(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2015-0016278
(43) 공개일자 2015년02월11일

(51) 국제특허분류(Int. Cl.)
G06F 12/10 (2006.01) *G06F 12/08* (2006.01)
(21) 출원번호 10-2014-7033343
(22) 출원일자(국제) 2013년05월08일
심사청구일자 없음
(85) 번역문제출일자 2014년11월27일
(86) 국제출원번호 PCT/GB2013/051186
(87) 국제공개번호 WO 2013/167886
국제공개일자 2013년11월14일
(30) 우선권주장
13/468,548 2012년05월10일 미국(US)

(71) 출원인
에이알엠 리미티드
영국 캠브리지 씨비1 9엔제이 체리힌톤 풀번로드 110
(72) 발명자
뵈처 마티아스
영국 캠브리지셔 씨비1 9엔제이 체리힌톤 풀번로드 110 에이알엠 리미티드
커셔 다니엘
오스트리아 미크론웨그1 에이 8101 그라트코른 엔엑스피 세미컨덕터즈
(74) 대리인
이화익

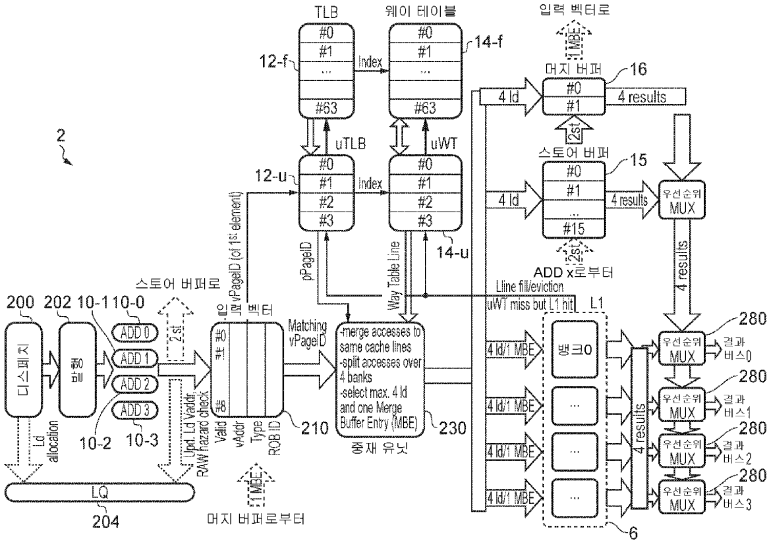
전체 청구항 수 : 총 34 항

(54) 발명의 명칭 **캐시 및 변환 색인 버퍼를 갖는 데이터 처리장치**

(57) 요 약

데이터 처리장치는 캐시와 TLB(translation look aside buffer)를 갖는다. 복수의 캐시 웨이 중 어느 것이 필요한 데이터를 저장하는지를 식별하기 위한 웨이 테이블이 제공된다. 각 웨이 테이블 엔트리는 TLB의 TLB 엔트리 중 하나에 대응하고, 대응하는 TLB 엔트리와 관련된 페이지의 각 메모리 위치에 대해서, 어느 캐시 웨이가 그 메모리 위치와 관련된 데이터를 저장하는지를 식별한다. 또한, 캐시는 같은 처리 사이클에서 M개의 액세스 요구들을 서비스하는 것이 가능하다. 아비터는 선택된 펜딩 액세스 요구들이 최대 N개의 상이한 가상 페이지 어드레스들을 지정한다는 것을 보장하는 방식으로 캐시에 의해 서비스하기 위한 펜딩 액세스 요구들을 선택할 수 있고, 여기서 N < M이다.

대 표 도



특허청구의 범위

청구항 1

데이터 처리장치로서,

데이터를 처리하도록 구성된 처리회로와,

데이터를 저장하기 위한 복수의 캐시 웨이들(cache ways)를 구비하는 캐시와,

메모리 위치의 관련된 페이지에 대한 가상-물리 어드레스 맵핑을 각각이 식별하는 복수의 TLB(translation lookaside buffer) 엔트리를 저장하도록 구성된 변환 색인 버퍼(TLB)와,

상기 복수의 TLB 엔트리 중의 하나에 각각이 대응하는 복수의 웨이 테이블 엔트리를 저장하도록 구성된 웨이 테이블을 구비하고,

각 웨이 테이블 엔트리는, 대응하는 TLB 엔트리와 관련된 페이지의 각 메모리 위치에 대해서, 상기 캐시의 상기 캐시 웨이들 중의 어느 것이, 만약 있다면, 메모리 위치와 관련된 데이터를 저장하는지를 식별하기 위한 웨이 정보를 포함하고,

타겟 가상 어드레스에 대응하는 타겟 데이터에 대한 상기 처리회로로부터의 데이터 액세스 요구에 응답해서, 상기 TLB는 상기 타겟 가상 어드레스에 대응하는 타겟 TLB 엔트리에 액세스하도록 구성되고, 상기 웨이 테이블은 상기 타겟 TLB 엔트리에 대응하는 웨이 테이블 엔트리에 액세스하도록 구성되며, 상기 캐시는 타겟 데이터와 관련된 메모리 위치에 대한 상기 대응하는 웨이 테이블 엔트리에 포함된 웨이 정보에 의존해서 캐시 액세스를 행하도록 구성되는 것을 특징으로 하는, 데이터 처리장치.

청구항 2

제 1 항에 있어서,

상기 캐시의 각 캐시 위치는 캐시 위치에 저장된 데이터와 관련된 메모리 어드레스의 일부를 식별하기 위한 대응하는 태그 데이터를 갖는 것을 특징으로 하는, 데이터 처리장치.

청구항 3

제 2 항에 있어서,

타겟 데이터와 관련된 메모리 위치에 대한 상기 대응하는 웨이 테이블 엔트리에 포함된 상기 웨이 정보가, 어느 캐시 웨이가 타겟 데이터를 저장하는지를 나타내면, 상기 캐시는, 상기 태그 데이터에 액세스하는 일없이 웨이 정보에 나타낸 캐시 웨이로부터의 타겟 데이터에 액세스함으로써 상기 캐시 액세스를 행하도록 구성되는 것을 특징으로 하는, 데이터 처리장치.

청구항 4

제 3 항에 있어서,

상기 웨이 정보가 어느 캐시 웨이가 타겟 데이터를 저장하는지를 나타내면, 상기 캐시는, 또한 웨이 정보에 나타낸 상기 캐시 웨이 이외의 캐시 웨이들에 액세스하는 일없이 상기 캐시 액세스를 행하도록 구성되는 것을 특징으로 하는, 데이터 처리장치.

청구항 5

제 2 항 내지 제 4 항 중 어느 한 항에 있어서,

타겟 데이터와 관련된 메모리 위치에 대한 상기 대응하는 웨이 테이블 엔트리에 포함된 상기 웨이 정보가 어느 캐시 웨이가 타겟 데이터를 저장하는지를 나타내지 않으면, 상기 캐시는, 상기 캐시 웨이들의 각각으로부터 데이터에 병렬로 액세스해서, 상기 액세스된 데이터의 어느 것이 상기 타겟 데이터에 대응하는지 여부를 상기 태그 데이터로부터 판정함으로써 상기 캐시 액세스를 행하도록 구성되는 것을 특징으로 하는, 데이터 처리장치.

청구항 6

제 1 항 내지 제 5 항 중 어느 한 항에 있어서,

상기 캐시에 할당되어 있는 메모리 위치와 관련된 데이터에 응답해서, 상기 캐시는, 상기 웨이 테이블을 제어해 상기 메모리 위치에 대응하는 웨이 정보를 갱신하여 어느 캐시 웨이가 상기 데이터를 저장하는지를 나타내도록 구성되는 것을 특징으로 하는, 데이터 처리장치.

청구항 7

제 1 항 내지 제 6 항 중 어느 한 항에 있어서,

각 웨이 테이블 엔트리는, 대응하는 TLB 엔트리와 관련된 페이지의 각 메모리 위치에 대해서, 메모리 위치와 관련된 데이터가 상기 캐시에 저장되는지 여부를 식별하기 위한 유효 정보를 포함하는 것을 특징으로 하는, 데이터 처리장치.

청구항 8

제 7 항에 있어서,

상기 캐시에 할당되어 있는 메모리 위치와 관련된 데이터에 응답해서, 상기 캐시는, 상기 웨이 테이블을 제어해 상기 메모리 위치에 대응하는 상기 유효 정보를 갱신하여 상기 데이터가 상기 캐시에 저장된다는 것을 나타내도록 구성되고,

상기 캐시로부터 축출되는 메모리 위치와 관련된 데이터에 응답해서, 상기 캐시는, 상기 웨이 테이블을 제어해 상기 메모리 위치에 대응하는 상기 유효 정보를 갱신하여 상기 캐시에 데이터가 저장되지 않는다는 것을 나타내도록 구성되는 것을 특징으로 하는, 데이터 처리장치.

청구항 9

제 2 항 내지 제 8 항 중 어느 한 항에 있어서,

상기 태그 데이터는 대응하는 캐시 위치에 저장된 데이터와 관련된 물리 어드레스의 일부를 식별하는 것을 특징으로 하는, 데이터 처리장치.

청구항 10

제 9 항에 있어서,

상기 TLB는 내용 어드레스지정 메모리(content addressable memory)를 구비하고, 상기 TLB 엔트리는 물리 어드레스 및 가상 어드레스 양자에 의해서 검색 가능한 것을 특징으로 하는, 데이터 처리장치.

청구항 11

제 5 항에 있어서,

상기 웨이 정보가 어느 캐시 웨이가 상기 타겟 데이터를 저장하는지를 나타내지 않고, 상기 캐시가 상기 캐시 웨이들 중의 하나로부터 액세스된 데이터가 상기 타겟 데이터에 대응한다는 것을 상기 태그 데이터로부터 판정하면, 상기 캐시는, 상기 웨이 정보를 제어해 상기 웨이 정보를 갱신하여 상기 타겟 데이터를 저장하는 캐시 웨이를 식별하도록 구성되는 것을 특징으로 하는, 데이터 처리장치.

청구항 12

제 11 항에 있어서,

가장 최근에 액세스된 웨이 테이블 엔트리를 식별하는 웨이 테이블 엔트리 식별자를 저장하도록 구성된 저장소자를 구비하고,

상기 웨이 정보가 어느 캐시 웨이가 상기 타겟 데이터를 저장하는지를 나타내지 않고, 상기 캐시 웨이들 중의 하나로부터 액세스된 데이터가 상기 타겟 데이터에 대응한다는 것을 상기 캐시가 상기 태그 데이터로부터 판정하면, 상기 캐시는, 상기 웨이 테이블을 제어해서 상기 웨이 테이블 엔트리 식별자에 의해 식별된 웨이 테이블 엔트리의 웨이 정보를 갱신하도록 구성되는 것을 특징으로 하는, 데이터 처리장치.

청구항 13

제 1 항 내지 제 12 항 중 어느 한 항에 있어서,

상기 TLB는 제1 레벨 TLB이고, 상기 장치는 적어도 한 개의 추가 레벨 TLB를 구비하며,

상기 타겟 가상 어드레스에 대응하는 상기 타겟 데이터에 대한 상기 데이터 액세스 요구에 응답해서, 상기 제1 레벨 TLB는 상기 타겟 가상 어드레스에 대응하는 타겟 TLB 엔트리를 검색하도록 구성되고,

상기 제1 레벨 TLB가 상기 타겟 TLB 엔트리를 저장하지 않으면, 상기 제1 레벨 TLB는 상기 적어도 한 개의 추가 레벨 TLB로부터 상기 타겟 TLB 엔트리에 액세스하도록 구성되는 것을 특징으로 하는, 데이터 처리장치.

청구항 14

제 13 항에 있어서,

상기 제1 레벨 TLB의 각 TLB 엔트리는 상기 웨이 테이블 내에 대응하는 웨이 테이블 엔트리를 갖는 것을 특징으로 하는, 데이터 처리장치.

청구항 15

제 14 항에 있어서,

상기 웨이 테이블은 제1 레벨 웨이 테이블을 구비하고, 상기 장치는 적어도 한 개의 추가 레벨 웨이 테이블을 구비하며,

상기 적어도 한 개의 추가 레벨 TLB의 각 TLB 엔트리는 상기 적어도 한 개의 추가 레벨 웨이 테이블 내에 대응하는 웨이 테이블 엔트리를 갖고,

상기 제1 레벨 웨이 테이블이 상기 타겟 TLB 엔트리에 대응하는 웨이 테이블 엔트리를 저장하지 않으면, 상기 제1 레벨 웨이 테이블은, 상기 적어도 한 개의 추가 레벨 웨이 테이블로부터 상기 타겟 TLB 엔트리에 대응하는 웨이 테이블 엔트리에 액세스하도록 구성되는 것을 특징으로 하는, 데이터 처리장치.

청구항 16

제 1 항 내지 제 15 항 중 어느 한 항에 있어서,

상기 TLB 및 상기 웨이 테이블은 상기 데이터 처리장치에 의해 별도로 저장되는 것을 특징으로 하는, 데이터 처리장치.

청구항 17

제 1 항 내지 제 15 항 중 어느 한 항에 있어서,

상기 TLB 및 상기 웨이 테이블은 복수의 엔트리를 저장하도록 구성된 공통 메모리를 구비하고, 상기 공통 메모리의 각 엔트리는 상기 TLB의 TLB 엔트리와 상기 웨이 테이블의 대응하는 웨이 테이블 엔트리를 포함하는 것을 특징으로 하는, 데이터 처리장치.

