(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2017-0097633
(43) 공개일자 2017년08월28일

(51) 국제특허분류(Int. Cl.)
G06F 9/30 (2017.01) *G06F 9/38* (2006.01)
(52) CPC특허분류
G06F 9/30145 (2013.01)
G06F 9/3001 (2013.01)
(21) 출원번호 10-2017-7013959
(22) 출원일자(국제) 2015년11월23일
심사청구일자 없음
(85) 번역문제출일자 2017년05월23일
(86) 국제출원번호 PCT/US2015/062098
(87) 국제공개번호 WO 2016/105767
국제공개일자 2016년06월30일
(30) 우선권주장
14/582,053 2014년12월23일 미국(US)

(71) 출원인
인텔 코포레이션
미합중국 캘리포니아 95054 산타클라라 미션 칼리지 블러바드 2200
(72) 발명자
라이, 패트릭 피.
미국 94539 캘리포니아주 프리몬트 튤립 테라스 49200
손다그, 타일러 엔.
미국 95054 캘리포니아주 산타 클라라 메일스탑 에스씨12-331 미션 칼리지 블러바드 2200
(뒷면에 계속)
(74) 대리인
양영준, 김연송, 백만기

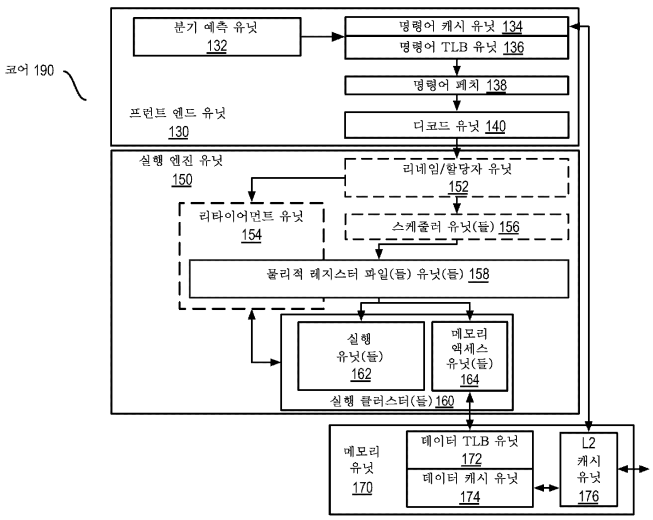
전체 청구항 수 : 총 24 항

(54) 발명의 명칭 **융합된 단일 사이클 증가-비교-점프를 수행하기 위한 명령어 및 로직**

(57) 요 약

일 실시예에서, 명령어 세트 아키텍처의 다수의 매크로 명령어들을 단일 매크로 명령어로 융합하기 위해 바이너리 변환이 사용된다. 융합 가능한 명령어 시퀀스는 증가, 비교 및 점프 명령어들의 시퀀스를 포함한다. 일 실시예에서, 처리 디바이스는 융합된 매크로 명령어에 대한 지원을 제공한다. 일 실시예에서, 처리 디바이스는 프로세서 파이프라인의 단일 실행 스테이지 내에서 융합된 매크로 명령어를 실행한다. 일 실시예에서, 융합된 매크로 명령어는 단일 실행 사이클 내에서 수행된다.

대 표 도



(52) CPC특허분류
G06F 9/30021 (2013.01)
G06F 9/30058 (2013.01)
G06F 9/383 (2013.01)
(72) 발명자
윈켈, 세바스티안
미국 94024 캘리포니아주 로스 알토스 팜 로드 28
세칼라키스, 폴리크로니스
미국 95134 캘리포니아주 산 호세 아파트먼트 1417 크레슨트 빌리지 서클 330
셔크만, 에단
미국 95054 캘리포니아주 산타 클라라 줄리엣 레인 3600

명 세 서

청구범위

청구항 1

처리 장치로서,

융합된 명령어를 제1 피연산자 및 제2 피연산자를 포함하는 디코딩된 융합된 명령어로 디코딩하는 디코드 로직; 및

증가 연산, 비교 연산, 및 점프 연산을 단일 머신-레벨 매크로 명령어로서 수행하기 위해 상기 융합된 디코딩된 명령어를 실행하는 실행 유닛

을 포함하는, 처리 장치.

청구항 2

제1항에 있어서,

상기 융합된 명령어를 페치하는 명령어 페치 유닛 및 상기 증가 연산의 결과를 상기 제1 피연산자 또는 제2 피연산자에 의해 지정된 레지스터에 커밋하는 레지스터 파일 유닛을 추가로 포함하는, 처리 장치.

청구항 3

제1항에 있어서,

상기 실행 유닛은 상기 증가 연산 및 비교 연산을 수행하는 산술 논리 유닛(ALU) 및 상기 점프 연산을 수행하는 점프 실행 유닛을 포함하는, 처리 장치.

청구항 4

제1항에 있어서,

상기 제1 피연산자 및 제2 피연산자는 상기 비교 연산과 관련되고, 상기 제1 피연산자 또는 제2 피연산자 중 하나는 상기 증가 연산과 관련되는, 처리 장치.

청구항 5

제4항에 있어서,

상기 디코딩된 융합된 명령어는 상기 점프 연산과 관련된 점프 타깃 피연산자를 추가로 포함하는, 처리 장치.

청구항 6

제5항에 있어서,

상기 실행 유닛은 추가로 단일 사이클에서 상기 증가 연산, 비교 연산 및 점프 연산을 실행하는 것인, 처리 장치.

청구항 7

제5항에 있어서,

상기 점프 연산은 상기 비교 연산을 조건으로 하는, 처리 장치.

청구항 8

제7항에 있어서,

상기 점프 연산은 상기 비교 연산에 의해 설정된 제로 플래그를 조건으로 하는, 처리 장치.

청구항 9

제7항에 있어서,

상기 점프 연산은 상기 비교 연산에 의해 설정된 캐리 플래그를 조건으로 하는, 처리 장치.

청구항 10

제7항에 있어서,

상기 점프 연산은 상기 비교 연산에 의해 설정된 오버플로우 플래그를 조건으로 하는, 처리 장치.

청구항 11

제7항에 있어서,

상기 점프 연산은 상기 비교 연산에 의해 설정된 부호 플래그를 조건으로 하는, 처리 장치.

청구항 12

다수의 매크로 명령어들을 단일 매크로 명령어로 융합하는 방법으로서,

증가 명령어, 비교 명령어 및 점프 명령어를 포함하는 명령어 시퀀스에 대해 제1 소스 코드 블록을 스캔하는 단계;

상기 명령어 시퀀스를 검출한 후, 데이터 종속성에 대해 상기 명령어 시퀀스를 스캔하는 단계;

상기 명령어 시퀀스 내의 코드 프래그먼트들을 재순서화하는 단계; 및

증가 명령어, 비교 명령어 및 점프 명령어의 세트를, 프로세서에 의해 실행될 때 상기 프로세서로 하여금 증가 연산, 비교 연산 및 점프 연산을 수행하게 하는 단일 융합된 명령어로 대체하는 단계

를 포함하는, 방법.

청구항 13

제12항에 있어서,

상기 프로세서는 단일 프로세서 파이프라인 실행 사이클에서 상기 융합된 명령어를 실행하는 것인, 방법.

청구항 14

제13항에 있어서,

상기 프로세서는 산술 논리 유닛(ALU)에 대한 캐리 입력을 어설트(assert)함으로써 제1 피연산자 또는 제2 피연산자를 증가시키면서 상기 ALU를 사용하여 상기 증가 명령어 및 비교 명령어와 관련된 제1 피연산자 및 제2 피연산자에 대해 비교 연산을 수행함으로써 상기 사이클에서 상기 융합된 명령어를 수행하는 것인, 방법.

청구항 15

제14항에 있어서, 상기 점프 연산이 수행되어야 하는지를 결정하기 위해 상기 프로세서 내의 점프 실행 유닛을 사용하여 상기 비교 연산에 의해 상기 ALU로부터의 플래그 출력을 평가하는 단계를 추가로 포함하는, 방법.

청구항 16

제15항에 있어서,

상기 프로세서는 분기 예측 프로세서이고, 상기 점프 명령어와 관련된 분기가 실행될 것임을 예측하는 단계, 상기 융합된 명령어의 점프 연산이 실행되는지를 결정하는 단계, 및 상기 점프 명령어에 대해 예측된 상기 분기를 해결(resolve)하는 단계를 추가로 포함하는, 방법.

청구항 17

제12항 내지 제16항 중 어느 한 항에서와 같은 방법을 수행하기 위한 수단을 포함하는 시스템.

청구항 18

하나 이상의 프로세서에 의해 실행될 때, 상기 하나 이상의 프로세서로 하여금 제12항 내지 제16항 중 어느 한 항에서와 같은 방법을 포함하는 동작들을 수행하게 하는 명령어를 저장하는 비일시적 머신 판독 가능 매체.

청구항 19

융합된 매크로 명령어를 수행하는 방법으로서,

융합된 명령어를 제1 피연산자 및 제2 피연산자를 포함하는 디코딩된 융합된 명령어로 디코딩하는 단계; 및

증가 연산, 비교 연산 및 점프 연산을 단일 머신-레벨 매크로 명령어로서 수행하기 위해 상기 융합 디코딩된 명령어를 실행하는 단계

를 포함하는, 방법.

청구항 20

제19항에 있어서,

단일 실행 사이클에서 상기 융합된 디코딩된 명령어를 실행하는 단계를 추가로 포함하는, 방법.

청구항 21

제19항에 있어서,

상기 연산들의 결과에 기초하여 다음 명령어 포인터를 업데이트하는 단계를 추가로 포함하는, 방법.

청구항 22

제19항에 있어서,

상기 증가 연산의 결과를 상기 제1 피연산자 또는 제2 피연산자에 의해 지시되는 레지스터로 커밋하는 단계를 추가로 포함하는, 방법.

청구항 23

제19항에 있어서, 상기 점프 연산의 결과에 기초하여 분기 예측을 해결하는 단계를 추가로 포함하는, 방법.

청구항 24

적어도 하나의 머신에 의해 수행되는 경우, 상기 적어도 하나의 머신으로 하여금 제19항 내지 제23항 중 어느 한 항에서와 같은 방법을 포함하는 동작들을 수행하는 적어도 하나의 집적 회로를 제조하게 하는 데이터가 저장된 머신 판독 가능 매체.

발명의 설명

기 술 분 야

[0001] 본 발명은 프로세서 또는 다른 처리 로직에 의해 실행될 때, 다수의 명령어를 하나의 머신 명령어로 융합하는 것을 포함하는 논리적, 수학적, 또는 다른 기능적 연산을 수행하는 처리 로직, 마이크로프로세서, 및 관련 명령어 세트 아키텍처의 분야에 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. 바이너리 변환("BT")은 하나의 소스("게스트") ISA 용으로 작성된 바이너리를 또 다른 타깃("호스트") ISA로 변환하는 일반적인 기술이다. BT를 사용하여, 하이 레벨 소스 코드를 다시 컴파일하거나 로우 레벨 어셈블리 코드를 다시 작성하지 않고도 하나의 프로세서 ISA 용으로 작성된 애플리케이션 바이너리를 상이한 아키텍처의 프로세

서상에서 실행하는 것이 가능하다. 대부분의 레거시 컴퓨터 애플리케이션은 바이너리 포맷으로만 사용 가능하기 때문에, BT는 프로세서가 그것을 위해 작성되지 않고 사용 가능하지 않은 애플리케이션들을 실행할 수 있게 할 가능성으로 인해 매우 매력적이다. 바이너리 변환은 동적으로 또는 정적으로 수행될 수 있다. 동적 BT(DBT)는 애플리케이션이 실행될 때 런타임에 바이너리 변환을 수행한다. 정적 BT(SBT)는 바이너리가 실행되기 전에 바이너리에 대해 수행된다.

도면의 간단한 설명

[0003] 실시예들은 첨부 도면들에서 예시로서 제한이 아니라 예로서 예시된다.

도 1a는 실시예들에 따른, 예시적인 순차적(in-order) 페치, 디코드, 리타이어 파이프라인 및 예시적인 레지스터 리네이밍, 비순차적(out-of-order) 발행/실행 파이프라인 양자 모두를 도시하는 블록도이다.

도 1b는 실시예들에 따라 프로세서에 포함될, 순차적 페치, 디코드, 리타이어 코어의 예시적인 실시예 및 예시적인 레지스터 리네이밍, 비순차적 발행/실행 아키텍처 코어 양자 모두를 도시하는 블록도이다.

도 2a 및 도 2b는 더 구체적인 예시적인 순차적 코어 아키텍처의 블록도이다.

도 3은 통합 메모리 제어기 및 특수 목적 로직을 갖는 단일 코어 프로세서 및 멀티코어 프로세서의 블록도이다.

도 4는 일 실시예에 따른 시스템의 블록도이다.

도 5는 일 실시예에 따른 제2 시스템의 블록도이다.

도 6은 일 실시예에 따른 제3 시스템의 블록도이다.

도 7은 일 실시예에 따른 시스템 온 칩(SoC)의 블록도이다.

도 8은 실시예들에 따른 소스 명령어 세트에서의 바이너리 명령어들을 타깃 명령어 세트에서의 바이너리 명령어들로 변환하기 위한 소프트웨어 명령어 변환기의 사용을 대조하는 블록도를 도시한다.

도 9a 및 도 9b는 일 실시예에 따른, 융합된 increment_compare_jump 연산을 수행하기 위한 비트 조작 연산을 도시하는 블록도이다.

도 10a 및 도 10b는 일 실시예에 따른, increment_compare_jump 명령어들의 예시적인 프로세서 구현을 도시하는 블록도이다.

도 11은 일 실시예에 따른 융합된 increment_compare_jump 연산을 수행하기 위한 로직을 포함하는 처리 시스템의 블록도이다.

도 12는 일 실시예에 따른, 예시적인 융합된 increment_compare_jump 명령어를 처리하기 위한 로직에 대한 흐름도이다.

도 13a 및 도 13b는 실시예들에 따른 일반적인 벡터 친화적 명령어 포맷 및 그것의 명령어 템플릿을 나타내는 블록도이다.

도 14a 내지 도 14d는 본 발명의 실시예들에 따른 예시적인 특정 벡터 친화적 명령어 포맷을 나타내는 블록도이다.

도 15는 일 실시예에 따른 스칼라 및 벡터 레지스터 아키텍처의 블록도이다.

발명을 실시하기 위한 구체적인 내용

[0004] 게스트와 호스트 ISA 간 바이너리 변환 외에도, 단일 ISA 내에서 바이너리 실행을 최적화하기 위해 SBT와 DBT 양자 모두를 사용할 수 있다. 예를 들어, 명령어 세트 아키텍처의 다수의 매크로 명령어를 단일 매크로 명령어로 융합하기 위해 바이너리 변환이 사용될 수 있다. 일 실시예에서, 처리 디바이스는 융합된 매크로 명령어에 대한 지원을 제공한다. 용어 "명령어"는 일반적으로 매크로 명령어(macroinstructions)를 지칭하는데, 이는 프로세서가 매크로 명령어로부터 디코딩하는 마이크로 명령어 또는 마이크로 연산(micro-operations)(예를 들어, 마이크로 op(micro-ops))과 대조적으로, 실행을 위해 프로세서에 제공되는 명령어이다. 마이크로 명령어 또는 마이크로 op는 매크로 명령어와 관련된 로직을 구현하기 위한 연산을 수행하도록 프로세서상의 실행 유닛에 지시하도록 구성될 수 있다.

[0005] 아래에서는 프로세서 코어 아키텍처가 설명되며 그 다음에 본 명세서에 기술된 실시예에 따른 예시적인 프로세서 및 컴퓨터 아키텍처에 대한 설명이 뒤따른다. 이하에서 설명되는 본 발명의 실시예에 대한 완전한 이해를 제공하기 위해 다수의 특정 세부 사항이 제시된다. 그러나, 실시예들은 이들 특정 세부 사항 중 일부가 없이 실시될 수 있다는 것이 관련 기술분야의 통상의 기술자에게 명백할 것이다. 다른 예들에서는, 다양한 실시예들의 기본 원리들을 모호하게 하는 것을 피하기 위해 공지된 구조들 및 디바이스들은 블록도 형태로 도시된다.

[0006] 프로세서 코어들은 상이한 방식들로, 상이한 목적들을 위해, 그리고 상이한 프로세서들에서 구현될 수 있다. 예를 들어, 그러한 코어의 구현은 다음을 포함할 수 있다: 1) 범용 컴퓨팅을 위한 범용 순차적 코어; 2) 범용 컴퓨팅을 위한 고성능 범용 비순차적 코어; 3) 주로 그래픽 및/또는 과학적 (스루풋) 컴퓨팅을 위한 특수 목적 코어. 프로세서는 단일 프로세서 코어를 사용하여 구현될 수 있거나 다중 프로세서 코어를 포함할 수 있다. 프로세서 내의 프로세서 코어는 아키텍처 명령어 세트 측면에서 동종 또는 이종일 수 있다.

[0007] 상이한 프로세서의 구현은 다음을 포함한다: 1) 범용 컴퓨팅을 위한 하나 이상의 범용 순차적 코어 및/또는 범용 컴퓨팅을 위한 하나 이상의 범용 비순차적 코어를 포함하는 중앙 프로세서; 및 2) 주로 그래픽 및/또는 과학을 위해 의도된 하나 이상의 특수 목적 코어(예를 들어, 많은 통합된 코어 프로세서)를 포함하는 코프로세서. 이러한 상이한 프로세서들은 다음에 언급한 것들을 포함하는 상이한 컴퓨터 시스템 아키텍처들을 초래한다: 1) 중앙 시스템 프로세서와 별도의 칩 상에 있는 코프로세서; 2) 중앙 시스템 프로세서와 별도의 다이에 있지만 동일한 패키지에 있는 코프로세서; 3) 다른 프로세서 코어와 동일한 다이상에 있는 코프로세서(이 경우, 이러한 코프로세서는 때때로 통합 그래픽 및/또는 과학적 (스루풋) 로직과 같은 특수 목적 로직, 또는 특수 목적 코어로 언급됨); 및 4) 기술된 프로세서(때로는 애플리케이션 코어(들) 또는 애플리케이션 프로세서(들)로 언급됨), 전술한 코프로세서, 및 추가의 기능을 동일한 다이상에 포함될 수 있는 시스템 온 칩.

[0008] **예시적인 코어 아키텍처들**

[0009] 순차적 및 비순차적 코어 블록도

[0010] 도 1a는 실시예에 따른, 예시적인 순차적 파이프라인 및 예시적인 레지스터 리네이밍 비순차적 발행/실행 파이프라인을 도시하는 블록도이다. 도 1b는 실시예에 따른 프로세서에 포함될 순차적 아키텍처 코어의 예시적인 실시예 및 예시적인 레지스터 리네이밍, 비순차적 발행/실행 아키텍처 코어의 양자 모두를 도시하는 블록도이다. 도 1a 및 도 1b에서의 실선 박스들은 순차적 파이프라인 및 순차적 코어를 도시하는 반면, 파선 박스들의 옵션의 부가는 레지스터 리네이밍, 비순차적 발행/실행 파이프라인 및 코어를 도시한다. 순차적 양태가 비순차적 양태의 서브세트라는 점을 고려하여, 비순차적 양태가 설명될 것이다.

[0011] 도 1a에서, 프로세서 파이프라인 (100)은 페치 스테이지(102), 길이 디코드 스테이지(104), 디코드 스테이지(106), 할당 스테이지(108), 리네이밍 스테이지(110), 스케줄링(디스패치 또는 발행으로도 알려져 있음) 스테이지(112), 레지스터 판독/메모리 판독 스테이지(114), 실행 스테이지(116), 라이트백(write back)/메모리 기입 스테이지(118), 예외 처리 스테이지(122) 및 커밋 스테이지(124)를 포함한다.

[0012] 도 1b는 실행 엔진 유닛(150)에 연결되는 프런트 엔드 유닛(130)을 포함하는 프로세서 코어(190)를 도시하며, 이들 두 개의 유닛 모두는 메모리 유닛(170)에 연결된다. 코어(190)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 옵션으로서, 코어(190)는, 예를 들어, 네트워크 또는 통신 코어, 압축 엔진, 코프로세서 코어, 범용 컴퓨팅 그래픽스 처리 유닛(GPGPU: general purpose computing graphics processing unit) 코어, 그래픽스 코어 등과 같은 특수 목적 코어일 수 있다.

[0013] 프런트 엔드 유닛(130)은 명령어 캐시 유닛(134)에 연결된 분기 예측 유닛(132)을 포함하고, 이 명령어 캐시 유닛은 명령어 변환 색인 버퍼(translation lookaside buffer, TLB)(136)에 연결되고, 이 명령어 변환 색인 버퍼는 명령어 페치 유닛(138)에 연결되고, 이 명령어 페치 유닛은 디코드 유닛(140)에 연결된다. 디코드 유닛(140)(또는 디코더)은 명령어들을 디코딩하고, 출력으로서 하나 이상의 마이크로 연산들, 마이크로코드 엔트리 포인트들, 마이크로 명령어들, 다른 명령어들, 또는 다른 제어 신호들을 생성할 수 있는데, 이들은 오리지널 명령어들로부터 디코딩되거나, 또는 다른 방식으로 오리지널 명령어들을 반영하거나 오리지널 명령어들로부터 도출된다. 디코드 유닛(140)은 다양한 상이한 메커니즘들을 이용하여 구현될 수 있다. 적절한 메커니즘의 예는 탐색표, 하드웨어 구현, 프로그램 가능 논리 어레이(PLA), 마이크로코드 판독 전용 메모리(ROM) 등을 포함하지만 이에 한정되지 않는다. 일 실시예에서, 코어(190)는(예를 들어, 디코드 유닛(140)에서 또는 다른 방식으로 프런트 엔드 유닛(130) 내에) 특정 매크로명령어들을 위한 마이크로코드를 저장하는 마이크로코드 ROM 또는 다

른 매체를 포함한다. 디코드 유닛(140)은 실행 엔진 유닛(150)에서의 리네임/할당자 유닛(152)에 연결된다.

[0014] 실행 엔진 유닛(150)은, 하나 이상의 스케줄러 유닛(들)(156)의 세트 및 리타이어먼트 유닛(154)에 연결된 리네임/할당자 유닛(152)을 포함한다. 스케줄러 유닛(들)(156)은, 예약 스테이션들, 중앙 명령어 윈도우 등을 비롯한 임의의 수의 상이한 스케줄러들을 나타낸다. 스케줄러 유닛(들)(156)은 물리적 레지스터 파일(들) 유닛(들)(158)에 연결된다. 물리적 레지스터 파일(들) 유닛들(158) 각각은 하나 이상의 물리적 레지스터 파일을 나타내고, 이들 중 상이한 물리적 레지스터 파일들은 스칼라 정수, 스칼라 부동 소수점, 패킹된 정수, 패킹된 부동 소수점, 벡터 정수, 벡터 부동 소수점, 상태(예를 들어, 실행될 다음 명령어의 어드레스인 명령어 포인터) 등과 같은 하나 이상의 상이한 데이터 유형을 저장한다. 일 실시예에서, 물리적 레지스터 파일(들) 유닛(158)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)(158)은,(예를 들어, 재순서화 버퍼(들) 및 리타이어먼트 레지스터 파일(들)을 이용하여; 미래 파일(들), 이력 버퍼(들) 및 리타이어먼트 레지스터 파일(들)을 이용하여; 레지스터 맵들 및 레지스터들의 풀을 이용하거나 하여) 레지스터 리네이밍 및 비순차적 실행이 구현될 수 있는 다양한 방식들을 예시하기 위해 리타이어먼트 유닛(154)에 의해 중첩된다. 리타이어먼트 유닛(154) 및 물리적 레지스터 파일(들) 유닛(들)(158)은 실행 클러스터(들)(160)에 연결된다. 실행 클러스터(들)(160)는 하나 이상의 실행 유닛들(162)의 세트 및 하나 이상의 메모리 액세스 유닛들(164)의 세트를 포함한다. 실행 유닛들(162)은 다양한 유형의 데이터(예를 들어, 스칼라 부동 소수점, 패킹된 정수, 패킹된 부동 소수점, 벡터 정수, 벡터 부동 소수점)에 대해 다양한 연산들(예를 들어, 시프트, 덧셈, 뺄셈, 곱셈)을 수행할 수 있다. 일부 실시예들은 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)(156), 물리적 레지스터 파일(들) 유닛(들)(158) 및 실행 클러스터(들)(160)는 가능하게는 복수 개인 것으로 도시되어 있는데, 그 이유는 특정 실시예들이 특정 유형의 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 자신의 스케줄러 유닛, 물리적 레지스터 파일(들) 유닛 및/또는 실행 클러스터를 각각 갖는 메모리 액세스 파이프라인 - 별개의 메모리 액세스 파이프라인의 경우에, 이 파이프라인의 실행 클러스터만이 메모리 액세스 유닛(들)(164)을 갖는 특정 실시예들이 구현됨)을 생성하기 때문이다. 별개의 파이프라인들이 사용되는 경우, 이들 파이프라인 중 하나 이상은 비순차적 발행/실행일 수 있고 나머지는 순차적일 수 있다는 점도 이해해야 한다.

