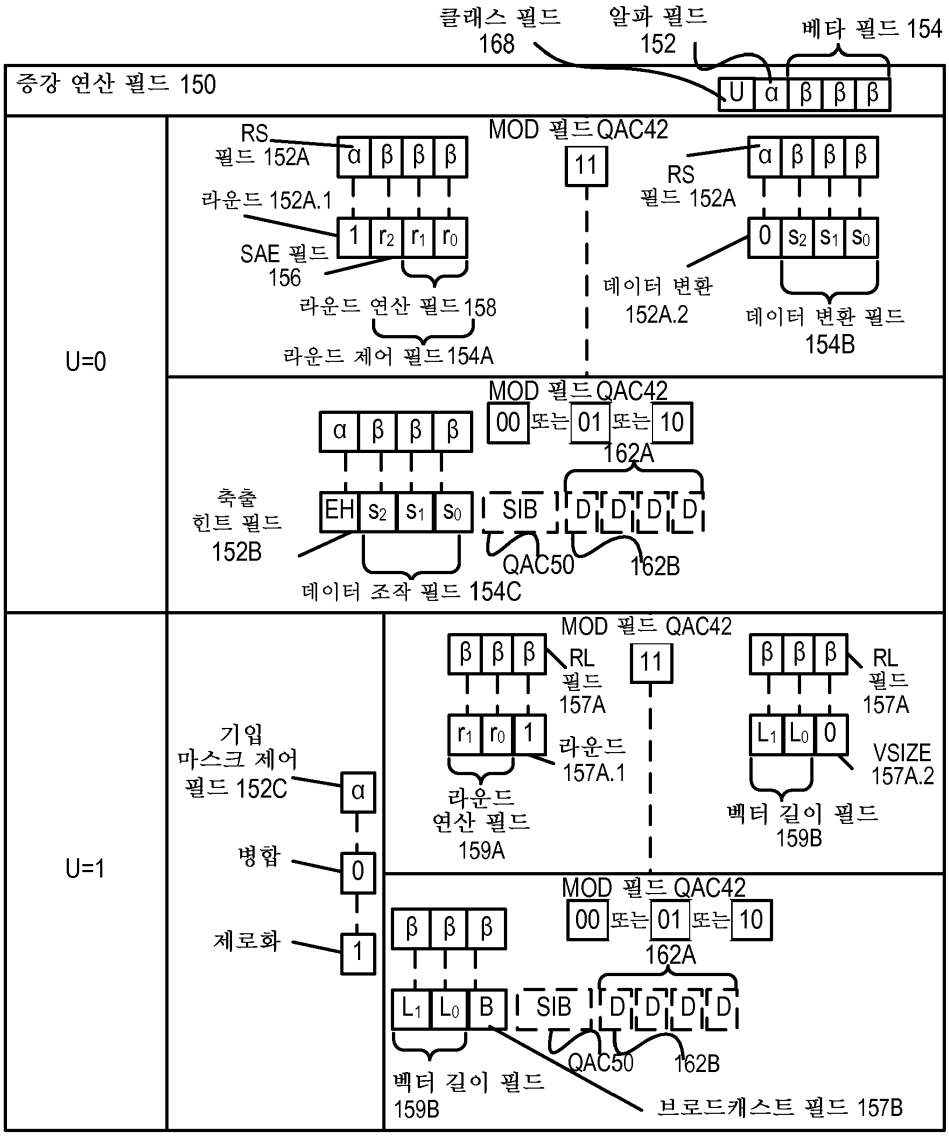
도면2b


풀 OPCODE 필드 QAB74
포맷 필드
QAB40
OPCODE 맵 215
데이터 요소 폭 필드
QAB64
0x62
P P
M M M M
W
Y Y Y Y Y Y Y Y
프리픽스
인코딩 필드
225
베이스 연산 필드
QAB42
실제 OPCODE 필드 230

