(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2009년09월21일
(11) 등록번호 10-0918530
(24) 등록일자 2009년09월15일

(51) Int. Cl.
G06F 12/10 (2006.01)
(21) 출원번호 10-2004-0079969
(22) 출원일자 2004년10월07일
심사청구일자 2007년06월15일
(65) 공개번호 10-2005-0051540
(43) 공개일자 2005년06월01일
(30) 우선권주장
10/723,823 2003년11월26일 미국(US)
(56) 선행기술조사문헌
TELLER P.J. In: TRANSLATION-LOOKASIDE BUFFER CONSISTENCY, COMPUTER, IEEE SERVICE CENTER, VOL.23, NO. 6, pp.26-36 (1990.06.01) 1부.*
US20030200402 A1*
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
마이크로소프트 코포레이션
미국 워싱턴주 (우편번호 : 98052) 레드몬드 원 마이크로소프트 웨이
(72) 발명자
코헨,에르네스트에스.
미국 98052 워싱턴주 레드몬드 원 마이크로소프트 웨이 마이크로소프트 코포레이션 내
(74) 대리인
백만기, 이중희, 주성민

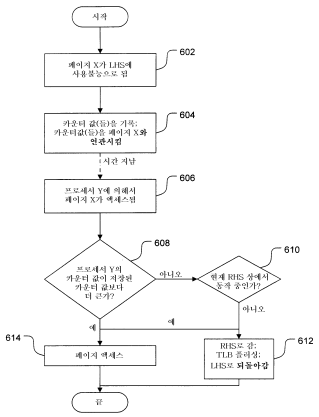
전체 청구항 수 : 총 11 항 심사관 : 이종익

(54) 어드레스 매핑 캐시들의 엔트리 소거 방법, 및 사용 관리 시스템

(57) 요 약

어드레스 변환 제어(ATC)는 메모리 액세스 정책을 구현하기 위해서 가상 및 물리 어드레스들 사이의 매핑들을 제한한다. 다중 프로세서 시스템에서 각각의 프로세서는 가상 어드레스들의 변환의 속도를 빠르게 하기 위해서 매핑들을 캐싱하는 변환 색인 버퍼(TLB)를 보유한다. 각각의 프로세서는 또한 카운터를 보유한다. 프로세서의 TLB가 플러싱될 때마다, 상기 프로세서의 카운터는 증가된다. 페이지에 대한 링크가 어드레스 변환 맵으로부터 제거될 때, 상기 모든 프로세서들에 대한 상기 카운터 값들이 기록된다. 상기 페이지가 프로세서에 의해서 액세스될 때, 상기 페이지에 대한 링크가 상기 맵으로부터 제거된 이후에 상기 프로세서의 TLB가 플러싱되었는지를 판정하기 위해서 상기 기록된 카운터 값들이 상기 프로세서의 현재 카운터 값과 비교된다. 비용이 많이 드는 TLB 플러싱 동작은 필요할 때까지 지연되지만, 무효한 TLB 엔트리가 상기 액세스 정책을 위배하면서 사용되지는 않을 정도로 빠르게 발생한다.

대 표 도 - 도6



특허청구의 범위

청구항 1

삭제

청구항 2

삭제

청구항 3

삭제

청구항 4

삭제

청구항 5

삭제

청구항 6

삭제

청구항 7

다수의 매핑 캐시들 중 제1 캐시로부터 쓸모없게 된 엔트리들을 소거 - 상기 다수의 매핑 캐시들 각각은 컴퓨팅 디바이스의 다수의 처리 유닛들 중 대응되는 유닛과 연관되며, 상기 캐시들 각각은 가상 어드레스들을 물리적 어드레스들로 변환하는 데에 사용되며 어드레스 변환 맵에 기초하여 매핑들을 저장함 - 하는 방법으로서,

카운터를 보유하는 단계;

상기 다수의 매핑 캐시들 중 상기 제1 캐시가 플러싱될 때마다 상기 카운터를 갱신하는 단계;

상기 어드레스 변환 맵에서의 변화에 응답하여 상기 카운터의 값을 기록함으로써, 기록된 카운터 값이 저장되는 단계;

상기 기록된 카운터 값과 상기 카운터의 값의 비교에 기초하여, 상기 어드레스 변환 맵에서의 상기 변화가 발생한 이후로 상기 다수의 매핑 캐시들 중 상기 제1 캐시가 분명하게 플러싱되지 않았다는 것을 판정하는 단계; 및

상기 다수의 매핑 캐시들 중 상기 제1 캐시를 플러싱하는 단계를 포함하며,

정책은 메모리에 허용가능한 액세스를 규정하며,

상기 방법은,

상기 어드레스 변환 맵이, 상기 정책에 위배하여 엔티티가 상기 메모리를 액세스하는 것을 허용하는 가상 어드레스 매핑들을 상기 엔티티에 노출시키지 않도록 상기 어드레스 변환 맵의 내용을 제어하는 단계를 더 포함하며,

상기 변화는 상기 정책에 따라 상기 맵을 위치시키거나 보유하는 상기 맵에 대한 수정이나, 상기 맵에 대한 상기 엔티티의 기록 액세스를 제한하는 상기 맵에 대한 수정 중 하나를 포함하는 방법.

청구항 8

제7항에 있어서,

상기 어드레스 변환 맵은 상기 메모리의 부분에 대한 링크를 포함하며, 상기 어드레스 변환 맵의 내용을 제어하는 단계는 상기 메모리의 상기 부분을 상기 엔티티가 액세스할 수 없도록 하는 단계를 포함하며, 상기 변화는 상기 어드레스 변환 맵으로부터 상기 메모리의 상기 부분에 대한 모든 링크들을 제거하는 것을 포함하는 방법.

청구항 9

삭제

청구항 10

삭제

청구항 11

어드레스 매핑 캐시들의 사용을 관리하는 시스템에 있어서,

다수의 프로세서들 -상기 프로세서들 각각은 매핑 캐시 및 이에 연관된 카운터를 가짐- ;

어드레스 변환 맵을 저장하는 메모리 -상기 매핑 캐시들 각각은 상기 어드레스 변환 맵에 기초한 매핑들을 저장하며, 상기 메모리에 대한 액세스를 제어하는 정책이 존재하며, 상기 어드레스 변환 맵의 내용들은 상기 정책에 위배하여 상기 메모리로의 액세스를 허용하는 매핑들의 노출을 방지하도록 제어됨- ;

상기 매핑 캐시들 중 제1 캐시를 플러싱하고 상기 매핑 캐시들 중 상기 제1 캐시가 플러싱될 때 상기 카운터들 중 제1 카운터를 증가시키는 제1 논리;

상기 어드레스 변환 맵에서의 변화 또는 상기 어드레스 변환 맵에 관한 속성(property)의 변화에 응답하여 상기 제1 카운터의 값을 기록하는 제2 논리 -상기 기록된 카운터 값은 상기 변화에 관련해서 저장됨- ;

상기 제1 카운터의 현재 값과 상기 기록된 카운터 값을 비교하여, 상기 비교가 상기 매핑 캐시들 중 상기 제1 캐시가 상기 변화 이후에 플러싱되지 않았다는 것을 지시한다면, 상기 매핑 캐시들 중 상기 제1 캐시가 플러싱되도록 하는 제3 논리를 포함하는 시스템.

청구항 12

제11항에 있어서,

상기 프로세서들 각각은 상기 카운터들 중 상기 제1 카운터와 연관되며, 상기 제1 논리는 상기 매핑 캐시들 중 임의의 캐시가 플러싱될 때 상기 카운터들 중 상기 제1 카운터를 증가시키는 시스템.

청구항 13

제11항에 있어서,

상기 매핑 캐시들 각각은 다수의 카운터들 중의 서로 다른 카운터와 연관되며, 상기 제1 논리는 소정의 매핑 캐시가 플러싱될 때 상기 소정의 매핑 캐시에 대응하는 상기 카운터를 증가시키는 시스템.

청구항 14

제11항에 있어서,

상기 변화는 상기 어드레스 변환 맵을 상기 메모리의 제1 페이지로의 모든 링크들이 상기 어드레스 변환 맵으로부터 제거되는 상태로 위치시키는 것을 포함하는 시스템.

청구항 15

제11항에 있어서,

상기 변화는 상기 어드레스 변환 맵을 상기 메모리의 제1 페이지로의 어떠한 기록가능한 링크도 존재하지 않는 상태로 위치시키는 것을 포함하는 시스템.

청구항 16

제11항에 있어서,

상기 제3 논리는 상기 변화의 결과가 사용되어야 한다는 탐지에 응답하여 호출(invoke)되는 시스템.

청구항 17

제16항에 있어서,

상기 변화는 상기 어드레스 변환 맵을 상기 메모리의 제1 페이지로의 모든 링크들이 상기 어드레스 변환 맵으로부터 제거되는 상태로 위치시키는 것을 포함하며, 상기 탐지는 상기 제1 페이지 상의 위치로 변환된 가상 어드레스에 기초하는 시스템.

청구항 18

삭제

청구항 19

삭제

청구항 20

어드레스 매핑들을 캐시하는 변환 색인(lookaside) 버퍼의 플러싱을 관리하는 방법을 수행하기 위한 컴퓨터 실행가능한 명령들이 부호화되어 있는 컴퓨터 판독가능한 저장 매체에 있어서, 상기 방법은,

가상 어드레스에 의해서 목적 위치를 지시하는 액세스 요청을 수신하는 단계;

상기 목적 위치의 물리적 어드레스를 획득하기 위해서 다수의 매핑 캐시들 중 제1 캐시를 이용하여 상기 가상 어드레스를 변환하는 단계 -상기 다수의 매핑 캐시들 중 상기 제1 캐시가 플러싱될 때마다 카운터가 갱신됨- ;

(1) 상기 목적 어드레스를 포함하는 페이지와 연관된 저장된 카운터 값을 (2) 현재 카운터 값과 비교하는 단계;

상기 저장된 카운터 값과 상기 현재 카운터 값 사이의 상기 비교에 기초하여 상기 페이지의 매핑에 영향을 주는 이벤트 이후에 상기 변환 색인 버퍼가 플러싱되지 않았다는 것을 판정하는 단계; 및

상기 변환 색인 버퍼를 플러싱하는 단계를 포함하며,

정책은 소프트웨어 엔티티에 대해 메모리의 액세스 가능성을 규정하며, 상기 이벤트는, 상기 어드레스 매핑들이 기초하는 어드레스 변환 맵으로부터 상기 메모리의 페이지로의 링크들을 제거하는 것을 포함하며, 상기 페이지는 상기 정책하에서 상기 소프트웨어 엔티티에 대해 액세스 불가능한 컴퓨터 판독 가능한 저장 매체.

청구항 21

어드레스 매핑들을 캐시하는 변환 색인(lookaside) 버퍼의 플러싱을 관리하는 방법을 수행하기 위한 컴퓨터 실행가능한 명령들이 부호화되어 있는 컴퓨터 판독가능한 저장 매체에 있어서, 상기 방법은,

가상 어드레스에 의해서 목적 위치를 지시하는 액세스 요청을 수신하는 단계;

상기 목적 위치의 물리적 어드레스를 획득하기 위해서 다수의 매핑 캐시들 중 제1 캐시를 이용하여 상기 가상 어드레스를 변환하는 단계 -상기 다수의 매핑 캐시들 중 상기 제1 캐시가 플러싱될 때마다 카운터가 갱신됨- ;

(1) 상기 목적 어드레스를 포함하는 페이지와 연관된 저장된 카운터 값을 (2) 현재 카운터 값과 비교하는 단계;

상기 저장된 카운터 값과 상기 현재 카운터 값 사이의 상기 비교에 기초하여 상기 페이지의 매핑에 영향을 주는 이벤트 이후에 상기 변환 색인 버퍼가 플러싱되지 않았다는 것을 판정하는 단계; 및

상기 변환 색인 버퍼를 플러싱하는 단계를 포함하며,

정책은 소프트웨어 엔티티로의 메모리의 액세스 가능성을 정의하며, 상기 이벤트는 페이지로의 매핑들을 기록불가능하도록 어드레스 변환 맵을 조정하는 것을 포함하며, 상기 정책은 (1) 상기 페이지를 상기 소프트웨어 엔티티에 의해 기록불가능한 것으로 규정하거나, (2) 상기 페이지는 상기 어드레스 변환 맵의 부분을 저장하는 것 중에 하나인 컴퓨터 판독 가능한 저장 매체.

