(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2010년11월09일
(11) 등록번호 10-0993018
(24) 등록일자 2010년11월02일

(51) Int. Cl.
G06F 12/08 (2006.01) *G06F 9/32* (2006.01)
G06F 9/34 (2006.01) *G06F 9/00* (2006.01)
(21) 출원번호 10-2006-7008482
(22) 출원일자(국제출원일자) 2004년06월28일
심사청구일자 2009년05월04일
(85) 번역문제출일자 2006년05월01일
(65) 공개번호 10-2006-0108644
(43) 공개일자 2006년10월18일
(86) 국제출원번호 PCT/US2004/020721
(87) 국제공개번호 WO 2005/041024
국제공개일자 2005년05월06일
(30) 우선권주장
10/676,437 2003년10월01일 미국(US)
(56) 선행기술조사문헌
KR1020010006188 A
KR1020010006194 A
US20010014941 A1
KR100546087 B1
전체 청구항 수 : 총 10 항

(73) 특허권자
글로벌파운드리즈 인크.
케이만 아일랜드 케이와이1-1104 그랜드 케이만 어그랜드 하우스 피.오.박스 309 메이플즈 코포레이트 서비시즈 리미티드
(72) 발명자
알서프 미쉘
미국 텍사스 78796 오스틴 켐퍼 코브 2103
스마우스 그레고리 윌리암
미국 텍사스 78739 오스틴 사빈 힐 레인 11120
(뒷면에 계속)
(74) 대리인
박장원

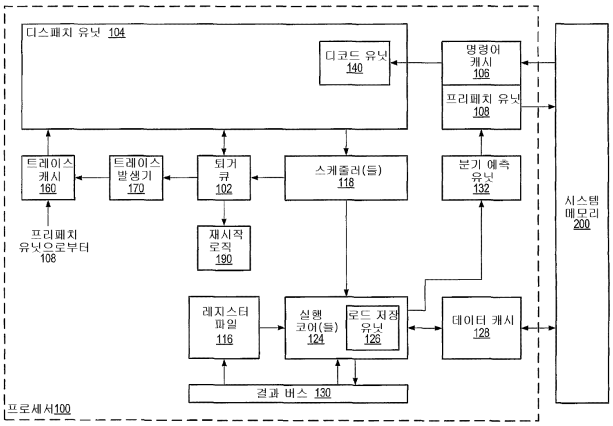
심사관 : 권오성

(54) 트레이스 캐시 기반 프로세서에서 예외 명령어들을 처리하는 시스템 및 방법

(57) 요 약

시스템은 명령어 캐시(106), 트레이스 캐시(160) 및 트레이스 발생기(170)를 포함하며, 상기 트레이스 캐시(160)는 다수의 트레이스 캐시 엔트리들(162)을 포함하고, 그리고 상기 트레이스 발생기(170)는 상기 명령어 캐시(106) 및 트레이스 캐시(160)에 연결된다. 상기 트레이스 발생기(170)는 다수의 트레이스 캐시 엔트리들(162) 중 하나에 저장을 위해 상기 명령어 캐시(106)에 의해 출력되는 명령어들의 그룹을 수신하도록 구성된다. 상기 트레이스 발생기(170)는 상기 명령어들의 그룹 내에서 예외 명령어를 검출하도록 구성되고 상기 예외 명령어가 어떠한 비-예외(non-exceptional) 명령어와도 상기 다수의 트레이스 캐시 엔트리들(162) 중 동일한 트레이스 캐시 엔트리에 저장되는 것을 방지하도록 구성된다.

대 표 도



(72) 발명자

픽케트 제임스 케이.

미국 텍사스 78733 오스틴 팔로미노 리지 드라이브 1700 #2

엠친 브라이언 디.

미국 텍사스 78610 부다 카운티 옥스 드라이브 25

필리포 마이클 에이.

미국 텍사스 78652 만차카 차파랄 로드 2030

샌더 벤자민 티.

미국 텍사스 78735 오스틴 메디슨 크릭 5701

특허청구의 범위

청구항 1

명령어 캐시(106)와;

복수의 트레이스 캐시 엔트리들(162)을 포함하는 트레이스 캐시(160)와; 그리고

상기 명령어 캐시(106) 및 상기 트레이스 캐시(160)에 결합되는 트레이스 발생기(170)를 포함하여 구성되며,

상기 트레이스 발생기(170)는 상기 복수의 트레이스 캐시 엔트리들(162) 중 하나에 저장하기 위해 상기 명령어 캐시(106)에 의해 출력되는 명령어들의 그룹을 수신하며, 상기 트레이스 발생기(170)는 상기 명령어들의 그룹 내에서 예외 명령어를 검출하고, 상기 예외 명령어가 임의의 비-예외 명령어와 동일한 상기 복수의 트레이스 캐시 엔트리들(162) 중 하나에 저장되는 것을 막는 것을 특징으로 하는 시스템.

청구항 2

제 1 항에 있어서,

상기 트레이스 발생기(170)는 상기 명령어들을, 적어도 부분적으로 디코딩된 형태로 상기 트레이스 캐시(160)에 저장하는 것을 특징으로 하는 시스템.

청구항 3

제 2 항에 있어서,

실행된 명령어들을 퇴거하기 위하여 결합되는 퇴거큐(retire queue)(102)를 더 포함하고, 상기 퇴거큐(102)는 임의의 명령어와 동일한 트레이스 캐시 엔트리(162)에 포함되는 모든 명령어들이 퇴거 준비가 될 때까지, 상기 트레이스 캐시(160)로부터 페치되는 상기 임의의 명령어의 퇴거를 막는 것을 특징으로 하는 시스템.

청구항 4

제 2 항에 있어서,

상기 명령어 캐시(106)로부터 수신되는 명령어들을 디스패치하는 디스패치 유닛(104)을 더 포함하고, 상기 디스패치 유닛(104)은 상기 명령어 캐시(106)로부터 수신되는 명령어들의 그룹 내에서 예외 명령어를 검출하고 상기 예외 명령어의 표시를 상기 트레이스 발생기(170)에 제공하며, 상기 트레이스 발생기(170)는 상기 디스패치 유닛(104)으로부터의 상기 표시에 응답하여 상기 예외 명령어를 검출하는 것을 특징으로 하는 시스템.

청구항 5

제 2 항에 있어서,

상기 트레이스 발생기(170)는 상기 예외 명령어를 1개 이상의 다른 명령어들과는 다른 트레이스 캐시 엔트리(162)에 저장하고, 상기 트레이스 발생기(170)는 상기 예외 명령어와 함께 임의의 다른 명령어들을 상기 다른 트레이스 캐시 엔트리(162)에 저장하지 않는 것을 특징으로 하는 시스템.

청구항 6

트레이스 캐시(160) 내의 트레이스 캐시 엔트리(162)에 저장하기 위한 명령어들의 그룹을 수신하는 단계와;

상기 명령어들의 그룹에 포함되는 1개 이상의 명령어들을 상기 트레이스 캐시 엔트리(162)에 저장하는 단계와;

상기 명령어들의 그룹 내에서 예외 명령어를 검출하는 단계와; 그리고

상기 검출에 응답하여, 상기 1개 이상의 명령어들을 갖는 상기 트레이스 캐시 엔트리(162) 내에 상기 예외 명령어를 저장하지 않는 단계를 포함하는 것을 특징으로 하는 방법.

청구항 7

제 6 항에 있어서,

상기 저장하는 단계는 상기 1개 이상의 명령어들을 적어도 부분적으로 디코딩된 형태로 상기 트레이스 캐시 엔트리(162)에 저장하는 단계를 포함하는 것을 특징으로 하는 방법.

청구항 8

제 7 항에 있어서,

임의의 명령어와 동일한 트레이스 캐시 엔트리(162)에 포함되는 모든 명령어들이 퇴거 준비가 될 때까지, 상기 트레이스 캐시(160)로부터 페치되는 상기 임의의 명령어의 퇴거를 막는 단계를 더 포함하는 것을 특징으로 하는 방법.

청구항 9

제 8 항에 있어서,

상기 임의의 명령어의 퇴거 이후 경과되는 사이클의 수를 모니터하는 단계와; 그리고

상기 사이클의 수가 임계 수를 넘는 경우, 상기 트레이스 캐시(160)로부터 페치되는 명령어들을 실행하는 프로세싱 파이프라인을 플러시(flush)하고 상기 명령어 캐시(106)로부터의 실행을 재개하는 단계를 더 포함하는 것을 특징으로 하는 방법.

청구항 10

시스템 메모리(404)와; 그리고

상기 시스템 메모리(404)에 결합되는 프로세서(100)를 포함하는 컴퓨터 시스템(400)에 있어서,

상기 프로세서(100)는,

명령어 캐시(106)와;

복수의 트레이스 캐시 엔트리들(162)을 포함하는 트레이스 캐시(160)와; 그리고

상기 명령어 캐시(106) 및 상기 트레이스 캐시(160)에 결합되는 트레이스 발생기(170)를 포함하며,

상기 트레이스 발생기(170)는 상기 복수의 트레이스 캐시 엔트리들(162) 중 하나에 저장하기 위해 상기 명령어 캐시(106)에 의해 출력되는 명령어들의 그룹을 수신하며, 상기 트레이스 발생기(170)는 상기 명령어들의 그룹 내에서 예외 명령어를 검출하고, 그리고 상기 트레이스 발생기(170)는 상기 예외 명령어가 상기 복수의 트레이스 캐시 엔트리들(162) 중 임의의 비-예외 명령어와 동일한 트레이스 캐시 엔트리에 저장되는 것을 막는 것을 특징으로 하는 컴퓨터 시스템(400).

명 세 서

기 술 분 야

[0001] 본 발명은 프로세서 분야에 관한 것이고, 더욱 상세하게는 트레이스 캐시(trace cache)를 갖는 프로세서들에 관한 것이다.

배 경 기 술

[0002] 프로세서에서 처리되는 명령어(instruction)들은 1들 및 0들의 시퀀스(sequence)로 인코딩된다. 일부 프로세서 아키텍처들에서, 명령어들은 특정수의 바이트(byte)들과 같은 고정 길이로 인코딩된다. x86 구조와 같은 다른 아키텍처들에서 명령어들의 길이는 다양하다. 상기 x86 프로세서 아키텍처는 가변 길이 명령어 세트(variable length instruction set)(즉, 다른 수의 바이트들에 의해 각각 규정된 가변 명령어들이 있는 명령어 세트)를 규정한다. 예를 들어, 상기 80386 및 x86 프로세서들의 추후 버전은 특정 명령어를 특정하기 위해 1 내지 15 바이트들을 이용한다. 명령어들은 1-2 바이트들인 오피코드(opcode)와(여기서 어드레싱 모드(addressing mode)를 특정하기 위해 추가의 바이트들이 부가됨), 오퍼랜드(operand)와, 그리고 실행된 명령어에 관한 추가적인 상세(detail)를 포함한다.

일부 프로세서 아키텍처들에서, 각 명령어는 실행 전에 하나 이상의 단순한 오퍼레이션들(operations)로 디코딩된다. 명령어의 디코딩은 명령어의 각 논리 레지스터(logical register)가 매핑되는 물리적 레지스터를 결정하

고 및/또는 상기 명령어의 결과를 저장할 물리적 레지스터를 할당하기 위해 레지스터 재명명 맵(register renaming map)을 액세싱하는 것을 또한 포함한다.

명령어들의 상기 프로세서 내에 만들어진 분기 예측들에 부분적으로 근거하여 프로세서의 디코드 부분으로 페치(fetch)된다. 일반적으로, 프로세서의 디코드 부분들 및 명령어 페치의 대역폭은 실행코어(execution core)들이 각 실행 사이클 동안에 충분히 이용되는지를 결정한다. 따라서, 실행코어들이 가능한한 작업에 충분히 공급되도록 하기 위해 프로세서의 디코드 부분들 및 명령어 페치에서 충분한 대역폭을 제공할 수 있는 것이 바람직하다.

발명의 상세한 설명

[0003] 트레이스 캐시 기반의 프로세서 내의 예외 명령어들(exceptional instructions)을 취급하기 위한 방법들 및 시스템들의 다양한 실시예들이 개시된다. 일부 실시예들에 있어서, 이 시스템은 명령어 캐시와, 복수의 트레이스 캐시 엔트리들을 포함한 트레이스 캐시와, 그리고 명령어 캐시 및 트레이스 캐시에 결합되는 트레이스 발생기를 포함한다. 트레이스 발생기는 복수의 트레이스 캐시 엔트리들중 하나에 저장하도록 명령어 캐시에 의해 출력되는 명령어들의 그룹을 수신하도록 구성된다. 트레이스 발생기는, 명령어들의 그룹 내에서 예외 명령어를 검출한 다음, 그 예외 명령어가 복수의 트레이스 캐시 엔트리들 중 임의의 비 예외 명령어와 동일한 것에 저장되는 것을 막도록 구성된다. 일 실시예에서, 트레이스 발생기는 트레이스 캐시 내에 예외 명령어를 전혀 저장하지 않도록 구성될 수 있다. 다른 실시예에서, 트레이스 발생기는 예외 명령어를 1개 이상의 다른 명령어들과 다른 트레이스 캐시 엔트리에 저장하고, 상기 다른 트레이스 캐시 엔트리에 상기 예외 명령어와 함께 임의의 다른 명령어를 저장하지 않도록 구성된다. 트레이스 발생기는 명령어들을 적어도 부분적으로 디코드된 형태로 트레이스 캐시에 저장하도록 구성된다.