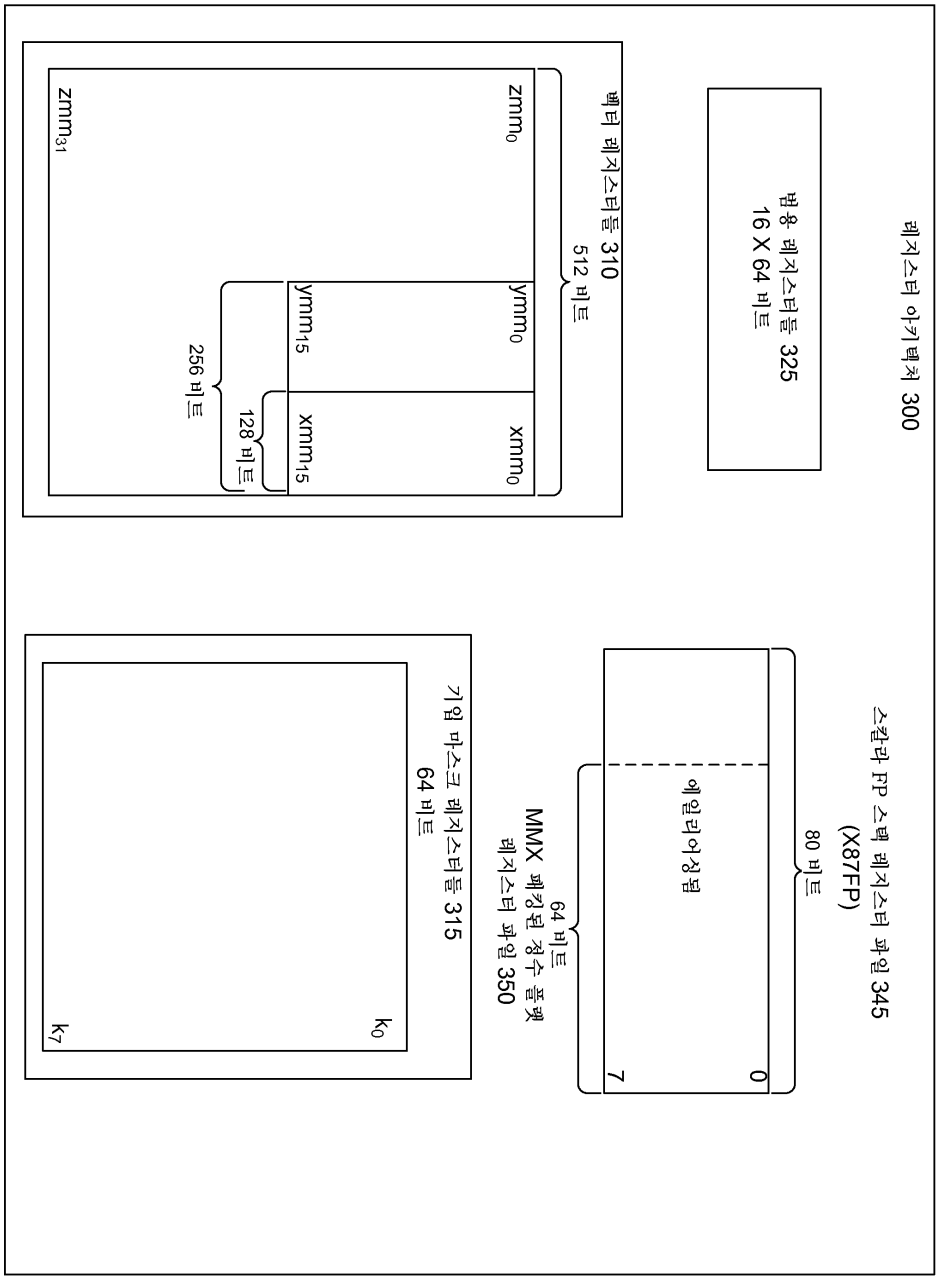
도면2c


레지스터 인덱스 필드 QAB44
R X B R'
V'
REG
R/M
V V V V
XXX
BBB
REX
205
REX'
210
244
246
VVVV 필드
220
254
256

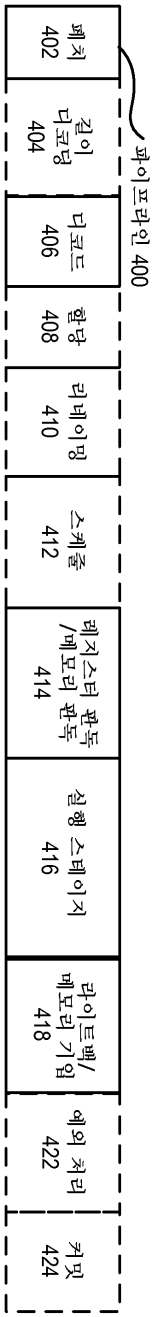
도면2d


클래스 필드 168
알파 필드 152
베타 필드 154
증강 연산 필드 150
U α β β β
U=0
RS 필드 152A
α β β β
라운드 152A.1
1 r2 r1 r0
SAE 필드 156
라운드 연산 필드 158
라운드 제어 필드 154A
MOD 필드 QAC42
11
RS 필드 152A
α β β β
0 s2 s1 s0
데이터 변환 152A.2
데이터 변환 필드 154B
MOD 필드 QAC42
00 또는 01 또는 10
162A
α β β β
축출 힌트 필드 152B
EH s2 s1 s0
SIB
D D D D
QAC50
162B
데이터 조작 필드 154C
U=1
기입 마스크 제어 필드 152C
α
병합
0
제로화
1
MOD 필드 QAC42
β β β
RL 필드 157A
11
r1 r0 1
라운드 157A.1
라운드 연산 필드 159A
β β β
RL 필드 157A
L1 L0 0
VSIZE 157A.2
벡터 길이 필드 159B
MOD 필드 QAC42
00 또는 01 또는 10
162A
β β β
L1 L0 B
SIB
D D D D
QAC50
162B
벡터 길이 필드 159B
브로드캐스트 필드 157B