청구항 22

삭제

청구항 23

삭제

청구항 24

삭제

청구항 25

삭제

청구항 26

삭제

청구항 27

삭제

청구항 28

삭제

명 세 서

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

<18> 본 발명은 일반적으로 메모리 관리 분야에 관한 것으로, 보다 구체적으로는 어드레스 변환 캐시들의 플러싱에 관한 것이다.

<19> 대부분의 컴퓨터 시스템들은 가상 어드레스 메커니즘을 제공하며, 그것에 의하여 가상 어드레스들은 물리적 어드레스들에 매핑된다. 메모리 위치에 액세스하기 위한 요청이 가상 어드레스를 사용해서 행해질 때, 액세스가 시도되는 목적 메모리 위치의 대응된 물리적 어드레스로 가상 어드레스가 변환된다. 어드레스 변환 테이블들의 세트가 가상 어드레스들과 물리적 어드레스들 사이의 매핑을 정의한다. 변환 테이블들은 보통 메모리에 저장되며, 그래서 어드레스의 변환은 테이블들을 판독하기 위해서 메모리 액세스를 요구한다. 테이블들을 판독하기 위해서 요구되는 이런 메모리 액세스 동작이 목적 위치에서 수행될 액세스 동작에 추가된다. 따라서, 가상 어드레스 할당이 사용될 때, 시스템에 의해서 수행되는 메모리 액세스들의 횟수는, 모든 액세스 요청들이 물리적 어드레스에 의해서 행해진 경우에 발생하게 되는 횟수에 비하여 두배가 될 수 있다. 일부 가상 어드레스들은 상기한 매핑이 스테이지들에서 디레퍼런스(dereference)되도록 요구한다는 점에서 다중 레벨화되어 있고, 그것은 어드레스 변환을 수행하는데 두 개 이상의 메모리 액세스를 취한다는 것을 의미한다(그것에 의해서 하나의 기본적인 액세스 요청을 수행하기 위해서 필요한 메모리 액세스는 세배 또는 그 이상이 요구된다).

<20> 어드레스를 변환하기 위해서 발생해야 하는 메모리 액세스들의 횟수를 줄이기 위해서, 많은 가상 어드레스 시스템들은 변환 색인 버퍼(TLB: Translation Lookaside Buffer)로 불리는 일종의 캐시를 사용한다. 최근에 액세스되었던 메모리 페이지들은 가까운 미래에 다시 액세스될 수 있기 때문에, 가상 페이지 기술어(descriptor)를 물리적 페이지 위치로 변환하기 위해서 어드레스 변환 테이블들이 사용되었던 경우에는, 가상 페이지와 물리적 페이지 사이의 대응 관계(correspondence)가 TLB에 캐싱된다. 어드레스 변환이 수행될 필요가 있을 때마다, TLB가 요청된 메모리의 유닛이 위치된 페이지에 대한 캐시된 매핑을 포함하는지를 판정하기 위해서 TLB가 조사된다. 관련된 매핑이 TLB에 캐시되었다면, 캐시 복사본이 사용된다; 그렇지 않다면, 어드레스는 변환 테이블들로부터 변환된다. TLB를 액세스하는 것이 메모리 내의 변환 테이블들을 액세스하는 것보다 더 빠르기 때문에, 일반적인 경우인, 연속적인 메모리 액세스들이 동일한 그룹의 페이지들에 위치된 경우에 TLB의 사용이 수행 속도를 증가시킨다.

<21> TLB들은 가상 메모리가 메모리 보호를 제공하기 위해서 사용될 때 일부 추가적인 문제를 야기한다. 메모리 보호는 어떤 소프트웨어 구성요소들이 어떤 물리적 메모리 페이지에 어떤 종류의 액세스(즉, 판독, 기록)를 수행

할 수 있는지를 제어하는 보안 정책(policy)을 실시하려고 시도한다; 이런 보호는 가상 메모리에 의해 가상 어드레스로부터 물리적 어드레스로의 변환에 대한 편집을 제어하는 것에 의해서 실시될 수 있다. (이러한 제어는 매핑을 생성하는 운영 체제이나 그러한 매핑에 대한 변화들을 필터링하는 어드레스 변환 제어(ATC: Address Translation Control) 시스템 중 하나에 의해서 실행될 수 있다.) 하지만, 어드레스 변환이 수정될 때, 구(old) 매핑은 여전히 TLB에 존재할 수 있다. 그러므로, 일부 소프트웨어 구성 요소를 위한 페이지에 대한 일부 액세스 권리를 취소하도록 어드레스 변환 테이블들이 편집될 때, 이러한 구 매핑들이 TLB로부터 플러싱될 때까지 상기한 구성 요소는 그 페이지로의 액세스를 유지할 것이다. 이것을 시행하는 보통의 방법은 관련된 TLB들이 그러한 동작의 일부분으로서 플러싱되도록 하는 것이다.

발명이 이루고자 하는 기술적 과제

<22> 하지만, TLB의 플러싱은 비용이 많이 들며, 특히 공유된 메모리 다중 프로세서상에서 그러하다. 새로운 보안 정책을 위반하는 오래된 매핑을 포함할 수 있는 모든 프로세서는 자신의 TLB를 플러싱하기 위한 신호를 받아야 한다; 이러한 신호를 주고 받는 것은 비교적 느린 프로세서간 인터럽트(IPI: interprocessor interrupt)를 통상적으로 요구한다. 또한, 플러싱은 그 자체로 비교적 비용이 많이 든다.

<23> 전술한 관점에서 종래 기술의 결점들을 극복한 메커니즘이 필요하다.

발명의 구성 및 작용

<24> 본 발명은 TLB들의 레이지 플러싱(lazy flushing)을 지원하는 메커니즘을 제공한다. 각각의 TLB에 대해서, 카운터가 보유되고, TLB가 플러싱될 때마다, 카운터는 증가된다. 각각의 프로세서가 자신의 TLB를 보유하고 있는 다중 프로세서 시스템에서는, 각각의 프로세서에 대한 개별적인 카운터가 있을 수 있으며, 해당 프로세서가 자신의 TLB를 플러싱할 때 특정 프로세서에 대한 카운터가 증가된다. 트리거 이벤트(triggering event)가 발생했을 때, 관련된 카운터의 값들이 트리거 이벤트가 완료된 후에 기록된다. "트리거 이벤트"는 오래된 TLB 엔트리들이 정책의 위배를 초래할 수 있는 방법으로, 어드레스 변환 맵, 또는 맵을 지배하는 메모리 액세스 정책에 영향을 주는 이벤트이다; 관련된 카운터는 그러한 오래된 엔트리들을 포함할 수 있는 TLB의 카운터이다. 예를 들면, 메모리의 어떤 페이지는 정책하에서 출입 금지(off-limit)로 선언될 수 있고, 상기 페이지에 대한 모든 매핑들은 어드레스 변환 맵으로부터 제거될 수 있다; 맵에서 페이지의 링크 제거는 트리거 이벤트의 예이고, 페이지로의 매핑을 포함하는 임의의 TLB들에 대한 카운터들이 연관된다. 그것은 트리거 이벤트가 수행되고, 트리거 이벤트의 결과들 또는 효과가 실제로 어드레스 변환 과정에서 사용되기 전에 상당한 시간이 경과하는 경우일 수 있다. (예를 들면, 메모리의 소정의 페이지에 대한 매핑은 소정의 시점에서 바뀔 수 있지만, 링크가 끊긴 페이지를 재사용하기 위해서 어떤 시도가 행해지기 전에 수백만 개의 동작들이 발생할 수 있다.) 트리거 이벤트의 결과들이 사용될 때, 관련된 TLB들(만약 있다면)이 트리거 이벤트 이후에 플러싱되지 않았는지를 판정하기 위해서, 저장된 카운터 값들이 현재의 카운터 값들과 비교된다. 관계된 임의의 TLB들이 기록된 값들과 일치하는 카운터 값들을 가진다면, 그러한 모든 TLB들은 기존의 방법으로 플러싱된다.

<25> 카운터 값이 변한다면, TLB가 안전하며, 즉 적용가능한 액세스 정책의 위배를 초래할 수 있는 오래된 TLB 엔트리들이 확실히 없고, 카운터 값이 변하지 않는다면 TLB는 불안전하다는 것을 주의해야 한다. 따라서, 임의의 크기의 카운터를 사용하고, 플러싱에 관한 임의의 방법으로 카운터를 바꾸는 것(또는 전혀 바꾸지 않는 것)이 안전하다. 다른 가능한 방법은 가장 마지막의 플러싱의 시간을 타임스탬프(timestamp)하기 위해서 실시간 카운터를 사용하는 것이다; 클록들이 일정한 시간 스큐(skew) 내에서 동기화되었다면, 플러싱의 개시자(initiator)는 변환이 수정되었던 시간을 단순히 기록할 수 있고, 플러싱의 타임스탬프가 변환 수정 시간을 적어도 상기 시간 스큐만큼 초과하게 될 때 상기 플러싱을 마무리하는 것으로 가정할 수 있다.

<26> 고보증(high-assurance) 및 비-고보증(non-high-assurance) 환경(각각, "오른쪽(right side)" 및 "왼쪽(left side)")이 나란히 존재하는 시스템에서, 및 모든 TLB 플러싱이 오른쪽으로의 트립(trip)을 요구하는 경우에, 본 발명의 메커니즘은 오른쪽으로의 불필요한 트립들을 피하기 위해서 사용될 수 있다. 예를 들면, 오른쪽과 왼쪽 사이의 각각의 변화는 TLB 플러싱을 초래할 수 있고, 각각의 프로세서에 대한 카운터는 프로세서가 오른쪽에서 왼쪽으로 이동할 때마다 증가할 수 있다. 다른 말로 하자면, 오른쪽이 TLB의 플러싱을 신뢰성있게 수행하기(또는 검증하기) 때문에, 카운터는 TLB가 메모리 액세스 제어 스킴(scheme)에 손상을 줄 수 있는 엔트리들을 포함할 수 있었던 마지막 시간을 반영한다. 페이지가 왼쪽으로 액세스될 수 없도록 (또는 왼쪽에 의해서 기록될 수 없도록)될 때마다, 페이지는 필연적으로 카운터(들)의 값으로 "타임스탬프화"된다. 상기 페이지가 프로세서에 의해서 액세스될 때, 프로세서가 (1) 현재 오른쪽에 있는지, 또는 (2) 페이지의 상태가 변한 이후로 TLB를 신뢰

성있게 플러싱했는지가 판정된다. 후자의 판정은 프로세서의 카운터 값이 상기 페이지를 위해서 저장된 값보다 더 크다는 것을 보장하는 것에 의해서 수행될 수 있다.

<27> 본 발명의 다른 특징들은 이하에서 설명될 것이다.

<28> 상술한 요약뿐만 아니라 바람직한 실시예들의 이하에서의 상세한 설명은 첨부된 도면들과 관련해서 더 잘 이해될 것이다. 본 발명을 예시할 목적으로 본 발명의 예시적인 구성을 도면에 도시한다. 하지만, 본 발명은 예시된 특정 방법들 및 수단들에 국한되지는 않는다.

