(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2011년02월08일
(11) 등록번호 10-1013237
(24) 등록일자 2011년01월28일

(51) Int. Cl.
G06F 12/08 (2006.01) *G06F 13/28* (2006.01)
(21) 출원번호 10-2007-7021818
(22) 출원일자(국제출원일자) 2006년03월17일
심사청구일자 2008년09월26일
(85) 번역문제출일자 2007년09월21일
(65) 공개번호 10-2007-0119653
(43) 공개일자 2007년12월20일
(86) 국제출원번호 PCT/US2006/010038
(87) 국제공개번호 WO 2006/104747
국제공개일자 2006년10월05일
(30) 우선권주장
11/093,130 2005년03월29일 미국(US)
(56) 선행기술조사문헌
US20030115424 A1
전체 청구항 수 : 총 10 항

(73) 특허권자
인터내셔널 비지네스 머신즈 코포레이션
미국 10504 뉴욕주 아몬크 뉴오차드 로드
(72) 발명자
블럼리치 마티아스 에이
미국 코네티컷주 06877 리지필드 플로리다 힐 로드 76
가라 알렌 지
미국 뉴욕주 10549 마운트 키스코 메리언 에버뉴 38
살라퓨라 발렌티나
미국 뉴욕주 10514 차파쿠아 브룩 레인 31
(74) 대리인
윤여원, 허정훈

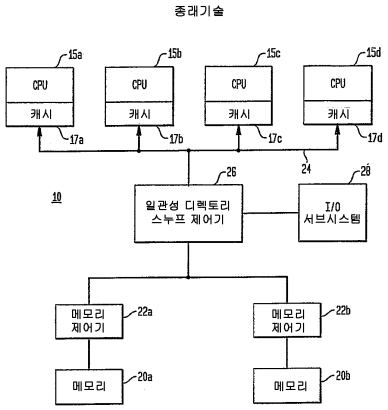
심사관 : 이상헌

(54) 스트림 레지스터를 이용하여 스누프 요구를 필터링하는방법 및 장치

(57) 요 약

다중 처리 유닛을 구비한 멀티프로세서 컴퓨팅 환경에서 캐시 일관성을 지원하는 방법 및 장치를 제공하고, 상기 각 처리 유닛은 서로 관련된 로컬 캐시 메모리를 갖는다. 스누프 필터 장치는 각 처리 유닛과 관련되고, 스트림 레지스터 세트 및 관련 스트림 레지스터 비교 로직을 사용하여 필터링 방법을 구현하는 적어도 하나의 스누프 필터 프리미티브를 포함한다. 복수의 스트림 레지스터 세트로부터, 적어도 하나의 스트림 레지스터 세트는 능동으로 되고, 적어도 하나의 스트림 레지스터 세트는 임의의 시점에서 히스토릭으로 라벨이 붙여진다. 또한, 스누프 필터 블록은 캐시 랩 검출 로직에 동작적으로 결합되고, 이것에 의해 능동 스트림 레지스터 세트의 콘텐츠가 캐시 랩 조건 검출시에 히스토릭 스트림 레지스터 세트로 전환되고, 적어도 하나의 능동 스트림 레지스터 세트의 콘텐츠는 리세트된다. 각 필터 프리미티브는 수신된 스누프 요구가 프로세서에 회송되어야 하는지 또는 버려져야 하는지를 결정하는 스트림 레지스터 비교 로직을 구현한다.

대 표 도 - 도1



특허청구의 범위

청구항 1

다중 처리 유닛을 가진 컴퓨팅 환경의 각각의 처리 유닛에 관련된 스누프 필터 장치로서, 각각의 처리 유닛은 서로 관련된 하나 이상의 캐시 메모리를 가지고, 상기 스누프 필터 장치는 관련된 처리 유닛에 1:1 대응하고, 상기 스누프 필터 장치는

관련된 처리 유닛의 캐시 메모리 레벨에 로드된 콘텐츠의 캐시 라인 어드레스를 추적하도록 구성된 제1 메모리 기억장치 수단 - 상기 제1 메모리 기억장치 수단은 제1 복수의 스트림 레지스터 세트를 포함하고, 각각의 스트림 레지스터 세트는 하나 이상의 스트림 레지스터를 포함하고, 각각의 스트림 레지스터는 베이스 레지스터 및 대응하는 마스크 레지스터를 포함하고, 상기 베이스 레지스터는 상기 스트림 레지스터에 의해 나타내어지는 모든 캐시 라인 어드레스에 공통인 어드레스 비트를 추적하고, 상기 대응하는 마스크 레지스터는 대응하는 베이스 레지스터에 포함되는 베이스 레지스터 어드레스와 상기 처리 유닛의 상기 하나 이상의 캐쉬 메모리로의 후속(subsequent) 로드/저장 어드레스 사이에서 상이함(differences)을 나타내는 비트를 추적하고, 상이한(differing) 비트의 위치가 상기 마스크 레지스터에 표시되어 상기 베이스 레지스터의 대응하는 비트가 중요하지 않다는 것을 표시함 - 과;

하나 이상의 메모리 기록 소스로부터 스누프 요구를 수신하는 수단과;

수신된 스누프 요구의 어드레스를 상기 제1 메모리 기억장치 수단의 상기 제1 복수의 스트림 레지스터 세트에 저장된 어드레스에 대해 비교하는 스누프 체크 로직 수단과;

상기 스트림 레지스터 세트에 저장된 어드레스와의 일치에 응답해서 상기 처리 유닛에 상기 수신된 스누프 요구를 회송하고, 일치가 없으면 상기 스누프 요구를 버리는 수단

을 포함하고, 처리 유닛에 회송되는 스누프 요구의 수를 크게 감소시킴으로써 상기 컴퓨팅 환경의 성능을 증가시키는 스누프 필터 장치.

청구항 2

제1항에 있어서, 상기 스누프 요구의 메모리 기록 소스는 상기 다중 처리 유닛 중의 하나를 포함하는 것인, 스누프 필터 장치.

청구항 3

제1항에 있어서, 상기 스누프 요구의 메모리 기록 소스는 직접 메모리 액세스(DMA) 엔진을 포함하는 것인, 스누프 필터 장치.

청구항 4

제1항에 있어서, 상기 스누프 필터 장치는 각각, 스누프 요구의 부분집합을 수신하고 엔큐하고 상기 스누프 필터 장치의 관련 처리 유닛에 회송하기 위한, 상기 하나 이상의 메모리 기록 소스에 대응하는 복수의 프로세서 스누프 필터 큐 수단을 더 포함하는 것인, 스누프 필터 장치.

청구항 5

제4항에 있어서, 상기 스누프 필터 장치는 각각, 모든 스누프 필터 큐 수단들 사이에서 중재를 행하고, 상기 관련 처리 유닛에 회송하기 위해 상기 복수의 프로세서 스누프 필터 큐 수단 각각으로부터 회송된 모든 스누프 요구를 직렬화하는 수단을 더 포함하는 것인, 스누프 필터 장치.

청구항 6

다중 처리 유닛을 가진 컴퓨팅 환경에서 캐시 일관성을 지원하는 스누프 필터링 방법으로서, 각각의 처리 유닛은 서로 관련된 하나 이상의 캐시 메모리 및 관련된 스누프 필터 장치를 가지고, 스누프 필터 장치는 관련된 처리 유닛에 1:1 대응하여 제공되고, 상기 방법은

처리 유닛에 관련된 각각의 스누프 필터 장치에서,

관련된 처리 유닛의 캐시 메모리 레벨에 로드된 데이터의 캐시 라인 어드레스를 추적하고 캐시 라인 어드레스를

제1 메모리 기억장치 수단에 저장 - 상기 제1 메모리 기억장치 수단은 제1 복수의 스트림 레지스터 세트를 포함하고, 각각의 스트림 레지스터 세트는 하나 이상의 스트림 레지스터를 포함하고, 각각의 스트림 레지스터는 베이스 레지스터 및 대응하는 마스크 레지스터를 포함하고, 상기 베이스 레지스터는 상기 스트림 레지스터에 의해 나타내어지는 모든 캐시 라인 어드레스에 공통인 어드레스 비트를 추적하고, 상기 마스크 레지스터는 대응하는 베이스 레지스터에 포함되는 베이스 레지스터 어드레스와 상기 처리 유닛의 상기 캐쉬 메모리로의 후속(subsequent) 로드/저장 어드레스 사이에서 상이함(differences)을 나타내는 비트를 추적하고, 상이한(differing) 비트의 위치가 상기 마스크 레지스터에 표시되어 상기 베이스 레지스터의 대응하는 비트가 중요하지 않다는 것을 표시함 - 하는 단계와;

복수의 메모리 기록 소스로부터 스누프 요구를 수신하는 단계와;

수신된 스누프 요구의 어드레스를 상기 제1 메모리 기억장치 수단에 저장된 어드레스에 대해 비교하는 단계와;

상기 제1 메모리 기억장치 수단에 저장된 어드레스와의 일치에 응답해서 상기 처리 유닛에 상기 수신된 스누프 요구를 회송하고, 일치가 없으면 상기 스누프 요구를 버리는 단계

를 포함하고, 처리 유닛에 회송되는 스누프 요구의 수를 크게 감소시킴으로써 상기 컴퓨팅 환경의 성능을 증가시키는 스누프 필터링 방법.

청구항 7

제6항에 있어서, 상기 관련 스누프 필터 장치는 각각 상기 복수의 메모리 기록 소스 각각에 대응하는 복수의 포트 스누프 필터를 포함하고, 상기 수신된 스누프 요구를 회송하는 단계는,

포트 스누프 필터에 대응하는 각각의 프로세서 스누프 필터 큐 수단에서, 상기 스누프 필터 장치의 관련 처리 유닛에 회송되는 스누프 요구의 부분집합을 엔큐하는 단계를 포함하는 것인, 스누프 필터링 방법.

청구항 8

제7항에 있어서, 상기 수신된 스누프 요구를 회송하는 단계는,

모든 스누프 필터 큐 수단들 사이에서 중재를 행하고, 상기 관련 처리 유닛에 회송하기 위해 상기 프로세서 스누프 필터 큐 수단 각각으로부터 회송된 모든 스누프 요구를 직렬화하는 단계를 더 포함하는 것인, 스누프 필터링 방법.

청구항 9

제6항에 있어서, 스트림 레지스터 세트의 상기 베이스 레지스터 및 대응하는 마스크 레지스터를 수행된 각 캐시 로드의 캐시 라인 어드레스로 업데이트하는 단계를 더 포함하고, 상기 업데이트하는 단계는 어떤 스트림 레지스터 세트의 베이스 레지스터 및 마스크 레지스터를 업데이트할 것인지를 레지스터 선택 기준에 기초하여 결정하는 단계를 포함하는 것인, 스누프 필터링 방법.

청구항 10

다중 처리 유닛을 포함하는 컴퓨팅 환경에서 캐시 일관성(coherency)을 지원하기 위한 컴퓨터 프로그램이 기록된 컴퓨터 판독가능 기록매체로서, 각각의 처리 유닛은 서로 관련된 하나 이상의 캐시 메모리 및 관련된 스누프 필터 장치를 가지고, 스누프 필터 장치는 관련된 처리 유닛에 1:1 대응하여 제공되고, 상기 컴퓨터 프로그램은 스누프 필터링 방법을 구현하고, 상기 스누프 필터링 방법은

처리 유닛에 관련된 각각의 스누프 필터 장치에서,

관련된 처리 유닛의 캐시 메모리 레벨에 로드된 데이터의 캐시 라인 어드레스를 추적하고 캐시 라인 어드레스를 제1 메모리 기억장치 수단에 저장 - 상기 제1 메모리 기억장치 수단은 제1 복수의 스트림 레지스터 세트를 포함하고, 각각의 스트림 레지스터 세트는 하나 이상의 스트림 레지스터를 포함하고, 각각의 스트림 레지스터는 베이스 레지스터 및 대응하는 마스크 레지스터를 포함하고, 상기 베이스 레지스터는 상기 스트림 레지스터에 의해 나타내어지는 모든 캐시 라인 어드레스에 공통인 어드레스 비트를 추적하고, 상기 마스크 레지스터는 대응하는 베이스 레지스터에 포함되는 베이스 레지스터 어드레스와 상기 처리 유닛의 상기 캐쉬 메모리로의 후속(subsequent) 로드/저장 어드레스 사이에서 상이함(differences)을 나타내는 비트를 추적하고, 상이한(differing) 비트의 위치가 상기 마스크 레지스터에 표시되어 상기 베이스 레지스터의 대응하는 비트가 중요하

지 않다는 것을 표시함 - 하는 단계와;

복수의 메모리 기록 소스로부터 스누프 요구를 수신하는 단계와;

수신된 스누프 요구의 어드레스를 상기 제1 메모리 기억장치 수단에 저장된 어드레스에 대해 비교하는 단계와;

상기 제1 메모리 기억장치 수단에 저장된 어드레스와의 일치에 응답해서 상기 처리 유닛에 상기 수신된 스누프 요구를 회송하고, 일치가 없으면 상기 스누프 요구를 버리는 단계

를 포함하고, 처리 유닛에 회송되는 스누프 요구의 수를 크게 감소시킴으로써 상기 컴퓨팅 환경의 성능을 증가시키는 것인,

컴퓨터 판독가능 기록매체.

명 세 서

기 술 분 야

[0001] 본 발명은 일반적으로 멀티프로세서 구조를 가진 컴퓨터 시스템에 관한 것이고, 특히, 메모리 액세스 요구를 처리하기 위한 신규의 멀티프로세서 컴퓨터 시스템 및 이러한 멀티프로세서 시스템에서의 캐시 일관성(cache coherence)의 구현에 관한 것이다.

배 경 기 술

[0002] 고성능 컴퓨팅을 달성하기 위해, 다수의 개별 프로세서가 상호접속되어 병렬 처리가 가능한 멀티프로세서 컴퓨터 시스템을 형성한다. 다중 프로세서는 단일 칩으로 형성될 수도 있고, 또는 멀티프로세서 컴퓨터 시스템에 상호접속되고 한 개 또는 수 개의 프로세서를 각각 내포하는 수 개의 칩으로 형성될 수도 있다.

[0003] 멀티프로세서 컴퓨터 시스템 내의 프로세서들은 짧은 액세스 시간 때문에 및 메인 메모리에 대한 메모리 요구의 수를 줄이기 위해 개인(private) 캐시 메모리를 사용한다(캐시는 프로세서에 대해서 국부적(local)이고 데이터에 대해 고속 액세스를 제공한다). 그러나, 멀티프로세서 시스템에서 캐시를 관리하는 것은 복잡하다. 다중 개인 캐시는 멀티프로세서 시스템에서 동시에 존재할 수 있는 메인 메모리 데이터의 다중 복사(multiple copies) 때문에 멀티 캐시 일관성 문제(또는 스테일 데이터 문제)를 유도한다.

[0004] 소규모 공유 메모리 멀티프로세서 시스템은 단일 버스에 의해 상호 접속되는 복수의 프로세서(또는 그 그룹)를 갖는다. 그러나, 프로세서의 속도가 증가함에 따라, 버스를 효과적으로 공유할 수 있는 프로세서의 가능한 수가 감소한다.

[0005] 다중 프로세서들 간에 일관성을 유지하는 프로토콜은 캐시 일관성 프로토콜(cache coherence protocol)이라고 부른다. 캐시 일관성 프로토콜은 프로세서들 사이에서 임의의 데이터 블록 공유를 추적한다. 데이터 공유가 어떻게 추적되는가에 따라서, 캐시 일관성 프로토콜은 2가지의 부류, 즉 1) 디렉토리 기반형과 2) 스누핑(snooping)으로 그룹지어질 수 있다.

[0006] 디렉토리 기반형 방법에서, 물리 메모리 블록의 공유 상태는 일관성 디렉토리라고 불리는 하나의 장소에서만 유지된다. 일관성 디렉토리는 일반적으로 멀티프로세서 컴퓨터 시스템의 어떤 프로세서가 어떤 메모리 라인을 소유하는지를 계속하여 추적하는 큰 블록의 메모리이다. 불리하게도, 일관성 디렉토리는 전형적으로 크고 느리다. 이들은 메모리에 대한 각 액세스가 공통 디렉토리를 통과할 것을 요구함으로써 각 메모리 액세스 요구에 대해 추가적인 잠재시간(latency)을 유도하기 때문에 전체 시스템 성능을 심각하게 감퇴시킬 수 있다.

[0007] 도 1은 캐시 일관성을 위하여 일관성 디렉토리 방법을 이용하는 전형적인 종래의 멀티프로세서 시스템(10)을 도시한 것이다. 멀티프로세서 시스템(10)은 공유 버스(24)를 통해 상호접속되고 메모리 제어기(22a, 22b)를 통해 메인 메모리(20a, 20b)에 각각 접속되는 다수의 프로세서(15a, ..., 15d)를 포함한다. 각 프로세서(15a, ..., 15d)는 자신의 개인 캐시(17a, ..., 17d)를 각각 가지며, 상기 개인 캐시는 N-웨이 세트 조합형(N-way set associative)이다. 프로세서로부터 메모리로의 각 요구는 프로세서 버스(24) 상에 놓여지고 일관성 디렉토리(26)로 지향된다. 때때로, 일관성 제어기에는 불필요한 스누프 요구를 모든 캐시 에이전트에 방송하는 필요성을 제거하기 위해 특수 서브시스템 내에 유지된 캐시 라인의 위치를 추적하는 모듈이 포함된다. 이 유닛은 종종 "스누프 제어기" 또는 "스누프 필터"라고 표시된다. I/O 서브시스템(28)으로부터의 모든 메모리 액세스 요구도 또한 일관성 제어기(26)로 지향된다. 메인 메모리 대신에, 메인 메모리에 접속된 2차 캐시가 사용될 수 있다.

프로세서는 프로세서 클러스터로 그룹지어질 수 있으며, 각 클러스터는 일관성 제어기(26)에 접속되는 그 자신의 클러스터 버스를 갖는다. 각 메모리 요구가 일관성 디렉토리를 통과하기 때문에, 요구된 메모리 블록의 상태를 체크하기 위해 각 요구에 추가의 사이클이 추가된다.

[0008] 스누핑 방법에서, 중앙집중화 상태(centralized state)가 유지되지 않고, 오히려 각 캐시가 데이터 블록의 공유 상태를 국부적으로 유지한다. 캐시는 일반적으로 공유 메모리 버스상에 있고, 모든 캐시 제어기는 버스를 스누프(감시)하여 이들이 요구된 데이터 블록의 카피를 갖는지를 판정한다. 공통으로 사용되는 스누핑 방법은 "기록 무효화"(write-invalidate) 프로토콜이다. 이 프로토콜에서, 프로세서는 프로세서가 데이터를 기록하기 전에 그 데이터에 대한 배타적 액세스를 갖는 것을 확실히 한다. 각각의 기록에서, 다른 모든 캐시 내에 있는 데이터의 모든 다른 카피들은 무효화된다. 만일 2개 이상의 프로세서가 동일한 데이터를 동시에 기록하려고 시도하면, 그들 중의 하나만이 경쟁에서 승리하고 다른 프로세서의 카피는 무효화 된다.

[0009] 기록 무효화 프로토콜 기반 시스템에서 기록을 수행하기 위해, 프로세서는 공유 버스를 획득하고 버스에서 무효화되는 어드레스를 방송한다. 모든 프로세서는 버스상에서 스누프하고, 데이터가 그들의 캐시 내에 있는지를 체크한다. 만일 있으면, 그 데이터들은 무효화된다. 따라서, 공유 버스를 사용하면 기록 직렬화(write serialization)가 강화된다.

[0010] 불리하게도, 스누핑 방법에서의 모든 버스 트랜잭션은 캐시 어드레스 태그를 체크해야 하는데, 이것은 CPU 캐시 액세스를 방해할 수 있다. 가장 최근의 구조물에서, 이것은 전형적으로 어드레스 태그를 이중화함으로써 감소되고, 따라서 CPU와 스누핑 요구는 병렬로 진행할 수도 있다. 대안적인 방법은 내포된 멀티레벨 캐시를 사용하여 1차 캐시의 모든 엔트리가 더 낮은 레벨의 캐시에서 이중화되도록 하는 것이다. 그러면, 스누프 행동(activity)이 2차 레벨 캐시에서 수행되어 CPU 행동을 방해하지 않는다.

[0011] 도 2는 캐시 일관성을 위하여 스누핑 방법을 이용하는 전형적인 종래의 멀티프로세서 시스템(50)을 도시한 것이다. 멀티프로세서 시스템(50)은 공유 버스(56)를 통해 메인 메모리(58)에 상호접속되는 다수의 프로세서(52a, ..., 52c)를 포함한다. 각 프로세서(52a, ..., 52c)는 자신의 개인 캐시(54a, ..., 54c)를 가지며, 상기 개인 캐시는 N-웨이 세트 조합형이다. 프로세서로부터 메모리로의 각 기록 요구는 프로세서 버스(56) 상에 놓여진다. 모든 프로세서는 버스상에서 스누프하고, 기록된 어드레스가 또한 그들의 캐시 내에 있는지를 확인하기 위해 그들의 캐시를 체크한다. 만일 있으면, 이 어드레스에 대응하는 데이터는 무효화된다. 몇개의 멀티프로세서 시스템은 무효화되는 캐시 라인이 특수 캐시 내에 유지되는지를 추적하기 위해 각 프로세서에 모듈을 국부적으로 추가하고, 그에 따라서 로컬 스누핑 행동을 효과적으로 감소시킨다. 이 유닛은 종종 "스누프 필터"라고 표시된다. 메인 메모리 대신에, 메인 메모리에 접속된 2차 캐시가 사용될 수 있다.