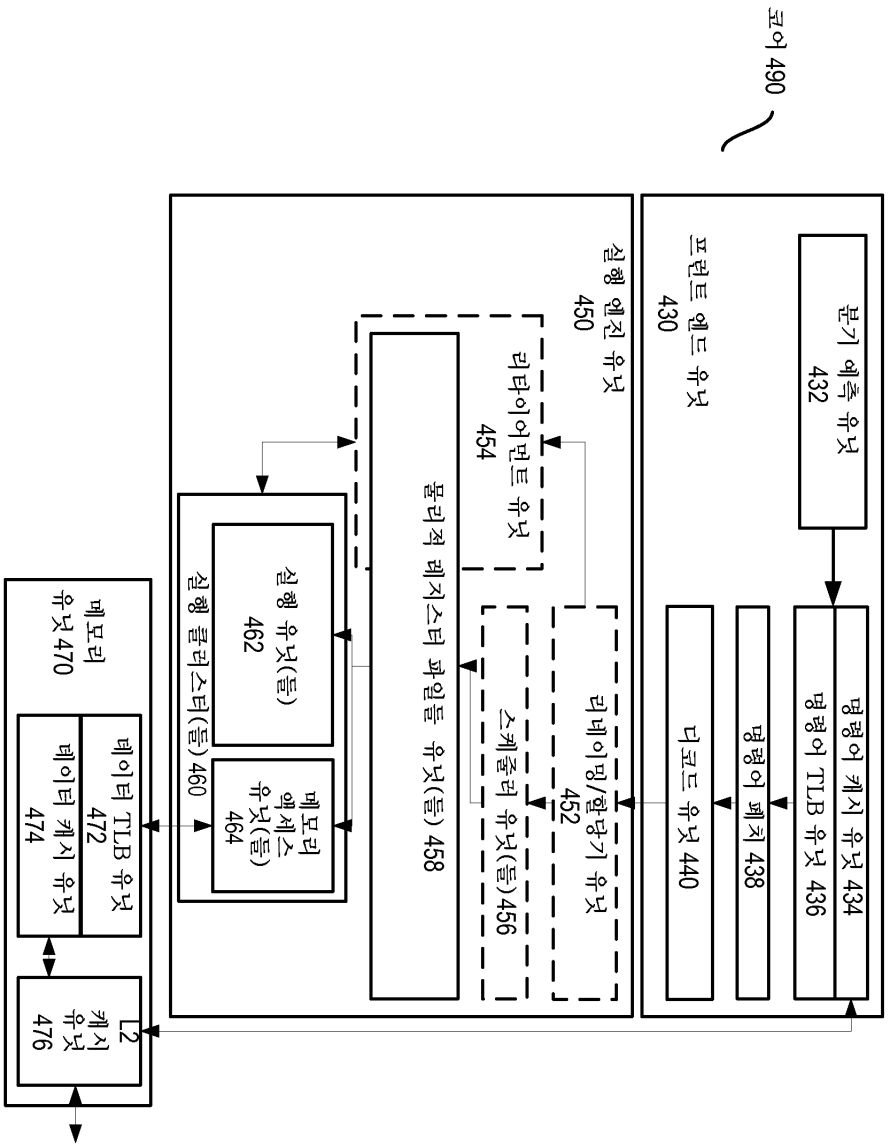
도면3


레지스터 아키텍처 300
범용 레지스터들 325
16 X 64 비트
벡터 레지스터들 310
512 비트
zmm0
zmm31
ymm0
ymm15
xmm0
xmm15
256 비트
128 비트
스칼라 FP 스택 레지스터 파일 345
(X87FP)
80 비트
0
7
에일리어싱 됨
64 비트
MMX 패킹된 정수 플랫
레지스터 파일 350
기입 마스크 레지스터들 315
64 비트
k0
k7

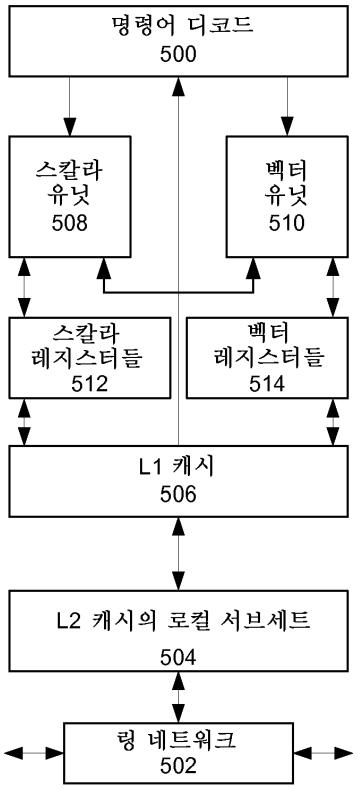
도면4a


파이프라인 400
페치 402
길이 디코딩 404
디코드 406
할당 408
리네이밍 410
스케줄 412
레지스터 판독 /메모리 판독 414
실행 스테이지 416
라이트백/ 메모리 기입 418
예외 처리 422
커밋 424

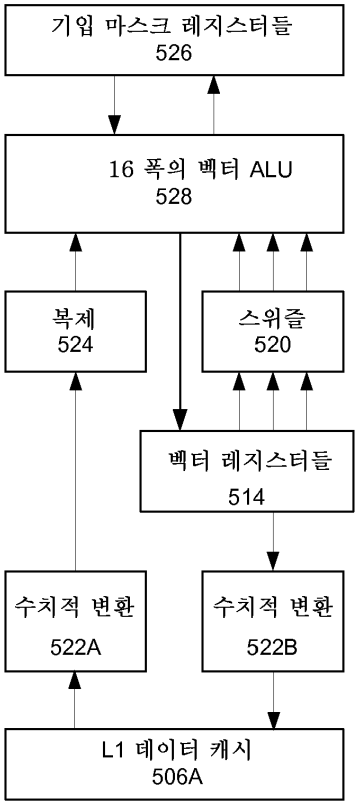
도면4b


코어 490
분기 예측 유닛 432
명령어 캐시 유닛 434
명령어 TLB 유닛 436
명령어 페치 438
디코드 유닛 440
프런트 엔드 유닛 430
리네이밍/할당기 유닛 452
스케줄러 유닛(들)456
물리적 레지스터 파일들 유닛(들) 458
리타이어먼트 유닛 454
실행 엔진 유닛 450
실행 유닛(들) 462
메모리 액세스 유닛(들) 464
실행 클러스터(들) 460
메모리 유닛 470
데이터 TLB 유닛 472
데이터 캐시 유닛 474
L2 캐시 유닛 476

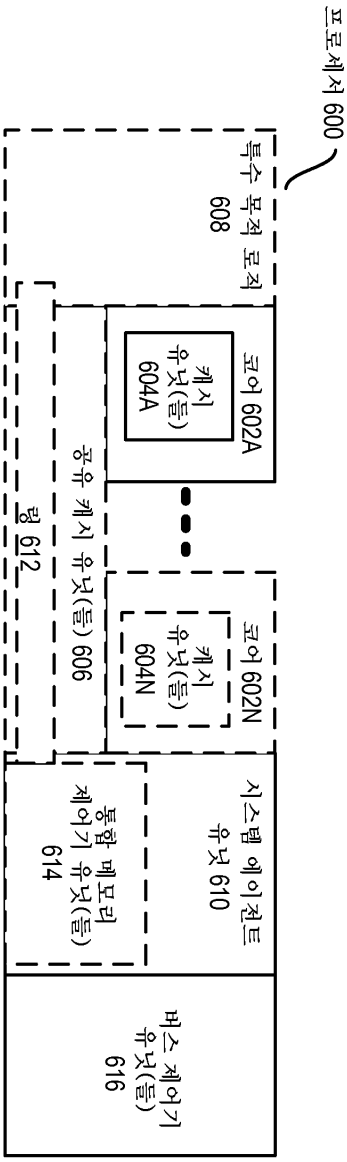
도면5a


명령어 디코드
500
스칼라
유닛
508
벡터
유닛
510
스칼라
레지스터들
512
벡터
레지스터들
514
L1 캐시
506
L2 캐시의 로컬 서브세트
504
링 네트워크
502