<29> 〈실시예〉

<30> 개요

<31> 어드레스 변환 제어(ATC: Address Translation Control)는 가상 어드레스들을 물리적 어드레스들로 변환하는 데 사용되는 매핑들을 동적으로 제어하는 것에 의해서 메모리 액세스 정책을 구현하는 데 사용될 수 있다. 변환 색인 버퍼(TLB)가 매핑들을 캐싱하는 데 사용될 때, 액세스 정책의 현재의 상태와 일치하지 않는 구 매핑들이 TLB에 남아있을 수 있고, 그것에 의해서 카운터를 운영하는 데 사용하는 메모리를 상기 정책에 노출시킨다. TLB는 플러싱될 수 있지만, TLB의 플러싱은 비용이 많이 드는 동작이며, 바람직하게는, 필요 이상으로 자주 수행되지는 않아야 한다. 본 발명은 플러싱이 필요할 때까지 TLB의 플러싱을 지연시키는, 느슨한 TLB 플러싱을 위한 메커니즘을 제공한다. 상기 메커니즘은, 바람직하게는, 각각의 프로세서가 자기 자신의 TLB를 보유하는, 상기 자기 자신의 TLB들은 다른 프로세서들의 TLB들로부터 개별적으로 플러싱될 수 있는 다중 프로세서 시스템에서 사용될 수 있다.

<32> 예시적인 컴퓨팅 구성

<33> 도 1은 본 발명의 양상들이 구현될 수 있는 예시적인 컴퓨팅 환경을 도시한다. 컴퓨팅 시스템 환경(100)은 적절한 컴퓨팅 환경의 일례일 뿐이며, 본 발명의 기능 또는 사용의 범위에 대한 한계를 제시하는 것으로써 의도되지 않는다. 컴퓨팅 환경(100)은 예시적인 동작 환경(100)에서 도시되는 구성 요소들의 하나 또는 그것들의 조합에 대해서 어떠한 의존성이나 요구도 가지지 않는 것으로 해석되어야 한다.

<34> 본 발명은 많은 다른 범용 또는 특정 목적 컴퓨팅 시스템 환경 또는 구성으로 동작한다. 본 발명의 이용에 적합할 수 있는 공지된 컴퓨팅 시스템들, 환경들, 및/또는 구성들의 예로써는 개인용 컴퓨터들, 서버 컴퓨터들, 휴대용 또는 랩톱 디바이스들, 다중 프로세서 시스템들, 마이크로프로세서-기초 시스템들, 셋톱 박스들, 프로그래밍이 가능한 가전 제품들, 네트워크 PC들, 미니컴퓨터들, 메인프레임 컴퓨터들, 임베디드(embedded) 시스템들, 상기 시스템들 또는 디바이스들의 일부를 포함하는 분산 컴퓨팅 환경 등을 포함하지만, 그에 국한되지는 않는다.

<35> 본 발명은 컴퓨터에 의해서 실행되는, 프로그램 모듈들과 같은, 컴퓨터 실행가능한 명령들의 일반적인 문맥으로 설명될 수 있다. 일반적으로, 프로그램 모듈들은 특정 업무들을 수행하거나 특정 추상적인 데이터 형태들을 구현하는 루틴들, 프로그램들, 객체들(objects), 구성 요소들, 데이터 구조들 등을 포함한다. 본 발명은 업무들이 통신 네트워크 또는 다른 데이터 전송 매체를 통해서 링크되는 원격 처리 디바이스들에 의해서 수행되는 분산된 컴퓨팅 환경에서 실행될 수도 있다. 분산된 컴퓨팅 환경에서, 프로그램 모듈들 및 다른 데이터는 메모리 저장 디바이스들을 포함하는 로컬 및 원격 컴퓨터 저장 매체 모두에 위치될 수 있다.

<36> 도 1을 참조해 보면, 본 발명을 구현하는 예시적인 시스템은 컴퓨터(110)의 형태로 범용 컴퓨팅 디바이스를 포함한다. 컴퓨터(110)의 구성 요소들은 처리 유닛(120), 시스템 메모리(130), 및 시스템 메모리를 포함한 다양한 시스템 구성 요소를 처리 유닛(120)에 결합시키는 시스템 버스(121)를 포함하지만, 그에 국한되지는 않는다. 처리 유닛(120)은 다중-쓰레드(multi-threaded) 프로세서상에 지원되는 유닛과 같은 다수의 논리적 처리 유닛을 나타낼 수 있다. 시스템 버스(121)는 다양한 버스 구조 중 일부를 사용하는 로컬 버스, 주변 장치 버스, 및 메모리 버스 또는 메모리 제어기를 포함하는 여러가지 형태들의 버스 구조들 중 일부일 수 있다. 단지 예로써, 그러한 구조는 산업 표준 구조(ISA: Industry Standard Architecture) 버스, 마이크로 채널 구조(MCA: Micro Channel Architecture) 버스, 확장 ISA(EISA) 버스, 전자 공학 협회(VESA: Video Electronics Standards Association) 로컬 버스, 및 주변 장치 구성 요소 내부 상호 접속(PCI: Peripheral Component Interconnect) 버스(메자닌(Mezzanine) 버스로도 알려짐)를 포함하지만, 그에 국한되지는 않는다. 시스템 버스(121)는 통신 디바이스에서, 점대점 접속, 스위칭 패브릭(switching fabric) 등으로써 구현될 수도 있다.

<37> 컴퓨터(110)는 보통 다양한 컴퓨터 판독 가능한 매체를 포함한다. 컴퓨터 판독 가능한 매체는 컴퓨터(110)에

의해서 액세스될 수 있는 사용가능한 매체일 수 있고 휘발성 및 비휘발성 매체, 착탈식 및 비착탈식 매체 모두를 포함한다. 예로써, 컴퓨터 판독 가능한 매체는 컴퓨터 저장 매체 및 통신 매체를 포함할 수 있지만, 이에 한정되는 것은 아니다. 컴퓨터 저장 매체는 컴퓨터 판독 가능한 명령들, 데이터 구조들, 프로그램 모듈들 또는 다른 데이터와 같은 정보의 저장을 위한 임의의 방법 또는 기술로 구현되는 휘발성 및 비휘발성, 착탈식 및 비착탈식 매체 모두를 포함한다. 컴퓨터 저장 매체는 RAM, ROM, EEPROM, 플래시 메모리 또는 다른 메모리 기술, CDROM, 디지털 다용도 디스크들(DVD) 또는 다른 광학 디스크 저장 장치, 자기 카세트, 자기 테이프, 자기 디스크 저장 장치 또는 다른 자기 저장 디바이스들, 또는 희망된 정보를 저장하는 데 사용될 수 있고 컴퓨터(110)에 의해서 액세스될 수 있는 다른 매체를 포함하지만, 그에 국한되지는 않는다. 통상, 통신 매체는 컴퓨터 판독 가능한 명령들, 데이터 구조들, 프로그램 모듈들 또는 다른 데이터를 반송파(carrier wave) 또는 다른 수송 메커니즘와 같은 변조된 데이터 신호로 구체화하고 임의의 정보 배달 매체를 포함한다. 용어 "변조된 데이터 신호"는 신호 내의 정보를 부호화하는 방법으로 그 신호의 하나 이상의 특성을 설정 또는 변경시킨 신호를 의미한다. 예로써, 통신 매체는 유선 네트워크 또는 직통 유선 접속(direct-wired)과 같은 유선 매체, 및 음향(acoustic), RF, 적외선 및 다른 무선 매체와 같은 무선 매체를 포함하나, 이에 한정되는 것은 아니다. 상기 매체의 모든 조합들이 또한 컴퓨터 판독 가능한 매체의 범위내에 포함되어야 한다.

<38> 시스템 메모리(130)는 ROM(131) 및 RAM(132)와 같은 휘발성 및/또는 비휘발성 메모리의 형태로 된 컴퓨터 저장 매체를 포함한다. 기동시, 컴퓨터(110) 내부의 요소들 사이에서 정보를 전송하는 것을 돕는 기본 루틴들을 포함하는, 기본 입/출력 시스템(133)(BIOS)은 보통 ROM(131)에서 저장된다. RAM(132)는 보통 처리 유닛(120)에 즉시 액세스될 수 있고/있거나 현재 처리 유닛(120)에 의해서 동작되는 프로그램 모듈들 및/또는 데이터를 포함한다. 예로써, 도 1은 운영 체제(134), 애플리케이션 프로그램들(135), 다른 프로그램 모듈들(136), 및 프로그램 데이터(137)를 도시하지만, 이에 한정되는 것은 아니다.

<39> 컴퓨터(110)는 다른 착탈식/비착탈식, 휘발성/비휘발성 컴퓨터 저장 매체를 또한 포함할 수 있다. 단지 예로써, 도 1은 비착탈식, 비휘발성 자기 매체에 기록할 수 있거나 이로부터 판독할 수 있는 하드디스크 드라이브(141), 착탈식, 비휘발성 자기 디스크(152)에 기록할 수 있거나 이로부터 판독할 수 있는 자기디스크 드라이브(151), 및 CDROM 또는 다른 광학 매체와 같은 착탈식, 비휘발성 광학 디스크(156)에 기록할 수 있거나 이로부터 판독할 수 있는 광학 디스크 드라이브(155)를 도시한다. 예시적인 운영 환경에서 사용될 수 있는 다른 착탈식/비착탈식, 휘발성/비휘발성 컴퓨터 저장 매체는 자기 테이프 카세트, 플래시 메모리 카드들, 디지털 다용도 디스크들, 디지털 비디오 테이프, 고체 상태 RAM, 고체 상태 ROM 등을 포함하지만 그에 국한되지는 않는다. 하드디스크 드라이브(141)는 보통 인터페이스(140)와 같은 비착탈식 메모리 인터페이스를 통해서 시스템 버스(121)에 접속되고, 자기 디스크 드라이브(151) 및 광학 디스크 드라이브(155)는 인터페이스(150)와 같은, 착탈식 메모리 인터페이스에 의해서 시스템 버스(121)에 보통 접속된다.

<40> 상기 설명되었고 도 1에서 도시된 드라이브들 및 그 관련된 컴퓨터 저장 매체는 컴퓨터 판독가능한 명령들, 데이터 구조들, 프로그램 모듈들 및 컴퓨터(110)의 그밖의 다른 데이터를 저장한다. 도 1에서, 예를 들면, 하드디스크 드라이브(141)는 운영 체제(144), 애플리케이션 프로그램들(145), 다른 프로그램 모듈들(146), 및 프로그램 데이터(147)를 저장하는 것으로 도시된다. 이러한 구성 요소들은 운영 체제(134), 애플리케이션 프로그램들(135), 다른 프로그램 모듈들(136), 및 프로그램 데이터(137)와 다를 수도 있고 동일할 수도 있음을 주의해야 한다. 운영 체제(144), 애플리케이션 프로그램들(145), 다른 프로그램 모듈들(146), 및 프로그램 데이터(147)에 다른 번호가 주어졌다는 것은 적어도 이들이 서로 다른 사본이라는 것을 나타낸다. 사용자는, 키보드(162), 일반적으로 마우스라 지칭되는 포인팅 디바이스(161), 트랙볼 또는 터치 패드로 불려지는 입력 디바이스들을 통해서 명령들 및 정보를 컴퓨터(110)로 입력할 수 있다. 다른 입력 디바이스들(도시되지는 않음)은 마이크로폰, 조이스틱, 게임 패드, 위성 접시, 스캐너 등을 포함할 수 있다. 이들 및 다른 입력 디바이스들은 시스템 버스에 연결된 사용자 입력 인터페이스(160)를 통해서 처리 유닛(120)에 종종 접속되지만, 병렬 포트, 게임 포트 또는 범용 시리얼 버스(USB)와 같은 다른 인터페이스, 및 버스 구조들에 의해서 접속될 수 있다. 모니터(191) 또는 디스플레이 디바이스의 다른 형태가 비디오 인터페이스(190)와 같은, 인터페이스를 통해서 시스템 버스(121)에 또한 접속될 수 있다. 모니터이외에, 컴퓨터들은 또한 출력 주변 장치 인터페이스(195)를 통해서 접속될 수 있는, 스피커들(197) 및 프린터(196)와 같은 다른 주변 장치 출력 디바이스들을 포함할 수 있다.