[0012] 버스상의 프로세서의 수가 증가함에 따라, 스누핑 행동이 또한 증가한다. 캐시에 대한 불필요한 스누프 요구는 프로세서 성능을 감퇴시킬 수 있고, 캐시 디렉토리에 액세스하는 각 스누프 요구는 전력을 소모한다. 또한, 스누핑 행동을 지원하기 위해 각 프로세서용의 캐시 디렉토리를 이중화하면 칩의 사이즈를 크게 증가시킨다. 이것은 전력 예산(power budget)이 제한된 단일 칩상의 시스템에 있어서는 특히 중요하다.

[0013] 이어서, 멀티프로세서 시스템에서 발견되는 종래의 스누핑 방법의 각종 문제점들을 다루는 종래의 참고 문헌에 대해서 설명한다.

[0014] 특히, 미국 특허 출원 US2003/0135696A1 및 미국 특허 제6,704,845B2호는 둘 다 스누프 필터를 포함한 일관성 디렉토리 기반 방법에 있어서 스누프 필터 내의 엔트리를 교체하기 위한 교체 정책 방법에 대하여 개시하고 있다. 스누프 필터는 캐시 라인이 캐시된 캐시 메모리 블록 및 그 상태에 대한 정보를 포함하고 있다. 미국 특허 출원 US2004/0003184A1은 원격 노드에 의해 액세스되는 로컬 캐시 라인들을 기록하는 짝수 및 홀수 어드레스 라인을 기록하기 위한 서브스누프 필터를 포함한 스누프 필터에 대해서 개시하고 있다(서브필터는 동일한 필터링 방법을 사용한다). 이 문헌들은 각각 멀티프로세서 시스템의 각 캐시에 제시되는 스누프 요구의 수를 국부적으로 감소시키는 시스템 및 방법에 대해서 교시하거나 제안하고 있지 않다. 이들은 또한 수 개의 스누프 필터를 각종 필터링 방법과 결합시키는 것 및 캐시에 대해 스누핑 정보의 점대점 상호접속을 제공하는 것에 대하여 교시하거나 제안하고 있지 않다.

[0015] 미국 특허 출원 US2003/0070016A1 및 US2003/0065843A1은 스누프 필터를 포함한 중앙 일관성 디렉토리를 가진 멀티프로세서 시스템에 대해서 개시하고 있다. 상기 출원들에 개시되어 있는 스누프 필터는 스누프 요구를 처리하기 위한 사이클의 수를 감소시키지만, 캐시에게 제시되는 스누프 요구의 수를 감소시키지 못한다.

[0016] 미국 특허 제5,966,729호는 캐시 일관성을 위해 스누핑 방법을 이용하여 버스를 공유하는 멀티프로세서 시스템

및 각 프로세서 그룹에 국부적으로 관련된 스누프 필터에 대해서 개시하고 있다. 스누핑 행동을 줄이기 위해, 특수 캐시 라인에서 "관심" 및 "비관심"의 원격 프로세서 그룹의 목록이 유지된다. 스누프 요구는 "관심"으로 표시된 프로세서 그룹에만 보내지고, 따라서 방송되는 스누프 요구의 수를 감소시킨다. 그러나, 상기 특허는 로컬 프로세서에 대한 스누프 요구의 수를 감소시키는 방법에 대해서 개시하고 있지 않으며, 오히려 "비관심"으로 표시된 다른 프로세서 그룹에 보내지는 스누프 요구의 수를 감소시키는 방법에 대해서 개시하고 있다. 이 해법은 프로세서 그룹의 캐시에서 각 라인의 관심 그룹에 대한 정보를 목록에 유지할 것을 요구하며, 이것은 크기면에서 프로세서 그룹 내의 각 프로세서의 캐시 디렉토리를 이중화하는 것에 필적하고, 따라서 칩의 사이즈를 크게 증가시킨다.

[0017] 미국 특허 제6,389,517B1호는 2개의 액세스 큐(queue)를 가진 스누프 액세스 및 프로세서 둘 다로부터 캐시에 대한 동시 액세스를 가능하게 하는 캐시 일관성 스누핑 방법에 대해서 개시하고 있다. 이 문헌에 기술된 실시예는 공유 버스 구성에 관한 것이다. 이 특허는 캐시에 제시되는 스누프 요구의 수를 감소시키는 방법에 대해서 개시하고 있지 않다.

[0018] 미국 특허 제5,572,701호는 고속 버스 및 프로세서에 대한 저속 버스의 간섭을 감소시키는 버스 기반 스누프 방법에 대해서 개시하고 있다. 스누프 버스 제어 유닛은 프로세서가 고속 버스를 해방(release)할 때까지 저속 버스로부터의 어드레스 및 데이터를 버퍼링한다. 그 다음에, 스누프 버스 제어 유닛은 데이터를 전송하고 캐시의 대응하는 라인을 무효화한다. 이 문헌은 모든 구성 요소들이 고속 버스를 통하여 통신하는 멀티프로세서 시스템에 대해서 개시하고 있지 않다.

[0019] A. Moshovos, G. Memik, B. Falsafi 및 A. Choudhary에 의한 "JETTY: filtering snoops for reduced energy consumption in SMP servers"("제티")의 문헌에는 하드웨어 필터를 이용하여 스누프 요구를 감소시키는 몇가지 제안이 개시되어 있다. 이 문헌은 스누프 요구가 공유 시스템 버스를 통해 분배되는 멀티프로세서 시스템을 개시하고 있다. 프로세서에 제시되는 스누프 요구의 수를 감소시키기 위해, 하나 이상의 각종 스누프 필터가 사용된다.

[0020] 그러나, 제티에 개시된 시스템은 성능, 지원되는 시스템 및 더 구체적으로 상호접속 구조, 및 멀티포팅(multiporting)에 대한 지원 부족에 대해서 심각한 제한을 갖는다. 더 구체적으로, 제티에서 개시된 방법은 시스템을 통한 공통 이벤트 오더링을 확립한 공유 시스템 버스에 기반을 두고 있다. 이러한 총체적 시간 오더링이 필터 구조를 단순화하기 위해 바람직하지만, 이것은 가능한 시스템 구성을 단일 공유 버스를 갖는 것으로 제한한다. 또한, 공유 버스 시스템은 단일의 총체적 리소스에 대한 경쟁 때문에 확장성(scalability)이 제한되는 것으로 알려져 있다. 또한, 글로벌 버스는 글로벌 버스에 부착되는 다중 구성 요소의 높은 부하 때문에 저속으로 되는 경향이 있고, 칩 멀티프로세서에 배치하기에 불충분하다.

[0021] 따라서, 고도로 최적화된 고대역폭 시스템에서, 성형(star)과 같은 다른 시스템 구조 또는 점대점(point-to-point) 구현을 제공하는 것이 바람직하다. 이러한 구조는 단일 전송기 및 송신기만을 갖기 때문에 부하를 감소시켜서 고속 프로토콜의 사용을 가능하게 하고, 칩 멀티프로세서에서 평면도를 단순화시키 때문에 유리하다. 점대점 프로토콜의 사용은 또한 수 개의 진행중인 송신을 동시에 가능하게 하고, 이것에 의해 데이터 전송 병행 및 전체 데이터 수율을 증가시킨다.

[0022] 제티의 다른 제한 사항은 제티에서처럼 수 개의 요구에 대해 동시에 스누프 필터링을 수행하는 능력이 없다는 것이고, 수 개의 프로세서로부터의 동시 스누프 요구는 시스템 버스에 의해 직렬화되어야 한다. 수 개의 스누프 요구를 동시에 처리할 수 있게 하면, 임의의 한 시점에서 취급될 수 있는 요구의 수를 크게 증가시키고, 따라서 시스템의 전체 성능을 개선한다.

[0023] 종래 기술의 제한 사항을 설명하면, 시스템 설계 옵션을 제한하지 않고 전체 성능 및 전력 효율을 증가시키기 위해 스누프 필터를 통합하는 시스템이 필요하고, 더 구체적으로 말하면, 공통 버스를 요구하지 않는 시스템에서 스누프 필터링을 지원하는 방법 및 장치가 필요하다는 것이 명백하다.

[0024] 또한, 스누프 필터링을 이용한 고성능 시스템 구현을 위해 점대점 접속을 이용하여 시스템을 지원하는 스누프 필터 구조가 필요하다.

[0025] 아울러, 시스템 성능을 개선하기 위해 다중 메모리 기록기(writer)로부터의 요구를 동시에 필터링하기 위한 다중 스누프 필터 유닛의 동시 동작이 필요하다.

[0026] 더 나아가, 이러한 스누프 필터를 활용하는 시스템에서 고속의 시스템 클럭 속도를 달성하기 위해 파이프라인

형식으로 구현될 수 있는 신규의 고성능 스누프 필터를 제공할 필요가 있다.

[0027] 또한, 종래 기술의 제한을 능가하는 높은 필터링 효율을 가진 스누프 필터가 필요하다.

발명의 상세한 설명

[0028] 그러므로, 본 발명의 목적은 캐시 일관성 멀티프로세서 시스템에서 단일 프로세서 유닛에 제시되는 스누프 요구의 수를 감소시키기 위한 단순한 방법 및 장치를 제공하는데 있다.

[0029] 본 발명의 다른 목적은 입력 스누프 요구를 걸러내는 각 프로세서 유닛에 단순 하드웨어 장치를 국부적으로 추가함으로써 로컬 스누핑 행동을 효과적으로 감소시키는 방법 및 장치를 제공하는 것이다. 여기에서 설명하는 스누프 필터는 프로세서 유닛과 관련된 로컬 캐시 메모리에 로드된 데이터를 추적하기 위해 스트림 레지스터를 사용하고, 캐시 누락이 야기된 대부분의 스누프 요구를 걸러내지만, 국부적으로 캐시된 것으로 스트림 레지스터에서 표시된 데이터의 스누프 요구는 걸러내지 않음으로써 캐시 누락된 스누프 요구를 식별한다. 프로세서당 스누프 요구의 수를 감소시키면 시스템 성능을 증가시키고 전력을 감소시킨다.

[0030] 본 발명의 제1 태양에 따르면, 서로 관련된 하나 이상의 캐시 메모리를 각각 가진 다중 처리 유닛을 가진 컴퓨팅 환경의 단일 처리 유닛과 관련된 스누프 필터 장치를 제공하는데, 상기 스누프 필터는,

[0031] 관련 프로세서의 캐시 메모리 레벨에 로드된 데이터의 캐시 라인 어드레스를 추적하도록 구성된 제1 메모리 기억장치 수단과;

[0032] 하나 이상의 메모리 기록 소스로부터 스누프 요구를 수신하는 수단과;

[0033] 메모리 기억장치 수단에 저장된 어드레스와 수신된 스누프 요구의 어드레스를 비교하는 스누프 체크 논리 수단과;

[0034] 메모리 기억장치 수단의 어드레스와의 일치에 응답해서 프로세서에 수신 스누프 요구를 회송하고, 일치하지 않으면 스누프 요구를 버리는 수단을 포함하고,

[0035] 이렇게 하여 처리 유닛에 회송되는 스누프 요구의 수를 크게 감소시킴으로써 컴퓨팅 환경의 성능을 증가시킨다.

[0036] 더 구체적으로, 스누프 필터 장치는 각 처리 유닛과 관련되고, 스트림 레지스터 세트 및 관련 스트림 레지스터 비교 로직의 사용에 기초하여 필터링 방법을 구현하는 적어도 하나의 스누프 필터 프리미티브(primitive)를 포함한다. 복수의 스트림 레지스터 세트로부터, 적어도 하나의 스트림 레지스터 세트는 능동(active)이고, 적어도 하나의 스트림 레지스터 세트는 임의의 시점에서 히스토릭(historic)이라고 라벨이 붙여진다. 또한, 스누프 필터 블록은 캐시 랩(wrap) 검출 로직과 동작적으로 결합되고, 이것에 의해 능동 스트림 레지스터 세트의 콘텐츠는 캐시 랩 조건 검출시에 히스토릭 스트림 레지스터 세트로 전환되며, 적어도 하나의 능동 스트림 레지스터 세트의 콘텐츠는 리세트된다. 각 필터 프리미티브는 수신된 스누프 요구가 프로세서로 회송되어야 하는지 또는 버려져야 하는지를 결정하는 스트림 레지스터 비교 로직을 구현한다. 수신된 스누프 요구는 스트림 레지스터와 비교되고, 엔트리가 캐시에 있는지를 표시하는 결정이 행하여지지만, 그 실제 존재 상태는 표시하지 않는다.

[0037] 또한, 스트림 레지스터 세트에 기초를 둔 필터 프리미티브는 베이스 레지스터와 마스크 레지스터의 쌍을 이룬 세트를 복수개 포함한다. 상기 스누프 필터 유닛이 관련된 캐시 계층에 로드되어 있는 각각의 새로운 데이터에 대해서, 메모리 요구의 어드레스는 정확히 하나의 베이스 레지스터에 기록되고, 쌍을 이룬 마스크 레지스터가 업데이트된다. 본 발명에 따르면, 선택된 스트림 레지스터의 마스크 레지스터는 덜 차별적으로 됨으로써 미리 기록된 어드레스에 대한 차이를 계속하여 추적하도록 업데이트된다.

[0038] 본 발명의 제2 태양에 따르면, 서로 관련된 하나 이상의 캐시 메모리 및 관련된 스누프 필터 장치를 각각 가진 다중 처리 유닛을 가진 컴퓨팅 환경에서 캐시 일관성을 지원하는 스누프 필터링 방법을 제공하는데, 이 방법은,

[0039] 처리 유닛 내의 각 스누프 필터 장치에 대해서,

[0040] 관련 프로세서의 캐시 메모리 레벨에 로드된 데이터의 캐시 라인 어드레스를 추적하고, 이 캐시 라인 어드레스를 제1 메모리 기억장치 수단에 저장하는 단계와;

[0041] 복수의 메모리 기록 소스로부터 스누프 요구를 수신하는 단계와;

[0042] 메모리 기억장치 수단에 저장된 어드레스와 수신된 스누프 요구의 어드레스를 비교하는 단계와;

[0043] 메모리 기억장치 수단의 어드레스와의 일치에 응답해서 프로세서에 수신 스누프 요구를 회송하고, 일치하지 않

으면 스누프 요구를 버리는 단계를 포함하고,

[0044] 이렇게 하여 처리 유닛에 회송되는 스누프 요구의 수를 크게 감소시킴으로써 컴퓨팅 환경의 성능을 증가시킨다.

[0045] 유익하게, 종래의 시스템에서는 수 개의 프로세서로부터의 동시 스누프 요구가 시스템 버스에 의해 직렬화되어야 하지만, 본 발명은 스누프 필터링이 수 개의 요구에 대해서 동시에 수행되게 한다. 수 개의 스누프 요구의 동시 처리를 가능하게 하면, 임의의 한 시점에서 취급될 수 있는 요구의 수를 크게 증가시키고, 따라서 전체 시스템 성능을 증가시킨다.

[0046] 본 발명의 목적, 특징 및 장점들은 첨부도면과 함께하는 이하의 상세한 설명으로부터 당업자에게 명백하게 될 것이다.

실 시 예

[0068] 이제 도면을 참조하면(특히, 도 3을 참조하면), 도 3은 캐시 일관성을 위해 스누핑 방법을 사용하는 멀티프로세서 시스템의 전체적인 기본 구조를 보인 것이다. 양호한 실시예에서, 멀티프로세서 시스템은 로컬 L1 데이터 및 명령어 캐시를 가진 N개의 프로세서(100a, ..., 100n)(또는 DCU_1~DCU_N이라고 표시되는 CPU)와, 이들의 관련 L2 캐시(120a, ..., 120n)로 구성된다. 메인 메모리(130)는 공유되고 온칩 또는 오프칩으로 구현될 수 있다. 대안적인 실시예에서, 메인 메모리 대신에, 메인 메모리에 액세스하는 공유 L3를 사용할 수 있다. 양호한 실시예에서, 프로세서 코어(100a, ..., 100n)는 PPC440 또는 PPC405와 같은 파워PC(PowerPC) 코어이지만, 임의의 다른 프로세서 코어도 사용가능하고, 또는 단일 멀티프로세서 시스템에서 각종 프로세서의 임의 조합이 본 발명의 범위로부터 벗어나지 않고 사용될 수 있다. 프로세서 코어(100a, ..., 100n)는 시스템 로컬 버스(150)에 의해 상호접속된다.

[0069] 프로세서에 제시되는 스누프 요구의 수를 줄이고, 그에 따라서 프로세서 및 시스템 성능에서 스누핑의 충격을 줄이며, 불필요한 스누프 요구에 의한 전력 소모를 감소시키기 위해, 멀티프로세서 시스템(10)의 각 프로세서 코어(100a, ..., 100n)에는 스누프 필터(140a, ..., 140n)가 제공된다. 양호한 실시예는 스누핑 요구를 전송하기 위해 종래 기술의 시스템에서 전형적으로 사용되었던 시스템 버스(150)를 사용하지 않는다. 그 대신, 점대점(point-to-point) 상호접속(160)을 구현하고, 이것에 의해 각 프로세서의 관련 스누프 필터는 시스템 내의 임의의 다른 프로세서와 관련된 각 스누프 필터에 직접 접속된다. 따라서, 스누프 요구는 시스템 로컬 버스를 통해 전송되는 모든 다른 메모리 요구로부터 분리되고, 이로써 가끔 시스템 병목현상을 야기하는 버스 혼잡을 감소시킨다. 단일 프로세서에 대한 모든 스누프 요구는 스누프 필터(140a, ..., 140n)에 회송(forward)되는데, 상기 스누프 필터(140a, ..., 140n)는 동일한 필터링 방법을 가진, 또는 수 개의 상이한 필터링 방법을 가진, 또는 이들의 조합을 가진 복수의 서브필터를 포함하며, 이것에 대해서는 뒤에서 상세히 설명된다. 스누프 필터는 각 스누프 요구를 처리하고, 모든 요구 중에서 프로세서의 캐시에 있을만한 일부만을 프로세서에 제시한다.

[0070] 각 프로세서에 대해서, 스누프 요구는 점대점 상호접속(160)을 이용하여 모든 다른 프로세서 스누프 필터에 직접 접속된다. 따라서, 다른 프로세서로부터 몇개의 스누프 요구(기록 및 무효화 시도로부터 발생함)가 동시에 발생할 수 있다. 이 요구는 시스템 버스를 이용하는 전형적인 스누핑 방법에서처럼 더 이상 직렬화되지 않고, 이 직렬화는 버스에 의해 수행된다. 즉, 다수의 스누프 요구가 스누프 필터에서 동시에 처리될 수 있고, 이것에 대해서는 뒤에서 더 자세히 설명된다. 프로세서가 단지 하나의 스누프 포트를 갖기 때문에, 스누프 필터에 의해 필터링되지 않은 스누프 요구는 프로세서에 제시되는 큐에서 직렬화된다. 그러나, 프로세서에 전달된 요구의 수는 모든 스누프 요구의 미리 필터링된 수보다 훨씬 더 적고, 이로써 시스템 성능에서 캐시 일관성 구현의 충격을 감소시킨다.

[0071] 스누프 필터 블록에 내포된 큐의 큐 오버플로우 조건을 방지하기 위해, 토큰 기반 흐름 제어 시스템이 각각의 점대점 링크에 대해서 구현되어 동시 미해결(outstanding) 요구의 수를 제한한다. 토큰 기반 흐름 제어에 따르면, 다른 모든 프로세서 유닛 및 수반되는 스누프 필터 블록에 대한 스누프 요구를 또한 개시하는 각각의 메모리 기록기는 다음 기록 요구를 전송할 수 있고, 스누프 필터 블록의 모든 포트에 대해 이용가능한 토큰을 갖는 경우에만 점대점 접속을 갖는다. 접속되는 적어도 하나의 원격 포트로부터 가용인 토큰이 없으면, 상기 스누프 필터 포트로부터의 적어도 하나의 토큰이 다시 가용으로 될 때까지 상기 메모리 기록기로부터 스누프 요구가 전송될 수 없다.

[0072] 도 4는 스누핑 요구를 위해 점대점 상호접속을 가진 캐시 일관성에 대해 스누핑 방법을 사용하는 기본 멀티프로세서 시스템에 대한 본 발명의 다른 실시예를 보인 것이고, 여기에서 스누프 필터는 L2 캐시와 메인 메모리

(230) 사이에 배치된다. 이 실시예에 따른 멀티프로세서 시스템은 로컬 L1 데이터 및 명령어 캐시를 가진 N개의 프로세서(200a, ..., 200n)(또는 DCU_1~DCU_N이라고 표시되는 CPU)와, 이들의 관련 L2 캐시(220a, ..., 220n)로 구성된다. 메인 메모리(230)는 공유되고 온칩 또는 오프칩으로 구현될 수 있다. 대안적인 실시예에서, 메인 메모리 대신에, 메인 메모리에 액세스하는 공유 L3 캐시를 사용할 수 있다. 프로세서(200a, ..., 200n)로부터의 모든 메모리 액세스 요구는 시스템 로컬 버스(250)를 통해 전송된다. 도 4에 도시된 실시예에서, 멀티프로세서 시스템의 각 프로세서는 각각의 스누프 필터(240a, ..., 240n)와 쌍을 이룬다. 점대점 상호접속(260)은 양호한 실시예에서 시스템 버스의 혼잡을 감소시키도록 스누프 요구를 전송하기 위해 사용된다. 점대점 접속 방법(260)에서, 각 프로세서의 관련 스누프 필터는 시스템 내의 모든 다른 프로세서와 관련된 각 스누프 필터와 직접 접속된다. 단일 프로세서에 대한 모든 스누프 요구는 그 스누프 필터에 회송되고, 스누프 필터는 각 스누프 요구를 처리하여 모든 요구 중에서 적당한 일부만을 프로세서에 회송한다. 이 실시예에서, 스누프 요구는 (도 3의 실시예에서처럼 L1이 아니라) L2 캐시 레벨에서 필터링되지만, 본 발명은 임의의 캐시 레벨에도 적용가능하고, 본 발명의 범위로부터 벗어나지 않고 캐시 계층의 다른 레벨용으로도 사용될 수 있다.