[0004] 상기 시스템은 또한 실행된 명령어들을 퇴거(retire)하도록 결합된 퇴거큐(retire queue)를 포함한다. 퇴거큐는, 상기 명령어와 동일한 트레이스 캐시 엔트리에 포함되는 모든 명령어들이 퇴거 준비가 될 때까지, 트레이스 캐시로부터 페치된 모든 명령어의 퇴거를 막도록 구성된다. 대안적으로, 퇴거큐는, 상기 명령어와 동일한 트레이스 캐시 엔트리 내의 동일한 활성 그룹(liveness group)에 포함되는 모든 명령어들이 퇴거 준비가 될 때까지, 트레이스 캐시로부터 페치된 모든 명령어의 퇴거를 막도록 구성될 수 있다.

[0005] 상기 시스템은, 퇴거큐가 명령어를 마지막으로 퇴거한 이후 경과된 사이클의 수를 모니터하도록 구성된 재시작 로직(restart logic)를 포함한다. 사이클의 수가 임계 수를 넘으면, 재시작 로직은 트레이스 캐시로부터 페치된 명령어들을 실행하는 프로세싱 파이프라인을 플러시(flush)한 다음 상기 명령어 캐시로부터의 실행을 재개하도록 구성된다.

[0006] 상기 시스템에 포함된 디스패치 유닛은 트레이스 캐시로부터 수신된 명령어들을 디스패치하도록 구성된다. 디스패치 유닛은, 트레이스 캐시로부터 수신된 명령어들의 그룹 내에서 재실행이 불가능한 명령어를 검출한 다음, 상기 재실행이 불가능한 명령어의 표시를 재시작 로직에 제공하도록 구성된다. 이 표시에 응답하여, 재시작 로직은 재실행이 불가능한 명령어의 실행 이전에 프로세싱 파이프라인을 플러시하고 명령어 캐시로부터 실행을 재개하도록 구성된다.

[0007] 디스패치 유닛은 명령어 캐시로부터 수신된 명령어들의 그룹 내에서 예외 명령어를 검출한 다음 상기 예외 명령어의 표시를 트레이스 발생기에 제공하도록 구성되며, 이에 의해 상기 트레이스 발생기가 예외 명령어를 검출하도록 한다. 트레이스 발생기는 또한 예외 명령어들의 1개 이상의 특성들(예를 들어, 오피코드, 어드레싱 모드 등)에 응답하여 예외 명령어들을 검출하도록 구성된다.

[0008] 일부 실시예에서, 본 발명의 방법은 트레이스 캐시 내의 트레이스 캐시 엔트리에 저장하기 위한 명령어들의 그룹을 수신하는 단계와; 상기 명령어들의 그룹에 포함되는 1개 이상의 명령어들을 상기 트레이스 캐시 엔트리에 저장하는 단계와; 상기 명령어들의 그룹 내에서 예외 명령어를 검출하는 단계와; 그리고 상기 검출에 응답하여, 상기 1개 이상의 명령어들을 갖는 상기 트레이스 캐시 엔트리 내에 상기 예외 명령어를 저장하지 않는 단계를 포함한다.

실 시 예

[0019] 도 1은 프로세서(100)의 일 실시예에 포함된 로직 컴포넌트(component)들의 블록 다이어그램이다. 프로세서(100)는 시스템 메모리(200)에 저장된 명령어들을 실행하도록 구성된다. 이들 명령어들 중 다수는 시스템 메모리(200)에 저장된 데이터를 오퍼레이팅(operate)하며, 상기 시스템 메모리(200)는 컴퓨터 시스템을 통해 물리적

으로 분산될 수 있고 하나 이상의 프로세서(100)들에 의해 액세스될 수 있다. 일부 실시예들에서, 상기 프로세서(100)는 x86 아키텍처와 호환가능하도록 설계된다.

[0020] 프로세서(100)는 명령어 캐시(106)와 데이터 캐시(128)를 포함한다. 프로세서(100)는 상기 시스템 메모리(200)에 연결된 프리페치 유닛(prefetch unit)(108)을 포함한다. 프리페치 유닛(108)은 명령어 캐시(106) 내에 저장을 위해 상기 시스템 메모리(200)로부터 명령어 코드를 프리페치한다. 일 실시예에서, 프리페치 유닛(108)은 상기 시스템 메모리(200)로부터 명령어 캐시(106)로 코드를 버스트(burst)하도록 구성된다. 프리페치 유닛(108)은 다양한 특정 코드 프리페치 기술 및 알고리즘을 이용한다. 프리페치 유닛(108)은 또한 명령어 캐시(106)로부터 명령어들을 그리고 트레이스 캐시(160)로부터 디스패치 유닛(104)으로 트레이스들을 페치한다. 트레이스 캐시(160)에서 소정의 명령어 어드레스 미스(miss)에 응답하여 명령어 캐시(106)로부터 명령어들이 또한 페치된다. 유사하게, 명령어 캐시(106)에서 소정의 어드레스 미스에 응답하여 시스템 메모리(200)로부터 명령어들이 페치된다.

[0021] 디스패치 유닛(104)은 명령어 캐시(106)로부터 명령어들을 수신하고 트레이스 캐시(160)로부터 디코딩된 및/또는 부분적으로 디코딩된 명령어들을 수신하도록 구성된다. 상기 디스패치 유닛(104)은 명령어 캐시(106)로부터 수신된 명령어들을 디코딩하는 디코드 유닛(140)을 포함한다. 상기 디스패치 유닛(104)은 또한 마이크로코딩된(microcoded) 명령어들을 처리할 때 사용하기 위한 마이크로코드 유닛을 포함한다.

[0022] 디스패치 유닛(dispatch unit)(104)은 스케줄러(들)(118)에 디코딩된 명령어들을 디스패치하도록 구성된다. 하나 이상의 스케줄러들(118)이 디스패치 유닛(104)으로부터 디스패치된 명령어들을 수신하고 하나 이상의 실행 코어들(124)에 명령어들을 발행하도록 연결된다. 실행 코어(들)(124)는 데이터 캐시(128)로의 액세스를 수행하도록 구성된 로드 저장 유닛(126)을 포함한다. 실행 코어(들)(124)에 의해 생성된 결과들이 결과 버스(130)로 출력된다. 이러한 결과들은 후속으로 발행되는 명령어들에 대한 오퍼랜드(operand) 값으로서 사용되고 및/또는 레지스터 파일(116)에 저장된다. 퇴거큐(retire queue)(102)는 스케줄러(들)(118) 및 디스패치 유닛(104)에 연결된다. 상기 퇴거큐는 발행된 각 명령어가 언제 퇴거 될지를 결정하도록 구성된다.

[0023] 명령어 캐시(106)는 디스패치 유닛(104)에 의한 명령어들의 수령 전에 명령어들을 임시로 저장한다. 프리페치 유닛(108)을 통해 시스템 메모리(200)로부터 코드를 프리페칭 함으로써 명령어 코드가 명령어 캐시(106)에 제공된다. 명령어 캐시(106)는 다양한 구성으로 구현된다(예컨대, 세트-어소시에이티브(set-associative), 풀리-어소시에이티브(fully-associative), 또는 직접 매핑(direct-mapping)).

[0024] 본 명세서에서 사용되는 바와 같이, 상기 용어 명령어는 일반적으로 디코딩되지 않은(non-decoded), 부분적으로 디코딩된, 및 완전 디코딩된(fully decoded) 명령어들을 일컫는다. 부분적으로 및 완전 디코딩된 명령어들은 또한 오퍼레이션(operation)들로 지칭된다. 예를 들어, 디코딩되지 않은 명령어는 실행 코어(들)(124)에서 직접 실행될 수 있는 하나 이상의 컴포턴트 오퍼레이션들로 디코드 유닛(140)에 의해 디코딩되는 것으로 설명된다. 디스패치 유닛(104)은 오퍼랜드 어드레스 정보, 즉시 데이터 및/또는 변위 데이터뿐만 아니라 실행 코어(들)(124)에 의해 실행가능한 비트 인코딩된 오퍼레이션들(bit-encoded operation)을 포함하는 신호들을 출력한다. 레지스터의 업데이트를 포함하는 명령어를 수신하면, 상기 디스패치 유닛(104)은 추론 레지스터 상태(speculative register state)를 저장하기 위해 레지스터 파일(116) 내에 레지스터 위치를 예약(reserve)한다(대안적인 실시예에서, 각 레지스터에 대한 하나 이상의 추론 레지스터 상태들을 저장하기 위해 재배열(reorder) 버퍼가 사용된다). 레지스터 재명명을 용이하게 하기 위해 레지스터 맵은 소스 및 목적 오퍼랜드의 로직 레지스터 명칭들을 물리적 레지스터 명칭들로 변환한다. 이러한 레지스터 맵은 레지스터 파일(116) 내의 어떤 레지스터가 현재 할당되었는지 할당되지 않았는지를 추적한다.

[0025] 명령어들이 디스패치 유닛(104)에 의해 처리되는 때, 만약 필요한 오퍼랜드가 레지스터 위치라면, 레지스터 어드레스 정보는 레지스터 맵이나 재배열 버퍼(reorder buffer)에 라우팅된다. 예를 들면, x86 아키텍처에서 8개의 32-비트 로직 레지스터(예컨대, EAX, EBX, ECX, EDX, EBP, ESI, EDI 및 ESP)가 존재한다. 물리적 레지스터 파일(116)(또는 재배열 버퍼)은, 비순차 실행(out of order execution)을 허용하는, 상기 로직 레지스터들의 컨텐츠를 변경하는 결과들에 대한 저장부를 포함한다. 레지스터 파일(116) 내의 물리적 레지스터는 상기 로직 레지스터들 중 하나의 컨텐츠를 수정하는 각 명령어의 결과를 저장하도록 할당된다. 따라서, 특정 프로그램의 실행 동안에 수많은 포인트에서, 레지스터 파일(116)(또는, 대안적인 실시예에서 재배열 버퍼)은 소정의 로직 레지스터의 추론적으로 실행되는 컨텐츠를 포함하는 하나 이상의 레지스터들을 구비할 수 있다.

[0026] 레지스터 맵은 물리적 레지스터를 명령어에 대한 목적 오퍼랜드로서 지정된 특정 로직 레지스터에 지정한다. 레지스터 파일(116)은 소정의 명령어에서 소스 오퍼랜드로서 특정된 로직 레지스터에 지정된 하나 이상의 기

(previously) 할당된 물리적 레지스터들을 포함한다. 레지스터 맵은 상기 로직 레지스터에 가장 최근에 지정된 물리적 레지스터에 대한 태그를 제공한다. 상기 태그는 레지스터 파일(116) 내의 오퍼랜드의 데이터 값에 액세스하거나 상기 결과 버스(130) 상의 결과 포워딩을 통해 상기 데이터 값을 수신하는데 사용된다. 만약 오퍼랜드가 메모리 위치에 대응한다면, 상기 오퍼랜드 값은 로드 저장 유닛(126)을 통해 결과 버스(130) 상(레지스터 파일(116) 내의 저장 및/또는 결과 포워딩을 위한)에 제공된다. 오퍼랜드 데이터 값들은, 명령어가 하나 이상의 스케줄러(들)(118)에 의해 발행된 때, 실행 코어(들)(124)에 제공된다. 주목할 사항으로, 대안적인 실시예에서, 명령어가 디스패치된 때, (명령어가 발행된 때, 대응하는 실행 코어(124)에 제공되는 대신에) 오퍼랜드 값들이 대응하는 스케줄러(118)에 제공된다는 점이다.

