(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2015-0038328
(43) 공개일자 2015년04월08일

(51) 국제특허분류(Int. Cl.)
G06F 9/30 (2006.01)
(52) CPC특허분류
G06F 9/30032 (2013.01)
G06F 9/30018 (2013.01)
(21) 출원번호 10-2015-7004840
(22) 출원일자(국제) 2013년06월25일
심사청구일자 2015년02월25일
(85) 번역문제출일자 2015년02월25일
(86) 국제출원번호 PCT/US2013/047669
(87) 국제공개번호 WO 2014/051782
국제공개일자 2014년04월03일
(30) 우선권주장
13/630,131 2012년09월28일 미국(US)

(71) 출원인
인텔 코포레이션
미합중국 캘리포니아 95052 산타클라라 미션 칼리지 블러바드 2200
(72) 발명자
플로트니코브, 미하일
러시아 603024 니즈 니즈니 노브고로드 투르제네바 에스티알. 30
에르몰라에브, 이고르
러시아 603024 니즈 니즈니 노브고로드 67 프루크토바자 5/3
(뒷면에 계속)
(74) 대리인
양영준, 백만기

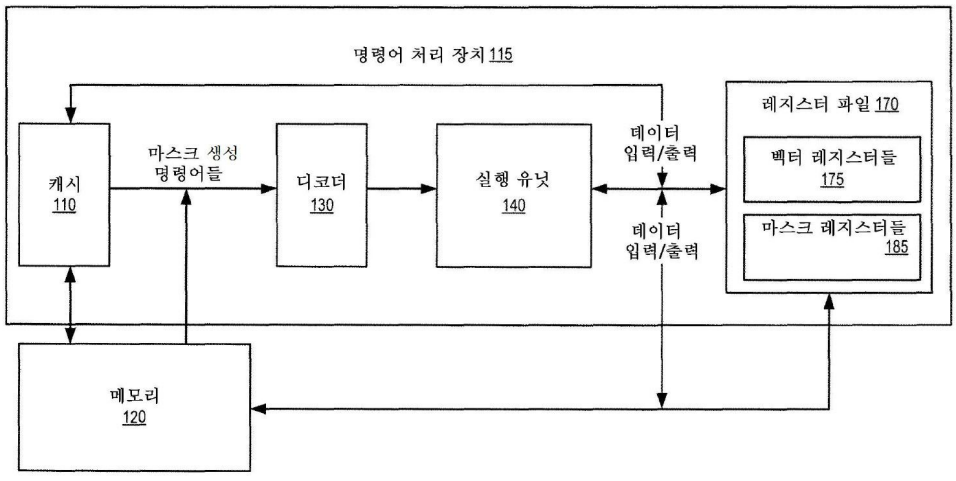
전체 청구항 수 : 총 20 항

(54) 발명의 명칭 **1들을 최하위 비트들이 되도록 풀링하면서 비트들을 좌측으로 시프팅하기 위한 명령어**

(57) 요 약

마스크 생성 명령어가 데이터 성분들의 어레이에 대한 벡터 연산들의 효율성을 향상시키기 위해 프로세서에 의해 실행된다. 프로세서는 벡터 레지스터들을 포함하는데, 벡터 레지스터들 중 하나가 어레이의 데이터 성분들을 저장한다. 프로세서는 적어도 제1 피연산자 및 제2 피연산자를 특정하는 마스크 생성 명령어를 수신하는 실행 회로를 더 포함한다. 마스크 생성 명령어에 응답하여, 실행 회로는 제2 피연산자에서 정의되는 횟수만큼 제1 피연산자의 비트들을 좌측으로 시프팅하고, 및 제1 피연산자의 최상위 비트가 좌측으로부터 시프팅 아웃될 때마다 1의 비트를 우측으로부터 풀인하여 결과를 발생한다. 결과에서의 각각의 비트는 어레이의 데이터 성분들 중 하나에 대응한다.

대 표 도



(52) CPC특허분류
G06F 9/30036 (2013.01)
(72) 발명자
나라이킨, 안드레이
러시아 603024 니즈 니즈니 노브고로드 투르제네바 에스티알. 30
발렌타인, 로버트
이스라엘 에이치에이 36054 키르야트 티본 레초브 하드가니오트 33-5

명 세 서

청구범위

청구항 1

장치로서:

복수의 벡터 레지스터 - 상기 복수의 벡터 레지스터 중 하나가 어레이의 데이터 성분들을 저장함 -; 및

상기 복수의 벡터 레지스터에 결합되는 실행 회로

를 포함하고,

상기 실행 회로는,

적어도 제1 피연산자(operand) 및 제2 피연산자를 특정하는 마스크 생성 명령어를 수신하고,

상기 마스크 생성 명령어에 응답하여, 상기 제2 피연산자에서 정의되는 횟수만큼 상기 제1 피연산자의 비트들을 좌측 시프팅시키고, 상기 제1 피연산자의 최상위 비트가 시프팅 아웃(shift out)될 때마다 1의 최하위 비트를 풀인(pull in)하고 그에 의해 복수의 비트를 포함하는 결과를 발생시키고, 상기 결과에서의 각각의 비트는 상기 어레이의 데이터 성분들 중 하나에 대응하는 장치.

청구항 2

제1항에 있어서, 상기 제2 피연산자는 벡터 연산의 나머지 루프에서 나머지 반복들의 횟수를 특정하는 장치.

청구항 3

제2항에 있어서, 상기 제2 피연산자는 루프 한계에서 상기 벡터 연산에 대한 현재 반복 카운트를 뺀 감산 결과를 특정하는 장치.

청구항 4

제1항에 있어서, 상기 복수의 벡터 레지스터는 제1 벡터 레지스터 및 제2 벡터 레지스터를 포함하고, 상기 제2 피연산자는 벡터 계산을 위해 상기 제1 벡터 레지스터에서의 기존 데이터 성분들 내로 통합될 상기 제2 벡터 레지스터에서의 데이터 성분들의 수를 특정하는 장치.

청구항 5

제1항에 있어서, 상기 제1 피연산자 및 상기 제2 피연산자 모두는 범용 레지스터들인 장치.

청구항 6

제1항에 있어서, 상기 제1 피연산자는 마스크 레지스터이고 상기 제2 피연산자는 범용 레지스터인 장치.

청구항 7

제1항에 있어서, 하나 이상의 상태 레지스터들이 상기 결과에 기초해 설정되는 장치.

청구항 8

방법으로서:

프로세서에 의해, 적어도 제1 피연산자 및 제2 피연산자를 특정하는 마스크 생성 명령어를 수신하는 단계; 및

상기 마스크 생성 명령어에 응답하여,

상기 제2 피연산자에서 정의되는 횟수만큼 상기 제1 피연산자의 비트들을 좌측 시프팅시키는 연산, 및

상기 제1 피연산자의 최상위 비트가 시프팅 아웃될 때마다 1의 최하위 비트를 풀인하고 그에 의해 복수의 비트를 포함하는 결과를 발생시키는 연산 - 상기 결과에서의 각각의 비트는 어레이의 데이터 성분에 대응함

- 을 실행하는 단계

를 포함하는 방법.

청구항 9

제7항에 있어서, 상기 제2 피연산자는 벡터 연산의 나머지 루프에서의 나머지 반복들의 횟수를 특정하는 방법.

청구항 10

제9항에 있어서, 상기 제2 피연산자는 루프 한계에서 상기 벡터 연산에 대한 현재 반복 카운트를 뺀 감산 결과를 특정하는 방법.

청구항 11

제7항에 있어서, 상기 제2 피연산자는 벡터 계산을 위해 제1 벡터 레지스터에서의 기존 데이터 성분들 내로 통합될 제2 벡터 레지스터에서의 데이터 성분들의 수를 특정하는 방법.

청구항 12

제7항에 있어서, 상기 제1 피연산자 및 상기 제2 피연산자 모두는 범용 레지스터들인 방법.

청구항 13

제7항에 있어서, 상기 제1 피연산자는 마스크 레지스터이고 상기 제2 피연산자는 범용 레지스터인 방법.

청구항 14

제7항에 있어서,

상기 결과에 기초하여 하나 이상의 상태 레지스터들을 변경하는 단계

를 더 포함하는 방법.

청구항 15

시스템으로서:

랜덤 액세스 메모리; 및

상기 랜덤 액세스 메모리에게 결합되는 프로세서

를 포함하고,

상기 프로세서는:

복수의 벡터 레지스터 - 상기 복수의 레지스터 중 하나가 어레이의 데이터 성분들을 저장함 -; 및

상기 복수의 벡터 레지스터에게 결합되는 실행 회로

를 포함하고,

상기 실행 회로는,

적어도 제1 피연산자 및 제2 피연산자를 특정하는 마스크 생성 명령어를 수신하고, 및

상기 마스크 생성 명령어에 응답하여, 상기 제2 피연산자에서 정의되는 횟수만큼 상기 제1 피연산자의 비트들을 좌측 시프팅시키고, 및 상기 제1 피연산자의 최상위 비트가 시프팅 아웃될 때마다 1의 최하위 비트를 풀인하고 그에 의해 복수의 비트를 포함하는 결과를 발생시키고, 상기 결과에서의 각각의 비트는 상기 어레이의 데이터 성분들 중 하나에 대응하는

시스템.

청구항 16

제15항에 있어서, 상기 제2 피연산자는 벡터 연산의 나머지 루프에서의 나머지 반복들의 횟수를 특정하는 시스

템.

청구항 17

제15항에 있어서, 상기 복수의 벡터 레지스터는 제1 벡터 레지스터 및 제2 벡터 레지스터를 포함하고, 상기 제2 피연산자는 벡터 계산을 위해 상기 제1 벡터 레지스터에서의 기존 데이터 성분들 내로 통합될 상기 제2 벡터 레지스터에서의 데이터 성분들의 수를 특정하는 시스템.

청구항 18

제15항에 있어서, 상기 제1 피연산자 및 상기 제2 피연산자 모두는 범용 레지스터들인 시스템.

청구항 19

제15항에 있어서, 상기 제1 피연산자는 마스크 레지스터이고 상기 제2 피연산자는 범용 레지스터인 시스템.

청구항 20

제15항에 있어서, 하나 이상의 상태 레지스터들은 상기 결과에 기초해 설정되는 시스템.

발명의 설명

기 술 분 야

[0001] 본 개시 내용은, 처리 로직, 마이크로프로세서들, 및 프로세서 또는 다른 처리 로직에 의해 실행될 때 논리적, 수학적, 또는 다른 함수 연산(functional operation)들을 실행하는 연관된 명령어 세트 아키텍처 분야에 관한 것이다.

배 경 기 술

[0002] 명령어 세트 또는 명령어 세트 아키텍처(instruction set architecture: ISA)는 프로그래밍에 관계된 컴퓨터 아키텍처의 일부이며, 또한 네이티브 데이터 형들, 명령어들, 레지스터 아키텍처, 어드레싱 모드들, 메모리 아키텍처, 인터럽트 및 예외 처리, 및 외부 입출력(I/O)을 포함할 수 있다. 용어 명령어는 여기서 일반적으로 매크로 명령어들, 즉 실행을 위해 프로세서[또는 명령어를 프로세서에 의해 처리될 하나 이상의 다른 명령어들로 (예를 들어, 정적 이진 번역(static binary translation), 동적 편집을 포함하는 동적 이진 번역을 이용하여) 번역하고, 모핑(morph)하고, 에뮬레이팅하고, 또는 다른 식으로 변환하는 명령어 변환기(instruction converter)]에게 제공되는 명령어들을 지칭하는데, 이 명령어들은 프로세서의 디코더가 매크로 명령어들을 디코딩한 결과인 마이크로 명령어들 또는 마이크로 연산들(micro-ops)과 대립하는 것이다.

[0003] ISA는 명령어 세트를 구현하는 프로세서의 내부 설계인 마이크로 아키텍처와 구별된다. 상이한 마이크로 아키텍처들을 갖는 프로세서들은 공통 명령어 세트를 공유할 수 있다. 예를 들어, Intel®Core™ 프로세서들, 및 미국 캘리포니아주 서니베일 소재의 Advanced Micro Devices, Inc.의 프로세서들은 (보다 새로운 버전들에서 부가된 몇몇 확장을 가지고) 거의 동일한 버전의 x86 명령어 세트를 구현하지만, 상이한 내부 설계들을 가진다. 예를 들어, ISA의 동일 레지스터 아키텍처가, 전용 물리적 레지스터들, 레지스터 리네이밍 메커니즘(register renaming mechanism), 기타 등등을 이용하는 하나 이상의 동적으로 할당된 물리적 레지스터들을 포함하는 공지된 기술들을 이용하여 상이한 마이크로 아키텍처들에서 상이한 방식들로 구현될 수 있다.

[0004] 대다수의 현대 ISA들은 패킹된 데이터 연산들 또는 SIMD(Single Instruction, Multiple Data) 연산들로서 지칭되기도 하는 벡터 연산들은 지원한다. 단 하나의 데이터 성분 또는 데이터 성분들의 쌍에 대해 연산하는 스칼라 명령어 대신에, 벡터 명령어(패킹된 데이터 명령어(packed data instruction) 또는 SIMD 명령어로도 지칭됨)는 다중 데이터 성분 또는 데이터 성분들의 다중 쌍에 대해 동시에 또는 병렬로 연산할 수 있다. 프로세서는 다중 연산을 동시에 또는 병렬로 실행하기 위해 벡터 명령어에 응답하는 병렬 실행 하드웨어를 가질 수 있다.

[0005] 벡터 연산은 하나의 연산으로 하나의 레지스터 또는 메모리 로케이션 내에 패킹된 다중 데이터 성분에 대해 연산한다. 이런 데이터 성분들은 벡터 데이터 성분 또는 패킹된 데이터 성분들로서 지칭된다. 벡터 데이터 성분들 각각은 다른 것들과 별개로 또는 독립적으로 연산될 수 있는 별개의 개별 데이터 피스(separate individual

piece of data)(예를 들어, 픽셀 컬러 등)를 나타낼 수 있다.

발명의 내용

도면의 간단한 설명

[0006] 실시예들은 첨부된 도면들의 그림들에서 제한적인 것이 아니라 예로서 도해된다.

도 1은 일 실시예에 따른 벡터 레지스터들 및 마스크 레지스터들을 포함하는 명령어 처리 장치의 블록도이다.

도 2a-2c는 일 실시예에 따른 마스크 생성 명령어들의 예들을 도해한다.

도 3a 및 3b는 일 실시예에 따른 어레이 데이터 정렬들의 예들을 도해한다.

도 3c는 일 실시예에 따라 마스크를 이용하는 마스킹된 벡터 명령어의 예를 도해한다.

도 4는 일 실시예에 따른 주어진 벡터 레지스터 폭 및 데이터 성분 폭에 대한 마스크 비트들의 수를 나타낸다.