[0073] 이제 도 5를 참조하면, 도 5는 본 발명에 따른 스누프 필터 장치의 고준위(high level) 블록도를 도시한 것이다. 멀티프로세서 시스템의 모든 다른 프로세서(1~N)로부터의 스누프 요구는 전용 점대점 상호접속 입력(300a, ..., 300n)을 통해 스누프 블록(310)에 회송된다. 스누프 블록(310)은 입력 스누프를 필터링하고 적당한 부분집합을 프로세서 스누프 인터페이스(340)를 통해 프로세서(320)에 회송한다. 또한, 스누프 블록(310)은 프로세서 및 L1 데이터 캐시 블록(320)으로부터 L2 캐시(330)까지의 모든 메모리 액세스 요구를 감시한다. 이들은 L1 캐시에서 누락된 요구일 뿐이다. 스누프 블록은 그 필터를 업데이트하기 위해 모든 판독 어드레스 및 제어 신호(360, 362)를 감시한다.

[0074] 도 6은 도 5에 도시된 스누프 블록(310)의 고준위 개요(high level schematic)를 도시한 것이다. 도 6에 도시된 바와 같이, 스누프 블록(310)은 다수("N")의 포트 스누프 필터(400a, ..., 400n)를 포함하는데, 이 다수의 포트 스누프 필터는 병렬로 동작하고 각각 N개의 메모리 기록기들(프로세서 또는 DMA 엔진 서브시스템 등) 중 하나의 소스에만 전용되는 것이다. 각각의 포트 스누프 필터(400a, ..., 400n)는 그 전용 입력(410a, ..., 410n)에 점대점으로 직접 접속된 단일 소스로부터 스누프 요구를 수신한다. 뒤에서 설명하는 바와 같이, 단일 포트 스누프 필터는 다수의 각종 스누프 필터 방법을 포함할 수 있다. 스누프 블록(310)은 추가적으로 스트림 레지스터 블록(430) 및 스누프 토큰 제어 블록(426)을 포함한다. 또한, 각각의 포트 스누프 필터(400a, ..., 400n)는 프로세서의 L1 레벨 캐시에서 누락된 관련 프로세서로부터의 모든 메모리 판독 액세스 요구(412)를 감시한다. 이 정보는 또한 사용을 위해 스트림 레지스터 블록(430)에 제공되는데, 이것에 대해서는 뒤에서 더 자세히 설명된다.

[0075] 동작시에, 포트 스누프 필터(400a, ..., 400n)는 입력 스누프 요구를 처리하여 각각의 스누프 포트와 관련된 하나의 큐를 가진 각각의 스누프 큐(420a, ..., 420n)에 모든 스누프 요구의 부분집합을 회송한다. 큐 중재 블록(422)은 모든 스누프 큐(420) 사이에서 중재를 제공하고, 스누프 큐(420)로부터의 모든 스누프 요구를 공정하게 직렬화한다. 스누프 큐 오버플로우 조건을 검출하기 위한 로직이 제공되고, 각 큐의 상태는 원격 메모리 기록기로부터 스누프 요구의 흐름을 제어하는 스누프 토큰 제어 유닛(426)에 입력된다. 프로세서 또는 DMA 엔진인 메모리 기록기는 모든 스누프 필터로부터 가용인 토큰을 갖는 경우에만 기록을 메모리에 제공하고 스누프 요구를 모든 스누프 필터에 제공할 수 있다. 프로세서가 기록을 제공할 수 있는 토큰을 필요로 하지 않는 유일한 스누프 필터는 그 자신의 로컬 스누프 필터이다. 이 메카니즘은 스누프 큐가 오버플로우되지 않게 하는 것을 보장한다. 스누프 요구는 중재자(422)가 선택한 스누프 큐로부터 프로세서 스누프 인터페이스(408)를 거쳐 프로세서에 회송된다.

[0076] 도 7은 단일 스누프 포트 필터(400)의 고준위 개요를 도시한 것이다. 스누프 포트 필터 블록(400)은 각종 필터링 알고리즘을 구현하는 다중 필터 유닛을 포함한다. 양호한 실시예에서, 3개의 스누프 필터 블록(440, 444, 448)이 병렬로 동작하고, 각 블록은 상이한 스누프 필터 알고리즘을 구현한다. 스누프 필터 블록은 스누프 캐시(440), 스트림 레지스터 체크 유닛(444) 및 범위 필터(range filter)(448)라고 표시되어 있다. 일 실시예에서, 각각의 병렬 스누프 필터 블록은 그 입력에서 단일 소스로부터의 동일한 스누프 요구(410)를 동시에 수신한다. 또한, 스누프 캐시(440)는 L1 레벨 캐시에서 누락한 프로세서로부터의 모든 메모리 판독 액세스 요구(412)를 감시하고, 스트림 레지스터 체크 유닛(444)은 도 6에 도시된 스트림 레지스터 유닛(430)으로부터의 상태 입력(432)을 수신한다.

[0077] 양호한 실시예에 따르면, 스누프 캐시 블록(440)은 스누프 요구의 임시 집약성(temporal locality property)에 기초하는 알고리즘을 이용하여 스누프 요구(410)를 필터링한다. 이것은 만일 특정 위치의 단일 스누프 요구가

만들어졌으면 동일한 위치에 대한 다른 요구가 곧 만들어질 가능성이 있다는 것을 의미한다. 스누프 캐시는 로컬 캐시에 대해 만들어진 모든 부하를 감시하고, 만일 필요하다면 그 상태를 업데이트한다. 스트림 레지스터 체크 블록(444)은 현재 로컬 캐시 콘텐츠의 합집합(superset)을 결정하는 알고리즘을 이용하여 스누프 요구(410)를 필터링한다. 캐시 콘텐츠의 근사치는 스트림 레지스터 블록(430)(도 6)에 포함되고, 스트림 레지스터 상태(432)는 각각의 스누프 포트 필터(400)에 회송된다. 이 상태에 기초해서, 각각의 새로운 스누프 요구(410)에 대하여, 스누프 어드레스가 로컬 캐시에 내포될 가능성이 있는지에 대한 판정이 이루어진다. 스누프 포트 필터의 제3 필터링 유닛은 범위 필터(448)이다. 이 필터링 방법을 위하여, 2개의 범위 어드레스, 즉 최소 범위 어드레스와 최대 범위 어드레스가 지정된다. 스누프 요구의 필터링은 스누프 요구가 상기 2개의 범위 어드레스에 의해 결정된 어드레스 범위 내에 있는지를 최초 결정함으로써 수행된다. 이 조건이 부합되면, 스누프 요구는 버려지고, 그렇지 않으면 스누프 요구는 결정 로직 블록(450)에 회송된다. 반대로, 요구는 어드레스 범위 내에 포함될 때 또는 그렇지 않고 버려질 때 회송될 수도 있으며, 이것은 본 발명의 범위에서 벗어나는 것이 아니다. 특히, 결정 로직 블록(450)은 각각의 개별 스누프 필터 유닛을 인에이블 또는 디스에이블하는 제어 신호(454)와 함께 3개의 필터 유닛(440, 444, 448) 모두의 결과(456)를 수신한다. 대응하는 제어 신호가 인에이블되는 스누프 필터 유닛의 결과만이 각 필터링 결정에서 고려된다. 만일 필터링 유닛(440, 444 또는 448) 중 임의의 하나가 스누프 요구(410)가 버려져야 한다고 결정하면, 스누프 요구는 버려진다. 이 유닛의 결과적인 출력은 스누프 요구를 대응하는 스누프 큐(452)에 추가하거나 스누프 요구를 버리고, 스누프 토큰(458)을 버려진 스누프 요구를 개시하였던 원격 프로세서 또는 DMA 유닛으로 되돌려 보낸다.

[0078] 양호한 실시예에서는 전술한 알고리즘을 구현하는 3개의 필터링 유닛만이 포트 스누프 필터에 포함되는 것으로 하였지만, 당업자라면, 본 발명의 범위에서 벗어나지 않고, 다른 수의 스누프 필터 유닛이 단일 포트 스누프 필터에 포함될 수 있고, 또는 임의의 다른 스누프 필터 알고리즘이 포트 스누프 필터에서 구현될 수도 있으며, 또는 스누프 알고리즘의 조합이 구현될 수 있다는 것을 알 것이다.

[0079] 도 8(a) 및 도 8(b)는 도 6의 스누프 필터 블록(310)의 2개의 대안적 실시예의 고준위 개요를 도시한 것이다. 도 6을 참조하여 설명한 바와 같이, 스누프 블록은 각종 필터링 방법, 동일한 필터링 방법, 또는 이들의 조합을 이용할 수 있는 다수의 스누프 필터를 포함할 수 있다. 도 8(a)에 도시된 바와 같이, N개의 포트 스누프 필터(460a, ..., 460n)는 N개의 원격 메모리 기록기 각각에 대해 하나씩 병렬로 동작한다. 각각의 포트 스누프 필터(460a, ..., 460n)는 점대점으로 접속된 단일 전용 소스로부터의 스누프 요구를 그 각각의 입력(462a, ..., 462n)에서 수신한다. 또한, 각각의 스누프 필터(460a, ..., 460n)는 L1 레벨 캐시에서 누락된 로컬 프로세서의 메모리 로드 요구(464)를 모두 감시한다. 또한, 스누프 블록의 다른 유닛으로부터의 다른 신호들은 만일 구현되는 필터 알고리즘에 의해 요구되면 포트 스누프 필터에 공급될 필요가 있을 수 있다. 필요로 하는 정확한 신호는 단일 포트 스누프 필터(460)에서 구현되는 하나 이상의 스누프 필터 알고리즘에 의해 결정된다. 또한, 모든 포트 스누프 필터는 동일 세트의 필터링 알고리즘을 구현하여서는 안된다는 것을 이해하여야 한다.

[0080] 포트 스누프 필터(460a, ..., 460n)는 입력되는 스누프를 필터링하고, 적당한 필터링되지 않은 스누프 요구의 부분집합을 각각의 큐(466a, ..., 466n) 및 큐 중재 블록(468)에 회송한다. 여기에서, 스누프 요구는 직렬화되고 다음 스누프 필터(470)에 제시되며, 상기 다음 스누프 필터가 모든 원격 메모리 기록기로부터의 입력을 취급한다. 이 공유 스누프 필터(470)는 제시된 모든 스누프 요구를 처리하고 모든 요구의 부분집합을 스누프 큐(472)에 회송한다. 스누프 요구는 스누프 큐(472)로부터 프로세서 스누프 인터페이스(474)를 거쳐 프로세서에 회송된다. 도 8(a)에 도시된 구성 대신에 복수의 공유 스누프 필터(470)를 가질 수도 있고 공유 스누프 필터를 전혀 갖지 않을 수도 있다는 것을 이해하여야 한다. 복수의 공유 필터를 갖는 경우, 필터는 병렬로 또는 직렬로(이 경우에 하나의 필터의 출력은 다음 필터의 입력으로 된다) 배열될 수 있다. 만일 필터가 하나 이상의 소스(즉, 복수의 소스 사이에 공유되는 것)로부터 입력을 가지면, 그 자신의 입력 큐 및 스누프 요구를 직렬화하기 위한 중재자를 가져야 한다. 모든 스누프 요구 중에서 최종 오더링된 부분집합은 스누프 큐(472)에 배치되고, 스누프 요구는 프로세서 스누프 인터페이스(474)를 거쳐 프로세서에 회송된다. 선택적으로, 스누프 큐의 스누프의 수가 미리 정해진 레벨 아래로 될 때까지 일부 또는 모든 원격 메모리 기록기가 스누프 요구를 추가로 발행하는 것을 중단시키기 위해 스누프 큐가 충만(full)될 때를 표시하는 스누프 큐 충만 표시 신호(476)가 제공된다.

[0081] 유사하게, 도 8(b)는 스누프 블록(310)에서 스누프 필터를 다르게 구성한 다른 실시예를 도시한 것이다. N개의 원격 메모리 기록기 중 하나로부터의 스누프 요구만을 각각 수신하는(즉, 스누프 필터가 부착된 프로세서를 배제함) N개의 포트 스누프 필터(480a, ..., 480n)는 병렬로 동작한다. 각각의 포트 스누프 필터(480a, ..., 480n)는 단일 소스로부터의 스누프 요구만을 그 각각의 입력(482a, ..., 482n)에서 수신한다. 공유 스누프 필터

(484)는 포트 스누프 필터 장치(480a, ..., 480n)와 병렬로 접속된다. 대안적인 실시예에서, 하나 이상의 공유 스누프 필터가 병렬로 부착될 수 있다. 공유 스누프 필터(484)는 N개의 원격 메모리 기록기 모두로부터의 입력을 취급한다. 하나 이상의 입력을 가지면, 공유 필터(484)는 스누프 요구를 직렬화하기 위한 그 자신의 입력 큐(486) 및 큐 중재자(488)를 갖는다. 또한 도 8(b)에 도시된 실시예에서, 모든 포트 스누프 필터(480a, ..., 480n)는 L1 레벨 캐시에서 누락된 로컬 프로세서로부터의 모든 메모리 판독 액세스 요구(490)를 감시한다. 스누프 필터(480a, ..., 480n, 484)는 입력되는 스누프 요구를 필터링하고, 적당한 필터링되지 않은 부분집합을 다음 공유 스누프 필터(492a, ..., 492n)의 입력 큐에 회송한다. 여기에서, 필터링되지 않은 스누프 요구는 큐 중재자(494)에 의해 직렬화되고 프로세서 스누프 인터페이스(496)를 거쳐 프로세서에 회송된다. 만일 스누프 큐 장치(492a, ..., 492n 또는 486) 중의 하나가 충만되면, 스누프 큐의 스누프의 수가 미리 정해진 레벨 아래로 될 때까지 모든(또는 일부) 원격 메모리 기록기가 스누프 요구를 추가로 발행하는 것을 중단시키기 위해 스누프 큐 충만 표시(498)가 기동된다.

[0082] 이제 도 9를 참조하면, 도 9는 스누프 필터 블록(310)의 다른 실시예를 도시한 것이다. 스누프 필터 블록은 포트 스누프 필터(400, 460a, ..., 460n, 및 480a, ..., 480n)(도 8(a), 8(b))에 대응하는 N개의 포트 스누프 필터(500a, ..., 500n)를 포함한다. 각 포트 스누프 필터(500a, ..., 500n)는 스누프 캐시 장치(502a, ..., 502n)와 스누프 체크 로직(504a, ..., 504n)을 포함한다. 스누프 캐시 장치(502a, ..., 502n)는 하나의 소스로부터의 최근 스누프 요구를 계속하여 추적하는 스누프 필터링 알고리즘을 구현한다. 상기 스누프 요구의 소스는 다른 프로세서, DMA 엔진 또는 임의의 다른 유닛일 수 있다. 단일 소스로부터의 각각의 새로운 스누프 요구에 대해서, 스누프 요구의 어드레스는 스누프 체크 논리 블록(504)의 스누프 캐시에 대해서 체크된다. 만일 이 비교의 결과가 일치하면, 즉, 스누프 요구가 스누프 캐시에서 발견되면, 스누프된 데이터는 프로세서의 로컬 L1 레벨 캐시에 있지 않는 것으로 보증된다. 따라서, 스누프 요구는 스누프 큐(506) 및 스누프 큐 중재자(508)에게 회송되지 않는다. 만일 현재 스누프 요구에 대해서 스누프 캐시(502a, ..., 502n)에서 일치가 발견되지 않으면, 스누프 요구의 어드레스는 신호(514a, ..., 514n)를 이용하여 스누프 캐시에 추가된다. 동시에, 스누프 요구는 스누프 큐(506)에 회송된다.

[0083] 모든 스누프 캐시 장치(502a, ..., 502n)는 또한 로컬 프로세서로부터 판독 어드레스 및 요구(512)를 수신하고, 메모리 판독 액세스 어드레스를 스누프 캐시(502a, ..., 502n)의 엔트리와 비교한다. 만일 요구가 스누프 캐시의 엔트리 중 하나와 일치하면, 이 엔트리는 스누프 캐시로부터 제거되고, 이제 캐시 라인이 프로세서의 제1 레벨 캐시에 위치될 것이다. 양호한 실시예에서, 병렬로 동작하는 다수의 스누프 캐시가 사용되었고, 각 스누프 캐시는 단일 원격 메모리 기록기로부터의 스누프 요구를 계속하여 추적한다. 필터링 후에, 필터링되지 않은 스누프 요구의 일부가 실시예에 따라서 다음 포트 스누프 필터에 회송되거나, 또는 하나 이상의 공유 스누프 필터용으로 큐되거나, 또는 프로세서 인터페이스의 스누프 큐에 배치될 수 있다.

[0084] 단일 스누프 캐시 장치(502)는 M개의 캐시 라인(엔트리)의 내부 조직을 포함하고, 각 엔트리는 2개의 필드, 즉 어드레스 태그 필드와 유효 라인 벡터를 갖는 것으로 이해된다. 스누프 캐시의 어드레스 태그 필드는 로컬 프로세서의 L1 캐시의 어드레스 태그와 전형적으로 동일하지 않고, 유효 라인 벡터에서 표시된 비트 수만큼 더 작다. 특히, 유효 라인 벡터는 대응하는 어드레스 태그 필드에 의해 표시된 동일한 상위 비트를 모두 공유하는 수 개의 연속적인 캐시 라인의 그룹을 인코드한다. 따라서, 어드레스로부터의 n개의 최하위 비트는 2^n개의 연속적인 L1 캐시 라인을 인코딩하기 위해 사용된다. n이 제로(0)인 극단적인 경우에, 스누프 캐시의 전체 엔트리는 단지 하나의 L1 캐시 라인을 표시한다. 이 경우, 유효 라인 벡터는 "유효" 비트에 대응하는 단지 하나의 비트를 갖는다.

[0085] 스누프 캐시에서 어드레스 태그 필드의 사이즈는 L1 캐시 라인의 사이즈 및 유효 라인 벡터를 인코딩하기 위해 사용된 비트의 수에 의해 결정된다. 예시적인 실시예에서, 32 비트의 어드레스 길이(31:0), 32 바이트 길이의 L1 캐시 라인 및 32 비트의 유효 라인 벡터에 대하여, 어드레스 비트 (31:10)이 어드레스 태그 필드로서 사용되고, (비트 31이 최상위 비트임), 어드레스 비트 (9:5)가 유효 라인 벡터에서 인코드되며, 어드레스 비트 (4:0)은 이들이 캐시 라인 바이트 옵셋을 인코드하기 때문에 무시된다. 예로서, 3개의 상이한 메모리 기록기(N=3)에 대한 3개의 스누프 캐시가 아래에 목록되어 있다. 각 스누프 캐시는 M=4의 엔트리를 가지며, 좌측에 어드레스 태그 필드가 있고, 어드레스로부터 5개의 비트가 32개의 연속적인 캐시 라인을 추적하기 위한 유효 라인 벡터를 인코드하기 위해 사용된다.

[0086] ● 스누프 요구 소스 1

[0087] 엔트리 1:**01c019e** 00000000000000000001000000000000

[0088] 엔트리 2:**01c01a0** 00000000000000000000000100000000

[0089] 엔트리 3:**01c01a2** 00000000000000000000000000010000

[0090] 엔트리 4:**01407ff** 00000000000000000000000110000000

[0091] ● 스누프 요구 소스 2

[0092] 엔트리 1:**01c01e3** 00010000000000000000000000000000

[0093] 엔트리 2:**01c01e5** 00000001000000000000000000000000

[0094] 엔트리 3:**01c01e7** 00000000000100000000000000000000

[0095] 엔트리 4:**0140bff** 00000000000000000000000110000000

[0096] ● 스누프 요구 소스 3

[0097] 엔트리 1:**01c0227** 00000000000000000001000000000000

[0098] 엔트리 2:**01c0229** 00000000000000000000000100000000

[0099] 엔트리 3:**01c022b** 00000000000000000000000000010000

[0100] 엔트리 4:**0140fff** 00000000000000000000000110000000

[0101] 이 예에서, 소스 1 스누프 캐시의 엔트리 1은 16진수 01c019ec의 어드레스가 최근에 무효화되어 L1 캐시에 있을 수 없다는 것을 기록하고 있다. 그러므로, 동일 캐시 라인에 대한 다음 스누프 요구가 걸러내질 것이다(버려짐). 유사하게, 소스 1 스누프 캐시의 엔트리 4는 캐시 라인 어드레스 01407ff7 및 01407ff8에 대한 스누프 요구가 걸러내지게 할 것이다.

[0102] 이제 도 10을 참조하면, 단일 스누프 소스에 대해 스누프 캐시 장치를 구현하는 스누프 필터의 제어 흐름이 도시되어 있다. 동작의 시작부분에서, 스누프 캐시의 모든 M개의 라인이 리세트된다(단계 600). 스누프 소스 i로부터 새로운 스누프 요구가 수신된 때, 스누프 요구의 어드레스는 "어드레스 태그" 필드(526)로 및 유효 라인 벡터(524)를 액세스하기 위해 사용된 비트로 해부(parse)된다. 스누프 요구의 유효 라인 벡터는 어드레스 태그 필드와 일치하는 어드레스 비트를 가진 각각의 L1 캐시에 대응하는 하나의 비트만을 갖는다. 이것은 단계 602에서 수행된다. 단계 604에서, 스누프 요구의 "태그" 필드는 스누프 소스 i와 관련된 스누프 캐시의 모든 태그 필드에 대해서 체크된다. 만일 스누프 요구 어드레스 태그가 스누프 캐시에 저장된 어드레스 태그 중의 하나와 동일하면, 어드레스 태그 필드는 스누프 캐시에서 히트(hit)한다. 그 다음에, 히트가 검출된 스누프 캐시 엔트리의 유효 라인 벡터가 스누프 요구의 유효 라인 벡터와 비교된다. 만일 스누프 요구의 유효 라인 벡터의 비트 집합에 대응하는 스누프 캐시 라인의 유효 라인 벡터의 비트가 설정되면, 유효 라인 벡터도 또한 히트한다. 하나의 양호한 실시예에서, 유효 라인 벡터 체크는 비트 피연산자(operand)에 대해서 논리 연산을 수행함으로써 구현된다. 따라서, 예를 들면, 유효 라인 벡터 체크는 스누프 요구의 유효 라인 벡터와 스누프 캐시 라인의 유효 라인 벡터를 AND 연산하고, 결과가 0인지를 체크함으로써 수행될 수 있다. 본 발명의 범위에서 벗어나지 않고 다른 구현예가 추가적으로 사용될 수 있다. 또한, 유효 라인 벡터 히트의 체크는 어드레스 태그 히트의 체크와 병렬로 구현될 수 있다.