[0027] 도 1의 프로세서(100)는 비순차 실행을 지원한다. 퇴거큐(102)(또는 대안적으로 재배열 버퍼)는 레지스터 읽기(read) 및 기록(write) 오퍼레이션들의 본래의(original) 프로그램 시퀀스를 기억하고, 추론 명령어 실행 및 분기 오예측 복구(branch misprediction recovery)을 허용하고, 그리고 정확한 예외(precise exception)를 촉진한다. 수많은 실시예들에서, 퇴거큐(102)는 재배열 버퍼와 유사하게 기능한다. 그러나, 전형적인 재배열 버퍼와 달리, 퇴거큐(102)는 임의의 데이터 값 저장부를 제공하지 않는다. 대안적인 실시예들에서, 퇴거큐(102)는 재배열 버퍼와 더욱 유사하게 기능하고 또한 추론 레지스터 상태들에 대한 데이터 값 저장부를 제공함으로써 레지스터 재명명을 지원한다. 일부 실시예들에서, 퇴거큐(102)는 FIFO(first-in-first-out) 구성으로 구현되며, 여기서 오퍼레이션들은 이들이 유효화(validate)될 때 버퍼의 "바닥"으로 이동하여, 상기 큐의 "상부"에 새로운 엔트리들을 위한 공간을 만든다. 명령어들이 퇴거될 때, 퇴거큐(102)는 추론 레지스터 상태들을 저장하는데 더 이상 필요하지 않은 레지스터들을 레지스터 파일(116)에서 할당해제(deallocate)하고, 어떤 레지스터들이 현재 비었는지(free)를 나타내는 신호들을 레지스터 맵에 제공한다. 추론 레지스터 상태들을 생성시켰던 명령어들이 유효화될 때까지, 레지스터 파일(116)(또는 대안적인 실시예들에서 재배열 버퍼) 내에 상기 추론 레지스터 상태들을 유지함으로써, 분기예측이 올바르지 않은 경우에, 오예측된 경로를 따라 추론적으로 실행된 명령어들의 결과들이 상기 레지스터 파일(116)에서 무효화될 수 있다.

[0028] 디스패치 유닛(104)의 출력에서 제공되는 비트 인코딩된 오퍼레이션들 및 즉시 데이터는 하나 이상의 스케줄러들(118)에 라우팅된다. 주목할 사항으로, 본원에서 사용되는 바와 같이 스케줄러는 명령어들의 실행이 준비된 때를 검출하고, 준비된 명령어들을 하나 이상의 실행 유닛들에 발행하는 디바이스이다. 예를 들면, 예약 스테이션이 일 유형의 스케줄러이다. 각 스케줄러(118)는 실행 코어(124)로의 발행 대기 중인 수 개의 명령어들에 대한 명령어 정보(예컨대, 오퍼랜드 값, 오퍼랜드 태그 및/또는 즉시 데이터뿐 아니라 비트 인코딩된 실행 비트들)를 홀딩(holding)할 수 있다. 일부 실시예에서, 각 스케줄러(118)는 오퍼랜드 값 저장부를 제공하지 않는다. 대신에, 오퍼랜드 값들이 언제 실행 코어(들)(124)에 의해 읽기 가능하게될 수 있을지를 결정하기 위해(레지스터 파일(116) 또는 결과 버스(130)로부터), 각 스케줄러는 발행된 명령어들 및 레지스터 파일(116)에서 이용가능한 결과들을 모니터한다. 일부 실시예에서, 각 스케줄러(118)는 전용 기능 유닛(dedicated functional unit)(예컨대, 정수 연산 유닛(integer unit), 부동 소수점 유닛(floating point unit), 로드 저장 유닛 등) 및/또는 실행 코어(124)와 결합되어 있다. 다른 실시예들에서, 단일 스케줄러(118)는 하나 보다 많은 수령자에 명령어들을 발행한다.

[0029] 스케줄러(118)는 실행 코어(들)(124)에 의해 실행될 오퍼레이션 정보를 임시로 저장하도록 제공된다. 명령어들은 실행시에 적시에 이용가능하게 된 필요한 모든 오퍼랜드 값들에 응답하여 실행하기 위해 실행 코어(들)(124)에 발행된다. 따라서, 명령어들이 실행되는 순서는 본래의 프로그램 명령어 시퀀스의 순서와 동일하지 않을 수 있다.

[0030] 일 실시예에서, 실행 코어(들)(124) 각각은 덧셈 및 뺄셈의 정수 계산 오퍼레이션들 뿐 아니라 이동, 회전, 로지컬 오퍼레이션 및 분기 오퍼레이션들을 수행하도록 구성되는 정수 실행 유닛과 같은 기능 유닛들을 포함한다. 부동 소수점 유닛이 또한 부동 소수점 오퍼레이션들을 제공하도록 포함된다. 하나 이상의 실행 코어(들)(124)이 로드 저장 유닛(126)에 의해 수행될 로드 및 저장 메모리 오퍼레이션들에 대한 어드레스 생성을 수행하도록 구성된다.

[0031] 상기 실행 코어(들)(124)은 또한 조건부 분기 명령어들의 실행에 관한 정보를 분기 예측 유닛(132)에 제공한다. 만약 상기 실행 코어(124)로부터의 정보가 분기 예측이 올바르지 않음을 나타낸다면, 상기 분기 예측 유닛(132)은 상기 명령어 프로세싱 파이프라인에 입력된 오예측 분기에 후속하는 명령어들을 플러시(flush)하고 프리페치 유닛(108)의 방향을 재설정한다(redirect). 그 다음, 상기 방향이 재설정된 프리페치 유닛(108)은 명령어 캐시(106), 트레이스 캐시(160) 및/또는 시스템 메모리(614)로부터 올바른 명령어들 세트를 페치하기 시작한다. 이러한 상황에서, 상기 오예측 분기 명령어 후에 일어난 본래 프로그램 시퀀스에서의 명령어들의 결과들은, 추

론적으로 실행되었고 로드 저장 유닛(126) 및/또는 레지스터 파일(116)에 임시로 저장되었던 결과들을 포함하여 폐기된다.

[0032] 만약 레지스터 값이 업데이트 된다면, 실행 코어(들)(124) 내의 컴포넌트들에 의해 생성된 결과들은 레지스터 파일(116)로의 결과 버스(130) 상에 출력된다. 만약 메모리 위치의 컨텐츠들이 변경된다면, 실행 코어(들)(124) 내에서 생성된 결과들은 로드 저장 유닛(126)으로 제공된다.

[0033] 로드 저장 유닛(126)은 실행 코어(들)(124)와 데이터 캐시(128) 사이에 인터페이스를 제공한다. 일 실시예에서, 로드 저장 유닛(126)은 펜딩(pending) 로드 또는 저장을 위한 데이터와 어드레스 정보에 대한 수개의 저장 위치를 구비한 로드 저장 버퍼로 구성될 수 있다. 또한, 로드 저장 유닛(126)은 데이터 코히어런시(coherency)가 유지됨을 보장하기 위해 펜딩 저장 명령어들에 대한 로드 명령어들의 종속성 검사를 수행한다.

[0034] 데이터 캐시(128)는 로드 저장 유닛(126)과 시스템 메모리(200) 사이에 전송되는 데이터를 임시로 저장하도록 제공된 캐시 메모리이다. 전술한 명령어 캐시(106)와 같이, 데이터 캐시(128)는 세트 어소시에이티브 구성을 포함하는 다양한 특유의 메모리 구성으로 구현될 수 있다. 추가로, 데이터 캐시(106) 및 명령어 캐시(128)는 일부 실시예에서 통합된 캐시로 구현될 수 있다.

[0035] 퇴거큐(102)는 트레이스 발생기(170)에 프로그램 트레이스들을 식별하는 신호들을 제공한다. 트레이스 발생기(170)는 필유닛(fill unit)으로서 또한 기술된다. 트레이스 발생기(170)는 퇴거큐(102)에 의해 식별된 트레이스들을 트레이스 캐시(160)에 저장한다. 각 트레이스는 수 개의 서로 다른 기본 블록들의 부분으로서 프로그램 순서와는 다른 순서로 저장된 명령어들을 포함한다. 기본 블록은 모두 실행되거나 모두 실행되지 않을 명령어들의 세트이다. 즉, 기본 블록에서 임의의 명령어가 실행된다면, 상기 기본 블록의 다른 모든 명령어들도 또한 실행될 것이다. 기본 블록의 예는 분기 명령어 후에 바로 시작하고 다른 분기 명령으로 종료되는 명령어들의 세트이다. 트레이스 캐시(160)에 저장된 트레이스들은 일부 실시예들에서 디코딩된 또는 부분적으로 디코딩된 명령어들을 포함한다. 본 명세서에서 사용되는 바와 같이, "트레이스"는 트레이스 캐시(160)의 단일 트레이스 캐시 엔트리(예컨대, 단일 로우(row) 또는 라인) 내에 저장된 명령어들의 그룹이다. 엔트리 내의 모든 정보는 동시에 액세스된다(예컨대, 트레이스 캐시 읽기 사이클의 표명(assertion)에 응답하여). 대안적인 실시예들에서, 트레이스 발생기(170)가 프로세서의 전방-후방(front-end)(예컨대, 상기 디스패치 유닛의 전 또는 후)에 연결되고 페치되는 및/또는 디스패치되는 명령어들에 응답하여 트레이스들을 발생하도록 구성된다는 점을 주목하여야 한다.

[0036] 프리페치 유닛(108)은 트레이스 캐시(160)로부터 디스패치 유닛(104)으로 명령어들을 페치한다. 트레이스 캐시(160)로부터 명령어들을 페칭하는 것은 명령어 캐시(106)로부터 페칭하는 것에 비해 분기 경계들에 걸쳐 개선된 페칭 성능을 제공한다. 예를 들어, 만약 분기가 예측 테이큰(predicted taken)된다면, 상기 예측 테이큰된 경로의 명령어들은 트레이스 캐시(160) 내의 상기 분기와 동일한 트레이스에 이미 저장되어 있다. 또한, 명령어들이 적어도 부분적으로 디코딩된 형태로 트레이스 캐시(160)에 저장된 실시예에서, 상기 트레이스 캐시로부터 명령어들을 페칭하는 것은 상기 디코드 유닛(140)이 적어도 부분적으로 바이패스(bypass)되도록하여, 결과적으로 상기 캐시된 명령어들에 대한 디스패치 사이클들의 수를 줄인다. 따라서, 상기 트레이스 캐시(160)는 트레이스들이 한번 넘게 실행될 때 수회의 반복 실행들 동안에 상기 캐시된 명령어들을 부분적으로 (또는 완전히) 디코딩하는데 드는 시간이 감각(amortize)되도록 한다.

[0037] 주목할 사항으로 프로세서(100)는 본 명세서에서 도시된 것에 추가하여 다른 수많은 컴포넌트를 포함하고 및/또는 이들에 연결될 수 있다. 예를 들면, 추가의 캐시 레벨들이 프로세서(100)와 시스템 메모리(200) 사이에(프로세서(100)의 내부 및/또는 외부에) 포함될 수 있다. 유사하게, 프로세서(100)는 일부 실시예에서 시스템 메모리(200)를 제어하도록 구성된 통합된 메모리 제어기를 포함한다. 추가로, 로직 컴포넌트들 사이의 상호접속부(interconnect)들은 실시예들에 따라 가변한다.

[0038] **트레이스 캐시**

[0039] 도 2는 일 실시예에 따른 예시적인 트레이스 캐시(160)를 나타낸다. 트레이스 캐시(160)는 복수개의 엔트리들(162)을 포함한다. 각 엔트리(162)는 복수개의 명령어들(165)을 포함하는 트레이스를 저장한다. 트레이스 내의 명령어들(165)은 프로그램 순서로 저장되지 않는다. 예를 들어, 엔트리(162)는 분기 명령어와, 그리고 (프로그램 순서에 있어서 분기 다음에 오는 명령어와는 대조적으로) 분기가 테이큰(taken)될 때 그 분기의 목적인 명령어 모두를 저장한다. 일부 실시예에서, 각 트레이스 캐시 엔트리(162) 내의 명령어들(165)은 적어도 부분적으로 디코딩된 형태로 저장된다. 여기에서 이용되는 용어 "트레이스"는 단일 트레이스 캐시 엔트리(162)에 저장되는

1개 이상의 명령어들의 그룹을 나타내는 데에 이용된다.

[0040] 다시 도 1을 참조하여, 주목할 사항으로서, 트레이스 캐시(160)로부터 페치되는 명령어 스트림의 프로세싱은 명령어 캐시(106)로부터 페치되는 명령어 스트림의 프로세싱과 다르다. 명령어 캐시(106)로부터의 명령어 스트림은, 명령어들이 1개 이상의 컴포넌트 오퍼레이션들로 디코딩된 이후조차도, 명령어 경계(boundary)를 식별하는 정보를 포함한다. 퇴거큐(102)가 이러한 경계 정보를 이용하여, 특정 명령어에 대응하는 오퍼레이션(들)이 언제 퇴거 될지를 결정한다. 전형적으로, 퇴거큐(102)는 본래의 프로그램 순서로 오퍼레이션들을 퇴거하도록 동작한다. 동일한 명령어에 대응하는 오퍼레이션(들)은 동시에 퇴거된다.

[0041] 대조적으로, 트레이스 캐시(160)로부터의 명령어 스트림은, 트레이스 캐시(160)에 저장하기 이전에 명령어들을 디코드 및/또는 변경한(예를 들어, 트레이스 캐시(160) 내에서의 저장 효율을 증가시키기 위해 명령어들을 결합함으로써) 경우에는, 명령어 경계들을 식별하는 정보를 포함하지 않는다. 결과적으로, 퇴거큐(102)는 이러한 명령어 스트림에 포함되는 명령어들을 커서 입도(courser granularity)로 퇴거해야 한다. 예를 들어, 임의의 실시예에서, 퇴거큐(102)는 트레이스 캐시 내의 모든 명령어들이 퇴거 준비가 될 때까지 그 트레이스에 포함되는 임의의 명령어들을 퇴거하지 않는다.