<41> 컴퓨터(110)는 원격 컴퓨터(180)와 같은, 하나 이상의 원격 컴퓨터들로의 논리 접속들을 사용해서 네트워크 환경에서 동작할 수 있다. 원격 컴퓨터(180)는, 개인용 컴퓨터, 서버, 라우터, 네트워크 PC, 피어 디바이스 또는 다른 공통 네트워크 노드일 수 있고, 도 1에서는 오직 하나의 메모리 저장 디바이스(181)가 도시되어 있지만, 보통 컴퓨터(110)와 관련해 상기 설명된 요소들 중 대부분 또는 모두를 포함한다. 도 1에서 도시된 논리 접속

들은 근거리 통신망(LAN)(171) 및 원거리 통신망(WAN)(173)을 포함하지만, 다른 네트워크들을 또한 포함할 수 있다. 그러한 네트워크 환경들은 사무실들, 회사 전체에 걸친 컴퓨터 네트워크들, 인트라넷 및 인터넷에서 일반적이다.

<42> LAN 네트워크 환경에서 사용될 때, 컴퓨터(110)는 네트워크 인터페이스 또는 어댑터(170)를 통해서 LAN(171)으로 접속된다. WAN 네트워크 환경에서 사용될 때, 컴퓨터(110)는 통상적으로 인터넷과 같은, WAN(173)을 통해서 통신들을 성립시키기 위한 모뎀(172) 또는 다른 수단을 포함한다. 내장형 또는 외장형일 수 있는, 모뎀(172)은 사용자 입력 인터페이스(160), 또는 다른 적절한 메커니즘을 통해서 시스템 버스(121)로 접속될 수 있다. 네트워크 환경에서, 컴퓨터(110) 또는 그의 일부와 관련하여 설명된 프로그램 모듈들은 원격 메모리 저장 디바이스에서 저장될 수 있다. 예로써, 도 1은 원격 애플리케이션 프로그램들(185)을 메모리 디바이스(181)상에 상주하는 것으로 도시하지만, 이에 국한되는 것은 아니다. 도시된 네트워크 접속들은 예시적이며 컴퓨터들 간의 통신 링크를 성립시키기 위한 다른 수단이 이용될 수 있음을 이해해야 할 것이다.

<43> 예시적인 가상 어드레스 스킴

<44> 도 2는 가상 어드레스 시스템의 예를 도시한다. 도 2에서 도시된 예는 페이지-형태 가상 어드레스 스킴이지만, 가상 어드레스 할당이 세그멘테이션(segmentation)과 같은, 다른 모델들에 기초할 수 있다고 이해될 것이다. 도 2에서 도시된 스킴은 INTEL x86 프로세서 상에서 이용 가능한 가상 어드레스 할당 스킴들 중의 하나와 같은, 두 개의 레벨 어드레스(two-level) 스킴이다. 이하에서 설명되듯이, 가상 페이지 식별자를 물리적 페이지로 변환하기 위해서 두 개의 레벨의 인다이렉션(indirection)을 사용해야 한다는 의미에서 이 스킴은 "두 개의 레벨"이다.

<45> 이러한 페이지 스킴에서, 페이지 디렉토리(202)는 엔트리들의 세트를 포함한다. 엔트리의 예시적 구조는 도 3과 관련해 이하에서 보다 구체적으로 설명되지만, 본질적으로, 각각의 엔트리는 페이지 테이블들(204(1), 204(2), 또는 204(3))과 같은, 특정 페이지 테이블의 물리적 위치(즉, 페이지 프레임 번호, "PFN(Page Frame number)")를 확인한다. 각각의 페이지 테이블은, 차례로, 각각의 엔트리가 물리적 위치(페이지 프레임 번호)를 페이지들(206(1), 206(2), 206(3), 또는 206(4))과 같은 특정 데이터 페이지로 식별하는, 엔트리들의 세트를 포함한다. 데이터 페이지들은 RAM(132)의 규정된 길이의 인접한 부분들이다. 데이터 페이지들은 데이터의 일부 형태를 저장할 수 있고, 정규 데이터를 저장하는 것 이외에, 데이터 페이지들은 또한 페이지 디렉토리(202) 및 페이지들(204(1) 내지 204(3))의 내용들을 저장하는 데 또한 사용됨을 주의해야 한다. 따라서, 소정의 페이지는 디렉토리, 테이블, 데이터 페이지일 수 있거나, 상기 3개의 구조들의 조합으로써 다수의 역할을 할 수 있다.

<46> 특정 페이지의 위치를 찾기 위해서 페이지 디렉토리(레벨 1) 및 페이지 테이블(레벨 2) 모두를 조사하는 것이 필요하기 때문에, 도 2에서 도시된 가상 어드레스 스킴은 두 개의 레벨 가상 어드레스 스킴이다. 본 분야의 당업자들은 임의의 레벨 수를 갖는 가상 어드레스 시스템을 설계하는 것이 가능하며, 본 발명의 원리들이 그러한 모든 가상 어드레스 스킴들에 적용될 수 있음을 이해할 것이다. 본 분야에 공지된 것처럼, INTEL x86 프로세서는 한 개, 두 개 또는 세 개의 레벨들을 갖는 가상 어드레스들을 지원하며, 보통, "넓은(large)" 페이지들(즉, 길이가 4 메가바이트들인 페이지들)은 1개의 레벨 가상 어드레스들을 사용하는 반면에, "작은" 페이지들(즉, 길이가 4 킬로바이트들인 페이지들)은 두 개의 레벨 가상 어드레스들을 사용하는, "혼성(hybrid)" 스킴을 사용한다.

<47> 도 2의 페이지 스킴에서, 페이지 상의 일정 바이트는 페이지 디렉토리 오프셋(211), 페이지 테이블 오프셋(212), 및 페이지 오프셋(213)을 포함하는, 가상 어드레스(210)에 의해서 식별될 수 있다. 따라서, 물리적 어드레스의 위치를 찾기 위해서, 어드레스들의 변환을 수행하는 메모리 관리 유닛(MMU: Memory Management Unit)(220)은 페이지 디렉토리(202)에서 특정 엔트리의 위치를 찾기 위해서 페이지 디렉토리 오프셋(211)을 사용한다. 예를 들면, 오프셋(211)은 0일 수 있으며, 이는 페이지 디렉토리(202)에서의 0번째 엔트리가 참고되어야 함을 지시한다. 이 엔트리는 페이지 테이블이 저장되는 PFN을 포함하며, 그래서 MMU(220)는 이러한 PFN을 페이지 테이블들 중 하나(즉, 페이지 테이블(204(1))의 위치를 찾기 위해서 사용한다. MMU(220)는 그후 페이지 테이블 오프셋(212)을 식별된 페이지 테이블로의 인덱스로써 사용하며, 상기 오프셋에서 검색된 엔트리를 가져온다. 엔트리는 데이터 페이지(즉, 페이지(206(1))의 PFN을 포함하며, 그래서 MMU(220)는 물리적 메모리의 특정 바이트의 위치를 찾기 위해서 식별된 페이지의 기본 어드레스에 페이지 오프셋(213)을 추가시킨다. MMU(220)는 어드레스들의 단순한 변환이외에 다른 다양한 기능들을 수행하기 위해서 적응될 수도 있다: 예를 들어, MMU(220)는 테이블에서의 페이지의 엔트리가 "존재하지 않는 것"으로 표시되면 디스크로부터 페이지를 불러올 수 있고; MMU(220)는 페이지가 "판독용"으로 표시되면 기록 액세스를 허용치 않을 수 있다.

<48> 도 2의 가상 어드레스 스킴에서, 페이지 디렉토리의 위치(즉, PFN)는 그 자체가 저장 위치(201)에 저장된다. MMU(220)는 가상 어드레스(210)를 변환하기 시작할 때 페이지 디렉토리(202)의 위치를 찾기 위해서 이러한 저장 위치의 내용들을 사용한다. 따라서, 다수의 페이지 맵들이 존재할 수 있고, 특정한 맵이 소정의 맵의 페이지 디렉토리의 PFN을 포함하도록 저장 위치(201)의 내용들을 설정하는 것에 의해서 현재 사용을 위해서 선택될 수 있다. INTEL x86 프로세서의 예에서, 저장 위치(201)는 CR3으로 불리는 레지스터에 상당한다.

<49> 어드레스 변환 제어를 통한 고보증 환경 및 은닉 메모리

<50> 상기 설명된 가상 어드레스 시스템의 한가지 특징은 소정의 어드레스 변환 맵하에서 가상 어드레스에 의해서 매핑되지 않는 물리적 어드레스가 있을 수 있다는 점을 이해해야 할 것이다. 따라서, 거의 모든 액세스 요청들이 가상 어드레스에 의해서 행해지는, INTEL x86 프로세서와 같은 시스템에서, 소스의 어드레스 변환 맵이 출입 금지(off-limit) 메모리를 가리키지 않도록 보장하는 것에 의해서 주어진 소스에 대해 출입 금지인 물리적 메모리의 부분을 만드는 것이 가능하다. 그러한 출입 금지 메모리를 달성하기 위해서 어드레스 변환 맵에 대한 제어를 실행하는 것은 어드레스 변환 제어(ATC(Address Translation Control))라고 불린다. 메모리의 출입 금지 부분은 때때로 "은닉 메모리"로 불리며, ATC는 은닉 메모리를 달성하기 위한 한가지 방법이다.

<51> 은닉 메모리의 한가지 사용은 고보증 환경 및 비-고보증 환경이 동일한 머신에서 공존할 때이다. 고보증 환경에는 비-고보증 환경에 의해서 액세스될 수 없는 은닉 메모리가 제공될 수 있다. 고보증 환경의 개념-및 은닉 메모리와의 관계-은 도 3을 참조하여 이하에서 설명된다.

<52> 도 3은 동일한 머신에서 공존하는 두 개의 환경들을 도시한다: 고보증 환경(360), 및 비-고보증 환경(350). 편의상, 이러한 환경들은 RHS(right-hand-side) 및 LHS(left-hand-side)로 각각 불릴 수 있다; RHS(360)는 고보증이며, LHS(350)는 비-고보증이다. 환경이 "고보증"이라는 것이 의미하는 것은 환경에서 수행되는 기능들이 정확하게 수행될 수 있는 높은 레벨의 보증이 있다는 것이다. 이렇게, LHS(350)는 다양한 기능들(306(1), 306(2),...306(n))을 수행하고, RHS(360)는 다양한 기능들(308(1), 308(2),...,308(m))을 수행한다. LHS(350)는 기능들(306(1) 내지 306(n))이 어떻게 행동할 것인지를 설명하는 명세(302;specification)와 관련되어 진다. 유사하게, RHS(360)는 기능들(308(1) 내지 308(m))이 어떻게 행동할 것인지를 설명하는 명세(304)와 관련되어 진다. 명세(302 및 304)는 기록될 수도 있고 그렇지 않을 수도 있다. 예를 들면, LHS(350)는 다양한 서비스들, 드라이브들, 시스템 호출 등이 어떻게 행동할 것인지를 설명하는 상세한 기록된 매뉴얼과 함께 운영되는 업무용 운영 체제일 수 있다. 또는, 단순히 주어진 환경이 어떻게 행동할 지에 대한 일반적인 이해가 존재할 수 있고, 이러한 이해는 명세를 구성할 수 있다.

<53> 명세가 어떠한 형태를 갖는 지에 관계없이, 거의 모든 소프트웨어가 소프트웨어가 예상치 않은 방법으로 행동하도록 하는 버그들, 백도어(backdoor)(알려진 것 및 발견되지 않은 것 모두), 논리 오류 등을 포함하고 있다는 것을 이해할 것이다. 하지만, 소프트웨어의 주어진 부분이 희망된 방법으로 행동할 것이라는 보증의 레벨, 즉 소프트웨어의 행동이 그 명세에서 설명된 것과 실제로 일치하는 보증의 레벨을 평가하는 것이 가능하다. 도 3의 예에서, RHS(360)는, RHS(360)가 수행하도록 설계된, 기능들(308(1) 내지 308(m))이 실제로 명세(304)에 일치하여 수행되는 비교적 높은 레벨의 보증이 있다는 의미에서 "고보증"이다. 유사하게, LHS(350)는 LHS(350)가 명세(302)에 일치하여 기능들(306(1) 내지 306(n))을 수행하는 비교적 낮은 레벨의 보증이 있다는 의미에서 "비-고보증"이다. 이런 의미에서 "비교적 높음" 및 "비교적 낮음"은 RHS(360)가 명세(304)에 따라 행동할 보증의 레벨이 LHS(350)가 명세(302)에 따라서 행동할 보증의 레벨보다 높다는 것을 의미한다.