도 5는 일 실시예에 따라 마스크 생성 명령어에 응답하여 실행될 연산들을 예시하는 흐름도이다.

도 6은 일 실시예에 따라 소스 명령어 세트에서의 이진 명령어들을 타깃 명령어 세트에서의 이진 명령어들로 변환하는 소프트웨어 명령어 변환기의 사용을 예시하는 블록도이다.

도 7a는 일 실시예에 따른 순차적 및 비순차적 파이프라인의 블록도이다.

도 7b는 일 실시예에 따른 순차적 및 비순차적 코어의 블록도이다.

도 8a-8b는 일 실시예에 따른 더 특정적이고 예시적인 순차적 코어 아키텍처의 블록도들이다.

도 9는 일 실시예에 따른 프로세서의 블록도이다.

도 10은 일 실시예에 따른 시스템의 블록도이다.

도 11은 일 실시예에 따른 제2 시스템의 블록도이다.

도 12는 본 발명의 실시예에 따른 제3 시스템의 블록도이다.

도 13은 일 실시예에 따른 SoC(system-on-a-chip)의 블록도이다.

발명을 실시하기 위한 구체적인 내용

[0007] 후속하는 설명에서, 수많은 특정 세부 사항들이 제시된다. 그러나, 본 발명의 실시예들이 이러한 특정한 세부 사항들 없이도 실시될 수 있다는 것이 이해된다. 다른 사례들에서, 공지된 회로들, 구조들 및 기술들은 이 설명의 이해를 모호하게 하지 않기 위해 상세히 보여지지 않았다.

[0008] 본 명세서에서 기술되는 실시예들은 프로세서가 마스킹된 벡터 명령어들에 의해 이용될 마스크를 생성하도록 야기하거나 이를 초래하도록 연산 가능한 마스크 생성 명령어들을 제공한다. 마스킹된 벡터 명령어들은 계산 루프의 트립 카운트(trip-count)(즉, 반복들의 횟수)가 벡터 레지스터 내에 맞추어질 수 있는 성분들의 수에 의해 나누어떨어지지 않는 시나리오에 적용될 수 있다. 그러므로, 나머지 반복들이 개별적으로 다뤄질 필요가 있다. 나머지 반복들에서의 성분들을 처리하기 위해, 마스크 생성 명령어는 적절한 술어 마스크(predicate mask)를 생성하는데, 이 술어 마스크는 계산들로부터 벡터 레지스터의 (예를 들어, 최상위 성분들의) 일부를 생략하거나 마스킹 오프(mask off)하여 어떤 예외들(예를 들어, 할당된 메모리 뒤의의 액세스 또는/및 정의되지 않은 결과들에 의해 야기되는 예외들)도 산출되지 않도록 할 것이다.

[0009] 마스크 생성 명령어는 또한 다른 시나리오들에 사용될 수 있다. 예를 들어, 명령어는 희소 벡터 계산들을 위한 데이터 누산(data accumulation)에서 제어 마스크를 갱신하는데 사용될 수 있다. 데이터 누산은 다중 반복에 걸쳐서 실행될 수 있다. 반복들의 일부에서, 일부 데이터 성분들은 계산을 빠져나갈 수 있고 일부 새로운 데이터 성분들은 계산에 참여할 수 있다. 제어 마스크는 추가 계산을 요구하는 성분들을 추적하기 위해 갱신된다. 제어 마스크는 벡터 계산의 효율성을 향상시키기 위해 마스크 벡터 명령어들에서 활용될 수 있다.

[0010] 벡터 명령어들과 유사하게, 마스킹된 벡터 명령어는 프로세서가 하나 이상의 벡터 피연산자들의 데이터 성분들에 대한 벡터 연산을 실행하는 것을 야기하거나 이를 초래하도록 연산 가능하다(operable). 게다가 각각의 마스킹된 벡터 명령어는 벡터 연산을 마스킹하고, 술어화하고(predicate), 또는 조건부로 제어하기 위해 마스크를

이용한다. 마스크들은 데이터 성분당 그래뉼래리티(per-data element granularity)로 벡터 처리를 마스킹하거나 또는 조건부로 제어하도록 연산 가능하다. 예를 들어, 마스크들은, 단일 소스 벡터 피연산자로부터의 개별 데이터 성분들 또는 두 개의 소스 벡터 피연산자들로부터의 대응하는 데이터 성분들의 개별 쌍들에 대해 실행되는 벡터 연산의 결과가 목적지에 저장될지의 여부를 마스킹하도록 동작 가능할 수 있다. 마스킹된 벡터 명령어들은 각각의 데이터 성분 또는 대응하는 데이터 성분들의 쌍의 벡터 처리가 데이터 성분들과 개별적으로 및 독립적으로 술어화되거나 조건부로 제어되는 것을 허용한다. 마스킹된 벡터 명령어들, 연산들, 및 마스크들은, 예를 들어, 증가된 코드 밀도 및/또는 더 큰 명령어 처리량과 같은 소정의 장점들을 제공할 수 있다.

[0011] 도 1은 본 명세서에서 기술되는 마스크 생성 명령어들을 포함하는, 명령어들을 실행하도록 동작 가능한 회로를 포함하는 실행 유닛(140)을 갖는 명령어 처리 장치(115)의 실시예의 블록도이다. 몇몇 실시예들에서, 명령어 처리 장치(115)는 프로세서, 다중 코어 프로세서의 프로세서 코어, 또는 전자적 시스템의 처리 요소일 수 있다.

[0012] 디코더(130)는 고급 기계어 명령어들 또는 매크로 명령어들의 형태로 들어오는 명령어들(incoming instructions)을 수신하고, 이들을 디코딩하여 하위 마이크로 연산들, 마이크로 코드 엔트리 포인트들, 마이크로 명령어들, 또는 최초의 고급 명령어를 반영하는 및/또는 그로부터 도출되는 다른 하위 명령어들 또는 제어 신호들을 발생시킨다. 하위 명령어들 또는 제어 신호들은 하위(예를 들어, 회로 레벨 또는 하드웨어 레벨) 연산들을 통해 고급 명령어의 연산을 구현할 수 있다. 디코더(130)는 다양하고 상이한 메커니즘들을 이용하여 구현될 수 있다. 적합한 메커니즘들의 예는, 마이크로 코드, 룩업 테이블들, 하드웨어 구현들, PLA(programmable logic array)들, 관련 기술 분야에 알려진 디코더들을 구현하는 데에 이용되는 다른 메커니즘들, 기타 등등을 포함하지만, 이들로만 제한되지는 않는다.

[0013] 디코더(130)는 캐시(110), 메모리(120) 또는 다른 소스들로부터의 명령어들을 수신할 수 있다. 디코딩된 명령어들은 실행 유닛(140)에게 보내진다. 실행 유닛(140)은 디코더(130)로부터 하나 이상의 마이크로 연산들, 마이크로 코드 엔트리 포인트들, 마이크로 명령어들, 다른 명령어들, 또는 수신된 명령어들을 반영하거나 그로부터 도출되는 다른 제어 신호들을 수신할 수 있다. 실행 유닛(140)은 레지스터 파일(170), 캐시(110), 및/또는 메모리(120)로부터 데이터 입력을 수신하고 이들에게 데이터 출력을 발생한다.

[0014] 일 실시예에서, 레지스터 파일(170)은 레지스터들로도 지칭되는 아키텍처 레지스터들을 포함한다. 달리 특정되거나 매우 명백하지 않으면, 아키텍처 레지스터들, 레지스터 파일, 및 레지스터들이라는 구들은 소프트웨어 및/또는 프로그래머에게 보일 수 있는(예로, 소프트웨어 가시적) 레지스터들 및/또는 피연산자(operand)들을 식별하기 위해 매크로 명령어들에 의해 특정되는 레지스터들을 지칭하기 위해 여기에 사용된다. 이런 레지스터들은 주어진 마이크로 아키텍처에서의 다른 비 아키텍처 레지스터들(예를 들어, 임시 레지스터들, 리오더 버퍼들(reorder buffers), 리타이어먼트 레지스터들(retirement registers) 등)과 대조된다.

[0015] 대안적으로, 하나 이상의 다른 실시예들에서, 디코더(130)를 갖는 것이 아니라, 명령어 처리 장치(115)는 그 대신에 명령어 에뮬레이터, 번역기, 모퍼(morpher), 인터프리터, 또는 다른 명령어 변환 로직을 가질 수 있다. 다양하고 상이한 유형의 명령어 변환 로직이 관련 분야에 알려져 있고 또한 소프웨어, 하드웨어, 펌웨어, 또는 이들의 조합으로 구현될 수 있다. 명령어 변환 로직은 마스크 생성 명령어들 중 하나 이상을 수신하고, 그 명령어를 하나 이상의 대응하는 도출된 명령어들 또는 제어 신호들로 에뮬레이팅하고, 번역하고, 모핑하고, 인터프리팅하고, 또는 다른 방식으로 변환할 수 있다. 또 다른 실시예들에서, 명령어 처리 장치(115)는 디코더 및 부가적 명령어 변환 로직 모두를 가질 수 있다. 예를 들어, 명령어 처리 장치(115)는 마스크 생성 명령어들 중 하나 이상을 하나 이상의 중간 명령어들로 변환하는 명령어 변환 로직, 및 하나 이상의 중간 명령어들을 명령어 처리 장치의 본래 하드웨어에 의해 실행 가능한 하나 이상의 하위 명령어들 또는 제어 신호들이 되도록 디코딩하는 디코더를 가질 수 있다. 명령어 변환 로직의 일부 또는 전부는, 별개의 다이상에 또는 오프 다이(off-die) 메모리에 자리 잡는 것과 같이, 명령어 처리 장치의 나머지로부터 오프 다이로 자리잡을 수 있다.

[0016] 일 실시예에 따라, 레지스터 파일(170)은 둘 모두가 마스크 생성 명령어들의 피연산자들을 저장하는 데에 사용될 수 있는 벡터 레지스터들(175)의 세트 및 마스크 레지스터들(185)의 세트를 포함한다. 각각의 벡터 레지스터(175)는 512 비트, 256 비트, 또는 128 비트 폭일 수 있거나, 상이한 벡터 폭이 이용될 수 있다. 각각의 마스크 레지스터(185)는 다수의 마스크 비트를 포함하는데, 각각의 마스크 비트는 벡터 레지스터들(175) 중 하나의 벡터 레지스터의 하나의 데이터 성분에 대응한다. 각각의 마스크 비트가 한 벡터 레지스터의 한 데이터 성분을 마스킹하는 데에 사용되므로, 64 비트의 마스크 레지스터는 512 비트 레지스터의 64개의 8 비트 데이터 성분을 마스킹하는 데에 사용될 수 있다. 상이한 사이즈(예를 들어, 16 비트, 32 비트 또는 64 비트)의 데이터 성분들 및 상이한 폭(예를 들어, 256 비트 또는 128 비트)을 가진 벡터 레지스터에 대해, 상이한 수의 마스크

비트들이 벡터 연산과 연계하여 이용될 수 있다.

[0017] 설명을 불명료하게 하는 것을 피하기 위하여, 비교적 간단한 명령어 처리 장치(115)가 도시되고 설명되었다. 다른 실시예들이 둘 이상의 실행 유닛을 가질 수 있다는 것을 이해해야 한다. 예를 들어, 장치(115)는, 예를 들어, 산술 유닛들, ALU(arithmetic logic unit)들, 정수 유닛들, 부동 소수점 유닛들, 기타 등등과 같은 다중의 다양한 유형의 실행 유닛을 포함할 수 있다. 명령어 처리 장치 또는 프로세서들의 또 다른 실시예들은 다중 코어, 논리적 프로세서들, 또는 실행 엔진들을 가질 수 있다. 명령어 처리 장치(115)의 수많은 실시예들은 도 7-13에 대하여 이후에 제공될 것이다.

[0018] 본 발명의 실시예들에 따라, 본 명세서에서 기술되는 마스크 생성 명령어는 명령어의 레지스터 피연산자에서 비트들을 시프팅시킴으로써 마스크를 생성한다. 레지스터 피연산자는 마스크 레지스터 또는 범용 레지스터일 수 있다. 도 2a-2c는 마스크 생성 명령어들을 위한 의사 코드의 예들을 도시한다. 이러한 도면들에서, r1, r2는 독립적 사이즈들의 범용 레지스터들을 나타내고(예를 들어, r1이 32 비트일 수 있는 한편, r2가 64 비트이다), k1은 마스크 레지스터를 나타낸다. 값 KL은 마스크 비트들의 수를 나타내는데, 이는 명령어의 끝에 첨부되는 연상 기호(mnemonic) B/W/D/Q로부터 결정될 수 있다.

[0019] 도 2a는 마스크 생성 명령어 KSHLONES[B/W/D/Q] k1, r2의 예를 기술한다. 연상 기호들 B/W/D/Q는 명령어 KSHLONES가 4가지 형태: KSHLONESB, KSHLONESW, KSHLONESD 및 KSHLONESQ를 갖는다는 것을 의미하는데, 이것들은 제각기 8, 16, 32, 64 비트들의 마스크들에 대응한다. 이 예에서, k1 마스크는 소스 피연산자 및 목적지의 양쪽의 역할을 한다. 다른 소스 피연산자는 범용 레지스터 또는 메모리로부터의 값이다.

[0020] KSHLONES 명령어는 소스 피연산자(r2 또는 메모리)에 정의되는 횟수만큼 좌측으로 k1 마스크의 비트들을 시프팅하고 또한 하위 비트 위치들을 채우기 위해 1들을 풀인(pull in)한다. 본 명세서의 용어 "좌측 시프팅(shift left 또는 left-shift)"은 비트들이 최하위 비트(LSB)로부터 최상위 비트(MSB)로의 방향으로 시프팅된다는 것을 의미한다. 즉, k1 마스크가 1 비트 위치씩 좌측으로 시프팅될 때마다, 1의 비트 값을 풀인하여 최하위 비트 위치를 채운다. 예를 들어 k1=1:0:0:0:1:1:0:0 및 r2=4이라면, "KSHLONESB k1, r2"는 결과 k1=1:1:0:0:1:1:1:1을 산출할 것인데, 여기서 각각의 "0" 및 "1"은 비트 값을 나타낸다. 결과로 생기는 (목적지) k1에 남아 있는 그런 k1 비트들이 위치에 대해서만 시프팅되고 이들의 값들이 시프팅에 의해 수정되지 않는 것을 유의하라. LSB 위치들에 더해지는 새로운 비트들은 모두 1들이다.