[0015] 메모리 액세스 유닛들(164)의 세트는 메모리 유닛(170)에 연결되고, 이 메모리 유닛은 레벨 2(L2) 캐시 유닛(176)에 연결되는 데이터 캐시 유닛(174)에 연결된 데이터 TLB 유닛(172)을 포함한다. 하나의 예시적인 실시예에서, 메모리 액세스 유닛들(164)은 로드 유닛(load unit), 어드레스 저장 유닛(store address unit) 및 데이터 저장 유닛(store data unit)을 포함할 수 있으며, 이들 각각은 메모리 유닛(170)에서의 데이터 TLB 유닛(172)에 연결된다. 명령어 캐시 유닛(134)은 메모리 유닛(170)에서의 레벨 2(L2) 캐시 유닛(176)에 또한 연결된다. L2 캐시 유닛(176)은 하나 이상의 다른 레벨의 캐시에 그리고 궁극적으로는 메인 메모리에 연결된다.

[0016] 예로서, 예시적인 레지스터 리네이밍, 비순차적 발행/실행 코어 아키텍처는 다음과 같이 파이프라인 (100)을 구현할 수 있다: 1) 명령어 페치(138)는 페치 및 길이 디코딩 스테이지들(102 및 104)을 수행하고; 2) 디코드 유닛(140)은 디코드 스테이지(106)를 수행하고; 3) 리네임/할당자 유닛(152)은 할당 스테이지(108) 및 리네이밍 스테이지(110)를 수행하고; 4) 스케줄러 유닛(들)(156)은 스케줄 스테이지(112)를 수행하고; 5) 물리적 레지스터 파일(들) 유닛(들)(158) 및 메모리 유닛(170)은 레지스터 판독/메모리 판독 스테이지(114)를 수행하고; 실행 클러스터(160)는 실행 스테이지(116)를 수행하고; 6) 메모리 유닛(170) 및 물리적 레지스터 파일(들) 유닛(들)(158)은 라이트백/메모리 기입 스테이지(118)를 수행하고; 7) 다양한 유닛들이 예외 처리 스테이지(122)에 수반될 수 있고; 8) 리타이어먼트 유닛(154) 및 물리적 레지스터 파일(들) 유닛(들)(158)은 커밋 스테이지(124)를 수행한다.

[0017] 코어(190)는 본 명세서에 설명된 명령어(들)를 포함하여, 하나 이상의 명령어 세트들(예를 들어, (더 새로운 버전이 추가된 소정의 확장을 갖는) x86 명령어 세트; 캘리포니아주 서니베일에 있는 MIPS Technologies의 MIPS 명령어 세트; 캘리포니아주 서니베일에 있는 ARM Holdings의 (NEON과 같은 옵션의 부가 확장을 갖는) ARM® 명령어 세트)을 지원할 수 있다. 일 실시예에서, 코어(190)는 패킹된 데이터 명령어 세트 확장(예를 들어, AVX1, AVX2)을 지원하는 로직을 포함하며, 따라서 많은 멀티미디어 애플리케이션들에 의해 사용되는 연산들이 패킹된 데이터를 사용하여 수행되는 것을 허용한다.

[0018] 코어가 (연산들 또는 스레드들의 2개 이상의 병렬 세트를 실행하는) 멀티스레딩을 지원할 수 있고, 시간 슬라이싱된 멀티스레딩, 동시 멀티스레딩을 포함하는 다양한 방식으로(이 경우 단일 물리 코어는 물리 코어가 동시에 멀티스레딩하고 있는 각각의 스레드에게 논리 코어를 제공한다), 또는 이들의 조합(예를 들어, Intel® Hyperthreading technology에서와 같은 시간 슬라이싱된 페칭 및 디코딩 및 그 후의 동시 멀티스레딩)으로 지원할 수 있음을 이해해야 한다.

[0019] 레지스터 리네이밍이 비순차적 실행의 상황에서 설명되었지만, 레지스터 리네이밍은 순차적 아키텍처에서 사용될 수도 있다는 점을 이해해야 한다. 프로세서의 예시된 실시예가 별개의 명령어 및 데이터 캐시 유닛들(134/174) 및 공유 L2 캐시 유닛(176)을 또한 포함하지만, 대안적인 실시예들은, 예를 들어 레벨 1(L1) 내부 캐시 또는 다중 레벨의 내부 캐시와 같이, 명령어들 및 데이터 양자 모두에 대한 단일의 내부 캐시를 가질 수 있다. 일부 실시예들에서, 시스템은 내부 캐시와, 코어 및/또는 프로세서에 대해 외부에 있는 외부 캐시의 조합을 포함할 수 있다. 대안적으로, 모든 캐시는 코어 및/또는 프로세서에 대해 외부에 있을 수 있다.

[0020] 구체적인 예시적인 순차적 코어 아키텍처

[0021] 도 2a 및 도 2b는 더 구체적인 예시적인 순차적 코어 아키텍처의 블록도들이고, 이 코어는 칩에 있는 수개의 로직 블록들(동일한 유형 및/또는 상이한 유형들의 다른 코어들을 포함함) 중 하나일 것이다. 로직 블록들은 애플리케이션에 따라, 일부 고정된 기능 로직, 메모리 I/O 인터페이스들, 및 다른 필요한 I/O 로직을 갖는 고 대역폭 인터커넥트 네트워크(예를 들어, 링 네트워크)를 통해 통신한다.

[0022] 도 2a는 실시예에 따른, 레벨 2(L2) 캐시의 로컬 서브세트(204)를 갖는 단일 프로세서 코어를, 온-다이 인터커넥트 네트워크(202)로의 그의 접속과 함께 예시하는 블록도이다. 일 실시예에서, 명령어 디코더(200)는 패킹된 데이터 명령어 세트 확장을 갖는 x86 명령어 세트를 지원한다. L1 캐시(206)는 스칼라 유닛 및 벡터 유닛에 대한 캐시 메모리로의 낮은 레이턴시 액세스들을 허용한다. (설계를 간략화하기 위한) 일 실시예에서, 스칼라 유닛(208) 및 벡터 유닛(210)은 별개의 레지스터 세트(각기, 스칼라 레지스터들(212) 및 벡터 레지스터들(214))를 사용하고, 이들 사이에 전송되는 데이터는 메모리에 기입되고 이후 레벨 1(L1) 캐시(206)로부터 리드 백(read back)되는 반면, 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).

[0023] L2 캐시의 로컬 서브세트(204)는, 프로세서 코어 당 하나씩인 별개의 로컬 서브세트들로 분할되는 글로벌 L2 캐시의 일부이다. 각각의 프로세서 코어는 L2 캐시의 그 자신의 로컬 서브세트(204)에 대한 직접 액세스 경로를 갖는다. 프로세서 코어에 의해 판독된 데이터는 자신의 L2 캐시 서브세트(204)에 저장되며, 다른 프로세서 코어들이 그들 자신의 로컬 L2 캐시 서브세트들에 액세스하는 것과 병렬로 빠르게 액세스될 수 있다. 프로세서 코어에 의해 기입되는 데이터는 그 자신의 L2 캐시 서브세트(204)에 저장되고 또한 필요하다면 다른 서브세트들로부터 플러싱된다. 링 네트워크는 공유 데이터에 대한 일관성(coherency)를 보장한다. 링 네트워크는 양-방향성이어서, 프로세서 코어들, L2 캐시들 및 다른 논리 블록들과 같은 에이전트들이 칩 내에서 상호 통신하는 것을 허용한다. 각각의 링 데이터-경로는 방향당 1012 비트 폭이다.

[0024] 도 2b는 일 실시예에 따른 도 2a의 프로세서 코어의 일부분의 확대도이다. 도 2b는 L1 캐시(204)의 L1 데이터 캐시(206A) 부분뿐만 아니라, 벡터 유닛(210) 및 벡터 레지스터들(214)에 관한 보다 상세한 사항을 포함한다. 구체적으로, 벡터 유닛(210)은 16-폭 벡터-처리 유닛(VPU)(16-폭 산술 논리 유닛(arithmetic logic unit, ALU)(228) 참조)이고, 이는 정수, 단정밀도 부동 소수점 및 배정밀도 부동 소수점 명령어 중 하나 이상을 실행한다. VPU는 스위즐 유닛(swizzle unit)(220)을 이용하는 레지스터 입력들의 스위즐링, 수치 변환 유닛들(222A-B)을 이용하는 수치 변환, 및 메모리 입력에 대한 복제 유닛(224)을 이용하는 복제를 지원한다. 기입 마스크 레지스터들(226)은 결과적인 벡터 기입들을 서술하는 것(predicating)을 허용한다.

[0025] 통합 메모리 제어기 및 특수 목적 로직을 갖는 프로세서

[0026] 도 3은 일 실시예에 따라 2개 이상의 코어를 가질 수 있고, 통합 메모리 제어기를 가질 수 있고 통합 그래픽을 가질 수 있는 프로세서(300)의 블록도이다. 도 3의 실선으로 도시된 박스는 단일 코어(302A), 시스템 에이전트(310), 하나 이상의 버스 제어기 유닛(316)의 세트를 갖는 프로세서(300)를 도시하고, 점선으로 된 박스의 옵션의 부가는 다중 코어(302A-N), 시스템 에이전트 유닛(310) 내의 하나 이상의 통합 메모리 제어기 유닛(들)(314)의 세트, 및 특수 목적 로직(308)을 갖는 대안의 프로세서(300)를 도시한다.

[0027] 그러므로, 프로세서(300)의 상이한 구현들은 다음을 포함할 수 있다: 1) (하나 이상의 코어들을 포함할 수

있는) 통합 그래픽 및/또는 과학적 (스루풋) 로직인 특수 목적 로직(308), 및 하나 이상의 범용 코어들(예를 들어, 범용 순차적 코어들, 범용 비순차적 코어들, 이 둘의 조합)인 코어(302A-N)를 구비한 CPU; 2) 그래픽 및/또는 과학적 (스루풋)을 위해 주로 의도된 많은 수의 특수 목적 코어들인 코어들(302A-N)을 구비한 코프로세서; 및 3) 많은 수의 범용 순차적 코어들인 코어들(302A-N)을 구비한 코프로세서. 따라서, 프로세서(300)는 범용 프로세서와, 예를 들어 네트워크 또는 통신 프로세서, 압축 엔진, 그래픽 프로세서, GPGPU(general purpose graphics processing unit), 고스루풋 MIC(many integrated core) 코프로세서(30개 이상의 코어를 포함함), 임베디드 프로세서와 같은 코프로세서 또는 특수 목적 프로세서, 또는 그와 유사한 것일 수 있다. 프로세서는 하나 이상의 칩 상에 구현될 수 있다. 프로세서(300)는, 예를 들어, BiCMOS, CMOS, 또는 NMOS와같은 복수의 프로세스 기술 중 임의의 것을 이용하여 하나 이상의 기판 상에 구현될 수 있고/있거나 그 일부일 수 있다.

[0028] 메모리 계층구조는 코어들 내의 하나 이상의 레벨의 캐시, 하나 이상의 공유 캐시 유닛들(306)의 세트, 및 통합 메모리 제어기 유닛들(314)의 세트에 결합된 외부 메모리(도시되지 않음)를 포함한다. 공유 캐시 유닛들(306)의 세트는, 레벨 2(L2), 레벨 3(L3), 레벨 4(L4) 또는 다른 레벨 캐시와 같은 하나 이상의 중간 레벨 캐시, 최종 레벨 캐시(last level cache)(LLC) 및/또는 이들의 조합을 포함할 수 있다. 일 실시예에서는 링 기반 인터커넥트 유닛(312)이 통합 그래픽 로직(308), 공유 캐시 유닛들(306)의 세트 및 시스템 에이전트 유닛(310)/통합 메모리 제어기 유닛(들)(314)을 상호접속하지만, 대안 실시예들은 이러한 유닛들을 상호접속하는 임의 수의 공지된 기술들을 이용할 수 있다. 일 실시예에서, 하나 이상의 캐시 유닛들(306)과 코어들(302A-N) 사이의 일관성이 유지된다.

[0029] 일부 실시예들에서, 코어들(302A-N) 중 하나 이상은 멀티스레딩이 가능하다. 시스템 에이전트(310)는 코어(302A-N)를 조정 및 동작시키는 컴포넌트를 포함한다. 시스템 에이전트 유닛(310)은, 예를 들어 전력 제어 유닛(power control unit, PCU)) 및 디스플레이 유닛을 포함할 수 있다. PCU는, 코어들(302A-N) 및 통합 그래픽 로직(308)의 전력 상태를 조절하기 위해 필요한 로직 및 컴포넌트들일 수 있거나 이들을 포함할 수 있다. 디스플레이 유닛은 하나 이상의 외부 접속된 디스플레이들을 구동하기 위한 것이다.

[0030] 코어들(302A-N)은 아키텍처 명령어 세트에 관하여 동종일 수도 있고 이종일 수도 있는데; 즉, 코어들(302A-N) 중 2개 이상은 동일한 명령어 세트의 실행이 가능할 수 있는 한편, 다른 것들은 그 명령어 세트의 서브세트만을 또는 상이한 명령어 세트의 실행이 가능할 수 있다.

[0031] **예시적인 컴퓨터 아키텍처**

[0032] 도 4 내지 도 7은 예시적인 컴퓨터 아키텍처의 블록도이다. 랩톱들, 데스크톱들, 핸드헬드 PC들, 퍼스널 디지털 어시스턴트들, 엔지니어링 워크스테이션들, 서버들, 네트워크 디바이스들, 네트워크 허브들, 스위치들, 임베디드 프로세서들, DSP들(digital signal processors), 그래픽 디바이스들, 비디오 게임 디바이스들, 셋톱박스들, 마이크로 제어기들, 휴대 전화들, 휴대용 미디어 플레이어들, 핸드헬드 디바이스들, 및 다양한 다른 전자 디바이스들에 대해 본 기술분야에 알려진 다른 시스템 설계들 및 구성들 또한 적합하다. 일반적으로, 본 명세서에 개시되는 바와 같은 프로세서 및/또는 다른 실행 로직을 통합할 수 있는 매우 다양한 시스템들 또는 전자 디바이스들이 일반적으로 적합하다.

[0033] 도 4는 일 실시예에 따른 시스템(400)의 블록도를 도시한다. 시스템(400)은 제어기 허브(420)에 연결되는 하나 이상의 프로세서(410, 415)를 포함할 수 있다. 일 실시예에서, 제어기 허브(420)는 그래픽 메모리 제어기 허브(GMCH)(490) 및 입력/출력 허브(IOH)(450)를 포함한다(이들은 별개의 칩들에 있을 수도 있음); GMCH(490)는 메모리(440) 및 코프로세서(445)에 결합되는 메모리 및 그래픽 제어기를 포함한다; IOH(450)는 입력/출력(I/O) 디바이스(460)를 GMCH(490)에 연결한다. 대안적으로, 메모리 및 그래픽 제어기들 중 하나 또는 양자 모두는 프로세서 내에 통합되고(본 명세서에 설명된 바와 같이), 메모리(440) 및 코프로세서(445)는 IOH(450)와 단일 칩에 있는 제어기 허브(420) 및 프로세서(410)에 직접 연결된다.

[0034] 부가적인 프로세서들(415)의 옵션의 속성은 도 4에서 파선들로 표시되어 있다. 각각의 프로세서(410, 415)는 본 명세서에 설명된 하나 이상의 처리 코어를 포함할 수 있고 프로세서(300)의 일부 버전일 수 있다.

[0035] 메모리(440)는 예를 들어, DRAM(dynamic random access memory), PCM(phase change memory), 또는 이 둘의 조합일 수 있다. 적어도 하나의 실시예에서, 제어기 허브(420)는 프런트사이드 버스(FSB)와 같은 멀티 드롭 버스, QPI(QuickPath Interconnect)와 같은 포인트-투-포인트 인터페이스, 또는 유사한 접속(495)을 통해 프로세서(들)(410, 415)와 통신한다.

[0036] 일 실시예에서, 코프로세서(445)는, 예를 들어, 고스루풋 MIC 프로세서, 네트워크 또는 통신 프로세서, 압축 엔

진, 그래픽 프로세서, GPGPU, 임베디드 프로세서 등과 같은 특수 목적 프로세서이다. 일 실시예에서, 제어기 허브(420)는 통합 그래픽 가속기를 포함할 수 있다.

[0037] 아키텍처, 마이크로아키텍처, 열, 전력 소비 특성, 및 그와 유사한 것을 포함하여 이점에 대한 여러 기준들의 관점에서 물리적인 리소스들(410, 415) 간에 다양한 차이가 있을 수 있다.

[0038] 일 실시예에서, 프로세서(410)는 일반 유형의 데이터 처리 연산들을 제어하는 명령어들을 실행한다. 명령어들 내에는 코프로세서 명령어들이 임베딩될 수 있다. 프로세서(410)는 이러한 코프로세서 명령어들을 부속된 코프로세서(445)에 의해 실행되어야 하는 유형의 것으로 인식한다. 따라서, 프로세서(410)는 이러한 코프로세서 명령어들(또는 코프로세서 명령어들을 나타내는 제어 신호들)을 코프로세서 버스 또는 다른 인터커넥트 상에서 코프로세서(445)에 발행한다. 코프로세서(들)(445)는 수신된 코프로세서 명령어들을 수신하고 실행한다.

[0039] 도 5는 일 실시예에 따른 제1의 더 구체적인 예시적인 시스템(500)의 블록도를 도시한다. 도 5에 도시된 바와 같이, 멀티프로세서 시스템(500)은 포인트-투-포인트 인터커넥트 시스템이며, 포인트-투-포인트 인터커넥트(550)를 통해 결합되는 제1 프로세서(570) 및 제2 프로세서(580)를 포함한다. 프로세서(570 및 580) 각각은 프로세서(300)의 일부 버전일 수 있다. 본 발명의 일 실시예에서, 프로세서(570 및 580)는 각각 프로세서(410 및 415)이고, 코프로세서(538)는 코프로세서(445)이다. 또 다른 실시예에서, 프로세서(570 및 580)는 각각 프로세서(410) 코프로세서(445)이다.

[0040] 통합 메모리 제어기(IMC) 유닛(572 및 582)을 각각 포함하는 프로세서(570 및 580)가 도시되어 있다. 프로세서(570)는 또한 버스 제어기의 일부로서 포인트-투-포인트(P-P) 인터페이스(576 및 578)를 포함한다; 유사하게, 제2 프로세서(580)는 P-P 인터페이스(586 및 588)를 포함한다. 프로세서(570 및 580)는 P-P 인터페이스 회로(578 및 588)를 사용하여 포인트-투-포인트(P-P) 인터페이스(550)를 통해 정보를 교환할 수 있다. 도 5에 도시된 바와 같이, IMC들(572 및 582)은 프로세서들을, 각각의 프로세서에 로컬로 부속된 메인 메모리의 일부일 수도 있는, 각각의 메모리들, 즉, 메모리(532) 및 메모리(534)에 연결한다.

[0041] 프로세서(570, 580)는 각각 포인트 투 포인트 인터페이스 회로(576, 594, 586, 598)를 사용하여 별개의 P-P 인터페이스(552, 554)를 통해 칩셋(590)과 정보를 교환할 수 있다. 칩셋(590)은 옵션으로 고성능 인터페이스(539)를 통해 코프로세서(538)와 정보를 교환할 수 있다. 일 실시예에서, 코프로세서(538)는, 예를 들어, 고스루풋 MIC 프로세서, 네트워크 또는 통신 프로세서, 압축 엔진, 그래픽 프로세서, GPGPU, 임베디드 프로세서 등과 같은 특수 목적 프로세서이다.

[0042] 공유 캐시(도시되지 않음)는 어느 한 프로세서에 포함되거나, 양자 모두의 프로세서의 외부이지만 여전히 P-P 인터커넥트를 통해 프로세서들과 접속될 수 있어서, 프로세서가 저 전력 모드에 놓이는 경우 어느 한쪽 또는 양자 모두의 프로세서의 로컬 캐시 정보가 공유된 캐시에 저장될 수 있다.

[0043] 칩셋(590)은 인터페이스(596)를 통해 제1 버스(516)에 연결될 수 있다. 일 실시예에서, 제1 버스(516)는 PCI(Peripheral Component Interconnect) 버스, 또는 PCI 익스프레스 버스 또는 또 다른 3세대 I/O 인터커넥트 버스와 같은 버스일 수 있지만, 실시예들의 범위는 그에 한정되지는 않는다.

[0044] 도 5에 도시된 바와 같이, 다양한 I/O 디바이스(514)가 제1 버스(516)를 제2 버스(520)에 연결하는 버스 브리지(518)와 함께 제1 버스(516)에 연결될 수 있다. 일 실시예에서, 코프로세서, 고스루풋 MIC 프로세서, GPGPU, (예를 들어, 그래픽 가속기 또는 디지털 신호 처리(DSP) 유닛과 같은) 가속기, 필드 프로그램가능 게이트 어레이 또는 임의의 다른 프로세서와 같은 하나 이상의 부가적인 프로세서(들)(515)가 제1 버스(516)에 연결된다. 일 실시예에서, 제2 버스(520)는 LPC(low pin count) 버스일 수 있다. 일 실시예에서, 예를 들어, 키보드 및/또는 마우스(522), 통신 디바이스(527) 및 명령어/코드 및 데이터(530)를 포함할 수 있는 디스크 드라이브 또는 다른 대용량 저장 디바이스와 같은 스토리지 유닛(528)을 포함하는 다양한 디바이스가 제2 버스(520)에 연결될 수 있다. 또한, 오디오 I/O(524)가 제2 버스(520)에 연결될 수 있다. 다른 아키텍처들도 가능하다는 점에 유의한다. 예를 들어, 도 5의 포인트-투-포인트 아키텍처 대신에, 시스템은 멀티 드롭 버스 또는 다른 이러한 아키텍처를 구현할 수 있다.

[0045] 도 6은 일 실시예에 따른 제2의 더 구체적인 예시적인 시스템(600)의 블록도를 도시한다. 도 5 및 도 6에서의 유사한 요소들은 유사한 참조 번호들을 지니며, 도 6의 다른 양태들을 모호하게 하는 것을 피하기 위해 도 6으로부터 도 5의 특정 양태들이 생략되었다.

[0046] 도 6은 프로세서(570, 580)가 각각 통합 메모리 및 I/O 제어 로직("CL")(572 및 582)을 포함할 수 있음을 도시한다. 따라서, CL(572, 582)는 통합 메모리 제어기 유닛을 포함하고 I/O 제어 로직을 포함한다. 도 6은 메모

리들(532, 534)만이 CL(572, 582)에 연결되는 것이 아니라, I/O 디바이스들(614)도 또한 제어 로직(572, 582)에 연결되는 것을 도시한다. 레거시 I/O 디바이스들(615)이 칩셋(590)에 연결된다.