[0103] 단계 606에서, "태그" 필드가 일치하는지 및 유효 라인 벡터의 대응하는 비트가 세트되었는지에 대한 판정이 이루어진다. 만일 "태그" 필드가 일치하고 유효 라인 벡터의 대응 비트가 설정되었으면, 스누프 요구는 캐시에 있지 않는 것으로 보증된다(단계 606). 따라서, 이 스누프 요구는 캐시에 회송되지 않고 걸러내진다(단계 608).

[0104] 그렇지 않고, 만일 어드레스 "태그" 필드가 스누프 캐시에서 히트하지만 유효 라인 벡터의 비트가 설정되어 있지 않으면, 또는 대안적으로 만일 태그가 스누프 캐시에서 히트하지 않으면, 이것은 라인이 캐시에 있을 수 있다는 것을 표시한다. 결과적으로, 스누프 요구는 이 요구를 스누프 큐에 배치함으로써 캐시에 회송된다(단계 612). 이 스누프 요구는 또한 새로운 엔트리로서 스누프 캐시에 추가된다(단계 610).

[0105] 이제 도 11을 참조하면, 스누프 캐시에 새로운 정보를 추가하는 처리를 나타내는 단계 610(도 10)의 상세가 도시되어 있다. 이것은 이제부터 설명할 몇가지 태스크에 의해 달성된다. 단계 614에서, 어드레스 태그가 스누프 캐시에 이미 저장되어 있는지(즉, 어드레스 태그가 히트하였는지)에 대한 판정이 먼저 이루어진다. 이 단계에서, 단계 602(도 10)에서 계산된 정보가 사용될 수 있다. 어드레스 태그 체크가 히트로 나타났으면, 처리

는 단계 624로 진행하고, 이 단계에서는 스누프 요구에 대응하는 선택된 스누프 캐시 엔트리의 유효 라인 벡터의 비트가 설정된다. 만일 단계 614에서 어드레스 태그 체크가 누락을 나타내면, 새로운 스누프 캐시 엔트리가 새로운 어드레스 태그에 대해 할당되어야 하고, 처리는 단계 616으로 진행하여 스누프 캐시에서 가용인 빈 엔트리가 있는지에 대한 판정이 이루어진다. 만일 가용인 빈 엔트리가 있다고 판정되면, 제1의 가용인 빈 엔트리가 선택된다(단계 620). 그렇지 않고, 만일 스누프 캐시에서 빈 엔트리가 없다고 판정되면, 스누프 캐시에서 능동 엔트리 중의 하나가 교체용으로 선택된다(단계 618). 교체 정책은 라운드 로빈, 최근 사용된 것, 무작위(random), 또는 본 발명을 벗어나지 않고 당업계에 알려진 임의의 다른 교체 정책일 수 있다. 단계 622에 이어서, 새로운 어드레스 태그는 선택된 스누프 캐시 라인에 기록되고 대응하는 유효 라인 벡터가 클리어된다. 그 다음에, 단계 624에 표시된 바와 같이, 스누프 요구의 유효 라인 벡터에서 설정된 비트에 대응하는 선택된 스누프 캐시 엔트리의 유효 라인 벡터의 비트가 설정된다.

[0106] 또 다른 실시예에서, 새로운 정보가 스누프 캐시의 스누프 요구의 히트 또는 누락에 기초하여 스누프 캐시에 추가되지 않고, 그 대신, 전체 스누프 캐시 라인이 되거나 유효 라인 벡터의 단일 비트를 설정하는 새로운 값의 추가가 결정 논리 블록(450)(도 7)의 결정에 기초하여 행하여진다. 이 실시예에서, 새로운 정보는 결정 논리 블록이 스누프 요구를 걸러내지 않은 경우에만 스누프 캐시에 추가된다. 스누프 포트 필터 블록(400)(도 7)의 임의의 다른 필터가 스누프 요구를 걸러내면(즉, 데이터가 로컬 L1 캐시에 없다고 판정하면), 스누프 캐시에는 새로운 정보가 추가되지 않고, 연산 단계는 스누프 캐시 히트의 경우와 동일하게 된다. 이 실시예의 장점은 더 적은 용장성 정보가 저장되기 때문에 스누프 캐시가 더 잘 수행한다는 점이다.

[0107] 이제, 도 12를 참조하면, 스누프 캐시로부터 엔트리를 제거하는 제어 흐름이 도시되어 있다. 로컬 L1 레벨 캐시에서 누락한 각각의 로컬 프로세서 메모리 판독 요구에 대해서, 메모리 요구의 어드레스는 모든 스누프 요구 소스와 관련된 모든 스누프 캐시 내의 모든 엔트리에 대하여 체크된다. 단계 630에서, 메모리 판독 요구의 어드레스는 어드레스 태그 필드로 및 유효 라인 벡터를 인코딩하기 위한 비트로 해부된다. 이것은 단계 630에서 수행된다. 단계 632에서는 하나 이상의 태그 히트가 있는지 여부가 판정된다. 이것은 모든 스누프 소스와 관련된 모든 스누프 캐시의 모든 태그 필드에 대해서 메모리 요구의 "태그" 필드를 체크함으로써 달성된다. 만일 태그 체크가 누락하면, 이 어드레스는 걸러내지지 않고 아무것도 행하여지지 않는다. 따라서, 제어 흐름은 단계 630으로 되돌아가서 프로세서로부터의 다음 캐시 누락을 기다린다.

[0108] 단계 632로 되돌아가서, 만일 어드레스 태그와 모든 스누프 캐시와의 비교 결과 하나 이상의 히트가 있다고 판정되면, 히트가 있는 모든 스누프 캐시로부터 정보가 제거된다. 따라서, 단계 634에서, 메모리 판독 어드레스의 적당한 낮은 차수 비트가 유효 라인 벡터로 디코드되고, 단계 635에 표시된 것처럼, 히트인 스누프 캐시 엔트리의 유효 라인 벡터와 일치된다. 처리는 이제 단계 636으로 진행하고, 이 단계에서는 판독 어드레스 벡터에서 설정된 유일한 비트가 스누프 캐시의 유효 라인 벡터에서도 또한 설정되는지 판정된다. 만일 이러한 유효 라인 벡터 히트가 없으면(어드레스 태그 필드 히트와 상관없이), 이 메모리 어드레스는 걸러내지지 않고 특수 스누프 캐시에서 아무것도 변경되지 않는다. 따라서, 제어 흐름은 단계 640으로 진행하여 모든 어드레스 태그 히트가 처리되었는지가 체크되고, 만일 처리되지 않았으면, 처리는 단계 635로 복귀한다.

[0109] 그러나, 만일 단계 636에서 유효 라인 벡터에서 판독 어드레스 벡터가 히트라고 판정되면, 판독 어드레스가 걸러내진다. 대응하는 유효 라인 벡터 비트는 메모리 판독 어드레스가 제1 레벨 캐시에 로드되려고 하기 때문에 클리어된다. 유효 라인 벡터에서 대응 비트의 이러한 클리어는 단계 638에서 수행된다. 만일 유효 라인 벡터로부터 대응 비트를 제거한 후에 유효 라인 벡터의 비트 세트의 수가 0으로 되면, 어드레스 태그 필드는 스누프 캐시로부터 추가로 제거되어 엔트리를 비어있게 한다. 다음 단계 640에서 표시된 바와 같이, 유효 라인 벡터 비트의 체크, 그 클리어, 및 만일 필요하다면 어드레스 태그의 클리어에 관한 동일한 처리가 로컬 L1 캐시에서 누락된 메모리 판독 요구를 히트한 모든 스누프 캐시에 대해서 반복된다. 모든 히트인 어드레스 태그 라인이 처리되는 이러한 조건은 단계 640에서 체크된다. 일단 모든 캐시 라인이 체크되면, 처리는 단계 630으로 복귀한다.

[0110] 또 다른 실시예에서, 로컬 메모리 요구는 모든 스누프 캐시의 모든 어드레스 태그와 동시에 비교된다. 이와 동시에, 로컬 메모리 요구의 유효 라인 벡터 인코딩은 동시에 히트한 모든 스누프 캐시의 모든 유효 라인 벡터와 비교될 수 있다. 그 다음에, 상기 2개의 결과, 즉 어드레스 태그 히트와 유효 라인 벡터 히트는 결합되어 대응하는 유효 라인 벡터 비트가 제거되어야 하는 모든 스누프 캐시 라인을 결정하고, 이러한 모든 비트들은 모든 스누프 캐시로부터의 히트하는 캐시 라인에서 동시에 제거될 수 있다.

[0111] 이제 도 13을 참조하면, 스트림 레지스터를 구현하는 스누프 필터 장치의 블록도가 도시되어 있다. 하나의 양호한 실시예에서, 스누프 필터 유닛은 2세트의 스트림 레지스터 및 마스크(700), 스누프 체크 논리 블록(702), 캐

시 랩 검출 논리 블록(706), 스트림 레지스터 선택 논리 블록(704), 필터 큐(703) 및 프로세서 중재 및 다중화 논리 블록(710)을 포함한다. 여기에서 상세히 설명되는 바와 같이, 캐시에 없는 것을 계속하여 추적하는 스누프 캐시 필터와 달리, 스트림 레지스터 및 마스크 세트(700)는 프로세서의 캐시에 로드된 최근 데이터를 계속하여 추적한다. 구체적으로 말하면, 스트림 레지스터는 최소한 캐시에 있는 라인들을 계속하여 추적하지만, 실제로 캐시에 있지 않은 일부 라인들이 캐시되는 것으로 가정할 수 있다. 그러나, 일부 불필요한 스누프 요구를 캐시에 회송하는 것은 정확성에 영향을 주지 않는다.

[0112] 스트림 레지스터 필터의 핵심은 스트림 레지스터(700) 자체이다. 이 레지스터들 중의 하나는 캐시가 새로운 라인을 로드할 때마다 업데이트되고, 이것은 적당한 제어 신호(716)에 의해 스트림 레지스터에 제시된다. 도 13의 논리 블록(704)은 현재 스트림 레지스터 상태, 및 신호(716)로 캐시에 로드되는 새로운 라인의 어드레스에 기초하여 업데이트 할 특수 레지스터를 선택할 책임을 갖는다.

[0113] 동작시에, N개의 원격 프로세서 중의 하나로부터 수신된 스누프 요구는 도 14의 우측에 도시된 신호(714)로서 도달한다. 스누프 체크 논리 블록(702)은 도달하는 스누프 요구(714)의 어드레스를 스트림 레지스터(700)의 상태와 비교하여 스누프 요구가 캐시 내에 있을 가능성이 있는지를 판정하는 포트 필터의 집합을 포함한다. 만일 가능성이 있으면, 요구는 큐(703)로 회송되고, 요구들은 실제 캐시 스누프로서 캐시에 회송되기를 기다린다. N개의 원격 프로세서가 각각 전용 스누프 요구 큐(703)를 갖는 도 13의 큐잉 구조는 다수의 스누프 요구가 걸러내지고 엔큐(enqueued)될 필요가 없기 때문에 최대 스누프 요구율이 가능하도록 설계된다. 대안적인 큐잉 구조는 본 발명의 일반적인 범위를 벗어나지 않고도 가능하다.

[0114] 중재 및 다중화 논리 블록(710)은 공정한 방식으로 N개의 스누프 요구 큐(703) 사이에서 캐시의 스누프 인터페이스를 단순히 공유하여 모든 요구의 순방향 진행을 보증한다.

[0115] 이제, 단일 스트림 레지스터가 업데이트되는 방법에 대하여 설명한다. 스트림 레지스터는 실제로 한 쌍의 레지스터, 즉 "베이스" 레지스터와 "마스크" 레지스터, 및 유효 비트를 포함한다. 베이스 레지스터는 스트림 레지스터에 의해 표시된 모든 캐시 라인에 공통인 어드레스 비트를 계속하여 추적하고, 대응하는 마스크 레지스터는 이들이 어떤 비트인지를 계속하여 추적한다. 유효 비트는 스트림 레지스터가 사용중에 있고, 원격 스누프 요구(714)를 걸러내기로 결정한 때 스누프 체크 논리 블록(702)에 의해 참고되어야 한다는 것을 단순히 표시한다. 이하의 설명에서의 예를 이해하기 위해, 32 바이트의 캐시 라인 크기를 가진 2^{32} 바이트의 어드레스 공간을 생각하자. 이 경우, 캐시 라인 로드 어드레스는 길이가 27 비트이고, 스트림 레지스터의 베이스 레지스터와 마스크 레지스터도 또한 길이가 27 비트이다.

[0116] 초기에, 유효 비트는 0으로 설정되어 스트림 레지스터가 사용중이 아니고 베이스 및 마스크 레지스터의 콘텐츠가 무관하다는 것을 표시한다. 제1 캐시 라인 로드 어드레스가 이 스트림 레지스터에 추가되는 경우, 유효 비트는 1로 설정되고, 베이스 레지스터는 라인 어드레스로 설정되며, 마스크 레지스터의 모든 비트들은 1로 설정되어 베이스 레지스터의 모든 비트들이 중요하다는 것을 표시한다. 즉, 베이스 레지스터에 저장된 어드레스와 정확히 일치하는 어드레스는 캐시에 있는 것으로 고려되고, 임의의 비트 또는 비트들에서 상이한 어드레스는 캐시에 없는 것으로 고려된다. 예를 들면, 제1 캐시 라인 로드 어드레스가 0x1708fb1(0x 접두사는 16진수를 표시한다)로 주어졌다고 가정한다. 이 경우, 로드 후의 스트림 레지스터의 콘텐츠는 다음과 같다.

[0117] 베이스=0x1708fb1 마스크=0x7ffffff 유효=1

[0118] 결국, 제2 캐시 라인 로드 어드레스가 이 스트림 레지스터에 추가되는 경우, 제2 어드레스는 베이스 레지스터와 비교되어 비트가 상이한지 결정한다. 그 다음에, 마스크 레지스터는 상이한 비트 위치가 마스크에서 0으로 되도록 업데이트된다. 따라서, 이 0들은 베이스 레지스터의 대응하는 비트가 "돈케어"(don't care), 즉 임의의 값(0 또는 1)을 취할 수 있는 것임을 표시한다. 그러므로, 이 비트들은 스트림 레지스터와의 비교에 있어서 더 이상 중요하지 않다. 예를 들면, 제2 캐시 라인 로드 어드레스는 0x1708fb2라고 말할 수 있다. 이 제2 로드 후의 스트림 레지스터의 콘텐츠는 다음과 같다.

[0119] 베이스=0x1708fb1 마스크=0x7ffffc 유효=1

[0120] 다시 말해서, 제2 어드레스와 베이스 레지스터는 2개의 최하위 비트에서 상이하고, 이 비트들이 마스크 레지스터에서 클리어되게 한다. 이 점에서, 스트림 레지스터는 2개의 최하위 비트를 더 이상 구별할 수 없기 때문에, 어드레스 0x1708fb0, 0x1708fb1, 0x1708fb2, 및 0x1708fb3이 모두 캐시에 있을 수 있다고 표시한다. 그러나, 실제로 로드된 2개의 어드레스가 캐시에 있는 것으로 고려된다는 것을 아는 것은 중요하다. 따라서, 이 메카니즘은 스트림 레지스터에 제시된 모든 어드레스는 그 내부에 포함될 것을 보증한다. 제한으로, 마스크 레지스터

는 모두 0으로 되고, 모든 가능한 어드레스는 레지스터에 포함되며, 캐시에 있는 것으로 고려된다. 명백하게, 여기에서 설명하는 메카니즘은 어드레스를 스트림 레지스터에 계속하여 추가하기 위해 사용될 수 있다.

[0121] 각각의 캐시 라인 로드 어드레스는 다수의 스트림 레지스터 중 정확히 하나에 추가된다. 그러므로, 스트림 레지스터의 수집은 완전한 캐시 상태를 나타낸다. 어떤 레지스터를 업데이트할 것인지에 관한 결정은 도 13의 업데이트 선택 논리 블록(704)에 의해 행하여진다. 하나의 가능한 선택 기준은 라인 로드 어드레스로부터 최소 해밍 거리(Hamming distance)에 의해 스트림 레지스터(즉, 최소수의 마스크 레지스터 비트가 0으로 변화하는 스트림 레지스터)를 선택하는 것이다. 또 다른 선택 기준은 베이스 레지스터의 최상위 비트가 라인 로드 어드레스의 비트들과 일치하는 스트림 레지스터를 선택하는 것이다. 다른 선택 기준도 가능하고, 본 발명의 범위에서 벗어나지 않고 구현될 수 있다.

[0122] 업데이트할 스트림 어드레스 레지스터를 선택함에 있어서, 라인 로드 어드레스는 대응하는 마스크 레지스터와 병렬로 결합된 모든 베이스 레지스터와 비교된다. 그 다음에, 라인 로드 어드레스는 여기에서 설명하는 바와 같이 선택된 스트림 레지스터에 추가된다.

[0123] 스누프 체크 논리 블록(702)은 스누프 어드레스(714)를 다음과 같이 모든 스트림 레지스터와 비교함으로써 스누프 어드레스가 캐시 내에 있을 수 있는지를 판정한다. 즉, 스누프 어드레스(714)는 캐시 라인 내에서 옵셋에 대응하는 낮은 차수의 비트를 제거함으로써 라인 어드레스로 변환된다. 이 라인 어드레스는 베이스 레지스터와 스누프 라인 어드레스 사이에서 비트 논리 배타적 OR를 수행하고, 그 다음에 그 결과와 마스크 레지스터를 비트 논리 AND함으로써 단일 스트림 레지스터와 비교된다. 만일 상기 2개의 논리 연산의 최종 결과가 0이 아닌 임의의 비트를 가지면, 스누프 어드레스는 스트림 레지스터에서 "누락"하였고, 스트림 레지스터가 관계되는 한 캐시에 없다고 알려진다. 동일한 비교가 모든 스트림 레지스터에 대해 병렬로 수행되고, 만일 스누프 라인 어드레스가 모든 스트림 레지스터에서 누락하면, 스누프 어드레스는 캐시에 없는 것으로 알려지고 걸러내질 수 있다(즉, 캐시에 회송되지 않는다). 반대로, 만일 스누프 어드레스가 스트림 레지스터 중 임의의 하나에서 히트이면, 스누프 레지스터는 캐시에 회송되어야 한다.

[0124] 스누프 체크 논리 블록(702)은 N개의 원격 스누프 요구 포트 각각에 대해서 이중화되지만, 이들은 모두 동일한 스트림 레지스터(700) 세트를 공유한다.

[0125] 시간이 경과함에 따라, 캐시 라인 로드 어드레스가 스트림 레지스터에 추가되기 때문에, 이들은 무엇이 실제로 캐시에 있는지에 관한 지식면에서 점점 덜 정확하게 된다. 상기 예에서 설명한 것처럼, 0으로 되는 각 마스크 비트는 대응하는 스트림 레지스터가 캐시에 있는 것으로서 지정하는 캐시의 수를 2의 계수만큼 증가시킨다. 일반적으로, 무용의 스누프 요구를 프로세서에 회송하는 문제점(즉, 이들을 필터링하는데 실패함)은 0인 마스크 비트의 수가 증가함에 따라 더 나빠진다. 그러므로, 스트림 레지스터 스누프 필터에는 레지스터를 초기 조건으로 재순환시키기 위한 메카니즘이 제공된다. 이 메카니즘은 일반적으로 캐시에 로드된 라인들을 이미 그곳에 있는 라인들로 교체하는 관측에 기초를 두고 있다. 라인이 교체될 때마다, 라인은 스트림 레지스터로부터 제거될 수 있는데, 그 이유는 스트림 레지스터가 어떤 라인이 캐시에 있는지만을 추적하기 때문이다. 개별 라인들을 제거하는 대신에, 스트림 레지스터 스누프 필터는 제거되는 라인들을 효과적으로 한 묶음으로 묶어서 캐시가 완전히 교체될 때마다 레지스터를 클리어한다. 그러나, 이러한 교체를 행하는 새로운 캐시 라인은 스트림 레지스터에 또한 추가될 수 있고, 따라서, 그 레지스터의 콘텐츠들은 단순히 버려질 수 없다.