<54> 일반적으로 MICROSOFT WINDOWS 운영 체제들 중 하나와 같은, 모든 기능을 갖춘 업무용 운영 체제는 비-고보증 환경의 경우이다. 그러한 환경들은 디바이스 드라이버들, 플러그인, 및 다른 형태의 확장 도구들이 자유롭게 추가될 수 있는 개방형 구조를 지원하며, 그것은 생각할 수 있는 모든 상황에서 이러한 운영 체제의 행동을 확인하기 어렵게 만들 수 있다. 따라서, 어떠한 형태의 보안이 희망되어 질 때, 고보증 운영 체제와 나란히 모든 기능을 갖춘 운영 체제를 작동시키는 것이 유용하다. 고보증 운영 체제는 적은 수의 기능들을 제공하는 작은 운영 체제이다. 고보증 운영 체제에 의해서 제공되는 기능이 작기 때문에, 운영 체제의 동작에 영향을 줄 수 있는 변수들이 다 적어지고, 그래서 고보증 운영 체제의 행동은 높은 단계의 확실성을 검증하기가 더욱 쉽다.

<55> 바람직한 실시예에서, 고보증 환경은 은닉 메모리, 즉 비-고보증 환경으로 액세스될 수 없는 메모리의 부분을 포함한다. 따라서, RHS(360)는 LHS(350)하에서 작동되는 처리들에 의해서 기록되거나 판독될 위험없이, 은닉 메모리에 비밀 데이터(즉, 암호화 키(cryptographic key))를 저장할 수 있다. 예를 들면, 명세(304)는 RHS(360)기 외부 간섭으로부터 정보를 지키는 기능을 가지며, 은닉 메모리는 RHS(360)가 이러한 기능을 수행하는 것을 허용하도록 제공할 수 있다. 게다가, RHS(360)가 다양한 기능을 수행하기 위해서 사용하는 코드는,

LHS(350)하에서 운영되는 처리들을 이러한 코드를 (RHS(360)이 명세 외부에서 활동하도록 할 수 있는)다른 코드로 중첩쓰기하지 않도록 방지하기 위해서, 은닉 메모리에 저장될 수 있다. 도 3은 RHS(360)에 의해서 사용될 수 있는 은닉 메모리(312)를 도시한다. 물리적 어드레스 공간(310)은 주어진 컴퓨팅 디바이스상에서 사용 가능한 모든 물리적 메모리 위치들을 포함한다. 은닉 메모리(312)는 상기 위치들의 서브세트이다. 도 3에서 도시된 바와 같이, RHS(360)는 모든 물리적 어드레스 공간(310)을 액세스하지만, LHS(350)는 은닉 메모리(312)를 구성하는 물리적 어드레스 공간(310)의 일부를 액세스할 수 없다. (도 3이 어드레스 공간의 은닉된 및 은닉되지 않은 부분들이 각각 인접하여 있는 것으로 도시되어 있지만, 그러한 인접에 대한 어떠한 요구도 없다는 것을 주의해야 한다. 게다가, RHS(360)가 전체 물리적 어드레스 공간에 대한 액세스를 가지거나, LHS(350)가 은닉 메모리(312)의 외부에 있는 물리적 어드레스 공간의 모든 부분에 대한 액세스를 가질 필요는 않다.)

<56> 상기 설명된 바와 같이, 거의 모든 메모리 액세스 요청들이 물리적 어드레스에 의해서 행해질 수 있는 특정한 시스템들이 존재한다. 그러한 시스템 상에서 은닉 메모리(312)를 구현하는 한가지 방법은 은닉 메모리(312)를 위한 어떠한 가상 어드레스들도 LHS(350)에 노출되지 않는 방법으로 어드레스 변환 맵들의 내용을 제어하는 것이다. (물리적 어드레스에 의해서 목적(target)을 식별하려는 액세스 요청들의 몇 가지 형태가 존재할 수 있는 경우에, 은닉 메모리로의 액세스가 직접적 메모리 액세스 디바이스들로부터 또는 다른 소스들로부터 물리적 액세스 요청들을 필터링하는 배제 백터(exclusion vector)와 같은, 일부 보조적 협력 메커니즘에 의해서 제한될 수 있다.) 많은 알고리즘들은 LHS(350)가 은닉 메모리(312)로 이끌 수 있는 어드레스 변환 맵을 사용할 수 없도록 보장할 수 있지만, 이러한 알고리즘의 핵심 개념은 (1) 프로세서가 LHS(350)에서 동작할 때, CR3(도 2에서 도시되는, 저장 위치(201))으로 불러와지는 일정한 맵은 은닉 메모리(312)에 포함되는 페이지들로 이끌지 않는 것을 보장하며; (2) LHS(350)에서 활성 맵을 편집하기 위한 어떠한 시도가 있는 경우, 그 편집이 은닉 메모리(312)의 페이지로의 링크를 초래하지 않도록 보장하기 위해서 계획된 편집을 평가한다.

<57> 다음은 ATC를 통해서 은닉 메모리를 구현하는 예시적인 알고리즘이다.

<58> D1을 페이지 디렉토리들로 사용될 수 있는 페이지들의 세트라고 하자. D2는 페이지 테이블들로써 사용될 수 있는 페이지들의 세트로 하자. D는 D1 및 D2의 합집합이라고 하자(즉, D = D1 U D2). "존재하는(present)" 것으로 표시된 (즉, 존재하는(present) 비트가 설정되어있는) 페이지 디렉토리 또는 페이지 테이블에서의 각각의 엔트리는 "링크(link)"로 불려진다. D2에서의 페이지는 D1에서의 일정 페이지로부터 문제의 D2 페이지로의 적은 판독-기록 링크가 있다면 "기록-활성(write-active)"이다. ("작은" 링크는 디렉토리에서 테이블로의 링크, 즉 궁극적으로는 작은 페이지로 이끄는 디렉토리에서의 링크이다. "넓은" 링크는 넓은 페이지를 지시하는 디렉토리에서의 링크이다.) 페이지의 일부 엔티티가 판독 및/또는 기록 액세스가 허용되는 페이지를 규정하는 정책이 있다고 가정한다.

<59> 다음의 불변량들은 유지된다:

<60> - CR3는 D1에 존재한다;

<61> - 모든 D1 및 D2 페이지들은 관련된 정책하에서 판독 가능하다;

<62> - D1으로부터의 모든 작은 링크는 D2 페이지를 지시한다;

<63> - D2로부터의 링크들은 정책하에서 판독 가능한 페이지를 지시한다;

<64> - 기록-활성 D2 페이지로부터의 모든 판독-기록 링크는 정책하에서 기록 가능하며 D에 존재하지 않는 페이지를 지시한다;

<65> - D1페이지로부터의 넓은 링크의 넓은 페이지 목적에 포함된 모든 작은 페이지는 정책하에서 판독될 수 있다; 링크가 판독-기록이라면, 작은 페이지도 정책하에서 기록될 수 있으며 D1에는 존재하지 않는다.

<66> 예를 들면, 정책이 은닉 메모리(312)를 액세스될 수 없는 것으로 규정하면, LHS(350)에 의해서 사용될 수 있는 모든 어드레스 변환 테이블들을 위한 상기 불변량들을 보유하는 것은 가상 어드레스에 의해서 그 목적을 식별하며 LHS(350)에서 발생하는 어떤 액세스 요청도 은닉 메모리(312)에 도달할 수 없다고 보장할 것이다.

<67> ATC로서 변환 색인 버퍼들의 사용

<68> 변환 색인 버퍼(TLB)는 본질적으로 매핑 데이터가 저장되는 캐시이다. TLB의 기본 개념은 최근에 액세스되었던 페이지들이 다시 액세스될 것이고, 따라서 주어진 가상 페이지에서 대응하는 물리적 페이지로의 최근에 사용된 매핑들이 TLB에서 저장된다는 것이다. (적어도 INTEL x86 프로세서상에서 사용되는 가장 일반적인 두 개의 레

벨 가상 어드레스 스킴에서) 어드레스 변환 테이블로부터 매핑을 계산하는 것은 두 개의 메모리 액세스를 요구한다: 관계된 페이지 테이블의 위치를 찾기 위해서 페이지 디렉토리로의 첫 번째 액세스와 그후, 관계된 데이터 페이지를 찾기 위해서 페이지 테이블로의 두 번째 액세스. 액세스가 요청되는 실제 목적 위치는 이들 두 개의 (다른) 메모리 액세스들이 목적 데이터의 물리적 위치를 찾기 위해서 수행된 후에 오직 액세스될 수 있다. TLB는 최근에 사용된 매핑 정보가 유지되는 빠른 메모리를 제공하는 것에 의해서 그러한 추가적인 메모리 액세스에 대한 필요성을 없앤다. 따라서, 수행될 필요가 있는 모든 어드레스 변환에 대해서, 프로세서는 주어진 가상 페이지가 TLB에 저장되어 있는지를 판정하기 위해서 먼저 프로세서의 TLB를 참조한다. 매핑 데이터가 TLB에서 존재하지 않는다면 어드레스 변환 테이블이 참조된다.

<69> 도 4는 프로세서가 TLB(402)를 사용하는 방법을 도시한다. 프로세서(120)는 코드의 부분을 실행하고, 상기 실행 동안에, 명령들이 메모리(400)의 판독 또는 기록 목적 위치(406)에 발생한다. 명령은 목적 위치(406)의 물리적 어드레스를 식별하는 것이 아니라, 목적 위치(406)의 가상 어드레스를 지정한다. 이렇게, 프로세서(120)(또는, 많은 경우에, 프로세서(120)와 결합된 메모리 관리 유닛)는 가상 어드레스를 물리적 어드레스로 변환해야 한다. 프로세서(120)는 먼저 가상 어드레스를 변환할 때 사용될 수 있는 매핑이 저장되었는지를 판정하기 위해서 TLB(402)를 검색한다(곡선 1). 그러한 매핑이 존재한다면, 이러한 매핑이 어드레스를 변환하기 위해서 사용된다. 그러한 매핑이 존재하지 않는다면, 어드레스 변환 맵(404)의 관계된 부분들이 메모리(400)로부터 판독되고(곡선 2), 이러한 맵 부분들은 가상 어드레스를 변환하는 데 사용된다. 어드레스가 어떻게 변환되는지와 상관없이, 변환된 어드레스는 물리적 목적 위치(406)를 액세스하는 데 사용된다(곡선 3). TLB가 효율성을 제공하는 이유는 TLB(402)에서 정보를 검색하는 것이 메모리(400)에서 어드레스 변환 맵(404)을 판독하는 것보다 더 빠르기 때문이다. 게다가, 어떤 설계에서는, TLB(402)에 저장된 데이터가 두 단계 변환 과정을 한 단계로 축약할 수 있다: 즉, 어드레스 변환 맵을 사용하는 것은 개별적인 단계들에서 페이지 디렉토리 및 페이지 테이블을 판독하는 것을 요구한다; 하지만, 단독 PD 오프셋/PT 오프셋 조합은 TLB(402)에 저장될 수 있고, 그것에 의해서 가상 페이지가 단독 단계로 PD/PT 오프셋 조합에 의해서 식별하는 것을 허용한다.