[0047] 도 7은 일 실시예에 따른 SoC(700)의 블록도를 도시한다. 도 3에서의 유사한 요소들은 유사한 참조 번호들을 지닌다. 또한, 파선 박스들은 더 진보된 SoC들에 대한 선택적인 특징들이다. 도 7에서, 인터커넥트 유닛(들)(702)은: 하나 이상의 코어(202A-N)의 세트 및 공유 캐시 유닛(들)(306)을 포함하는 애플리케이션 프로세서(710); 시스템 에이전트 유닛(310); 버스 제어기 유닛(들)(316); 통합 메모리 제어기 유닛(들)(314); 통합 그래픽 로직, 이미지 프로세서, 오디오 프로세서 및 비디오 프로세서를 포함할 수 있는 세트 또는 하나 이상의 코프로세서(720); 정적 랜덤 액세스 메모리(SRAM) 유닛(730); 직접 메모리 액세스(DMA) 유닛(732); 및 하나 이상의 외부 디스플레이에 연결하기 위한 디스플레이 유닛(740)에 연결된다. 일 실시예에서, 코프로세서(들)(720)는, 예를 들어, 네트워크 또는 통신 프로세서, 압축 엔진, GPGPU, 고스루풋 MIC 프로세서, 임베디드 프로세서 등과 같은 특수 목적 프로세서를 포함한다.

[0048] 본 명세서에 개시된 메커니즘들의 실시예들은 하드웨어, 소프트웨어, 펌웨어, 또는 이러한 구현 접근법들의 조합으로 구현될 수 있다. 실시예들은 적어도 하나의 프로세서, 스토리지 시스템(휘발성 및 비휘발성 메모리 및/또는 스토리지 요소들을 포함함), 적어도 하나의 입력 디바이스, 및 적어도 하나의 출력 디바이스를 포함하는 프로그램가능 시스템들 상에서 실행되는 컴퓨터 프로그램들 또는 프로그램 코드로서 구현될 수 있다.

[0049] 도 5에 도시된 코드(530)와 같은 프로그램 코드가 본 명세서에 설명된 기능을 수행하고 출력 정보를 생성하기 위해 입력 명령어에 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.

[0050] 프로그램 코드는 처리 시스템과 통신하기 위해 하이 레벨 절차형 또는 객체 지향형 프로그래밍 언어로 구현될 수 있다. 또한, 프로그램 코드는 요구되는 경우에 어셈블리 또는 머신 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.

[0051] 적어도 하나의 실시예의 하나 이상의 양태들은, 머신에 의해 판독될 때에 이 머신으로 하여금 본 명세서에 설명된 기술들을 수행하기 위한 로직을 제조하게 하는, 프로세서 내의 다양한 로직을 표현하는 머신 판독 가능 매체 상에 저장된 대표적인 데이터에 의해 구현될 수 있다. "IP 코어"라고 알려진 이러한 표현들은, 유형인 머신 판독 가능한 매체("테이프")에 저장될 수 있으며, 로직이나 프로세서를 실제로 만드는 제작 머신 내에 로딩하기 위해 다양한 고객이나 제조 설비에 공급될 수도 있다. 예를 들어, ARM Holdings, Ltd. 및 중국 과학원의 컴퓨팅 기술 연구소(ICT)에서 개발된 프로세서와 같은 IP 코어는 다양한 고객이나 라이선스 사용권자에게 라이센스가 부여되거나 판매될 수 있으며 이러한 고객 또는 라이선스 사용권자가 제작한 프로세서들에서 구현될 수 있다.

[0052] 이러한 머신 판독 가능 저장 매체는 하드 디스크와, 플로피 디스크, 광 디스크, CD-ROM(compact disk read-only memory), CD-RW(compact disk rewritable) 및 광자기 디스크를 포함하는 임의의 다른 유형의 디스크, DRAM(dynamic random access memory), SRAM(static random access memory), EPROM(erasable programmable read-only memory), 플래시 메모리, EEPROM(electrically erasable programmable read-only memory)과 같은 ROM(read-only memory), RAM(random access memory), PCM(phase change memory)을 포함하는 반도체 디바이스, 자기 또는 광 카드, 또는 전자 명령어들을 저장하는 데 적합한 임의의 다른 유형의 매체와 같은 저장 매체를 포함하는, 머신 또는 디바이스에 의해 제조 또는 형성되는 물품들의 비일시적이고 유형인 구성들을 포함할 수 있지만, 이들로 제한되지 않는다.

[0053] 따라서, 본 개시의 실시예들은, 또한, 명령어들을 포함하거나, 또는 본 명세서에 개시되는 구조들, 회로들, 장치들, 프로세서들 및/또는 시스템 특징들을 정의하는, HDL(Hardware Description Language) 등의 설계 데이터를 포함하는 비일시적이고 유형인 머신 판독 가능 매체를 포함한다. 이러한 실시예들은 프로그램 제품들로 또한 언급될 수 있다.

[0054] **에뮬레이션(바이너리 변환, 코드 모핑 등을 포함함)**

[0055] 본 명세서에 기술된 단일 명령어 세트 최적화 이외에, 명령어 변환은 소스 명령어 세트로부터의 명령어를 타깃 명령어 세트로 변환하는 데 사용될 수 있다. 예를 들어, 명령어 변환기는 명령어를 코어에 의해 처리될 하나 이상의 다른 명령어로(예를 들어, 정적 바이너리 변환, 동적 컴파일(dynamic compilation)을 포함하는 동적 바

이너리 변환을 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.

[0056] 도 8은 실시예에 따른 소스 명령어 세트에서의 바이너리 명령어들을 타깃 명령어 세트에서의 바이너리 명령어들로 변환하는 소프트웨어 명령어 변환기의 사용을 대조하는 블록도이다. 도시된 실시예에서, 명령어 변환기는 소프트웨어 명령어 변환기이지만, 대안적으로 명령어 변환기는 소프트웨어, 펌웨어, 하드웨어 또는 이들의 다양한 조합으로 구현될 수 있다. 도 8은 적어도 하나의 x86 명령어 세트 코어를 갖는 프로세서(816)에 의해 기본적으로 실행될 수 있는 x86 바이너리 코드(806)를 생성하기 위해 하이 레벨 언어(802)의 프로그램이 x86 컴파일러(804)를 사용하여 컴파일될 수 있는 것을 도시한다.

[0057] 적어도 하나의 x86 명령어 세트 코어를 갖는 프로세서(816)는, 적어도 하나의 x86 명령어 세트 코어를 갖는 인텔(Intel®) 프로세서와 실질적으로 동일한 결과를 달성하기 위해서, (1) 인텔 x86 명령어 세트 코어의 명령어 세트의 상당부 또는 (2) 적어도 하나의 x86 명령어 세트 코어를 갖는 인텔 프로세서 상에서 실행되도록 되어 있는 오브젝트 코드 버전의 애플리케이션들 또는 다른 소프트웨어를 호환가능하게 실행하거나 또는 다른 방식으로 처리함으로써, 적어도 하나의 x86 명령어 세트 코어를 갖는 인텔 프로세서와 실질적으로 동일한 기능을 수행할 수 있는 임의의 프로세서를 나타낸다. x86 컴파일러(804)는, 추가 연계 처리(linkage processing)를 수반하거나 수반하지 않고서 적어도 하나의 x86 명령어 세트 코어를 갖는 프로세서(816)상에서 실행될 수 있는 x86 바이너리 코드(806)(예를 들어, 오브젝트 코드)를 발생하도록 동작할 수 있는 컴파일러를 나타낸다. 유사하게, 도 8은 적어도 하나의 x86 명령어 세트 코어가 없는 프로세서(814)(예를 들어, 미국 캘리포니아주 서니베일 소재의 MIPS Technologies의 MIPS 명령어 세트를 실행하는 및/또는 미국 캘리포니아주 서니베일 소재의 ARM Holdings의 ARM 명령어 세트를 실행하는 코어들을 갖는 프로세서)에 의해 기본적으로 실행될 수 있는 대안 명령어 세트 바이너리 코드(810)를 생성하기 위해 하이 레벨 언어(802)의 프로그램이 대안 명령어 세트 컴파일러(808)를 사용하여 컴파일될 수 있다는 것을 나타낸 것이다.

[0058] 명령어 변환기(812)는 x86 바이너리 코드(806)를 x86 명령어 세트 코어를 구비하지 않은 프로세서(814)에 의해 기본적으로 실행될 수 있는 코드로 변환하는 데 사용된다. 이러한 변환된 코드는 대안적인 명령어 세트 바이너리 코드(810)와 동일할 가능성이 낮은데, 그 이유는 이것을 할 수 있는 명령어 변환기가 제조되기 어렵기 때문이다; 그러나, 변환된 코드는 일반 연산을 달성할 것이며, 대안적인 명령어 세트로부터의 명령어들로 이루어질 것이다. 따라서, 명령어 변환기(812)는 에뮬레이션, 시뮬레이션, 또는 임의의 다른 처리를 통해 x86 명령어 세트 프로세서 또는 코어를 갖지 않는 프로세서 또는 다른 전자 디바이스로 하여금 x86 바이너리 코드(806)를 실행하도록 허용하는 소프트웨어, 펌웨어, 하드웨어, 또는 이들의 조합을 나타낸다.

[0059] 동적 바이너리 변환 최적화 시스템

[0060] DBT 시스템은 융합 가능한 명령어 시퀀스를 발견하고 다수의 명령어를 단일 명령어로 융합함으로써 그 명령어 시퀀스를 최적화할 수 있는 동적 바이너리 변환 최적화 시스템으로서 구성될 수 있다. 도 9a 및 도 9b는 다수의 명령어를 융합된 명령어로 융합하는 것을 포함하는 런타임 바이너리 최적화를 수행하는 예시적인 바이너리 변환 시스템 및 로직을 도시한다. 도 9a는 일 실시예에 따른 동적 바이너리 변환을 위해 구성된 컴퓨팅 시스템의 블록도이다. 도 9b는 소스 코드 블록 내의 명령어들을 단일 융합된 명령어로 융합하는 로직의 흐름도이다.

[0061] 도 9a의 시스템(900)은 시스템 메모리(904)에 연결된 프로세서(902)를 포함한다. 일 실시예에서, 시스템은 프로세서(902)와 연결되거나 통합되는 캐시 메모리(905)(예를 들어, 도 1의 데이터 캐시 유닛(174) 또는 L2 캐시 유닛(176)) 및 스크래치패드 메모리(907)를 추가로 포함한다. 프로세서(902)는 물리적 레지스터들(906)의 세트 및 하나 이상의 코어 처리 유닛(예를 들어, "코어"(903A-N))을 포함한다. 일 실시예에서, 코어 처리 유닛들 각각은 다수의 동시 스레드들을 실행하도록 구성된다.

[0062] 시스템 메모리(904)는 소스 바이너리 애플리케이션(910), 동적 바이너리 변환 시스템(915) 및 호스트 운영 체제("OS")(920)를 호스팅할 수 있다. 동적 바이너리 변환 시스템(915)은 타깃 바이너리 코드 블록(들)(912), 레지스터 매핑 모듈(916)을 포함하는 동적 바이너리 변환기 코드(914) 및/또는 소스 레지스터 스토리지(918)를 포함한다. 소스 바이너리 애플리케이션(910)은 어셈블링된 로우 레벨 코드 또는 컴파일된 하이 레벨 코드일 수 있는 소스 바이너리 코드 블록들의 세트를 포함한다. 소스 바이너리 코드 블록은 증가, 비교 및 점프 명령어를 포함하는 분기 로직을 포함할 수 있는 명령어 시퀀스이다.

[0063] 일 실시예에서, 타깃 바이너리 코드 블록(들)(912)은 "코드 캐시"(911)로 지정된 시스템 메모리의 영역에 저장된다. 코드 캐시(911)는 소스 바이너리 코드 블록의 하나 이상의 대응하는 블록들로부터 변환된 타깃 바이너리 코드 블록(들)(912)에 대한 스토리지로서 사용된다. 시스템 메모리(904)는 프로세서 레지스터들(906)로/로부터 데이터를 로드/저장하도록 구성된 소스 레지스터 스토리지(918)를 호스팅할 수 있다. 일부 실시예에서, 캐시 메모리(905) 및/또는 스크래치-패드 메모리(907)는 프로세서 레지스터(들)(906)로/로부터 데이터를 로드/저장하도록 구성된다.

[0064] 일 실시예에서, 동적 바이너리 변환기 코드(914) 및 레지스터 매핑 모듈(916)은 소스 바이너리 애플리케이션(910)의 블록(들)을 타깃 바이너리 코드 블록(들)(912)로 변환하기 위해 소스 바이너리 애플리케이션(910)에 대해 연산하도록 하나 이상의 코어에 의해 실행된다. 타깃 바이너리 코드 블록(들)(912)은 소스 바이너리 애플리케이션(910)의 대응하는 소스 바이너리 코드 블록의 기능을 포함하도록 구성된다. 일 실시예에서, 소스 바이너리 애플리케이션의 소스 바이너리 코드 블록의 다수의 명령어들을 더 적은 수의 명령어들로 결합(예를 들어, 융합)하여, 더 적은 수의 명령어들에 대해 수행되는 소스 바이너리 애플리케이션과 동일한 기능을 포함하는 최적화된 타깃 바이너리 코드(912)를 생성할 수 있다. 예를 들어, 소스 바이너리 애플리케이션(910)은 카운터를 증가 또는 감소시키고, 카운터를 상수와 비교하고, 특정 제한이 충족되면(예를 들어, 루프 변수가 아직 N까지 증가하지 않았다면 - 여기서 N은 원하는 루프 반복 횟수) 점프를 호출하는 것을 포함하는 비교 및 점프 명령어 시퀀스를 포함할 수 있다. 일 실시예에서, DBT 시스템(915)은 3개의 별개의 증가, 비교 및 점프 명령어를 단일 명령어로 압축(예를 들어, 융합)하도록 구성된다.

[0065] 시스템(900)이 바이너리 코드 블록을 실행하기 위한 호출을 수신하면, DBT 시스템(915)은 융합 가능한 명령어에 대해 코드 블록을 스캔하고 명령어 시퀀스들을 융합된 명령어로 결합한다. 명령어들을 스캔하고 최적화하는 예시적인 로직이 도 9b에 도시되어 있다. DBT 시스템(915)이 도시되어 있지만, 일 실시예에서, 바이너리가 실행되기 전에 바이너리에 대해 SBT가 수행되고 발견되는 임의의 정적으로 융합 가능한 명령어 시퀀스(예를 들어, 정적 분석을 통해 안전하다고 결정된 명령어 시퀀스)를 융합하여 실행을 위한 최적화된 바이너리를 생성할 수 있다.

[0066] 도 9b의 920에 도시된 바와 같이, 시스템은 바이너리 코드 블록을 실행하기 위한 호출을 수신한다. 일 실시예에서, 시스템은 922에 도시된 바와 같이 증가, 비교 및 점프 명령어 시퀀스에 대해 스캔한다. 도 9b의 924에서 명령어 시퀀스가 검출되면, 926에서 변환 로직은 검출된 데이터 시퀀스 내에 임의의 데이터 종속성이 존재하는지를 결정하는 것을 포함하는 추가 연산을 수행할 수 있다. 그렇지 않다면, 시스템은 다음 코드 블록이 존재하면 932에서 다음 사용 가능한 코드 블록으로 진행한다. 예시적인 검출된 코드 시퀀스가 아래의 표 1에 제시된다.

[0067] 표 1: 예시적인 프로그램 코드

(1)	ADD EAX, 1
(2)	<0 or more instructions_fragment_A>
(3)	CMP EAX, constant_value_for_compare
(4)	<0 or more instructions_fragment_B>
(5)	JE jump_label

[0068]

[0069] 표 1의 예시적인 명령어들에서, 증가 명령어가 라인 (1)에 제시되고, 비교 명령어가 라인 (3)에 제시되고, 점프 명령어가 라인 (5)에 제시된다. 라인 (2)는 코드 fragment_A를 나타내며, 이는 라인 (1)에서의 증가와 라인 (3)에서의 비교 사이에 0개 이상의 명령어를 포함할 수 있다. 라인 (4)는 코드 fragment_B를 나타내며, 이는 라인 (3)에서의 비교와 라인 (5)에서의 점프 사이에 0개 이상의 명령어를 포함할 수 있다. JE(jump if equal; 같으면 점프) 명령어가 라인 (5)에 제시되어 있지만, 실시예는 임의의 특정 점프 명령어로 제한되지 않는다. 또한, CMP(비교) 명령어가 제시되어 있지만, 다른 비교 연산(예를 들어, TEST)이 또한 융합될 수 있다.

[0070] ADD, CMP, 및 JE 명령어 사이의 명령어 프래그먼트는 임의의 다른 명령어를 포함하지 않을 수 있다. 이 경우 ADD/CMP/JE 시퀀스는 연속적일 것이다. 그러나, 다른 명령어들이 프래그먼트 내의 코드 시퀀스에 존재할 수 있

다. 코드 시퀀스에서 임의의 추가 명령어를 재순서화하기 전에, 변환 로직은 코드 시퀀스를 스캔하여 926에서 임의의 데이터 종속성이 존재하는지를 결정한다. 만약 fragment_A 또는 fragment_B에서의 명령어의 피연산자 중 임의의 피연산자가 덧셈, 비교 또는 점프 명령어에 대한 피연산자에 종속한다면, 명령어들을 재순서화하는 것이 허용 가능하지 안을 수 있고, 변환 로직은 932에서 다음 사용 가능한 코드 블록으로 진행한다(그러한 코드 블록이 존재한다면). 또한, fragment_A 또는 fragment_B에 임의의 추가 분기 명령어가 존재한다면 명령어들을 재순서화하는 것이 허용 가능하지 않을 수 있다. 그러나, 일부 실시예에서는 점프 명령어 바로 다음에 추가 분기 명령어가 허용된다.

[0071] 그러나, fragment_A 또는 fragment_B의 명령어가 덧셈, 비교 또는 점프 명령어의 피연산자와 데이터 종속성을 갖지 않는다면, 들어오는 코드 스트림에 추가 명령어를 허용하는 것은 합법적이며 변환기는 임의의 데이터 종속성을 위반하지 않고 이러한 명령어들을 자유로이 재순서화해야 한다. 따라서, 번역 로직은, 블록 928에서, 검출된 명령어 시퀀스 내의 코드 프래그먼트 내의 임의의 명령어를 재순서화할 수 있다. 블록 930에서, 변환 로직은 별개의 증가, 비교, 점프 명령어들을, 비교 연산을 위한 레지스터 및 상수 값, 및 점프 연산을 위한 점프 라벨을 포함하여, 명령어 시퀀스를 수행하는 데 필요한 피연산자들을 포함하는 단일 increment_compare_jump 명령어로 대체한다. 예시적인 재순서화된 코드 시퀀스가 아래의 표 2에 제시된다.

[0072] 표 2: 예시적인 프로그램 코드

(6)	<0 or more instructions_fragment_A>
(7)	<0 or more instructions_fragment_B>
(8)	INC_CMP_JE EAX, constant_value_for_compare, jump_label

[0073]

[0074] 상기 표 2에 제시된 바와 같이, fragment_A 및 fragment_B에 대한 명령어는 라인 (6) 및 라인 (7)에 제시된 바와 같이 재순서화될 수 있다. 라인 8에 제시된 바와 같이, 융합된 increment_compare_jump 연산이 증가 연산, 비교 연산 및 점프 연산을 위한 피연산자를 포함하여 삽입된다.

[0075] 예시적인 융합된 명령어 프로세서 구현

[0076] 도 10a 및 도 10b는 increment_compare_jump 명령어의 예시적인 프로세서 구현을 나타내는 블록도이다. 몇몇 실시예에서, 프로세서를 구현하는 것은 명령어를 구현하기 위한 수개의 아키텍처 특징을 포함한다. 도 10a는 일 실시예에 따른 연산을 수행하기 위한 로직을 포함하는 프로세서 코어의 블록도이다. 도 10b는 일 실시예에 따른, increment_compare_jump 명령어를 구현하기 위한 예시적인 구체적인 마이크로 아키텍처의 블록도이다.

[0077] 도 10a에 도시된 바와 같이, 일 실시예에서 프로세서 코어(1000)는 실행될 명령어를 페치하기 위한 순차적 프런트 엔드(1001)를 포함하고 나중에 프로세서 파이프라인에서 사용될 명령어들을 준비한다. 일 실시예에서, 프런트 엔드(1001)는 도 1의 프런트 엔드 유닛(130)과 유사하며, 메모리로부터 명령어를 선제적으로 페치하기 위한 명령어 프리페처(1026)를 포함하는 컴포넌트를 추가로 포함한다. 페치된 명령어는 명령어를 디코딩 또는 해석하기 위해 명령어 디코더(1028)에 공급될 수 있다.

[0078] 일 실시예에서, 디코더(1028)는 수신된 명령어를, 머신이 실행할 수 있는 "마이크로-명령어" 또는 "마이크로 연산"(마이크로 op 또는 uop라고도 함)이라 불리는 하나 이상의 연산으로 디코딩한다. 다른 실시예들에서, 디코더는, 명령어를, 일 실시예에 따른 연산들을 수행하기 위해 마이크로 아키텍처에 의해 이용되는 opcode 및 대응하는 데이터 및 제어 필드들로 파싱한다. 일 실시예에서, 트레이스 캐시(1029)는 디코딩된 uop들을 취하여, 이들을 실행을 위해 uop 큐(1034)에서 프로그램 순서화된 시퀀스들 또는 트레이스들로 어셈블링한다.

[0079] 일 실시예에서, 프로세서 코어(1000)는 복합 명령어 세트를 구현한다. 트레이스 캐시(1029)가 복합 명령어를 만날 때, 마이크로코드 ROM(1032)은 연산을 완료하는 데 필요한 uop들을 제공한다. 일부 명령어들은 단일의 마이크로 op로 변환되는 한편, 다른 것들은 전체 연산(full operation)을 완료하는 데 수개의 마이크로 op를 필요로 한다. 일 실시예에서, 명령어는 명령어 디코더(1028)에서 처리하기 위한 소수의 마이크로 op들로 디코딩될 수 있다. 또 다른 실시예에서, 연산을 달성하는 데 다수의 마이크로 op가 필요한 경우, 명령어는 마이크로코드 ROM(1032) 내에 저장될 수 있다. 예를 들어, 일 실시예에서, 명령어를 완료하기 위해 4개보다 많은 마이크로 연산이 필요한 경우, 디코더(1028)는 마이크로코드 ROM(1032)에 액세스하여 명령어를 수행한다.

[0080] 트레이스 캐시(1029)는, 마이크로코드 ROM(1032)으로부터 일 실시예에 따른 하나 이상의 명령어를 완료하기 위

한 마이크로코드 시퀀스들을 판독하기 위해 올바른 마이크로 명령어 포인터를 결정하기 위해 엔트리 포인트 프로그램가능 로직 어레이(programmable logic array, PLA)를 참조한다. 마이크로코드 ROM(1032)이 명령어에 대한 마이크로 op들의 시퀀싱을 완료한 이후에, 머신의 프런트 엔드(1001)는 트레이스 캐시(1029)로부터 마이크로 op들을 페치하는 것을 재개한다. 일 실시예에서, 프로세서 코어(1000)는 명령어들이 실행을 위해 준비되는 비순차적 실행 엔진(1003)을 포함한다. 비순차적 실행 로직은 명령어가 명령어 파이프라인을 통해 진행될 때 성능을 최적화하기 위해 명령어 흐름을 재순서화하기 위한 다수의 버퍼를 가지고 있다. 마이크로코드 지원을 위해 구성된 실시예에서, 할당자 로직은 각각의 uop가 실행 중에 사용하는 머신 버퍼 및 리소스를 할당한다. 또한, 레지스터 리네이밍 로직은 레지스터 파일 내의 물리적 레지스터에서 논리적 레지스터를 물리적 레지스터로 리네이밍한다.