[0126] 이러한 딜레마를 해결하기 위해, 스트림 레지스터 스누프 필터는 초기 캐시 상태로 시작하는 동작을 수행하고, 스트림 레지스터 업데이트는 위에서 설명한 바와 같이 발생한다. 캐시 랩 검출 논리 블록(706)에는 캐시 업데이트 신호(717)에 의해 표시된 캐시 업데이트를 감시하는 기능 및 초기 상태로 나타나는 모든 캐시 라인이 새로운 라인으로 덮어쓰기되는 때, 즉 캐시가 "랩"되는 때를 결정하는 기능이 주어진다. 그 점에서, 모든 스트림 레지스터의 콘텐츠(이들을 "능동" 세트라고 부른다)는 스트림 레지스터의 제2 "히스토리" 세트에 복사되고, 능동 세트의 스트림 레지스터는 모두 무효 상태로 복귀되어 캐시 라인 로드 어드레스의 축적을 다시 시작한다. 또한, 랩(wrap)시의 캐시 상태는 다음 캐시 랩을 검출하기 위하여 새로운 초기 상태로 된다. 히스토리 세트 내의 스트림 레지스터는 업데이트되지 않는다. 그러나, 그 스트림 레지스터들은 스누프 어드레스가 캐시에 있는지 여부를 결정하는 때 스누프 체크 논리 블록(702)에 의해 능동 세트와 동일하게 취급된다. 이 메카니즘으로, 스트림 레지스터는 캐시가 덮어쓰기 될 때 주기적으로 재순환된다.

[0127] 캐시 업데이트 정책 및 캐시 업데이트 신호(717)에 따라서 캐시 래핑을 검출하는 많은 방법이 있다. 예를 들어서, 캐시가 덮어쓰기되는 라인을 지정하면, 단순 스코어보드를 이용하여 임의의 특수 라인이 덮어쓰기되는 최초 시간을 결정할 수 있고, 카운터를 이용하여 모든 라인이 적어도 1회 덮어쓰기되는 때를 결정할 수 있다. 캐시

래핑을 검출하기 위한 어떠한 메카니즘도 본 발명의 범위를 벗어나지 않고 사용될 수 있다.

[0128] 도 14는 필터가 N개의 원격 프로세서에 의해 완전하게 공유되는 스트림 레지스터 스누프 필터의 다른 실시예를 도시한 것이다. 즉, 개별 스누프 요구 포트(714)는 도 13을 참조하여 설명한 실시예에서 도시된 바와 같이 그들 자신의 스누프 체크 논리 블록(702)을 갖지 않는다. 이 실시예에서, 스누프 요구는 공유된 스누프 체크 논리 블록(701)에 입력되기 전에 큐 구조로 엔큐(enqueue)된다. 큐된 요구는 중재 및 다중화 논리 블록(705)를 통해 스누프 체크 논리 블록(701)에 공정한 방식으로 회송된다. 스누프 체크 논리 블록(701)의 기능은 상기 사항을 제외하면 도 13을 참조하여 위에서 설명한 이전 스트림 레지스터 스누프 필터 체크 로직과 동일하다. 명백하게, 대안적인 큐잉 구조(708)가 가능하고 본 발명의 일반적인 범위로부터 벗어나지 않는다.

[0129] 양호한 실시예에서는 2세트의 스트림 레지스터가 사용되지만, 2 이상의 세트도 본 발명의 범위로부터 벗어나지 않고 사용될 수 있다. 예를 들면, 4세트의 스트림 레지스터를 구현하는 실시예에서는 2세트의 능동 레지스터 A와 B 및 2세트의 대응하는 히스토리 레지스터가 구현된다. 이 실시예에서, A세트의 스트림 레지스터는 캐시의 하나의 부분집합에 관한 정보를 포함하고, B세트의 스트림 레지스터는 캐시의 다른 부분집합에 관한 정보를 포함할 수 있다. 캐시를 각 세트의 스트림 레지스터 A와 B에 할당된 부분으로 분할하는 것은 캐시를 2개의 동일한 부분으로 나눔으로써 수행될 수 있지만, 다른 분할도 사용될 수 있다. 또한, 스트림 레지스터 세트의 수는 2 이상도 가능하다. 예를 들면, 세트 관련 캐시의 각 캐시 세트에 할당된 1세트의 스트림 레지스터가 있을 수 있다.

[0130] 또 다른 실시예에서, 스트림 레지스터의 하나 이상의 히스토리 세트를 구비하여 능동 세트가 더욱 빈번하게 재순환되게 할 수 있다. 그러나, 레지스터에 의해 커버되는 캐시 라인이 아직 캐시에 있을 때 레지스터가 클리어되지 않도록 캐시 랩 검출에 관한 히스토리 레지스터를 관리하는데 있어서 주의를 기울여야 한다. 레지스터가 클리어되지 않게 보증하는 하나의 방법은 히스토리 레지스터를 능동 세트의 스트림 레지스터에 추가하고, 그 다음에 캐시가 랩될 때 그 히스토리 레지스터(및 능동 레지스터) 모두를 제2 세트의 히스토리 레지스터에 복사하는 것이다. 이것은 본질적으로 히스토리의 제2 "영역(dimension)"을 여기에서 설명하는 바와 같이 스트림 레지스터 스누프 필터의 양호한 실시예에 추가한다.

[0131] 이제 도 15를 참조하면, 쌍을 이룬 베이스 레지스터와 마스크 레지스터 세트를 이용하는 스누프 필터에 대한 제어 흐름의 상세한 처리 흐름도가 도시되어 있다. 동작의 시작시에, 모든 스트림 레지스터와 마스크 및 스누프 큐는 리세트되고(단계 730), 시스템은 임의의 스누프 소스로부터의 다음 스누프 요구를 기다린다(단계 732). 새로운 스누프 요구가 수신되는 경우, 스누프 요구의 어드레스는 모든 어드레스 스트림 레지스터 및 마스크(스트림 레지스터의 양 세트)에 대해서 체크된다(단계 734). 스누프 요구의 어드레스는 수반된 마스크와 함께 모든 스트림 레지스터(모든 어드레스 스트림 레지스터 및 마스크(스트림 레지스터의 양 세트))에 대해서 체크된다. 현재 스누프 요구의 비교 결과가 쌍을 이룬 마스크 레지스터와 결합된 스트림 레지스터와 일치한다고 단계 736에서 결정되면, 스누프된 캐시 라인은 캐시 내에 있을 것이고, 스누프 요구는 스누프 요구를 스누프 큐에 배치함으로써 캐시로 회송된다(단계 740). 처리는 단계 732로 되돌아가서 다음 스누프 요구를 기다린다. 그러나, 만일 스누프 요구가 스트림 레지스터의 양 세트에서 쌍을 이룬 마스크 레지스터와 결합된 어떠한 스트림 레지스터와도 일치하지 않으면, 스누프된 캐시 라인은 캐시에 없는 것으로 보증된다. 따라서, 이 스누프 요구는 단계 738에서 걸러내지고, 처리는 단계 732로 복귀한다.

[0132] 이제 도 16을 참조하면, 교체된 캐시 라인용의 2개의 스트림 레지스터 세트 및 캐시 랩 검출 논리 블록을 업데이트하기 위한 제어 흐름이 도시되어 있다. 동작의 시작시에, 모든 스트림 레지스터와 마스크는 리세트되고 캐시 랩 검출 로직이 클리어되며(단계 750), 제1 세트의 레지스터가 기동된다. L1 캐시에서 누락한 각 프로세서 메모리 요구(로드 또는 저장 동작을 포함한다)에 대해서, 메모리 요구의 어드레스는 능동 어드레스 스트림 레지스터 세트라고 언급되는 제1 세트의 스트림 레지스터에 추가된다. 제1 세트의 레지스터로부터의 모든 어드레스 스트림 레지스터는 구현된 레지스터 선택 기준에 의해 지정된 최상의 정합(match)을 선택하기 위해 체크되고, 제1의 빈 스트림 레지스터가 선택될 수 있다. 메모리 요구의 어드레스는 단계 752에 표시된 것처럼 능동 레지스터 세트 내의 선택된 스트림 어드레스 레지스터에 저장되고, 쌍을 이룬 마스크는 어드레스 중의 어떤 비트가 관련되고 어떤 비트가 관련되지 않은지를 반영하기 위해 업데이트된다. 그 다음에, 단계 754에서, 캐시 랩 검출 로직이 캐시에 로드된 새로운 데이터를 반영하도록 업데이트된다. 캐시 랩 검출 블록은 캐시의 모든 라인들이 능동 레지스터의 최초 사용이 개시된 이후 교체되었는지를 계속하여 추적한다. 따라서, 단계 756에서, 캐시 랩 조건이 존재하는지 여부에 대한 판정이 행하여진다. 만일 단계 756에서 캐시 랩 조건이 검출되지 않으면, 제어 흐름은 단계 752로 되돌아가서 시스템이 다음 프로세서 메모리 요구를 기다린다. 그렇지 않고 캐시 랩 조건이 검출되면, 제어는 단계 758로 진행하여 캐시 랩 검출 논리 블록이 클리어되고 제2 스트림 레지스터 및 마스크 세트가 클리어된다(단계 758). 다음 단계 760으로 진행해서, 시스템은 다음 프로세서 메모리 요구를 기다린다.

새로운 메모리 요구에 대해서, 제2 레지스터 세트로부터의 모든 어드레스 스트림 레지스터는 예를 들면 구현된 레지스터 선택 기준에 의해 지정된 최상의 정합을 선택하기 위해 체크되고, 예를 들면, 제1의 빈 스트림 레지스터가 선택된다. 메모리 요구의 어드레스는 제2 레지스터 세트의 선택된 스트림 어드레스 레지스터에 저장되고(단계 760), 쌍을 이룬 마스크는 어드레스의 어떤 비트가 관련되는지를 반영하기 위해 업데이트된다. 단계 762로 진행해서, 캐시 랩 검출 로직은 캐시에 로드된 새로운 데이터를 반영하도록 업데이트된다. 캐시 랩 검출 로직은 제2 레지스터 세트의 최초 사용이 개시된 이후 교체된 캐시의 모든 라인을 계속하여 추적하기 때문에, 캐시 랩 조건이 존재하는지 여부에 대한 판정이 단계 764에서 이루어진다. 만일 단계 764에서 캐시 랩 이벤트가 검출되지 않으면, 시스템은 단계 760으로 복귀하여 다음 프로세서 메모리 요구를 기다린다. 그러나, 만일 캐시 랩 이벤트가 검출되면, 제1 세트의 레지스터와 마스크를 다시 사용할 것이다. 따라서, 제1 레지스터 세트로부터의 모든 레지스터 및 쌍을 이룬 마스크가 리세트되고, 캐시 랩 검출 로직은 단계 766에서 클리어된다. 제1 레지스터 세트는 캐시의 콘텐츠를 근사화하기 위한 능동으로서 다시 사용될 것이고, 제어 흐름은 단계 752로 되돌아간다.

[0133] 스트림 레지스터 스누프 필터의 사용과 관련하여 여기에서 설명한 것처럼, 스누프 요구를 차단하기 위한 각 스트림 레지스터 필터의 전력은 0으로 설정된 마스크 비트의 수가 증가함에 따라 감소한다. 예를 들어서, 모든 마스크 비트가 0이면, 모든 스누프 요구는 그를 통하여 보내져야 한다. 그러나, 이들 마스크 비트가 한번에 하나의 비트만 0으로 설정된다고 가정하면(즉, 각 로드는 1비트 만큼만 스트림 레지스터와 다르다), 그 경우에, 스트림 레지스터와 정확히 2비트가 상이한 어드레스에 대한 스누프 요구는 비록 이 어드레스가 캐시에 있을 수 없더라도 통하게 할 것이다. 따라서, 추가적인 필터링 능력은 상이한 비트의 수와 같은 더 복잡하거나 미묘한 차이를 검출할 수 있게 하는 시그너쳐 필터(signature filter)를 구현함으로써 제공된다. 일반적인 아이디어는 마스크 필터와 시그너쳐 필터가 둘 다 어드레스가 캐시에 있음을 표시하는 경우에만 스트림 레지스터로부터 스누프가 회송되는 것이다.

[0134] 도 17을 참조하면, 어드레스(901)와 스트림 레지스터(902)를 입력으로서 취하고 스트림 레지스터에 관한 어드레스의 시그너쳐(903)를 연산하는 시그너쳐 함수(900)가 도시되어 있다. 가능한 시그너쳐 함수로는 다음과 같이 여러가지가 있다.

[0135] 1. 스트림 레지스터 어드레스와 상이한 어드레스의 비트 수. 이 수를 s라고 표시한다. 잘라버림(truncation)은 공간을 절약하기 위해 사용될 수 있다. 예를 들면, 시그너쳐를 임의의 상수 M에 대해서 min(M,s)로 설정한다.

[0136] 2. 어드레스가 N비트 길이를 가지면, 시그너쳐는 s=i 일 때 비트 i가 1인 것을 제외하고 모든 비트가 0인 길이 B=(N+1) 비트인 벡터이다. 공간을 절약하기 위해, 이것은 길이 B+1(B+1<N)인 벡터로 잘라버림될 수 있고, 이때, min(s,B)=i이면 비트 i가 1이다.

[0137] 3. 어드레스를 k(k>1) 그룹의 비트로 나눈다. 그룹 i의 길이는 L(i) 비트이고 M(i)=L(i)+1로 한다. 그룹 i의 스트림 레지스터 비트와 상이한 그룹 i의 어드레스 비트의 수를 s(i)라고 한다. 이때, 시그너쳐는 (s(1), s(2)..., s(k))로 주어지고, 이것은 단순히 각 그룹에서 상이한 비트의 수이다. 이 그룹들은 해체된 비트의 세트, 또는 부분적으로 중첩된 비트의 세트(즉, 어드레스의 일부 비트는 하나 이상의 그룹에 있다)로 구성될 수 있다. 시그너쳐의 길이는 B(1)+...+B(k) 비트이고, 여기에서 B(i)는 s(i)의 모든 가능한 값을 표시하기 위해 요구되는 비트의 수이다.

[0138] 4. 시그너쳐가 각 그룹에 대응하는 k 비트 벡터로 구성되는 상기 (2)와 (3)의 조합. 그룹 j의 비트 i는 만일 s(j)=i이면 1로 설정된다. 만일 그룹 i가 길이 L(i) 비트를 가지면, s(i)의 모든 가능한 값을 인코드하기 위해 M(i)=(L(i)+1) 비트가 필요하다. 시그너쳐는 M(1)+...+M(k) 비트 길이이다. 잘라버림은 공간을 절약하기 위해 사용될 수 있다. 예를 들면, 그룹 j의 비트 i는 만일 임의의 상수 M에 대해서 min(M,s(j))=i이면 1로 설정된다.

[0139] 5. 상기 (3)에서와 마찬가지이지만, 여기에서는 s(1),..., s(k)의 M(1)*...*M(k)개의 다른 독특한 조합이 있다. 각 조합에는 정수 q가 할당되고, 시그너쳐는 비트 q가 1인 것을 제외하고 모두 0인 벡터로 설정된다. 잘라버림은 상기 (4)에서처럼 공간을 줄일 수 있다.

[0140] 6. 어드레스를 k(k>1)개의 비트 그룹으로 나누고 p(i)를 그룹 i의 어드레스 비트의 패리티로 한다. 이때, 시그너쳐는 (p(1), p(2),.....,p(k))로 주어진다.

[0141] 7. 상기 (6)에서와 마찬가지이지만, 2^k개의 패리티 조합 각각을 정수 q에 대해서 인코드하고, 비트 q를 1로 한 것을 제외하고 길이 2^k인 비트 벡터를 0으로 복귀시킨다.

[0142] 여러가지 다른 시그너쳐도 가능하다는 것을 이해하여야 한다.

[0143] 만일 어드레스(901)가 캐시에 대한 로드이면, 시그너쳐(903)는 시그너쳐 레지스터 업데이터(904)로 공급된다. 업데이터는 또한 시그너쳐 레지스터(905)의 이전 값을 입력으로서 취하고 그 값을 새로운 값(906)으로 교체한다. 시그너쳐 레지스터를 업데이트하는 적당한 방법은 시그너쳐의 형태에 따라 선택된다. S_old가 시그너쳐 레지스터의 구값(old value)을 나타내고, S_new가 시그너쳐 레지스터의 신값(new value)을 나타내며, V가 시그너쳐(903)의 값을 나타낸다고 하자. 상기 시그너쳐 함수에 대응해서, 시그너쳐 업데이터(904)는 하기의 것을 연산한다.

[0144] 1. S_new = max(S_old, V). 이것은 스트림 레지스터와 상이한 비트의 최대수를 계속하여 추적한다.

[0145] 2. S_new = S_old bit-wise-or V. 이것은 상이한 비트의 수의 스코어보드를 유지한다.

[0146] 3. S_new = max(S_old, V). 이것은 스트림 레지스터와 상이한 각 그룹의 비트의 최대수를 계속하여 추적한다.

[0147] 4. S_new = S_old bit-wise-or V. 이것은 각 그룹의 상이한 비트의 수의 스코어보드를 유지한다.

[0148] 5. S_new = S_old bit-wise-or V. 이것은 동시에 발생하는 각 그룹의 상이한 비트의 수의 스코어보드를 유지한다.

[0149] 6. S_new = S_old bit-wise-or V. 이것은 각 그룹의 패리티의 스코어보드를 유지한다.

[0150] 7. S_new = S_old bit-wise-or V. 이것은 동시에 발생하는 각 그룹의 패리티의 스코어보드를 유지한다.

[0151] 스누프 요구가 들어올 때, 그 시그너쳐는 연산되어 시그너쳐 레지스터와 비교된다. 비교 결과 일치된 것이 없으면, 어드레스는 캐시에 있을 수 없고, 어드레스가 캐시에 있다고 정상 스트림 레지스터 및 마스크 필터가 표시하는 경우에도 요구는 필터링된다. 스누프는 어드레스가 캐시에 있다고 시그너쳐 레지스터 및 마스크 레지스터가 표시하는 경우에만 회송된다.

[0152] 시그너쳐 필터링 메카니즘은 도 18에 도시되어 있다. 캐시에 대한 로드 어드레스(1001)는 마스크 업데이트 로직(1002)에 전송되고, 이 마스크 업데이트 로직(1002)은 전술한 바와 같이 동작하여 이전 마스크 레지스터(1003) 및 스트림 레지스터(1004)를 취하고 마스크 레지스터(1003)를 업데이트한다. 이 어드레스(1001)는 또한 시그너쳐 함수(1005)에 공급되고, 시그너쳐 함수(1005)는 또한 스트림 레지스터(1004)를 입력으로서 취하여 시그너쳐(1006)를 발생한다. 시그너쳐(1006)와 이전 시그너쳐 레지스터(1008)는 시그너쳐 업데이트 로직(1007)에 공급되고, 시그너쳐 업데이트 로직(1007)은 시그너쳐 레지스터(1008)의 신값을 생성한다.

[0153] 스누프 어드레스(1009a) 요구가 들어올 때, 그 요구는 마스크 필터(1010)에 의해 수신되고 처리되어 마스크 스누프 요구(1011)를 생성한다. 또한, 동일한 스누프 어드레스(1009b로 표시됨)와 스트림 레지스터(1004)는 시그너쳐 함수(1012)에 공급되어 시그너쳐(1013)를 생성한다. 시그너쳐 함수(1005, 1012)는 동일한 로직이어야 한다. 즉, 만일 시그너쳐 함수(1005, 1012)가 동일한 입력을 가지면, 이들은 동일한 출력을 생성할 것이다. 스누프 요구(1013)의 시그너쳐 및 시그너쳐 레지스터는 시그너쳐 필터(1014)에 공급된다.

[0154] 이 필터는 이 시그너쳐를 가진 요구가 캐시에 있는지 및 그 정확한 동작을 시그너쳐의 유형에 따라서 판정하여야 한다. "스코어보드" 유형의 시그너쳐 업데이터인 경우, 스누프 시그너쳐는 시그너쳐 레지스터와 비트 단위로 AND 연산된다. 그 결과가 0이 아니면, 시그너쳐 스누프 요구(1015)가 만들어진다(즉, 이 신호는 만일 요구가 만들어지면 1로 설정되고 요구가 만들어지지 않으면 0으로 설정된다). "최대 비트수 변경" 유형의 시그너쳐 업데이터인 경우, 스누프 시그너쳐가 시그너쳐 레지스터와 같거나 더 적은지를 알기 위한 체크가 행하여진다(각 그룹에 대해서 1회 비교). 이러한 비교가 모두 참(true)이면, 어드레스는 캐시에 있고 시그너쳐 스누프 요구(1015)가 만들어진다. 마스크 스누프 요구(1011) 및 시그너쳐 스누프 요구(1015)는 논리 소자(1016)에서 함께 AND 연산되어 스누프 요구 신호(1017)를 발생한다. 만일 이 신호가 1이면, 스누프 벡터 리스트 또는 적용된 범위 필터(도 7 참조)에 의해 제외되지 않는 한 스누프 요구가 발생될 것이다. 그러나, 구체적으로, 상기 스누프 요구는 다른 스트림 레지스터로부터 시그너쳐 마스크 필터의 결과에 의해 제외될 수 없다.

[0155] 시그너쳐 레지스터는 스트림 레지스터가 최초로 설정 또는 리세트되는 것과 같은 시간에 적절히 설정된다. 스코어보드 유형 및 최대 유형(max-type)의 시그너쳐에 대해서, 시그너쳐 레지스터는 모두 0으로 설정된다(스트림 레지스터와 다른 비트들은 없다고 표시한다).

[0156] 스트림 레지스터 필터는 여기에서 캐시 랩 조건이라고 부르는 특수 시작 상태에 관하여, 캐시의 전체 콘텐츠가 교체되는 때를 아는 것에 의존한다. 세트 관련 캐시는 캐시 내의 모든 세트들이 교체되는 때 랩되는 것으로 생

각된다. 통상적으로, 일부 세트는 다른 것보다 더 빨리 교체되고, 모든 세트가 교체되고 캐시가 랩되기 전에 업데이트를 계속할 것이다. 그러므로, 캐시 랩 검출의 시작점은 이전 캐시 랩의 시간에 캐시 세트의 상태이다.