[0042] 일부 트레이스 캐시 기반 프로세서에서 발생하는 하나의 문제는 특정 명령어들(여기에서는 "예외" 명령어들이라 불린다)이, 이러한 명령어들이 퇴거될 수 있는 때에 영향을 미치는 특별한 처리를 요구한다는 것이다. 예외 명령어와 동일한 트레이스 내의 (프로그램 순서에 있어서 보다 일찍 발생하는 명령어들을 포함하는) 다른 명령어들의 퇴거는 그 예외 명령어가 퇴거 준비가 되었느냐에 의존하기 때문에, 이러한 예외 명령어는 그 트레이스 내의 다른 명령어들의 퇴거를 차단한다. 임의의 상황에서, 예외 명령어의 비 퇴거(non-retirement)가 동일 트레이스에 포함되는 보다 오래된 오퍼레이션들의 퇴거를 차단하는 경우, 프로세서는 교착상태(deadlock)가 된다. 또한, 트레이스 캐시로부터 제공되는 명령어 경계 정보의 결여로 인해, 예외 명령어가 동일 트레이스 내의 다른 명령어들의 퇴거를 차단하는 경우, 그 트레이스 내의 퇴거불가능한 모든 명령어들 (및 프로그램 순서에 있어서 이미 디스패치된 모든 더욱 새로운(younger) 명령어들)이 명령어 캐시로부터 다시 페치되어 재실행되어야 한다.

[0043] 예외 명령어들은 예외 또는 인터럽트를 생성하는 명령어들을 포함한다. 예외 명령어는 비-예외(non-exceptional) 명령어와 비교하여 완료하는 데에 상당히 더 오래 걸릴 가능성이 있다. 예를 들어, 실행 코어(124)는 전형적인 비-예외 명령어를 1 내지 3 실행 사이클(one to three execution cycle)에서 완료한다. 대조적으로, 예외 명령어는 실행할 많은 실행 사이클들 같이 10배(또는 그 이상)를 필요로 하는 인터럽트를 생성한다. 적어도 일부 예외 명령어들을 이용하게 되면, (프로그램 순서에 있어서) 특정의 예외 명령어보다 오래된 모든 명령어들이 그 예외 명령어와 독립적으로 퇴거될 수 있는 것이 유익하거나, 또는 심지어 요구된다. 상기 설명한 바와 같이, 트레이스에 있어서 명령어 경계 정보의 결여는 동일 트레이스 내의 다른 명령어들이 독립적으로 퇴거될 수 있는 능력을 감소시키거나 또는 없앤다. 이에 따라, 트레이스 내에 다른 명령어들과 함께 예외 명령어를 포함시키게 되면, 궁극적으로, 경계 정보를 얻기 위해서는 그 트레이스 내의 모든 명령어들이 명령어 캐시로부터 다시 페치되어야 한다.

[0044] 다른 명령어들의 퇴거를 차단하고 및/또는 다른 명령어들이 폐기되고, 다시 페치되고, 재실행되게 하는 예외 명령어의 가능성을 줄이기 위해, 트레이스 발생기(170)는 예외 명령어들을 검출한 다음, 이러한 명령어들을 모든 다른 명령어들을 포함하는 트레이스 내에 저장하지 않도록 구성된다. 일부 실시예에서, 트레이스 발생기(170)는 검출된 예외 명령어를 트레이스 내에 저장하며, 상기 트레이스는 그 예외 명령어만을 포함한다. 비록 이러한 구성은 트레이스 캐시 저장 효율을 감소시키기는 하지만, 명령어 캐시(106)와 트레이스 캐시(160) 간에서 페칭을 스위치(switch)하는 대신에, 트레이스 캐시로부터 페칭을 계속할 수 있게 한다. 다른 실시예들에서, 트레이스 발생기(170)는 검출된 예외 명령어들을 트레이스 캐시(160)에 전혀 저장하지 않음으로써, 예외 명령어들이 명령어 캐시(106)로부터 항상 페치되게 한다. 트레이스 발생기(170)는 트레이스 캐시 내의 트레이스 캐시 엔트리에 명령어들의 그룹을 저장하는 수단의 예이다.

[0045] 트레이스 발생기(17)는 특정 오피코드, 오퍼랜드들, 어드레싱 모드들 및 다른 특성들에 대한 트레이스 캐시에서 저장을 위한 각 후보(candidate) 명령어를 검사함으로써 예외 명령어들을 검출한다. 일 실시예에서, 잠재적으로 예외 명령어들의 특성들을 식별하는 것이 각 트레이스 발생기(170)에 프로그램된다. 이러한 실시예들에서, 트레이스 발생기(170)는 특정 유형의 명령어를 예외 명령어로서 식별하도록 구성되며, 이는 비록 이러한 오퍼레이션들 유형이 특정 환경하에서 특별한 처리 없이 실행가능할지라도 그러하다는 점을 주목해야 한다. 일부 실시예에서, 디코드 유닛(140)은 명령어 디코드 동안에 예외 명령어들을 식별하는 태그들을 생성한다. 각 태그는 상기 태그에 의해 예외로서 식별되는 디코드된 명령어와 함께 상기 프로세서(100)를 통과한다.

[0046] 따라서, 디코딩된 명령어들이 예외적인지를 독립적으로 결정하는 대신(또는 상기 결정에 부가하여), 트레이스 발생기(170)는 예외 명령어(exceptional instruction)를 식별하기 위해 디코드 유닛(140)에 의해 생성된 태그들에 의존할 수 있다. 따라서, 트레이스 발생기(170) 및 디코드 유닛(140) 모두는 명령어들 그룹 내의 예외 명령어를 검출하기 위한 예시적인 수단들이다.

[0047] 트레이스 캐시(160)에 저장되는 명령어들 스트림 내의 예외 명령어의 검출에 응답하여, 트레이스 발생기(170)는 (트레이스 내에 부가적인 명령어들을 저장하기 위한 공간이 이용가능하더라도) 현재 트레이스로의 명령어들의 저장을 종료할 수 있다. 만일 트레이스 발생기(170)가 트레이스 캐시(160) 내의 자립형 트레이스에 예외 명령어들을 저장하도록 구성되어 있다면, 트레이스 발생기는 예외 명령어를 저장하기 위해 새로운 트레이스 캐시 엔트리를 할당한다. 트레이스 발생기(170)는 예외명령의 직후에 발생하는 다음의 명령을 프로그램 순서에서 또 다른 트레이스 캐시 엔트리에 저장할 수 있다. 만일 대신에 트레이스 발생기(170)가 트레이스 캐시(160)로부터 완전히 예외 명령어들을 배제하도록 구성된다면, 트레이스 발생기(170)는 예외 명령어를 검출하자마자 현재 트레이스를 단순히 종료하고(어떠한 더 오래된 명령어들도 현재 트레이스에 이미 저장되어 있지 않은 경우가 아닌한), 그리고 프로그램 순서에서 상기 트레이스 캐시에 저장하기 위한 후보인 다음 비-예외 명령어를 후속하는 트레이스에 저장하기 시작한다. 유의할 것은, 트레이스 발생기(170)는 예외 명령어가 페치할 다음 명령어임을 나타내기 위해 현재 트레이스 내에 흐름 정보를 설정할 수 있다는 것이다. 따라서, 트레이스가 페치되는 것에 응답하여, 프리페치 유닛(170)은 명령어 캐시(106)로부터의 흐름 정보에서 식별된 명령어를 프리페치하도록 구성될 수 있다. 따라서, 실행 스트림은 트레이스 캐시(160)로부터 명령어 캐시(106)로 스위칭될 것인바, 이는 트레이스 캐시(160)로부터 예외 명령어의 배제에 기인한 것이다.

[0048] 일부 예외 명령어들은 트레이스 발생기(170)에 의해 검출될 수 없다. 예를 들어, 일부 명령어들은 이러한 명령어들이 실행될 때마다 상이하게 동작할 수 있다. 이러한 명령어들의 성질 (예를 들어 이러한 명령어들은 거의 예외적이지 않음) 때문에, 트레이스 발생기(170)는 이러한 명령어들을 자동으로 예외적인 것으로 식별하지 않도록 구성될 수 있다. 그 결과, 이러한 예외 명령어들은 트레이스 발생기(170)에 의해 다른 명령어들을 갖는 트레이스 내에 저장될 수 있다. 이러한 예외 명령어들의 포함이 동일한 트레이스 내에 다른 명령어들의 퇴거(retirement)의 차단에 기인한 문제점들을 유발하는 것을 방지하기 위해, 재시작 로직(190)은 퇴거큐(102)가 임의의 오퍼레이션들을 마지막으로 퇴거시키고 나서 얼마나 많은 클록 사이클들이 경과했는지를 모니터링할 수 있다. 만일 어떤 오퍼레이션들도 소정 수의 클록 사이클 내에서 퇴거되지 않았다면, 재시작 로직(190)은 명령어 캐시(106)로부터 프로세서(100)의 실행을 재시작한다. 일 실시예에서, 재시작 로직(190)은 프로세싱 파이프라인을 플러시하고 명령어들을 다시 페치함으로써 프로세서(100) 내의 가장 오래된 미결의(outstanding) 비-퇴거 명령어부터 실행을 재시작할 수 있다. 명령어들이 명령어 캐시(106)로부터 페치될 때, 명령어 경계들에 대한 정보가 퇴거큐(102)에 이용가능할 것이다. 따라서, 재시작 로직(190)에 의해 검출된 비-퇴거가 트레이스의 퇴거를 차단한 예외 명령어에 의해 유발된다면, 명령어 캐시(106)로부터 명령어들을 페치하는 것은 퇴거큐(102)가 다른 명령어들의 퇴거를 불필요하게 차단하지 않고서도 문제되는 예외 명령어를 독립적으로 퇴거할 수 있도록 할 수 있다.

[0049] 특정 명령어들이 재-실행가능한 것이 아닐 수 있고, 따라서 이러한 명령어들은, 만일 이 명령어들이 트레이스 캐시(160)로부터 실행되고, 동일한 트레이스에 포함된 예외 명령어에 의해 퇴거가 차단되고, 그 후에 재시작 논리(190)의 오퍼레이션에 기인하여 재-실행된다면, 부정확할 수 있다. 예를 들어, 특정 로드들(예를 들어 일부 I/O 로드들)의 실행은 판독되는 값을 수정할 수 있다. 이러한 명령어들이 재-실행되는 경우 발생할 수 있는 문제들을 방지하기 위해, 트레이스 발생기(170)는 이러한 명령어들을 다른 명령어들을 갖는 트레이스들 내에 저장되지 않는 다른 부류의 예외 명령어들로서 식별하도록 구성될 수 있다. 따라서, 트레이스 발생기(170)는 트레이스 캐시(160)로부터 이러한 예외 명령어들을 배제하거나, 이러한 예외 명령어들을 (예를 들어 예외 명령어들에 대해 개략적으로 전술한 바와 같은) 어떠한 다른 명령어들도 포함하지 않는 자립형 트레이스들에 저장할 수 있다.

[0050] 특정한 다른 예외 명령어들과 마찬가지로, 일부 재-실행가능하지 않은 명령어들은 트레이스 발생기(170)에 의해 검출되지 않을 수 있다. 예를 들어, 로드는 하나의 프로그램 반복 동안 시스템 메모리(200) 어드레스에 액세스하고 다음 반복 동안 I/O 레지스터에 액세스할 수 있다. 첫 번째 반복 후에, 로드는 트레이스에 저장될 수 있다. 다음 반복 동안, 로드는 I/O 레지스터에 대한 판독의 성질에 기인하여 재-실행가능하지 않을 수 있다. 따라서, 만일 로드와 동일한 트레이스 내의 다른 명령어가 예외적이라면 (예를 들어, 다른 명령어가 페이지 오류를 생성한다면), 트레이스의 퇴거는 재시작 로직(190)을 트리거링하기에 충분히 오랫동안 지연될 수 있다. 그러나 이는 로드가 수용불가하게 재-실행되도록 할 수 있다. 이러한 상황을 방지하기 위해, 디스패치 유닛(104) 또

는 프로세서(100) 내의 다른 컴포넌트는 로드가 실제로 실행되기 전에 일부 지점에서 재-실행가능하지 않은 것으로 상기 로드를 식별하도록 구성될 수 있다. 트레이스 캐시(106)로부터 페치된 명령어들 그룹 내의 재-실행가능하지 않은 명령어의 검출에 응답하여, 재시작 로직(190)은 (재-실행가능하지 않은 명령어의 실행 이전에) 프로세서 파이프라인이 플러시되도록 하고 명령어 캐시(106)로부터 실행을 재시작하도록 한다.