[0021] 도 2b는 마스크 생성 명령어 SHLONES[B/W/D/Q] r1, r2의 대안 실시예를 예시하는데, 이 명령어는 범용 레지스터 r1을 소스 피연산자와 목적지의 양자로서 사용한다. 이 형태의 명령어는 보수 비트 조작 명령어(complementary bit manipulation instruction)로서의 이것의 유용성을 가능하게 한다. 도 2c는 명령어가 직접적으로 제어 흐름에 대해 이용될 수 있도록 상태 플래그들(ZF, CF)을 변경하는 마스크 생성 명령어의 또 다른 대안 실시예를 예시한다. 마스크 생성 명령어의 또 다른 실시예는 시프팅된 결과(즉, 결과로 생기는 마스크)를 소스 피연산자들과는 상이한 목적지 레지스터에 저장한다; 예를 들어, KSHLONES k1, k2, r2 및 SHLONES r1, r2, r3. 앞서 기술된 마스크 생성 명령어들과 동일한 명령어 포맷들을 반드시 갖지는 않는 명령어들의 추가적 대안 실시예들이 존재할 수 있다. 하기 기술에서, 마스크 생성 명령어들의 다양한 형태들이 KSHLONES 및 이것의 변형들로서 지칭된다.

[0022] 도 3a 및 3b는 KSHLONES 및 이것의 변형들이 벡터 계산의 효율성을 향상시키는데 사용될 수 있는 예시적 시나리오들을 도해한다. 이러한 예들에서, 벡터 연산의 나머지 루프(remainder loop)에서의 나머지 어레이 성분들은 전체 벡터 레지스터를 채우지 못한다. 이런 성분들에 있어서, 벡터 레지스터가 최고 16개까지의 어레이 성분을 저장할 수 있다고 가정한다: 예를 들어, 벡터 레지스터는 512 비트를 가지고 각각의 어레이 성분은 32 비트 더블 워드이다. (도 3a에 도시된 바와 같이) 어레이 성분들의 전체 수가 35이고 루프의 시작이 벡터 레지스터와 정렬되면, 벡터화 루프에서 처리되지 않고 또한 개별적으로 다뤄질 필요가 있는 종단에서의 세 개의 나머지 어레이 성분이 있을 것이다. 어레이 성분들의 전체 수가 35이고 루프의 시작이 벡터 레지스터와 정렬되지 않으면 (도 3b에 도시된 바와 같이 제1 벡터화 루프에서의 2개의 어레이 성분), 벡터화 루프에서 처리되지 않고 또한 개별적으로 다뤄질 필요가 있는 종단에서의 하나의 나머지 어레이 성분이 있을 것이다. 본 명세서에서 기술되는 마스크 생성 명령어는 루프 벡터화를 향상시키기 위해 마스크 벡터 연산들에서의 나머지 어레이 성분들에 의해 사용될 수 있는 마스크를 생성한다.

[0023] 데이터 액세스의 효율성을 향상시키기 위해, 컴파일러가 최종 벡터화 루프에서의 나머지 어레이 성분들을 개별적으로 다루기 위한 코드를 발생할 수 있다. 그러나, 최종 벡터화 루프에서의 어레이 성분들의 수는, 어레이 성분들의 주소들 및/또는 루프 트립 카운트들이 컴파일링 시간에 알려지지 않으므로, 일반적으로 컴파일링 시간

에 결정(resolve)될 수 없다. 본 명세서에서 기술되는 실시예들에 의해, 컴파일링 시간에 컴파일러는 동일 작업들을 실행하는 다른 코드 시퀀스들 대신에 마스크 생성 명령어들 중 하나 이상을 생성할 수 있다. 그러므로, 컴파일러는 그 루프 최적화 작업을 단순화하기 위해 이러한 마스크 생성 명령어들을 활용할 수 있다. 대안 실시예들에서, 마스크 생성 명령어들은 프로그래머 또는 다른 코드 발생 엔티티에 의해 이용될 수 있다.

[0024] KSHLONES 명령어 및 이것의 변형들은 루프의 바로 종단에서의 나머지 데이터 성분들의 전체 사이즈가 벡터 레지스터의 폭보다 더 작은 시나리오를 다루는데 사용될 수 있다. 이것은 KSHLONES 명령어 및 이것의 변형들이 전폭 벡터 연산(full-width vector operation)을 형성하기에 충분한 루프에서의 반복들이 없을 때(즉, 어레이에서 충분한 데이터 성분들이 없을 때) 이용될 수 있다는 것을 의미한다.

[0025] 도 3c의 예에서, 어레이의 최종 3개의 데이터 성분(즉, A(32), A(33), A(34))은 소스 벡터(307)의 전폭을 점유하지 않는다. 즉, 전체 벡터 레지스터를 채우는 데에 A에 남겨진 충분한 성분들이 없다. 소스 벡터(307)가 그 최하위(lowest-order) 데이터 성분들로서 A(32), A(33), A(34)를 포함하기 때문에, 마스크(308)의 최하위 3개의 비트만이 A(32), A(33), A(34)에 대해, 가산이 실행되어야 하고 또한 가산 결과들이 저장되어야 한다는 것을 표시하기 위해 (예로, 1에) 설정된다. 마스크(308)의 고위(higher-order) 13개의 비트는 (예를 들어, 0으로) 클리어링된다. 마스크(308)는 KSHLONES 명령어 또는 이것의 변형들 중 하나를 실행하는 프로세서에 의해 발생되는 결과일 수 있다.

[0026] 일 실시예에서, (전체 벡터 레지스터를 채우기 위한) 어레이의 종단에서의 데이터 성분들의 부족은 어레이의 기본 주소에서의 초기 오정렬의 결과일 수 있다. 예를 들어, 이미지 처리 애플리케이션들에서, 종종 이미지 어레이의 사이즈는 벡터 레지스터 폭의 정수 배이다. 그러나, 이미지 어레이의 시작이 오정렬되면, 많은 수의 데이터 성분들이 루프의 종단에 남겨져서 전체 벡터 레지스터를 채울 수 없게 된다.

[0027] 마스크(308)의 사용은 어레이의 데이터 성분들이 피연산자들인 루프의 실행을 백터화하는 것을 돕는다. 도 3c의 예들에서, 반복 인덱스 i = 32, 33 및 34는 소스 벡터(307)가 마스크(308)와 함께 사용되는 마스킹된 벡터 연산으로 벡터화될 수 있다. 일 실시예에서, 루프의 검출 시에, 컴파일러는 본 명세서에서 기술되는 마스크 생성 명령어들 중 하나 이상을 포함하는 루프 최적화 코드를 발생할 수 있다.

[0028] 예시된 마스킹된 벡터 연산(303)을 위한 명령어는 스칼라 값에 가산될 소스 벡터를 표시한다. 다른 마스킹된 벡터 명령어들은 둘 이상의 소스 벡터들을 표시할 수 있다. 마스킹된 벡터 연산(303)의 명령어는 또한 마스크(308)를 특정한다. 각각의 마스크들은 다중 마스크 성분, 술어 성분들, 조건부 제어 성분들, 또는 플래그들을 포함한다. 예시에 나타난 바와 같이, 하나의 소스 벡터 피연산자를 수반하는 연산의 경우에 각각의 대응하는 소스 데이터 성분에 대한 하나의 그러한 마스크 성분 또는 플래그가 있을 수 있다. 일반적으로 각각의 성분 또는 플래그는 단일 비트일 수 있다. 단일 비트는 두 가지 상이한 가능성 중 어느 하나를 특정하는 것을 허용할 수 있다(예를 들어, 연산을 실행하는 것 대 연산을 실행하지 않는 것, 연산 결과를 저장하는 것 대 연산 결과를 저장하지 않는 것, 기타 등등). 대안적으로, 3 이상의 상이한 선택 사항들 중에서 선택하는 것이 요망된다면, 2개 이상의 비트들이 각각의 플래그 또는 성분에 대해 사용될 수 있다.

[0029] 예시된 관례에 따라, 주어진 마스크 비트가 1에 설정될 때, 벡터 연산의 결과는 소스 벡터의 대응하는 데이터 성분에 대해 실행되고, 이 결과의 대응하는 데이터 성분에 저장된다. 반대로, 주어진 마스크 비트가 0으로 클리어링될 때, 벡터 연산은 소스 벡터의 대응하는 데이터 성분에 대해 생략되거나(즉, 실행되지 않거나), 또는 결과가 이 결과의 대응하는 데이터 성분에 저장되도록 허용되지 않는다. 오히려, 또 다른 값이 결과 데이터 성분에 저장될 수 있다. 예를 들어, 소스 벡터로부터의 대응하는 데이터 성분의 수치 값이 저장된다. 대안 실시예에서, 0 또는 또 다른 미리 정해진 값이 결과의 대응하는 데이터 성분에 저장될 수 있다. 결과들이 저장되는 것을 허용하기 위해 비트들이 (즉, 0으로) 클리어링되거나, 또는 결과들이 저장되는 것을 허용하지 않기 위해 비트들이 (즉, 1에) 설정되는, 예시된 것과는 반대의 관례가 또한 가능하다.

[0030] 하기 예시적인 코드 시퀀스는 현재 반복 카운트가 rbx에 저장되고 루프 한계가 rcx에 저장되면서 나머지 루프를 위한 마스크를 생성시킨다. 도 3c의 예시된 실시예를 이용할 때, 현재 반복 카운트는 31이고 루프 한계는 34이다.

[0031] SUB rbx, rcx //남아 있는 반복들의 횟수를 계산

[0032] KXOR k1, k1, k1 //제로잉 마스크

[0033] KSHLONES k1, rbx // 나머지 루프를 위한 마스크를 생성

[0034] 나머지 루프를 위한 마스크를 생성하기 위해 KSHLONES 명령어(이것의 변형들을 포함함)를 사용하는 많은 장점들이 있다. KSHLONES 명령어들은 감산 결과로 연산한다. 그 연산의 일부로서 감산을 포함하는 또 다른 명령어에 대해, 감산 전에 피연산자 유형 비교를 실행하기 위해 부가적 사전 계산 오버헤드가 초래될 것이다. 추가로, KSHLONES 명령어들은 반복 카운터 및/또는 루프 한계가 네거티브일 수 있는 시나리오들을 커버할 수 있고, 이는 컴파일러가 코드를 최적화하는 데에 더 큰 가변성을 허용한다. 덧붙여, 나머지 루프를 위한 마스크를 생성하기 위한 코드는 3 국면(즉, 상기 코드 시퀀스에서의 3개의 명령어)으로 나누어지는데, 이것은 KSHLONES 명령어들의 사용에 있어서 실행 스케줄링을 향상시키고 더 많은 가변성 및 융통성을 제공한다. KSHLONES 명령어들은 피연산자들의 감산이 필요하지 않은 경우에 그 자체로 또는 다른 명령어들과 조합하여 이용될 수 있다. 예를 들어, 1들의 개수(N)가 알려져 있을 때, KSHLONES는 다음과 같이 N개의 최하위 비트에 1들을 가진 마스크를 생성하기 위해 이용될 수 있다: N = 5; k1= 0:0:0:0:0:0:0:0; KSHLONES k1, N은 k1 = 0:0:0:1:1:1:1:1을 초래한다.

[0035] KSHLONES 명령어들은 도 4의 예에 나타난 바와 같이, 희소 벡터 계산을 위한 데이터 누산에 또한 사용될 수 있다. 이 예에서, 한 쌍의 벡터 레지스터들(V1와 V2) 및 한 쌍의 마스크 레지스터들(K1과 K2)은 데이터 누산을 수행하기 위해 활용된다. V1과 V2의 양쪽은 모든 데이터 성분 위치들이 채워지지는 않은 희소 벡터(sparse vector)들이다. V1은 계산을 위해 벡터 성분들을 누산하기 위한 누산기의 역할을 하고 V2는 V1에서의 활용되지 않은 슬롯들에 채우기 위한 새로운 데이터 성분들을 제공한다. 마스크 레지스터들 K1 및 K2는 대응하는 벡터 레지스터들에서의 어느 위치들이 계산을 위한 유효 데이터 성분들을 포함하는지를 표시하는데 사용된다. 이 예에서 유효 데이터 성분들에 대응하는 마스크 비트들은 K1과 K2 모두에 대해 1에 설정된다. K2의 비트 값들이 V2의 동일 데이터 성분들에 대해 역으로 될 수 있다는 것이 이해된다.

[0036] 도 4의 예에서, V2는 B0으로서 표시되는 4개의 성분을 초기에 포함한다. K2에서의 대응하는 마스크 비트들은 이러한 4개의 성분의 위치들을 표시한다. N = POPCNT(K2)를 이용함으로써, N의 값은 1의 값을 갖는 K2 비트들의 수에 설정된다. 그러므로 이 예에서, N = 4이다. K1에서의 마스크 비트들은 초기 V1의 성분 위치들 0-2에 대응하는 세 개의 1을 포함한다. K1에 포함되는 정보는 누산된 성분들 A0의 수뿐만 아니라, V1 내부의 비어있는 슬롯들의 우측(right) 경계를 표시한다(이 예에서 우측 경계는 제3 성분 위치에 있다). K1은 COMPRESS 및/또는 EXPAND 명령어들을 포함하는 추가적 데이터 누산에 대해 그대로 또는 반전되어 이용될 수 있다.

[0037] 4개의 B0는 기존 벡터 명령어들을 이용하여, V1의 3-6의 성분 위치들 내로 압축되고 통합될 수 있다. 갱신된 V1은 초기 V1보다 더 조밀하게 되고, 그러므로 효율적 벡터 계산에 더 잘 따를 수 있다. 통합 후의 대응하는 K1은 K1 = KSHLONES(K1, N)에 의해 계산될 수 있는데, 이것은 K1의 소스 값에서의 1의 초기 3개의 비트를 보존하고 1의 4개의 부가 비트를 추가한다. K1의 소스 값을 보존하는 것은 통합 전후의 누산기 내부에 있는 성분들의 수를 추적하기 위한 개별 카운터들을 유지하기 위한 필요를 제거한다. 갱신된 V1이 벡터 계산에 이용된 후, 도 4의 연산들은 반복될 수 있어서 누산기가 벡터 계산을 위해 데이터 성분들을 누산하는 것을 유지할 수 있도록 한다.

[0038] 본 명세서에 개시되는 마스크 생성 명령어들은 일반적 용도를 갖는 범용 명령어들이다. 예를 들어, 이러한 명령어들은 단독으로 또는 다른 명령어들과 조합되어, 벡터 연산들의 나머지 루프를 위한 또는 희소 벡터 계산에서의 데이터 누산을 위한 마스크를 계산하기 위해 이용될 수 있다. 다른 용도들도 본 개시에 기초하여 상정된다.