[0157] 일 실시예에서, 캐시는 세트 조합형이고 라운드 로빈 교체 알고리즘을 사용하지만, 다른 교체 구현도 가능하다. 예를 들면, 캐시 랩 검출은 캐시가 최근에 사용된 것 및 무작위를 비롯해서 임의의 교체 정책을 구현할 때 달성된다. 이후의 설명에서 언급되는 바와 같이, 세트 조합형(SA) 캐시는 임의 갯수의 세트를 포함하고, 각 세트는 복수의 라인을 저장할 수 있다(각각 동일한 세트 지수를 갖는다). 세트 내의 라인들을 "웨이"(way)라고 부른다. 그러므로, 2-웨이 세트 조합형 캐시는 세트당 2개의 라인을 갖는다. 세트 내의 모든 웨이는 조사 중에 동시에 탐색되고, 그들 중의 하나만이 업데이트 중에 교체된다. 더욱이, 세트는 웨이의 부분집합이 각 파티션에 할당되도록 분할될 수 있다. 예를 들면, 4-웨이 SA 캐시는 2개의 2-웨이 SA 캐시로 분할될 수 있다. 가상 메모리 페이지 테이블(및 변환 색인 버퍼(TLB))은 (조사 및 업데이트 둘 다를 위해) 목표로 되는 특수 메모리 참조를 어떤 캐시가 분할하는지를 지정하는 파티션 식별자를 제공할 수 있다. 캐시 랩을 위해 업데이트될 웨이를 저장하는 레지스터는 웨이 번호를 저장하기 위해 충분히 클 필요가 있다. 예를 들면, 4-웨이 SA 캐시에 대해서 2 비트, 또는 32-웨이 SA 캐시에 대해서 5 비트로 되어야 한다. 각 세트는 다른 시간에 랩될 수 있기 때문에 세트당 그러한 레지스터가 한 개 있다.

[0158] 본 발명의 일 실시예에서, 캐시는 3개의 파티션으로 분할되고, 각 파티션은 캐시 웨이의 연속적인 부분집합을 포함하며, 그 부분집합은 각 캐시 세트 내에서 동일하다. 메모리 참조는 3개의 파티션 중의 하나에 캐시되도록 프로세서의 메모리 관리 유닛에 의해 지정된다. 파티션에 대한 업데이트는 다른 파티션과 무관하게 발생하고, 그래서 하나의 파티션은 전체 캐시가 랩되기 오래 전에 랩될 수 있다. 그러나, 파티션의 래핑의 검출은 업데이트되는 파티션을 알고 있을 때 전체 캐시의 래핑을 검출하는 것과 동일하다. 따라서, 뒤에서 설명하는 바와 같이, 캐시 래핑은 파티션 래핑 또는 전체 캐시 래핑을 포함한다.

[0159] 외부 로직이 캐시 업데이트를 검출하기 위해서, 캐시는 업데이트가 발생하는 것 및 어떤 라인이 덮어쓰기되는지에 대한 표시를 제공하여야 한다. 양호한 실시예의 로직은 이 정보가 세트 사양, 웨이 사양 및 업데이트 표시자에 의해 제공된다고 가정한다.

[0160] 도 19(a) 및 도 19(b)는 N-웨이 세트 조합형 캐시에 관한 양호한 실시예의 캐시 랩 검출 로직을 도시한 것이다. 이 실시예에서, 세트에 대한 업데이트는 항상 라운드 로빈 순서로 수행되는 것으로 가정된다. 즉, 덮어쓰기되도록 선정되는 "빅팀"(victim) 웨이는 항상 이전에 덮어쓰기된 1에 뒤따르는 1이다.

[0161] 도 19(a)는 논리 블록(920) 내의 단일 세트(이 실시예에서는 세트 "i")의 단일 파티션의 랩을 검출하기 위해 구현되는 로직의 일 실시예를 특별히 묘사한 것이다. 이 로직이 세트 i에서 랩을 검출하는 경우, 로직은 세트랩(i)(set_wrap(i)) 신호(910)를 어서트(assert)한다. 도 19(b)는 모든 N 세트의 캐시로부터의 개별 세트랩 신호(i)(910)가 어떻게 논리 OR 함수와 결합되어 전체 캐시(즉, 모든 세트)가 랩될 때 어서트되는 캐시랩(cache_wrap) 신호(912)를 생성하는가를 보여주고 있다. 도 19(a) 및 도 19(b)에 도시된 로직 및 회로는 단지 하나의 예시적인 구현을 보인 것이고, 당업자라면 본 발명의 범위에서 벗어나지 않고 많은 변형예 및 수정예가 가능하다는 것을 알 것이다.

[0162] 도 19(a)의 좌측에는 캐시 업데이트가 래핑을 감시받는 파티션 내에 포함되는 때를 결정하는 파티션 검출 논리 블록(922)이 도시되어 있다. 이 로직은 파티션이 "하부"(916)라고 표시된 웨이로부터 "상부"(918)라고 표시된 웨이까지 연장하는 것을 가정한다. 그러므로, 세트 랩 파티션을 검출하는 로직의 나머지는 업데이트가 있을 때의 상태만을 변경하고, 그 업데이트는 관심 대상 파티션 내에 포함된다. 파티션 검출 로직(922)은 세트 랩 검출 로직의 N개의 카피 모두에게 공통임을 주목한다.

[0163] 세트 랩 검출 로직 내에서, 공통 파티션 업데이트 표시자는 업데이트가 그 로직과 관련된 특수 세트 i에 대해서 행하여질 때만 행동하도록 자격이 주어진다. 이것은 세트 랩 검출 로직(926)의 색인에 세트 지정자(924)를 일치시킴으로써 행하여진다.

[0164] 논리 회로의 나머지는 다음과 같이 기능한다. 초기에, 플립플롭으로 구동되는 세트랩(i)(930)이 클리어되어 그 세트가 랩되지 않았음을 표시하고, 레지스터(928)는 세트 랩을 완료하기 위해 업데이트되어야 하는 웨이를 포함한다고 가정한다. 이 상태에서, 레지스터는 그 값을 보유한다. 웨이(914)가 레지스터(928)의 콘텐츠와 일치한다고 비교기 장치(919)에 의해 판정되어 캐시 업데이트가 발생하였을 때, 플립플롭으로 구동되는 세트랩(i)(930)은 로직 1이 로드되어 세트랩(i)(910)이 어서트되게 한다. 그 다음에, 캐시 업데이트는 업데이트된 웨이(914)가 레지스터(928)에 저장되게 하고, 레지스터(928)는 그 업데이트를 효과적으로 추적한다. 모든 캐시 세트가 랩되

었을 때, 합성 캐시랩(912) 신호는 도 19(b)에 도시된 것처럼 어서트되어 플립플롭(930)이 클리어되게 한다(리세트가 로드에 우선한다고 가정한다). 이것은 다음 세트 랩을 표시하도록 업데이트되어야 하는 웨이를 레지스터(928)가 저장하는 초기 상태로 회로를 복귀시킨다.

[0165] 따라서, 다수의 웨이를 저장하는 세트당 하나의 레지스터가 있고, 그 웨이가 덮어쓰기될 때 그 세트가 랩된다. 그러나, 세트들은 다른 때에 랩되고(액세스 패턴에 의존함), 전체 캐시는 모든 세트가 랩될 때까지 랩되는 것으로 생각되지 않는다. 그 점에서, 빅팀 웨이 포인터(즉, 덮어쓰기된 최종 웨이에 대한 포인터; 세트당 하나)의 상태는 다음 캐시 랩을 검출하기 위한 새로운 초기 조건으로 된다. 제1 실시예는 세트가 랩되는 시점과 전체 캐시가 랩되는 시점 사이에 덮어쓰기된 웨이를 전술한 레지스터가 계속하여 추적하게 함으로써 이 필요조건을 수용한다. 그 다음에, 전체 캐시가 랩될 때, 레지스터는 덮어쓰기된 웨이의 추적을 중단하고, 세트가 다시 랩되는 때를 결정하기 위한 비교의 기초로 된다.

[0166] 캐시 랩 검출 로직의 제2 실시예에서, 카운터가 구현되고, 전체 캐시가 랩될 때, 모든 세트 카운터가 파티션 내의 웨이의 수로 리세트된다. 웨이가 덮어쓰기될 때, 카운터는 카운트다운을 하고, 카운터가 0에 도달한 때 대응하는 세트가 랩된다. 모든 카운터가 0에 도달하였을 때, 캐시가 랩되고 처리를 다시 시작한다.

[0167] 따라서, 이 제2 실시예에 따르면, 도 19(a)에 도시된 박스(920) 내에 제공된 세트 랩 검출 로직은 레지스터 및 비교기가 아니라 로드가능한 카운터에 기초한다. 이 로직은 도 20에 도시되어 있다. 이 로직에서, 다운 카운터 장치(932)는 세트랩(i)(910)이 어서트되는 동안 파티션(936) 내의 웨이의 수로 로드된다(로드가 다운에 우선하는 것으로 가정한다). 모든 세트가 랩되고 캐시랩(912)이 어서트된 때, 플립플롭(930)으로 구동하는 세트랩(i)는 클리어되고 카운터(932)는 더 이상 로드되지 않는다. 그 다음에, 로직에 의해 추적되는 파티션(914) 및 세트(934)에 대한 각 업데이트는 카운터(932)가 1씩 다운 카운트하게 한다. 카운터가 0에 도달하면, 플립플롭(930)은 로직 1로 로드되어 세트랩(i)(910)이 어서트되게 하고, 로직을 초기 상태로 복귀시킨다.

[0168] 도 21에 도시된 캐시 랩 검출 로직의 제3 실시예는 최근에 사용된 것 및 무작위를 비롯하여 임의의 교체 정책을 구현하는 캐시와 함께 동작할 것이다. 이 경우, 스코어보드(940)는 덮어쓰기되는 정확한 캐시 웨이(914)를 계속하여 추적하기 위해 사용된다. 구체적으로, 스코어보드(940)는 임의의 웨이에 대한 최초 기록을 검출하기 위해 사용된다. 또한, 카운터(942)는 스코어보드 비트가 최초로 세트(즉, 0에서 1로 됨)된 횟수를 계속하여 추적한다. 이것은 덮어쓰기된 비트(구비트)가 0인 스코어보드 기록을 카운트함으로써 행하여진다. 카운터(942)는 파티션 사이즈(936)(즉, 파티션 내 웨이의 수)로 미리 로드되고, 이 카운터가 0에 도달하면 전체 캐시 파티션이 랩된다. 이것은 어서트되는 캐시랩(912) 신호에 의해 표시되고 카운터(942)가 다시 로드되고(로드가 다운에 우선하는 것으로 가정함) 스코어보드(940)가 클리어(즉, 리세트)되게 한다.

[0169] 지금까지 본 발명의 양호한 실시예를 스누핑이 기록 요구에서만 발생하고, 스누프 행동의 결과가 로컬 데이터 카피를 무효화시키는 라이트 스루 캐시(write-through cache)와 관련하여 설명하였지만, 본 발명은 이것으로 제한되는 것이 아니다. 예를 들면, 본 발명은 라이트 백 캐시(write-back cache) 구조에서도 또한 실행될 수 있다. 라이트 백 캐시에 따르면, 일관성 프로토콜은 추가적인 트랜잭션, 이것으로 제한되는 것이 아닌, 예를 들자면, 잘 알려진 MESI 프로토콜 또는 다른 일관성 프로토콜에 따르는 트랜잭션을 포함할 것이다. 라이트 백 캐시에 대한 일관성 프로토콜에 따르면, 원격 프로세서에서의 읽기 트랜잭션은 원격 캐시가 메인 메모리와 관련하여 가장 최근의 데이터 카피를 갖는지를 스누프 행동이 결정하게 한다. 만일 그러한 데이터 카피를 갖고 있으면, 데이터 전송은 여러가지 방법 중의 하나, 이것으로 제한되는 것이 아닌, 예를 들자면, 가장 최근 데이터를 가진 프로세서가 데이터를 메인 메모리에 기록하게 하는 것, 데이터를 가장 최근 카피의 소유자로부터 요구자에게 직접 전송하게 하는 것, 또는 특수 프로토콜의 스누프 개입에 따라 데이터를 전송하기 위한 임의의 다른 방법을 이용하여 수행된다. 본 발명에 따르면, 스누프 필터링 행동은 가속화 스누프 응답을 결정하기 위해 사용될 수 있다.

[0170] 비록, 양호한 실시예는 고정형 상호접속 토폴로지 및 고정형 스누프 필터링 동작에 관하여 설명하였지만, 본 발명의 일 태양에 있어서, 스누프 필터링 서브시스템은 스누프 필터 계층의 하나 이상의 태양에서 프로그램 가능하다. 본 발명의 프로그램 가능한 특징의 일 실시예에 따르면, 상호접속 토폴로지가 선택된다. 다양한 프로그램 가능 토폴로지에 따르면, 토폴로지 내의 상이한 필터들 사이에서 일 대 일 및 일 대 다수의 관계를 선택할 수 있다. 프로그램 가능 실시예의 다른 태양에 따르면, 제1 스누프 필터 및 그 다음에 제2 스누프 필터가 액세스되는 순서, 또는 대안적으로 제1 또는 제2 스누프 필터가 병렬로 액세스되는 것은 프로그램 제어하에 구성가능하다.

[0171] 본 발명의 프로그램 가능한 특징의 다른 실시예의 다른 태양에 따르면, 필터 서브유닛의 동작을 프로그램할 수

있다. 이것은 예를 들면 스누프되는 캐시의 조합성(associativity), 구현되는 일관성 구조 등과 같은 프로그램가능한 태양을 구성함으로써 스누프 필터의 구성가능한 태양의 형태로 될 수 있다. 프로그램 가능 필터 서브유닛의 다른 태양에 있어서, 필터 서브유닛은 프로그램가능 마이크로코드로 구현되고, 이것에 의해 프로그램가능 엔진이 명령어의 시퀀스를 실행하여 여기에서 설명하는 하나 이상의 양호한 실시예의 태양들을 구현한다. 일 태양에 있어서, 이것은 일반 마이크로코드 엔진이다. 다른 태양에 있어서, 이것은 최적화 프로그램가능 마이크로코드 엔진이고, 프로그램가능 마이크로코드 엔진은 스누프 필터 지정 조건을 검출하기 위해 특수화 지원 로직, 및 선택적으로 "캐시 랩 조건에서의 가지"와 같은 특수화 동작, 예를 들면, "캐시 랩 조건에서의 방해"와 같은, 마이크로코드 엔진에 전달되는 마이크로코드 엔진 지정 예외의 형태인 특수화 통지 이벤트 등을 갖는다.

[0172] 본 발명의 프로그램가능한 특징의 다른 실시예에서, 스누프 필터링의 일부 또는 모든 태양은 프로그램가능 스위치 매트릭스 또는 프로그램가능 게이트 어레이 패브릭을 통합하여 구현된다. 이 태양들 중의 하나에서, 스누프 서브유닛들 간의 라우팅은 프로그램가능 스위치 매트릭스를 구성함으로써 수행된다. 이 프로그램가능 실시예의 다른 태양에서, 스누프 필터 유닛의 행동들은 프로그램가능 게이트 어레이 논리 블록을 구성함으로써 구현된다. 본 발명의 다른 태양에서, 전체 스누프 필터 블록은 적어도 하나의 필드 프로그램가능 게이트 어레이 셀을 구성함으로써 구현된다.

[0173] 본 발명의 프로그램가능 특징의 다른 실시예에 따르면, 하나 이상의 스누프 필터 서브시스템이 디스에이블될 수 있고, 특정의 스누프 필터링 단계가 우회(bypass)될 수 있으며, 또는 스누프 필터링이 함께 디스에이블될 수 있다. 일 실시예에서, 이것은 스누프 필터의 구성을 구성 레지스터에 기록함으로써 달성된다. 다른 실시예에서, 이 구성은 입력 신호에 의해 선택될 수 있다.

[0174] 지금까지 본 발명의 양호한 실시예라고 생각되는 예를 도시하고 설명하였지만, 본 발명의 정신으로부터 벗어나지 않고 그 형태 및 상세를 여러가지로 수정 및 변경하는 것이 가능함은 물론이다. 그러므로, 본 발명은 여기에서 도시하고 설명한 것과 정확히 동일한 형태로 제한되는 것이 아니고, 첨부된 청구범위 내에 속하는 모든 수정 및 변형을 포함하는 것으로 해석되어야 한다.

도면의 간단한 설명

[0047] 도 1은 종래 기술에 따른 캐시 일관성을 위한 일관성 디렉토리를 가진 기본 멀티프로세서 구조를 보인 도이다.

[0048] 도 2는 종래 기술에 따른 캐시 일관성을 위한 스누핑 방법을 이용하는 기본 멀티프로세서 시스템을 보인 도이다.

[0049] 도 3은 본 발명에 따라 설명되는 점대점 접속을 이용하여 캐시 일관성을 위한 스누핑 방법을 이용하는 기본 멀티프로세서 시스템을 보인 도이다.

[0050] 도 4는 스누프 필터가 L2 캐시와 메인 메모리 사이에 배치된 점대점 접속을 이용하여 캐시 일관성을 위한 스누핑 방법을 이용하는 기본 멀티프로세서 시스템의 대안적 실시예를 보인 도이다.

[0051] 도 5는 본 발명의 양호한 실시예에 따른 스누프 필터 블록의 고준위 개요를 보인 도이다.

[0052] 도 6은 본 발명에 따른 다중 스누프 필터를 내포하는 스누프 블록의 고준위 개요를 도인 도이다.

[0053] 도 7은 본 발명에 따른 단일 스누프 포트 필터의 고준위 개요를 보인 도이다.

[0054] 도 8(a) 및 도 8(b)는 본 발명에 따른 스누프 블록의 2개의 대안적 실시예의 고준위 개요를 보인 도이다.

[0055] 도 9는 본 발명의 다른 실시예에 따른 다중 포트 스누프 필터를 포함하는 스누프 블록의 고준위 개요를 보인 도이다.

[0056] 도 10은 본 발명에 따라 단일 스누프 소스의 스누프 캐시를 구현하는 스누프 필터의 제어 흐름을 보인 도이다.

[0057] 도 11은 본 발명에 따라 포트 스누프 캐시에 새로운 엔트리를 추가하기 위한 제어 흐름 로직을 보인 도이다.

[0058] 도 12는 본 발명에 따라 스누프 캐시로부터 엔트리를 제거하기 위한 제어 흐름 로직을 보인 도이다.

[0059] 도 13은 본 발명에 따라 스트림 레지스터를 구현하는 스누프 필터의 블록도를 보인 도이다.

[0060] 도 14는 본 발명에 따라 스트림 레지스터 필터링 방법을 구현하는 스누프 필터의 다른 실시예를 보인 도이다.

[0061] 도 15는 본 발명에 따른 쌍을 이룬 스트림 레지스터 및 마스크 세트를 이용하여 스누프 필터의 흐름 제어를 보

인 블록도이다.

[0062] 도 16은 본 발명에 따른 교체된 캐시 라인의 2개의 스트림 레지스터 세트와 캐시 랩 검출 로직을 업데이트하기 위한 제어 흐름을 보인 블록도이다.

[0063] 도 17은 스트림 레지스터에 추가적인 필터링 능력을 제공하기 위한 시그너쳐 필터의 블록도이다.

[0064] 도 18은 본 발명에 따라 시그너쳐 파일을 이용한 필터링 메카니즘의 블록도이다.

[0065] 도 19(a) 및 도 19(b)는 N-웨이 세트 조합형 캐시의 예시적인 캐시 랩 검출 논리 회로(레지스터와 비교기)를 보인 도이다.

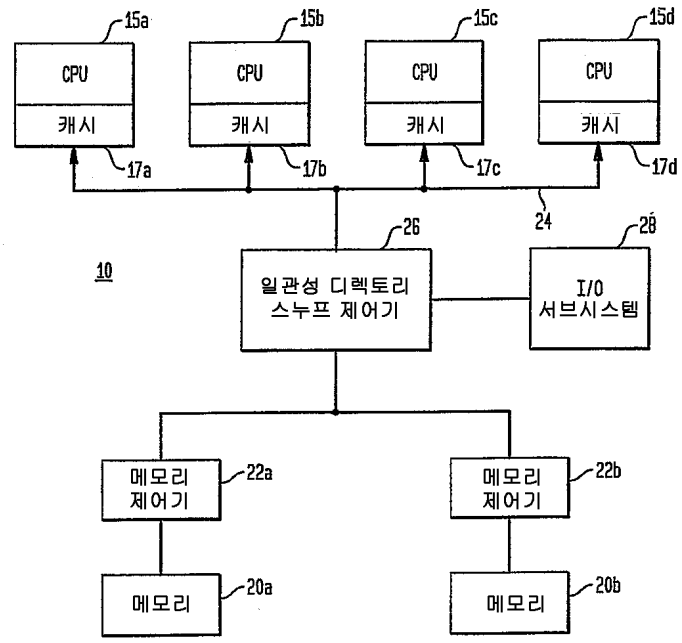
[0066] 도 20은 로드가능한 카운터에 기초한, 본 발명의 제2 실시예에 따른 N-웨이 세트 조합형 캐시의 예시적인 캐시 랩 검출 논리 회로를 보인 도이다.

[0067] 도 21은 스코어보드 레지스터에 기초한, 본 발명의 제3 실시예에 따른 N-웨이 세트 조합형 캐시의 예시적인 캐시 랩 검출 논리 회로를 보인 도이다.