청구항 18

데이터 처리장치로서,

데이터를 처리하는 처리 수단과,

데이터를 저장하고, 복수의 캐시 웨이들을 구비하는 캐시 수단과,

메모리 위치들의 관련 페이지에 대한 가상-물리 어드레스 맵핑을 각각이 식별하는 복수의 TLB(translation lookaside buffer) 엔트리를 저장하는 변환 색인 버퍼(TLB) 수단과,

상기 복수의 TLB 엔트리 중의 하나에 각각이 대응하는 복수의 웨이 테이블 엔트리를 저장하는 웨이 테이블 수단을 구비하고,

각 웨이 테이블 엔트리는, 대응하는 TLB 엔트리와 관련된 페이지의 각 메모리 위치에 대해서, 상기 캐시 수단의 상기 캐시 웨이들 중의 어느 것이, 만약 있다면, 메모리 위치와 관련된 데이터를 저장하는지를 식별하기 위한 웨이 정보를 포함하고,

타겟 가상 어드레스에 대응하는 타겟 데이터에 대한 상기 처리 수단으로부터의 데이터 액세스 요구에 응답해서, 상기 TLB 수단은, 상기 타겟 가상 어드레스에 대응하는 타겟 TLB 엔트리에 액세스하도록 구성되고, 상기 웨이 테이블 수단은, 상기 타겟 TLB 엔트리에 대응하는 웨이 테이블 엔트리에 액세스하도록 구성되며, 상기 캐시 수단은 타겟 데이터와 관련된 메모리 위치에 대한 상기 대응하는 웨이 테이블 엔트리에 포함된 웨이 정보에 의존해서 캐시 액세스를 행하도록 구성되는 것을 특징으로 하는, 데이터 처리장치.

청구항 19

데이터를 저장하기 위한 복수의 캐시 웨이들(cache ways)를 구비하는 캐시와, 메모리 위치들의 관련 페이지에 대한 가상-물리 어드레스 맵핑을 각각이 식별하는 복수의 TLB(translation lookaside buffer) 엔트리를 저장하도록 구성된 변환 색인 버퍼(TLB)와, 상기 복수의 TLB 엔트리 중의 하나에 각각이 대응하는 복수의 웨이 테이블 엔트리를 저장하도록 구성된 웨이 테이블을 구비하고, 각 웨이 테이블 엔트리가, 대응하는 TLB 엔트리와 관련된 페이지의 각 메모리 위치에 대해서, 상기 캐시의 상기 캐시 웨이들 중의 어느 것이, 만약 있다면, 메모리 위치와 관련된 데이터를 저장하는지를 식별하기 위한 웨이 정보를 포함하는, 데이터 처리장치의 방법으로서,

상기 방법은,

타겟 가상 어드레스에 대응하는 타겟 데이터에 대한 데이터 액세스 요구에 응답해서, 상기 타겟 가상 어드레스에 대응하는 타겟 TLB 엔트리에 액세스하는 상기 TLB, 및 상기 타겟 TLB 엔트리에 대응하는 웨이 테이블 엔트리에 액세스하는 상기 웨이 테이블과,

타겟 데이터와 관련된 메모리 위치에 대한 상기 대응하는 웨이 테이블 엔트리에 포함된 웨이 정보에 의존해서 캐시 액세스를 행하는 상기 캐시를 구비하는 것을 특징으로 하는, 데이터 처리장치의 방법.

청구항 20

데이터 처리장치로서,

데이터에 대한 액세스 요구들을 발행하도록 구성된 처리 회로와,

상기 액세스 요구들에 응답해서 데이터에의 액세스를 제공하도록 구성된 캐시와,

상기 액세스 요구들에서 지정된 가상 페이지 어드레스들과 상기 캐시가 사용하는 물리 페이지 어드레스들 사이에서 변환하도록 구성된 변환 색인 버퍼(TLB:translation lookaside buffer)와,

상기 처리 회로에 의해 발행된 펜딩(pending) 액세스 요구들을 버퍼링하도록 구성된 입력 버퍼와,

상기 입력 버퍼로부터의 상기 펜딩 액세스 요구들 중의 어느 것이 각 처리 사이클에서 상기 캐시에 의해 서비스되어야 하는지를 선택하도록 구성된 아비터(arbiter)를 구비하고,

상기 캐시는 같은 처리 사이클에서 M개의 액세스 요구들을 서비스하는 것이 가능하고, 여기서 M은 정수이며,

상기 아비터는 선택된 펜딩 액세스 요구들이 최대 N개의 상이한 가상 페이지 어드레스들을 지정하는 것을 보장하기 위해서 상기 펜딩 액세스 요구들을 선택하도록 구성되고, 여기서 N은 정수이며, N < M인 것을 특징으로 하는, 데이터 처리장치.

청구항 21

제 20 항에 있어서,

상기 TLB는 같은 처리 사이클에서 N개의 상이한 가상 페이지 어드레스들을 변환할 수 있는 것을 특징으로 하는, 데이터 처리장치.

청구항 22

제 20항 또는 제 21 항에 있어서,

상기 아비터는 상기 N개의 상이한 가상 페이지 어드레스들 중의 하나를 지정하는 펜딩 액세스 요구들을 후보 액세스 요구들로서 선택하고, 상기 후보 액세스 요구들 중의 적어도 하나를, 상기 캐시에 의해 서비스하기 위해서 선택하도록 구성되는 것을 특징으로 하는, 데이터 처리장치.

청구항 23

제 20 항 내지 제 22 항 중 어느 한 항에 있어서,

N=2인 것을 특징으로 하는, 데이터 처리장치.

청구항 24

제 20 항 내지 제 22 항 중 어느 한 항에 있어서,

N=1인 것을 특징으로 하는, 데이터 처리장치.

청구항 25

제 24 항에 있어서,

각 처리 사이클에 있어서,

(i) 상기 TLB는 주요한 액세스 요구가 지정한 가상 페이지 어드레스를 물리 페이지 어드레스로 변환하도록 구성

되고, 상기 주요한 액세스 요구가 상기 입력 버퍼로부터의 상기 펜딩 액세스 요구들 중의 하나를 포함하며,

(ii) 상기 아비터는 상기 주요한 액세스 요구와 같은 가상 페이지 어드레스를 지정하는 상기 입력 버퍼로부터의 펜딩 액세스 요구들 및 상기 주요한 액세스 요구를 후보 액세스 요구들로서 선택하도록 구성되며,

(iii) 상기 아비터는 상기 TLB에 의해 변환된 물리 페이지 어드레스를 이용해서 상기 캐시에 의해 서비스될 상기 후보 액세스 요구들 중의 적어도 하나를 선택하도록 구성되는 것을 특징으로 하는, 데이터 처리장치.

청구항 26

제 25 항에 있어서,

상기 주요한 액세스 요구는,

상기 펜딩 액세스 요구들 중의 가장 오래된 펜딩 액세스 요구와,

가장 높은 우선순위를 갖는 상기 펜딩 액세스 요구들 중의 하나의 펜딩 액세스 요구와,

가장 높은 우선순위의 액세스 타입을 갖는 상기 펜딩 액세스 요구들 중의 하나의 펜딩 액세스 요구 중 하나인 것을 특징으로 하는, 데이터 처리장치.

청구항 27

제 22 항에 있어서,

상기 캐시는 다수의 캐리 라인을 갖고, 다수의 후보 액세스 요구들이 동일한 캐시 라인에 저장된 데이터를 겨냥하면, 상기 아비터는 상기 다수의 후보 액세스 요구들을 병합해서 한 개의 병합된 액세스 요구를 형성하도록 구성되는 것을 특징으로 하는, 데이터 처리장치.

청구항 28

제 27 항에 있어서,

상기 입력 버퍼는 X개의 펜딩 액세스 요구들을 버퍼링하도록 구성되고, 상기 아비터는 최대 Y개의 후보 액세스 요구들을 한 개의 병합된 액세스 요구로 병합하도록 구성되고, 여기서 X 및 Y는 정수이며, $Y \leq X$인 것을 특징으로 하는, 데이터 처리장치.

청구항 29

제 20 항 내지 제 28 항 중 어느 한 항에 있어서,

상기 캐시는 데이터를 저장하기 위한 캐시 라인들의 M개의 뱅크를 구비하고, 각 뱅크는 독립적으로 액세스가능하며, 상기 캐시는 같은 처리 사이클에서 상이한 뱅크들에 지시된 M개의 액세스 요구들까지 서비스할 수 있는 것을 특징으로 하는, 데이터 처리장치.

청구항 30

제 29 항에 있어서,

뱅크 내의 캐시 라인을 겨냥하며 상기 N개의 상이한 가상 페이지 어드레스들 중의 하나를 지정하는 적어도 한 개의 뱅크 후보 액세스 요구를 갖는 캐시 라인의 각 뱅크에 대해서, 상기 아비터는

(i) 상기 적어도 한 개의 뱅크 후보 액세스 요구의 제1 뱅크 후보 액세스 요구를 선택하고,

(ii) 상기 제1 뱅크 후보 액세스 요구와 같은 캐시 라인을 겨냥하는 1개 이상의 다른 뱅크 후보 어드레스 요구

들이 있는 경우, 제1 뱅크 후보 액세스 요구와 상기 1개 이상의 다른 뱅크 후보 액세스 요구들을 병합해서 병합된 액세스 요구를 형성하고, 뱅크에 의해 서비스하기 위한 상기 병합된 액세스 요구를 선택하며,

(iii) 상기 제1 뱅크 후보 액세스 요구와 같은 캐시 라인을 겨냥하는 다른 뱅크 후보 액세스 요구들이 없는 경우에는, 뱅크에 의해 서비스하기 위한 상기 제1 뱅크 후보 액세스 요구를 선택하도록 구성되는 것을 특징으로 하는, 데이터 처리장치.

청구항 31

제 30 항에 있어서,

상기 제1 뱅크 후보 액세스 요구는,

상기 적어도 한 개의 뱅크 후보 액세스 요구 중의 가장 오래된 뱅크 후보 액세스 요구와,

가장 높은 우선순위를 갖는 상기 적어도 한 개의 뱅크 후보 액세스 요구 중의 하나의 뱅크 후보 액세스 요구와,

가장 높은 우선순위의 액세스 타입을 갖는 상기 적어도 한 개의 뱅크 후보 액세스 요구 중의 하나의 뱅크 후보 액세스 요구 중 하나인 것을 특징으로 하는, 데이터 처리장치.

청구항 32

제 20 항 내지 제 31 항 중 어느 한 항에 있어서,

상기 입력 버퍼는 현재의 처리 사이클에서 상기 아비터에 의해 선택되지 않는 펜딩 액세스 요구들을 저장하도록 구성된 저장회로를 구비하고, 상기 저장회로에 저장된 상기 펜딩 액세스 요구들은 다음 처리 사이클에서 상기 아비터에 의한 선택에 이용가능한 것을 특징으로 하는, 데이터 처리장치.

청구항 33

데이터 처리장치로서,

데이터에 대한 액세스 요구들을 발행하는 처리 수단과,

상기 액세스 요구들에 응답해서 데이터에의 액세스를 제공하는 캐시 수단과,

상기 캐시 수단에 의해 사용된 물리 페이지 어드레스들과 상기 액세스 요구들에서 지정된 가상 페이지 어드레스들 사이에서 변환하는 변환 색인 버퍼(TLB:translation lookaside buffer) 수단과,

상기 처리 수단에 의해 발행된 펜딩 액세스 요구들을 버퍼링하는 입력 버퍼 수단과,

상기 입력 버퍼로부터의 상기 펜딩 액세스 요구들 중의 어느 것이 각 처리 사이클에서 상기 캐시 수단에 의해 서비스되어야 하는지를 선택하는 중재 수단을 구비하고,

상기 캐시 수단은 같은 처리 사이클에서 M개의 액세스 요구들을 서비스할 수 있고, 여기서 M은 정수이며,

상기 중재 수단은 선택된 펜딩 액세스 요구들이 최대 N개의 상이한 가상 페이지 어드레스를 지정한다는 것을 보장하기 위해서 상기 펜딩 액세스 요구들을 선택하도록 구성되고, 여기서 N은 정수이며 $N < M$인 것을 특징으로 하는, 데이터 처리장치.

청구항 34

데이터에 대한 액세스 요구들을 발행하도록 구성된 처리 회로와, 상기 액세스 요구들에 응답해서 데이터에의 액세스를 제공하도록 구성된 캐시와, 상기 캐시에 의해 사용된 물리 페이지 어드레스들과 상기 액세스 요구들에서 지정된 가상 페이지 어드레스들 사이에서 변환하도록 구성된 변환 색인 버퍼(TLB:translation lookaside buffer)를 구비하는 데이터 처리장치의 방법으로서,

상기 방법은,

입력 버퍼에서 상기 처리 회로에 의해 발행된 펜딩 액세스 요구들을 버퍼링하는 단계와,

상기 입력 버퍼로부터 상기 펜딩 액세스 요구들 중의 어느 것이 각 처리 사이클에서 상기 캐시에 의해 서비스되어야 하는지를 선택하는 단계를 포함하고,

상기 캐시는 같은 처리 사이클에서 M개의 액세스 요구들을 서비스할 수 있고, 여기서 M은 정수이며,

상기 선택 단계는 선택된 펜딩 액세스 요구들이 최대 N개의 상이한 가상 페이지 어드레스들을 지정한다는 것을 보장하기 위해서 상기 펜딩 액세스 요구들을 선택하며, 여기서 N은 정수이고, N < M인 것을 특징으로 하는, 데이터 처리장치의 방법.