[0039] 도 5는 일 실시예에 따른 마스크 생성 명령어를 실행하기 위한 방법 500의 블록 흐름도이다. 방법 500은 프로세서(보다 상세하게는, 예를 들어, 도 1의 실행 유닛(140))가 적어도 제1 피연산자 및 제2 피연산자를 특정하는 마스크 생성 명령어를 수신하는 것으로 시작한다(블록 510). 마스크 생성 명령어들의 예들은 전술한 바와 같이 KSHLONES 명령어 및 이것의 변형들을 포함한다. 일 실시예에서, 제1 피연산자는 마스크 레지스터이고 제2 피연산자는 범용 레지스터이다. 대안 실시예에서, 제1 피연산자와 제2 피연산자는 모두 범용 레지스터들이다. 마스크 생성 명령어에 응답하여, 프로세서는 하기 연산들을 실행한다(블록 520): 제2 피연산자에서 정의되는 횟수만큼 제1 피연산자의 비트들을 좌측 시프팅하고(블록 530), 및 제1 피연산자의 최상위 비트가 (좌측으로) 시프팅 아웃될 때마다 1의 최하위 비트를 풀인하고, 그에 의해 결과를 발생한다(블록 540). 결과에서의 각각의 비트는 데이터 성분에 대응한다. 결과는 마스킹된 벡터 연산에 사용될 마스크이다.

[0040] 다양한 실시예들에서, 방법 500은 범용 프로세서, (예를 들어, 그래픽 프로세서 또는 디지털 신호 프로세서와 같은) 특수 목적 프로세서, 또는 다른 형태의 디지털 논리 소자 또는 명령어 처리 장치에 의해 실행될 수 있다. 몇몇 실시예들에서, 방법 500은 도 1의 명령어 처리 장치(115), 또는 도 7-13에 도시된 실시예들에서와 같은 비슷한 프로세서, 장치, 또는 시스템에 의해 실행될 수 있다. 더욱이, 도 1의 명령어 처리 장치(115)뿐만

아니라, 도 7-13에 도시된 프로세서, 장치, 또는 시스템은 방법 500의 것들과 동일하거나, 유사하거나, 또는 상이한 연산들 및 방법들의 실시예들을 실행할 수 있다.

[0041] 몇몇 실시예들에서, 도 1의 명령어 처리 장치(115)는 소스 명령어 세트로부터 타깃 명령어 세트로 명령어를 변환하는 명령어 변환기와 연계하여 동작할 수 있다. 예를 들어, 명령어 변환기는 명령어를 코어에 의해 처리될 하나 이상의 다른 명령어들로 (예를 들어, 정적 이진 번역, 동적 편집을 포함하는 동적 이진 번역을 이용하여) 번역하고, 모핑하고, 에뮬레이팅하고, 또는 다른 식으로 변환할 수 있다. 명령어 변환기는 소프트웨어, 하드웨어, 펌웨어, 또는 이것들의 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.

[0042] 도 6은 본 발명의 실시예들에 따른 소프트웨어 명령어 변환기의 사용을 대조시키는 블록도이다. 예시된 실시예에서, 명령어 변환기는 소프트웨어 명령어 변환기이지만, 대안적으로 명령어 변환기는 소프트웨어, 펌웨어, 하드웨어, 또는 이것들의 다양한 조합들로 구현될 수 있다. 도 6은 고급 언어(602)로 된 프로그램이 x86 컴파일러(604)를 이용하여 컴파일링되어 적어도 하나의 x86 명령어 세트 코어를 구비한 프로세서(616)에 의해 선천적으로 실행될 수 있는 x86 이진 코드(606)를 발생할 수 있다는 것을 보여준다. 적어도 하나의 x86 명령어 세트 코어를 구비한 프로세서(616)는, 적어도 하나의 x86 명령어 세트 코어를 구비한 인텔 프로세서와 실질적으로 동일한 결과를 달성하기 위하여, (1) 인텔 x86 명령어 세트 코어의 명령어 세트의 상당한 부분 또는 (2) 적어도 하나의 x86 명령어 세트 코어를 구비한 인텔 프로세서상에서 실행되는 것을 목표로 하는 애플리케이션들 또는 기타 소프트웨어의 오브젝트 코드 버전들을 호환 가능하게 실행하거나 기타 방식으로 처리함으로써 적어도 하나의 x86 명령어 세트 코어를 구비한 인텔 프로세서와 실질적으로 동일한 기능들을 실행할 수 있는 임의의 프로세서를 나타낸다. x86 컴파일러(604)는, 추가의 연계(linkage) 처리와 함께 또는 이것 없이, 적어도 하나의 x86 명령어 세트 코어를 구비한 프로세서(616)상에서 실행될 수 있는, x86 이진 코드(606)(예를 들어, 오브젝트 코드)를 발생하도록 동작 가능한 컴파일러를 나타낸다. 유사하게, 도 6은 고급 언어(602)로 된 프로그램이 대안 명령어 세트 컴파일러(608)를 사용하여 컴파일링되어 적어도 하나의 x86 명령어 세트 코어를 구비하지 않은 프로세서(614)(예컨대, 미국 캘리포니아주 서니베일 소재의 MIPS 테크놀로지스사의 MIPS 명령어 세트를 실행하는 및/또는 미국 캘리포니아주 서니베일 소재의 ARM 홀딩스사의 ARM 명령어 세트를 실행하는 코어들을 구비한 프로세서)에 의해 선천적으로 실행될 수 있는 대안 명령어 세트 이진 코드(610)를 발생할 수 있다는 것을 보여준다. 명령어 변환기(612)는 x86 이진 코드(606)를 x86 명령어 세트 코어를 구비하지 않은 프로세서(614)에 의해 선천적으로 실행될 수 있는 코드로 변환하는데 사용된다. 이 변환된 코드는 대안 명령어 세트 이진 코드(610)와 동일할 것 같지는 않은데, 그 이유는 이것을 할 수 있는 명령어 변환기를 만들기가 어렵기 때문이다; 그러나, 변환된 코드는 일반 연산을 달성할 것이고 대안 명령어 세트로부터의 명령어들로 이루어져 있을 것이다. 따라서, 명령어 변환기(612)는 에뮬레이션, 시뮬레이션, 또는 임의의 다른 처리를 통해 x86 명령어 세트 프로세서 또는 코어를 구비하지 않은 프로세서 또는 다른 전자 디바이스가 x86 이진 코드(606)를 실행하게끔 허용하는 소프트웨어, 펌웨어, 하드웨어, 또는 이들의 조합을 나타낸다.

[0043] 예시적 코어 아키텍처들

[0044] 순차적 및 비순차적 코어 블록도

[0045] 도 7a는 본 발명의 실시예들에 따른 예시적인 순차적(in-order) 파이프라인과 예시적인 레지스터 리네이밍 비순차적(out-of-order) 발행/실행 파이프라인 모두를 예시하는 블록도이다. 도 7b는 본 발명의 실시예들에 따른 프로세서에 포함될 순차적 아키텍처 코어와 예시적인 레지스터 리네이밍 비순차적 발행/실행 아키텍처 코어의 예시적 실시예 모두를 예시하는 블록도이다. 도 7a 및 도 7b에서의 실선 박스들은 순차적 파이프라인 및 순차적 코어를 예시하는 반면에, 점선 박스들의 옵션적인 추가는 레지스터 리네이밍 비순차적 발행/실행 파이프라인 및 코어를 예시한다. 순차적 양태가 비순차적 양태의 부분 집합이라는 것을 고려하여, 비순차적 양태가 설명될 것이다.

[0046] 도 7a에서, 프로세서 파이프라인(700)은, 페치 단(fetch stage)(702), 길이 디코딩 단(704), 디코딩 단(706), 할당 단(708), 리네이밍 단(710), (디스패치 또는 발행이라고도 알려진) 스케줄링 단(712), 레지스터 판독/메모리 판독 단(714), 실행 단(716), 라이트 백(write back)/메모리 기입 단(718), 예외 처리 단(722), 및 커밋 단(724)을 포함한다.

[0047] 도 7b는 실행 엔진 유닛(750)에 결합된 프론트 엔드 유닛(front end unit)(730)을 포함하는 프로세서 코어(790)를 도시하며, 양자 모두는 메모리 유닛(770)에 결합된다. 코어(790)는 RISC(reduced instruction set computing) 코어, CISC(complex instruction set computing) 코어, VLIW(very long instruction word) 코어,

또는 복합형 또는 대안 코어 타입일 수 있다. 또 다른 옵션으로서, 코어(790)는, 예를 들어, 네트워크 또는 통신 코어, 압축 엔진, 보조프로세서 코어, 범용 컴퓨팅 그래픽 프로세싱 유닛(GPGPU) 코어, 그래픽 코어 또는 그와 유사한 것과 같은 특수 목적 코어일 수 있다.

[0048] 프론트 엔드 유닛(730)은, 디코딩 유닛(740)에 결합되는 명령어 페치 유닛(738)에 결합되는 명령어 TLB(translation lookaside buffer)(736)에 결합되는 명령어 캐시 유닛(734)에 결합되는 분기 예측 유닛(732)을 포함한다. 디코딩 유닛(740)(또는 디코더)은 명령어들을 디코딩할 수 있으며, 또한 최초 명령어들로부터 디코딩되거나 다른 경우에는 이들을 반영하거나, 또는 이들로부터 도출되는, 하나 이상의 마이크로 연산들, 마이크로 코드 엔트리 포인트들, 마이크로 명령어들, 기타 명령어들 또는 다른 제어 신호들을 출력으로서 발생할 수 있다. 디코딩 유닛(740)은 다양하고 상이한 메커니즘들을 이용하여 구현될 수 있다. 적절한 메커니즘들의 예들은 룩업 테이블들, 하드웨어 구현들, PLA들(programmable logic arrays), 마이크로 코드 ROM들(read only memories), 기타 등등을 포함하지만 이것들에만 한정되지는 않는다. 일 실시예에서, 코어(790)는 특정 매크로 명령어들에 대한 마이크로 코드를 저장하는 마이크로 코드 ROM 또는 다른 매체를 (예를 들어, 디코딩 유닛(740) 내에 또는 그렇지 않은 경우에는 프론트 엔드 유닛(730) 내에) 포함한다. 디코딩 유닛(740)은 실행 엔진 유닛(750)에서의 리네이밍/할당기 유닛(752)에 결합된다.

[0049] 실행 엔진 유닛(750)은, 리타이어먼트 유닛(754) 및 하나 이상의 스케줄러 유닛(들)(756)의 세트에 결합되는 리네이밍/할당기 유닛(752)을 포함한다. 스케줄러 유닛(들)(756)은, 명령어 대기열들(reservations stations), 중앙 명령어 윈도, 기타 등등을 포함하는 임의 수의 상이한 스케줄러들을 나타낸다. 스케줄러 유닛(들)(756)은 물리적 레지스터 파일(들) 유닛(들)(758)에 결합된다. 물리적 레지스터 파일(들) 유닛들(758) 각각은 하나 이상의 물리적 레지스터 파일들을 나타내고, 이들 중 상이한 것들은 스칼라 정수, 스칼라 부동 소수점, 패킹된 정수, 패킹된 부동 소수점, 벡터 정수, 벡터 부동 소수점, 상태(예로서, 실행될 다음 명령어의 어드레스인 명령어 포인터) 등과 같은 하나 이상의 상이한 데이터 타입들을 저장한다. 일 실시예에서, 물리적 레지스터 파일(들) 유닛(758)은 벡터 레지스터 유닛, 기입 마스크 레지스터 유닛, 및 스칼라 레지스터 유닛을 포함한다. 이러한 레지스터 유닛들은 아키텍처 벡터 레지스터들, 벡터 마스크 레지스터들, 및 범용 레지스터들을 제공할 수 있다. 레지스터 리네이밍 및 비순차적 실행이 구현될 수 있는 다양한 방식들[예컨대, 리오더 버퍼(들) 및 리타이어먼트 레지스터 파일(들)을 사용하는 것; 장래 파일(future file)(들), 이력 버퍼(history buffer)(들), 및 리타이어먼트 레지스터 파일(들)을 사용하는 것; 레지스터 맵 및 레지스터들의 풀(pool)을 사용하는 것 등]을 예시하기 위해, 물리적 레지스터 파일(들) 유닛(들)(758)이 리타이어먼트 유닛(754)과 중첩되어 있다. 리타이어먼트 유닛(754) 및 물리적 레지스터 파일(들) 유닛(들)(758)은 실행 클러스터(들)(760)에 결합된다. 실행 클러스터(들)(760)는 하나 이상의 실행 유닛(762)의 세트, 및 하나 이상의 메모리 액세스 유닛들(764)의 세트을 포함한다. 실행 유닛들(762)은 다양한 타입의 데이터(예로서, 스칼라 부동 소수점, 패킹된 정수, 패킹된 부동 소수점, 벡터 정수, 벡터 부동 소수점)에 대해 다양한 연산들(예로서, 시프트, 가산, 감산, 승산)을 실행할 수 있다. 몇몇 실시예들은 특정한 기능이나 기능 세트에 전용된 복수의 실행 유닛을 포함할 수 있지만, 다른 실시예들은 단 하나의 실행 유닛, 또는 모두가 모든 기능들을 실행하는 다중 실행 유닛을 포함할 수도 있다. 스케줄러 유닛(들)(756), 물리적 레지스터 파일(들) 유닛(들)(758), 및 실행 클러스터(들)(760)는 가능하게는 복수개인 것으로 도시되어 있는데, 그 이유는 특정 실시예들은 특정 타입들의 데이터/연산들에 대해 별개의 파이프라인들(예를 들어, 스칼라 정수 파이프라인, 스칼라 부동 소수점/패킹된 정수/패킹된 부동 소수점/벡터 정수/벡터 부동 소수점 파이프라인, 및/또는 각각이 자신의 스케줄러 유닛, 물리적 레지스터 파일(들) 유닛, 및/또는 실행 클러스터를 갖는 메모리 액세스 파이프라인, 여기서 별개의 메모리 액세스 파이프라인의 경우에 이 파이프라인의 실행 클러스터만이 메모리 액세스 유닛(들)(764)을 갖는 특정 실시예들이 구현됨)을 발생할 수 있기 때문이다. 별개의 파이프라인들이 이용되는 경우, 이들 파이프라인들 중 하나 이상은 비순차적 발행/실행이고 나머지는 순차적일 수 있다는 점도 이해하여야 한다.

[0050] 메모리 액세스 유닛들(764)의 세트는 메모리 유닛(770)에 결합되고, 메모리 유닛은 레벨 2(L2) 캐시 유닛(776)에 결합된 데이터 캐시 유닛(774)에 결합된 데이터 TLB 유닛(772)을 포함한다. 하나의 예시적 실시예에서, 메모리 액세스 유닛들(764)은 로드 유닛, 저장 어드레스 유닛, 및 저장 데이터 유닛을 포함할 수 있으며, 이들 각각은 메모리 유닛(770)에서의 데이터 TLB 유닛(772)에 결합된다. 명령어 캐시 유닛(734)은 메모리 유닛(770)에서의 레벨 2(L2) 캐시 유닛(776)에 추가로 결합된다. L2 캐시 유닛(776)은 하나 이상의 다른 레벨의 캐시에 그리고 결국에는 주 메모리에 결합된다.