<70> ATC 시스템에서 TLB들을 사용할 때, ATC 불변량들은 TLB를 통해서 사용가능한 변환들을 고려하기 위해서 수정된다. 예를 들면, 앞서 설명했던 예시적인 ATC 알고리즘에서, 링크의 정의는 다음과 같이 수정될 수 있다: 링크가 물리적 메모리에 존재하거나 일부 TLB에 캐시되면 한 페이지에서 다른 페이지로의 링크가 존재한다. TLB들의 내용이 대부분의 구조들(즉, x86 프로세서들)에서 직접적으로 관측될 수는 없지만, 변환들이 오직 메모리를 통해서 TLB들을 입력하기 때문에 TLB들의 내용들이 묶여 있을 수 있다. 예시적인 알고리즘에서 불변량들을 사용할 때, TLB들은 페이지로의 판독 또는 판독-기록 액세스를 포기하면서, D1 또는 D2로부터 페이지를 제거하는 것에 대해서, 및 기록-활성에서 기록 불활성으로 또는 그 반대 방향으로 페이지의 상태를 변화시키는 D2로의 기록 또는 추가에 대해서 플러싱되어야 한다.

<71> 하지만, TLB의 플러싱은 실행을 느리게하는 비용이 많이 드는 작업이기 때문에, TLB 플러싱을 많이 줄여야 할 실제적인 동기가 있다. 먼저, TLB는 유용한 매핑 데이터로 차있을 정도로 효율성을 개선시키고, TLB의 플러싱은 TLB로부터 이러한 매핑 데이터를 제거한다. 두 번째, 실제 플러싱은 다음의 이유로 비용이 많이 든다: 도 3에서 설명된 구조에서, 은닉 메모리(312)의 유지 관리는 카운터-정책 매핑들의 부재에 따른다. 카운터-정책 매핑의 부재는, 바람직한 실시예에서는, 고보증 환경을 제공하는 요소이기 때문에, 매핑에 대한 제어는 그 자체로 고보증 환경에서 수행되어야 한다. 이렇게, 어드레스 변환 맵에 대한 제어는 고보증 환경에서 수행되어야 할 뿐 아니라, TLB의 플러싱 역시 고보증 환경에서 수행되어야(적어도 검증되어야) 한다. 따라서, ATC 시스템의 일부로서 신뢰되는 모든 TLB 플러싱은 현재 환경이 LHS(350)에서 RHS(360)로 변할 것을 요구한다. 한 환경에서 다른 환경으로 컨텍스트를 바꾸는 과정은, 그 자체로, 비용이 많이 들며, 그것은 플러싱의 비용을 증가시킨다. 따라서, 필요한 것보다 자주 TLB를 플러싱하는 것을 피하는 것이 필요하다.

<72> 본 발명은 이러한 플러싱들의 일부를 지연시키고 가능하다면 회피하는 메커니즘을 제공한다. 본 발명은, 일 실시예에서, 두 개의 하부 메커니즘에 기초한다. 제1 하부 메커니즘은 실제 메모리 맵을 수정하지 않는 동작들(예컨대, 페이지를 D2에 추가하는 것, 또는 페이지를 기록들에 의해 액세스될 수 없도록 하는 것)이 그러한 동작들의 완결이 메모리에 일정한 수정을 허용하도록 요구될 때까지 개념적으로 지연될 수 있는 것을 제공한다. 예를 들면, (보안 정책으로) 페이지로의 판독 액세스를 포기하는 활동은 상기 페이지가 수정될 가능성이 있을 때까지(다른 계산적 엔티티가 페이지로의 판독-기록 매핑을 생성할 때) 지연될 수 있다. 제2 하부 메커니즘은 페이지로의 특정 매핑이 특정 TLB에 존재하는지를 보다 명확하게 추적하기 위해서 타임스탬프들을 사용한다.

<73> 앞서 설명된 예시적 알고리즘에서, 플러싱들을 요구하는 지연될 수 있는 동작들은 두 개의 부류로 나뉘어질 수 있다: 페이지가 판독되는 것을 허용하는 일정한 변환들을 TLB들로부터 제거하기를 요구하는 부류(페이지에 대한

판독 액세스를 포기하는 것, 또는 D1 또는 D2 중 하나로부터 페이지를 제거하는 것), 및 페이지가 수정되는 것을 허용하는 일정한 변환들을 TLB들로부터 제거하기를 요구하는 부류(페이지에 대한 기록 액세스를 제거하는 것). 전자 "판독 플러싱 동작들" 및 후자 "기록 플러싱 동작들"을 호출하는, 이러한 동작들은 다음과 같이 지연될 수 있다.

<74> - 판독-플러싱 동작들은 일정한 계산적 엔티티가 페이지로의 판독-기록 매핑을 생성하려고 시도할 때까지, 또는 일정한 디바이스가 직접적인 메모리 액세스를 통해서 페이지를 수정할 수 있을 때까지, 또는 페이지가 ATC 알고리즘 그 자체에 의해서 수정될 때까지 지연될 수 있다.

<75> - 기록-플러싱 동작들은 일정한 계산적 엔티티가 페이지로의 판독 또는 판독-기록 매핑을 생성하려고 시도할 때까지, 또는 일정한 디바이스가 직접적인 메모리 액세스를 통해서 페이지를 수정할 수 있을 때까지, 또는 페이지가 ATC 알고리즘 그 자체에 의해서 수정될 때까지 지연될 수 있다.

<76> 판독-플러싱 동작을 수행할 때, TLB가 여전히 잠재적으로 문제의 페이지로의 판독 매핑을 포함하고 있다면 TLB를 플러싱하는 것이 오직 필요하다. 이것은 TLB가 페이지로의 판독 매핑이 있었던 마지막 시간 이후에 플러싱이 되지 않았다면 그러한 경우가 될 것이다. 유사하게, 판독-플러싱 동작을 수행할 때, 마지막으로 페이지로의 판독-기록 매핑이 있은 이후에 플러싱되지 않았다면 TLB를 플러싱하는 것이 오직 필요하다.

<77> TLB가 페이지로의 판독/판독-기록 매핑이 있었던 이후에 플러싱되었는지 판정하기 위해서, 각각의 페이지에 대한 두 개의 타임스탬프들(판독 타임스탬프 및 기록 타임스탬프) 및 각각의 TLB에 대해서 한 개의 타임스탬프(플러싱 타임스탬프)를 보유한다. 페이지로의 가장 마지막의 판독/판독-기록 매핑이 삭제될 때마다, 현재 시간이 판독/판독-기록 타임스탬프에 기록된다. TLB가 플러싱될 때마다, 현재 시간이 TLB의 플러싱 타임스탬프에 기록된다.

<78> 시간은 여러 가지 방법으로 측정될 수 있다. 제1 예시적인 방법으로, TLB를 위해서 사용되는 클록은 바로 플러싱 타임스탬프이다. 따라서, 페이지에 대한 판독 및 판독-기록 타임스탬프는 TLB 플러싱 타임스탬프로부터 복사된다. TLB가 플러싱될 때, 플러싱 타임스탬프가 변한다(방법은 중요하지 않음). 페이지 상의 판독/판독-기록 타임스탬프가 플러싱 타임스탬프와 다르다면, 판독/판독-기록 매핑이 TLB에 있지 않다는 것이 확인되는 것이다. 일치한다면, 플러싱 타임스탬프는 (아마도 여러번) 바뀌었지만 우연히 판독/판독-기록 타임스탬프와 일치하게 되고(예컨대, 카운터 랩어라운드(wraparound) 때문에), 어떠한 피해도 생기지 않는다: 단지 불필요한 플러싱이 있을 뿐이다.

<79> 제2 예시적인 방법으로, 시간은 실시간 또는 가상 클록으로 측정된다. 판독/판독-기록 타임스탬프 및 플러싱 타임스탬프들을 기록하기 위해서 사용된 클록들이 일정한 클록 스큐범위에서 항상 일치한다고 확인된다면, 판독/판독-기록 타임스탬프가 적어도 클록 스큐만큼 플러싱 타임스탬프보다 앞선 경우, 판독/판독-기록 매핑이 TLB에 있지 않다고 확인한다. 전체적으로 클록 스큐를 피하는 방법은 글로벌 클록, 즉 일부 TLB가 플러싱될 때 증가되는 공유된 카운터를 사용하는 방법이다; 하지만, 그러한 클록은 과열점(hotspot)이 될 수 있다.

<80> 제1 방법은 동기화된 클록들을 요구하지 않는다는 이점을 가진다. 하지만, 고려해야할 많은 수의 TLB들이 있다면, 제1 방법은 각각의 TLB에 대한 개별적인 판독/판독-기록 타임스탬프를 요구하지만, 그에 반하여 제2 방법은 오직 단독 클록 판독을 요구한다. 게다가, 그것은 클록들 그 자체에 대해서 메모리 경합을 제거한다. 상기 두 가지 방법들은 혼합될 수도 있고, 다른 메커니즘들(예컨대, 백터 클록들)이 사용될 수도 있다.

<81> 본 발명은 LHS(350)가 더 이상 어드레스 변환 맵과 일치하지 않는 구 TLB 엔트리를 사용하는 페이지에 실제로 액세스할 가능성이 있을 때까지, TLB의 명확한 플러싱들이 지연되기 위해서, TLB의 레이지 플러싱을 허용하는 메커니즘을 제공한다. 본질적으로, 페이지로의 마지막 매핑이 제거될 때, 페이지는 "타임스탬프화"된다(상기 설명한 바와 같이, "타임스탬프"는 실제로 시간을 포함하는 것이 아니라 연속적인 카운터들의 값을 포함한다). 나중에 페이지가 액세스될 때, 페이지가 타임스탬프화된 이후에 TLB가 플러싱되었는 지에 따라서 TLB를 플러싱했는 지에 대한 판정이 행해진다.

<82> 도 5에 도시된 바와 같이, 다중-프로세서 컴퓨터 시스템에서, 각각의 프로세서들(120(1) 내지 120(n))은 자신의 TLB(402(1)) 내지 402(n))를 보유한다. 다른 말로, 프로세서들(120(1) 내지 120(n))은 동일한 기초적인 어드레스 변환 테이블들을 각각 액세스할 수 있지만, 프로세서들은 상기 테이블들로부터의 매핑들을 개별적으로 캐싱할 수 있다. 추가적으로, 일 실시예에서, 각각의 프로세서는 LHS(350) 또는 RHS(360)에서 동작할 수 있다. 본 발명에 따르면, 카운터(502(1) 내지 502(n))는 각각의 프로세서들(120(1) 내지 120(n))과 연관된다. 프로세서의 환경이 RHS(360)에서 LHS(350)으로 변할 때마다, 그 프로세서와 연관된 카운터는 증가한다.

<83> 페이지가 LHS(350)에 사용가능하지 않게 될 때마다(즉, 페이지가 관련된 액세스 정책 하에서 LHS(350)에 사용가능하지 않게 되고 LHS(350)에 사용가능한 모든 매핑으로부터 링크가 끊겼을 때), 페이지는 효과적으로 카운터 값들로 "타임스탬프화"된다. (카운터는 물리적 의미로 "시간"을 세는 것이 아니라, 발생하는 특정한 사건에 반응하여 카운터들이 증가하기 때문에 시간 보존의 형태로써 보여질 수 있다.) 이렇게, 페이지에 대한 저장된 카운터 값들을 그 페이지에 대한 액세스를 시도하는 프로세서에 대한 현재 카운터 값과 비교하는 것에 의해서, 페이지가 LHS(350)에 사용가능하지 않게 된 이후로 프로세서가 RHS(360)로 들어갔는지에 대해 판정될 수 있다. (상기 설명한 바와 같이, RHS(360)에 들어가는 것은 TLB가 신뢰성있게 플러싱되게 한다.) 페이지가 링크가 끊긴 이후로 프로세서가 RHS(360)에 들어가지 않는다면, 프로세서는 RHS(360)에 들어가고 TLB를 플러싱한다. 페이지로 액세스하기 위한 시도가 LHS(350)에서 발생했다면, 프로세서는 LHS(350)로 되돌아가고 다시 페이지로 액세스하기 위해서 시도한다. 페이지가 링크가 끊긴 이후로 프로세서가 RHS(360)에 들어간다면(즉, 프로세서가 현재 RHS(360)에 있다면, 또는 페이지가 링크가 끊긴 이후의 어떤 지점에서, RHS(360)에 들어갔다면), 페이지가 LHS(350)에 사용가능하게 되었던 마지막 시간 이후로 TLB가 신뢰성있게 플러싱되었다고 알려지며, 이렇게 TLB는 페이지에 어떠한 매핑도 포함할 수 없고, 따라서 TLB는 상기 시간에 플러싱될 필요가 없다.