도면

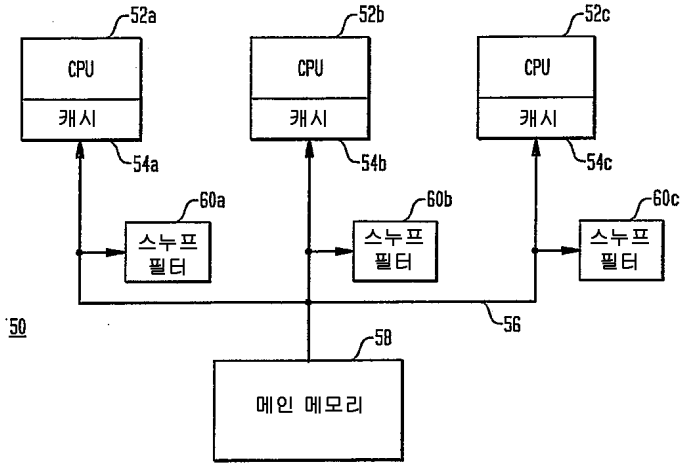
도면1

종래기술

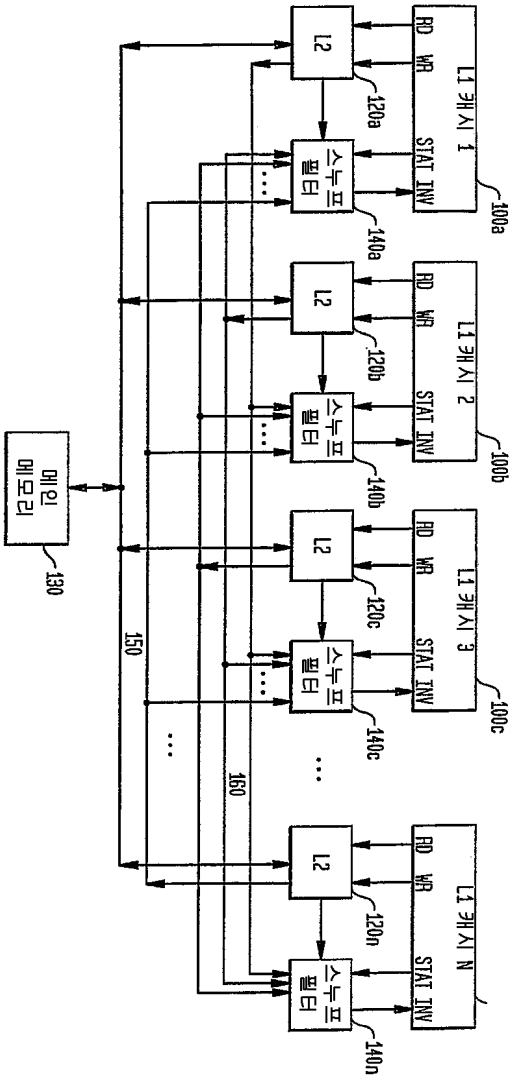


도면2

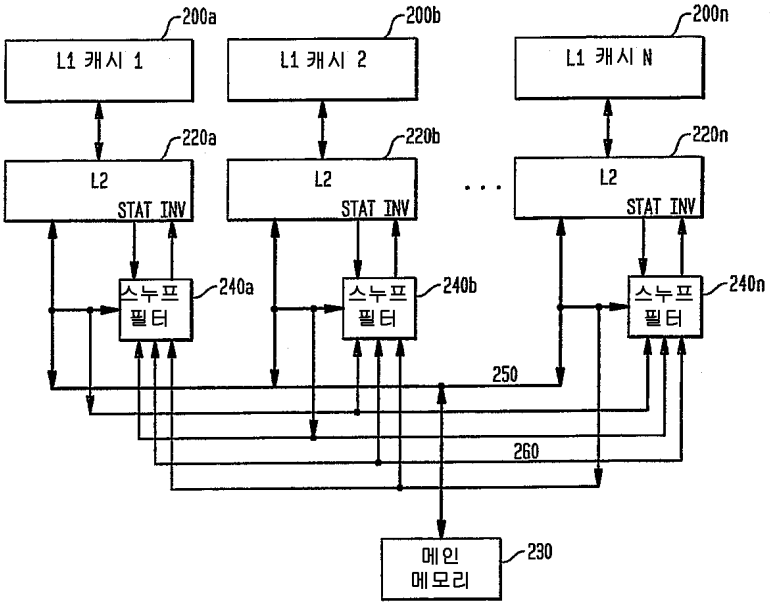
종래기술


52a
CPU
캐시
54a
52b
CPU
캐시
54b
52c
CPU
캐시
54c
60a
스누프
필터
60b
스누프
필터
60c
스누프
필터
50
56
58
메인 메모리

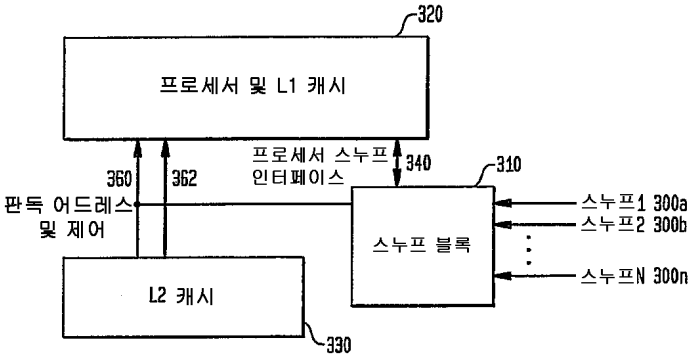
도면3


L1 캐시 1
L1 캐시 2
L1 캐시 3
L1 캐시 N
RD
WR
STAT INV
L2
스누프 필터
메인 메모리
100a
100b
100c
120a
120b
120c
120n
130
140a
140b
140c
140n
150
160

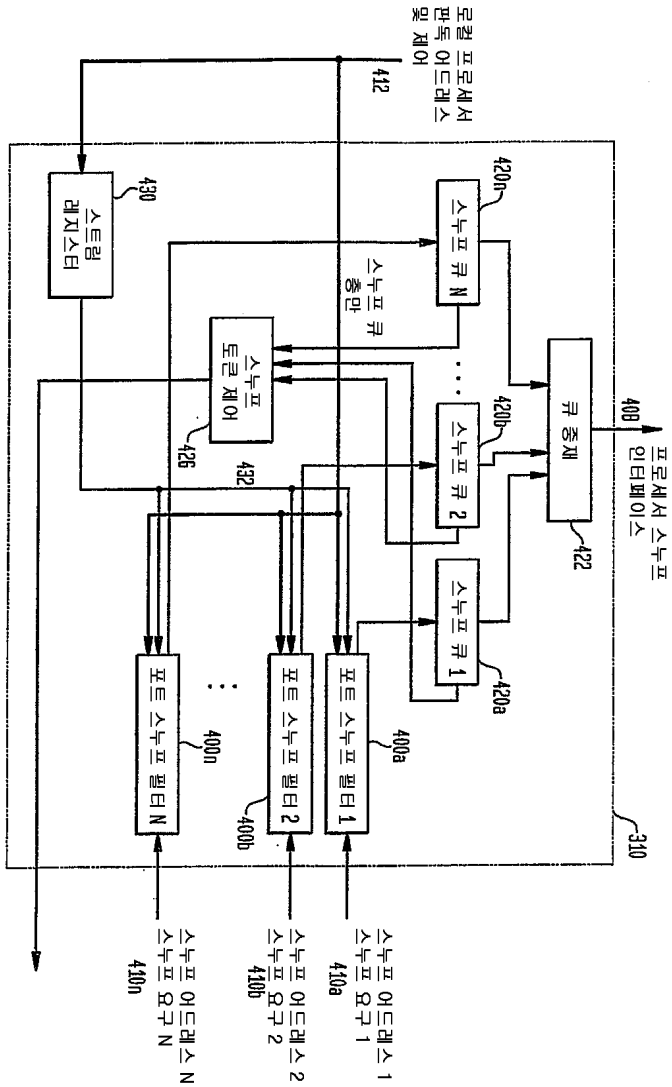
도면4


200a
200b
200n
L1 캐시 1
L1 캐시 2
L1 캐시 N
220a
220b
220n
L2
STAT INV
240a
240b
240n
스누프
필터
250
260
메인
메모리
230

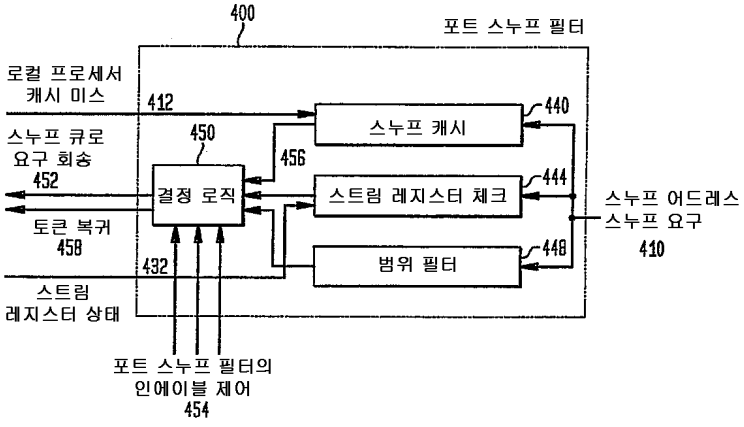
도면5


320
프로세서 및 L1 캐시
프로세서 스누프
인터페이스
340
310
360
362
판독 어드레스
및 제어
스누프 블록
스누프1 300a
스누프2 300b
스누프N 300n
L2 캐시
330

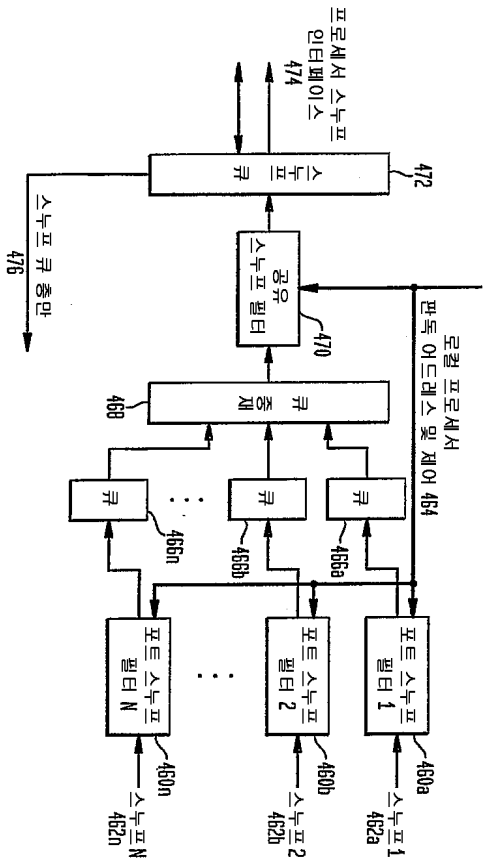
도면6


로컬 프로세서 판독 어드레스 및 제어
412
430
스트림 레지스터
420n
스누프 큐 N
스누프 큐 충만
스누프 토큰 제어
426
432
420b
스누프 큐 2
400
큐 중재
422
프로세서 스누프 인터페이스
420a
스누프 큐 1
포트 스누프 필터 N
400n
포트 스누프 필터 2
400b
포트 스누프 필터 1
400a
310
스누프 어드레스 N 스누프 요구 N
410n
스누프 어드레스 2 스누프 요구 2
410b
스누프 어드레스 1 스누프 요구 1
410a

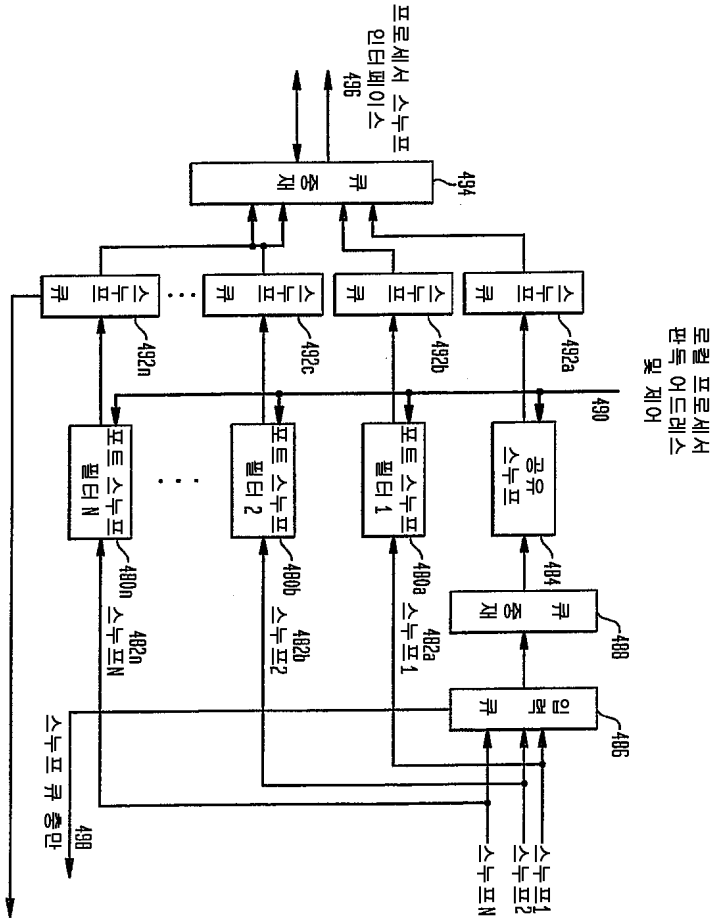
도면7


400
포트 스누프 필터
로컬 프로세서 캐시 미스
412
스누프 큐로 요구 회송
452
토큰 복귀
458
스트림 레지스터 상태
432
450
결정 로직
456
스누프 캐시
440
스트림 레지스터 체크
444
범위 필터
448
스누프 어드레스 스누프 요구
410
포트 스누프 필터의 인에이블 제어
454

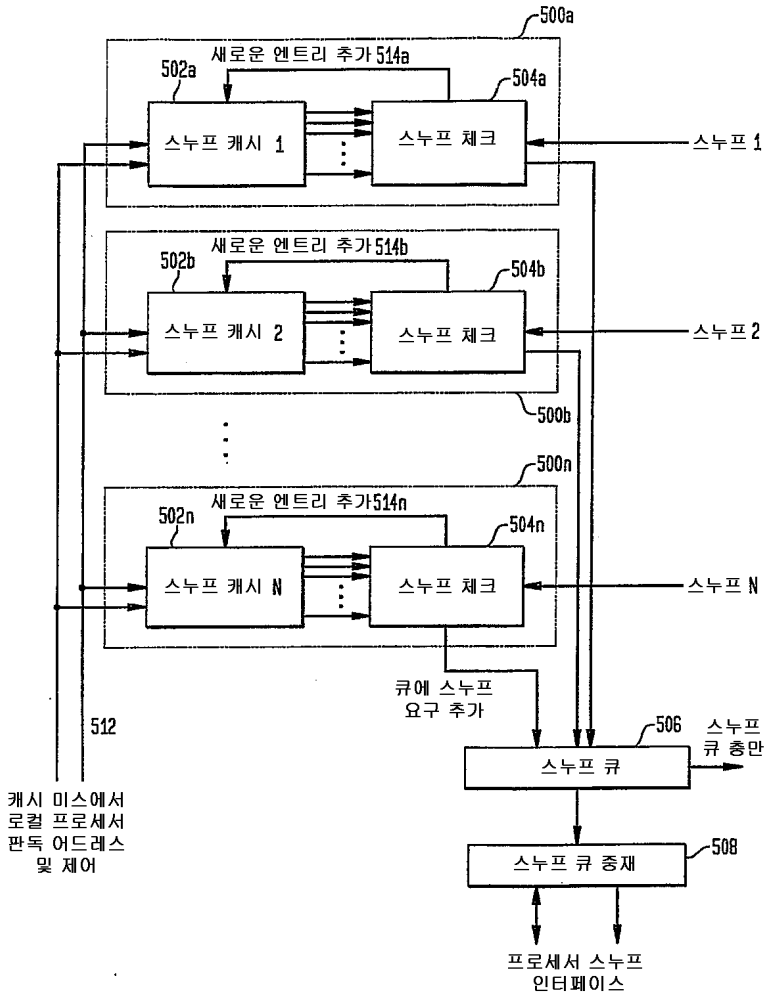
도면8a


프로세서 스누프
인터페이스
474
스누프 큐
472
스누프 큐 출력
476
공유
스누프 필터
470
로컬 프로세서
판독 어드레스 및 제어 464
큐
중재
468
큐
466n
큐
466b
큐
466a
포트 스누프
필터 N
460n
포트 스누프
필터 2
460b
포트 스누프
필터 1
460a
스누프 N
462n
스누프 2
462b
스누프 1
462a

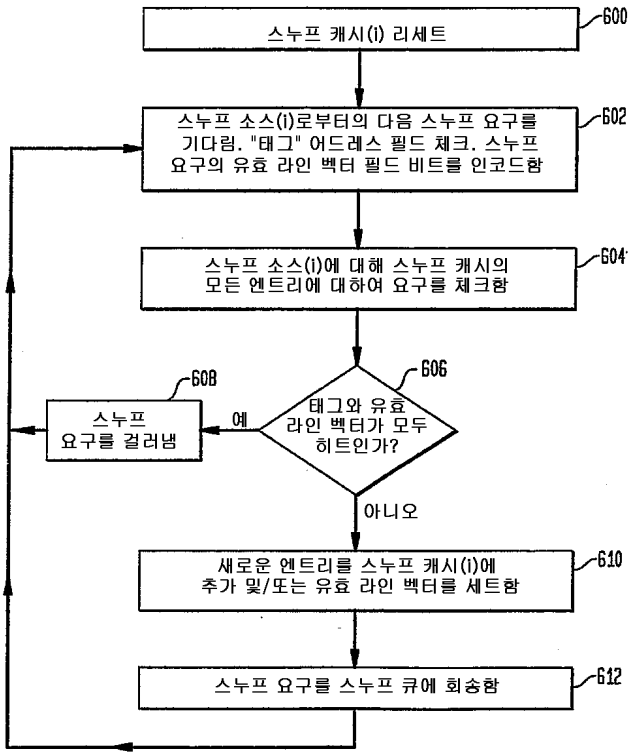
도면8b


프로세서 스누프 인터페이스 496
큐 중재 494
스누프 큐 492n
스누프 큐 492c
스누프 큐 492b
스누프 큐 492a
로컬 프로세서 판독 어드레스 및 제어 490
포트 스누프 필터 N
포트 스누프 필터 2
포트 스누프 필터 1
공유 스누프
480n 스누프N 482n
480b 스누프2 482b
480a 스누프1 482a
484
큐 중재 488
입력 큐 486
스누프 큐 출만 498
스누프N
스누프2
스누프1

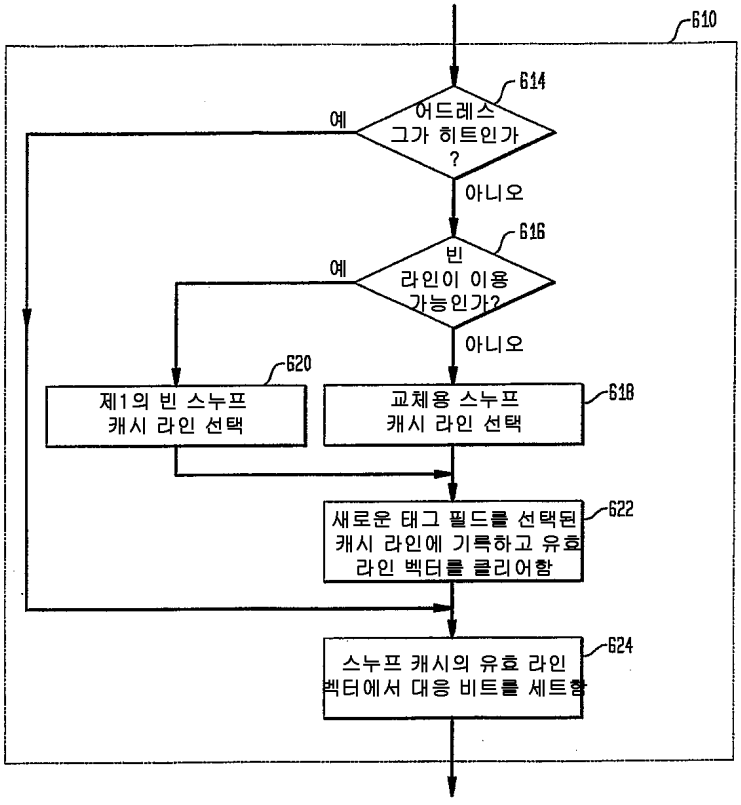
도면9


500a
새로운 엔트리 추가 514a
502a
504a
스누프 캐시 1
스누프 체크
스누프 1
새로운 엔트리 추가 514b
502b
504b
스누프 캐시 2
스누프 체크
스누프 2
500b
500n
새로운 엔트리 추가 514n
502n
504n
스누프 캐시 N
스누프 체크
스누프 N
큐에 스누프 요구 추가
512
506
스누프 큐 충만
스누프 큐
508
스누프 큐 중재
캐시 미스에서 로컬 프로세서 판독 어드레스 및 제어
프로세서 스누프 인터페이스