[0051] 예시로서, 예시적인 레지스터 리네이밍, 비순차적 발행/실행 코어 아키텍처는 다음과 같이 파이프라인(700)을 구현할 수 있다: 1) 명령어 페칭(738)이 페치 및 길이 디코딩 단(702 및 704)을 실행하고, 2) 디코딩 유닛(74

0)이 디코딩 단(706)을 실행하고, 3) 리네이밍/할당기 유닛(752)이 할당 단(708) 및 리네이밍 단(710)을 실행하고, 4) 스케줄러 유닛(들)(756)이 스케줄링 단(712)을 실행하고, 5) 물리적 레지스터 파일(들) 유닛(들)(758) 및 메모리 유닛(770)이 레지스터 판독/메모리 판독 단(714)을 실행하고, 실행 클러스터(760)가 실행 단(716)을 실행하고, 6) 메모리 유닛(770) 및 물리적 레지스터 파일(들) 유닛(들)(758)이 라이트 백/메모리 기입 단(718)을 실행하고; 7) 다양한 유닛들이 예외 처리 단(722)에 수반될 수 있고, 및 8) 리타이어먼트 유닛(754) 및 물리적 레지스터 파일(들) 유닛(들)(758)이 커밋 단(724)을 실행한다.

[0052] 코어(790)는, 여기 기술된 명령어(들)를 포함하여, 하나 이상의 명령어 세트들[예컨대, (보다 최신의 버전으로 추가된 몇몇 확장을 갖는) x86 명령어 세트; 미국 캘리포니아주 서니베일 소재의 MIPS 테크놀로지사의 MIPS 명령어 세트; 미국 캘리포니아주 서니베일 소재의 ARM 홀딩스사의 (NEON 등의 선택적 부가 확장을 갖는) ARM 명령어 세트]를 지원할 수 있다. 일 실시예에서, 코어(790)는 패킹된 데이터 명령어 세트 확장(예로서, SSE, AVX1, AVX2 등)을 지원하기 위한 로직을 포함하며, 그에 따라 많은 멀티미디어 애플리케이션들에 의해 사용되는 연산들이 패킹된 데이터를 이용하여 실행되는 것을 가능하게 한다.

[0053] 코어는 멀티스레딩(연산들 또는 스레드들의 2개 이상의 병렬 세트들을 실행하는 것)을 지원할 수 있고 또한 시분할 멀티스레딩(time sliced multithreading), (물리적 코어가 동시 멀티스레딩하고 있는 스레드들 각각에 대해 단일의 물리적 코어가 논리적 코어를 제공하는) 동시 멀티스레딩, 또는 이들의 조합(예를 들어, Intel® Hyperthreading 기술에서와 같은 시분할 페칭 및 디코딩 및 그 이후의 동시 멀티스레딩)을 포함하는 다양한 방식으로 멀티스레딩을 지원할 수 있다는 것을 이해하여야 한다.

[0054] 레지스터 리네이밍이 비순차적 실행의 맥락에서 설명되었지만, 레지스터 리네이밍은 순차적 아키텍처에서 이용될 수도 있다는 점을 이해하여야 한다. 프로세서의 예시된 실시예는 또한 별개의 명령어 및 데이터 캐시 유닛들(734/774) 및 공유된 L2 캐시 유닛(776)을 포함하고 있지만, 대안적 실시예들은 명령어와 데이터 모두에 대해 단일의 내부 캐시, 예를 들어, 레벨 1(L1) 내부 캐시를 가지거나 복수 레벨의 내부 캐시를 가질 수 있다. 몇몇 실시예들에서, 시스템은 내부 캐시와 코어 및/또는 프로세서의 외부에 있는 외부 캐시의 조합을 포함할 수 있다. 대안적으로, 캐시 모두가 코어 및/또는 프로세서에 대해 외부적일 수 있다.

[0055] 특정의 예시적인 순차적 코어 아키텍처

[0056] 도 8a-b는 더욱 구체적이고 예시적인 순차적 코어 아키텍처의 블록도를 도시하는데, 이 코어는 칩 내의 (동일한 타입 및/또는 상이한 타입들의 다른 코어들을 포함하는) 여러 개의 로직 블록들 중 하나일 수 있다. 로직 블록들은 애플리케이션에 의존하여, 어떤 고정 기능 로직, 메모리 I/O 인터페이스들, 및 다른 필요한 I/O 로직에 의해 고 대역폭 상호 접속 네트워크(예를 들어, 링 네트워크)를 통해서 통신한다.

[0057] 도 8a는 본 발명의 실시예들에 따라, 온 다이 상호 접속 네트워크(802)에게의 접속부 및 레벨 2(L2) 캐시의 로컬 서브세트(804)와 함께 단일 프로세서 코어를 블록도로 도시한 것이다. 일 실시예에서, 명령어 디코더(800)는 패킹된 데이터 명령어 세트 확장을 갖는 x86 명령어 세트를 지원한다. L1 캐시(806)는 스칼라 유닛 및 벡터 유닛 내로의 캐시 메모리에 대한 저 지연(low-latency) 액세스를 허용한다. (설계를 간략화하기 위한) 일 실시예에서, 스칼라 유닛(808) 및 벡터 유닛(810)은 별개의 레지스터 세트(제각기, 스칼라 레지스터들(812) 및 벡터 레지스터들(814))를 사용하고, 이들 사이에 전송되는 데이터는 메모리에 기입되고 이후 레벨 1(L1) 캐시(806)로부터 리드 백(read back)되는 반면, 본 발명의 대안 실시예들은 상이한 접근법을 사용할 수 있다(예를 들어, 단일 레지스터 세트를 사용하거나, 또는 기입 및 리드 백되지 않고 데이터가 2개의 레지스터 파일 사이에서 전송되게 허용하는 통신 경로를 포함함).

[0058] L2 캐시의 로컬 서브세트(804)는 별개의 로컬 서브세트들이 되도록 분할되는 글로벌 L2 캐시의 일부이며, 프로세서 코어당 하나이다. 각각의 프로세서 코어는 L2 캐시의 그 자신의 로컬 서브세트(804)로의 직접 액세스 경로를 갖는다. 프로세서 코어에 의해 판독되는 데이터는 그 L2 캐시 서브세트(804)에 저장되고 또한 이들 자신의 로컬 L2 캐시 서브세트들에 액세스하는 다른 프로세서 코어들과 병렬로, 빠르게 액세스될 수 있다. 프로세서 코어에 의해 기입되는 데이터는 그 자신의 L2 캐시 서브세트(804)에 저장되고 또한 필요하다면 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 서로 통신하는 것을 허용한다. 각각의 링 데이터 경로는 방향당 1012 비트 폭이다.

[0059] 도 8b는 본 발명의 실시예들에 따른 도 8a에서의 프로세서 코어의 부분의 확대도이다. 도 8b는 벡터 유닛(810) 및 벡터 레지스터(814)에 대한 더 상세한 부분뿐만 아니라 L1 캐시(804)의 L1 데이터 캐시(806A) 부분을 포함한

다. 구체적으로, 벡터 유닛(810)은 16 폭 벡터 프로세싱 유닛(VPU)(16 폭 ALU(828) 참조)이며, 이것은 정수 명령어, 단정밀도 부동 명령어, 및 배정밀도 부동 명령어 중 하나 이상을 실행한다. VPU는 스위즐링 유닛(820)에 의한 레지스터 입력들의 스위즐링(swizzling), 수치 변환 유닛(822A-B)에 의한 수치 변환, 및 메모리 입력에 대한 복제 유닛(824)에 의한 복제를 지원한다. 기입 마스크 레지스터들(826)은 결과적인 벡터 기입들을 술어화(predicate)하는 것을 허용한다.

[0060] 통합 메모리 컨트롤러 및 그래픽을 갖는 프로세서

[0061] 도 9는 본 발명의 실시예들에 따라 2 이상의 코어를 가질 수 있고, 통합 메모리 컨트롤러를 가질 수 있고, 및 통합 그래픽을 가질 수 있는 프로세서(900)의 블록도이다. 도 9의 실선 박스들은 단일 코어(902A), 시스템 에이전트(910), 하나 이상의 버스 컨트롤러 유닛들(916)의 세트를 구비한 프로세서(900)를 예시하는 반면, 점선 박스들의 옵션적 추가는 다중 코어(902A 내지 902N), 시스템 에이전트 유닛(910) 내의 하나 이상의 통합 메모리 컨트롤러 유닛(들)(914)의 세트, 및 특수 목적 로직(908)을 구비한 대안 프로세서(900)를 예시한다.

[0062] 그러므로, 프로세서(900)의 상이한 구현들은 다음을 포함할 수 있다: 1) (하나 이상의 코어들을 포함할 수 있는) 통합 그래픽 및/또는 과학용(처리량) 로직인 특수 목적 로직(908) 및 하나 이상의 범용 코어들(예를 들어, 범용 순차적 코어들, 범용 비순차적 코어들, 이 둘의 조합)인 코어(902A 내지 902N)를 구비한 CPU; 2) 그래픽 및/또는 과학용(처리량)을 위해 주로 의도된 많은 수의 특수 목적 코어들인 코어들(902A 내지 902N)을 구비한 보조프로세서; 및 3) 많은 수의 범용 순차적 코어들인 코어들(902A 내지 902N)을 구비한 보조프로세서. 그러므로, 프로세서(900)는 범용 프로세서, 예를 들어 네트워크 또는 통신 프로세서, 압축 엔진, 그래픽 프로세서, GPGPU(general purpose graphics processing unit), 고 처리량의 MIC(many integrated core) 보조프로세서(30개 이상의 코어를 포함함), 임베디드 프로세서와 같은 보조프로세서 또는 특수 목적 프로세서, 또는 그와 유사한 것일 수 있다. 프로세서는 하나 이상의 칩들상에 구현될 수 있다. 프로세서(900)는 예를 들어, BiCMOS, CMOS, 또는 NMOS와 같은 다수의 프로세스 기술 중 어느 하나를 이용하여 하나 이상의 기판들의 일부가 될 수 있고 및/또는 이들 기판상에 구현될 수 있다.

[0063] 메모리 계층 구조는 코어들 내의 하나 이상의 레벨의 캐시, 공유 캐시 유닛들(906)의 세트 또는 하나 이상의 공유 캐시 유닛들, 및 통합 메모리 컨트롤러 유닛들(914)의 세트에 결합된 외부 메모리(도시 안됨)를 포함한다. 공유 캐시 유닛들(906)의 세트는 레벨 2(L2), 레벨 3(L3), 레벨 4(L4), 또는 다른 레벨의 캐시와 같은 하나 이상의 중간 레벨 캐시, 최종 레벨 캐시(LLC), 및/또는 이들의 조합을 포함할 수 있다. 일 실시예에서 링 기반 상호 접속 유닛(912)이 통합 그래픽 로직(908), 공유 캐시 유닛들(906)의 세트, 및 시스템 에이전트 유닛(910)/통합 메모리 컨트롤러 유닛(들)(914)을 상호 접속하지만, 대안 실시예에서는 이러한 유닛들을 상호 접속하기 위한 공지 기법들 중 임의의 것을 사용할 수 있다. 일 실시예에서, 하나 이상의 캐시 유닛들(906)과 코어들(902A 내지 902N) 사이의 코히런시가 유지된다.

[0064] 몇몇 실시예들에서, 코어들(902A 내지 902N) 중 하나 이상은 멀티스레딩을 할 수 있다. 시스템 에이전트(910)는 코어들(902A 내지 902N)을 조정하고 동작시키는 그런 컴포넌트들을 포함한다. 시스템 에이전트 유닛(910)은 예를 들어 전력 제어 유닛(PCU) 및 디스플레이 유닛을 포함할 수 있다. PCU는 코어들(902A 내지 902N) 및 통합 그래픽 로직(908)의 전력 상태를 조절하는데 필요한 로직 및 컴포넌트일 수 있거나 이들을 포함할 수 있다. 디스플레이 유닛은 하나 이상의 외부적으로 접속된 디스플레이를 구동하기 위한 것이다.

[0065] 코어들(902A 내지 902N)은 아키텍처 명령어 세트의 관점에서 동질적이거나 이질적일 수 있다; 즉 코어들(902A 내지 902N) 중 2개 이상은 동일한 명령어 세트를 실행할 수 있는 한편, 그 외의 것들은 해당 명령어 세트의 서브세트 또는 상이한 명령어 세트만을 실행할 수 있다.

[0066] 예시적인 컴퓨터 아키텍처들

[0067] 도 10 내지 도 13은 예시적인 컴퓨터 아키텍처들의 블록도이다. 랩톱들, 데스크톱들, 핸드헬드 PC들, PDA들(personal digital assistants), 엔지니어링 워크스테이션들, 서버들, 네트워크 디바이스들, 네트워크 허브들, 스위치들, 임베디드 프로세서들, DSP들(digital signal processors), 그래픽 디바이스들, 비디오 게임 디바이스들, 셋톱박스들, 마이크로 컨트롤러들, 휴대 전화들, 휴대용 미디어 플레이어들, 핸드헬드 디바이스들, 및 다양한 그 밖의 전자 디바이스들에 대해 본 기술 분야에 알려진 다른 시스템 설계들 및 구성들도 적합하다. 일반적으로, 본 명세서에 개시된 바와 같은 프로세서 및/또는 다른 실행 로직을 수용할 수 있는 매우 다양한 시스템들 또는 전자 디바이스들이 일반적으로 적합하다.

[0068] 이제 도 10을 참조하면, 본 발명의 일 실시예에 따른 시스템(1000)의 블록도가 도시된다. 시스템(1000)은 하나

이상 프로세서들(1010, 1015)을 포함할 수 있고, 이 프로세서들은 컨트롤러 허브(1020)에 결합된다. 일 실시예에서, 컨트롤러 허브(1020)는 (별개의 칩들상에 있을 수 있는) 입력/출력 허브(IOH; 1050) 및 그래픽 메모리 컨트롤러 허브(GMCH; 1090)를 포함하고; GMCH(1090)는 메모리(1040)와 보조프로세서(1045)가 결합되어 있는 메모리 컨트롤러 및 그래픽 컨트롤러를 포함하고; IOH(1050)는 입력/출력(I/O) 디바이스들(1060)을 GMCH(1090)에 결합한다. 대안적으로, 메모리 컨트롤러와 그래픽 컨트롤러 중 하나 또는 모두는 (여기 기술된) 프로세서 내에 통합되고, 메모리(1040) 및 보조프로세서(1045)는 프로세서(1010), 및 IOH(1050)와 단일 칩 내에 있는 컨트롤러 허브(1020)에게 직접 결합된다.

[0069] 추가 프로세서들(1015)의 옵션적 속성은 도 10에서 파선으로 표시되어 있다. 각각의 프로세서(1010, 1015)는 여기서 기술된 프로세서 코어들 중 하나 이상을 포함할 수 있고, 프로세서(900)의 어떤 버전일 수 있다.