[0081] 일 실시예에서, 할당자는 하나는 메모리 연산을 위한 것이고 하나는 비-메모리 연산을 위한 것인, 2개의 uop 큐 중 하나의 uop 큐 내의 각각의 uop에 대한 엔트리를 다음과 같은 명령어 스케줄러들의 앞에 할당한다: 메모리 스케줄러, 고속 스케줄러(1002), 저속/일반 부동 소수점(1004), 및 단순 부동 소수점 스케줄러(1006). uop 스케줄러들(1002, 1004, 1006)은, uop들이 그들의 연산을 완료하는 데 필요로 하는 실행 리소스들의 이용가능성, 및 그들의 종속 입력 레지스터 피연산자 소스들(dependent input register operand sources)의 준비성(readiness)에 기초하여, uop가 실행될 준비가 된 때를 결정한다. 일 실시예의 고속 스케줄러(1002)는 메인 클록 사이클의 각각의 절반마다 스케줄링할 수 있는 한편, 다른 스케줄러들은 메인 프로세서 클록 사이클마다 한 번 스케줄링할 수 있다. 스케줄러들은 디스패치 포트들에 대하여 중재하여 실행을 위한 uop들을 스케줄링한다.

[0082] 레지스터 파일들(1008, 1010)은 스케줄러들(1002, 1004, 1006)과 실행 블록(1011) 내의 실행 유닛들(1012, 1014, 1016, 1018, 1020, 1022, 1024) 사이에 존재한다. 일 실시예에서 각각 정수 및 부동 소수점 연산을 위한, 별개의 레지스터 파일(1008, 1010)이 존재한다. 일 실시예에서, 각각의 레지스터 파일(1008, 1010)은 레지스터 파일에 아직 기록되지 않은 완료된 결과를 바이패스하거나 새로운 종속 uop들로 우회하거나 전달할 수 있는 바이패스 네트워크를 포함한다. 정수 레지스터 파일(1008) 및 부동 소수점 레지스터 파일(1010)은 또한 상대방과 데이터를 통신할 수 있다. 일 실시예에서, 정수 레지스터 파일(1008)은 2개의 별개의 레지스터 파일들, 즉 데이터의 하위 32 비트에 대한 하나의 레지스터 파일과 데이터의 상위 32 비트에 대한 제2 레지스터 파일로 분할된다. 일 실시예에서, 부동 소수점 레지스터 파일(1010)은 128 비트 폭 엔트리를 갖는다.

[0083] 실행 블록(1011)은 명령어들을 실행하기 위한 실행 유닛들(1012, 1014, 1016, 1018, 1020, 1022, 1024)을 포함한다. 레지스터 파일들(1008, 1010)은 마이크로 명령어들이 실행할 필요가 있는 정수 및 부동 소수점 데이터 피연산자 값들을 저장한다. 일 실시예의 프로세서 코어(1000)는 다음과 같은 다수의 실행 유닛으로 구성된다: 어드레스 생성 유닛(AGU)(1012), AGU(1014), 고속 ALU(1016), 고속 ALU(1018), 저속 ALU(1020), 부동 소수점 ALU(1022), 부동 소수점 이동 유닛(1024). 일 실시예에서, 부동 소수점 실행 블록들(1022, 1024)은 부동 소수점, MMX, SIMD 및 SSE, 또는 다른 연산들을 실행한다. 일 실시예의 부동 소수점 ALU(1022)는 나눗셈, 제곱근 및 나머지 마이크로 op들을 실행하는 64 비트 x 64 비트 부동 소수점 나눗셈기를 포함한다.

[0084] 일 실시예에서, 부동 소수점 값을 수반하는 명령어들은 부동 소수점 하드웨어로 처리될 수 있다. ALU 연산들은 고속 ALU 실행 유닛들(1016, 1018)로 간다. 일 실시예의 고속 ALU들(1016, 1018)은 1/2 클록 사이클의 유효 대기 시간으로 고속 연산을 실행할 수 있다. 일 실시예에서, 가장 복잡한 정수 연산들은 저속 ALU(1020)로 가는데, 그 이유는 저속 ALU(1020)가 곱셈기, 시프트, 플래그 로직 및 분기 처리와 같은 긴 대기 시간 유형의 연산들에 대한 정수 실행 하드웨어를 포함하기 때문이다. 메모리 로드/저장 연산은 AGU들(1012, 1014)에 의해 실행된다. 일 실시예에서, 정수 ALU들(1016, 1018, 1020)은 64 비트 데이터 피연산자들에 대해 정수 연산들을 수행하는 것과 관련하여 설명된다. 대안적인 실시예들에서, ALU들(1016, 1018, 1020)은, 16, 32, 128, 256 등을 비롯한 각종 데이터 비트를 지원하도록 구현될 수 있다. 유사하게, 부동 소수점 유닛들(1022, 1024)은 다양한 폭들의 비트들을 갖는 피연산자들의 범위를 지원하도록 구현될 수 있다. 일 실시예에서, 부동 소수점 유닛들(1022, 1024)은, SIMD 및 멀티미디어 명령어들과 관련하여 128-비트 폭의 패킹된 데이터 피연산자들에 대해 연산할 수 있다.

[0085] 일 실시예에서, uop들 스케줄러(1002, 1004, 1006)는 부모 로드가 실행을 완료하기 전에 종속 연산들을 디스패치한다. uop들이 추측적으로 스케줄되고 실행됨에 따라, 프로세서 코어(1000)는 또한 메모리 누락을 처리하는 로직을 포함한다. 데이터 로드가 데이터 캐시에서 누락되는 경우, 일시적으로 부정확한 데이터를 가지고 스케줄러를 떠난, 파이프라인에서 진행중인(in flight) 종속 연산들이 존재할 수 있다. 리플레이 메커니즘은 부정확한 데이터를 이용하는 명령어들을 추적하고 재실행한다. 일 실시예에서 종속 연산들만이 리플레이될 필요가

있고 비종속 연산들은 완료하도록 허용된다.

[0086] 일 실시예에서, 메모리 실행 유닛(MEI)(1041)이 포함된다. MEU(1041)는 메모리 순서 버퍼(memory order buffer, MOB)(1042), SRAM 유닛(1030), 데이터 TLB 유닛(1072), 데이터 캐시 유닛(1074) 및 L2 캐시 유닛(1076)을 포함한다.

[0087] 프로세서 코어(1000)는 다양한 컴포넌트들을 공유하거나 파티셔닝(partitioning)함으로써 동시 멀티스레딩 연산을 위해 구성될 수 있다. 프로세서에서 동작하는 임의의 스레드는 공유 컴포넌트에 액세스할 수 있다. 예를 들어, 공유 버퍼 또는 공유 캐시의 공간은 요청 스레드와 관계없이 스레드 연산에 할당될 수 있다. 일 실시예에서, 파티셔닝된 컴포넌트는 스레드마다 할당된다. 구체적으로 어느 컴포넌트가 공유되고 어느 컴포넌트가 파티셔닝되는지는 실시예에 따라 달라진다. 일 실시예에서, 실행 유닛들(예를 들어, 실행 블록(1011)) 및 데이터 캐시들(예를 들어, 데이터 TLB 유닛(1072), 데이터 캐시 유닛(1074))과 같은 프로세서 실행 리소스들은 공유 리소스이다. 일 실시예에서, L2 캐시 유닛(1076) 및 다른 상위 레벨 캐시 유닛(예를 들어, L3 캐시, L4 캐시)을 포함하는 다중 레벨 캐시는 모든 실행 스레드 간에 공유된다. 다른 프로세서 리소스는 스레드별로 파티셔닝되고 지정되거나 할당되고, 파티셔닝된 리소스의 특정 파티션들은 특정 스레드들에 전용된다. 예시적인 파티셔닝된 리소스는 MOB(1042), (예를 들어, 도 1b의 리네임/할당자 유닛(152) 및 리타이어먼트 유닛(154) 내의) 비순차적 엔진(1003)의 레지스터 에일리어스 테이블(RAT) 및 재순서화 버퍼(ROB), 및 프런트 엔드(1001)의 명령어 디코더(1028)와 관련된 하나 이상의 명령어 디코드 큐를 포함한다. 일 실시예에서, 명령어 TLB(예를 들어, 도 1b의 명령어 TLB 유닛(136)) 및 분기 예측 유닛(예를 들어, 도 1b의 분기 예측 유닛(132))도 파티셔닝된다.

[0088] 실행 블록(1011)의 예시적인 부분은 단일 사이클 increment_compare_jump 명령어를 구현하기 위한 마이크로 아키텍처(1050)를 나타내는 도 10b에 도시된 바와 같은 로직을 포함한다. 일 실시예에서, 도시된 마이크로 아키텍처(1050)는 프로세서 실행 파이프라인 내에서 실행 스테이지를 수행하도록 구성된다. 마이크로 아키텍처(1050)는 산술 논리 유닛(ALU)(1054) 및 점프 실행 유닛(JEU)(1056)을 포함하며, 분기 및 산술 명령어를 실행할 수 있다. 파이프라인 로직(1052A-B)은 마이크로 아키텍처를 이전 및 후속 파이프라인 스테이지를 위한 로직과 링크시켜, 피연산자(예를 들어, 피연산자_A(1060), 피연산자_B(1061))를 계산을 위해 ALU(1054)에 공급하여 ALU 계산의 결과(1063)(예를 들어, B+1)를 후속 파이프라인 스테이지에 전달한다. 일 실시예에서, 증가 연산의 결과는 입력 피연산자에 의해 지시된 적절한 레지스터에 커밋된다. 제어 유닛으로부터 ALU(1054)로의 제어 신호(1066)는 ALU 연산 중에서 선택하거나, 일 실시예에서 ALU에 opcode를 제공하는 데 사용된다. 제어 신호(1067)는 또한 JEU 연산을 제어하기 위해 제어 유닛으로부터 JEU에 제공된다.

[0089] 일 실시예에서, ALU(1054)는 비교 연산을 수행하는 데 사용된다. 뺄셈 연산은 수정 전 비교 명령어에 제공되는 피연산자_A(1060) 및 피연산자_B(1061)를 사용하여 수행될 수 있다. 뺄셈 연산(예를 들어, A-B)은 조건부 분기(예를 들어, 같으면 점프(jumb-equal), 같지 않으면 점프(jump-not-equal) 등)를 취할지를 결정하기 위해, JEU(1056)에 공급되는 플래그(예를 들면, 조건부 분기(1064)에 대한 ALU 플래그) 등등)를 생성하기 위해 수행된다.

[0090] 단일 실행 사이클 내에서 increment_compare_jump 명령어를 수행하기 위해, 각각의 컴포넌트는 해당 사이클 내의 적절한 시점에 적절한 입력이 필요로 한다. 예를 들어, ALU 플래그(1064)는 사이클의 초기에 JEU(1056)에 도착해야 하고, 다중 사이클 바이패스의 결과일 수 없다. 일 실시예에서, 타이밍 한계에 기초하여 조건부 점프를 위해 플래그의 특정 서브-세트(예를 들면, 캐리, 제로, 부호, 오버플로우 등)가 사용된다. 일 실시예에서, 아키텍처 플래그 레지스터의 모든 플래그는 패리티 플래그를 포함하는 점프 조건에 사용될 수 있다.

[0091] 일 실시예에서, increment_compare_jump 연산은 ALU(1054) 로의 캐리 입력(1062)을 이용함으로써 단일 사이클 내에서 수행된다. 예를 들어, 0번째 비트-슬라이스 덧셈기에 대한 캐리 입력(1062)이 어설트(assert)되어, 타이밍에 어떤 실질적인 영향도 주지 않으면서 ALU(1054)가 증가 및 비교(예를 들어, compare A-B +1)를 수행하게 할 수 있다. 이 계산은 필요한 경우 점프 계산을 수행하기 위해 적시에 점프 실행 유닛(1056)에 대한 ALU 플래그를 생성하기 위해 사이클의 초기에 수행될 수 있다. JEU(1056)는 ALU 플래그(1064) 플래그에 적어도 부분적으로 기초하여, 제어 흐름 변경을 개시하고 다음 명령어 포인터(NIP)를 업데이트하기 위해 프로세서 프런트 엔드에 제공되는 점프 타깃 어드레스를 포함하는 제어 리디렉션 정보(1065)를 생성한다.

[0092] 도 11은 일 실시예에 따른 increment_compare_jump 명령어를 수행하기 위한 로직을 포함하는 처리 시스템의 블록도이다. 예시적인 처리 시스템은 메인 메모리(1100)에 연결된 프로세서(1155)를 포함한다. 프로세서(1155)는 increment_compare_jump 명령어들을 디코딩하기 위한 디코드 로직(1131)을 갖는 디코드 유닛(1130)을 포함한다. 또한, 프로세서 실행 엔진 유닛(1140)은 명령어들을 실행하기 위한 추가 실행 로직(1141)을 포함한다. 레

지스터(1105)는 실행 유닛(1140)이 명령어 스트림을 실행함에 따라 피연산자, 제어 데이터 및 다른 유형의 데이터에 대한 레지스터 스토리지를 제공한다.

[0093] 단순화를 위해 도 11에는 단일 프로세서 코어("코어 0")의 세부 사항이 도시되어 있다. 그러나, 도 11에 도시된 각각의 코어는 코어 0과 동일한 로직 세트를 가질 수 있음을 이해할 것이다. 도시된 바와 같이, 각각의 코어는 지정된 캐시 관리 정책에 따라 명령어들 및 데이터를 캐싱하기 위한 전용 레벨 1(L1) 캐시(1112) 및 레벨 2(L2) 캐시(1111)를 포함할 수 있다. L1 캐시(1112)는 명령어들을 저장하기 위한 별개의 명령어 캐시(1120) 및 데이터를 저장하기 위한 별개의 데이터 캐시(1121)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(1100) 및/또는 공유 레벨 3(L3) 캐시(1116)로부터 명령어들을 페치하는 명령어 페치 유닛(1110); 명령어들을 디코딩하기 위한 디코드 유닛(1130); 명령어들을 실행하기 위한 실행 유닛(1140); 및 명령어들을 리타이어하고 그 결과들을 라이트 백(write back)하기 위한 라이트백/리타이어(writeback/retire) 유닛(1150)을 포함한다.

[0094] 명령어 페치 유닛(1110)은 메모리(1100)(또는 캐시들 중 하나)로부터 페치될 다음 명령어의 어드레스를 저장하기 위한 다음 명령어 포인터(1103); 어드레스 변환의 속도를 향상시키기 위해 최근에 사용된 가상-물리적 명령어 어드레스들의 맵을 저장하기 위한 명령어 변환 색인 버퍼(ITLB)(1104); 명령어 분기 어드레스를 추측적으로 예측하기 위한 분기 예측 유닛(1102); 분기 어드레스 및 타깃 어드레스를 저장하기 위한 분기 타깃 버퍼(branch target buffer, BTB)들(1101)을 포함하는 다양한 잘 알려진 컴포넌트들을 포함한다. 일단 패치되면, 명령어들은 그 후 디코드 유닛(1130), 실행 유닛(1140), 및 라이트백/리타이어 유닛(1150)을 포함하는 명령어 파이프라인의 나머지 스테이지들로 스트리밍된다.

[0095] 도 12는 일 실시예에 따른, increment_compare_jump 명령어를 처리하기 위한 로직에 대한 흐름도이다. 블록 1202에서, 명령어 파이프라인은 increment_compare_jump 연산을 수행하기 위한 명령어의 페치로 시작된다. 명령어는 명령어의 증가 및 비교 부분에 대한 제1 및 제2 입력 피연산자와 명령어의 조건부 점프 부분에 대한 점프 라벨 피연산자를 수락한다. 일 실시예에서, 제1 피연산자는 레지스터 또는 즉치 값일 수 있는 반면, 제2 피연산자는 레지스터, 즉치 값, 또는 메모리 어드레스일 수 있다. 일부 실시예에서 점프 라벨은 점프 타깃 어드레스로 변환되는 점프 명령어로부터의 즉치 값 오프셋이다.

[0096] 블록 1204에서, 디코드 유닛은 increment_compare_jump 명령어를 디코딩된 명령어로 디코딩한다. 일 실시예에서, 디코딩된 명령어는 단일 프로세서 사이클에서 실행되는 단일 연산이다. 일 실시예에서, 디코딩된 명령어는 명령어의 각각의 서브-엘리먼트를 수행하기 위한 하나 이상의 마이크로 연산을 포함한다. 마이크로 연산은 하드-와이어드(hard-wired)될 수 있거나 마이크로코드 연산은 실행 유닛과 같은 프로세서의 컴포넌트가 명령어를 구현하기 위한 다양한 연산을 수행하게 할 수 있다.

[0097] 블록 1206에서, 프로세서의 실행 유닛은 디코딩된 명령어를 실행하여, 증가하고, 비교하고, 비교에 기초하여 점프 타깃 라벨로 조건부로 점프(예를 들어, 분기)하기 위한 융합된 increment_compare_jump 연산을 수행한다. 일 실시예에서, ALU 비교(예를 들어, 뺄셈) 연산의 결과로 생기는 상태 플래그들 및, 관련 있다면, 임의의 다른 상태 플래그들에 기초하여, 점프 타깃 어드레스가 생성되어 프로세서 프런트 엔드에 전달된다.

[0098] 블록 1208에서, 프로세서 프런트 엔드는 연산들의 결과에 기초하여 다음 명령어 포인터를 업데이트하고, 프로세서의 리타이어먼트 유닛은 명령어를 리타이어한다. 일 실시예에서, 다음 명령어 포인터는 점프가 실행되는지의 여부에 기초하여 점프 타깃 어드레스로 또는 순차적으로 다음 명령어로 업데이트된다. 일 실시예에서, 비순차적 프로세서는 분기 예측 프로세서이고, 프로세서는 분기 예측을 해결(resolve)하기 위해 명령어의 결과를 사용한다. 분기 예측이 정확하다면 파이프라인의 명령어 흐름이 중단되지 않고 계속된다. 그러나, 분기 예측이 부정확하면, 프로세서는 예측 실패 복구 연산을 행하여 분기 예측 실패를 해결한다.

[0099] 일 실시예에서, 예측 실패가 검출될 때, JEU는 분기 예측 실패 이후에 페치된 명령어에 의해 생성된 상태를 프런트 엔드에서 클리어하고 새로운 명령어를 페치하기 시작할 어드레스를 프런트 엔드에 지시하는 신호(예를 들어, JE 클리어)를 어설트한다. 분기 예측 실패로부터 복구하는 데 소비된 프로세서 사이클은 예측 실패한 분기로부터 완전히 복구하는 데 필요한 사이클의 수인 프로세서 분기 예측 실패 페널티에 기여한다. 일 실시예에서, 명령어 융합은 별개의 명령어 시나리오와 비교하여 2개의 사이클에 의해 분기 예측 실패 패널티를 감소시킨다. 일 실시예에서, 별개의 증가, 비교 및 점프 명령어를 수반하는 분기 예측 실패로부터 복구하기 위해서는 3개의 프로세서 사이클이 필요하다.

[0100] 별개의 증가, 비교 및 점프 명령어 간의 비교가 아래 표에 제시되어 있다. 표 3은 별개의 증가, 비교 및 점프 명령어의 파이프라인 타이밍을 제시한다. 표 4는 융합된 단일 사이클 increment_compare_jump의 타이밍을 보여준다.

[0101] 표 3: 별개의 증가, 비교 및 점프 명령어 타이밍

스테이지 /시간	스케줄	RF Read	실행
N	INC		
N+1	CMP	INC	
N+2	JCC	CMP	INC
N+3		JCC	CMP
N+4			JCC: 분기 어드레스를 프런트엔드에 디스패치

[0102]

[0103] 상기 표 3에 제시된 바와 같이, 별개의 증가(INC), 비교(CMP) 및 점프(JCC) 명령어들은 스케줄링되고, 레지스터 파일 판독을 수행하고, 비순차적 프로세서(예를 들어, 비순차적 엔진(1003))에 의해 비순차적으로 실행된다. 명령어가 개별적으로 실행될 때 프로세서의 JEU는 분기 어드레스를 N+4까지 프런트 엔드에 디스패치할 수 없으므로, 프로세서가 분기를 잘못 예측하면 예측 페널티가 커진다.

[0104] 표 4: 별개의 증가, 비교 및 점프 명령어 타이밍

스테이지 /시간	스케줄	RF Read	실행
N	INC_CMP_JCC		
N+1		INC_CMP_JCC	
N+2			INC_CMP_JCC

[0105]

[0106] 상기 표 4에 제시된 바와 같이, 융합된 increment_compare_jump 명령어는 예약되고 레지스터 파일 판독을 수행하고, 별개의 명령어보다 2 사이클 일찍 실행된다. 또한 별개의 액션들을 수행하는 데 필요한 하드웨어 명령어의 수를 감소시킴으로써 다양한 기능 유닛에 대한 부담을 감소시켜, 해당 유닛들이 자유롭게 다른 연산들을 수행하게 할 수 있다. 일 실시예에서, 감소된 수의 명령어들이 프로세서 하드웨어 내에서 스케줄링되고 관리되기 때문에, 융합된 명령어는 스케줄링 및 부기(bookkeeping) 하드웨어에 대한 요구를 감소시킨다. 또한, 재순서화 버퍼 및 예약 스테이션에 필요한 리소스가 감소된다.

[0107] 일 실시예에서, 명령어 융합은 또한 별개의 명령어들의 레지스터들 사이에 명백한 종속성이 존재할 것이고, 단일 명령어가 사용될 때, 모든 레지스터 피연산자는 단일 명령어의 피연산자임을 고려할 때, 바이너리 변환 로직 내 및 프로세서 내 양자 모두에서 레지스터 할당 하드웨어에 대한 부담을 감소시킨다. 또한 융합된 명령어는 바이너리 변환 시스템에 대한 명령어 캐시 풋프린트를 감소시키고 명령어 페치 및 디코딩 대역폭 사용을 감소시킬 뿐만 아니라, 코드 밀도를 향상시킨다.

[0108] **예시적인 명령어 포맷**

[0109] 본 명세서에서 설명된 명령어(들)의 실시예들은 벡터 친화적 명령어 포맷을 포함하는 상이한 포맷으로 구현될 수 있다. 벡터 친화적 명령어 포맷은 벡터 명령어들에 적합한 명령어 포맷이다(예를 들어, 벡터 연산들에 특정적인 소정 필드들이 존재함). 벡터 및 스칼라 연산들 양자 모두가 벡터 친화적 명령어 포맷을 통해 지원되는 실시예들이 설명되지만, 대안적인 실시예들은 벡터 친화적 명령어 포맷의 벡터 연산들만을 사용한다.

[0110] 도 13a 및 도 13b는 실시예에 따른 일반적 벡터 친화적 명령어 포맷 및 그의 명령어 템플릿을 도시하는 블록도이다. 도 13a는 실시예에 따른 일반적 벡터 친화적 명령어 포맷 및 이것의 클래스 A 명령어 템플릿을 도시하는 블록도인 한편; 도 13b는 실시예에 따른 일반적 벡터 친화적 명령어 포맷 및 이것의 클래스 B 명령어 템플릿들을 도시하는 블록도이다. 구체적으로는, 일반 벡터 친화적 명령어 포맷(1300)에 대하여 클래스 A 및 클래스 B 명령어 템플릿들이 정의되고, 양자 모두는 메모리 액세스 없음(no memory access)(1305) 명령어 템플릿들 및 메모리 액세스(1320) 명령어 템플릿들을 포함한다. 벡터 친화적 명령어 포맷의 상황에서 일반적(generic)이라는 용어는 임의의 특정 명령어 세트에 얽매이지 않는 명령어 포맷을 지칭한다.