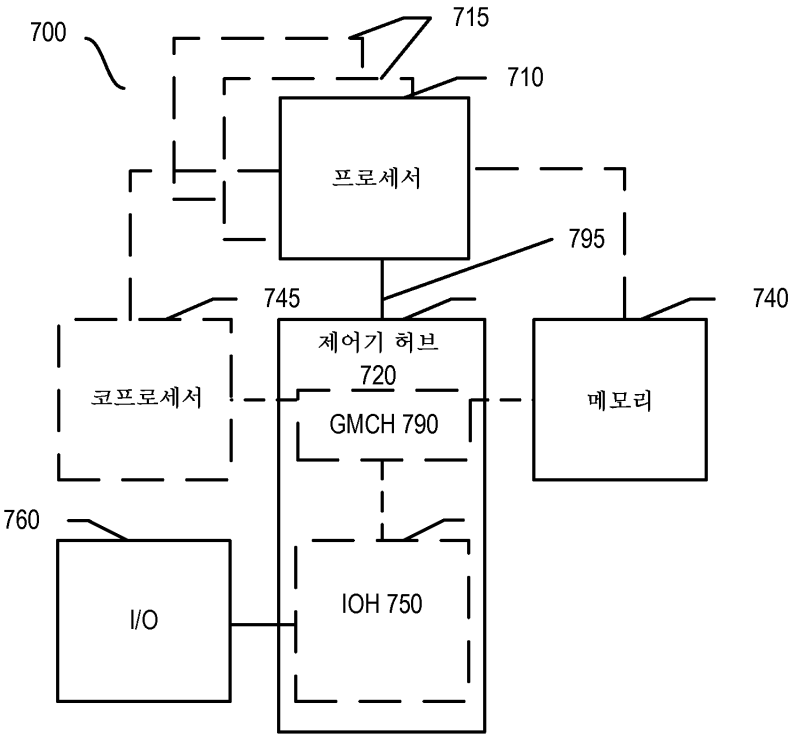
도면5b


기입 마스크 레지스터들
526
16 폭의 벡터 ALU
528
복제
524
스위즐
520
벡터 레지스터들
514
수치적 변환
522A
수치적 변환
522B
L1 데이터 캐시
506A

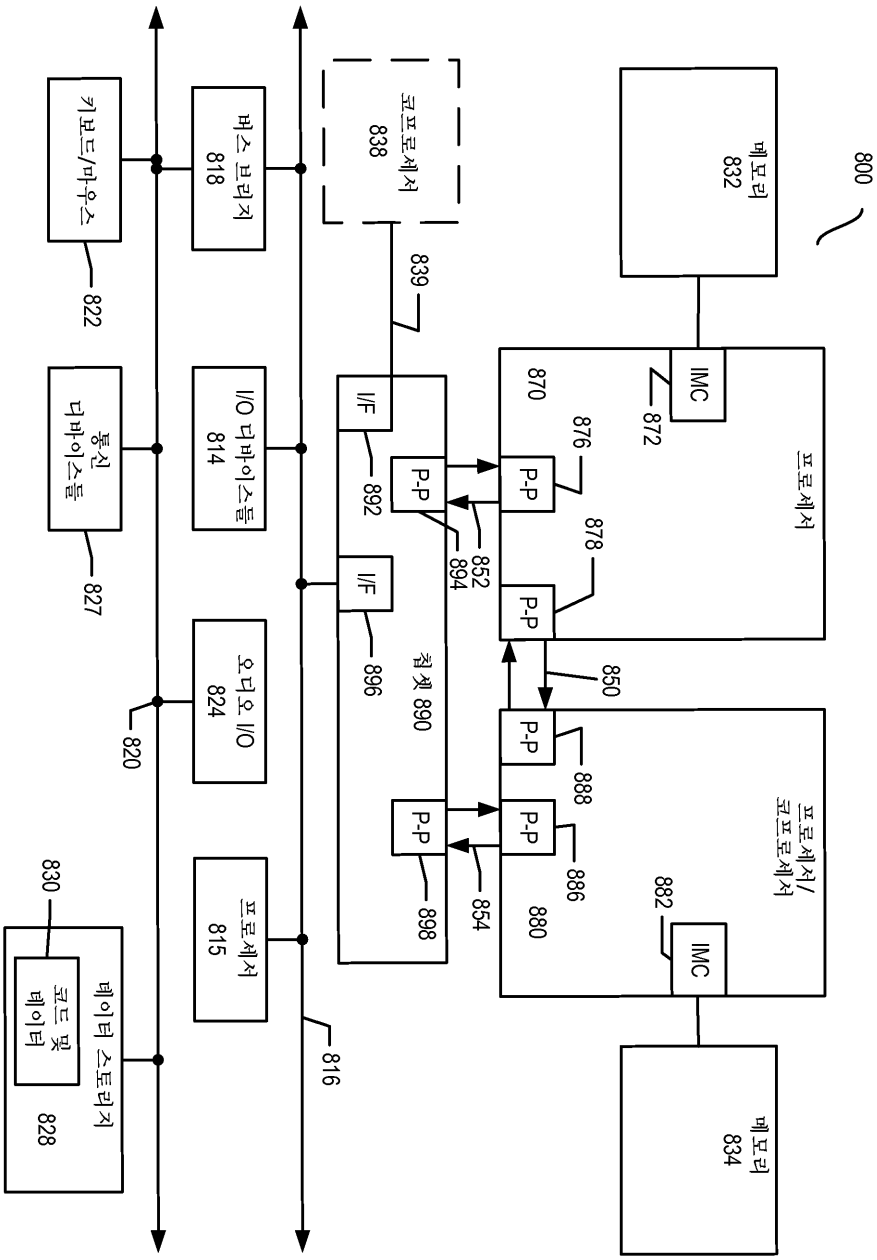
도면6


프로세서 600
특수 목적 로직 608
코어 602A
캐시 유닛(들) 604A
코어 602N
캐시 유닛(들) 604N
공유 캐시 유닛(들) 606
링 612
시스템 에이전트 유닛 610
통합 메모리 제어기 유닛(들) 614
버스 제어기 유닛(들) 616