[0070] 메모리(1040)는, 예를 들어, DRAM(dynamic random access memory), PCM(phase change memory), 또는 이 둘의 조합일 수 있다. 적어도 하나의 실시예에 대해, 컨트롤러 허브(1020)는 FSB(frontside bus)와 같은 멀티 드롭 버스, QPI(QuickPath Interconnect)와 같은 포인트 투 포인트 인터페이스, 또는 유사한 접속부(1095)를 통해 프로세서(들)(1010, 1015)와 통신한다.

[0071] 일 실시예에서, 보조프로세서(1045)는, 예를 들어, 고처리량 MIC 프로세서, 네트워크 또는 통신 프로세서, 압축 엔진, 그래픽 프로세서, GPGPU, 임베디드 프로세서, 또는 그와 유사한 것과 같은 특수 목적 프로세서이다. 일 실시예에서, 컨트롤러 허브(1020)는 통합 그래픽 가속기를 포함할 수 있다.

[0072] 아키텍처, 마이크로아키텍처, 열, 전력 소비 특성, 및 그와 유사한 것을 포함하여 이점에 대한 여러 기준들의 관점에서 물리적인 리소스들(1010, 1015) 간에 다양한 차이가 있을 수 있다.

[0073] 일 실시예에서, 프로세서(1010)는 일반 타입의 데이터 처리 연산들을 제어하는 명령어들을 실행한다. 명령어들 내에는 보조프로세서 명령어들이 임베디드될 수 있다. 프로세서(1010)는 이들 보조프로세서 명령어들이 소속된 보조프로세서(1045)에 의해 실행되어야 하는 타입인 것으로 인식한다. 따라서, 프로세서(1010)는 보조프로세서 버스 또는 다른 상호 접속부상에서 이러한 보조프로세서 명령어들(또는 보조프로세서 명령어들을 나타내는 제어 신호들)을 보조프로세서(1045)에게 발행한다. 보조프로세서(들)(1045)는 수신된 보조프로세서 명령어들을 수용하고 실행한다.

[0074] 이제 도 11을 참조하면, 본 발명의 일 실시예에 따른 제1의 더 특정적인 예시적 시스템(1100)의 블록도가 도시된다. 도 11에 도시된 바와 같이, 멀티프로세서 시스템(1100)은 포인트 투 포인트 상호 접속 시스템이고, 포인트 투 포인트 상호 접속부(1150)를 통해 결합되는 제1 프로세서(1170) 및 제2 프로세서(1180)를 포함한다. 프로세서(1170) 및 프로세서(1180) 각각은 프로세서(900)의 어떤 버전일 수 있다. 본 발명의 일 실시예에서, 프로세서들(1170 및 1180)은 제각기 프로세서들(1010 및 1015)인 한편, 보조프로세서(1138)는 보조프로세서(1045)이다. 또 다른 실시예에서, 프로세서들(1170 및 1180)은 제각기 프로세서(1010) 및 보조프로세서(1045)이다.

[0075] 프로세서들(1170 및 1180)이 통합 메모리 컨트롤러(IMC) 유닛들(1172 및 1182)을 제각기 포함하는 것으로 도시되어 있다. 프로세서(1170)는 그 버스 컨트롤러 유닛들의 일부로서 포인트 투 포인트(P-P) 인터페이스들(1176 및 1178)을 포함할 수 있고, 이와 유사하게 제2 프로세서(1180)는 P-P 인터페이스들(1186 및 1188)을 포함한다. 프로세서들(1170 및 1180)은 P-P 인터페이스 회로들(1178 및 1188)을 사용하여 포인트 투 포인트(P-P) 인터페이스(1150)를 통해 정보를 교환할 수 있다. 도 11에 도시된 바와 같이, IMC들(1172, 1182)은 프로세서들을 제각기 메모리들, 즉 메모리(1132) 및 메모리(1134)에 결합시키며, 이 메모리들은 제각기 프로세서들에게 국지적으로 소속된 메인 메모리의 부분들일 수 있다.

[0076] 프로세서들(1170, 1180)은 각각 포인트 투 포인트 인터페이스 회로들(1176, 1194, 1186, 1198)을 이용하여 개별 P-P 인터페이스들(1152, 1154)을 통해서 칩셋(1190)과 정보를 교환할 수 있다. 칩셋(1190)은 옵션으로서 고성능 인터페이스(1139)를 통해 보조프로세서(1138)와 정보를 교환할 수 있다. 일 실시예에서, 보조프로세서(1138)는 예를 들어, 고 처리량 MIC 프로세서, 네트워크 또는 통신 프로세서, 압축 엔진, 그래픽 프로세서, GPGPU, 임베디드 프로세서, 또는 그와 유사한 것과 같은 특수 목적 프로세서이다.

[0077] 공유 캐시(도시 안됨)는 어느 한 프로세서에 포함되거나, 양쪽 프로세서의 외부이지만 여전히 P-P 상호 접속부를 통해 프로세서들과 접속될 수 있어서, 프로세서가 저전력 모드에 놓이는 경우 양쪽 프로세서의 어느 한쪽 또는 모두의 국지적 캐시 정보가 공유 캐시에 저장될 수 있다.

[0078] 칩셋(1190)은 인터페이스(1196)를 통해 제1 버스(1116)에게 결합될 수 있다. 일 실시예에서, 제1 버스(1116)는 PCI 버스, 또는 PCI 익스프레스 버스, 또는 또 다른 3세대 I/O 상호 접속 버스와 같은 버스일 수 있는데, 본 발

명의 범위는 이것들에만 한정되는 것은 아니다.

[0079] 도 11에 도시되는 바와 같이, 다양한 I/O 디바이스들(1114)이, 제1 버스(1116)를 제2 버스(1120)에 결합하는 버스 브리지(1118)와 함께, 제1 버스(1116)에 결합될 수 있다. 일 실시예에서, 보조프로세서들, 고 처리량 MIC 프로세서들, GPGPU들, 가속기들(예를 들어, 그래픽 가속기들 또는 디지털 신호 처리(DSP) 유닛들과 같은 것), FPGA들(field programmable gate arrays), 또는 임의의 다른 프로세서와 같은 하나 이상의 추가 프로세서(들)(1115)가 제1 버스(1116)에 결합된다. 일 실시예에서, 제2 버스(1120)는 LPC(Low Pin Count) 버스일 수 있다. 일 실시예에서, 예를 들어, 키보드 및/또는 마우스(1122), 통신 디바이스들(1127), 및 디스크 드라이브 또는 명령어들/코드 및 데이터(1130)를 포함할 수 있는 다른 대용량 저장 디바이스와 같은 저장 유닛(1128)을 포함하는 다양한 디바이스들이 제2 버스(1120)에 결합될 수 있다. 또한, 오디오 I/O(1124)는 제2 버스(1120)에 결합될 수 있다. 다른 아키텍처들도 가능하다는 점에 유의한다. 예를 들어, 도 11의 포인트 투 포인트 아키텍처 대신에, 시스템은 멀티 드롭 버스 또는 다른 그러한 아키텍처를 구현할 수 있다.

[0080] 도 12를 이제 참조하면, 본 발명의 일 실시예에 따른 제2의 더 특정적인 예시적 시스템(1200)의 블록도가 도시된다. 도 11 및 도 12의 동일한 구성요소들은 동일한 참조 부호들을 가지며, 도 11의 특정 양태들은 도 12의 다른 양태들을 불명확하게 하는 것을 피하기 위해 도 12로부터 생략되었다.

[0081] 도 12는 프로세서들(1170, 1180)이 통합 메모리 및 I/O 제어 로직("CL")(1172 및 1182)을 제각기 포함할 수 있다는 것을 예시한다. 이로 인해, CL(1172, 1182)은 통합 메모리 컨트롤러 유닛들을 포함하고 또한 I/O 제어 로직을 포함한다. 도 12는 메모리들(1132, 1134)이 CL(1172, 1182)에 결합될 뿐만 아니라 I/O 디바이스들(1214)도 제어 로직(1172, 1182)에 결합된다는 것을 예시한다. 레거시 I/O 디바이스들(1215)이 칩셋(1190)에 결합된다.

[0082] 도 13을 이제 참조하면, 본 발명의 실시예에 따른 SoC(1300)의 블록도가 도시된다. 도 9에 있는 유사한 요소들은 동일한 참조 부호를 갖는다. 또한, 점선 박스들은 더욱 진보된 SoC들에 관한 옵션적 특징들이다. 도 13에서, 상호접속부 유닛(들)(1302)이: 하나 이상의 코어들(202A 내지 202N)의 세트 및 공유 캐시 유닛(들)(906)을 포함하는 애플리케이션 프로세서(1310); 시스템 에이전트 유닛(910); 버스 컨트롤러 유닛(들)(916); 통합 메모리 컨트롤러 유닛(들)(914); 통합 그래픽 로직, 이미지 프로세서, 오디오 프로세서, 및 비디오 프로세서를 포함할 수 있는 하나 이상의 보조프로세서(1320) 또는 그 세트; SRAM(static random access memory) 유닛(1330); DMA(direct memory access) 유닛(1332); 및 하나 이상의 외부 디스플레이에 결합하기 위한 디스플레이 유닛(1340)에 결합된다. 일 실시예에서, 보조프로세서(들)(1320)는, 예를 들어, 네트워크 또는 통신 프로세서, 압축 엔진, GPGPU, 고 처리량 MIC 프로세서, 임베디드 프로세서, 및 그와 유사한 것과 같은 특수 목적 프로세서를 포함한다.

[0083] 여기에 개시된 메커니즘들의 실시예들은 하드웨어, 소프트웨어, 펌웨어, 또는 이러한 구현 접근법들의 조합으로 구현될 수 있다. 본 발명의 실시예들은 적어도 하나의 프로세서, (휘발성 및/또는 비휘발성 메모리 및/또는 스토리지 요소들을 포함하는) 스토리지 시스템, 적어도 하나의 입력 디바이스, 및 적어도 하나의 출력 디바이스를 포함하는 프로그램가능 시스템상에서 실행되는 컴퓨터 코드 또는 컴퓨터 프로그램들로서 구현될 수 있다.

[0084] 도 11에 예시된 코드(1130)와 같은 프로그램 코드는 여기서 기술된 기능들을 실행하고 출력 정보를 발생하도록 입력 명령어들에 적용될 수 있다. 출력 정보는 공지 방식으로 하나 이상의 출력 디바이스들에게 적용될 수 있다. 본 발명의 목적을 위해, 처리 시스템은 예를 들어 DSP(digital signal processor), 마이크로컨트롤러, ASIC(application specific integrated circuit), 또는 마이크로프로세서와 같은 프로세서를 갖는 임의의 시스템을 포함한다.

[0085] 프로그램 코드는 처리 시스템과 통신하기 위해 고급의 절차적 또는 객체 지향적 프로그래밍 언어로 구현될 수 있다. 프로그램 코드는 또한 원하는 경우 어셈블리어 또는 기계어로 구현될 수 있다. 사실상, 여기 기술된 메커니즘들은 어떠한 특정의 프로그래밍 언어로만 그 범위가 한정되지 않는다. 어느 경우에나, 언어는 컴파일링되거나 인터프리팅된 언어일 수 있다.

[0086] 적어도 일 실시예의 하나 이상의 양태들은 기계에 의해 판독될 때 기계로 하여금 본 명세서에서 설명되는 기술들을 실행하기 위한 논리를 제조하게 하는, 프로세서 내의 다양한 논리를 표현하는, 기계 판독 가능 매체상에 저장된 대표적인 명령어들에 의해 구현될 수 있다. "IP 코어들"로서 알려진 그러한 표현들은 유형의 기계 판독 가능 매체 상에 저장될 수 있으며, 다양한 고객들 또는 제조 설비에 제공되어, 논리 또는 프로세서를 실제로 제조하는 제조 기계들 내에 로드될 수 있다.

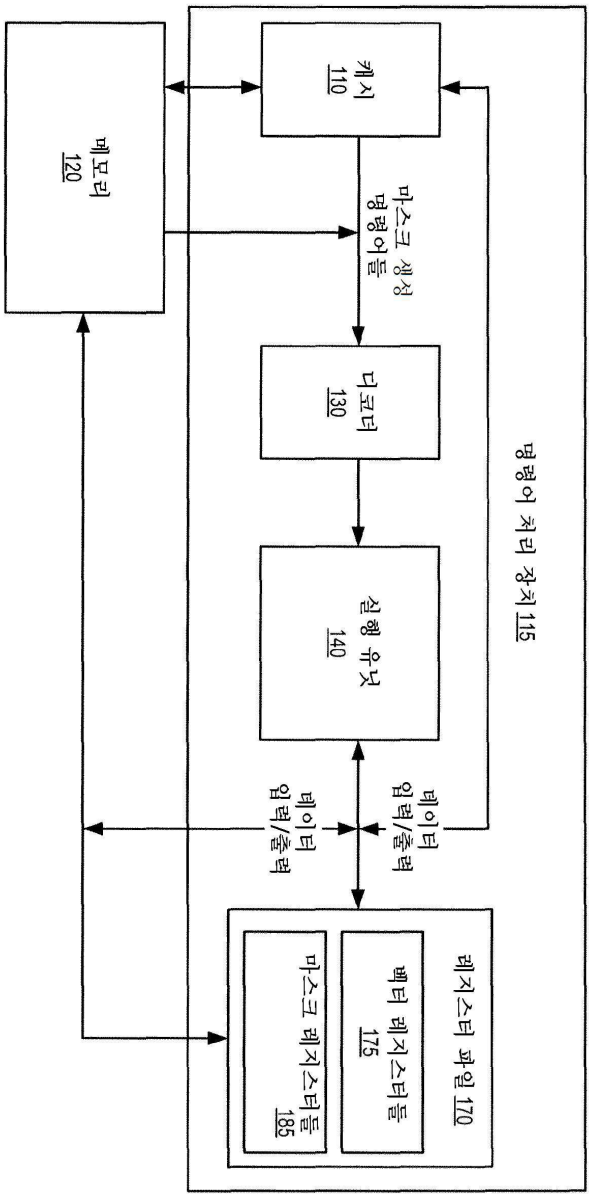
[0087] 그러한 기계 판독 가능 저장 매체는 하드 디스크들, 임의의 다른 유형의 디스크들로서 플로피 디스크들, 광 디스크들, CD-ROM들(compact disk read-only memories), CD-RW들(compact disk rewritable's), 및 광자기 디스크들을 포함하는 디스크, ROM들(read-only memories), 예를 들어 DRAM들(dynamic random access memories), SRAM들(static random access memories)과 같은 RAM들(random access memories), EPROM들(erasable programmable read-only memories), 플래시 메모리, EEPROM들(electrically erasable programmable read-only memories)과 같은 반도체 디바이스들, PCM(phase change memory), 자기 또는 광 카드들, 또는 전자적 명령어들을 저장하기에 적절한 임의의 다른 유형의 매체와 같은 저장 매체를 포함하여, 기계 또는 디바이스에 의해 제조되거나 형성되는 물품들의 비 일시적 유형의 어레이들을 포함할 수 있는데, 이것에만 한정되지는 않는다.