[0051] **예시적인 트레이스**

[0052] 도 3A는 적어도 부분적으로 디코딩된 명령어들을 저장하도록 구성된 트레이스 캐시 엔트리(162)의 일 실시예를 도시한다. 도 3A의 실시예에서, 트레이스 캐시 엔트리(162)는 디코딩된 및/또는 부분적으로 디코딩된 명령어들 그룹에 포함된 8개까지의 컴포넌트 오퍼레이션들을 오퍼레이션 저장 유닛들(166A - 166H)에 저장할 수 있다. 유의할 것은, 다른 실시예들은 8개 이하 또는 8개 이상의 부가적인 오퍼레이션 저장 유닛들(166)을 포함할 수 있고, 각 트레이스 캐시 엔트리(162) 내에 상이한 개수의 오퍼레이션들을 저장할 수 있다는 것이다.

[0053] 트레이스 캐시 엔트리(162) 내의 특정한 오퍼레이션 저장 유닛들(166)은 특정한 유형의 오퍼레이션들을 위해 예약될 수 있다. 예를 들어, 일 실시예에서, 오퍼레이션 저장 유닛들(166A-166D)이 메모리 오퍼레이션들을 저장하기 위해 사용될 수 있다. 마찬가지로, 오퍼레이션 저장 유닛들(166E-166H)이 데이터 오퍼레이션들을 저장하기 위해 사용될 수 있다. 유의할 것은, 다른 실시예들은 특정한 유형의 오퍼레이션들을 특정한 오퍼레이션 저장 유닛들에 서로 다르게 연관시킬 수 있다는 것이다(또는 전혀 연관시키지 않을 수 있다) .

[0054] 다수의 오퍼레이션 저장 유닛들(166)에 부가하여, 각 트레이스 캐시 엔트리(162)는 또한 식별 태그(164) 및 흐름 제어 정보(168)를 포함한다. 태그(164)는 명령어 캐시(106) 내의 태그와 유사하여, 프리페치 유닛(108)이 소정의 오퍼레이션이 트레이스 캐시(160)에서 히트하는지 또는 미스하는지를 결정할 수 있도록 한다. 예를 들어, 태그(164)는 트레이스 캐시 엔트리 내에 명령어를 식별하는 어드레스 비트들 모두 또는 일부를 포함할 수 있다(예를 들어, 태그는 트레이스 내에 저장된, 프로그램 순서에서, 가장 빠른 명령어의 어드레스를 포함할 수 있다). 일부 실시예에서, 태그는 각 명령어(또는 이하 더 상세히 설명되는, 적어도 각 활성(liveness) 그룹 내의 가장 이른 명령어)의 어드레스가 트레이스에 저장된 정보를 사용하여 독립적으로 식별될 수 있도록 충분한 정보를 포함할 수 있다. 다른 실시예에서, 트레이스 내의 가장 이른 명령어의 어드레스만이 이용가능할 수 있다.

[0055] 설명된 실시예에서, 각 트레이스는 2개까지의 분기 명령어들을 포함할 수 있다. 다른 실시예는 각 트레이스 내의 더 적거나 추가적인 분기 명령어들을 포함할 수 있다. 흐름 제어 정보(168)는 트레이스 내에 포함된 각 분기 명령어에 대한 라벨을 포함할 수 있다. 라벨은 각 분기의 레졸루션(resolution)(테이큰, 낫테이큰)에 따라 제어가 분기할 어드레스를 식별하는 표시일 수 있다. 따라서, 흐름 제어 정보(168)의 각 항목은 특정한 분기 오퍼레이션과 연관될 수 있다. 예를 들어, 일 실시예에서, 트레이스 내의 하나의 흐름 제어 정보 저장 위치는 트레이스 내의 제 1 분기 오퍼레이션과 연관될 수 있고 다른 흐름 제어 정보 저장 위치는 트레이스 내의 제 2 분기와 연관될 수 있다. 대안적으로, 상기 흐름 제어 정보는 상기 흐름 제어 정보가 관련된 분기 오퍼레이션을 식별하는 태그들 또는 다른 정보를 포함한다. 또 다른 실시예들에서, 어떤 흐름 제어 정보가 분기 오퍼레이션에 대응하는 지를 식별하는 분기 예측 및/또는 정보가 오퍼레이션 저장부(166) 내의 분기 오퍼레이션과 함께 저장된다.

[0056] **트레이스들 내에 증가된 퇴거 입도(retirement granularity) 제공**

[0057] 전술한 바와 같이, 트레이스 내에 명령어 경계들에 관한 정보가 제한되어 있다. 예를 들어, 만약 명령어들이 트레이스 내에 저장되기 전에 명령어들의 컴포넌트 오퍼레이션들로 부분적으로 디코딩된다면, 상기 트레이스 내에 서로 다른 명령어들을 묘사하는 어떠한 정보도 상기 트레이스에 포함되지 않는다. 게다가, 만약 디코딩된 후라면, 서로 다른 명령어들의 컴포넌트 오퍼레이션들이 결합(combine)되고, 재배열되고, 및/또는 수정되어, 명령어 경계들을 식별하는 것은 더욱 어려워진다.

[0058] 트레이스 내에 증가된 명령어 경계 정보를 제공하기 위해서, 트레이스 내에서 디코딩된 명령어들이 서로 다른 활성 그룹들로 세분된다. 트레이스 발생기(170)는 특정 트레이스 캐시 엔트리에 저장된 각 디코딩된 명령어에 상기 트레이스 내의 특정 활성 그룹에 속하는 것으로 태그를 붙인다. 각 활성 그룹 내에서 디코딩된 명령어들의 명령어 경계들에 관한 정보는 이용불가능하다. 그러나, 상기 활성 그룹들 스스로가 디코딩된 명령어들이 활성 그룹 경계들상에서 퇴거되도록 생성될 수 있다.

[0059] 트레이스 내의 각 오퍼레이션(166)은 관련 활성 표시를 갖는다. 각 오퍼레이션의 활성 표시는 그의 관련 오퍼레이션과 함께 오퍼레이션 저장 유닛들(166) 중 하나에 저장된다. 활성 표시들은 각 오퍼레이션이 속하는 활성 그룹을 식별한다. 각 활성 그룹은 동일한 기본 블럭의 일부분인 트레이스 내의 오퍼레이션들의 그룹이다. 예를 들

어, 트레이스 내에서, 프로그램 순서상, 제 1 분기 오퍼레이션까지의 오퍼레이션들 및 제 1 분기 오퍼레이션을 포함하는 오퍼레이션들이 하나의 활성 그룹에 포함된다. 실행이 제 1 분기 오퍼레이션의 결정(resolution)에 따르는 오퍼레이션들이 다른 활성 그룹에 포함될 수 있다. 따라서, 활성 그룹은 동일한 활성 그룹 내의 오퍼레이션들이 모두 실행되거나 모두 실행되지 않는다는 점에서 기본 블록과 유사하다. 활성 그룹은 동일한 기본 블록 내의 오퍼레이션들이 서로 다른 트레이스들에 포함된다는 점에서 기본 블록과는 다르다(예컨대, 기본 블록 내의 일부 오퍼레이션들은 일 트레이스 내의 최종 활성 그룹에 포함되고 상기 기본 블록의 나머지 오퍼레이션들은 다른 트레이스의 제 1 활성 그룹에 포함될 수 있다). 따라서, 상기 동일한 활성 그룹 내의 오퍼레이션들은 동일한 기본 블록의 부분이어야 하지만, 동일한 기본 블록 내의 오퍼레이션들은 동일한 활성 그룹의 일부분일 필요는 없다(즉, 만약 상기 기본 블록이 하나 보다 많은 트레이스 캐시 엔트리에 미치는 경우라면).

[0060] 상기 활성 표시는 상기 동일한 트레이스 캐시 엔트리에 포함된 분기 오퍼레이션(들)에 대한 각 오퍼레이션의 프로그램 순서를 식별한다. 제 1 활성 그룹의 오퍼레이션들은 상기 트레이스 내의 제 1 분기 오퍼레이션의 결과에 의존하지 않는다. 주의할 점은, 상기 제 1 분기 오퍼레이션의 실행이 조건적이지 않기 때문에, 상기 제 1 분기 오퍼레이션은 상기 제 1 활성 그룹의 부분이라는 점이다. 상기 제 1 활성 그룹 내의 상기 오퍼레이션들은 상기 제 1 분기 오퍼레이션 후에 일어난 오퍼레이션들의 제 2 활성 그룹을 식별하는데 사용되는 것과는 다른 활성 표시로 식별된다. 유사하게, 제 1 분기 오퍼레이션 및 제 2 분기 오퍼레이션(즉, 상기 제 2 활성 그룹에 포함된 최종 오퍼레이션) 모두의 결과에 따르는 제 3 활성 그룹 내의 명령어들은 또 다른 활성 표시로 식별된다. 활성 표시들은 오퍼레이션들이 그들이 프로그램 순서와 다른 순서로 트레이스 캐시 엔트리(162)에 저장되도록 하면서, 여전히 오퍼레이션들의 프로그램 순서(분기 오퍼레이션에 대한)가 또한 결정되도록 한다.

[0061] 추가적으로, 활성(liveness) 지시자들은 디스패치 유닛(104)이 특정 트레이스 내의 어떤 오퍼레이션들이 실행될 것인지를 예측하도록 한다. 제 1 활성 그룹에서의 오퍼레이션들(즉, 트레이스에 포함된 제 1 분기에 의존하지 않는 오퍼레이션들)이 항상 실행될 것이다(트레이스 내의 임의의 오퍼레이션들이 실행하는 것으로 가정한다). 하지만, 제 2 활성 그룹에서의 오퍼레이션들의 실행(즉, 제 1 분기에 의존하는 오퍼레이션들)은 제 1 분기의 결과에 의존한다. 예를 들어, 트레이스 발생기(17)가 프로세서(100)의 프로세싱 파이프라인의 후단에 포함되는 실시예들에서, 오퍼레이션들은 이 오퍼레이션들이 실행된 순서에 따라 트레이스들에 저장되어, 만일 상기 오퍼레이션들이 후속적으로 다시 실행되는 경우에, 명령어 캐시 대신에 트레이스 캐시로부터 액세스될 수 있게 된다. 따라서, 만일 트레이스 내에 저장된 제 1 분기가 실행되는 것으로 처음으로 테이큰되었다면, 제 2 활성 그룹에 저장된 오퍼레이션들은 제 1 분기가 테이큰 된다면 실행되어야할 오퍼레이션들이 될 것이다. 따라서, 만일 이후에 디스패치 유닛(104)에 트레이스가 제공되고 제 1 분기에 대한 현재 분기 예측이 "낫테이큰"인 경우에, 디스패치 유닛(104)은 제 2 활성 그룹 내의 오퍼레이션들이 실행되지 않아야 함을 예측할 수 있다. 제 1 분기 오퍼레이션과 관련된 흐름 제어 정보(168)가 또한 만일 제 1 분기가 낫테이큰되는 경우에 실행되어야 할 명령어들 또는 트레이스들을 프리페칭하기 시작하는데 사용될 수 있다.

[0062] 도 3B는 일 실시예에서 사용될 수 있는 예시적 활성 인코딩들을 도시한다. 제 1 활성 인코딩, " 무조건 활성"은 트레이스 내의 (프로그램 순서에서) 제 1 활성 그룹을 식별하는데에 사용될 수 있다. 이 활성 그룹 내의 오퍼레이션들은 만약 트레이스가 실행된다면 이들 오퍼레이션들이 항상 실행될 것이기 때문에 무조건 활성이다. 상기 제 1 활성 그룹은 상기 활성 그룹 내에 최종 오퍼레이션(프로그램 순서에서) 분기 오퍼레이션을 포함한다. 이러한 오퍼레이션들은 동일한 트레이스 내에 포함된 임의의 분기 오퍼레이션들에 의존하지 않는다. 많은 실시예들에서, 이러한 오퍼레이션들은 트레이스 내에서 서로에 대해 임의의 순서로 저장될 수 있다.

[0063] 후속 활성 인코딩, "제 1 분기 후속(subsequent to 1st branch)"은 트레이스 내에서 (프로그램 순서에서) 제 2 활성 그룹을 식별하는데에 사용된다. 이들 오퍼레이션들은 이전의 활성 그룹에 포함된 제 1 분기 오퍼레이션들의 결과에 의존한다. 주목할 점은, 만약 이 활성 그룹 내의 임의의 오퍼레이션들이 비-추론적으로 실행된다면, 이 활성 그룹 내의 모든 오퍼레이션들이 실행될 것이라는 점이다.