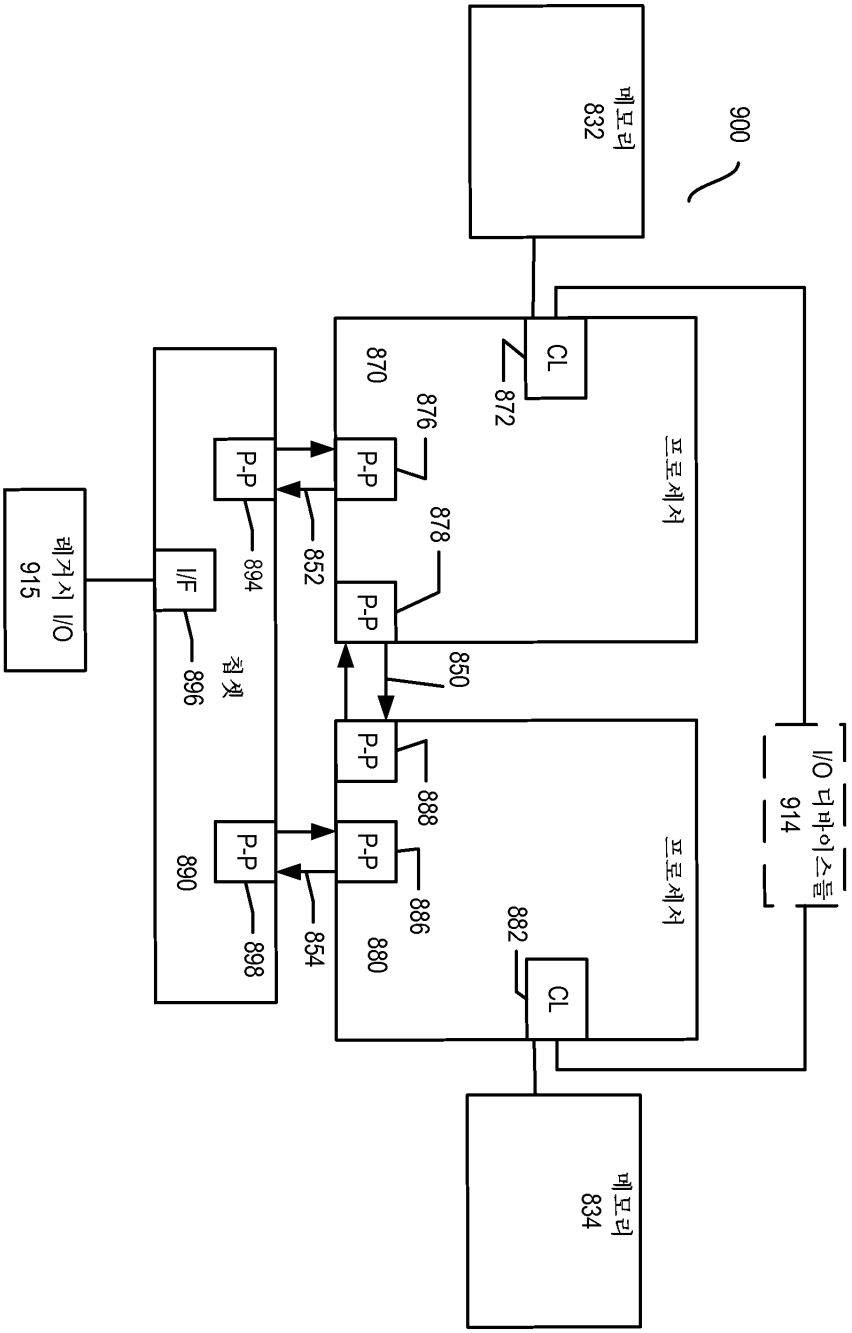
도면7


700
715
710
프로세서
795
745
740
제어기 허브
720
코프로세서
GMCH 790
메모리
760
I/O
IOH 750

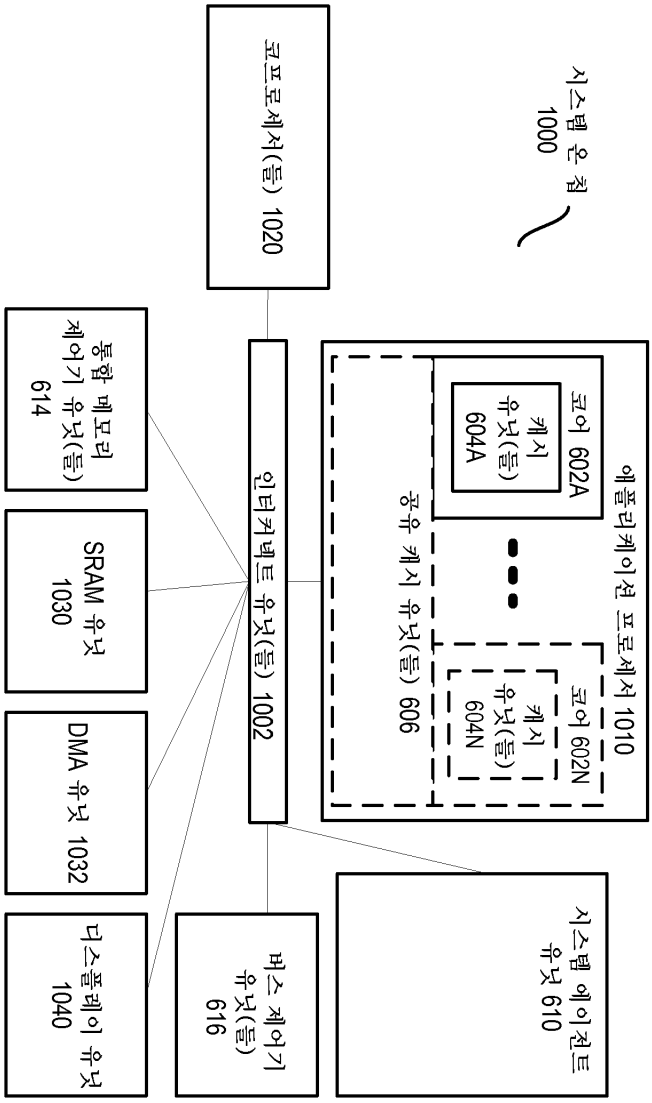
도면8


800
메모리
832
IMC
872
프로세서
870
876
P-P
878
P-P
850
P-P
888
P-P
886
프로세서/
코프로세서
882
880
IMC
메모리
834
852
894
854
P-P
P-P
898
I/F
892
I/F
896
칩셋 890
839
코프로세서
838
816
버스 브리지
818
I/O 디바이스들
814
오디오 I/O
824
820
프로세서
815
키보드/마우스
822
통신
디바이스들
827
데이터 스토리지
828
코드 및
데이터
830