[0088] 따라서, 본 발명의 실시예들은 명령어들을 포함하거나 또는 본 명세서에 설명된 구조들, 회로들, 장치들, 프로세서들 및/또는 시스템 특징들을 정의하는, HDL(Hardware Description Language)과 같은 설계 데이터를 포함하는 비 일시적인 유형의 기계 판독 가능 매체를 또한 포함한다. 이러한 실시예들은 프로그램 제품들로도 지칭될 수 있다.

[0089] 특정의 예시적 실시예들이 설명되고 첨부 도면들에서 도시되었지만, 그러한 실시예들은 단지 설명을 위한 것일 뿐이고 발명의 넓은 범위를 제한하는 것이 아니며, 이 개시를 연구할 때 관련 기술 분야의 통상의 기술자는 다양한 다른 변형들을 생각해낼 수 있으므로 이 발명은 도시되고 설명된 그 특정 구성들 및 어레이들에만 제한되지 않는다는 것을 이해해야 한다. 빠르게 성장하고 또한 추가 향상이 용이하게 예견되지 않는 이와 같은 기술 영역에서, 개시된 실시예들은 본 개시의 원리들 또는 첨부된 청구범위를 벗어나지 않고서 기술적 향상들을 가능하게 함으로써 용이하게 되듯이 배치 및 상세 사항에 있어서 쉽게 수정 가능할 수 있다.

도면

도면1



도면2a

```
KSHLONES[B/W/D/Q] k1, r2                ; 목적지, SRC1

KL← 8, 16, 32, 64                       ; 마스크 비트들의 수(연상 기호에 기초)
N = min(KL, SRC1)
for (i=0; i<N; i++){
        k1<<1+1
}`
```

도면2b

```
SHLONES[B/W/D/Q] r1, r2          ;목적지, SRC1

KL← 8, 16, 32, 64                ;마스크 비트들의 수(연상 기호에 기초)
N = min(KL, SRC1)
for (i=0; i<N; i++){
        r1<<1+1
}`
```

도면2c

```
KSHLONES[B/W/D/Q] k1, r2         ;목적지, SRC1

KL← 8, 16, 32, 64                ;마스크 비트들의 수(연상 기호에 기초)
N = min(KL, SRC1)
for (i=0; i<N; i++){
        k1<<1+1
}`
if (N==0) ZF = 1
if (N==KL) CF = 1
```

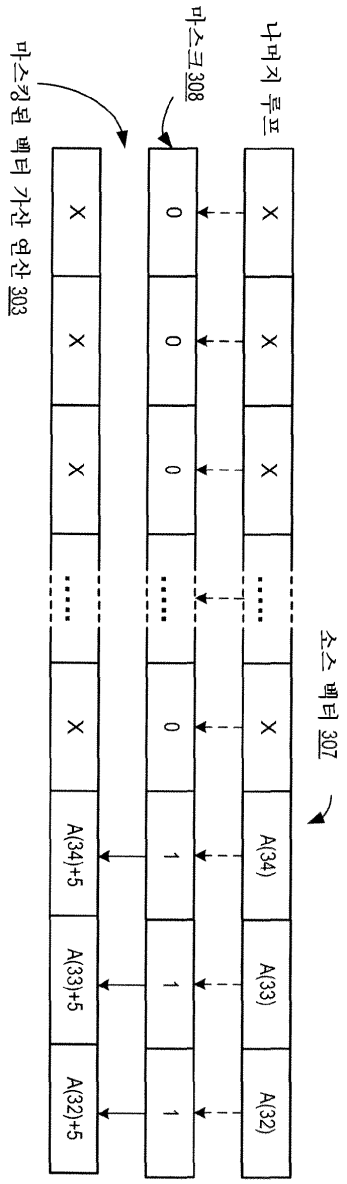
도면3a

첫번째 루프	A(15)	A(14)	A(13)	....	A(3)	A(2)	A(1)	A(0)
두번째 루프	A(31)	A(30)	A(29)	....	A(19)	A(18)	A(17)	A(16)
나머지 루프	X	X	X	....	X	A(34)	A(33)	A(32)

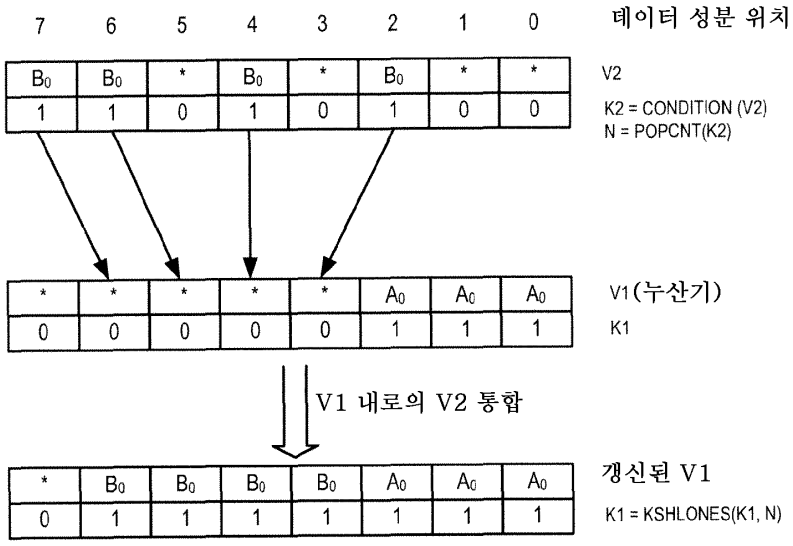
도면3b

첫번째 루프	X	X	X	....	X	X	A(1)	A(0)
두번째 루프	A(17)	A(16)	A(15)	....	A(5)	A(4)	A(3)	A(2)
세번째 루프	A(33)	A(32)	A(31)	....	A(21)	A(20)	A(19)	A(18)
나머지 루프	X	X	X	....	X	X	X	A(34)

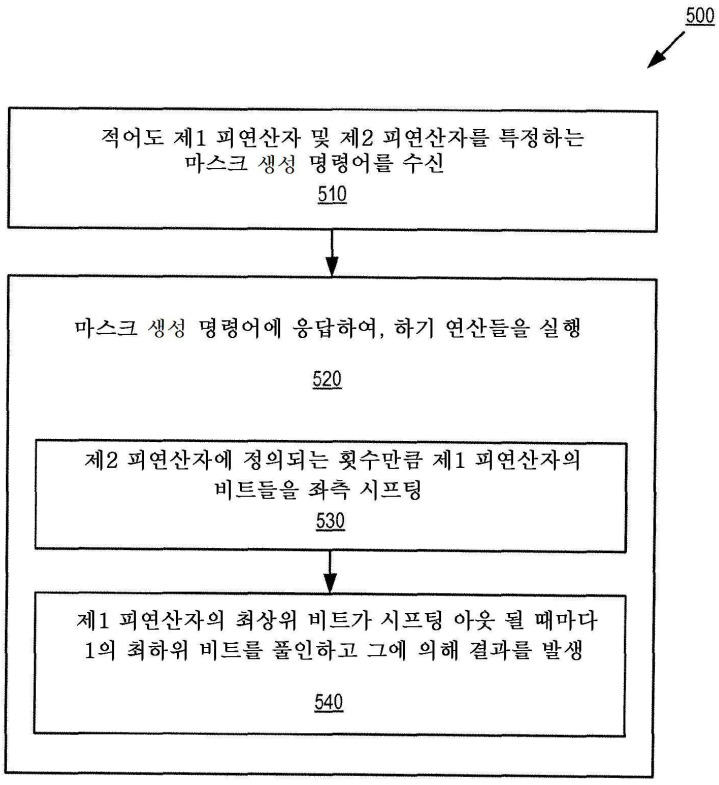
도면3c


나머지 루프
마스크 308
마스킹된 벡터 가산 연산 303
소스 벡터 307
X
X
X
.....
X
A(34)
A(33)
A(32)
0
0
0
.....
0
1
1
1
X
X
X
.....
X
A(34)+5
A(33)+5
A(32)+5

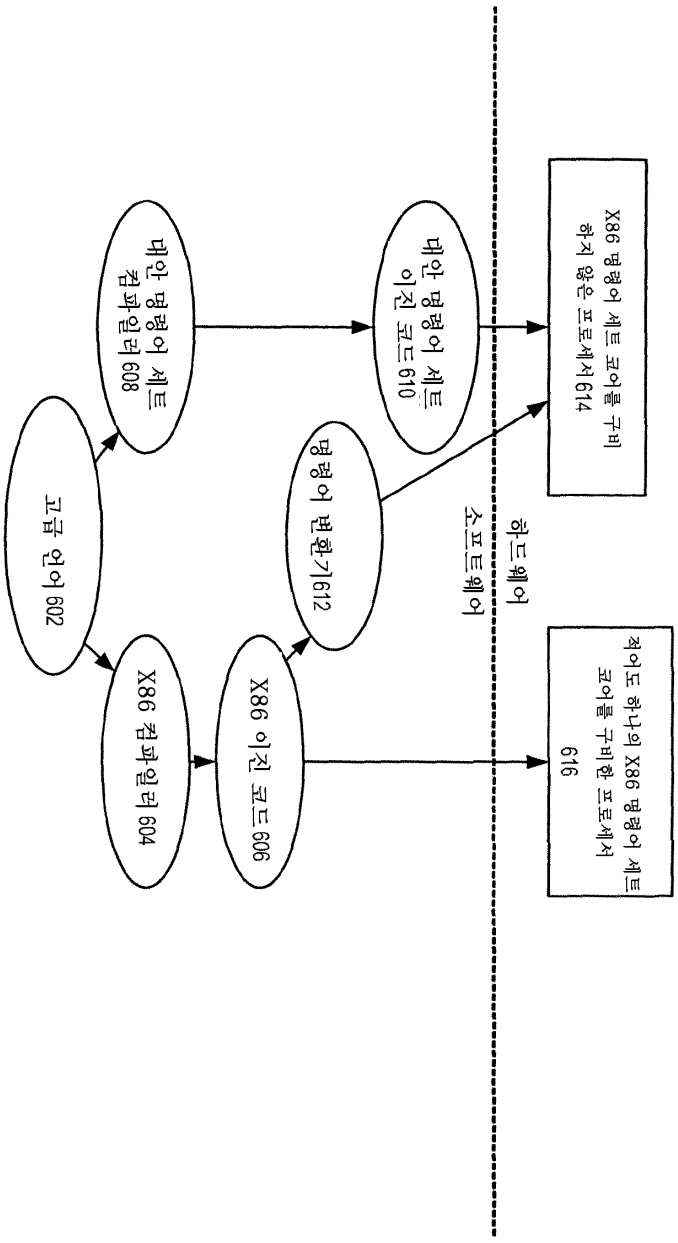
도면4


7 6 5 4 3 2 1 0
데이터 성분 위치
B0 B0 * B0 * B0 * *
1 1 0 1 0 1 0 0
V2
K2 = CONDITION (V2)
N = POPCNT(K2)
* * * * * A0 A0 A0
0 0 0 0 0 1 1 1
V1(누산기)
K1
V1 내로의 V2 통합
* B0 B0 B0 B0 A0 A0 A0
0 1 1 1 1 1 1 1
갱신된 V1
K1 = KSHLONES(K1, N)

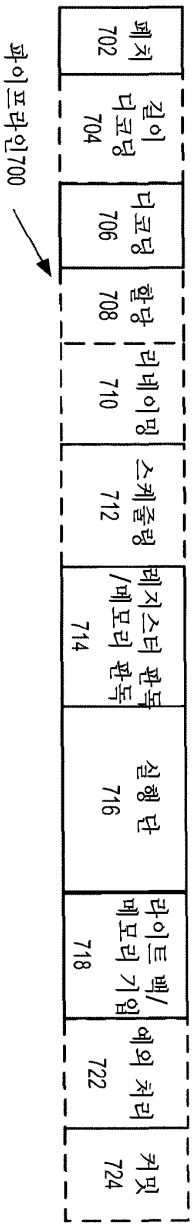
도면5


500
적어도 제1 피연산자 및 제2 피연산자를 특정하는 마스크 생성 명령어를 수신
510
마스크 생성 명령어에 응답하여, 하기 연산들을 실행
520
제2 피연산자에 정의되는 횟수만큼 제1 피연산자의 비트들을 좌측 시프팅
530
제1 피연산자의 최상위 비트가 시프팅 아웃 될 때마다 1의 최하위 비트를 풀인하고 그에 의해 결과를 발생
540

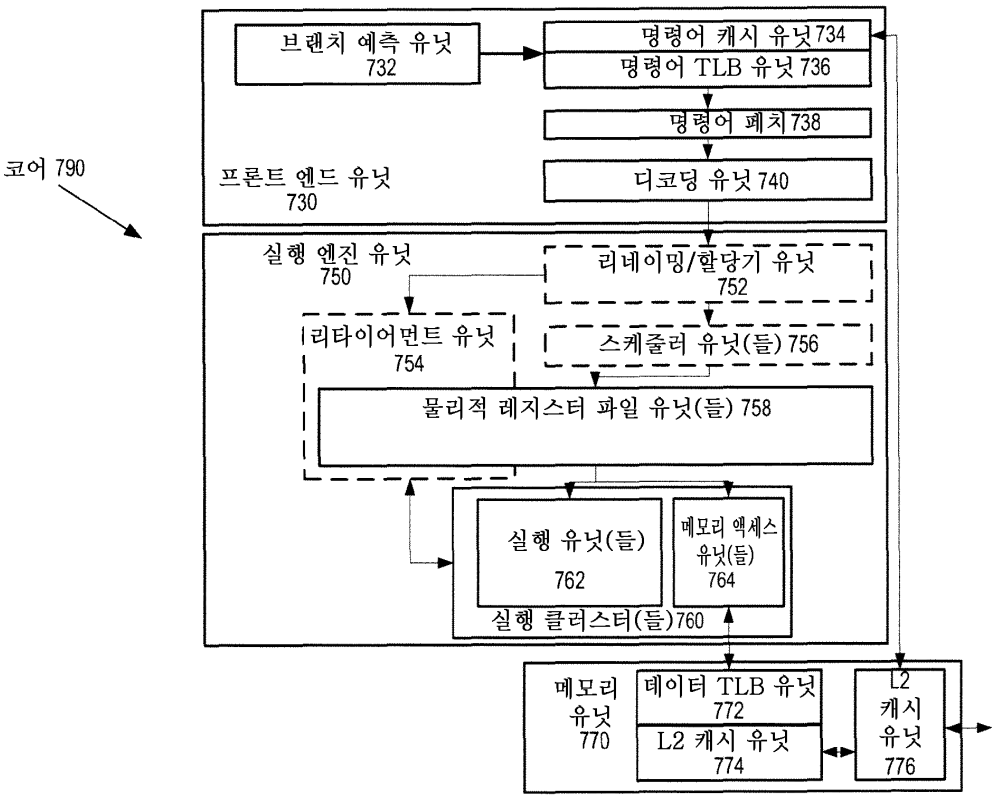
도면6


고급 언어 602
대안 명령어 세트 컴파일러 608
X86 컴파일러 604
명령어 변환기612
대안 명령어 세트 이진 코드 610
X86 이진 코드 606
하드웨어
소프트웨어
X86 명령어 세트 코어를 구비하지 않은 프로세서 614
적어도 하나의 X86 명령어 세트 코어를 구비한 프로세서 616