명 세 서

기 술 분 야

[0001] 본 발명은 데이터 처리의 분야에 관한 것이다. 특히, 본 발명은 캐시 및 변환 색인 버퍼(translation lookaside buffer)를 갖는 데이터 처리장치에 관한 것이다.

배 경 기 술

[0002] 데이터 처리장치는 데이터를 저장하기 위한 캐시를 갖는다. 이 캐시는 메모리에 저장된 데이터의 일부에 좀더 빠른 액세스를 제공한다. 데이터 처리장치는 또한 캐시 및/또는 메모리가 사용하는 물리 어드레스들 및 프로세서에 의해 지정된 가상 어드레스 사이에서 변환하는 변환 색인 버퍼(TLB;translation lookaside buffer)도 구비한다. 이 캐시가 물리적으로 인덱스되거나 물리적으로 태그된 캐시이면, 캐시 액세스가 이루어질 수 있기 전에 TLB를 이용해서 어드레스 변환을 요구한다. 캐시가 가상적으로 인덱스되고 가상적으로 태그되는 경우에도, TLB 변환은 여전히 액세스 메모리에 요구될 수 있다.

[0003] 본 기술은 변환 색인 버퍼를 갖는 시스템 내에서 캐시 액세스를 행하는 에너지 효율을 향상시키는 것을 목적으로 한다.

발명의 내용

[0004] 일 국면에서 보면, 본 발명은,

[0005] 데이터를 처리하도록 구성된 처리회로와,

[0006] 데이터를 저장하기 위한 복수의 캐시 웨이들(cache ways)를 구비하는 캐시와,

[0007] 메모리 위치의 관련된 페이지에 대한 가상-물리 어드레스 맵핑을 각각이 식별하는 복수의 TLB(translation lookaside buffer) 엔트리를 저장하도록 구성된 TLB와,

[0008] 상기 복수의 TLB 엔트리 중의 하나에 각각이 대응하는 복수의 웨이 테이블 엔트리를 저장하도록 구성된 웨이 테이블을 구비하는, 데이터 처리장치를 제공하고,

[0009] 각 웨이 테이블 엔트리는, 대응하는 TLB 엔트리와 관련된 페이지의 각 메모리 위치에 대해서, 상기 캐시의 상기 캐시 웨이들 중의 어느 것이, 만약 있다면, 메모리 위치와 관련된 데이터를 저장하는지를 식별하기 위한 웨이 정보를 포함하고,

[0010] 타겟 가상 어드레스에 대응하는 타겟 데이터에 대한 상기 처리회로로부터의 데이터 액세스 요구에 응답해서, 상기 TLB는 상기 타겟 가상 어드레스에 대응하는 타겟 TLB 엔트리에 액세스하도록 구성되고, 상기 웨이 테이블은 상기 타겟 TLB 엔트리에 대응하는 웨이 테이블 엔트리에 액세스하도록 구성되며, 상기 캐시는 타겟 데이터와 관련된 메모리 위치에 대한 상기 대응하는 웨이 테이블 엔트리에 포함된 웨이 정보에 의존해서 캐시 액세스를 행하도록 구성된다.

[0011] 캐시는 데이터를 저장하기 위한 복수의 캐시 웨이를 가질 수 있다. 예를 들면, 세트 결합 캐시(set-associative

cache)는 캐시 웨이들 중의 어느 것인가에 해당 위치에서의 특정 메모리 위치와 관련된 데이터를 저장할 수 있다. 캐시 액세스가 이루어지면, 캐시는 그 웨이들 중의 어느 것이 필요한 데이터를 저장하고 있는지를 결정한다. 이것을 지원하기 위해서, 웨이 테이블은 웨이 정보를 포함하는 웨이 테이블 엔트리들을 저장하기 위해서 제공되고, 웨이 정보의 각각은 주어진 메모리 위치와 관련된 데이터를 어느 캐시 웨이가 저장하는지를 식별한다. 그러므로, 캐시는 어느 웨이가 필요한 데이터를 저장할 것인지를 캐시 검색(lookup)을 행하기 전에 알 수 있고, 그래서 웨이 정보에 의존해서 좀더 효율적인 캐시 액세스를 수행할 수 있다.

[0012] 웨이 테이블은 가상-물리 어드레스 맵핑을 저장하기 위한 변환 색인 버퍼(TLB;translation lookaside buffer)에 결합된다. 웨이 테이블의 각 웨이 테이블 엔트리는 TLB의 TLB 엔트리에 대응할 수 있다. 각 웨이 테이블 엔트리에 포함된 웨이 정보는, 대응하는 TLB 엔트리와 관련된 페이지의 각 메모리 위치에 대해서, 어느 캐시 웨이가, 만약 있다면, 그 메모리 위치와 관련된 데이터를 저장하는지를 식별할 수 있다. 타겟 가상 어드레스에 근거한 싱글 테이블 검색은 양 타겟 TLB 엔트리와 대응하는 웨이 테이블 엔트리를 식별할 수 있기 때문에, 웨이 정보를 메모리의 페이지에 대응하는 엔트리로 그룹화하고, 대응하는 TLB 엔트리와 각 웨이 테이블 엔트리를 결합함으로써, 웨이 테이블을 검색하는 것이 더 효율적이게 된다. 그러므로, 필요한 엔트리들의 위치를 지정하기 위해서 양 TLB와 웨이 테이블에서 별도의 어드레스 검색이 필요 없다. 특정 값에 대한 테이블을 검색하는 것은 에너지를 소비하기 때문에, 본 기술은 정확한 웨이 테이블 엔트리를 식별하기 위해서 이미 TLB에 필요한 검색을 재이용해 좀더 에너지 효율적인 방식으로 TLB 및 웨이 테이블이 구현되는 것을 가능하게 한다.

[0013] 또한, 몇몇의 연속 데이터 액세스 요구들이 메모리의 같은 페이지와 관련된 데이터를 겨냥하는 것이 일반적이다. 웨이 정보를 메모리의 대응하는 페이지와 각각 관련된 웨이 테이블 엔트리로 그룹화함으로써, 그러한 데이터 액세스 요구들이 다수의 웨이 테이블 액세스를 행할 필요없이 싱글 웨이 테이블 엔트리를 이용해서 처리될 수 있다. 이것은 또한 웨이 정보가 액세스되는 효율을 향상시킨다.

[0014] 캐시의 각 캐시 위치가 캐시 위치에 저장된 데이터와 관련된 메모리 어드레스의 일부를 식별하기 위한 대응하는 태그 데이터를 갖는다. 태그 데이터는 캐시 웨이들 중의 어는 것이 원하는 데이터 값을 저장하는지를 결정하기 위해서 사용될 수 있다.

[0015] 데이터 액세스 요구에 응답해서, 어느 캐시 웨이가 타겟 데이터를 저장하는지를, 요청된 어드레스에 대한 관련 웨이 테이블 엔트리에 포함된 웨이 정보가 나타내면, 캐시는 태그 데이터에 액세스하는 일없이, 웨이 정보에 나타낸 캐시 웨이로부터 타겟 데이터에 액세스할 수 있다. 태그 데이터를 검색할 필요없이, 에너지 소모를 줄일 수 있다.

[0016] 또한, 어느 캐시 웨이가 타겟 데이터를 저장하는지를 웨어 정보가 나타내면, 웨이 정보에 나타나 있지 않은 다른 캐시 웨이들에 액세스할 필요가 없다. 그러므로, 타겟 데이터를 포함하지 않는 그러한 캐시 웨이들에 액세스하지 않음으로써 에너지 소모를 더 줄일 수 있다.

[0017] 어느 웨이가 타겟 정보를 저장하는지를 웨이 정보가 식별하지 못할 수 있다는 것이 가능하다. 예를 들면, 웨이 테이블이 단지 한정된 수의 테이블 엔트리를 갖고, 그래서 신규 엔트리가 요구될 때 웨이 테이블이 이미 가득 차있으면, 비록 캐시가 여전히 축출된 엔트리와 관련된 페이지 내의 어드레스에 대응하는 데이터를 포함하고 있더라도, 이전의 엔트리가 축출되어야 한다. 만약 데이터 액세스 요구가 축출된 웨이 테이블 엔트리에 대응하는 페이지 내의 메모리 위치에 겨냥해서 발행되고, 웨이 테이블 엔트리가 웨이 테이블로 다시 들어오게 되면, 어느 캐시 웨이가 타겟 데이터를 저장하는지를 웨이 테이블이 식별하지 못할 수 있다.

[0018] 따라서, 어느 캐시가 타겟 데이터를 저장하는지를 타겟 데이터에 대한 웨이 정보가 나타내지 않는 경우에는, 캐시가 병렬로 각 캐시 웨이로부터의 데이터에 액세스할 수 있고, 액세스된 데이터의 어느 것이 타겟 데이터에 대응하는지 여부를 태그 데이터로부터 판정할 수 있다. 그러므로, 어느 캐시 웨이가 데이터를 저장하는지를 식별하기 위해 웨이 테이블이 제공되더라도, 태그 데이터는 여전히 어느 웨이가 필요한 데이터를 저장하는지를 웨이 정보가 나타내지 않는 상황을 처리하는 데에 유용한다.

[0019] 웨이 테이블 내의 웨이 정보가 캐시의 상태를 정확하게 반영하는 것을 보장하기 위해서, 캐시된 데이터의 할당의 변화에 응답해서 웨이 정보가 갱신될 수 있다. 캐시에 할당되어 있는 데이터에 응답해서, 캐시가 웨이 테이블을 제어해 관련된 메모리 위치에 대응하는 웨이 정보를 갱신하여 어느 캐시 웨이가 데이터를 저장하는지를 나타낼 수 있다. 그래서, 그 데이터에 대한 다음 캐시 액세스시에, 캐시는 어느 웨이가 데이터를 저장하는지를 웨이 정보에 근거해서 판정할 수 있다.

[0020] 어느 캐시 웨이가 대응하는 데이터를 저장하는지를 식별하는 웨이 정보뿐 아니라, 각 웨이 테이블 엔트리는 대

응하는 TLB 엔트리와 관련된 페이지의 각 메모리 위치에 대해서, 그 메모리 위치와 관련된 데이터가 캐시에 저장되어 있는지 여부를 식별하기 위한 유효 정보도 포함할 수 있다. 이 유효 정보는 캐시 액세스에서 수행할 때 대응하는 웨이 정보를 고려할지 여부를 결정하기 위해서 캐시에 의해 사용될 수 있다.

[0021] 메모리 위치와 관련된 데이터가 캐시에 할당될 때, 캐시가 웨이 테이블을 제어해 그 메모리 위치에 대응하는 유효 정보를 갱신하여 데이터가 캐시에 저장된다는 것을 나타낼 수 있다. 한편, 메모리 위치와 관련된 데이터가 캐시로부터 축출될 때, 캐시가 웨이 테이블을 제어해 그 메모리 위치에 대응하는 유효 정보를 갱신하여 데이터가 캐시에 저장되어 있지 않다는 것을 나타낼 수 있다. 갱신된 유효 정보가 현재 데이터가 캐시에 저장되어 있지 않는다는 것을 나타내고, 그래서 그 메모리 위치에 대한 웨이 정보가 무시되어야 하기 때문에, 캐시 축출시 대응하는 웨이 정보를 갱신할 필요가 없다.

[0022] 웨이 정보 및 유효 정보는 웨이 테이블 엔트리에 대응하는 페이지의 각 메모리 위치에 대하여 상이한 방식으로 표현될 수 있다. 몇몇 실시예에 있어서는, 웨이 정보 및 유효 정보는 대응하는 페이지 내의 각 메모리 위치에 대해서 개별적으로 표시될 수 있다. 다른 실시예에 있어서는, 결합된 유효/웨이 정보 값은 각 메모리 위치에 대해서 제공될 수 있다. 다른 실시예에 있어서는, 웨이 정보가 한 그룹의 메모리 위치에 대해서 공동으로 설정될 수 있어 같은 그룹에서의 각 위치로부터의 데이터가 캐시의 같은 웨이에 배치되어야 한다.

[0023] 선택적으로, 유효 정보는 또한 데이터가 캐시 내에 있는지 아닌지를 예측하기 위해서 사용될 수 있다. 유효 정보가 정확하다고 보장될 수 있으면, 그리고나서 유효 정보가 캐시 내에 데이터가 저장되어 있지 않다는 것을 나타내면, 이것은 캐시 액세스가 캐시 미스(cache miss)를 유발해서, 데이터가 캐시 자체에 액세스하는 일없이 하부 레벨 캐시 또는 메모리로부터 페치될 수 있다는 것을 나타낸다. 대응하는 웨이 테이블 엔트리가 웨이 테이블로부터 축출되더라도 유효 정보가 정확하다는 것을 보장하기 위해서, 스토리지를 제공해서 축출된 웨이 테이블 엔트리가 웨이 테이블로 돌아올 때까지 축출된 웨이 테이블 엔트리를 버퍼링할 수 있다.

[0024] 본 기술에는 가상으로 태그된 캐시를 갖는 시스템이 이용될 수 있다. TLB에 의해 제공된 가상-물리 어드레스 맵핑은 가상으로 태그된 캐시에의 캐시 액세스를 행하는 데에 필수적이지 않지만, TLB는 그럼에도 불구하고 다른 이유로, 예를 들면 메모리 액세스가 필요한 경우에는, 어드레스 변환을 행할 수 있다. 그러므로, 그러한 시스템에 있어서는, 웨이 테이블은, TLB 변환이 행해지고 있고 웨이 테이블에 인덱스로서 TLB의 검색을 재이용한다는 사실을 이용할 수 있다.