[0064] 제 3 활성 인코딩, "제 2 분기 후속"은 트레이스 내의 (프로그램 순서에서) 제 3 기본 블록을 식별하는데에 사용된다. 이들 오퍼레이션들은 트레이스 내의 제 1 분기 및 제 2 분기 오퍼레이션들 모두의 결과에 의존한다. 따라서, 제 2 활성 인코딩을 갖는 오퍼레이션들이 실행되는 경우에도 이들 오퍼레이션들은 실행하지 않을 수 있다. 전술한 바와 같이, 이 활성 그룹 내의 임의의 오퍼레이션들이 비-추론적으로 실행된다면, 이 활성 그룹 내의 모든 오퍼레이션들이 실행될 것이다.

[0065] "무조건 불활성(unconditional dead)" 활성 인코딩은 미사용된 오퍼레이션 저장 유닛(166)을 식별하는데에 사용될 수 있다. 최대 수보다 많은 분기 오퍼레이션들이 오퍼레이션들의 세트 내에서 발생하는 경우에, 오퍼레이션

저장 유닛들(166)은 사용되지 않을 수 있다. 예를 들어, 만일 8개까지의 오퍼레이션들이 트레이스에 저장될 수 있으며, 또한 단지 2개의 분기 오퍼레이션들만이 트레이스에 저장될 수 있는 경우에, 8개의 오퍼레이션들의 소정의 세트가 3개의 분기 오퍼레이션들을 포함한다면, 8개보다 적은 오퍼레이션들이 트레이스에 저장될 수 있다. 따라서, 만일 트레이스 내에 저장될 수 있는 분기 오퍼레이션들의 개수(N)에 상한이 정해진 경우라면, 트레이스에 N번째 분기 오퍼레이션의 저장이 트레이스를 종료시킬 수 있다. 더욱이, 일부 실시예들에서, 만일 특정 오퍼레이션이 트레이스 내에 저장되는 경우에, 비록 오퍼레이션 저장부가 이용가능하더라도, (프로그램 순서에서) 후속 오퍼레이션들이 트레이스 내에 저장되지 않을 수 있다. 예를 들어, 서브루틴 호출 오퍼레이션들(subroutine call operations)이 트레이스를 종료시킬 수 있다(예컨대, 서브루틴 내의 제 1 오퍼레이션이 다른 트레이스 내의 제 1 오퍼레이션으로서 저장될 수 있게 되며, 이는 상기 오퍼레이션 태그가 트레이스 태그로서 사용되게 한다). 유사하게는, 서부루틴 복귀 오퍼레이션들(subroutine return operation)이 트레이스를 종료시킬 수 있다.

[0066] 도 1을 참조하면, 트레이스 발생기(170)는 각 오퍼레이션에 대한 적절한 활성 인코딩을 발생시키며, 상기 활성 인코딩을 트레이스 캐시(160)의 트레이스 캐시 엔트리(162) 내에 상기 오퍼레이션과 동일한 오퍼레이션 저장 유닛(166)에 저장하도록 구성된다. 트레이스 발생기(170)는 각 분기 오퍼레이션이 처음으로 실행되었던 각 분기의 결과에 관한 정보에 기초하여 각 오퍼레이션에 대한 활성 인코딩들을 발생시킬 수 있다. 이 정보는 퇴거큐(102)로부터 제공될 수 있다. 추가 정보는 하나 이상의 이전 실행들에서 각 분기에 대한 각 예측 성공 및/또는 예측된 결과를 식별하는 분기 예측 유닛(132)으로부터 제공될 수 있다.

[0067] 명령어들이 트레이스 캐시(16)로부터 페치되는 때에, 각 오퍼레이션과 관련된 활성 인코딩이 명령어와 함께 프로세서를 통해 전파될 수 있다. 퇴거큐(102)는 오퍼레이션들이 퇴거할 수 있는 경계들을 결정하기 위해 각 오퍼레이션에 대한 활성 인코딩을 사용할 수 있다. 예를 들어, 트레이스 내의 임의의 모든 오퍼레이션들이 퇴거될 준비가 될 때까지 상기 트레이스 내의 오퍼레이션들의 퇴거를 차단하는 대신에, 이 트레이스 내의 동일한 활성 그룹의 모든 오퍼레이션들이 퇴거 준비를 하자마자, 퇴거큐(102)는 상기 트레이스의 오퍼레이션들의 서브세트(subset)를 퇴거시킬 수 있다. 일부 실시예들에서, 이 증가된 퇴거 입도(retirement granularity)는 재시작 로직(190)이 차단된 퇴거 때문에 실행을 재시작될 필요의 가능성을 감소시킬 수 있다.

[0068] 도 4는 트레이스 캐시 내에 저장을 위한 트레이스들을 발생시키는 방법의 일 실시예를 도시한다. 단계 10에서, 하나 이상의 명령어들이 트레이스 캐시 내에 저장을 위해 수신된다. 만일 임의의 명령어들이 예외적인 경우에(예컨대, 만일 임의의 명령어들이 예외를 야기할 수 있거나 임의의 명령어들이 재실행될 수 없는 경우에), 단계 12 및 14에 기재된 바와 같이, 각 예외 명령어들은 임의의 다른 명령어와 동일한 트레이스에 저장될 수 없다. 만일 단계 10에서 어떠한 예외 명령어들도 수신되지 않은 경우라면, 단계 12 및 16에서 기재된 바와 같이, 상기 명령어들은 동일한 트레이스에서 함께 저장되고 및/또는 추가의 비-예외 명령어와 동일한 트레이스에 포함될 수 있다.

[0069] 도 5는 상기 트레이스 캐시 내의 트레이스에 우연히 포함된 검출되지 않은 예외 명령어들을 처리하는 방법의 일 실시예의 흐름도이다. 단계 20에서, 명령어 스트림은 트레이스 캐시로부터 페치된다. 단계 22에서 결정된 바와 같이, 만약 예외 명령어가 상기 페치된 명령어 스트림으로부터 페치된다면, 단계 24에 개시된 바와 같이, 파이프라인은 플러시되고 프로세서 실행은 명령어 캐시로부터 재시작된다.

[0070] 상기 명령어 스트림 내의 일부 예외 명령어들은 단계 22에서 검출되지 않는다. 만약 이러한 명령어가 예외를 야기하고, 상기 예외 명령어와 동일한 트레이스 내의 다른 명령어들의 퇴거를 차단한다면, 단계 26 및 28에 개시된 바와 같이, 상기 프로세서 파이프라인은 플러시(flush)되고 상기 실행은 상기 명령어 캐시로부터 재시작된다. 명령어 퇴거가 차단되는지를 결정하는 것은 명령어 퇴거를 모니터하는 것을 포함한다. 만약 어떠한 명령어들도 소정 수의 사이클 동안 퇴거되지 않는다면, 퇴거는 차단되는 것으로 식별된다. 만약 명령어 퇴거가 차단되지 않는다면, 단계 26 및 30에 개시된 바와 같이, 상기 트레이스 캐시로부터 페치된 명령어 스트림은 실행하고 퇴거하도록 된다.

[0071] **예시적인 컴퓨터 시스템들**

[0072] 도 6는 버스 브리지(402)를 통해 다양한 시스템 컴포넌트와 연결된 프로세서(100)를 포함하는 컴퓨터 시스템(400)의 일 실시예의 블록 다이어그램을 도시한다. 프로세서(100)는 전술한 바와 같은 디스패치 유닛(104), 트레이스 캐시(160), 퇴거큐(102), 재시작 로직(190), 및/또는 트레이스 캐시 발생기(170)를 포함한다. 컴퓨터 시스템의 다른 실시예들이 가능하고 또한 예기될 수 있다. 도시된 시스템에서, 주 메모리(404)가 메모리 버스(406)를 통해 버스 브리지(402)에 연결되고, 그래픽 제어기(408)가 AGP 버스(410)를 통해 버스 브리지(402)에

연결된다. 수개의 PCI 디바이스들(412A-412B)이 PCI 버스(414)를 통해 버스 브리지(402)에 연결된다. 제 2 버스 브리지(416)가 또한 EISA/ISA 버스(420)를 통해 하나 이상의 EISA, 또는 ISA 디바이스들로(418)로의 전기적 인터페이스를 제공하기 위해 제공된다. 본 예시에서, 프로세서(100)가 CPU 버스(424)를 통해 버스 브리지(402)에 연결되고 선택적 L2 캐시(428)에 연결된다. 일부 실시예에서, 프로세서(100)는 집적 L1 캐시(도시되지 않음)를 포함한다.

[0073] 버스 브리지(402)는 프로세서(100), 주 메모리(404), 그래픽 제어기(408), 및 PCI 버스(414)에 부착된 디바이스들 사이에 인터페이스를 제공한다. 오퍼레이션이 버스 브리지(402)에 연결된 디바이스들 중 하나로부터 수신되면, 버스 브리지(402)는 상기 오퍼레이션의 타겟(예컨대, 특정 디바이스 또는 PCI 버스(414)의 경우에 상기 타겟은 PCI 버스(414) 상에 있다)을 식별한다. 버스 브리지(402)는 상기 오퍼레이션을 상기 타겟 디바이스에 라우팅한다. 버스 브리지(402)는 일반적으로 오퍼레이션을 소스 디바이스나 버스에 의해 사용되는 프로토콜로부터 타겟 디바이스나 버스에 의해 사용되는 프로토콜로 변환한다.

[0074] PCI 버스(414)에 대한 ISA/EISA 버스로의 인터페이스를 제공하는 것에 추가하여, 제 2 버스 브리지(416)는 추가의 기능을 도입한다. 제 2 버스 브리지(416)의 외부에 또는 제 2 버스 브리지와 함께 통합된 입력/출력 제어기(도시되지 않음)가 컴퓨터 시스템(400)에 포함되어 키보드 및 마우스(422) 및 다양한 직렬 및 병렬 포트에 대한 운영상의 지원을 제공한다. 다른 실시예들에서, 외부 캐시 유닛(도시되지 않음)이 또한 프로세서(100)와 버스 브리지(402) 사이에서 CPU 버스(424)에 연결된다. 대안적으로, 외부 캐시가 버스 브리지(402)에 연결되고 외부 캐시에 대한 캐시 제어 로직이 버스 브리지(402)에 통합된다. L2 캐시(428)가 프로세서(100)의 후면 구성에서 도시된다. 주목할 사항으로, L2 캐시(428)가 프로세서(100)로부터 격리되어, 프로세서(100)를 갖는 카트리지(cartridge)(예컨대, 슬롯 1 또는 슬롯 A)로 통합되거나 프로세서(00)를 갖는 반도체 기판상에 통합될 수 있다는 점이다.

[0075] 주 메모리(404)는 애플리케이션 프로그램이 저장되고 프로세서(100)가 우선 실행하는 메모리이다. 적당한 주 메모리(404)는 DRAM을 포함한다. 예를 들면, 복수의 SDRAM나 램버스 DRAM(Rambus DRAM) 뱅크들이 적당할 수 있다.

[0076] PCI 디바이스들(412A-412B)은 네트워크 인터페이스 카드, 비디오 가속기, 오디오 카드, 하드 또는 플로피 디스크 드라이브 또는 드라이브 제어기, SCSI(소형 컴퓨터 시스템 인터페이스) 어댑터 및 전화 카드와 같은 다양한 주변 버스들의 예시이다. 유사하게, ISA 디바이스(418)는 모뎀, 사운드 카드, 및 GPIB 또는 필드 버스 인터페이스 카드(field bus interface card)와 같은 다양한 데이터 수집 카드와 같은 다양한 유형의 주변 디바이스들의 예시이다.

[0077] 디스플레이(426) 상에 텍스트와 영상의 렌더링(rendering)을 제어하도록 그래픽 제어기(408)가 제공된다. 그래픽 제어기(408)는 주 메모리(404)로 및 주 메모리(404)로부터 효과적으로 이동할 수 있는 3차원 데이터 구조를 렌더링하는 것으로 당해 업계에 일반적으로 알려진 전형적인 그래픽 가속기를 포함한다. 그래픽 제어기(408)는 따라서 AGP 버스(410)의 매스터(master)인바, 이는 버스 브리지(402) 내의 타겟 인터페이스로 액세스를 요청하고 수신할 수 있어서 주 메모리(404)로의 액세스를 획득할 수 있기 때문이다. 전용 그래픽 버스는 주 메모리(404)로부터 데이터의 급속 검색을 제공한다. 특정 오퍼레이션에 대해서, 그래픽 제어기(408)는 또한 AGP 버스(410) 상에 PCI 프로토콜 트랜잭션들을 생성하도록 구성된다. 따라서, 버스 브리지(402)의 AGP 인터페이스는 PCI 프로토콜 타겟 및 개시 트랜잭션들 뿐 아니라 AGP 프로토콜 트랜잭션들 모두를 지원하는 기능을 포함할 수 있다. 디스플레이(426)는 영상 또는 텍스트가 제공될 수 있는 임의의 전자 디스플레이이다. 적당한 디스플레이(426)는 음극선 튜브(CRT), 액정 디스플레이(LCD) 등을 포함할 수 있다.