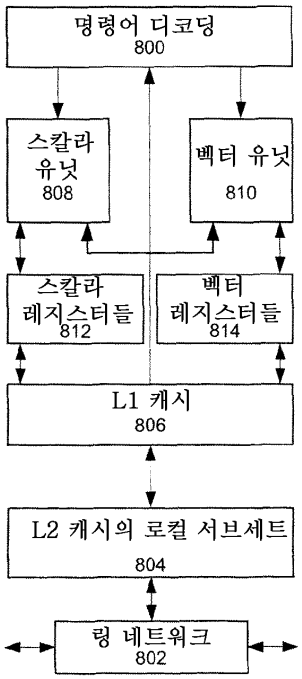
도면7a


파이프라인700
페치 702
길이 디코딩 704
디코딩 706
할당 708
리네이밍 710
스케줄링 712
레지스터 판독 /메모리 판독 714
실행 단 716
라이트 백/ 메모리 기입 718
예외 처리 722
커밋 724

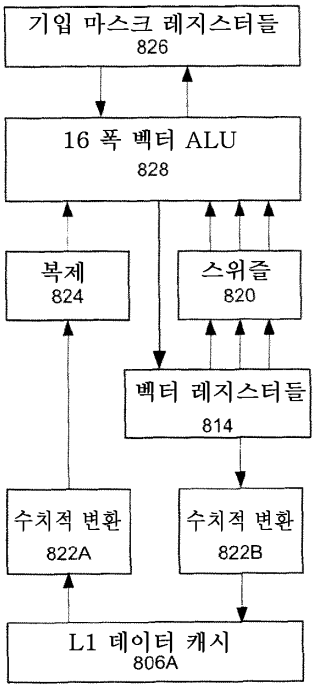
도면7b


코어 790
프론트 엔드 유닛
730
브랜치 예측 유닛
732
명령어 캐시 유닛734
명령어 TLB 유닛 736
명령어 페치738
디코딩 유닛 740
실행 엔진 유닛
750
리네이밍/할당기 유닛
752
리타이어먼트 유닛
754
스케줄러 유닛(들) 756
물리적 레지스터 파일 유닛(들) 758
실행 유닛(들)
762
메모리 액세스
유닛(들)
764
실행 클러스터(들)760
메모리
유닛
770
데이터 TLB 유닛
772
L2 캐시 유닛
774
L2
캐시
유닛
776

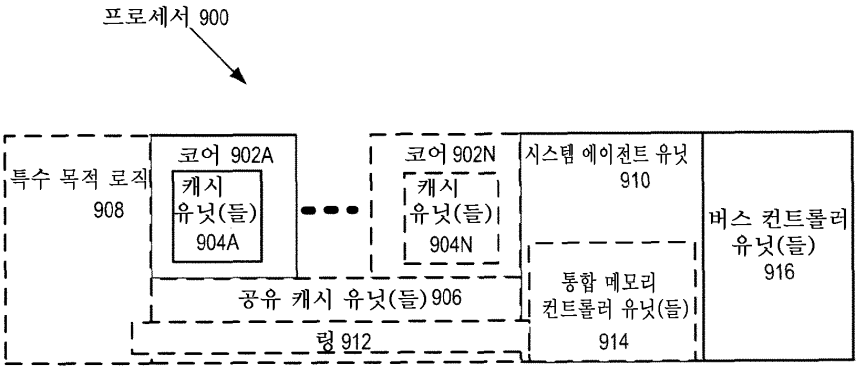
도면8a


명령어 디코딩
800
스칼라
유닛
808
벡터 유닛
810
스칼라
레지스터들
812
벡터
레지스터들
814
L1 캐시
806
L2 캐시의 로컬 서브세트
804
링 네트워크
802

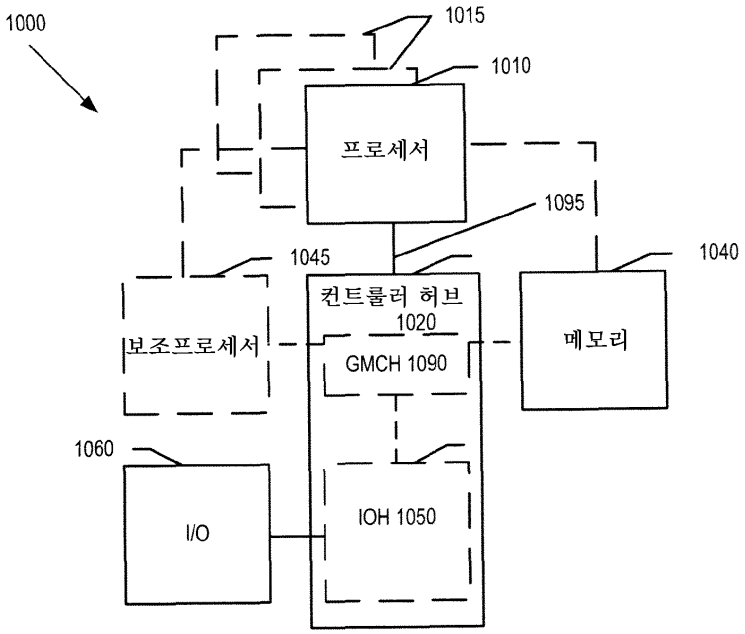
도면8b


기입 마스크 레지스터들
826
16 폭 벡터 ALU
828
복제
824
스위즐
820
벡터 레지스터들
814
수치적 변환
822A
수치적 변환
822B
L1 데이터 캐시
806A

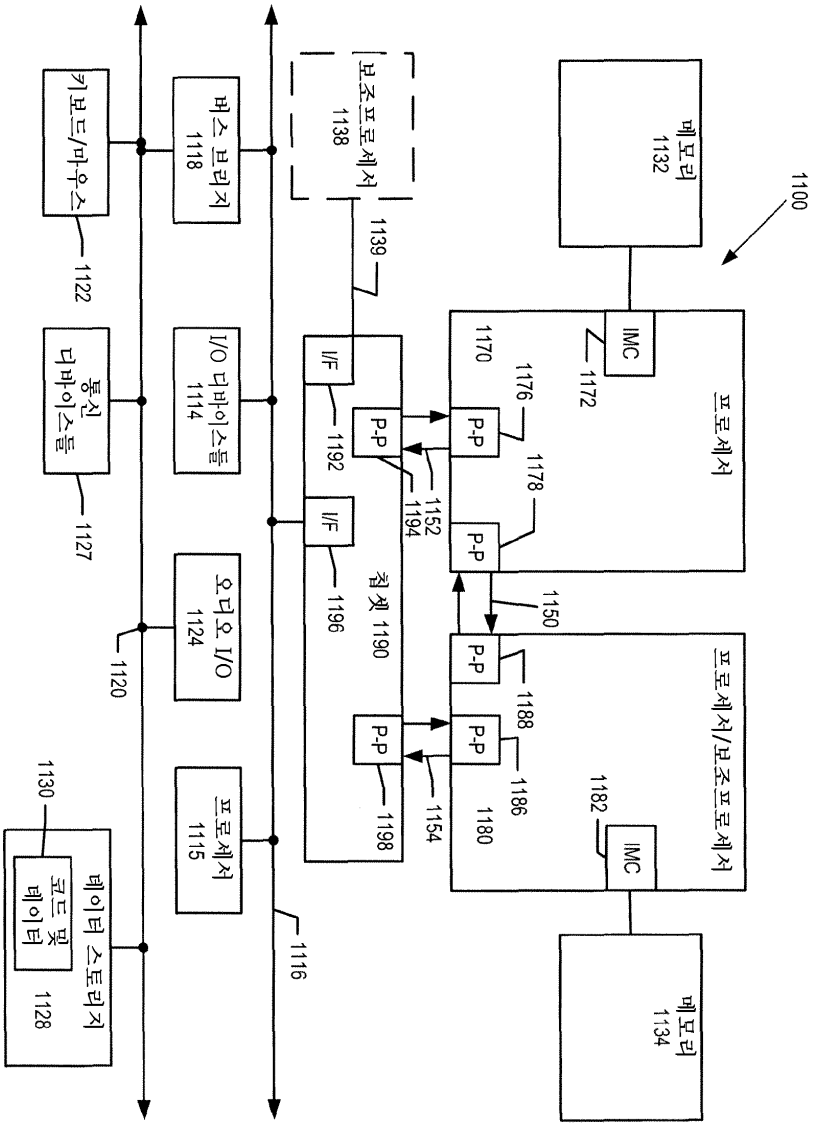
도면9


프로세서 900
특수 목적 로직
908
코어 902A
캐시
유닛(들)
904A
코어 902N
캐시
유닛(들)
904N
시스템 에이전트 유닛
910
버스 컨트롤러
유닛(들)
916
통합 메모리
컨트롤러 유닛(들)
914
공유 캐시 유닛(들) 906
링 912

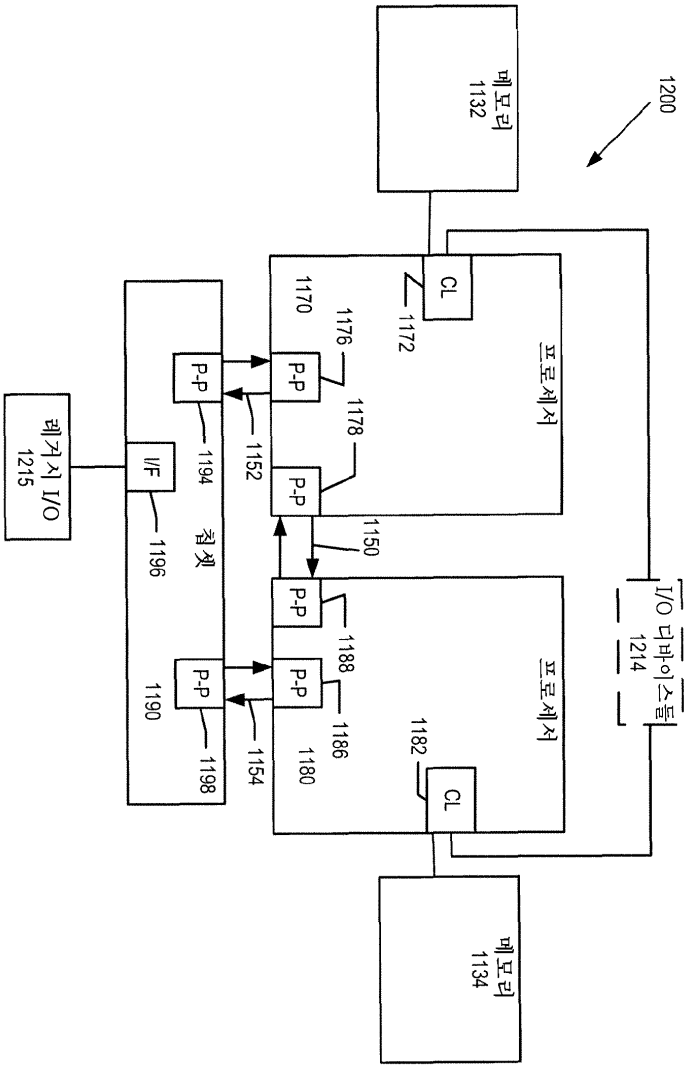
도면10


1000
1015
1010
프로세서
1095
1045
1040
컨트롤러 허브
1020
보조프로세서
GMCH 1090
메모리
1060
I/O
IOH 1050

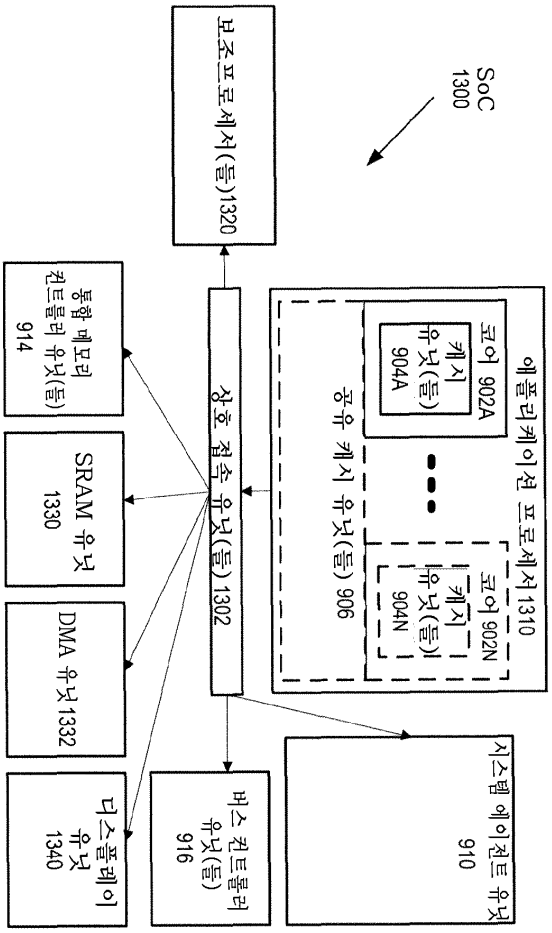
도면11


1100
메모리
1132
IMC
1172
프로세서
1170
P-P
1176
P-P
1178
P-P
1150
프로세서/보조프로세서
P-P
1188
P-P
1186
1180
IMC
1182
메모리
1134
보조프로세서
1138
1139
I/F
1192
I/F
1196
칩셋 1190
P-P
1152
1194
P-P
1154
1198
키보드/마우스
1122
버스 브리지
1118
통신 디바이스들
1127
I/O 디바이스들
1114
오디오 I/O
1124
1120
프로세서
1115
1116
데이터 스토리지
1130
코드 및 데이터
1128

도면12


1200
메모리
1132
CL
1172
1170
1176
1178
1150
프로세서
P-P
1152
1194
I/F
1196
칩셋
1190
1198
1154
레거시 I/O
1215
1188
1186
1180
1182
I/O 디바이스들
1214
메모리
1134

도면13


SoC
1300
보조프로세서(들)1320
통합 메모리 컨트롤러 유닛(들)
914
SRAM 유닛
1330
DMA 유닛1332
디스플레이 유닛
1340
상호 접속 유닛(들) 1302
버스 컨트롤러 유닛(들)
916
시스템 에이전트 유닛
910
애플리케이션 프로세서1310
코어 902A
캐시 유닛(들)
904A
공유 캐시 유닛(들) 906
코어 902N
캐시 유닛(들)
904N