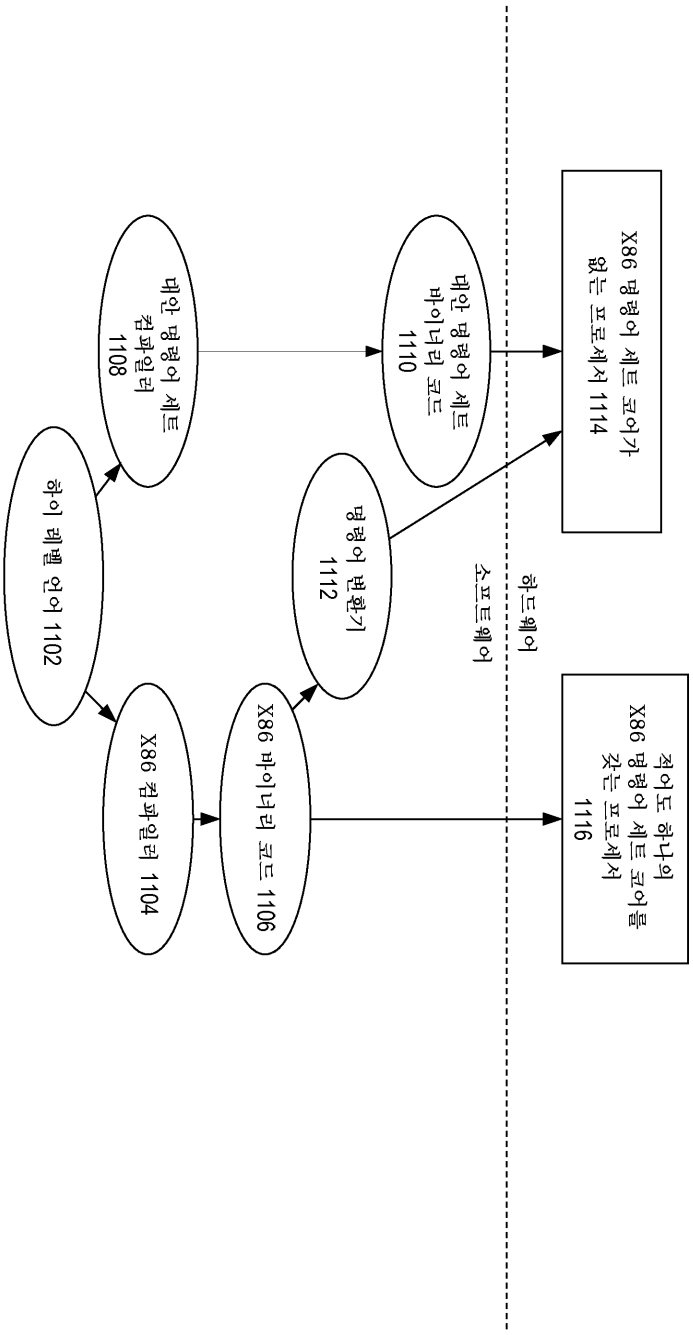
도면9


900
메모리
832
CL
872
870
876
878
프로세서
P-P
P-P
P-P
852
850
894
I/F
896
칩셋
레거시 I/O
915
P-P
P-P
P-P
888
886
854
898
890
880
882
CL
프로세서
I/O 디바이스들
914
메모리
834

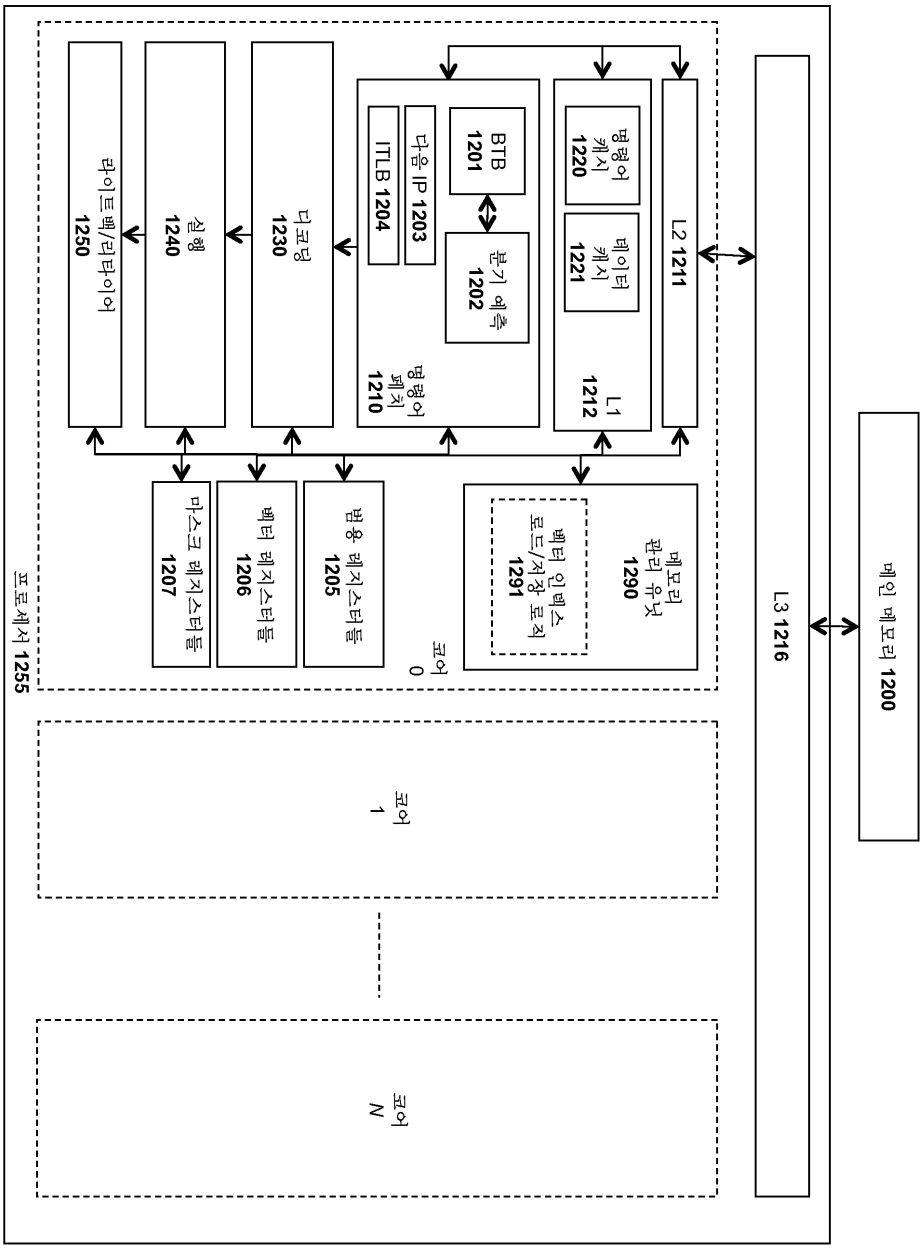
도면10


시스템 온 칩
1000
코프로세서(들) 1020
통합 메모리 제어기 유닛(들) 614
SRAM 유닛 1030
DMA 유닛 1032
디스플레이 유닛 1040
인터커넥트 유닛(들) 1002
버스 제어기 유닛(들) 616
시스템 에이전트 유닛 610
애플리케이션 프로세서 1010
코어 602A
캐시 유닛(들) 604A
코어 602N
캐시 유닛(들) 604N
공유 캐시 유닛(들) 606