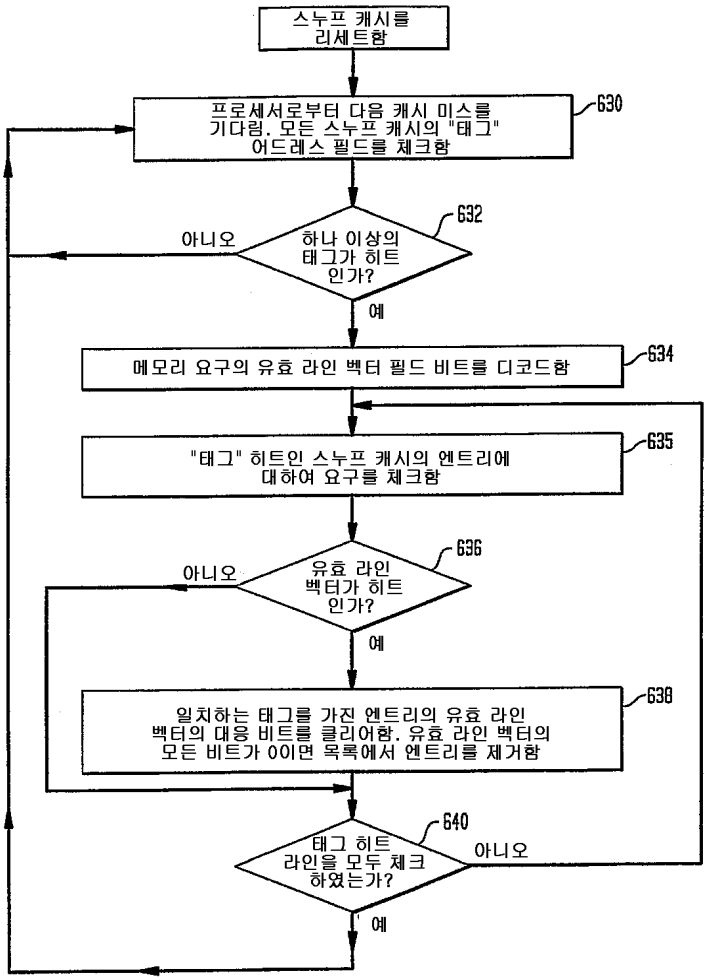
도면10


스누프 캐시(i) 리세트
600
스누프 소스(i)로부터의 다음 스누프 요구를 기다림. "태그" 어드레스 필드 체크. 스누프 요구의 유효 라인 벡터 필드 비트를 인코드함
602
스누프 소스(i)에 대해 스누프 캐시의 모든 엔트리에 대하여 요구를 체크함
604
태그와 유효 라인 벡터가 모두 히트인가?
606
예
스누프 요구를 걸러냄
608
아니오
새로운 엔트리를 스누프 캐시(i)에 추가 및/또는 유효 라인 벡터를 세트함
610
스누프 요구를 스누프 큐에 회송함
612

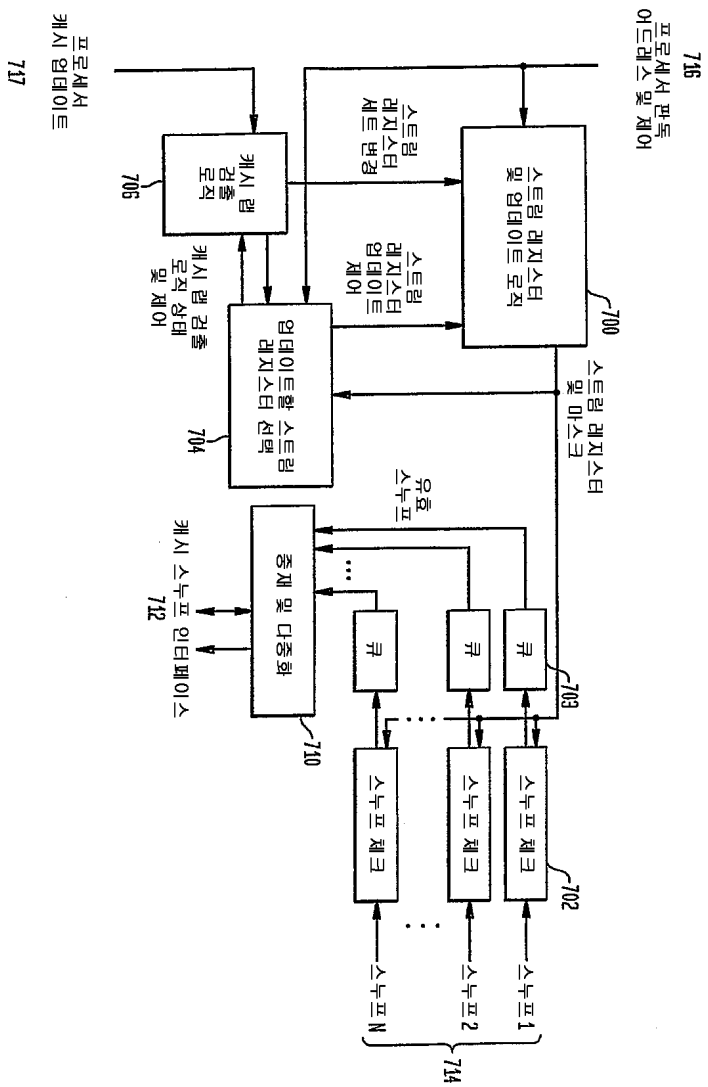
도면11


610
어드레스 그가 히트인가 ?
614
예
아니오
빈 라인이 이용 가능인가?
616
예
아니오
제1의 빈 스누프 캐시 라인 선택
620
교체용 스누프 캐시 라인 선택
618
새로운 태그 필드를 선택된 캐시 라인에 기록하고 유효 라인 벡터를 클리어함
622
스누프 캐시의 유효 라인 벡터에서 대응 비트를 세트함
624

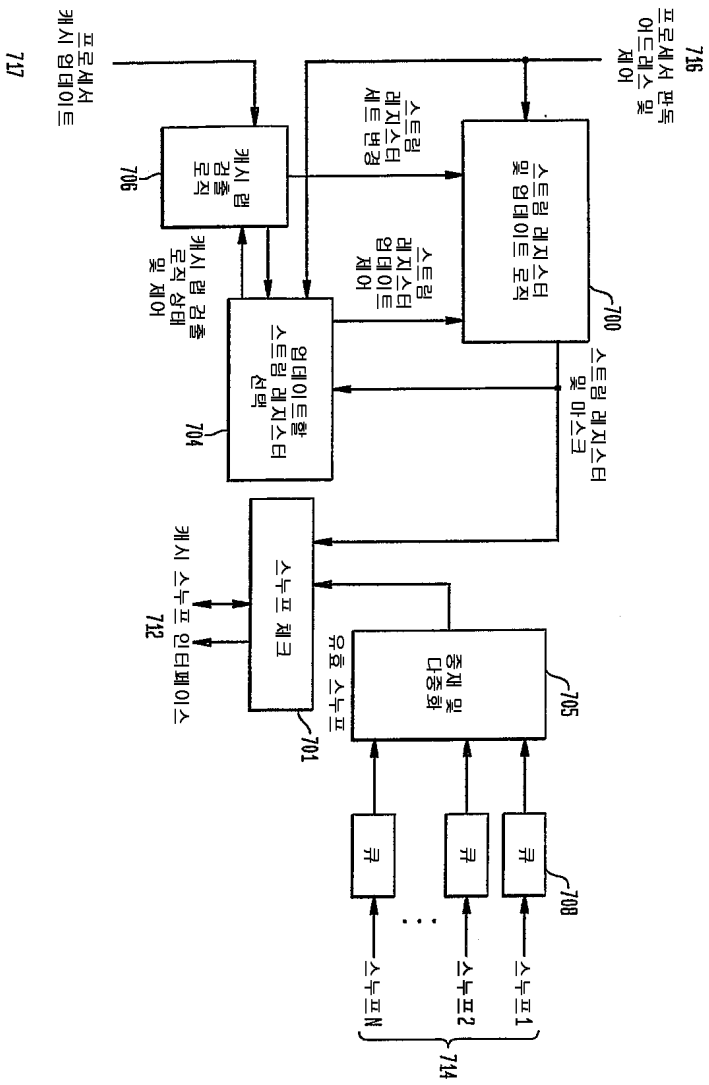
도면12


스누프 캐시를 리세트함
프로세서로부터 다음 캐시 미스를 기다림. 모든 스누프 캐시의 "태그" 어드레스 필드를 체크함
630
하나 이상의 태그가 히트 인가?
632
아니오
예
메모리 요구의 유효 라인 벡터 필드 비트를 디코드함
634
"태그" 히트인 스누프 캐시의 엔트리에 대하여 요구를 체크함
635
유효 라인 벡터가 히트 인가?
636
아니오
예
일치하는 태그를 가진 엔트리의 유효 라인 벡터의 대응 비트를 클리어함. 유효 라인 벡터의 모든 비트가 0이면 목록에서 엔트리를 제거함
638
태그 히트 라인을 모두 체크 하였는가?
640
아니오
예

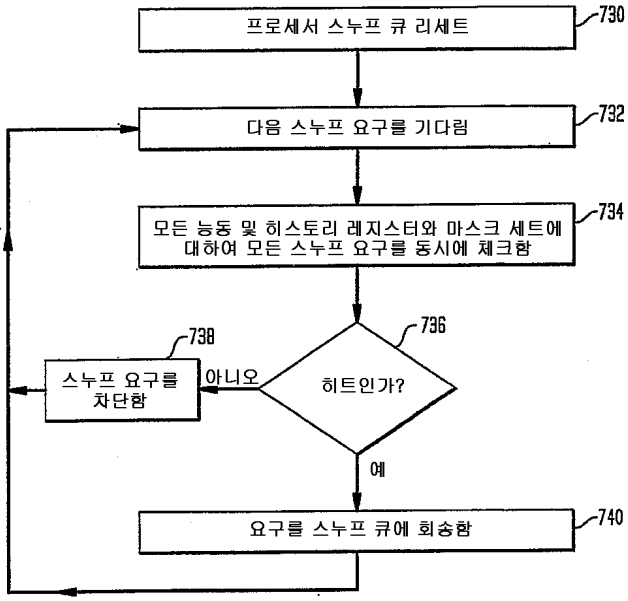
도면13


716
프로세서 판독 어드레스 및 제어
717
프로세서 캐시 업데이트
706
캐시 랩 검출 로직
스트림 레지스터 세트 변경
스트림 레지스터 업데이트 제어
캐시 랩 검출 로직 상태 및 제어
704
업데이트할 스트림 레지스터 선택
700
스트림 레지스터 및 업데이트 로직
스트림 레지스터 및 마스크
유효 스누프
712
캐시 스누프 인터페이스
710
중재 및 다중화
703
큐
큐
큐
702
스누프 체크
스누프 체크
스누프 체크
714
스누프 N
스누프 2
스누프 1

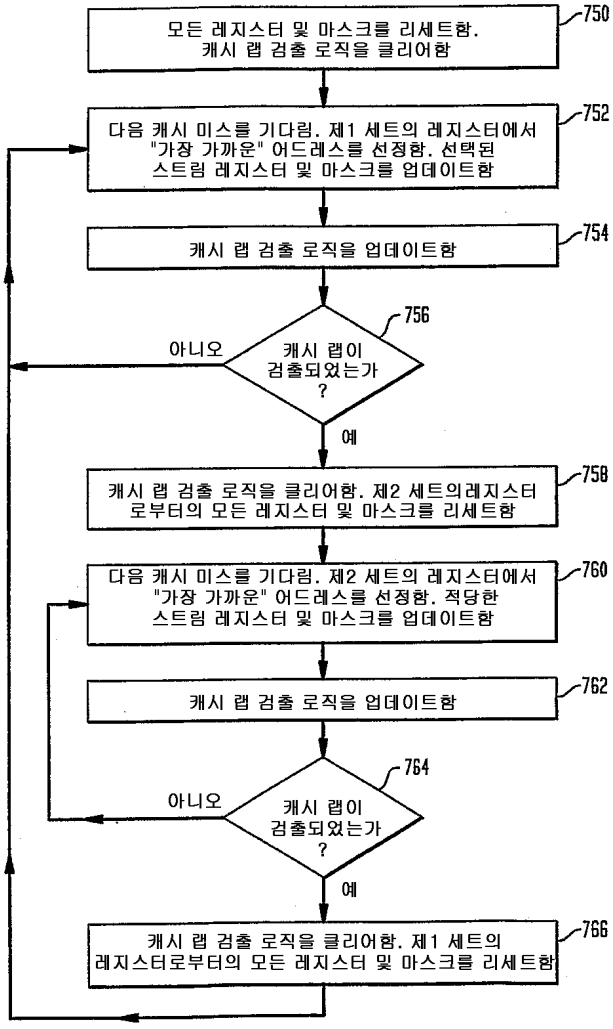
도면14


716
프로세서 판독 어드레스 및 제어
717
프로세서 캐시 업데이트
706
캐시 라인 검출 로직
캐시 라인 검출 로직 상태 및 제어
스트림 레지스터 세트 변경
스트림 레지스터 업데이트 제어
700
스트림 레지스터 및 업데이트 로직
스트림 레지스터 및 마스크
704
업데이트할 스트림 레지스터 선택
캐시 스누프 인터페이스
712
스누프 체크
701
유효 스누프
증재 및 다중화
705
큐
큐
큐
708
스누프N
스누프2
스누프1
714

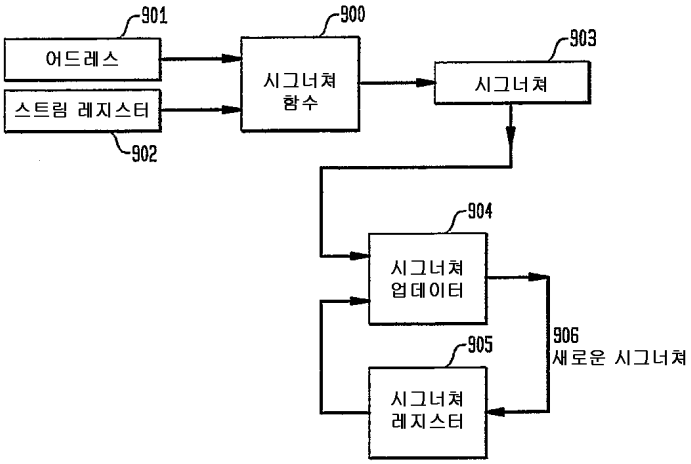
도면15


프로세서 스누프 큐 리세트
730
다음 스누프 요구를 기다림
732
모든 능동 및 히스토리 레지스터와 마스크 세트에 대하여 모든 스누프 요구를 동시에 체크함
734
히트인가?
736
아니오
스누프 요구를 차단함
738
예
요구를 스누프 큐에 회송함
740

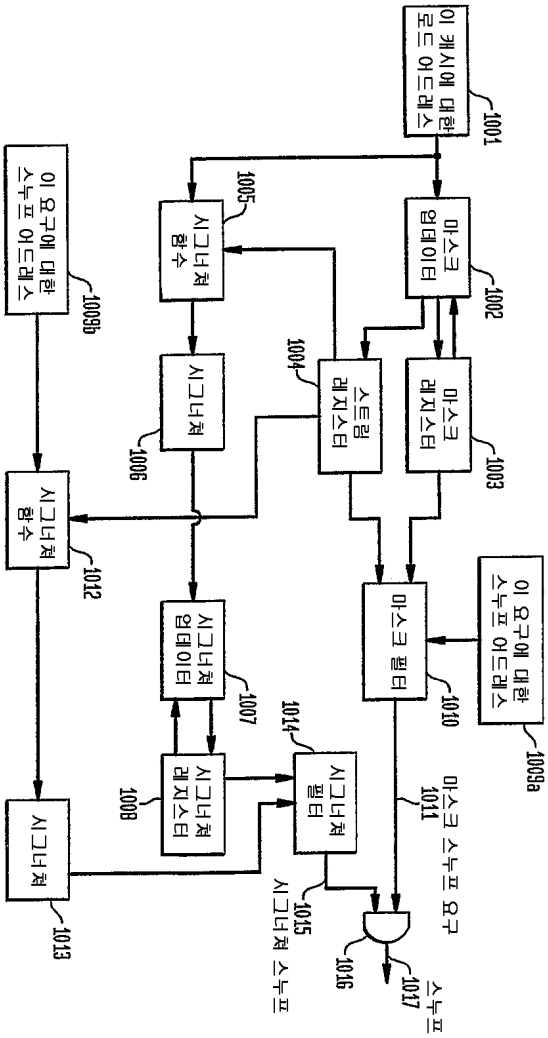
도면16


모든 레지스터 및 마스크를 리세트함. 캐시 랩 검출 로직을 클리어함
750
다음 캐시 미스를 기다림. 제1 세트의 레지스터에서 "가장 가까운" 어드레스를 선정함. 선택된 스트림 레지스터 및 마스크를 업데이트함
752
캐시 랩 검출 로직을 업데이트함
754
캐시 랩이 검출되었는가 ?
756
아니오
예
캐시 랩 검출 로직을 클리어함. 제2 세트의레지스터로부터의 모든 레지스터 및 마스크를 리세트함
758
다음 캐시 미스를 기다림. 제2 세트의 레지스터에서 "가장 가까운" 어드레스를 선정함. 적당한 스트림 레지스터 및 마스크를 업데이트함
760
캐시 랩 검출 로직을 업데이트함
762
캐시 랩이 검출되었는가 ?
764
아니오
예
캐시 랩 검출 로직을 클리어함. 제1 세트의 레지스터로부터의 모든 레지스터 및 마스크를 리세트함
766

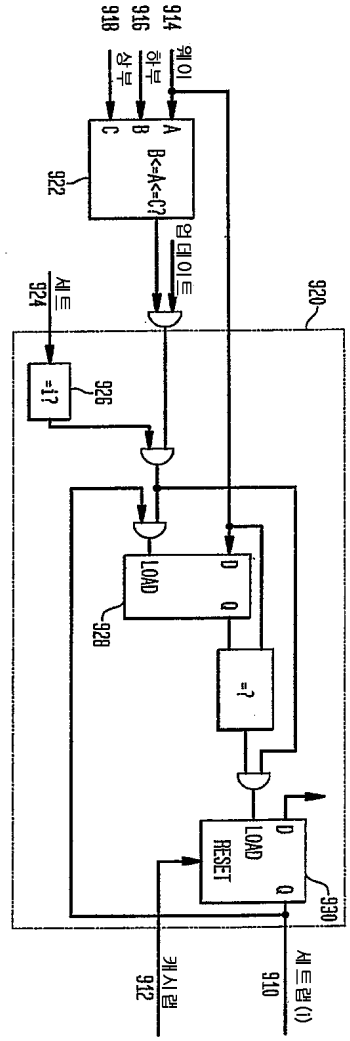
도면17


901
어드레스
902
스트림 레지스터
900
시그너쳐 함수
903
시그너쳐
904
시그너쳐 업데이터
905
시그너쳐 레지스터
906
새로운 시그너쳐

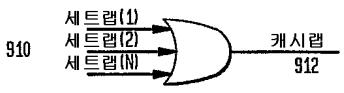
도면18


이 캐시에 대한 로드 어드레스
1001
마스크 업데이터
1002
마스크 레지스터
1003
스트림 레지스터
1004
시그너쳐 함수
1005
시그너쳐
1006
시그너쳐 업데이터
1007
시그너쳐 레지스터
1008
시그너쳐
1013
이 요구에 대한 스누프 어드레스
1009b
시그너쳐 함수
1012
마스크 필터
1010
이 요구에 대한 스누프 어드레스
1009a
시그너쳐 필터
1014
마스크 스누프 요구
1011
시그너쳐 스누프
1015
1016
스누프
1017

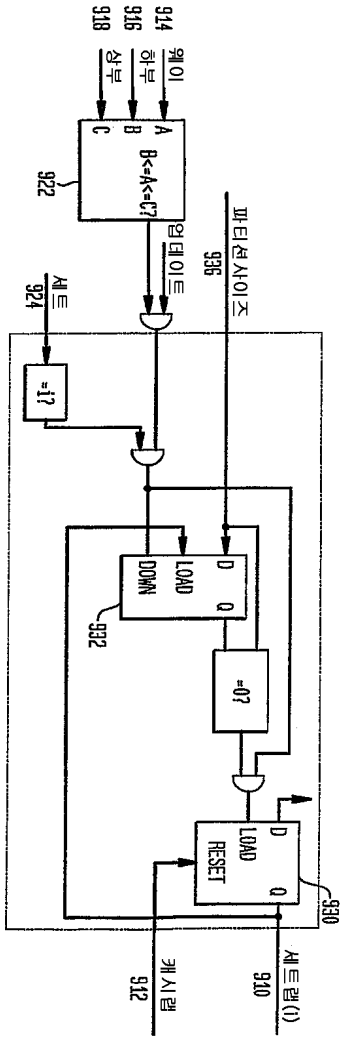
도면19a


914
916
918
웨이
하부
상부
A
B
C
B<=A<=C?
922
업데이트
세트
924
920
=1?
926
LOAD
D
Q
928
=?
LOAD
RESET
930
캐시랩
912
세트랩(1)
910

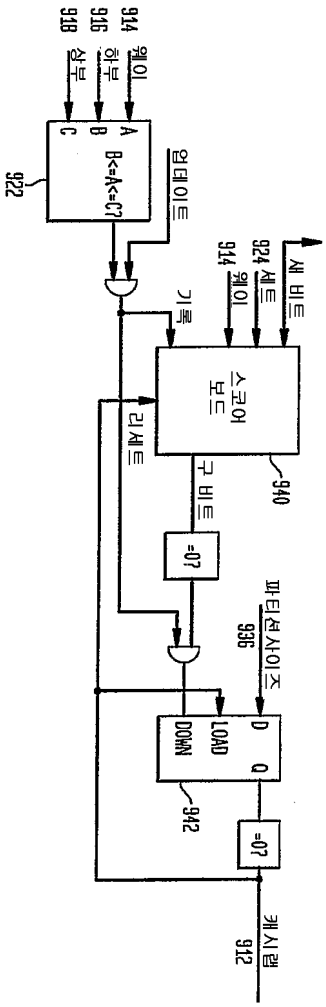
도면19b


910
세트랩(1)
세트랩(2)
세트랩(N)
캐시랩
912

도면20


914
웨이
936
하부
938
상부
A
B
C
B<=A<=C?
922
업데이트
936
파티션사이즈
924
세트
=1?
LOAD
DOWN
D
Q
932
=0?
LOAD
RESET
D
Q
930
912
캐시라
940
세트랄(1)

도면21


914
웨이
916
하부
918
상부
A
B
C
B<=A<=C?
922
업데이트
924
914
세트
웨이
새 비트
기록
스코어
보드
940
리셋트
구 비트
=0?
파티션사이즈
936
D
Q
LOAD
DOWN
942
=0?
캐시라인
912