[0025] 그렇지만, 본 기술은 태그 데이터가 대응하는 캐시 위치에 저장된 데이터와 관련된 물리 어드레스의 일부를 식별하는, 물리적으로 태그된 캐시를 갖는 시스템에 특히 유용하다. 그러한 시스템에 있어서, TLB에 의해 행해진 변환은, 캐시에 저장된 데이터의 어드레스를 식별하기 위해서, 캐시가 물리 어드레스를 이용하고 가상 어드레스를 이용하지 않기 때문에 캐시에 액세스하는 데에 필요하다. 그러므로, 캐시 액세스가 필요한 경우, TLB 변환은 이미 타겟 TLB 엔트리를 이용해서 행해질 것이고, 그래서 가상 어드레스에 의거한 TLB의 검색은, 웨이 테이블의 추가 검색 없이 대응하는 웨이 테이블 엔트리를 식별하기 위해서도 재이용될 수 있다.

[0026] TLB는 일반적으로 다수의 엔트리로 이루어지는 내용 어드레스지정 메모리를 구비할 수 있고, 각 엔트리는 가상 어드레스와 물리 어드레스 간의 맵핑을 저장한다. 일반적으로, TLB는 가상 어드레스에 의해 검색가능할 것이다. 그렇지만, 본 기술은, TLB가 물리 어드레스에 의해서도 검색될 수 있도록 TLB을 수정하는 것이 유용하다는 것을 인식한다. 이것은 물리적으로 태그된 캐시에 의해, 캐시가 물리 어드레스의 일부를 이용해서 데이터를 식별하기 때문이다. 웨이 테이블 엔트리가 캐시 할당 또는 캐시 축출에 응답해서 갱신되는 경우, 캐시는 할당된 데이터 혹은 축출된 데이터의 물리 어드레스의 일부를 TLB 및 웨이 테이블에 제공할 것이다. 물리 어드레스에 의해 검색가능하도록 TLB를 수정함으로써, 이것에 의해 웨이 테이블 엔트리가 TLB의 검색에 의해 식별될 수 있다.

[0027] 상기 설명한 바와 같이, 한정된 능력의 웨이 테이블로 인해, 웨이 테이블로부터 웨이 테이블 엔트리를 축출해서 다른 웨이 테이블 엔트리에 대한 공간을 만드는 것이 가능하다. 축출된 웨이 테이블 엔트리가 후단에서 웨이 테이블로 다시 로드(load)되는 경우에, 캐시 내의 대응하는 데이터가 캐시 내에 남아 있더라도, 어떤 유효 또는 웨이 정보를 표시하지 않을 수 있다. 그러므로, 대응하는 웨이 정보가 나중에 갱신되지 않으면, 데이터에의 다음 액세스는, 어느 웨이가 데이터를 저장하고 있는지를 식별하기 위해서 모든 캐시 웨이 및 태그 어레이를 이용하는 풀 캐시 액세스를 필요로 할 것이다. 비록 이것은 웨이 정보를 이용하는 감소된 액세스에 비해서 에너지 소모 페널티를 발생시키지만, 대응하는 캐시 데이터가 축출되기 전에 나중에 다시 축출되어 필요하게 되는 웨이 테이블 엔트리의 개수가 상당히 낮기 때문에, 이것은 중요한 문제가 아니다. 그러므로, 대응하는 캐시된 데이터를 할당하거나 축출하는 일없이 웨이 테이블에 복구되었던 엔트리의 웨이 정보를 갱신하기 위한 메카니즘을 제공하는 것이 필수적인 것은 아니다.

[0028] 그렇지만, 웨이 정보를 이용할 수 있는 캐시 액세스의 퍼센티지를 증가시켜서 에너지 효율을 향상시키기 위해서, 이 시나리오에서는 웨이 테이블 엔트리를 갱신하기 위한 갱신 메카니즘을 제공할 수 있다. 따라서, 데이터 액세스 요구 서비스시에, 원하는 데이터에 대한 웨이 정보가 어느 캐시 웨이가 타겟 데이터("웨이 테이블 미스")를 저장하는지를 나타내지 않는다고 판정하지만, 캐시 웨이 중의 하나로부터 액세스된 데이터가 타겟 데이터("캐시 히트(cache hit)")에 대응한다는 것을 캐시가 태그 데이터로부터 판정하면, 캐시는 웨이 테이블을 제어해 웨이 정보를 갱신해서 타겟 데이터를 저장하고 있는 캐시 웨이를 식별할 수 있다. 이렇게 함으로써, 웨이 테이블 엔트리가 웨이 테이블로부터 축출되어 나중에 웨이 테이블에 다시 배치되었더라도, 각 메모리 위치에 대한 웨이 테이블 정보는, 또 다른 캐시 액세스가 그 위치에 대응하는 데이터에 액세스할 때 재현될 수 있다. 그러므로, 풀 캐시 액세스 및 웨이 테이블 갱신이 필요할 때 초기 에너지 패널티가 있을 것이며, 웨이 정보가 갱신되었던 메모리 위치와 관련된 데이터에의 어떤 추가 액세스는, 타겟 캐시 웨이에만 액세스함으로써 에너지 소모를 감소시킬 수 있도록 갱신된 웨이 정보를 이용할 수 있을 것이다.

[0029] 웨이 테이블 미스 및 캐시 히트에 응답해서 웨이 정보를 갱신할 때, 웨이 테이블은 어느 웨이 테이블 엔트리가 갱신되어야 하는지를 식별해야 할 것이다. TLB 또는 웨이 테이블을 검색해서 정확한 웨이 테이블 엔트리를 식별해 갱신하는 것은 에너지 집약적일 수 있다. 갱신을 행할 때 소비되는 에너지를 줄이기 위해서, 웨이 테이블에는 가장 최근에 액세스된 웨이 테이블 엔트리를 식별하는 웨이 테이블 엔트리 식별자를 저장하는 저장소자가 설치될 수 있다. 웨이 테이블 미스 및 캐시 히트에 응답해서 웨이 정보가 갱신될 필요가 있으면, 웨이 테이블은 풀 테이블 검색을 행하는 일없이 웨이 테이블 엔트리로부터 갱신되어야 하는 엔트리를 식별할 수 있다. 이것은 갱신의 에너지 효율을 향상시킨다.

[0030] TLB는 TLB의 계층의 하나일 수 있다. 예를 들면, TLB는 제1 레벨 TLB일 수 있고 장치는 적어도 한 개의 추가 레벨 TLB을 구비할 수 있다. 일반적으로, 제1 레벨 TLB는 비교적 작아서 빨리 에너지 효율적으로 액세스될 수 있고, 추가 레벨 TLB는 더 커서 좀더 많은 엔트리를 저장할 수 있지만, TLB 엔트리를 액세스할 때 더 많은 에너지를 소비하여 더 큰 레이턴시를 갖는다. 그 이유는 더 큰 구조가 일반적으로 엔트리에 액세스하기 위해서 증가된 트랜지스의 수를 활성화시키는 것을 필요로 하기 때문이다. 타겟 가상 어드레스에 대응하는 타겟 데이터에 대한 데이터 액세스 요구에 응답해서, 제1 레벨 TLB는 타겟 가상 어드레스에 대응하는 타겟 TLB 엔트리를 검색할 수 있다. 제1 레벨이 타겟 TLB 엔트리를 저장하면, 적은 에너지 오버헤드(overhead)로 빨리 변환을 수행할 수 있다. 그렇지만, 제1 레벨 TLB가 타겟 TLB 엔트리를 저장하지 않으면, 타겟 TLB 엔트리가 추가 레벨 TLB로부터 액세스될 수 있고 제1 레벨 TLB는 타겟 TLB 엔트리를 저장하도록 갱신될 수 있다. 이 TLB의 계층은 멀티 레벨 캐시 계층과 유사하다.

[0031] 적어도 제1 레벨 TLB에 대해서는 웨이 테이블이 제공될 수 있다. 그러므로, 제1 레벨 TLB의 각 TLB 엔트리는 웨이 테이블에 대응하는 웨이 테이블 엔트리를 가질 수 있다. 어떤 추가 레벨 TLB가 대응하는 웨이 테이블 엔트리를 갖는 것은 필수적인 것은 아니다. TLB 엔트리가 제1 레벨 TLB에 제시되지 않으면, 하부 레벨로부터 페치될 필요가 있을 것이다. 적어도 한 개의 추가 레벨 TLB를 제공하면 TLB 엔트리를 제1 레벨 TLB에 페치하는 것과 관련된 레이턴시를 상당히 줄일 수 있다. 그 이유는 메모리의 페이지 테이블로부터 페이지 엔트리를 페치할 필요성을 제거하기 때문이다. 반대로, 필요한 웨이 테이블 엔트리가 웨이 테이블에 제시되지 않으면, 웨이 테이블 미스와 관련된 페널티가 TLB 미스와 관련된 페널티보다 훨씬 적다. 하부 레벨 TLB 엔트리 또는 페이지 테이블 엔트리를 참조해서 가상-물리 어드레스 맵핑이 결정되는, TLB 엔트리와 달리, 캐시 액세스가 이루어질 때, 상술한 바와 같이 웨이 정보가 웨이 테이블 미스 및 캐시 히트 시나리오에 응답해서 갱신될 수 있기 때문에 웨이 테이블의 웨이 정보가 스크래치(scratch)로부터 복원될 수 있다. 따라서, 추가 레벨 웨이 테이블보다는 추가 레벨 TLB를 제공하는 것이 더 중요하므로, 회로 면적 및 전력 소모를 절약하기 위해서, 제1 레벨 TLB에 대응하는 제1 레벨 웨이 테이블을 제공하지만, 추가 레벨 TLB에 대응하는 추가 레벨 웨이 테이블을 제공하지 않는 것은 충분할 수 있다.

[0032] 한편, 웨이 테이블 정보의 커버리지(coverage)를 향상시키고, 웨이 테이블 미스/캐시 히트 시나리오에 응답해서 웨이 정보를 갱신할 필요의 가능성을 줄이기 위해서, 웨이 테이블은 적어도 추가 레벨 TLB에 대응하는 적어도 한 개의 추가 레벨 웨이 테이블을 가진 멀티 레벨 계층에도 제공될 수 있다. 그러므로, 타겟 TLB 엔트리에 대응하는 웨이 테이블 엔트리가 제1 레벨 웨이 테이블에서 발견되지 않은 경우에는, 추가 레벨 웨이 테이블로부터 액세스될 수 있다. 또한, 제1 레벨 웨이 테이블이 새롭게 액세스된 엔트리로 갱신될 수 있다. TLB 및 웨이 테이블의 멀티 레벨 계층을 제공함으로써, 빠른 에너지 효율적인 액세스와 증가된 웨이 테이블 커버리지 간의 더 나은 밸런스를 달성할 수 있다. 자주 필요한 엔트리들은 좀더 에너지 효율적인 방식으로 더 빠른 액세스를 가능하게 하기 위해서 제1 레벨 TLB 및 웨이 테이블에 저장될 수 있고, 추가 레벨 TLB 및 추가 레벨 웨이 테이블은 많

은 덜 자주 사용되는 엔트리들의 저장을 허용하기 위해서 백업 스토어(backup store)로서 제공될 수 있다.

[0033] 제1 레벨 TLB 및 제1 레벨 웨이 테이블은 "마이크로 TLB" 및 "마이크로 웨이 테이블"이라고도 칭할 수 있고, 추가 레벨 TLB 및 추가 레벨 웨이 테이블은 간단히 TLB 및 웨이 테이블이라고 또는 "하부 레벨" TLB 및 웨이 테이블이라고 칭할 수 있다.

[0034] TLB 및 웨이 테이블은 서로 다른 방식으로 배열될 수 있다. 일 실시예에 있어서는, TLB 및 웨이 테이블은 데이터 처리장치에 의해 개별적으로 저장될 수 있다. 예를 들면, 별개의 메모리가 사용될 수도 있다. 이 경우에, 웨이 테이블에 액세스하기 위해서, TLB는 타겟 TLB 엔트리의 위치를 지정할 수 있고, 타겟 TLB 엔트리에 대응하는 웨이 테이블 엔트리를 식별하는 웨이 테이블에 인덱스를 제공할 수도 있다. 그 대신에, 웨이 테이블은 필요한 웨이 테이블 엔트리를 검색할 수 있게 되고, TLB에 타겟 TLB 엔트리의 식별자를 제공할 수 있다. 그러므로, 웨이 테이블 및 TLB가 별도로 제공되더라도, 다른 테이블의 추가 검색 없이 양 테이블에 관련 엔트리를 배치하는 데에는 TLB 또는 웨이 테이블의 단 한 개의 검색이면 충분하다.

[0035] 그 대신에, TLB 및 웨이 테이블은 복수의 엔트리를 저장하는 공통 메모리를 구비할 수 있다. 각 엔트리는 TLB 엔트리에 대응하는 일부분과 웨이 테이블 엔트리에 대응하는 또 다른 부분을 가질 수 있다. 이 경우, 공통 메모리는 데이터 액세스 요구의 가상 어드레스에 의거해서 대응하는 엔트리를 간단히 검색할 수 있고, 대응하는 웨이 테이블 엔트리의 웨이 정보 및 대응하는 TLB 엔트리의 맵핑된 물리 어드레스 양자를 복귀시킬 수 있다.