<84> 도 6은 페이지가 LHS(350)에 사용가능하지 않게 된 이후에 TLB가 플러싱되는 과정을 도시하는 흐름도이다. 도 6에 도시된 처리의 초기에, 주어진 페이지(페이지 X)는 적용가능한 액세스 정책하에서 LHS(350)로부터 액세스되어 사용가능하다고 가정된다. 어떠한 지점에서, 페이지 X는 액세스 정책하에서 LHS(350)에 사용가능하지 않게 되며(단계(602)), LHS(350)에 사용가능한 어드레스 변환 맵은 페이지 X가 상기 맵들로부터 링크가 끊겼다는 것을 보장하기 위해서 필요한 만큼 조정된다. 페이지 X가 링크가 끊긴 시간에, 머신 상의 모든 프로세서들에 대한 카운터 값들이 기록되고, 이런 기록은 이후의 검색(retrieval)을 위해서 페이지 X와 연관된다(단계(604)). 기록은 바람직하게는 머신 상의 각각의 프로세서들을 위한 현재 카운터 값을 포함한다.

<85> 페이지 X가 LHS(350)에 사용가능한 맵들로부터 링크가 끊긴 후에, 페이지 X가 실제로 액세스되기 전에 어느 정도의 시간이 흐를 수 있다. 하지만, 어떤 지점에서, 페이지 X를 위한 액세스 요청이 특정 프로세서(프로세서 Y)로부터 발생될 것이다(단계(606)). 페이지와 연관된 기록된 카운터 값들이 그후에 검색되고, 프로세서 Y에 대한 저장된 값이 프로세서 Y의 현재 카운터 값과 비교된다. (기록은 다른 프로세서들을 위한 저장된 카운터 값들을 또한 포함할 것이지만, 이러한 다른 값들은 프로세서 Y가 자신의 TLB를 플러싱할 필요가 있는지에 대해 판정할 때는 무시된다.) 프로세서 Y의 현재 카운터 값이 저장된 카운터 값보다 더 크다면, 프로세서 Y가 RHS(360)에 들어가고 페이지 X가 LHS(350)에 사용가능하지 않게 된 이후로 TLB를 플러싱하는 경우이어야 하기 때문에, 다른 질문 없이 페이지로의 액세스가 허용된다(단계(614)). 한편, 프로세서 Y에 대한 저장된 카운터 값이 프로세서 Y의 현재 카운터 값과 동일하다면, 페이지 X가 링크가 끊긴 이후로 RHS(360)에 들어가지 않은 것이고, 그것은 프로세서 Y의 TLB가 여전히 페이지 X로의 매핑을 포함할 수 있다는 것을 의미한다.

<86> 프로세서 Y가 RHS(360)에서 동작하고 있다면(단계(610)), TLB는 액세스 정책의 위배 없이 페이지 X로의 매핑을 포함할 수 있고, 따라서 페이지 X로의 액세스가 허용된다(단계(614)). 하지만, 프로세서 Y가 LHS(350)에서 동작하고 있다면(프로세서 Y가 페이지 X가 링크가 끊긴 이후로 TLB를 플러싱하지 않았다고 단계(608)로부터 알려짐), 프로세서 Y의 TLB의 내용들, 즉 페이지 X로의 매핑을 LHS(350)에 노출시킬 것이다. 이렇게, 프로세서 Y는 RHS(360)로 들어가고 그 TLB를 플러싱한다(단계(612)). TLB가 플러싱된 이후에, 프로세서 Y는 LHS(350)로 되돌아가고 액세스 요청이 재실행된다(단계(614))(그것에 의해서 새롭게-비워진 TLB를 가지고, 요청된 어드레스의 재변환을 요구한다.)

<87> LHS(350)에 액세스될 수 없는 페이지를 생성하는 것은 TLB를 플러싱할 필요를 트리거하는 이벤트일 뿐 아니라, 본 발명의 메커니즘 역시 다른 이유로 발생하는 TLB 플러싱을 트리거하는 데 사용될 수 있다. 예를 들면, (상기 정의된 바와 같이) 세트 D1 또는 D2중 하나로부터 페이지를 제거하는 것 또는 페이지를 위한 판독 액세스(또는 기록 액세스)를 포기하는 것은, 이러한 이벤트가 어떠한 매핑들이 관련된 정책하에서 적법한지에 대해서 영향을 줄 수 있기 때문에, TLB 플러싱에 대한 필요성을 야기할 수도 있다. 그러한 경우에, 본 발명의 메커니즘은 상기 설명된 방법과 유사한 방법으로 사용된다. 예컨대, D1 또는 D2로부터 페이지가 제거될 때, 또는 판독(또는 기록)액세스가 포기된 때, 카운터 값이 기록될 수 있고, 프로세서에 의한 페이지로의 액세스하려는 나중의 시도가 프로세서의 카운터 값과 페이지에 대한 저장된 카운터 값들 사이의 비교를 트리거할 것이다.

<88> 본 발명의 메커니즘들은 플러싱이 RHS(360)로의 트립을 요구하는 경우- 또는 LHS(350) 및 RHS(360) 모두가 사용가능한 경우조차도-에 국한되지는 않음을 또한 주의해야 한다. 더욱 일반적으로는, 본 발명의 메커니즘들은 오래된 엔트리들이 맵의 상태를 더 이상 반영하지 않는 TLB에 존재할 가능성이 있는 경우에 -예컨대, 운영 체제에

의해서 수행된 보통의 프로세스 고립(process isolation)의 경우에- 사용될 수 있다.

<89> 상기 예들은 단지 설명의 목적으로 제공되었으며 어떤 방식으로든 본 발명의 범위를 제한하지 않도록 구성되어 있음에 주의해야 한다. 본 발명이 다양한 실시예에 대해서 설명되었으며 동시에, 본 명세서에서 사용된 단어들은 제한하는 의미의 단어가 아니라, 설명 및 예시의 단어로 이해되어야 한다. 또한, 본 발명은 특정 수단, 재료, 및 실시예와 관련해서 본 명세서에서 설명되었지만, 본 발명은 본 명세서에서 설명된 특정사항들에 국한되는 것으로 의도된 것이 아니라, 본 발명은 첨부된 청구항들의 범위내에 있는 것과 같은, 기능적으로 동등한 구조들, 방법들 및 사용들로 확장된다. 본 명세서에서의 교시를 통해 장점을 갖는, 본 분야의 당업자들은 다양한 변형들 및 변화들이 본 발명의 범위 및 정신에서 벗어나지 않고 행해질 수 있음을 이해할 것이다.

발명의 효과

<90> TLB를 플러싱하는 작업은 기존의 방법으로는 불필요하게 비용이 많이 드는데 비해, 본 발명은 각각의 TLB에 대한 카운터 및 관련된 타임스탬프를 이용한 방법 등으로 TLB에 대한 레이지 플러싱을 지원함으로써 비용을 줄이는 효과가 있다.

도면의 간단한 설명

<1> 도 1은 본 발명의 양상들을 구현할 수 있는 예시적인 컴퓨팅 환경을 도시한 블록도;

<2> 도 2는 예시적인 가상 어드레스 시스템을 도시한 블록도.

<3> 도 3은 하나의 머신 상에 공존하는 두 환경들, 및 일부가 은닉된 메모리를 도시한 블록도.

<4> 도 4는 어드레스 변환 과정에서 사용되는 변환 색인 버퍼(TLB)를 도시하는 블록도.

<5> 도 5는 각각의 프로세서가 TLB 및 카운터와 관련되는, 다수의 프로세서들을 포함하는 시스템의 블록도.

<6> 도 6은 본 발명의 양상에 따른 TLB의 레이지 플러싱의 과정을 도시하는 흐름도.