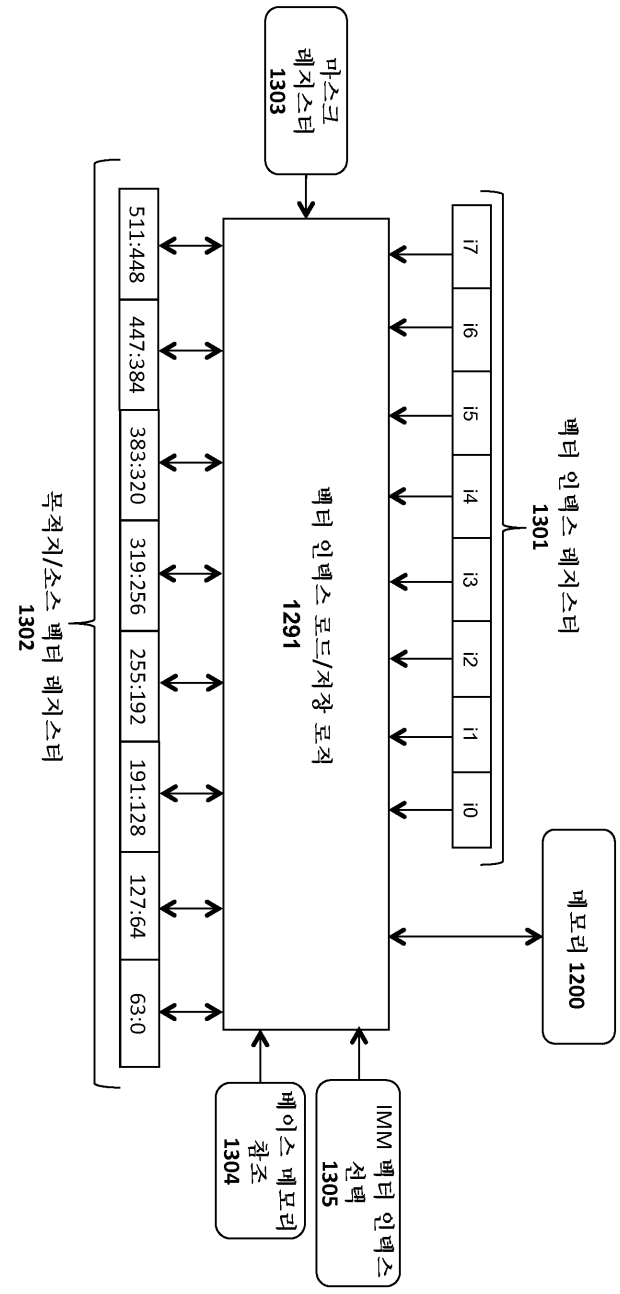
도면11


하이 레벨 언어 1102
대안 명령어 세트 컴파일러 1108
X86 컴파일러 1104
X86 바이너리 코드 1106
명령어 변환기 1112
대안 명령어 세트 바이너리 코드 1110
X86 명령어 세트 코어가 없는 프로세서 1114
적어도 하나의 X86 명령어 세트 코어를 갖는 프로세서 1116
소프트웨어
하드웨어

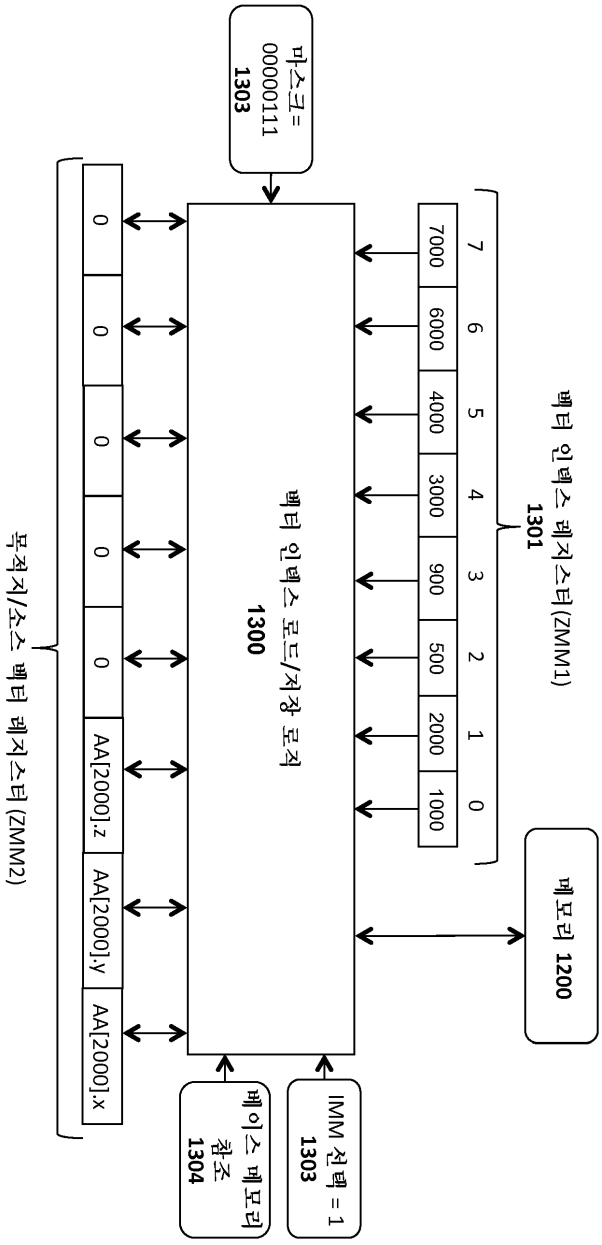
도면12


메인 메모리 1200
L3 1216
프로세서 1255
코어 0
명령어 캐시 1220
데이터 캐시 1221
L1 1212
L2 1211
명령어 페치 1210
BTB 1201
분기 예측 1202
다음 IP 1203
ITLB 1204
디코딩 1230
실행 1240
라이트백/리타이어 1250
메모리 관리 유닛 1290
벡터 인덱스 로드/저장 로직 1291
범용 레지스터들 1205
벡터 레지스터들 1206
마스크 레지스터들 1207
코어 1
코어 N