[0036] 추가 국면에서 보면, 본 발명은

[0037] 데이터를 처리하는 처리 수단과,

[0038] 데이터를 저장하고, 복수의 캐시 웨이들을 구비하는 캐시 수단과,

[0039] 메모리 위치들의 관련 페이지에 대한 가상-물리 어드레스 맵핑을 각각이 식별하는 복수의 TLB(translation lookaside buffer) 엔트리를 저장하는 TLB 수단과,

[0040] 상기 복수의 TLB 엔트리 중의 하나에 각각이 대응하는 복수의 웨이 테이블 엔트리를 저장하는 웨이 테이블 수단을 구비하는, 데이터 처리장치를 제공하고,

[0041] 각 웨이 테이블 엔트리는, 대응하는 TLB 엔트리와 관련된 페이지의 각 메모리 위치에 대해서, 상기 캐시 수단의 상기 캐시 웨이들 중의 어느 것이, 만약 있다면, 메모리 위치와 관련된 데이터를 저장하는지를 식별하기 위한 웨이 정보를 포함하고,

[0042] 타겟 가상 어드레스에 대응하는 타겟 데이터에 대한 상기 처리 수단으로부터의 데이터 액세스 요구에 응답해서, 상기 TLB 수단은, 상기 타겟 가상 어드레스에 대응하는 타겟 TLB 엔트리에 액세스하도록 구성되고, 상기 웨이 테이블 수단은, 상기 타겟 TLB 엔트리에 대응하는 웨이 테이블 엔트리에 액세스하도록 구성되며, 상기 캐시 수단은 타겟 데이터와 관련된 메모리 위치에 대한 상기 대응하는 웨이 테이블 엔트리에 포함된 웨이 정보에 의존해서 캐시 액세스를 행하도록 구성된다.

[0043] 또 다른 국면에서 보면, 본 발명은 데이터를 저장하기 위한 복수의 캐시 웨이들(cache ways)를 구비하는 캐시와, 메모리 위치들의 관련 페이지에 대한 가상-물리 어드레스 맵핑을 각각이 식별하는 복수의 TLB(translation lookaside buffer) 엔트리를 저장하도록 구성된 TLB와, 상기 복수의 TLB 엔트리 중의 하나에 각각이 대응하는 복수의 웨이 테이블 엔트리를 저장하도록 구성된 웨이 테이블을 구비하고, 각 웨이 테이블 엔트리가, 대응하는 TLB 엔트리와 관련된 페이지의 각 메모리 위치에 대해서, 상기 캐시의 상기 캐시 웨이들 중의 어느 것이, 만약 있다면, 메모리 위치와 관련된 데이터를 저장하는지를 식별하기 위한 웨이 정보를 포함하는, 데이터 처리장치의 방법을 제공하고,

[0044] 상기 방법은,

[0045] 타겟 가상 어드레스에 대응하는 타겟 데이터에 대한 데이터 액세스 요구에 응답해서, 상기 타겟 가상 어드레스에 대응하는 타겟 TLB 엔트리에 액세스하는 상기 TLB, 및 상기 타겟 TLB 엔트리에 대응하는 웨이 테이블 엔트리에 액세스하는 상기 웨이 테이블과,

[0046] 타겟 데이터와 관련된 메모리 위치에 대한 상기 대응하는 웨이 테이블 엔트리에 포함된 웨이 정보에 의존해서 캐시 액세스를 행하는 상기 캐시를 구비한다.

[0047] 또 다른 국면에서 보면, 본 발명은

[0048] 데이터에 대한 액세스 요구들을 발행하도록 구성된 처리 회로와,

[0049] 상기 액세스 요구들에 응답해서 데이터에의 액세스를 제공하도록 구성된 캐시와,

[0050] 상기 액세스 요구들에서 지정된 가상 페이지 어드레스들과 상기 캐시가 사용하는 물리 페이지 어드레스들 사이에서 변환하도록 구성된 TLB(translation lookaside buffer)와,

[0051] 상기 처리 회로에 의해 발행된 펜딩(pending) 액세스 요구들을 버퍼링하도록 구성된 입력 버퍼와,

[0052] 상기 입력 버퍼로부터의 상기 펜딩 액세스 요구들 중의 어느 것이 각 처리 사이클에서 상기 캐시에 의해 서비스되어야 하는지를 선택하도록 구성된 아비터(arbiter)를 구비하는, 데이터 처리장치를 제공하고,

[0053] 상기 캐시는 같은 처리 사이클에서 M개의 액세스 요구들을 서비스하는 것이 가능하고, 여기서 M은 정수이며,

[0054] 상기 아비터는 선택된 펜딩 액세스 요구들이 최대 N개의 상이한 가상 페이지 어드레스들을 지정하는 것을 보장하기 위해서 상기 펜딩 액세스 요구들을 선택하도록 구성되고, 여기서 N은 정수이며, N < M이다.

[0055] 변환 색인 버퍼를 갖는 시스템에서 캐시 액세스의 효율을 향상시킬 수 있는 또 다른 방법은, 캐시가 같은 처리 사이클에서 다수의 액세스 요구들을 서비스하는 방법과 관련이 있다. 캐시는 같은 처리 사이클에서 M개의 액세스 DYN를 서비스할 수 있고, 여기서 M은 1보다 큰 정수이다. 예를 들면, 각 뱅크가 동시에 캐시 요구를 독립적으로 서비스할 수 있도록 몇몇 캐시들을 쌓아올려(banked)도 된다.

[0056] 주어진 사이클에서 캐시에 의해 서비스될 수 있는 것보다 펜딩(pending)중인 더 많은 액세스 요구들이 있을 수도 있다. 그러므로, 본 기술은 처리 사이클들 사이에서 펜딩 액세스 요구들을 버퍼링하기 위한 입력 버퍼와, 펜딩 액세스 요구 중의 어느 것이 각 처리 사이클에서 캐시에 의해 서비스되어야 하는지를 선택하기 위한 아비터를 구비한다. 이 아비터는 캐시에 의해 서비스하기 해서 선택된 요구들이 최대 N개의 상이한 가상 페이지 어드레스들을 지정하는 것을 보장하기 위해서 펜딩 액세스 요구들을 선택하도록 구성되고, 여기서 N은 M보다 적은 정수이다. 가상 페이지 어드레스는 같은 가상-물리 어드레스 맵핑을 공유하는 모든 어드레스에 대해서 동일한 가상 어드레스의 일부이며, 즉 동일한 가상 페이지 어드레스를 공유하는 어드레스들은 TLB의 같은 TLB 엔트리를 이용해서 변환될 수 있다.

[0057] 같은 사이클에서 서비스될 수 있는 상이한 가상 페이지 어드레스들의 수를, 같은 처리 사이클에서 캐시에 의해 처리될 수 있는 총 액세스 요구들의 수보다 적게 되도록 제한하는 직관에 반대되는 것처럼 보일 수 있다. 상이한 가상 페이지 어드레스를 이용하는 몇몇 요구들이 선택되는 것을 방지하기 때문에 캐시의 능력이 완전히 사용되지 않는 상황으로 이어질 수 있다고 예상할 것이다. 그렇지만, 본 기술은, 처리회로가 동일한 가상 페이지 어드레스를 각각이 겨냥하는, 서로 근접해 있는 몇몇의 펜딩 액세스 요구들을 발행하는 것이 상당히 일반적이라고 인식한다. 예를 들면, 많은 프로그램 어플리케이션은 동일한 페이지 내의 인접하는 어드레스들과 관련된 데이터 또는 같은 데이터에의 연속적인 액세스를 필요로 한다. 그러므로, 주어진 사이클 중에 펜딩 중인 액세스 요구들이 비교적 제한된 세트의 상이한 가상 페이지 어드레스들을 겨냥하는 것이 일반적이다.

[0058] 본 기술은, 같은 사이클 내에서 처리될 수 있는 상이한 가상 페이지 어드레스들의 수를 제한함으로써, 변환 색인 버퍼(TLB:translation lookaside buffer) 및 아비터와 같은 시스템의 소자들을 좀더 효율적으로 구현할 수 있다고 인식하고 있다. 캐시가 같은 사이클에서 많은 상이한 가상 페이지 어드레스들을 겨냥하는 액세스 요구들을 서비스해야 했다면, TLB는 다수의 상이한 TLB 엔트리를 이용해서, 같은 사이클에서 많은 어드레스 변환을 행해야 할 것이고, 이것은 에너지 소모의 관점에서 비용이 많이 들것이다. 이 변환 색인 버퍼는 변환을 위해서 가상 페이지 어드레스들을 수신하기 위한 많은 포트를 가질 수도 있다. 또한, 아비터 또는 캐시는 데이터 액세스 요구들에 대한 물리 페이지 어드레스들을 비교하기 위한 비교기를 가질 수도 있다. 많은 상이한 페이지 어드레스들이 같은 사이클에서 처리되도록 요구되면, 그러한 많은 TLB 포트 및 비교기가 설치되어야 할 것이다. 각 포트 및 비교기는 동적 리키지 및 정적 리키지 양자에 시달려서, 데이터 처리장치의 에너지 소모를 증가시킬 것이다.

[0059] 그러므로, 동일한 처리 사이클에서 처리될 수 있는 상이한 가상 페이지 어드레스들의 수를 제한함으로써, 일부 액세스 요구들이 추가 사이클을 기다려야 하기 때문에 약간의 성능 저하가 있더라도, 이것은 드물며 TLB, 아비터 및 캐시의 복잡성을 줄임으로써 달성되는 에너지 소모의 감소는 이것을 보상하는 것 이상이기 때문에, 전체적인 효율 향상을 달성할 수 있다.

[0060] TLB는 같은 처리 사이클에서 N개의 상이한 가상 페이지 어드레스들을 변환할 수 있을 것이다. 그러므로, TLB는 캐시에 의해 서비스될 수 있는 M개의 액세스 요구 모두를 변환하기 위한 회로를 구비하고 있을 필요가 없다. 예

를 들면, 가상 페이지 어드레싱을 수신하기 위해서 TLB가 가지고 있는 포트의 개수와, 필요한 테이블 검색(table lookups)의 개수가, 시스템에 있어서의 정적 리키지 및 동적 리키지를 줄이기 위해서 감소될 수 있다.

[0061] 캐시에 의해 서비스되는 액세스 요구를 선택할 때, 아비터는, N개의 상이한 가상 페이지 어드레스들 중의 하나를 지정하는 이들 펜딩 액세스 요구를 후보 액세스 요구들로서 선택할 수도 있다. 이 아비터는 어느 요구들이 캐시에 의해 실제로 서비스되어야 하는지를 결정하기 위해서 후보 액세스 요구들 중에서 선택할 수도 있다.

[0062] 일 실시예에 있어서는, N=2이므로, 각 사이클에 있어서 캐시는 최대 2개의 상이한 가상 페이지 어드레스를 지정하는 펜딩 액세스 요구들만 처리할 수도 있다. 이것은 제1 메모리 위치로부터 판독된 후에 제2 메모리 위치에 기록되는 데이터를 필요로 하는 mem-카피 동작과 같은 몇몇 동작들이 있을 수 있기 때문에 유용할 수 있다. 그러한 동작은 종종 2개의 상이한 페이지 어드레스를 필요로 하는데, 하나는 제1 메모리 위치와 관련되어 있고, 또 다른 하나는 제2 메모리 위치와 관련되어 있다. 2개의 상이한 페이지 어드레스를 지정하는 액세스 요구들이 같은 사이클에서 처리될 수 있게 함으로써, 그러한 카피 동작이 좀더 효율적으로 수행될 수 있다. 그럼에도 불구하고, 동시에 처리될 수 있는 상이한 페이지 어드레스들의 개수를 2개로 제한함으로써, TLB, 아비터 및 캐시가 더 에너지 효율적인 방식으로 구현될 수 있다.

[0063] 그렇지만, 많은 실시예에 있어서는, N=1이면 충분하므로, 각 사이클에 있어서, 캐시에 의해 서비스하기 위해서 선택된 모든 펜딩 액세스 요구들이 같은 동일한 가상 페이지 어드레스를 지정한다. 이것은 바람직할 것이라는 직관에 반대되는 것처럼 보일 수 있지만, 실제로 같은 사이클에서의 많은 액세스 요구들은 동일한 페이지에 액세스하므로 이것은 중요한 성능 손실을 나타내지 않는다. 단지 한 개의 가상 페이지 어드레스만이 동일한 사이클에서 사용되도록 허용함으로써, TLB는 사이클당 한 개의 어드레스 변환을 행해야 하고, 아비터 또는 캐시 내의 물리 페이지 어드레스들을 비교하기 위한 어떤 비교기들은 그 사이클에서 서비스되는 모든 펜딩 요구들이 같은 페이지 어드레스를 갖는다고 가정할 수 있다. 이것은 메모리 어드레스의 보다 소수의 어드레스 비트들이 필요한 데이터를 식별하기 위해서 비교될 필요가 있다는 것을 의미하기 때문에 유용하다. 왜냐하면, 페이지 어드레스에 대응하는 어드레스 비트들이 각 요구에 대해서 같을 것이라고 가정할 수 있기 때문이다. 그러므로, 시스템을 사이클당 한 개의 페이지 어드레스로 제한함으로써 캐시를 액세스하기 위한 좀더 효율적인 시스템이 제공된다.

[0064] 동일한 사이클에서 서비스된 모든 액세스 요구들이 동일한 가상 페이지 어드레스를 겨냥하는 것을 보장하기 위해서, 아비터는 상이한 펜딩 액세스 요구들 간의 비교 동작을 행해도 된다. 예를 들면, 각 처리 사이클에 있어서, 펜딩 요구들 중의 하나가 주요한 액세스 요구로서 선택되어도 되고, 주요한 액세스 요구의 가상 페이지 어드레스가 TLB에 의해서 변환되어도 된다. 주요한 액세스 요구와 같은 가상 페이지 어드레스를 지정하는 다른 액세스 요구들만이 그 사이클에서 캐시에 의해 서비스되기 위한 후보여도 되고, 아비터는 어느 요구들이 실제로 서비스되어야 하는지를 결정하기 위해서 이들 후보들 중에서 선택해도 된다. 다음 사이클에 있어서는, 다른 요구가 주요한 요구로서 선택되어도 되므로, 다른 페이지 어드레스가 처리되어도 된다.