<7> 〈도면의 주요 부분에 대한 부호의 설명〉

<8> 132: RAM

<9> 201: 저장 위치

<10> 202: 페이지 디렉토리

<11> 204: 페이지 테이블

<12> 206: 페이지

<13> 210: 가상 어드레스

<14> 211: 페이지 디렉토리 오프셋

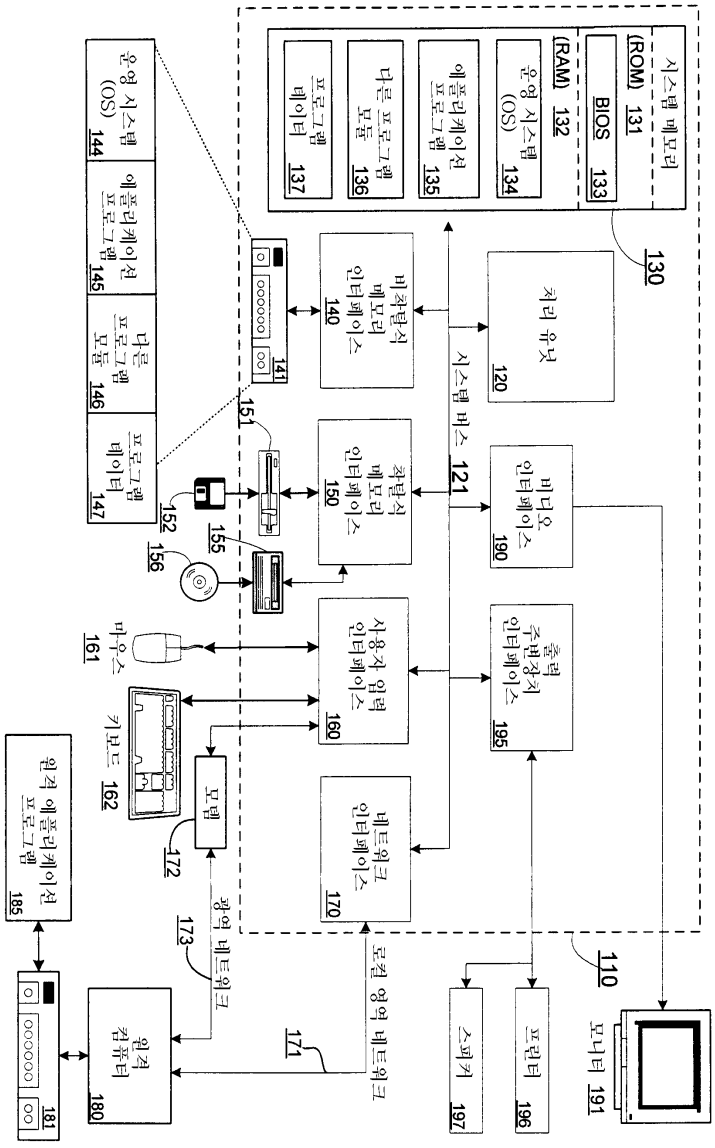
<15> 212: 페이지 테이블 오프셋

<16> 213: 페이지 오프셋

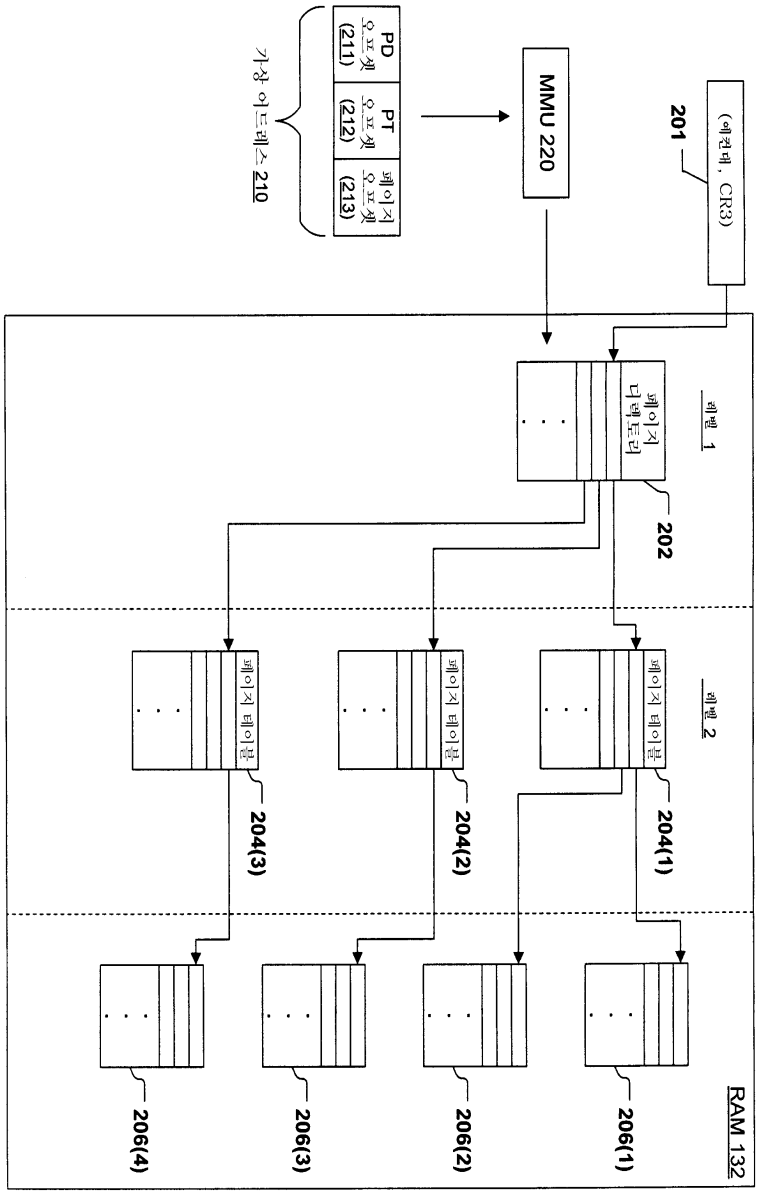
<17> 220: 메모리 관리 유닛

도면

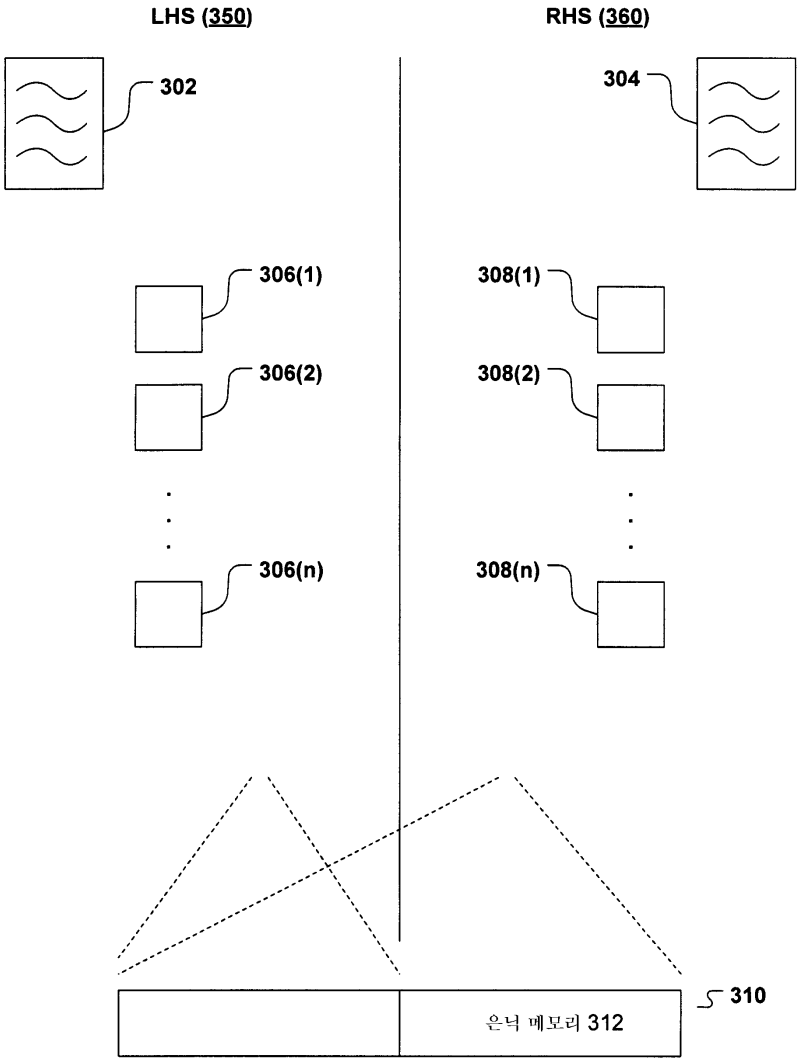
도면1


컴퓨팅 환경 100
시스템 메모리
(ROM) 131
BIOS 133
(RAM) 132
운영 시스템 (OS) 134
에플리케이션 프로그램 135
다른 프로그램 모듈 136
프로그램 데이터 137
130
처리 유닛 120
시스템 버스 121
비휘발성 메모리 인터페이스 140
휘발성 메모리 인터페이스 150
비디오 인터페이스 190
출력 주변장치 인터페이스 195
사용자 입력 인터페이스 160
네트워크 인터페이스 170
모뎀 172
광역 네트워크 173
로컬 영역 네트워크 171
모니터 191
프린터 196
스피커 197
110
키보드 162
마우스 161
151
152
155
156
141
운영 시스템 (OS) 144
에플리케이션 프로그램 145
다른 프로그램 모듈 146
프로그램 데이터 147
원격 에플리케이션 프로그램 185
원격 컴퓨터 180
181

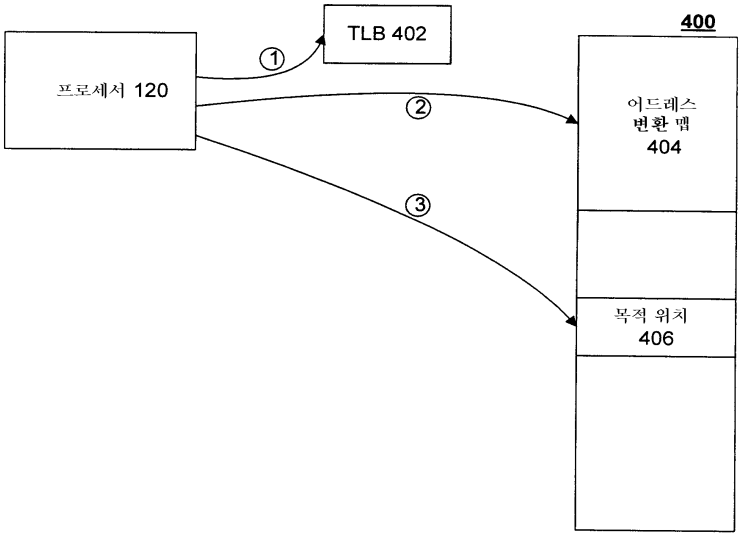
도면2


(예컨대, CR3)
201
MMU 220
PD 오프셋 (211)
PT 오프셋 (212)
페이지 오프셋 (213)
가상 어드레스 210
레벨 1
페이지 디렉토리
202
레벨 2
페이지 테이블
204(1)
204(2)
204(3)
RAM 132
206(1)
206(2)
206(3)
206(4)

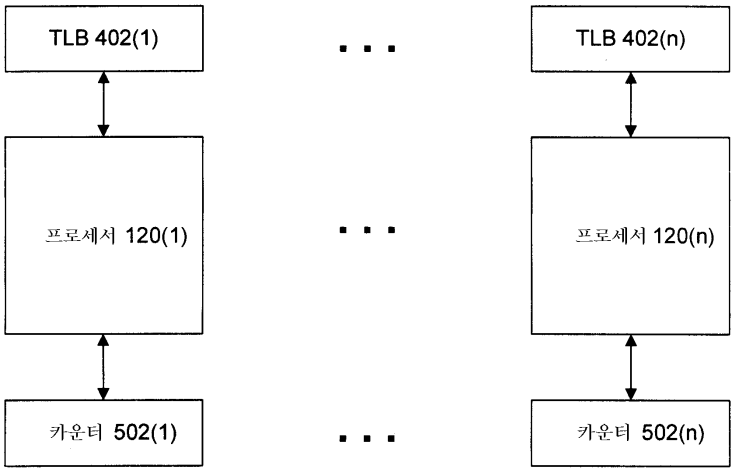
도면3


LHS (350)
RHS (360)
302
304
306(1)
306(2)
306(n)
308(1)
308(2)
308(n)
310
은닉 메모리 312

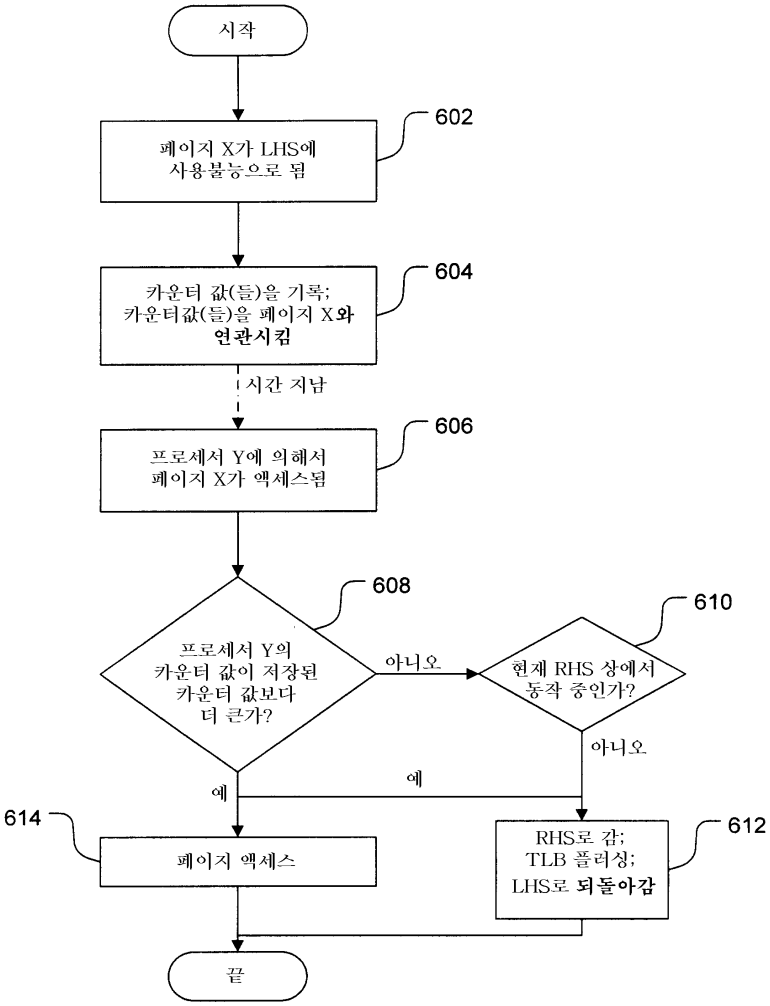
도면4


400
TLB 402
프로세서 120
①
②
③
어드레스
변환 맵
404
목적 위치
406

도면5


TLB 402(1)
TLB 402(n)
프로세서 120(1)
프로세서 120(n)
카운터 502(1)
카운터 502(n)

도면6


시작
페이지 X가 LHS에 사용불능으로 됨
602
카운터 값(들)을 기록; 카운터값(들)을 페이지 X와 연관시킴
604
시간 지남
프로세서 Y에 의해서 페이지 X가 액세스됨
606
프로세서 Y의 카운터 값이 저장된 카운터 값보다 더 큰가?
608
아니오
현재 RHS 상에서 동작 중인가?
610
아니오
예
예
페이지 액세스
614
RHS로 감; TLB 플러싱; LHS로 되돌아감
612
끝