[0078] 주목할 사항으로, 상기에서 AGP, PCI 및 ISA 또는 EISA 버스가 예시로서 사용되었지만, 모든 버스 아키텍처가 필요시에 대체될 수 있다는 점이다. 또한, 컴퓨터 시스템(400)은 추가의 프로세서(예컨대, 컴퓨터 시스템(400)의 선택적 컴포넌트로 도시된 프로세서(100a)를 포함하는 멀티프로세싱 컴퓨터 시스템일 수 있다. 프로세서(100a)는 프로세서(100)와 유사하다. 더욱 상세하게, 프로세서(100a)는 프로세서(100)의 동일한 카피(copy)이다. 프로세서(100a)는 독립적인 버스를 통해 버스 브리지(402)에 접속되거나(도 6에 도시된 바와 같이) 또는 프로세서와 CPU 버스(424)를 공유한다. 게다가, 프로세서(100a)는 L2 캐시(428)에 유사한 선택적인 L2 캐시(428a)에 연결된다.

[0079] 도 7에서, 컴퓨터 시스템(400)의 다른 실시예가 도시된다. 다른 실시예들이 가능하고 예기된다. 도 7의 실시예에서, 컴퓨터 시스템(400)은 수 개의 프로세싱 노드(612A, 612B, 612C 및 612D)를 포함한다. 각 프로세싱 노드는 각각의 프로세싱 노드(612A-612D) 내에 포함된 메모리 제어기(616A-616D)를 통해 각 메모리(614A-614D)에 연

결된다. 추가로, 프로세싱 노드(612A-612D)는 상기 프로세싱 노드들(612A-612D) 사이에서 통신하는데 사용되는 인터페이스 로직을 포함한다. 예를 들면, 프로세싱 노드(612A)는 프로세싱 노드(612B)와 통신하기 위한 인터페이스 로직(618A)과, 프로세싱 노드(612C)와 통신하기 위한 인터페이스 로직(618B), 및 또 다른 프로세싱 노드(도시되지 않음)와 통신하기 위한 제 3 인터페이스 로직(618C)을 포함한다. 유사하게, 프로세싱 노드(612B)는 인터페이스 로직(618D, 618E 및 618F)를 포함하고; 프로세싱 노드(612C)는 인터페이스 로직(618G, 618H 및 618I)를 포함하고; 그리고 프로세싱 노드(612D)는 인터페이스 로직(618J, 618K 및 618L)을 포함한다. 프로세싱 노드(612D)는 인터페이스 로직(618L)을 통해 복수의 입력/출력 디바이스들(예컨대, 데이지 체인 구성(daisy chain configuration) 내의 디바이스들(620A-620B))과 통신하도록 연결된다. 다른 프로세싱 노드들은 유사한 방식으로 다른 I/O 디바이스들과 통신할 것이다.

[0080] 프로세싱 노드들(612A-612D)은 프로세싱 노드들 사이의 통신을 위해 패킷 기반 링크를 실행한다. 본 실시예에서, 상기 링크는 단방향 라인들 세트들(sets of unidirectional lines)(예컨대, 라인(624A)은 프로세싱 노드(612A)로부터 프로세싱 노드(612B)로 패킷을 전송하는데 사용되고 라인(624B)은 프로세싱 노드(612B)로부터 프로세싱 노드(612A)로 패킷을 전송하는데 사용된다)로서 구현된다. 다른 라인들 세트(624C-624H)가 도 7에 도시된 바와 같은 다른 프로세싱 노드들 사이에 패킷을 전송하는데 사용된다. 일반적으로, 각 라인들 세트(624)는 하나 이상의 데이터 라인들, 상기 데이터 라인들에 대응하는 하나 이상의 클록 라인들(clock lines), 및 운반되는 패킷의 유형을 나타내는 하나 이상의 제어 라인들을 포함한다. 상기 링크는 프로세싱 노드들 사이에서 통신하기 위해 캐시 코히런트 방식으로 또는 프로세싱 노드와 I/O 디바이스(또는 PCI 버스 또는 ISA 버스와 같은 종래 구성의 I/O 버스에 대한 버스 브리지) 사이에서 통신하기 위해 넌코히런트(non-coherent) 방식으로 오퍼레이팅된다. 게다가, 상기 링크는 도시된 바와 같이 I/O 디바이스들 사이의 데이지 체인 구조를 사용하여 넌코히어런트 방식으로 오퍼레이팅될 수 있다. 주목할 사항은, 하나의 프로세싱 노드로부터 다른 프로세싱 노드로 전송되는 패킷은 하나 이상의 중간 노드들을 통과한다는 점이다. 예를 들면, 프로세싱 노드(612A)에 의해 프로세싱 노드(612D)로 전송된 패킷은 도 7에 도시된 바와 같이 프로세싱 노드(612B)나 프로세싱 노드(612C)를 통해 통과한다. 모든 적당한 라우팅 알고리즘이 사용될 수 있다. 컴퓨터 시스템(400)의 다른 실시예들은 도 6에 도시된 실시예들보다 많거나 적은 프로세싱 노드들을 포함할 수 있다.

[0081] 일반적으로, 노드들 사이의 상기 라인들(624) 상에 하나 이상의 비트 시간들로서 전송될 수 있다. 비트 시간은 대응하는 클록 라인들 상의 클록 신호의 상승 또는 하강 에지일 것이다. 패킷들은 트랜잭션을 개시하기 위한 커맨드 패킷, 캐시 코히어런시를 유지하기 위한 프로브 패킷, 및 프로브와 커맨드에 대한 응답으로부터 응답 패킷을 포함한다.

[0082] 메모리 제어기 및 인터페이스 로직에 더하여 프로세싱 노드들(612A-612D)은 하나 이상의 프로세서들을 포함한다. 일반적으로 말해서, 프로세싱 노드는 적어도 하나의 프로세서를 포함하고 선택적으로 필요한 경우에 메모리 및 다른 로직과 통신하기 위해 메모리 제어기를 포함한다. 더욱 상세하게, 각 프로세싱 노드(612A-612D)는 프로세서(100)의 하나 이상의 카피를 포함한다. 프로세싱 노드들(612)은 전술한 바와 같이 각각 디스패치 유닛(104), 트레이스 캐시(160), 퇴거큐(102), 재시작 로직(190), 및/또는 트레이스 캐시 발생기(170)를 포함하는 프로세서(100)를 포함한다. 외부 인터페이스 유닛은 메모리 제어기(616) 뿐 아니라 노드 내에 인터페이스 로직(618)을 포함한다.

[0083] 메모리들(614A-614D)은 모든 적당한 메모리 디바이스들을 포함한다. 예를 들면, 메모리(614A-614D)는 하나 이상의 RAMBUS DRAM들(RDRAM들), 동기식 DRAM(SDRAM), 정적 RAM등을 포함한다. 컴퓨터 시스템(400)의 어드레스 공간은 메모리들(614A-614D) 사이에 분할된다. 각 프로세싱 노드(612A-612D)는 어떤 어드레스가 어떤 메모리들(614A-614D)로 매핑되었는지, 따라서 특정 어드레스에 대한 메모리 요청이 어떤 프로세싱 노드(612A-612D)로 라우팅되어야 하는지를 결정하는데 사용되는 메모리 맵을 포함한다. 일 실시예에서, 컴퓨터 시스템(400) 내의 어드레스에 대한 코히어런시 포인트는 상기 어드레스에 대응하는 바이트를 저장하는 메모리에 연결된 메모리 제어기(616A-616D)이다. 즉, 메모리 제어기(616A-616D)는 대응하는 메모리(614A-614D)에 대한 각 메모리 액세스가 캐시 코히어런트 방식으로 발생하였음을 보증한다. 메모리 제어기(616A-616D)는 메모리들(614A-614D)로 인터페이스하는 제어 회로소자를 포함한다. 추가로, 메모리 제어기(616A-616D)는 메모리 요청들을 큐잉(queuing)하는 메모리 큐를 포함한다.

[0084] 인터페이스 로직(618A-618L)은 링크로부터 패킷을 수신하고 링크로 전송될 패킷들을 버퍼링하는 다양한 버퍼들을 포함한다. 컴퓨터 시스템(400)은 패킷 전송의 임의의 적당한 흐름 제어 메커니즘을 이용한다. 예를 들면, 일 실시예에서, 각 인터페이스 로직(618)은 인터페이스 로직이 접속되는 링크의 다른 종단에 있는 수신기 내의 각 버퍼 유형의 개수의 집계를 저장한다. 수신 인터페이스 로직이 패킷을 저장하기 위한 빈공간을 갖고 있지 않다

면 인터페이스 로직은 패킷을 전송하지 않는다. 수신 버퍼는 패킷을 순방향으로 라우팅함으로써 비워져 이용가능한 상태로 되기 때문에, 수신 인터페이스 로직은 상기 버퍼가 이용가능하게 되었음을 표시는 메시지를 전송 인터페이스 로직에 전송한다. 이러한 메커니즘은 "쿠폰-기반(coupon-based)" 시스템으로 지칭된다.

[0085] I/O 디바이스들(620A-620B)은 모든 적당한 I/O 디바이스들이다. 예를 들면, I/O 디바이스들(620A-620B)은 상기 디바이스들이 연결된 다른 컴퓨터 시스템과 통신하기 위한 디바이스들(예컨대, 네트워크 인터페이스 카드 또는 모뎀)을 포함한다. 게다가, I/O 디바이스들(620A-620B)은 비디오 가속기, 오디오 카드, 하드 또는 플로피 디스크 드라이브 또는 드라이브 제어기, SCSI(소형 컴퓨터 시스템 인터페이스) 어댑터 및 전화 카드, 사운드 카드, 및 GPIB 또는 필드 버스 인터페이스 카드(field bus interface card)와 같은 다양한 데이터 수집 카드를 포함한다. 주목할 사항으로, 용어 "I/O 디바이스" 및 용어 "주변 디바이스"는 본 명세서에서 동의어로서 사용된다.

[0086] 본 명세서에서 사용되는 바와 같이, 용어 "클록 사이클" 또는 "사이클"은 명령어 프로세싱 파이프라인들의 다양한 스테이지들이 그들의 작업을 완료하는 시간 간격을 일컫는다. 명령어 및 컴퓨팅 값들은 클록 사이클을 정의하는 클록 신호에 따라 메모리 요소(레지스터나 어레이와 같은)에 의해 획득된다. 예를 들면, 메모리 요소는 클록 신호의 상승 또는 하강 에지에 따른 값을 획득한다.

[0087] 삭제

[0088] 본 명세서를 일단 숙지한 당해 기술분야의 당업자에게는 다양한 수정 변경이 가능할 것이다. 후술하는 청구항들은 이러한 모든 수정 변경을 포괄하도록 해석되어야 한다.

산업상 이용 가능성

[0089] 본 발명은 일반적으로 프로세서 분야에 응용할 수 있다.

도면의 간단한 설명

[0009] 하기의 상세한 설명을 첨부된 도면과 함께 고려할 때 본 발명을 더욱 잘 이해할 수 있다.

[0010] 도 1은 프로세서의 일 실시예를 도시한다.

[0011] 도 2는 일 실시예에 따른 트레이스 캐시의 블록 다이어그램이다.

[0012] 도 3A는 일 실시예에 따른 예시적인 트레이스 캐시 엔트리를 예시한다.

[0013] 도 3B는 일 실시예에 따른 트레이스의 각 오퍼레이션이 속하는 활성 그룹(liveness group)을 식별하는데 사용되는 활성 인코딩의 테이블이다.

[0014] 도 4는 일 실시예에 따른 트레이스 캐시에 저장을 위한 트레이스들을 발생시키는 방법의 일 실시예의 흐름도이다.

[0015] 도 5는 일 실시예에 따른 예외 명령어를 포함하는 트레이스를 실행하는 방법의 일 실시예의 흐름도이다.