[0065] 주요한 액세스 요구는 입력 버퍼 내의 펜딩 액세스 요구들 중의 어느 하나여도 된다. 예를 들면, 주요한 액세스 요구가 펜딩 액세스 요구들 중의 가장 오래된 것이어도 된다. 이렇게 함으로써, 펜딩 액세스 요구가 무한정 입력 버퍼 내에 갇혀 있을 가능성이 감소된다. 그 대신에, 각 액세스 요구가 그것에 할당된 우선순위를 가져도 되고, 주요한 액세스 요구가 가장 높은 우선순위를 갖는 요구들 중의 하나여도 된다. 또한, 액세스 요구가 서로 다른 타입을 가져도 되고 몇몇 타입은 다른 타입보다 먼저 우선순위가 정해져도 되므로, 주요한 액세스 요구가 가장 높은 우선순위의 액세스 타입을 갖는 액세스 요구들 중의 하나여도 된다.

[0066] 동일한 사이클에서 처리되도록 허용되는 한 개 이상의 가상 페이지 어드레스가 있으면(즉, N=2 이상), 아비터는 동일한 사이클에서 처리되고 있는 서로 가상 페이지 어드레스에 대한 유사한 비교를 행해도 된다.

[0067] 캐시의 동일한 캐시 라인에 저장된 데이터를 겨냥하는 다수의 후보 액세스 요구가 있으면, 아비터는 다수의 액세스 요구들을 병합해도 되고, 각 개개의 요구가 차례차례 처리되었다면 캐시는 기록 또는 판독되는 병합된 데이터에 대응하는 한 개의 병합된 요구를 서비스해도 된다. 이것에 의해 오히려 펜딩 요구들이 동일한 사이클에서 서비스될 수 있게 된다.

[0068] 아비터는 단 한 개의 요구로 병합될 수 있는 요구들의 개수를 제한해도 된다. 예를 들면, 입력 버퍼가 X개의 펜딩 액세스 요구들을 버퍼링할 수 있으면, 아비터는 최대 Y개의 후보 액세스 요구들로부터 병합해도 된다. 여기서 X 및 Y는 정수이며 $Y \leq X$이다. $Y < X$인 경우는 요구들이 같은 캐시 라인에 액세스하는지 여부를 판정하기 위해서 각 요구와 관련된 어드레스들을 비교하기 위한 회로가 작은 면적 및 전력 소모 비용을 초래하기 때문에

유용하다. 모든 X 펜딩 요구들이 같은 캐시 라인에 액세스할 것 같지 않다. 따라서, 면적 및 전력 소모를 절약하기 위해서, 병합될 수 있는 후보 액세스 요구의 개수를 Y로 제한할 수 있다.

[0069] 캐시를 쌓아오려도 되므로 데이터를 저장하기 위한 캐시 라인의 M개의 뱅크가 있으며 각 뱅크는 다른 뱅크들과 독립적으로 액세스 가능하다. 이것은 캐시가 동일한 처리 사이클에서 서로 다른 뱅크들에 지시된 M개의 액세스 요구까지 서비스할 수 있다는 것을 의미한다. 이 경우, 주어진 사이클에 대해서 허용된 N개의 상이한 가상 페이지 어드레스들 중의 하나를 지정하는 후보 요구들이 선택되면, 아비터는 각 뱅크를 겨냥하는 어떤 뱅크 후보 액세스 요구들이 있는지 여부를 판정할 수 있다.

[0070] 적어도 한 개의 뱅크 후보 액세스 요구를 가지고 있는 각 뱅크에 대해서는, 아비터는 제1 뱅크 후보 액세스 요구로서 뱅크 후보 액세스 요구들 중의 하나를 선택해도 되고, 제1 뱅크 후보 액세스 요구와 같은 캐시 라인을 겨냥하는 1개 이상의 다른 뱅크 후보 액세스 요구가 있는지를 판정해도 된다. 같은 캐시 라인을 겨냥하는 다른 뱅크 후보 액세스 요구들이 있으면, 이들 요구는 제1 뱅크 후보 액세스 요구와 병합되어도 되고, 병합된 요구는 캐시의 뱅크에 의해 서비스하기 위해서 선택된다. 그렇지 않으면, 제1 뱅크 후보 액세스 요구들이 서비스를 위해서 선택되어도 된다. 캐시 액세스들이 캐시의 뱅크드 구조를 이용할 수 있는 뱅크들 전체에 걸쳐 퍼져 있는 동안 한 개의 사이클에서 동일한 캐시 라인을 겨냥하는 다수의 요구들이 서비스될 수 있기 때문에 이 기술은 좀 더 효율적인 캐시의 사용 및 성능 향상을 가능하게 한다.

[0071] 다시, 아비터는 그 뱅크에 대해서 펜딩 중인 후보 요구들 중의 어느 하나로서 제1 뱅크 후보 액세스 요구를 선택해도 된다. 예를 들면, 제1 뱅크 액세스 요구는,

[0072] 가장 오래된 뱅크 후보 액세스 요구와,

[0073] 가장 높은 우선순위를 갖는 뱅크 후보 액세스 요구와,

[0074] 가장 높은 우선순위의 액세스 타입을 갖는 뱅크 후보 액세스 요구 중 하나여도 된다.

[0075] 아비터는 하나의 사이클에서 처리될 수 있는 가상 페이지 어드레스의 개수를 제한하기 때문에, 그 사이클에서 처리될 수 없는 펜딩 액세스 요구들이 몇 개 있을 수 있다. 이들 요구를 수용하기 위해서, 입력 버퍼는 현재의 처리 사이클에서 아비터에 의해 선택되지 않는 펜딩 액세스 요구를 저장하는 저장회로를 가질 수도 있으므로, 다음 처리 사이클에서 아비터에 의한 선택에 이용가능하다. 또한, 다음 처리 사이클에 있어서, 아비터는, 요구들의 전진(forward progress)을 보장하기 위해서, 좀더 새로운 요구들에 대해서 저장회로부터의 오래된 요구를 선호하도록 구성되어도 된다.

[0076] 또 다른 국면에서 보면, 본 발명은

[0077] 데이터에 대한 액세스 요구들을 발행하는 처리 수단과,

[0078] 상기 액세스 요구들에 응답해서 데이터에의 액세스를 제공하는 캐시 수단과,

[0079] 상기 캐시 수단에 의해 사용된 물리 페이지 어드레스들과 상기 액세스 요구들에서 지정된 가상 페이지 어드레스들 사이에서 변환하는 TLB(translation lookaside buffer) 수단과,

[0080] 상기 처리 수단에 의해 발행된 펜딩 액세스 요구들을 버퍼링하는 입력 버퍼 수단과,

[0081] 상기 입력 버퍼로부터의 상기 펜딩 액세스 요구들 중의 어느 것이 각 처리 사이클에서 상기 캐시 수단에 의해 서비스되어야 하는지를 선택하는 중재 수단을 구비하는 데이터 처리장치를 제공하고,

[0082] 상기 캐시 수단은 같은 처리 사이클에서 M개의 액세스 요구들을 서비스할 수 있고, 여기서 M은 정수이며,

[0083] 상기 중재 수단은 선택된 펜딩 액세스 요구들이 최대 N개의 상이한 가상 페이지 어드레스를 지정한다는 것을 보장하기 위해서 상기 펜딩 액세스 요구들을 선택하도록 구성되고, 여기서 N은 정수이며 N < M이다.

[0084] 또 다른 국면에서 보면, 본 발명은 데이터에 대한 액세스 요구들을 발행하도록 구성된 처리 회로와, 상기 액세스 요구들에 응답해서 데이터에의 액세스를 제공하도록 구성된 캐시와, 상기 캐시에 의해 사용된 물리 페이지 어드레스들과 상기 액세스 요구들에서 지정된 가상 페이지 어드레스들 사이에서 변환하도록 구성된 TLB(translation lookaside buffer)를 구비하는 데이터 처리장치의 방법을 제공하고,

[0085] 상기 방법은,

[0086] 입력 버퍼에서 상기 처리 회로에 의해 발행된 펜딩 액세스 요구들을 버퍼링하는 단계와,

[0087] 상기 입력 버퍼로부터 상기 펜딩 액세스 요구들 중의 어느 것이 각 처리 사이클에서 상기 캐시에 의해 서비스되어야 하는지를 선택하는 단계를 포함하고,

[0088] 상기 캐시는 같은 처리 사이클에서 M개의 액세스 요구들을 서비스할 수 있고, 여기서 M은 정수이며,

[0089] 상기 선택 단계는 선택된 펜딩 액세스 요구들이 최대 N개의 상이한 가상 페이지 어드레스들을 지정한다는 것을 보장하기 위해서 상기 펜딩 액세스 요구들을 선택하며, 여기서 N은 정수이고, N < M이다.

[0090] 상기, 및 본 발명의 다른 목적, 특징 및 이점은 첨부도면과 관련해서 판독되는 예시적인 실시예의 이하의 상세한 설명으로부터 밝혀질 것이다.

도면의 간단한 설명

[0091] 도 1은 변환 색인 버퍼 및 캐시를 갖는 데이터 처리장치의 일례를 개략적으로 나타낸 것이다.

도 2는 뱅크 및 웨이를 갖는 캐시 구조의 일례이다.

도 3은 다수의 메모리 위치를 각각이 포함하는 페이지들로 이루어진 어드레스 공간을 나타낸 것이다.

도 4는 어드레스 변환 시에 그리고 캐시에 의해 사용될 때 메모리 어드레스를 해석하는 방법의 일례를 나타낸 것이다.

도 5는 TLB와 웨이 테이블 간의 관계와 웨이 테이블 엔트리 내의 유효 정보 및 웨이 정보를 나타내는 몇 가지의 예를 나타낸 것이다.

도 6은 웨이 테이블에 저장된 웨이 정보를 이용해서 캐시 액세스를 행하는 방법을 나타낸 것이다.

도 7은 웨이 테이블에 저장된 웨이 정보를 갱신하는 제1 예를 나타낸 것이다.

도 8은 웨이 테이블 내의 웨이 정보를 갱신하는 또 다른 예를 나타낸 것이다.

도 9는 캐시 라인필(linefill)에 응답해서 웨이 정보를 갱신하는 방법을 나타낸 것이다.

도 10은 캐시 라인 축출에 응답해서 웨이 정보를 갱신하는 방법을 나타낸 것이다.

도 11은 웨이 정보가 알려져 있지 않지만 대응하는 데이터가 캐시에 발견될 때 웨이 정보를 갱신하는 방법을 나타낸다.

도 12는 같은 사이클에서 다수의 액세스를 처리할 수 있는 캐시를 갖는 데이터 처리장치의 또 다른 예를 나타낸 것이다.

도 13은 캐시에 의해 서비스하는 것을 기다리는 펜딩 액세스 요구를 버퍼링하기 위한 입력 벡터의 예를 나타낸 것이다.

도 14는 어느 펜딩 액세스 요구가 캐시에 의해 서비스되어야 하는지를 선택하기 위한 중재 유닛의 예를 개략적으로 나타낸 것이다.

도 15는 도 12의 장치를 이용해서 캐시 액세스를 행하는 방법을 나타낸다.

도 16은 (i) 같은 메모리 페이지를 겨냥하는 대표적인 연속 액세스의 수, (ii) 다른 페이지에의 1개의 중간 액세스를 허용하는 동일한 페이지를 겨냥하는 대표적인 액세스의 수, 및 (iii) 동일한 캐시 라인을 겨냥하는 액세스의 수를 나타내는 시뮬레이션 결과를 나타낸 것이다.

도 17은 대표적인 고성능 비순차(out-of-order) 프로세서 캐시 인터페이스와, 메모리 어드레스 비트 필드용 명명 규칙(naming conventions)을 나타낸 것이다.

도 18은 본 기술을 이용해서 달성되는 성능의 시뮬레이션의 결과를 나타낸 것이다.

도 19는 본 기술을 이용해서 달성되는 정규화된 에너지 소모의 시뮬레이션의 결과를 나타낸 것이다.

도 20은 로드(loads) 중에서 L1 데이터를 공유하는 효과를 나타낸 것이다.

도 21은 웨이 결정을 이용하지 않는 시스템, 본 기술의 웨이 테이블을 이용하는 시스템, 및 8, 16 또는 32 엔트

리를 갖는 작은 웨이 테이블을 이용하는 시뮬레이션의 에너지 소모를 비교한 것이다.

도 22는 전체 액세스 카운트에 대한 웨이 판정에 의해 커버되는 L1 액세스의 비를 나타낸 것이다.