[0111] 벡터 친화적 명령어 포맷이 다음을 지원하는 실시예들이 설명될 것이다: 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). 그러나, 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.

[0112] 도 13a의 클래스 A 명령어 템플릿들은 다음을 포함한다: 1) 메모리 액세스 없음(1305) 명령어 템플릿들 내에 메모리 액세스 없음, 풀 라운드 제어형 연산(1310) 명령어 템플릿 및 메모리 액세스 없음, 데이터 변환형 연산(1315) 명령어 템플릿이 도시되어 있고; 2) 메모리 액세스(1320) 명령어 템플릿들 내에 메모리 액세스, 일시적(1325) 명령어 템플릿 및 메모리 액세스, 비일시적(1330) 명령어 템플릿이 도시되어 있다. 도 13b의 클래스 B 명령어 템플릿들은 다음을 포함한다: 1) 메모리 액세스 없음(1305) 명령어 템플릿들 내에 메모리 액세스 없음, 기입 마스크 제어, 부분 라운드 제어형 연산(1312) 명령어 템플릿 및 메모리 액세스 없음, 기입 마스크 제어, vsize 유형 연산(1317) 명령어 템플릿이 도시되어 있고; 2) 메모리 액세스(1320) 명령어 템플릿들 내에 메모리 액세스, 기입 마스크 제어(1327) 명령어 템플릿이 도시되어 있다.

[0113] 일반적 벡터 친화적 명령어 포맷(1300)은 도 13a 및 도 13b에 도시된 순서로 아래 나열된 다음의 필드들을 포함한다.

[0114] 포맷 필드(1340) - 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.

[0115] 베이스 연산 필드(1342) - 그의 내용은 상이한 베이스 연산들을 구별한다.

[0116] 레지스터 인덱스 필드(1344) - 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).

[0117] 변경자 필드(Modifier field)(1346) - 그의 내용은 메모리 액세스하지 않는 것들로부터 메모리 액세스를 지정하는 일반적 벡터 명령어 포맷 내의 명령어들의 발생들을 구별하는데, 즉, 메모리 액세스 없음(1305) 명령어 템플릿들과 메모리 액세스(1320) 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.

[0118] 증강(Augmentation) 연산 필드(1350) - 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(1368), 알파 필드(1352), 및 베타 필드(1354)로 분할된다. 증강 연산 필드(1350)는 연산들의 공통 그룹들이 2, 3, 또는 4개의 명령어보다는 단일 명령어에서 수행될 수 있게 한다.

[0119] 스케일 필드(1360) - 그의 내용은 메모리 어드레스 생성을 위한(예를 들어, 2^{scale} * index + base를 이용하는 어드레스 생성을 위한) 인덱스 필드의 내용의 스케일링(scaling)을 허용한다.

[0120] 변위 필드(1362A) - 그의 내용은 (예를 들어, 2^{scale} * index + base + displacement를 이용하는 어드레스 생성을 위한) 메모리 어드레스 생성의 부분으로서 이용된다.

[0121] 변위 인자 필드(Displacement Factor Field)(1362B)(변위 인자 필드(1362B) 바로 위의 변위 필드(1362A)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; 1374)(본 명세서에서 나중에 설명됨) 및 데이터 조작 필드(1354C)에 기초하여 실행시간에 프로세서 하드웨어에 의해 결정된다. 변위 필드(1362A) 및 변위 인자 필드(1362B)는 그것들이 메모리 액세스 없음(1305) 명령어 템플릿들을 위해 이용되지 않고 및/또는 상이한 실시예들은 둘 중 하나만 구현하거나 또는 아무것도 구현하지 않을 수 있다는 점에서 선택적이다.

[0122] 데이터 요소 폭 필드(1364) - 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 폭만이 지원되고/되거나 데이터 요소 폭들이 opcode들의 일부 양태를 이용하여 지원되는 경우에 필요하지 않는다는 점에서 선택적이다.

[0123] 기입 마스크 필드(1370) - 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(1370)는 로드, 저장, 산술, 로직 등을 포함한 부분 벡터 연산들을 허용한다. 기입 마스크 필드(1370)의 내용이 이용될 기입 마스크를 포함하는 다수의 기입 마스크 레지스터들 중 하나를 선택하는(및 따라서 기입 마스크 필드(1370)의 내용은 수행될 마스킹을 간접적으로 식별하는) 본 발명의 실시예들이 설명되지만, 대안적인 실시예들은 그 대신에 또는 추가적으로 마스크 기입 필드(1370)의 내용이 수행될 마스킹을 직접 특정할 수 있게 한다.

[0124] 즉치 필드(1372) - 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.

[0125] 클래스 필드(1368) - 그의 내용은 명령어들의 상이한 클래스들 간을 구별한다. 도 13a-b를 참조하면, 이 필드의 내용들은 클래스 A 및 클래스 B 명령어들 간을 선택한다. 도 13a-b에서, 라운딩된 코너 정사각형들(rounded corner squares)을 이용하여 특정 값이 필드(예를 들어, 도 13a-b에서 클래스 필드(1368)에 대해 각각 클래스 A(1368A) 및 클래스 B(1368B))에 존재함을 나타낸다.

[0126] 클래스 A의 명령어 템플릿

[0127] 클래스 A의 메모리 액세스 없음(1305) 명령어 템플릿들의 경우, 알파 필드(1352)는 RS 필드(1352A)로서 해석되고, 그 내용은 상이한 증강 연산 유형들 중 어느 것이 수행되어야 하는지를 구별하고(예를 들어, 라운드(1352A.1) 및 데이터 변환(1352A.2)은 각각 메모리 액세스 없음, 라운드 유형 연산(1310) 및 메모리 액세스 없음, 데이터 변환형 연산(1315) 명령어 템플릿들에 대해 지정되고), 베타 필드(1354)는 지정된 유형의 연산들 중 어느 것이 수행되어야 하는지를 구별한다. 메모리 액세스 없음(1305) 명령어 템플릿들에서, 스케일 필드(1360), 변위 필드(1362A), 및 변위 스케일 필드(1362B)는 존재하지 않는다.

[0128] 메모리 액세스 없음 명령어 템플릿 - 풀 라운드 제어형 연산

[0129] 메모리 액세스 없음 풀 라운드 제어형 연산(1310) 명령어 템플릿에서, 베타 필드(1354)는 라운드 제어 필드(1354A)로서 해석되고, 그 내용(들)은 정적 라운딩을 제공한다. 본 발명의 설명된 실시예들에서, 라운드 제어 필드(1354A)는 SAE(suppress all floating point exceptions) 필드(1356) 및 라운드 연산 제어 필드(1358)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(1358)를 가질 수 있다)을 지원할 수 있다.

[0130] SAE 필드(1356) - 그의 내용은 예외 이벤트 보고를 디스에이블할 것인지 여부를 구별하고; SAE 필드(1356)의 내용이 억제가 인에이블됨을 나타낼 때, 주어진 명령어는 임의의 종류의 부동 소수점 예외 플래그를 보고하지 않고, 임의의 부동 소수점 예외 핸들러를 발생시키지 않는다.

[0131] 라운드 연산 제어 필드(1358) - 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(1358)는 명령어별로 라운딩 모드의 변경을 허용한다. 프로세서가 라운딩 모드들을 지정하기 위한 제어 레지스터를 포함하는 본 발명의 일 실시예에서, 라운드 연산 제어 필드(1350)의 내용은 그 레지스터 값을 무효로 한다.

[0132] 메모리 액세스 없음 명령어 템플릿 - 데이터 변환형 연산

[0133] 메모리 액세스 없음 데이터 변환형 연산(1315) 명령어 템플릿에서, 베타 필드(1354)는 데이터 변환 필드(1354B)로서 해석되고, 그 내용은 다수의 데이터 변환(예를 들어, 데이터 변환 없음, 스위즐(swizzle), 브로드캐스트) 중 어느 것이 수행되어야 하는지를 구별한다.

[0134] 클래스 A의 메모리 액세스(1320) 명령어 템플릿의 경우에서, 알파 필드(1352)는 축출 힌트 필드(1352B)로서 해석되고, 그 내용은 이용될 축출 힌트들 중 하나를 구별하지만(도 13a에서, 일시적(1352B.1) 및 비일시적(1352B.2)이 각각 메모리 액세스, 일시적(1325) 명령어 템플릿 및 메모리 액세스, 비일시적(1330) 명령어 템플릿에 대해 특정된다), 베타 필드(1354)는 데이터 조작 필드(1354C)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(1320) 명령어 템플릿들은 스케일 필드(1360), 및 선택적으로 변위 필드(1362A) 또는 변위 스케일 필드(1362B)를 포함한다.

[0135] 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.

[0136] 메모리 액세스 명령어 템플릿 - 일시적

[0137] 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.

[0138] 메모리 액세스 명령어 템플릿 - 비일시적

[0139] 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.

[0140] 클래스 B의 명령어 템플릿

[0141] 클래스 B의 명령어 템플릿의 경우에, 알파 필드(1352)는 기입 마스크 제어(Z) 필드(1352C)로서 해석되고, 그 내용은 기입 마스크 필드(1370)에 의해 제어된 기입 마스킹이 병합 또는 제로화이어야 하는지를 구별한다.

[0142] 클래스 B의 메모리 액세스 없음(1305) 명령어 템플릿들의 경우에, 베타 필드(1354)의 부분은 RL 필드(1357A)로서 해석되고, 그 내용은 수행될 상이한 증강 연산 유형들 중 하나를 구별하지만(예를 들어, 라운드(1357A.1) 및 벡터 길이(VSIZE)(1357A.2)는 각각 메모리 액세스 없음, 기입 마스크 제어, 부분 라운드 제어형 연산(1312) 명령어 템플릿 및 메모리 액세스 없음, 기입 마스크 제어, VSIZE 유형 연산(1317) 명령어 템플릿에 대해 특정된다), 베타 필드(1354)의 나머지는 수행될 특정된 유형의 연산들 중 어느 하나를 구별한다. 메모리 액세스 없음(1305) 명령어 템플릿들에서, 스케일 필드(1360), 변위 필드(1362A), 및 변위 스케일 필드(1362B)는 존

재하지 않는다.

[0143] 메모리 액세스 없음, 기입 마스크 제어, 부분 라운드 제어형 연산(1310) 명령어 템플릿에서, 베타 필드(1354)의 나머지는 라운드 연산 필드(1359A)로서 해석되고, 예외 이벤트 보고는 디스에이블된다(주어진 명령어는 임의의 종류의 부동 소수점 예외 플래그를 보고하지 않고, 임의의 부동 소수점 예외 핸들러를 발생시키지 않는다).

[0144] 라운드 연산 제어 필드(1359A) - 단지 라운드 연산 제어 필드(1358)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(1359A)는 명령어별로 라운딩 모드의 변경을 허용한다. 프로세서가 라운딩 모드들을 지정하기 위한 제어 레지스터를 포함하는 본 발명의 일 실시예에서, 라운드 연산 제어 필드(1350)의 내용은 그 레지스터 값을 무효로 한다.

[0145] 메모리 액세스 없음, 기입 마스크 제어, VSIZE 유형 연산(1317) 명령어 템플릿에서, 베타 필드(1354)의 나머지는 벡터 길이 필드(1359B)로서 해석되고, 그 내용은 수행될 다수의 데이터 벡터 길이들(예를 들어, 128, 256, 또는 512 바이트) 중 하나를 구별한다.

[0146] 클래스 B의 메모리 액세스(1320) 명령어 템플릿의 경우에, 베타 필드(1354)의 부분은 브로드캐스트 필드(1357B)로서 해석되고, 그 내용은 브로드캐스트 유형 데이터 조작 연산이 수행될 것인지 여부를 구별하지만, 베타 필드(1354)의 나머지는 벡터 길이 필드(1359B)로서 해석된다. 메모리 액세스(1320) 명령어 템플릿들은 스케일 필드(1360), 및 선택적으로 변위 필드(1362A) 또는 변위 스케일 필드(1362B)를 포함한다.

[0147] 일반적 벡터 친화적 명령어 포맷(1300)과 관련하여, 포맷 필드(1340), 베이스 연산 필드(1342), 및 데이터 요소 폭 필드(1364)를 포함하는 풀 opcode 필드(1374)가 도시된다. 풀 opcode 필드(1374)가 이들 필드들 전부를 포함하는 일 실시예가 도시되지만, 풀 opcode 필드(1374)는 그것들 전부를 지원하지 않는 실시예들에 있어서 이들 필드들 전부보다 적게 포함한다. 풀 opcode 필드(1374)는 연산 코드(operation code)(opcode)를 제공한다.

[0148] 증강 연산 필드(1350), 데이터 요소 폭 필드(1364), 및 기입 마스크 필드(1370)는 이러한 특징들이 일반적 벡터 친화적 명령어 포맷에서 명령어별로 특정될 수 있게 한다.

[0149] 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.

[0150] 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 (스루풋) 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).

[0151] 예시적인 특정적 벡터 친화적 명령어 포맷

[0152] 도 14는 본 발명의 실시예에 따른 예시적인 특정적 벡터 친화적 명령어 포맷을 나타내는 블록도이다. 도 14는 필드들의 위치, 크기, 해석 및 순서뿐만 아니라, 이들 필드의 일부에 대한 값들을 지정한다는 점에서 특정적인 특정적 벡터 친화적 명령어 포맷(1400)을 도시한다. 특정적 벡터 친화적 명령어 포맷(1400)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 14로부터의 필드들이 매핑하는 도 13으로부터의 필드들이 예시된다.

[0153] 본 발명의 실시예들은 예시의 목적으로 일반적 벡터 친화적 명령어 포맷(1300)의 문맥에서 특정적 벡터 친화적 명령어 포맷(1400)을 참조하여 설명되지만, 본 발명은 청구되는 경우를 제외하고 특정적 벡터 친화적 명령어 포맷(1400)으로 한정되지 않는다는 것을 이해해야 한다. 예를 들어, 일반적 벡터 친화적 명령어 포맷(1300)은 다양한 필드에 대한 다양한 가능한 크기들을 고려하지만, 특정적 벡터 친화적 명령어 포맷(1400)은 특정 크기들의 필드들을 갖는 것으로서 도시된다. 특정 예에 의해, 데이터 요소 폭 필드(1364)는 특정적 벡터 친화적 명령어 포맷(1400)에서 1 비트 필드로서 도시되지만, 본 발명은 그것으로 한정되지 않는다(즉, 일반적 벡터 친화적 명령어 포맷(1300)은 데이터 요소 폭 필드(1364)의 다른 크기들을 고려한다).

[0154] 일반적 벡터 친화적 명령어 포맷(1300)은 도 14a에 도시된 순서로 아래에 나열된 다음의 필드들을 포함한다.

[0155] EVEX 프리픽스(바이트 0-3)(1402) - 4 바이트 형태로 인코딩된다.

[0156] 포맷 필드(1340)(EVEX 바이트 0, 비트들 [7:0]) - 제1 바이트(EVEX 바이트 0)는 포맷 필드(1340)이고, 그것은 0x62(본 발명의 일 실시예에서 벡터 친화적 명령어 포맷을 구별하는 데 이용되는 고유 값)을 포함한다.

[0157] 제2 내지 제4 바이트(EVEX 바이트 1-3)는 특정 능력을 제공하는 다수의 비트 필드를 포함한다.

[0158] REX 필드(1405)(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.

[0159] REX' 필드(1310) - 이것은 REX' 필드(1310)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.

[0160] opcode 맵 필드(1415)(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.

[0161] 데이터 요소 폭 필드(1364)(EVEX 바이트 2, 비트 [7]-W) - 표기법 EVEX.W에 의해 표현된다. EVEX.W는 데이터 형(32 비트 데이터 요소 또는 64 비트 데이터 요소)의 입도(크기)를 정의하는 데 사용된다.

[0162] EVEX.vvvv(1420)(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(1420)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.

[0163] EVEX.U 클래스 필드(1368)(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.

[0164] 프리픽스 인코딩 필드(1425)(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.

[0165] 알파 필드(1352)(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.

[0166] 베타 필드(1354)(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.

[0167] REX' 필드(1310) - 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.

[0168] 기입 마스크 필드(1370)(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).

[0169] 실제 opcode 필드(1430)(바이트 4)는 또한 opcode 바이트로 알려진다. opcode의 일부는 이 필드에서 특정된다.

[0170] MOD R/M 필드(1440)(바이트 5)는 MOD 필드(1442), Reg 필드(1444), 및 R/M 필드(1446)를 포함한다. 전술한 바와 같이, MOD 필드(1442)의 내용은 메모리 액세스와 메모리 액세스 없음 연산들 사이를 구별한다. Reg 필드(1444)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(1446)의 역할은 메모리 어드레스를 참조하는 명령어 피연산자를 인코딩하거나, 목적지 레지스터 피연산자 또는 소스 레지스터 피연산자를 인코딩하는 것을 포함할 수 있다.

[0171] SIB(Scale, Index, Base) 바이트(바이트 6) - 전술한 바와 같이, 스케일 필드(1350)의 내용은 메모리 어드레스 생성을 위해 이용된다. SIB.xxx(1454) 및 SIB.bbb(1456) - 이 필드들의 내용들은 레지스터 인덱스들 Xxxx 및 Bbbb과 관련하여 앞서 언급하였다.

[0172] 변위 필드(1362A)(바이트들 7-10) - MOD 필드(1442)가 10을 포함할 때, 바이트들 7-10은 변위 필드(1362A)이고, 그것은 레거시 32-비트 변위(disp32)와 동일하게 작용하고, 바이트 입도에서 작용한다.

[0173] 변위 인자 필드(1362B)(바이트 7) - MOD 필드(1442)가 01을 포함할 때, 바이트 7은 변위 인자 필드(1362B)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(1362B)는 disp8의 재해석이고; 변위 인자 필드(1362B)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(1362B)는 레거시 x86 명령어 세트 8-비트 변위를 대체한다. 따라서, 변위 인자 필드(1362B)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.

[0174] 즉치 필드(1372)는 전술한 바와 같이 동작한다.

[0175] 풀 opcode 필드

[0176] 도 14b는 본 발명의 일 실시예에 따른 풀 opcode 필드(1374)를 구성하는 특정적 벡터 친화적 명령어 포맷(1400)의 필드들을 도시하는 블록도이다. 구체적으로, 풀 opcode 필드(1374)는 포맷 필드(1340), 베이스 연산 필드(1342), 및 데이터 요소 폭(W) 필드(1364)를 포함한다. 베이스 연산 필드(1342)는 프리픽스 인코딩 필드(1425), opcode 맵 필드(1415), 및 실제 opcode 필드(1430)를 포함한다.

[0177] 레지스터 인덱스 필드

[0178] 도 14c는 본 발명의 일 실시예에 따른 레지스터 인덱스 필드(1344)를 구성하는 특정적 벡터 친화적 명령어 포맷(1400)의 필드들을 도시하는 블록도이다. 구체적으로, 레지스터 인덱스 필드(1344)는 REX 필드(1405), REX' 필드(1410), MODR/M.reg 필드(1444), MODR/M.r/m 필드(1446), VVVV 필드(1420), xxx 필드(1454), 및 bbb 필드(1456)를 포함한다.

[0179] 증강 연산 필드

[0180] 도 14d는 본 발명의 일 실시예에 따른 증강 연산 필드(1350)를 구성하는 특정적 벡터 친화적 명령어 포맷(1400)의 필드들을 도시하는 블록도이다. 클래스(U) 필드(1368)가 0을 포함할 때, 그것은 EVEX.U0(클래스 A(1368A))을 의미하고; 그것이 1을 포함할 때, 그것은 EVEX.U1(클래스 B(1368B))를 의미한다. U=0이고 MOD 필드(1442)가 11을 포함할 때(메모리 액세스 없음 연산을 의미함), 알파 필드(1352)(EVEX 바이트 3, 비트 [7]-EH)는 rs 필드(1352A)로서 해석된다. rs 필드(1352A)가 1을 포함할 때(라운드 1352A.1), 베타 필드(1354)(EVEX 바이트 3, 비트들 [6:4]- SSS)는 라운드 제어 필드(1354A)로서 해석된다. 라운드 제어 필드(1354A)는 1 비트 SAE 필드(1356) 및 2 비트 라운드 연산 필드(1358)를 포함한다. rs 필드(1352A)가 0을 포함할 때(데이터 변환 1352A.2), 베타 필드(1354)(EVEX 바이트 3, 비트들 [6:4]- SSS)는 3 비트 데이터 변환 필드(1354B)로서 해석된다. U=0이고 MOD 필드(1442)가 00, 01, 또는 10을 포함할 때(메모리 액세스 연산을 의미함), 알파 필드(1352)(EVEX 바이트 3, 비트 [7]-EH)는 축출 힌트(EH) 필드(1352B)로서 해석되고, 베타 필드(1354)(EVEX 바이트 3, 비트들 [6:4]- SSS)는 3 비트 데이터 조작 필드(1354C)로서 해석된다.

[0181] U=1일 때, 알파 필드(1352)(EVEX 바이트 3, 비트 [7]-EH)는 기입 마스크 제어(Z) 필드(1352C)로서 해석된다. U=1이고 MOD 필드(1442)가 11을 포함할 때(메모리 액세스 없음 연산을 의미함), 베타 필드(1354)의 부분(EVEX 바이트 3, 비트 [4]- S_0)은 RL 필드(1357A)로서 해석되고; 그것이 1을 포함할 때(라운드 1357A.1), 베타 필드(1354)의 나머지(EVEX 바이트 3, 비트 [6-5]-S_{2-1})는 라운드 연산 필드(1359A)로서 해석되고, RL 필드(1357A)가 0을 포함할 때(VSIZE 1357.A2), 베타 필드(1354)의 나머지(EVEX 바이트 3, 비트 [6-5]-S_{2-1})는 벡터 길이 필드(1359B)(EVEX 바이트 3, 비트 [6-5]-L_{1-0})로서 해석된다. U=1이고 MOD 필드(1442)가 00, 01, 또는 10을 포함할 때(메모리 액세스 연산을 의미함), 베타 필드(1354)(EVEX 바이트 3, 비트들 [6:4]-SSS)는 벡터 길이 필드(1359B)(EVEX 바이트 3, 비트 [6-5]-L_{1-0}) 및 브로드캐스트 필드(1357B)(EVEX 바이트 3, 비트 [4]-B)로서 해석된다.

[0182] 예시적인 레지스터 아키텍처

[0183] 도 15은 본 발명의 일 실시예에 따른 레지스터 아키텍처(1500)의 블록도이다. 도시된 실시예에서, 폭이 512 비트인 32개의 벡터 레지스터들(1510)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(1400)은 아래 표 5에 예시된 바와 같이 이들 오버레이된 레지스터 파일에 대해 동작한다.

[0184] 표 5 - 레지스터 파일

조정 가능 벡터 길이	클래스	연산	레지스터
벡터 길이 필드(1359B)를 포함하지 않는 명령어 템플릿	A (도 13A; U=0)	1310, 1315, 1325, 1330	zmm 레지스터 (벡터 길이는 64 바이트)
	B (도 13B; U=1)	1312	zmm 레지스터 (벡터 길이 는 64 바이트)
벡터 길이 필드(1359B)를 포함하는 명령어 템플릿	B (도 13B; U=1)	1317, 1327	벡터 길이 필드(1359B)에 따라 zmm, ymm, 또는 xmm 레지스터(벡터 길이는 64 바이트, 32 바이트, 또는 16 바이트)

[0185]

[0186] 다시 말해, 벡터 길이 필드(1359B)는 최대 길이와 하나 이상의 다른 더 짧은 길이들 사이에서 선택하고, 각각의 그러한 더 짧은 길이는 선행 길이의 절반 길이이고; 벡터 길이 필드(1359B)를 갖지 않는 명령어 템플릿들은 최대 벡터 길이에 대해 동작한다. 또한, 일 실시예에서, 특정적 벡터 친화적 명령어 포맷(1400)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.