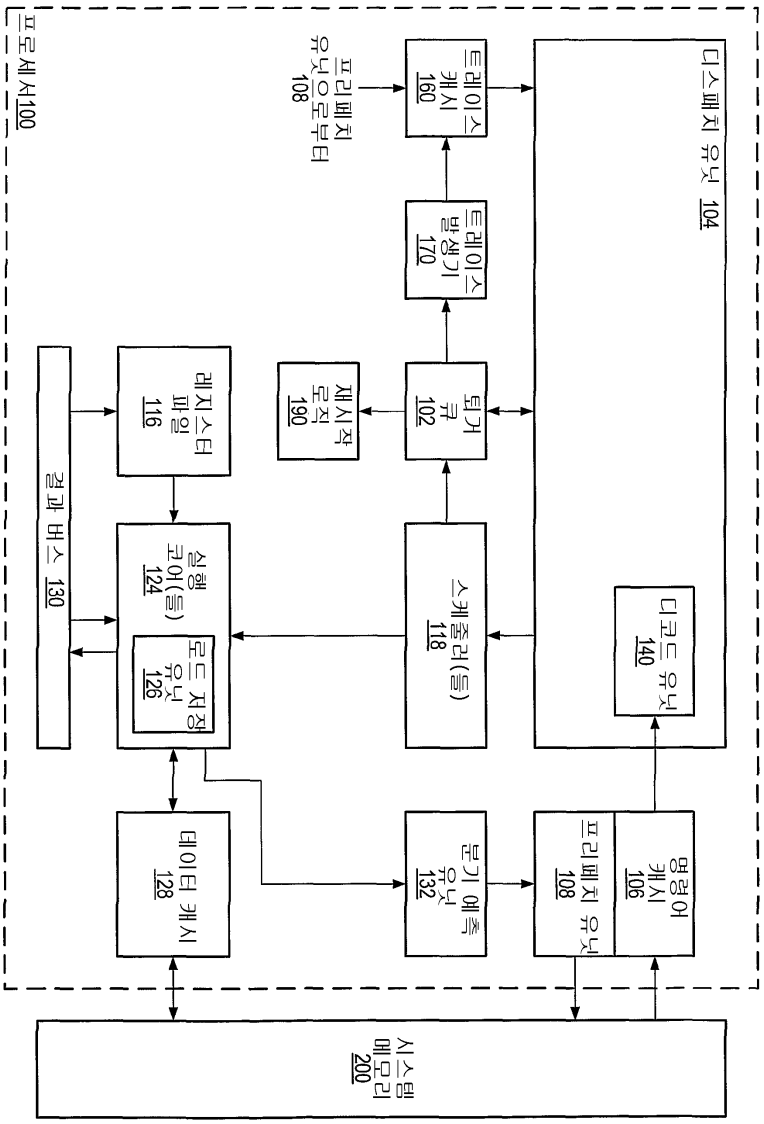
[0016] 도 6은 일 실시예에 따른 컴퓨터 시스템의 블록 다이어그램을 도시한다

[0017] 도 7은 다른 실시예에 따른 컴퓨터 시스템의 블록 다이어그램이다.

[0018] 본 발명은 다양한 수정들 및 대체 형상들이 가능하지만, 이들의 특정 실시예들이 도면에서 예시로서 도시되고 본 명세서에서 상세하게 설명될 것이다. 그러나 이들에 대한 도면들 및 상세한 설명은 본 발명을 개시된 특정 형태로 제한하기 위함이 아니고, 첨부된 청구항들에 의해 정의된 본 발명의 정신 및 범위 내에 있는 모든 수정물, 균등물 및 대체물들을 포함하도록 의도된다. 게다가, 표현 "이다(may)"는 본원을 통해 허용적인 의미(즉, 가능성을 갖는, 할 수 있는)로 사용되며 필수적인 의미(즉, 강제성)로 사용되지 않음을 주목하여야 한다. 표현 "포함한다" 및 이것에서 파생된 표현들은 "포함하지만, 이에 한정되지 않는다"를 의미한다. 표현 "접속된"은 "직접 또는 간접 접속된"을 의미하고 "연결된"은 "직접 또는 간접 연결된"을 의미한다.

도면

도면1


프로세서 100
프리페치 유닛으로부터 108
트레이스 캐시 160
트레이스 발생기 170
디스패치 유닛 104
퇴거 큐 102
재시작 로직 190
레지스터 파일 116
결과 버스 130
실행 코어(들) 124
로드 저장 유닛 126
스케줄러(들) 118
디코드 유닛 140
데이터 캐시 128
분기 예측 유닛 132
프리페치 유닛 108
명령어 캐시 106
시스템 메모리 200

도면2

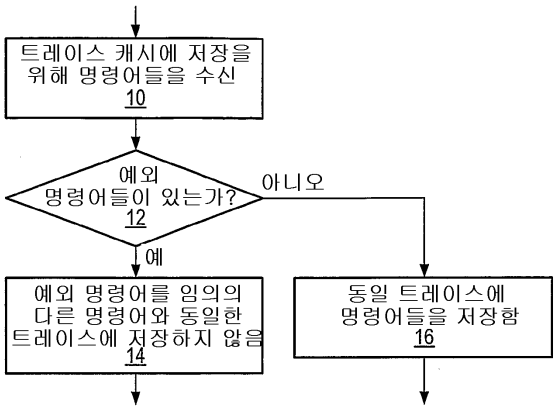

트레이스 캐시
160
162
태그
164
명령어들
165
흐름 제어
168

도면3A

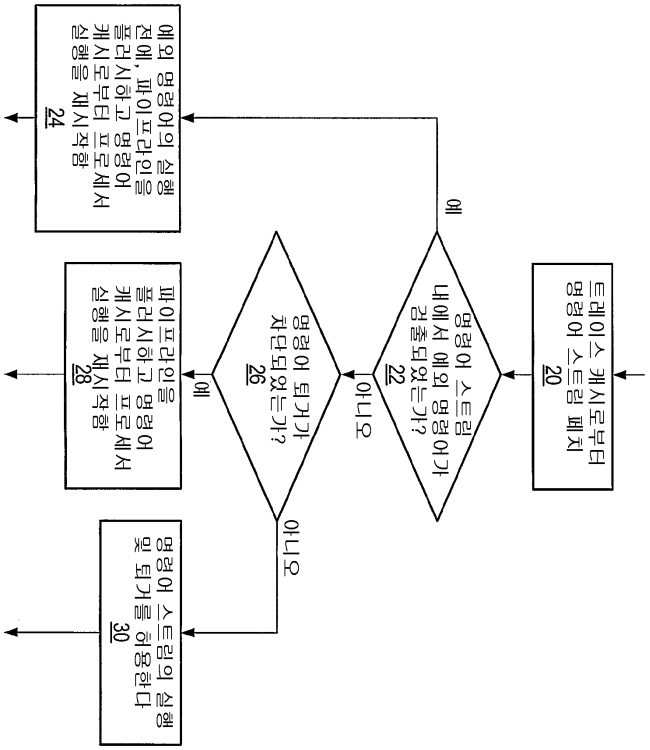
도면3B

활성	인코딩 266
무조건 활성	11
제 1 분기 후속	10
제 2 분기 후속	01
무조건 불활성	00

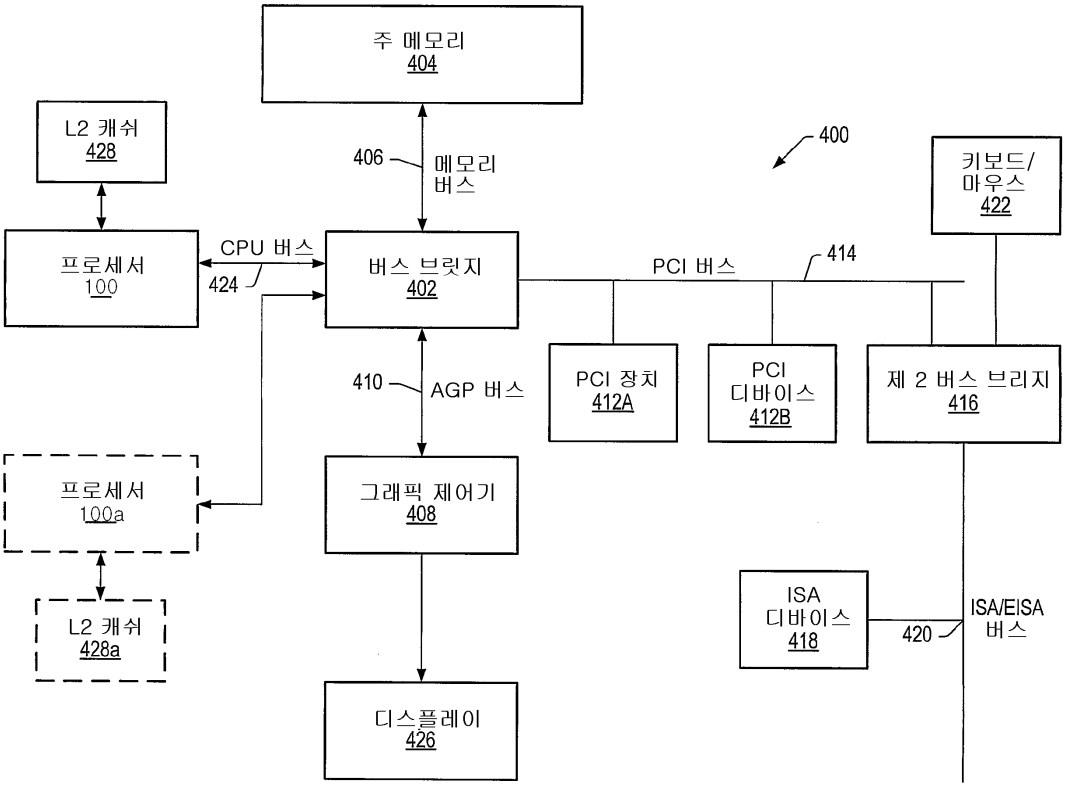
도면4



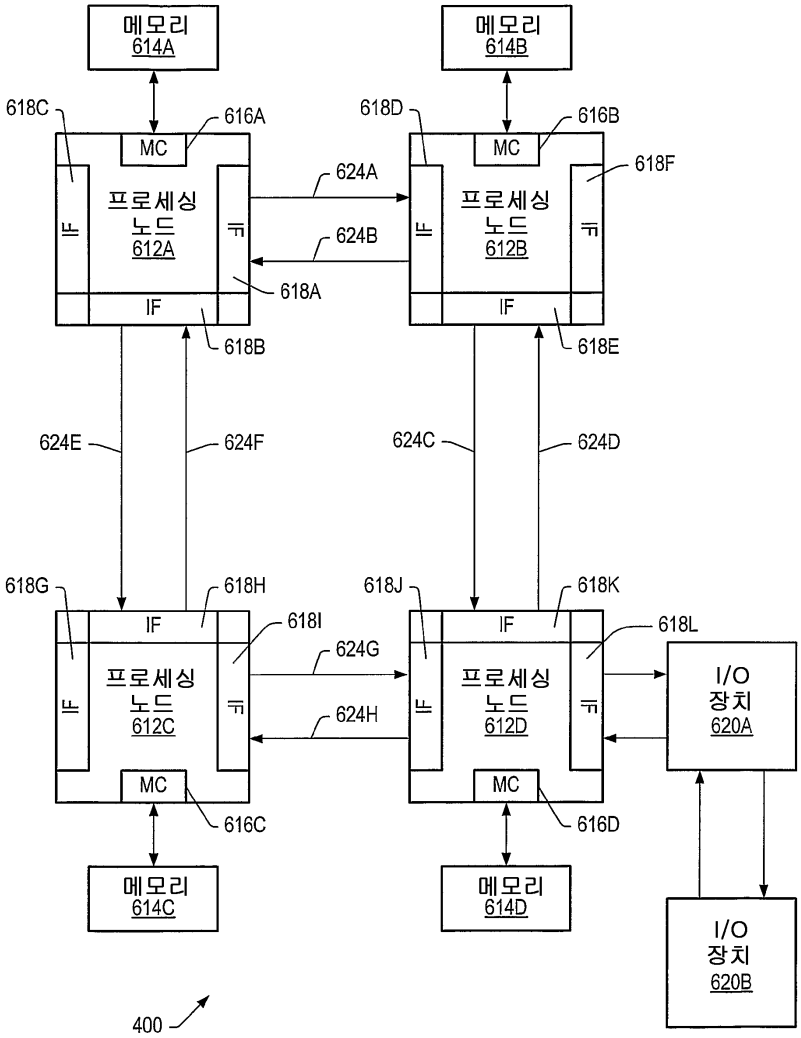
도면5


트레이스 캐시로부터 명령어 스트림 페치 20
명령어 스트림 내에서 예외 명령이 검출되었는가? 22
아니오
예
명령어 퇴거가 차단되었는가? 26
아니오
예
파이프라인을 플러시하고 명령어 캐시로부터 프로세서 실행을 재시작함 28
명령어 스트림의 실행 및 퇴거를 허용한다 30
예외 명령어의 실행 전에, 파이프라인을 플러시하고 명령어 캐시로부터 프로세서 실행을 재시작함 24

도면6


주 메모리 404
L2 캐쉬 428
406
메모리 버스
400
키보드/ 마우스 422
프로세서 100
CPU 버스 424
버스 브릿지 402
PCI 버스
414
410
AGP 버스
PCI 장치 412A
PCI 디바이스 412B
제 2 버스 브리지 416
프로세서 100a
그래픽 제어기 408
ISA 디바이스 418
420
ISA/EISA 버스
L2 캐쉬 428a
디스플레이 426

도면7


메모리
614A
메모리
614B
618C
616A
MC
618D
616B
MC
618F
프로세싱
노드
612A
624A
624B
프로세싱
노드
612B
IF
IF
IF
IF
618A
618B
618E
624E
624F
624C
624D
618G
618H
618J
618K
IF
IF
618I
618L
624G
624H
프로세싱
노드
612C
프로세싱
노드
612D
I/O
장치
620A
MC
MC
616C
616D
메모리
614C
메모리
614D
I/O
장치
620B
400