발명을 실시하기 위한 구체적인 내용

[0092] 도 1은 데이터를 처리하기 위한 프로세서(4)를 구비하는 데이터 처리장치(2)를 개략적으로 나타낸 것이다. 프로세서(4)를 대신하여 데이터를 저장하기 위한 캐시(6)가 제공되어 있다. 캐시(6)는 적어도 한 개의 추가 레벨 캐시(L2 캐시)를 포함하는 캐시 계층의 멤버인 레벨 1(L1) 캐시이다. L1 캐시(6)는 L2 캐시 인터페이스(8)를 통해서 L2 캐시 및 메모리(도 1에는 미도시)와 통신한다. 프로세서(4)가 필요로 하는 데이터가 L1 캐시(6) 내에 존재하지 않으면, L1 캐시(6)는 L2 캐시 또는 메모리로부터 데이터를 요구하고, 데이터를 캐시(6)의 라인에 할당한다. 캐시 간섭성 방법(coherency policy)(예를 들면, 라이트-백(write-back) 또는 라이트- 스루(write-through) 방법)을 이용해서 L1 캐시(6), L2 캐시 및 메모리 내의 데이터의 간섭성을 유지한다. 프로세서(4)에의 서비스된 캐시 액세스의 결과를 복귀하기 위한 버스(9)가 제공된다.

[0093] 프로세서(4)는 캐시(6)로부터 데이터를 액세스하기 위한 데이터 액세스 요구를 발행할 수 있다. 특별한 데이터 액세스 요구에 필요한 데이터의 가상 어드레스를 계산하기 위한 어드레스 계산 스테이지(10)가 제공된다. 이 예에 있어서의 L1 캐시(6)는 캐시에 저장된 데이터를 식별하기 위해서 물리 어드레스를 이용하는 물리적으로 인덱스된, 물리적으로 태그된(Physically indexed, physically tagged:PIPT) 캐시이다. L2 캐시(및 제공될 수 있는 어떤 추가 레벨 캐시) 및 메모리도 물리 어드레스를 이용한다. 변환 색인 버퍼(TLB)(12)는 어드레스 계산 스테이지(10)에 의해서 계산된 가상 어드레스를 캐시(6) 및 메모리가 이용하는 물리 어드레스로 변환하기 위해서 제공된다. 캐시(6)의 어느 캐시 웨이가 필요한 데이터를 저장하고 있는지를 나타내는 웨이 정보를 저장하기 위한 TLB(12)에 대응하는 웨이 테이블(14)도 제공된다. 이 웨이 테이블에 대해서는 이하에 좀더 상세히 설명한다.

[0094] 장치(2)는 또한 스토어 버터(SB)(15)와 머지 버퍼(MB)(16)도 구비한다. 프로세서(4)에 의해 투기적으로 발생되었던 스토어 요구들을 처리하기 위한 스토어 버퍼(15)가 사용된다. 성능을 향상시키기 위해서, 스토어 요구들은 스토어 요구가 행해져야 하는지 여부가 실제로 알려지기 전에 프로세서에 의해 투기적으로 발행되어도 된다. 예를 들면, 비순차 프로세서에서, 또는 분기 명령에 의하면, 이전의 명령이 완료했을 때까지 스토어 요구와 관련된 명령이 실제로 필요한지 여부가 알려져 있지 않지만, 성능을 향상시키기 위해서, 미리 명령을 실행할 수 있고, 그 후에 명령이 실제로 필요했다고 판명되는 경우에만, 그 명령의 결과를 수용할 수 있다. 로드(loads) 요구와 달리, 스토어 요구들은 캐시의 건축학적 상태를 변경할 것이고, 그래서 스토어 요구를 수용하지 않으면 투기적으로 발행된 스토어 요구를 실행하는 것을 바람직하지 않다. 따라서, 투기적으로 발행된 스토어 요구들은, 투기적으로 발행된 스토어 요구가 실제로 실행되어야 한다는 것이 알려질 때까지 스토어 버퍼(15) 내에 배치될 수 있다. 프로세서(4)가 명령의 투기적 실행을 허용하지 않으면, 스토어 버터(15)가 제외될 수 있다는 것을 알 수 있을 것이다.

[0095] 스토어 요구가 수용될 때, 머지 버퍼(16)에 전해진다. 다수의 수용된 스토어 요구가 캐시(6)의 같은 캐시 라인에 대응하여 펜딩 중에 있으면, 성능을 향상시키고 (예를 들면, L1 캐시에) 에너지 집중 메모리 액세스를 피하기 위해서, 머지 버터(16)가 이들 요구를 병합해서 하나의 동작으로 캐시(6)에 의해 실행될 수 있는 한 개의 요구를 형성할 수 있다. 머지 버퍼(16) 내의 병합된 요구는 병합된 데이터를 캐시(6)에 기록도록 캐시에 의해 서비스된다.

[0096] 도 2는 캐시(6)의 구조의 일례를 나타낸다. 캐시(6)는 몇 개의 캐시 뱅크(20)를 가지고 있다. 캐시(6)의 각 뱅크(20)는 다른 뱅크로부터 독립적으로 액세스됨으로써 병렬 액세스가 처리 사이클 내에서 상이한 뱅크에 대해서 이루어질 수 있다. 그렇지만, 각 뱅크에 대해서, 한 개의 캐시 라인만이 한 개의 사이클에서 액세스될 수 있다. 캐시(6)는 어느 뱅크(20)가 대응하는 데이터를 저장하는지 메모리 어드레스의 일부로부터 결정할 수 있고 - 예를 들면 2개의 뱅크로 최하위 비트(least significant bit)는, 데이터가 뱅크 0에 저장되어야 하는지 뱅크 1에 저장되어야 하는지를 식별할 수 있다.

[0097] 캐시(6)는 n-웨이 세트 결합 캐시이다. 이것은 각 캐시 뱅크가 n개의 가능한 위치를 제공하여 각 캐시 웨이(24)에서 한 개의 특정 어드레스에 대응하는 캐시 라인을 저장하는 것을 의미한다. 도 2의 예에 있어서, n=4이므로, 각 뱅크가 4 캐시 웨이(24)를 갖고, 각 캐시 웨이(24)는 태그 데이터(22) 및 데이터(23)로 이루어진 다수의 캐시 라인을 포함한다. 캐시의 태그 데이터(22)는 총괄하여 "태그 어레이"라고 지칭할 수 있다. 각 캐시 라인에 대해서는, 태그 데이터(22)가 대응하는 데이터(23)와 관련된 메모리 어드레스의 일부를 저장하여, 캐시(6)가 각 캐시 라인에 어느 데이터가 저장되는지를 식별하는 것을 허용한다. 특정 데이터 값이 저장될 수 있는 n개의 가

능한 위치가 있지만, 캐시만이 한 번에 하나의 데이터 값의 카피를 홀드하고, 다른 (n-1)개의 가능한 위치는 다른 데이터 값을 저장하기 위해서 이용가능하다. 도 2에 나타낸 캐시 구조는 단순한 일례이며, 한 개의 캐시 웨이를 각각 나타내는 n개의 캐시 뱅크를 갖는 캐시 등, 다른 구성이 가능하다는 것을 알 수 있을 것이다. 따라서, 본 기술은 상이한 캐시 구조에 적응될 수 있다.

[0098] 도 4의 하부 부분은 캐시(6)가 메모리 어드레스를 해석하는 방법의 일례를 나타낸 것이다. 이 어드레스는 태그부(50), 인덱스부(52) 및 라인 오프셋부(54)를 구비한다. 인덱스부(52)는 대응하는 데이터 값을 저장할 수 있는 캐시 웨이(24)의 각각 내의 위치를 식별한다. 데이터 값이 캐시 내에 저장될 때, 대응하는 어드레스의 태그부(50)는 데이터를 저장하는 캐시 라인에 대응하는 태그 데이터(22)로서 저장된다. 캐시로부터 데이터에 액세스할 때, 요청된 어드레스의 태그부(50)를, 인덱스부(52)에 의해 식별되는 각 웨이(24)에서의 캐시 라인에 대한 태그 데이터(22)와 비교할 수 있고, 매치가 있으면, 매칭 태그부에 대응하는 캐시 라인이 필요한 데이터를 저장한다. 태그 매치가 없는 경우에는, 데이터가 캐시 내에 저장되지 않으므로, 메모리로부터 또는 하부 레벨 캐시로부터 요청된다. 어드레스의 라인 오프셋부(54)는 필요한 데이터가 저장되는 캐시 라인 내의 위치를 식별한다.

[0099] 도 3은 다수의 페이지(30)로 분할된 메모리 어드레스 공간의 일례를 나타내고, 각 페이지는 다수의 메모리 위치(40)를 구비한다. 페이지(30)는 같은 가상-물리 어드레스 맵핑이 적용되는 메모리 위치의 단위이다. 도 4의 상부 부분은 메모리 어드레스를 TLB(12)로 해석하는 방법의 일례를 나타낸다. 각 메모리 어드레스는 페이지 ID부(44)와 페이지 오프셋부(46)를 구비한다. 페이지 ID부(44)는 어드레스를 포함하는 어드레스 공간의 페이지(30)를 식별한다. 페이지 ID부(44)는 가상 어드레스 및 그것의 대응하는 물리 어드레스에 대해서 서로 다를 수 있다. 따라서, TLB는 페이지 ID부(44)를 이용해서 가상 어드레스의 가상 페이지 ID를 대응하는 물리 어드레스의 물리적 페이지 ID로 변환하기 위한 대응하는 TLB 엔트리를 식별한다. 한편, 어드레스의 페이지 오프셋부(46)는 페이지(30) 내의 특정 메모리 위치(40)를 식별한다. 페이지 오프셋부(46)는 TLB에 의해 가상 어드레스로부터 물리 어드레스로 변환하는 중에는 그대로 남아 있다.

[0100] 도 4는 페이지 사이즈가 4KB이고, 캐시(6)가 64바이트 와이드 캐시 라인에서의 데이터의 32KB를 저장하는 4웨이 세트 관련 캐시인 특별한 예를 나타낸다. 상이한 페이지 사이즈 및 캐시 라인 사이즈와, 상이한 개수의 캐시 웨이를 사용해도 된다는 것을 알 수 있을 것이다. 이 경우, 도 4에 나타낸 예로부터 페이지 ID부(44), 페이지 오프셋부(46), 태그부(50), 인덱스부(52) 및 라인 오프셋부(54)의 비트의 개수가 변할 수 있다.

[0101] 캐시(6)는 2개의 캐시 액세스 모드, 즉 표준 캐시 액세스 모드 및 감소된 캐시 액세스 모드를 갖는다. 표준 액세스 모드에 있어서는, 상술한 바와 같이, 데이터 웨이(24)의 각각이 액세스되고, 태그 데이터(22)를 이용해서 어느 데이터 웨이가 필요한 데이터를 저장하는지를 식별한다.

[0102] 또한, 캐시(6)는 표준 액세스 모드와 비교해서 에너지를 절약하기 위해서 사용될 수 있는 감소된 캐시 액세스 모드도 갖는다. 웨이 테이블(14)은 주어진 메모리 어드레스와 관련된 데이터를 어느 캐시 웨이(24)가 저장하는지를 나타내는 웨이 정보를 포함하고 있다. 그러므로, 웨이 정보를 이용해서 필요한 데이터 웨이(24)를 식별할 수 있고, 그래서 다른 데이터 웨이(24) 또는 태그 어레이(22)에 액세스할 필요가 없다.

[0103] 도 5는 웨이 테이블의 구조를 나타낸다. 웨이 테이블(14)은 다수의 웨이 테이블 엔트리(60)를 갖는다. 각 웨이 테이블 엔트리(60)는 TLB(12)에서의 대응하는 TLB 엔트리(62)에 대응한다. 각 TLB 엔트리(62)는 메모리 위치의 관련된 페이지(30)에 대한 가상-물리적 어드레스 맵핑을 나타낸다. 예를 들면, 도 5에 나타낸 바와 같이, TLB 엔트리 #0은 제1 페이지 30-1에 대한 가상-물리 어드레스 맵핑을 나타내고, TLB 엔트리 #1은 관련된 페이지 30-1에 대한 가상-물리 어드레스 맵핑을 나타낸다.

[0104] 웨이 테이블(14)의 각 웨이 테이블 엔트리(60)는 대응하는 TLB 엔트리(62)와 관련된 페이지 내의 각 메모리 위치에 대한 웨이 정보를 저장한다. 이 웨이 정보는, 만약 있다면, 대응하는 메모리 위치와 관련된 데이터를 어느 캐시 웨이(24)가 저장하는지를 나타낸다. 예를 들면, 웨이 테이블 엔트리 #0은 페이지 30-0 내의 메모리 위치의 각각에 대한 웨이 정보를 식별하고, 웨이 테이블 엔트리 #1은 페이지 30-1 내의 각 메모리 위치에 대한 웨이 정보를 식별한다. 웨이 테이블(14) 및 TLB(12)의 사이즈는 제한되어 있어, 페이지의 서브세트만이 주어진 시간에 해당 웨이 테이블 엔트리(60)와 TLB 엔트리(62)를 가져도 된다. 주어진 페이지에 대해서 새로운 TLB 엔트리(62)와 웨이 테이블 엔트리(60)가 필요하면, 기존의 TLB 엔트(62)와 웨이 테이블 엔트리(60)가 새로운 엔트리에 대한 웨이를 만들기 위해서 축출되어도 된다.

[0105] 웨이 테이블 엔트리(60)는 또한 해당 페이지 내의 각 메모리 위치와 관련된 데이터가 캐시 내에 저장되어 있는지 아닌지를 나타내는 유효 정보도 유지한다. 도 5는 각 웨이 테이블 엔트리(60) 내에서 유효 정보 및 웨이 정

보가 표현될 수 있는 방법의 3가지의 예를 나타낸다.