[0187] 기입 마스크 레지스터들(1515) - 도시된 실시예에서, 각각 64 비트 크기인 8개의 기입 마스크 레지스터(k0 내지 k7)가 존재한다. 대안적인 실시예에서, 기입 마스크 레지스터들(1515)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.

[0188] 범용 레지스터들(1525) - 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.

[0189] MMX 패킹 정수 플랫 레지스터 파일(1550)이 에일리어싱되는 스칼라 부동 소수점 스택 레지스터 파일(x87 스택)(1545)-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.

[0190] 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.

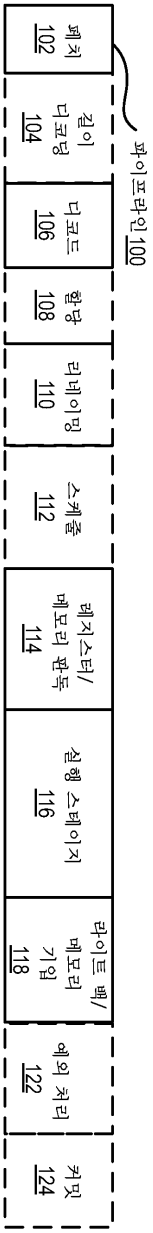
[0191] 일 실시예에서 본 명세서에 설명된 명령어들은 특정 연산을 수행하도록 구성된 또는 미리 결정된 기능을 갖는 주문형 집적 회로(ASIC)와 같은 하드웨어의 특정 구성을 지칭한다. 이러한 전자 디바이스들은 전형적으로, 하나 이상의 저장 디바이스(비일시적인 머신 판독 가능 저장 매체), 사용자 입력/출력 디바이스(예를 들어, 키보드, 터치스크린 및/또는 디스플레이) 및 네트워크 접속과 같은 하나 이상의 다른 컴포넌트에 연결된 하나 이상의 프로세서의 세트를 포함한다. 프로세서들의 세트와 다른 컴포넌트들의 연결은 전형적으로 하나 이상의 버스 및 브리지(버스 제어기로 또한 지칭됨)를 통해 이루어진다. 저장 디바이스, 및 네트워크 트래픽을 반송하는 신호들은 하나 이상의 머신 판독 가능 저장 매체 및 머신 판독 가능 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.

[0192] 전술한 설명에서, 본 발명은 특정 예시적인 실시예들을 참조하여 설명되었다. 그러나, 첨부된 청구항들에서 제시된 바와 같은 본 발명의 더 넓은 사상 및 범위를 벗어남이 없이 그에 대한 다양한 수정 및 변경들이 행해질 수 있다는 것은 명백할 것이다. 특정 경우에, 본 발명의 주제를 모호하게 하는 것을 피하기 위해 잘 알려진 구

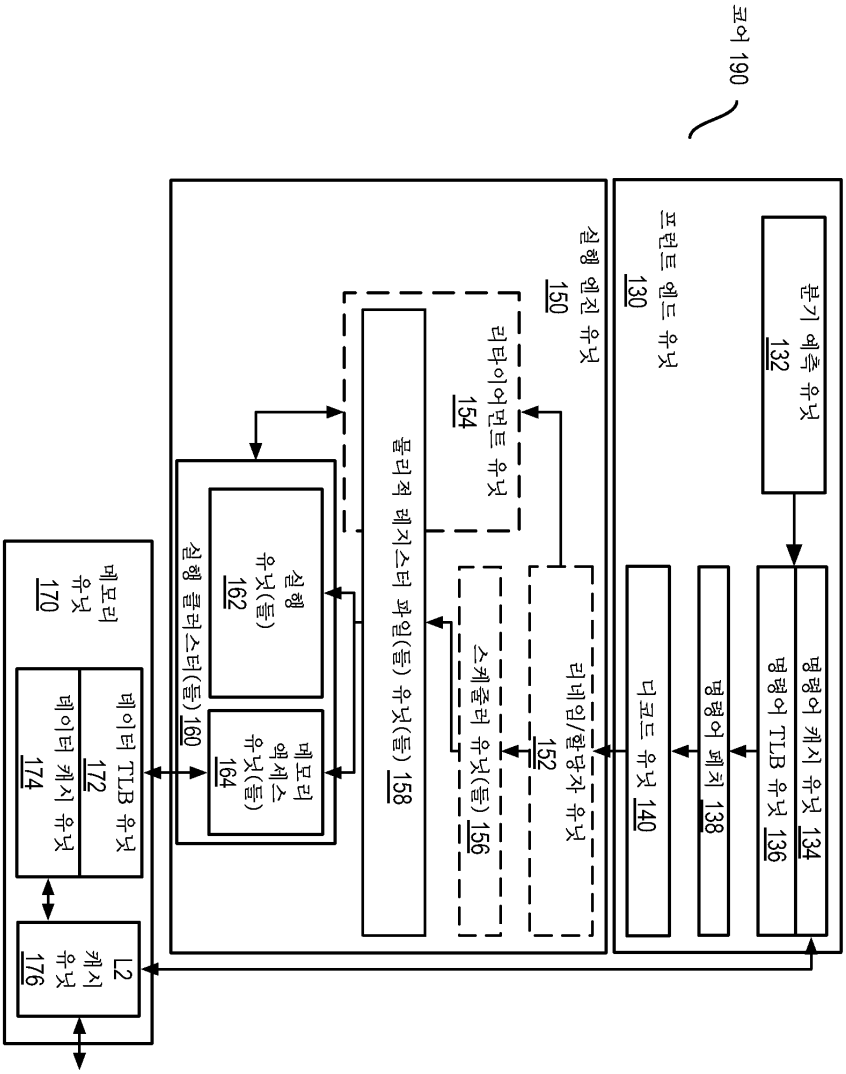
조들 및 기능들은 상세하게 설명되지 않았다. 명세서 및 도면은 따라서 제한적 개념이 아닌 예시적인 것으로 간주되어야 한다. 따라서, 본 발명의 범위 및 사상은 이하의 청구항들에 관하여 판단되어야 한다.

도면

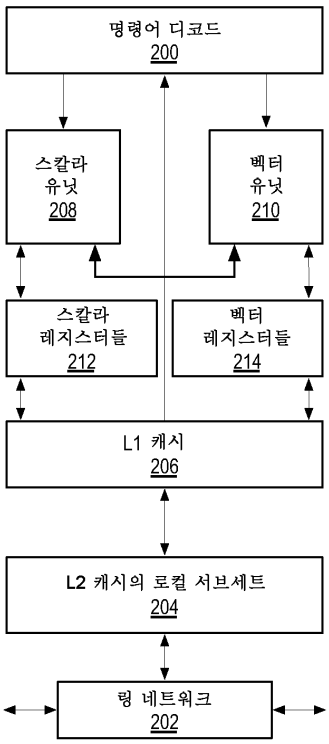
도면1a



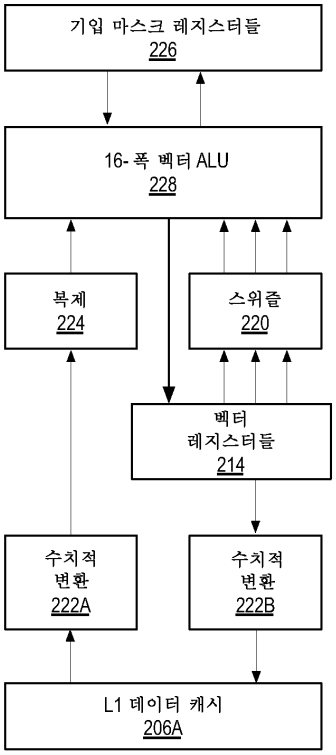
도면1b


코어 190
프런트 엔드 유닛 130
분기 예측 유닛 132
명령어 캐시 유닛 134
명령어 TLB 유닛 136
명령어 페치 138
디코드 유닛 140
실행 엔진 유닛 150
리네임/할당자 유닛 152
스케줄러 유닛(들) 156
리타이어먼트 유닛 154
물리적 레지스터 파일(들) 유닛(들) 158
실행 클러스터(들) 160
실행 유닛(들) 162
메모리 액세스 유닛(들) 164
메모리 유닛 170
데이터 TLB 유닛 172
데이터 캐시 유닛 174
L2 캐시 유닛 176

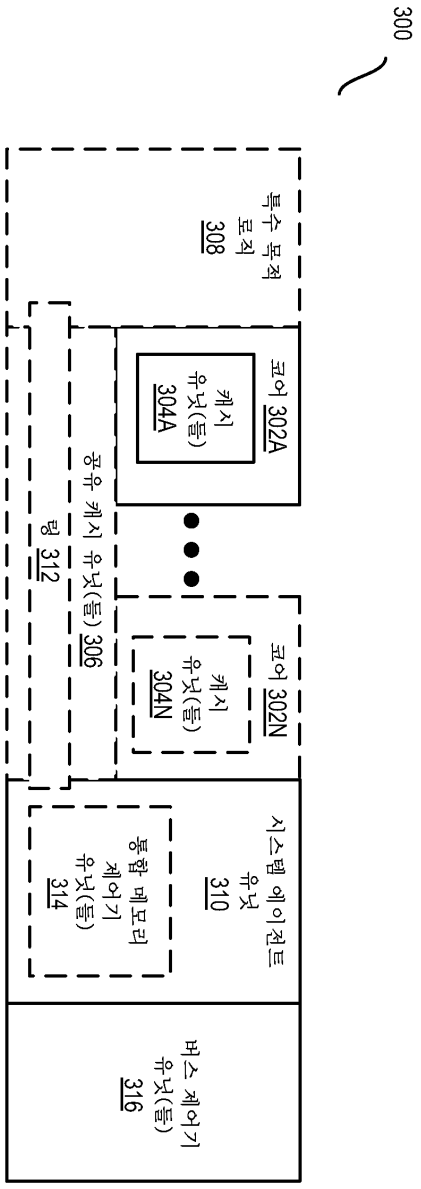
도면2a


명령어 디코드
200
스칼라
유닛
208
벡터
유닛
210
스칼라
레지스터들
212
벡터
레지스터들
214
L1 캐시
206
L2 캐시의 로컬 서브세트
204
링 네트워크
202

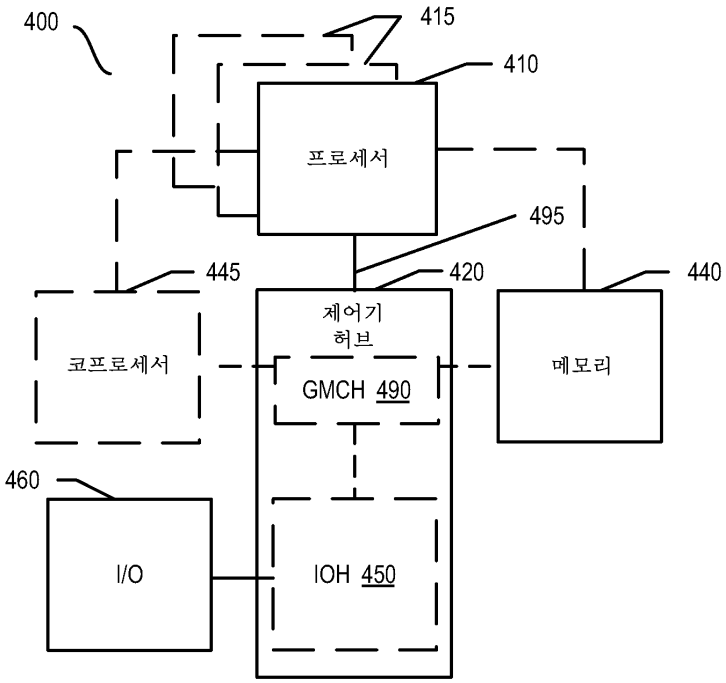
도면2b


기입 마스크 레지스터들
226
16-폭 벡터 ALU
228
복제
224
스위즐
220
벡터
레지스터들
214
수치적
변환
222A
수치적
변환
222B
L1 데이터 캐시
206A

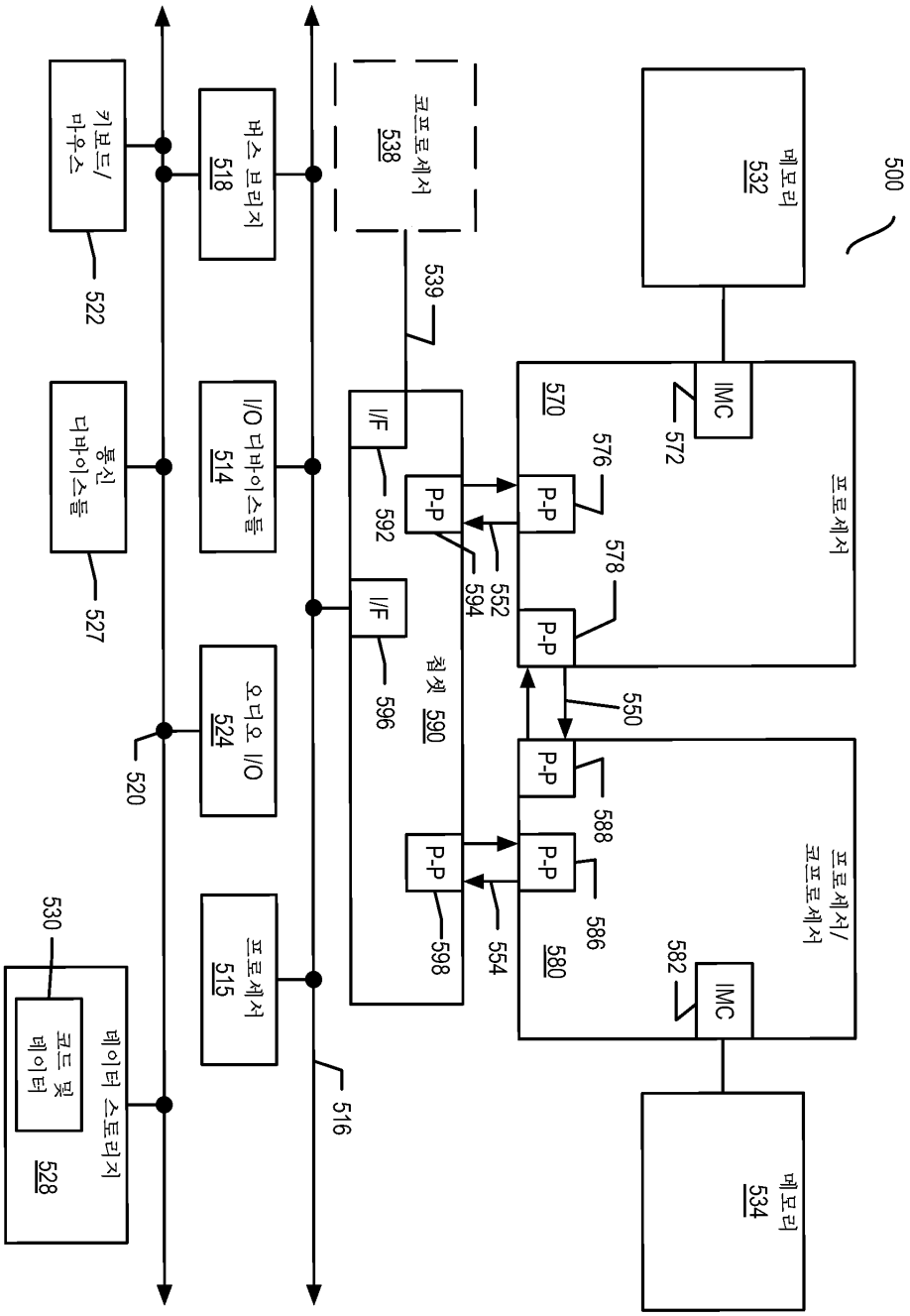
도면3


300
특수 목적 로직 308
코어 302A
캐시 유닛(들) 304A
코어 302N
캐시 유닛(들) 304N
공유 캐시 유닛(들) 306
링 312
시스템 에이전트 유닛 310
통합 메모리 제어기 유닛(들) 314
버스 제어기 유닛(들) 316

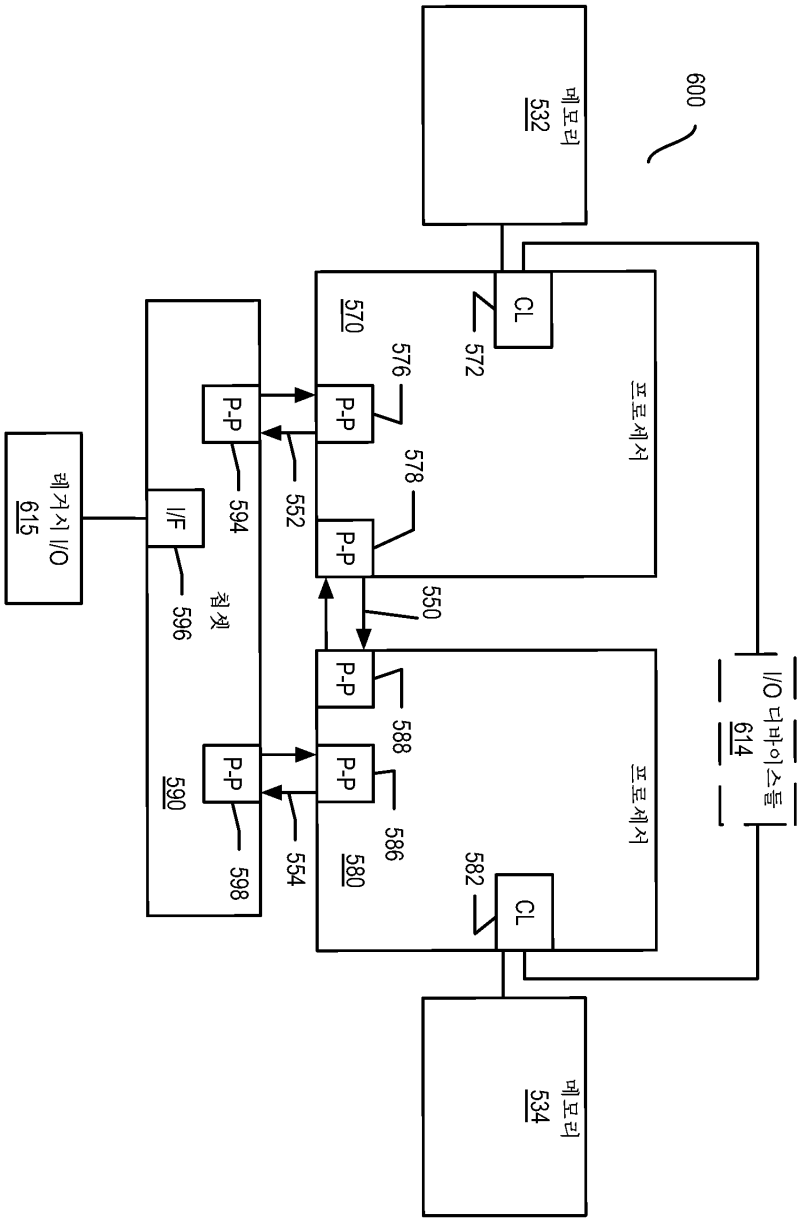
도면4


400
415
410
프로세서
495
445
420
440
제어기
허브
코프로세서
GMCH 490
메모리
460
I/O
IOH 450

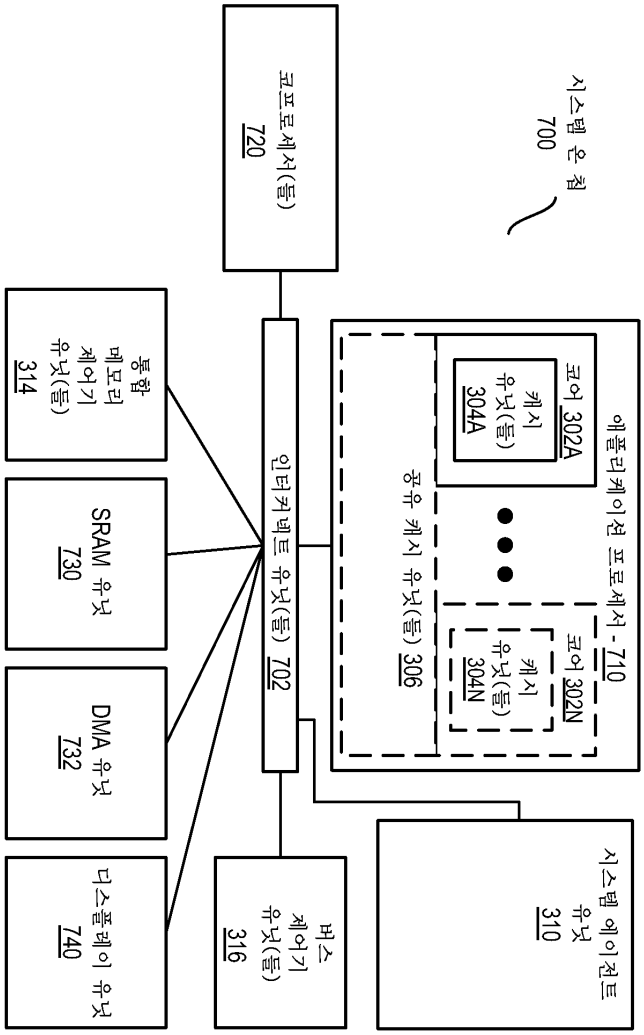
도면5


500
메모리
532
IMC
572
프로세서
570
576
P-P
578
P-P
552
594
I/F
592
539
코프로세서
538
550
P-P
588
P-P
586
프로세서/
코프로세서
580
582
IMC
메모리
534
칩셋 590
I/F
596
P-P
598
554
516
버스 브리지
518
I/O 디바이스들
514
오디오 I/O
524
520
프로세서
515
키보드/
마우스
522
통신
디바이스들
527
데이터 스토리지
528
코드 및
데이터
530

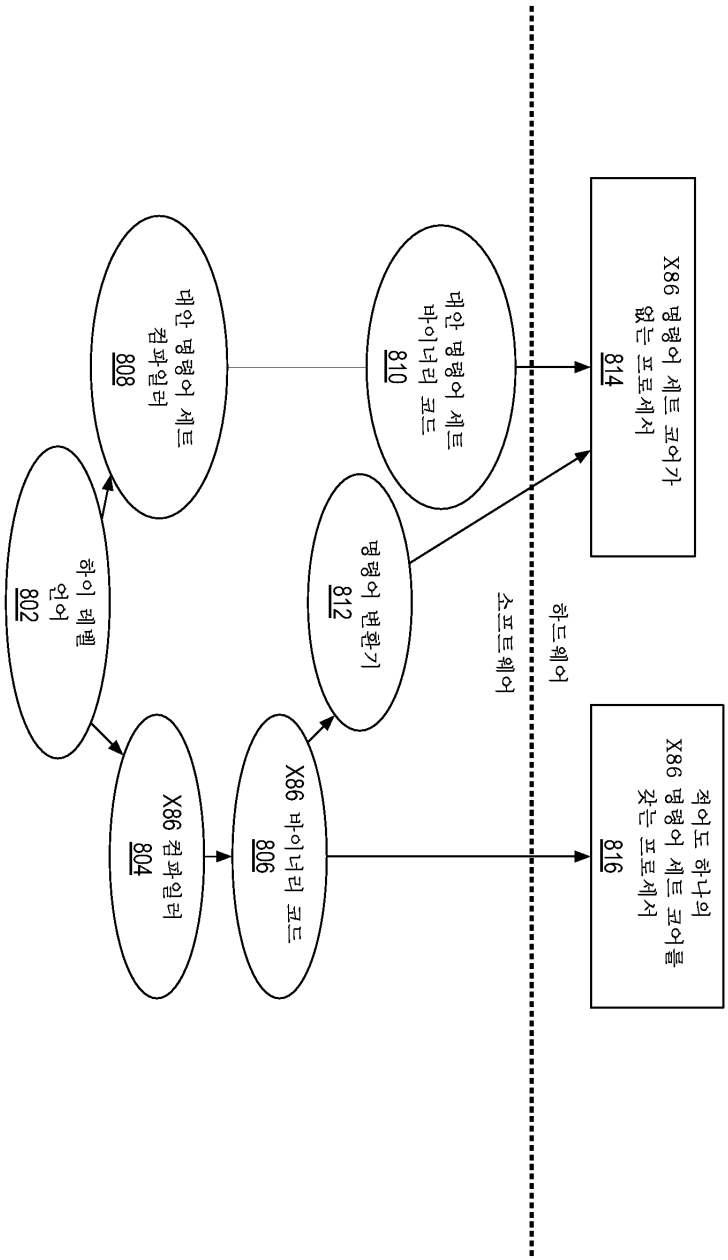
도면6


600
메모리
532
CL
572
570
576
578
프로세서
P-P
P-P
P-P
552
550
588
586
582
580
554
P-P
P-P
P-P
I/F
칩셋
596
590
594
598
레거시 I/O
615
I/O 디바이스들
614
메모리
534