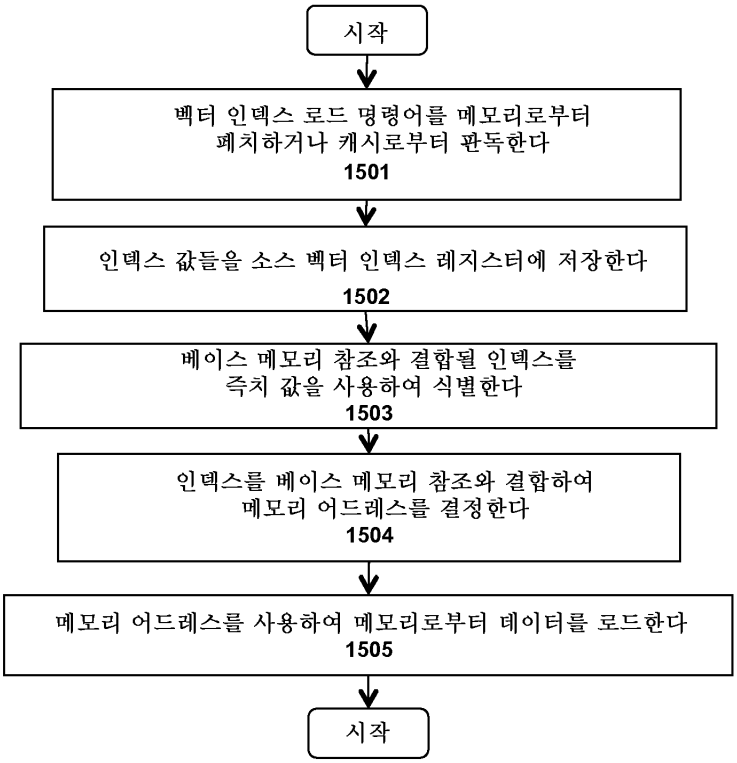
도면13


마스크 레지스터 1303
벡터 인덱스 로드/저장 로직 1291
벡터 인덱스 레지스터 1301
i7
i6
i5
i4
i3
i2
i1
i0
목적지/소스 벡터 레지스터 1302
511:448
447:384
383:320
319:256
255:192
191:128
127:64
63:0
메모리 1200
베이스 메모리 참조 1304
IMM 벡터 인덱스 선택 1305

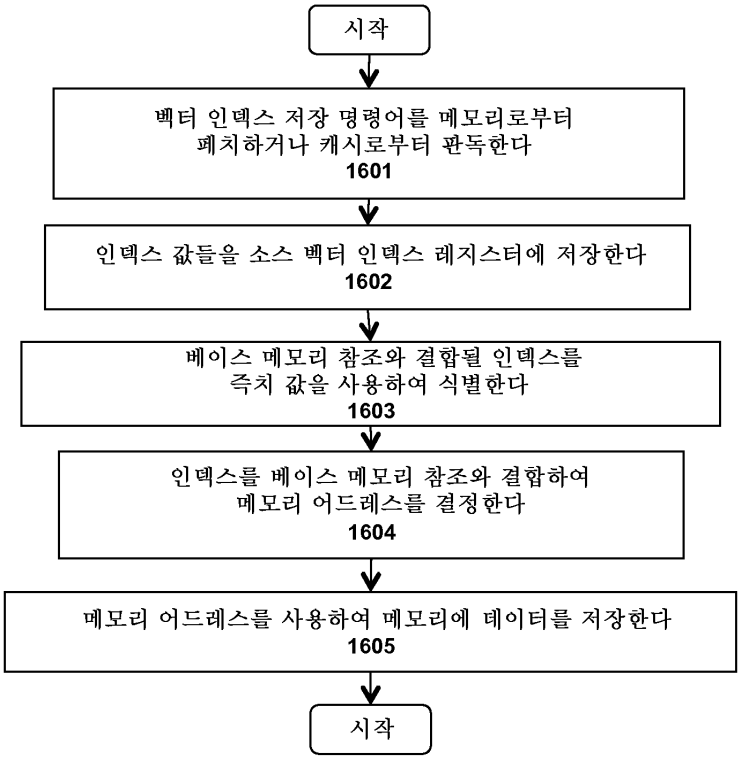
도면14


마스크=
00000111
1303
벡터 인덱스 레지스터(ZMM1)
1301
7
6
5
4
3
2
1
0
7000
6000
4000
3000
900
500
2000
1000
벡터 인덱스 로드/저장 로직
1300
메모리 1200
목적지/소스 벡터 레지스터 (ZMM2)
1302
0
0
0
0
0
AA[2000].z
AA[2000].y
AA[2000].x
베이스 메모리
참조
1304
IMM 선택 = 1
1303

도면15


시작
벡터 인덱스 로드 명령어를 메모리로부터
페치하거나 캐시로부터 판독한다
1501
인덱스 값들을 소스 벡터 인덱스 레지스터에 저장한다
1502
베이스 메모리 참조와 결합될 인덱스를
즉치 값을 사용하여 식별한다
1503
인덱스를 베이스 메모리 참조와 결합하여
메모리 어드레스를 결정한다
1504
메모리 어드레스를 사용하여 메모리로부터 데이터를 로드한다
1505
시작

도면16


시작
벡터 인덱스 저장 명령어를 메모리로부터
페치하거나 캐시로부터 판독한다
1601
인덱스 값들을 소스 벡터 인덱스 레지스터에 저장한다
1602
베이스 메모리 참조와 결합될 인덱스를
즉치 값을 사용하여 식별한다
1603
인덱스를 베이스 메모리 참조와 결합하여
메모리 어드레스를 결정한다
1604
메모리 어드레스를 사용하여 메모리에 데이터를 저장한다
1605
시작