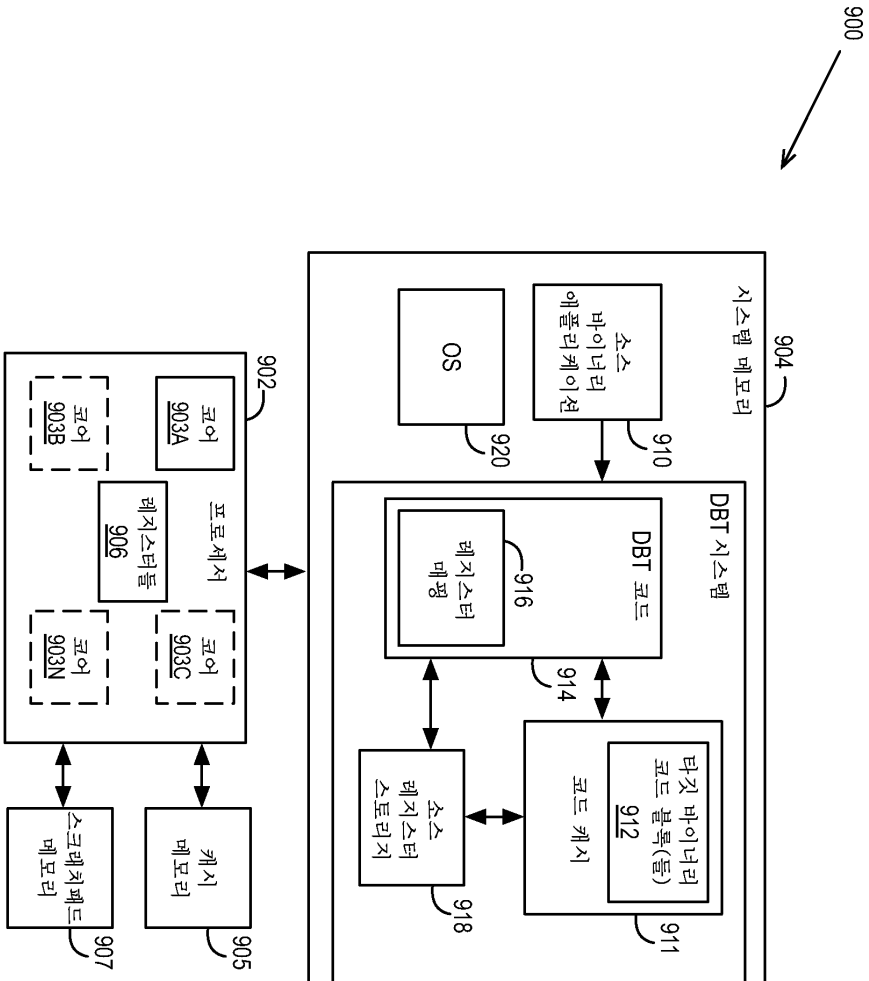
도면7


시스템 온 칩
700
코프로세서(들)
720
통합 메모리 제어기 유닛(들)
314
SRAM 유닛
730
DMA 유닛
732
디스플레이 유닛
740
인터커넥트 유닛(들) 702
버스 제어기 유닛(들)
316
애플리케이션 프로세서 - 710
코어 302A
캐시 유닛(들) 304A
코어 302N
캐시 유닛(들) 304N
공유 캐시 유닛(들) 306
시스템 에이전트 유닛
310

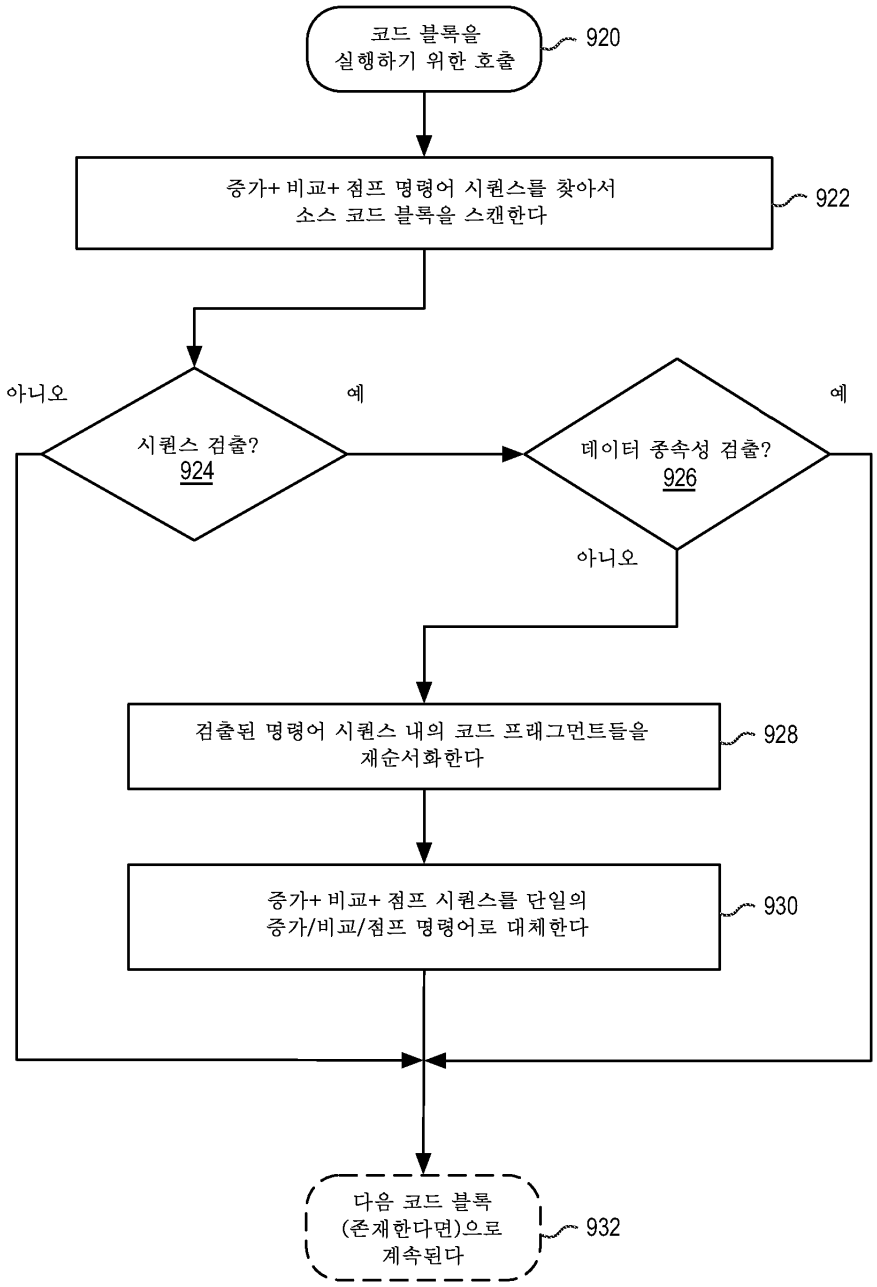
도면8


하이 레벨 언어
802
X86 컴파일러
804
X86 바이너리 코드
806
대안 명령어 세트 컴파일러
808
대안 명령어 세트 바이너리 코드
810
명령어 변환기
812
X86 명령어 세트 코어가 없는 프로세서
814
적어도 하나의 X86 명령어 세트 코어를 갖는 프로세서
816
소프트웨어
하드웨어

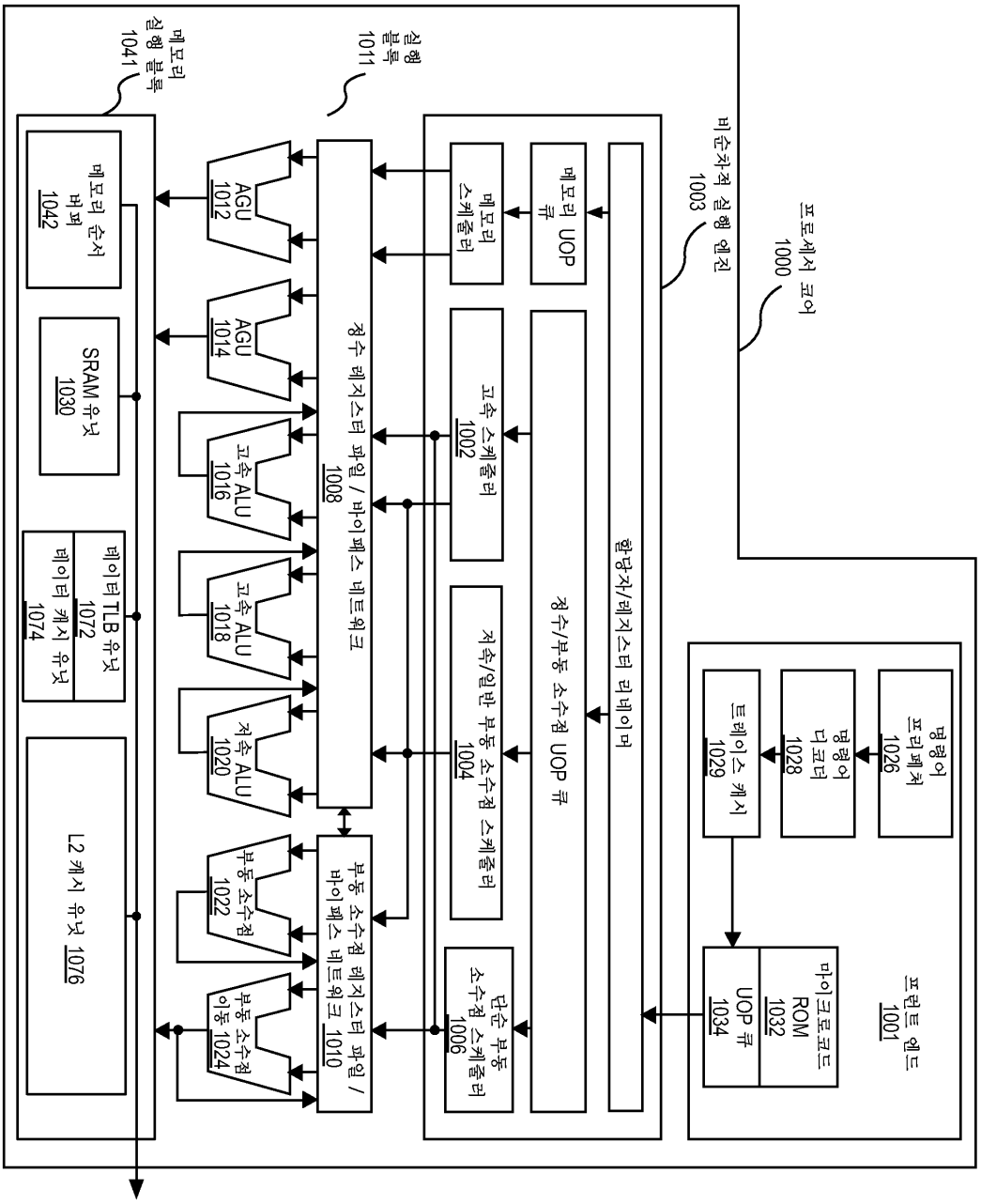
도면9a


900
시스템 메모리
904
OS
920
소스 바이너리 애플리케이션
910
DBT 시스템
DBT 코드
914
레지스터 매핑
916
타깃 바이너리 코드 블록(들)
912
코드 캐시
911
소스 레지스터 스토리지
918
915
프로세서
902
코어 903A
코어 903B
레지스터들 906
코어 903C
코어 903N
캐시 메모리
905
스크래치패드 메모리
907

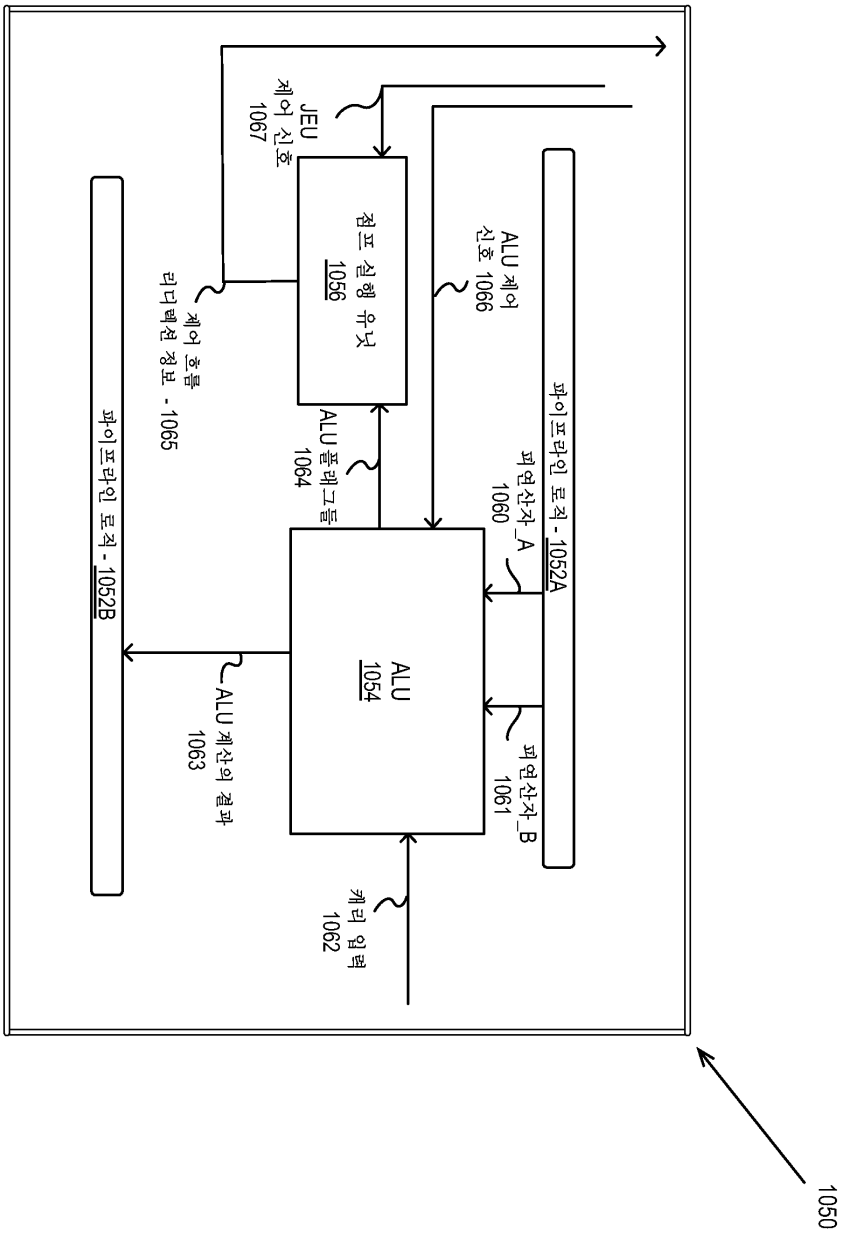
도면9b


코드 블록을
실행하기 위한 호출
920
증가+ 비교+ 점프 명령어 시퀀스를 찾아서
소스 코드 블록을 스캔한다
922
아니오
예
시퀀스 검출?
924
데이터 종속성 검출?
926
예
아니오
검출된 명령어 시퀀스 내의 코드 프래그먼트들을
재순서화한다
928
증가+ 비교+ 점프 시퀀스를 단일의
증가/비교/점프 명령어로 대체한다
930
다음 코드 블록
(존재한다면)으로
계속된다
932

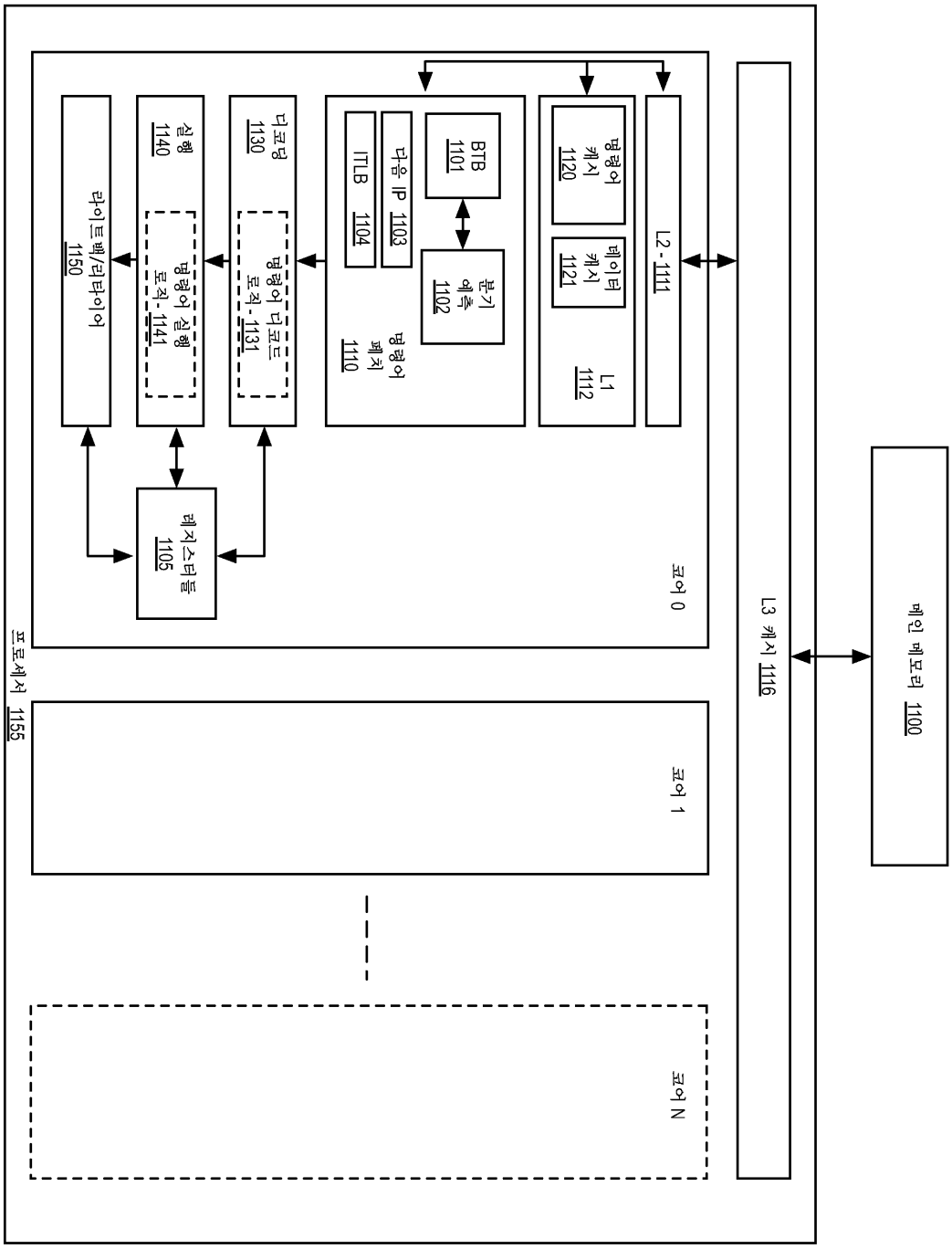
도면10a


프로세서 코어
1000
비순차적 실행 엔진
1003
프런트 엔드
1001
명령어 프리페처
1026
명령어 디코더
1028
트레이스 캐시
1029
마이크로코드 ROM
1032
UOP 큐
1034
할당자/레지스터 리네이머
정수/부동 소수점 UOP 큐
메모리 UOP 큐
메모리 스케줄러
고속 스케줄러
1002
저속/일반 부동 소수점 스케줄러
1004
단순 부동 소수점 스케줄러
1006
실행 블록
1011
정수 레지스터 파일 / 바이패스 네트워크
1008
부동 소수점 레지스터 파일 / 바이패스 네트워크 1010
AGU 1012
AGU 1014
고속 ALU 1016
고속 ALU 1018
저속 ALU 1020
부동 소수점 1022
부동 소수점 이동 1024
메모리 실행 블록
1041
메모리 순서 버퍼
1042
SRAM 유닛
1030
데이터 TLB 유닛
1072
데이터 캐시 유닛
1074
L2 캐시 유닛 1076

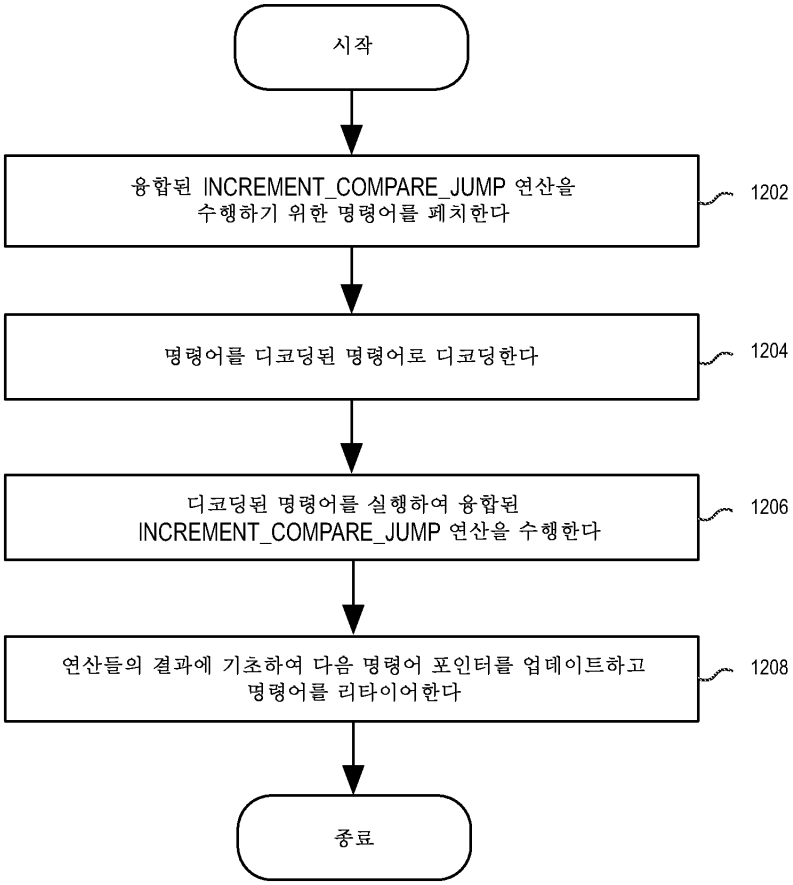
도면10b


1050
파이프라인 로직 - 1052A
파이프라인 로직 - 1052B
점프 실행 유닛 1056
ALU 1054
JEU 제어 신호 1067
ALU 제어 신호 1066
제어 흐름 리디렉션 정보 - 1065
ALU 플래그들 1064
피연산자_A 1060
피연산자_B 1061
ALU 계산의 결과 1063
캐리 입력 1062

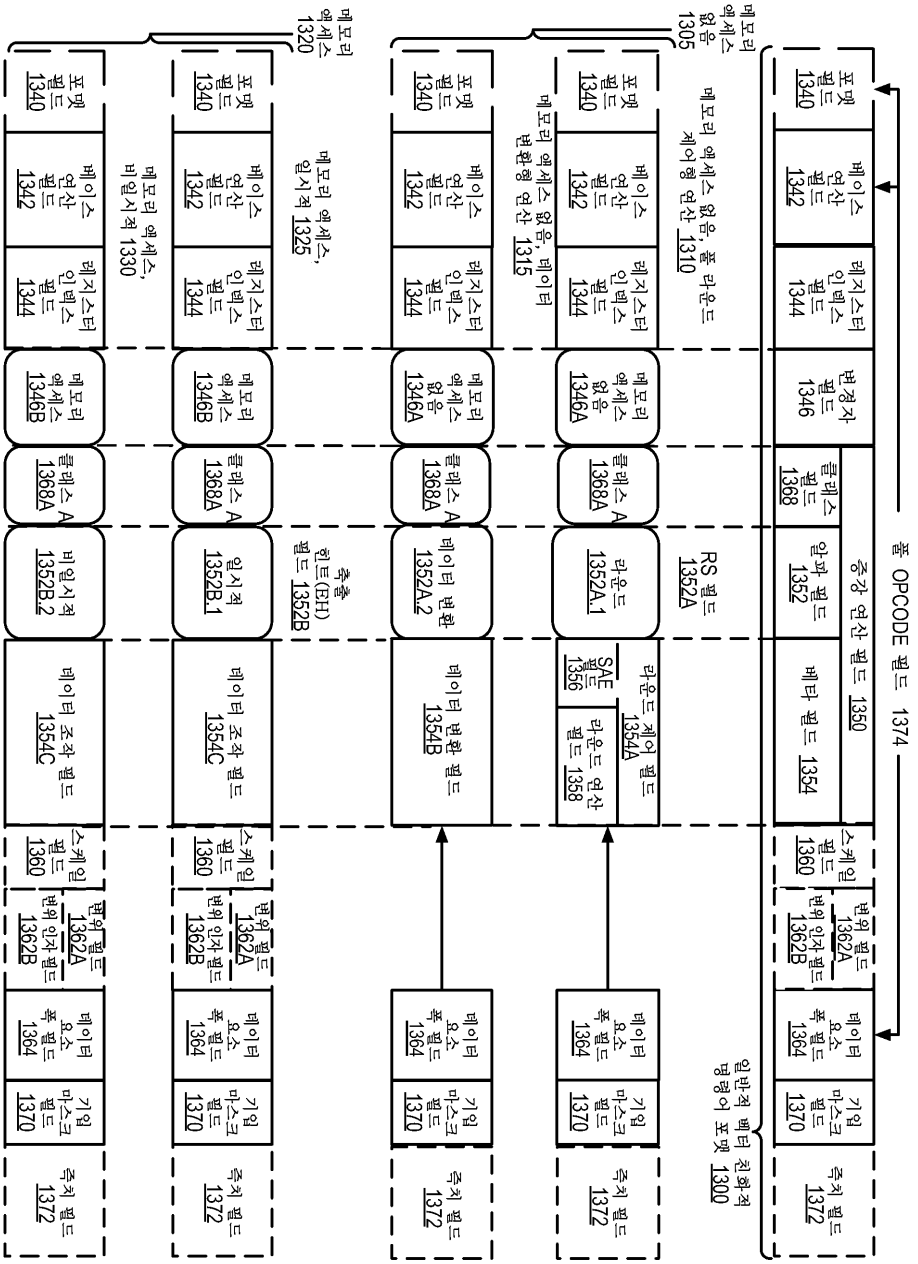
도면11


프로세서 1155
코어 0
명령어 캐시 1120
데이터 캐시 1121
L1 1112
L2 - 1111
BTB 1101
분기 예측 1102
다음 IP 1103
ITLB 1104
명령어 페치 1110
디코딩 1130
명령어 디코드 로직 - 1131
실행 1140
명령어 실행 로직 - 1141
라이트백/리타이어 1150
레지스터들 1105
코어 1
코어 N
L3 캐시 1116
메인 메모리 1100

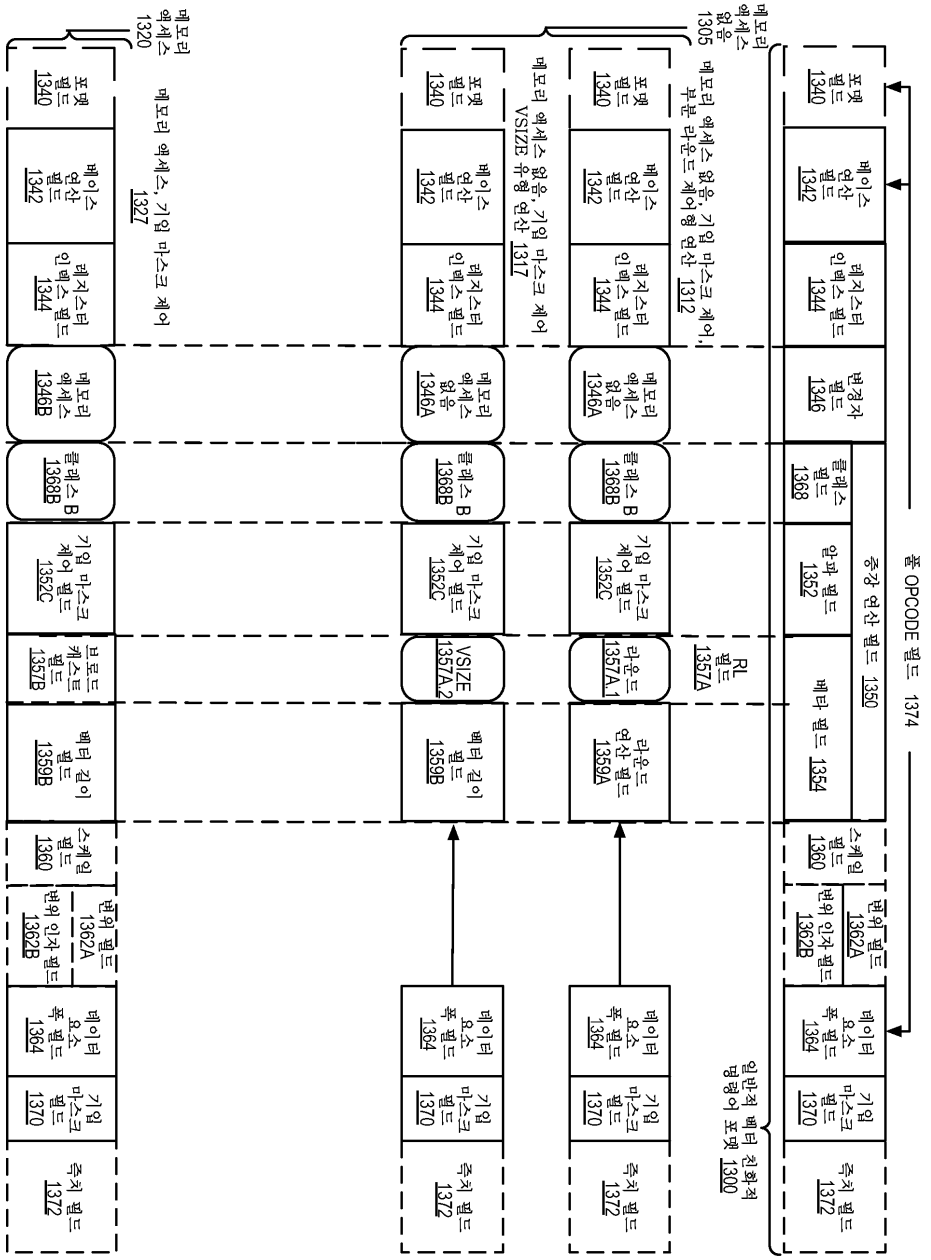
도면12


시작
융합된 INCREMENT_COMPARE_JUMP 연산을 수행하기 위한 명령어를 페치한다
1202
명령어를 디코딩된 명령어로 디코딩한다
1204
디코딩된 명령어를 실행하여 융합된 INCREMENT_COMPARE_JUMP 연산을 수행한다
1206
연산들의 결과에 기초하여 다음 명령어 포인터를 업데이트하고 명령어를 리타이어한다
1208
종료

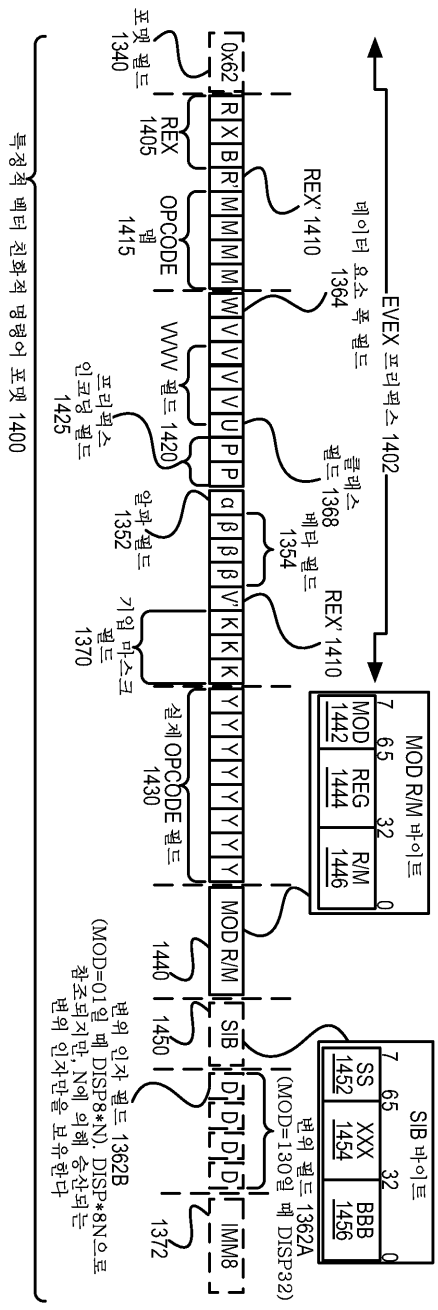
도면13a


폴 OPCODE 필드 1374
포맷 필드 1340
베이스 연산 필드 1342
레지스터 인덱스 필드 1344
변경자 필드 1346
클래스 필드 1368
중강 연산 필드 1350
스케일 필드 1360
변위 필드 1362A
변위 인자 필드 1362B
데이터 요소 폭 필드 1364
기입 마스크 필드 1370
즉치 필드 1372
일반적 벡터 친화적 명령어 포맷 1300
메모리 액세스 없음 1305
메모리 액세스 없음, 폴 라운드 제어형 연산 1310
메모리 액세스 없음, 데이터 변환형 연산 1315
메모리 액세스 1320
메모리 액세스, 일시적 1325
메모리 액세스, 비일시적 1330
메모리 액세스 없음 1346A
클래스 A 1368A
알파 필드 1352
RS 필드 1352A
라운드 1352A.1
데이터 변환 1352A.2
SAE 필드 1356
라운드 제어 필드 1354A
라운드 연산 필드 1358
데이터 변환 필드 1354B
메모리 액세스 1346B
클래스 B 1368B
베타 필드 1354
축출 힌트(EH) 필드 1352B
일시적 1352B.1
비일시적 1352B.2
데이터 조차 필드 1354C

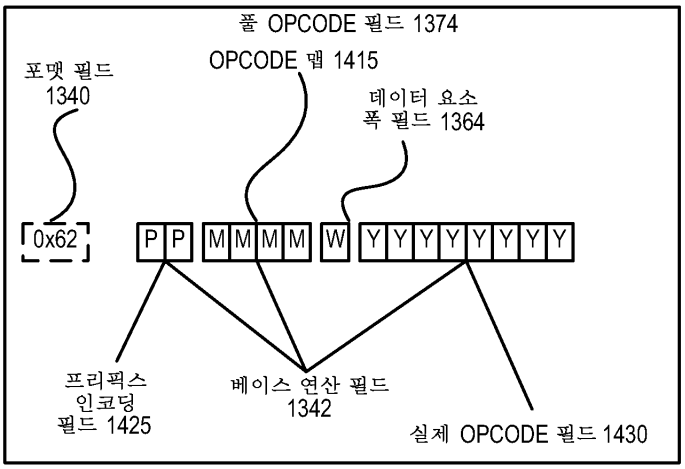
도면13b


메모리 액세스 없음 1305
메모리 액세스 없음, 기입 마스크 제어, 부분 라운드 제어형 연산 1312
메모리 액세스 없음, 기입 마스크 제어, VSIZE 유형 연산 1317
메모리 액세스 1320
메모리 액세스, 기입 마스크 제어 1327
포맷 필드 1340
베이스 연산 필드 1342
레지스터 인덱스 필드 1344
변경자 필드 1346
메모리 액세스 없음 1346A
메모리 액세스 1346B
클래스 필드 1368
클래스 B 1368B
알파 필드 1352
기입 마스크 제어 필드 1352C
베타 필드 1354
증강 연산 필드 1350
RL 필드 1357A
라운드 1357A.1
VSIZE 1357A.2
브로드캐스트 필드 1357B
라운드 연산 필드 1359A
벡터 길이 필드 1359B
스케일 필드 1360
변위 필드 1362A
변위 인자 필드 1362B
데이터 요소 폭 필드 1364
기입 마스크 필드 1370
즉치 필드 1372
풀 OPCODE 필드 1374
일반적 벡터 친화적 명령어 포맷 1300

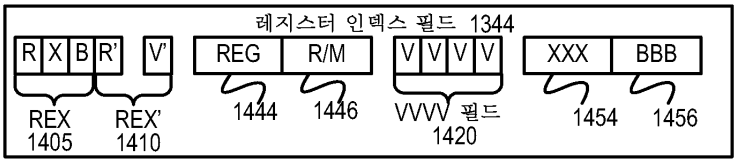
도면14a


특정적 벡터 친화적 명령어 포맷 1400
포맷 필드 1340
0x62
REX 1405
R X B R' M M M M
OPCODE 맵 1415
W V V V V U P P
VVVV 필드 1420
프리픽스 인코딩 필드 1425
α β β β V' K K K
기입 마스크 필드 1370
알파 필드 1352
베타 필드 1354
REX' 1410
클래스 필드 1368
EVEX 프리픽스 1402
데이터 요소 폭 필드 1364
Y Y Y Y Y Y Y Y
실제 OPCODE 필드 1430
MOD R/M
SIB
D D D D
IMM8
1440
1450
1372
변위 인자 필드 1362B
(MOD=01일 때 DISP8*N). DISP*8N으로 참조되지만, N에 의해 승산되는 변위 인자만을 보유한다
변위 필드 1362A
(MOD=10일 때 DISP32)
MOD R/M 바이트
7 6 5 3 2 0
MOD 1442
REG 1444
R/M 1446
SIB 바이트
SS 1452
XXX 1454
BBB 1456

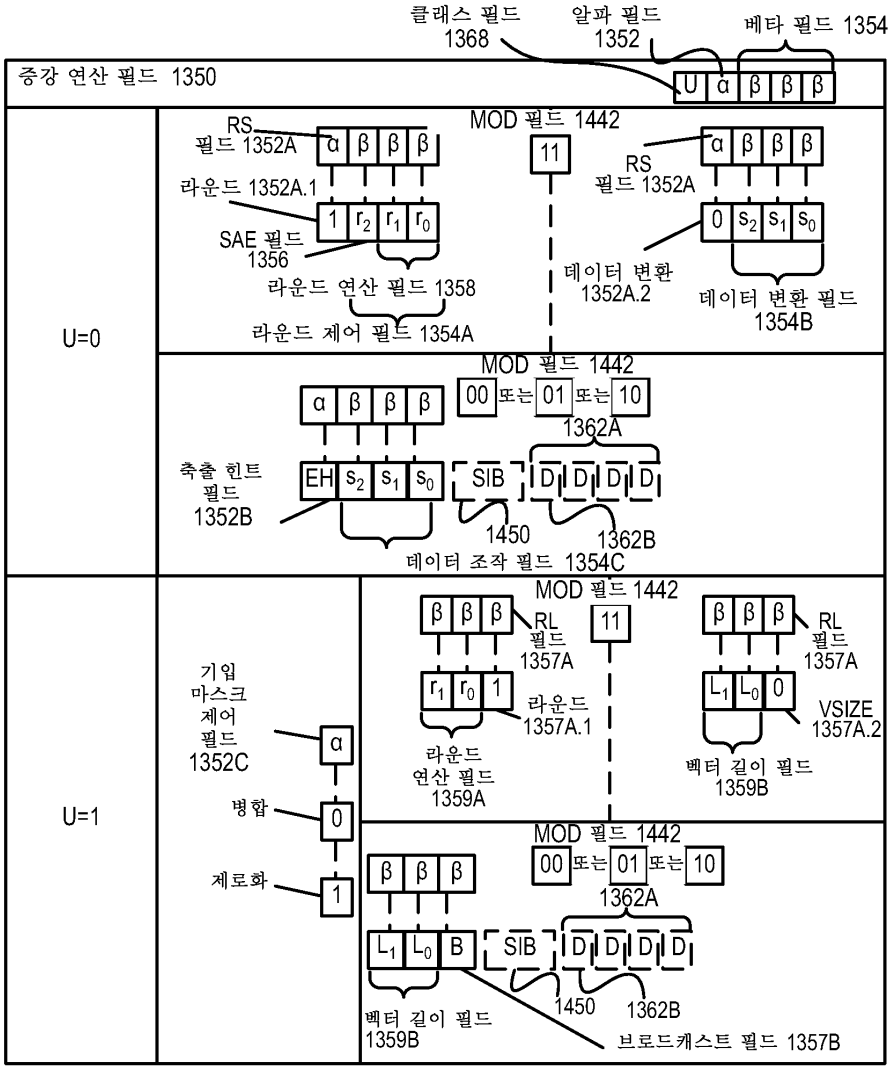
도면14b


풀 OPCODE 필드 1374
포맷 필드
1340
OPCODE 맵 1415
데이터 요소
폭 필드 1364
0x62
P P
M M M M
W
Y Y Y Y Y Y Y Y
프리픽스
인코딩
필드 1425
베이스 연산 필드
1342
실제 OPCODE 필드 1430

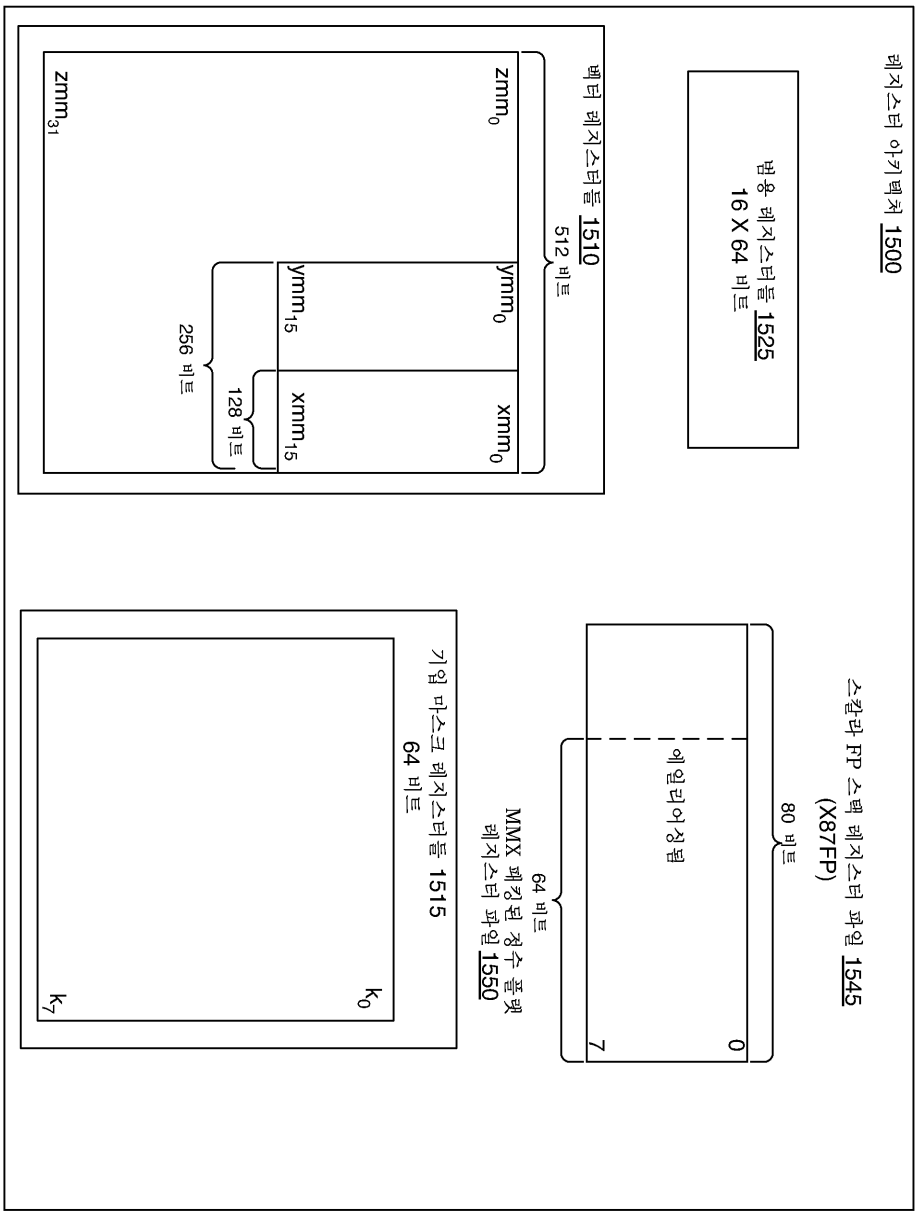
도면14c


레지스터 인덱스 필드 1344
R X B R'
V'
REG
R/M
V V V V
XXX
BBB
REX
1405
REX'
1410
1444
1446
VVVV 필드
1420
1454
1456

도면14d


클래스 필드 1368
알파 필드 1352
베타 필드 1354
증강 연산 필드 1350
U α β β β
U=0
RS 필드 1352A
α β β β
MOD 필드 1442
11
라운드 1352A.1
1 r2 r1 r0
SAE 필드 1356
라운드 연산 필드 1358
라운드 제어 필드 1354A
RS 필드 1352A
α β β β
0 s2 s1 s0
데이터 변환 1352A.2
데이터 변환 필드 1354B
MOD 필드 1442
α β β β
00 또는 01 또는 10
1362A
축출 힌트 필드 1352B
EH s2 s1 s0
SIB
D D D D
1450
1362B
데이터 조작 필드 1354C
U=1
기입 마스크 제어 필드 1352C
α
병합
0
제로화
1
MOD 필드 1442
β β β
RL 필드 1357A
11
r1 r0 1
라운드 1357A.1
라운드 연산 필드 1359A
β β β
RL 필드 1357A
L1 L0 0
VSIZE 1357A.2
벡터 길이 필드 1359B
MOD 필드 1442
β β β
00 또는 01 또는 10
1362A
L1 L0 B
SIB
D D D D
1450
1362B
벡터 길이 필드 1359B
브로드캐스트 필드 1357B

도면15


레지스터 아키텍처 1500
범용 레지스터들 1525
16 X 64 비트
벡터 레지스터들 1510
512 비트
zmm0
ymm0
xmm0
ymm15
xmm15
256 비트
128 비트
zmm31
스칼라 FP 스택 레지스터 파일 1545
(X87FP)
80 비트
0
에일리어싱 됨
7
64 비트
MMX 패킹된 정수 플랫
레지스터 파일 1550
기입 마스크 레지스터들 1515
64 비트
k0
k7