[0106] 첫 번째 예(A)에 있어서, 웨이 테이블 엔트리(60)는 그 웨이 테이블 엔트리(60)에 대응하는 페이지(30) 내의 각 메모리 위치(라인)에 대한 유효 필드(70) 및 웨이 필드(72)를 포함한다. 이 예에 있어서, 유효 필드(70)는 해당 메모리 위치로부터의 데이터가 캐시(6) 내에 저장되면 제1 값(예를 들면, 1의 비트값)과, 해당 메모리 위치로부터의 데이터가 캐시(6) 내에 저장되어 있지 않는다면 제2 값(예를 들면, 0의 비트값)을 갖는 단 한 개의 비트만을 구비한다. 웨이 필드(72)는 해당 메모리 위치로부터의 데이터를 4개의 데이터 웨이(24) 중의 어느 것이 저장하는지를 나타내는 2비트 값을 구비한다.

[0107] 두 번째 예(B)에 있어서, 유효 정보 및 웨이 정보는 단 한 개의 필드(74)와 결합되고, 특정 메모리 위치로부터의 데이터가 배치될 수 있는 웨이(24)의 개수가 1개 감소된다. 2비트 결합된 유효/웨이 필드(74)는 대응하는 페이지(30) 내의 각 메모리 위치에 대해서 제공된다. 결합된 필드(74)에 대한 4개의 전위(예를 들면, 0b00) 중의 하나를 이용해서 데이터가 캐시(즉, 유효 정보)에 저장되지 않는다는 것을 나타낸다. 다른 3개의 전위값(예를 들면, 0b01, 0b10, 0b11)을 캐시 웨이(24) 중의 3개에 할당해서, 그 캐시 웨이(즉, 웨이 정보)에 데이터가 저장되는 것을 나타낸다. 따라서, 대응하는 메모리 위치와 관련된 데이터는 웨이 정보의 3개의 전위값으로 나타내지 않은 캐시 웨이의 네 번째에는 배치되지 않아도 된다. 이 접근법은 예(A)에 비해서 각 웨이 테이블 엔트리에 필요한 비트의 개수를 줄이고, 더 적은 수의 비트가 있기 때문에, 웨이 테이블 액세스당 에너지 소모가 감소된다. 캐시 미스율(miss rate)에 대한 이 접근법의 영향은, 프로세서(4)에 의해 실행된 대부분의 스레드(threads)가 L1 캐시(6)의 모든 4개의 웨이를 이용하지 않기 때문에 최소이다. 또한, 예(B)에 있어서의 웨이 테이블 표현에 의해서도, 캐시는 여전히 4웨이 세트 관련 캐시로서 실현될 수 있다. 특히, 4개의 뱅크(20)를 가진 캐시는 개개의 뱅크 내의 라인 0-3과 같은 뱅크 내의 라인 0,4,8,..,60을 홀드해도 된다. 라인 0-3에 대해서 웨이 0을 무료라고 간주하고, 라인 4-7에 대해서 웨이 1을 무료라고 간주함으로써, 몇몇 형태의 특별한 장소를 나타내는 종래의 스레드(thread)는 아마도 캐시를 4웨이 세트 관련이라고 인식할 것이다.

[0108] 도 5는 또한 다수의 연속 메모리 위치(라인)에 대하여 유효 및 웨이 정보를 함께 그룹화하는 각 웨이 테이블 엔트리 내의 웨이 및 유효 정보를 나타내는 세 번째 예(C)를 나타낸다. 예를 들면, 4개의 메모리 위치(0..3,4..7, 등)의 각 그룹은, 그 메모리 위치와 관련된 데이터가 캐시에 저장되어 있는지를 식별하는, 그룹 내의 각 메모리 위치에 대해서 1비트를 포함하는 대응하는 유효 필드(76)를 갖는다. 메모리 위치의 각 그룹은 또한 그 그룹 내부의 메모리 위치들과 관련된 데이터를 4 데이터 웨이(24)의 어느 것이 저장하는지를 식별하는 2비트 값을 포함하는 웨이 필드(78)도 갖는다. 이 예에 있어서, 캐시(6)는 동일한 캐시 웨이(24) 내의 같은 그룹의 각 메모리 위치와 관련된 데이터를 저장하도록 제한된다. 이 접근법은 각 웨이 테이블 엔트리(60)에 필요한 비트의 개수를 상당히 줄이므로 웨이 테이블 액세스당 에너지 소모도 줄인다.

[0109] 도 5에 나타낸 예는 단지 예시이며 웨이 테이블 엔트리 내에서 유효 정보 및 웨이 정보를 표현할 수 있는 많은 상이한 웨이가 있다는 것을 인식할 것이다. 어느 표현을 사용하든, 도 5에 나타낸 바와 같이 TLB(12)와 웨이 테이블(14)을 결합으로써, 캐시 액세스를 행할 때 대응하는 엔트리에 대해서 TLB(12) 및 웨이 테이블(14) 양자를 검색할 필요가 없다. 대신, 주어진 메모리 액세스 요구에 대해서, TLB(12)와 웨이 테이블(14) 중 하나를 검색해서 메모리 액세스 요구의 가상 어드레스에 대응하는 엔트리를 식별하고, 다른 테이블을 제어해서 추가 검색을 행하지 않고 대응하는 엔트리에 액세스할 수 있다. 예를 들면, 도 5에 나타낸 바와 같이, TLB(12)는 필요한 가상 어드레스의 가상 페이지 ID부(44)를 각 TLB 엔트리(62) 내의 가상 페이지 ID와 매칭시킴으로써 대응하는 TLB 엔트리(62)를 검색할 수 있고, 일단 타겟 TLB 엔트리가 식별되었다면, TLB는 웨이 테이블(14)을 제어해서 대응하는 웨이 테이블 엔트리(60)를 복귀시킨다. 2개의 테이블(12, 14)의 검색의 중복을 피함으로써, 에너지 소모를 줄일 수 있다.

[0110] 도 5의 예에 있어서, TLB(12) 및 웨이 테이블(14)이 개별적으로 도시되어 있다. 예를 들면, TLB(12) 및 웨이 테이블(14)은 별개의 메모리로서 제공될 수 있다. TLB(12)가 필요한 TLB 엔트리(62)를 배치했을 경우, TLB(12)는 캐시(6)에 액세스하기 위해서 사용되는 대응하는 웨이 테이블 엔트리를 식별하는 웨이 테이블(14)에 인덱스값을 제공할 수 있다.

[0111] 한편, 다수의 엔트리를 갖는 공통 메모리로서 TLB(12) 및 웨이 테이블(14)를 구현하는 것도 가능하다. 공통 메모리의 각 엔트리는 대응하는 TLB 엔트리(62)와 대응하는 웨이 테이블 엔트리(60) 모두를 구비할 수 있다. 그러한 실시예에 있어서, 공통 메모리는 현재의 데이터 액세스 요구에 대한 어드레스의 가상 페이지 ID에 대응하는 가상 페이지 ID(44)를 갖는 엔트리를 검색하고, 그 후에 그 공통 메모리 엔트리로부터 대응하는 웨이 및 유효 정보와 물리 페이지 ID를 복귀시킬 것이다.

[0112] 도 6은 웨이 테이블(14)에 저장된 정보에 의거해서 감소된 캐시 액세스를 행하는 방법을 나타낸다. 스텝 100에서는, 프로세서(4)는 데이터 액세스 요구를 발행하고, 어드레스 계산 스테이지(10)는 데이터 액세스 요구에 대한 가상 어드레스를 계산한다. 스텝 102에서는, TLB(12)는 대응하는 TLB 엔트리를 검색한다(즉, 가상 페이지 ID가 데이터 액세스 요구에서 가상 어드레스의 가상 페이지 ID와 매치하는 TLB 엔트리 12). TLB(12)는 TLB 엔트리로 지정된 것처럼, 가상 페이지 ID에 대응하는 물리 페이지 ID(44)를 복귀시킨다. 스텝 104에서는, 웨이 테이블(14)은 TLB(12)에 의해 액세스된 TLB 엔트리(62)에 대응하는 웨이 테이블 엔트리(60)에 액세스한다. 스텝 S106에서는, 캐시(6)는 웨이 테이블 엔트리 내의 웨이 정보에 의거해서 캐시 액세스를 행한다. 웨이 테이블 엔트리의 유효 정보가, 캐시에 데이터가 저장되어 있다는 것을 나타내면, 캐시는 태그 어레이(22) 또는 다른 캐시 웨이(24)에 액세스하는 일없이, 웨이 정보에 나타낸 캐시 웨이(24)만이 액세스되는 감소된 캐시 액세스를 행한다. 한편, 유효 정보가 캐시에 데이터가 저장되어 있지 않는다는 것을 나타내면, 캐시는 모든 데이터 웨이(24)가 액세스되는 표준 캐시 액세스를 행하고, 태그 어레이(22)를 이용해서 어느 데이터 웨이(24)가, 만약 있다면, 필요한 데이터를 저장하는지를 식별한다.

[0113] 이렇게 해서, 웨이 테이블은 가능하면 감소된 캐시 액세스가 행해지도록 허용하고, 표준 캐시 액세스는 웨이 테이블이 필요한 정보를 제공하지 않을 때 사용된다. 표준 액세스 모드가 여전히 필요함에 따라, 태그 어레이(22)가 감소된 액세스 모드에 필요하지 않다는 사실에도 불구하고 태그 어레이(22)가 여전히 제공된다.

[0114] 도 7 및 8에 나타낸 바와 같이, TLB(12) 및 웨이 테이블(14)은 TLB 및 웨이 테이블의 다수의 레벨로 이루어진 계층 구조를 가질 수 있다. 이 장치는 마이크로-TLB(uTLB) 12-u 및 추가 레벨 TLB 12-f를 구비하고, 웨이 테이블은 마이크로-웨이 테이블(uWT) 14-u 및 추가 레벨 웨이 테이블(WT) 14-f를 구비한다. 마이크로-TLB 12-u 및 마이크로-웨이 테이블 14-u는, 추가 레벨 TLB 12-f 또는 추가 레벨 웨이 테이블 14-f보다 빨리 에너지 효율적으로 TLB 엔트리 또는 웨이 테이블 엔트리를 검색할 수 있는 작고 빠른 액세스, 구조이다. 한편, 추가 레벨 TLB 및 추가 레벨 웨이 테이블은 다수의 TLB 엔트리 및 웨이 테이블 엔트리를 허용하지만, 에너지 비용이 증가한다. 따라서, 데이터 액세스 요구가 대응하는 마이크로-TLB 엔트리 및 마이크로-웨이 테이블 엔트리를 가지고 있지 않는 페이지에 대응하는 어드레스를 지정하면, TLB 및 웨이 테이블 엔트리가 추가 레벨 TLB/웨이 테이블로부터 마이크로-TLB 및 마이크로-웨이 테이블에 반입될 수 있다. 마이크로-TLB 12-u 및 마이크로-웨이 테이블 14-u가 이미 가득 차있으면, 또 다른 엔트리(예를 들면, 최소 최근에 사용된 엔트리 또는 최소 최근에 할당된 엔트리)가 각 테이블로부터 축출될 수 있다. 그러한 계층 테이블 구조를 제공함으로써, 멀리-레벨 캐시 계층과 같은 방식으로, 에너지 효율, 빠른 액세스 및 증가된 TLB/웨이 테이블 커버리지(coverage) 간의 밸런스 향상을 달성할 수 있다.

[0115] 웨이 테이블 정보를 유지하기 위해서, 웨이 정보는 캐시(6)에서의 데이터의 할당을 변경하는 동작에 응답해서 갱신된다. 도 7에 나타낸 바와 같이, 웨이 정보는 캐시(6)의 캐시 라인에 데이터가 할당될 때(L1 line fill) 혹은 캐시(6)로부터 데이터가 축출될 때 갱신될 수 있다. 캐시 라인필(cache linefill) 또는 캐시 축출이 발생하면, 캐시(6)는 할당된 또는 축출된 데이터와 관련된 물리 어드레스를 TLB(12) 및 웨이 테이블(14)에 제공한다. TLB는 물리 페이지 ID에 의거해서 검색가능하고, 캐시(6)에 의해 식별된 물리 어드레스에 대응하는 TLB 엔트리를 식별한다. 그리고나서 TLB는 웨이 테이블(14)을 제어해서 대응하는 웨이 테이블 엔트리를 갱신한다. 웨이 테이블(14)은 캐시(6)에 의해 제공된 물리 어드레스의 페이지 오프셋부의 일부에 의거해서 갱신되는 웨이/유효 필드를 식별한다.

[0116] 캐시 라인필에 있어서, 마이크로 웨이 테이블 14-u 또는 웨이 테이블 14-f 내의 웨이 테이블 엔트리(60)의 대응하는 필드를 갱신함으로써, 유효 정보는 지금 캐시 내에 정보가 저장되어 있다는 것을 나타내고, 웨이 정보는 지금 어느 웨이가 데이터를 저장하는지를 나타낸다. 한편, 캐시로부터 데이터가 축출되면, 유효 정보를 갱신해서, 캐시에 더 이상 데이터가 저장되지 않는다는 것을 나타내고, 또 그 메모리 위치에 대응하는 웨이 정보가 다음의 캐시 액세스에 사용되지 않아야 한다는 것을 나타낸다.

[0117] 마이크로-웨이 테이블 엔트리가 갱신되면, 추가 레벨 웨이 테이블 14-f 내의 대응하는 엔트리는, 마이크로-웨이 테이블 14-u로부터 대응하는 마이크로-웨이 테이블 엔트리가 축출될 때까지 갱신되지 않는다. 추가-레벨 웨이 테이블 14-f의 갱신을 연기하면 에너지가 절약되는데, 그 이유는 마이크로-웨이 테이블 14-u에서의 엔트리가 축출되기 전에 여러 번 갱신되어도, 추가-레벨 웨이 테이블 엔트리만 한 번 갱신되어야 하기 때문이다.

[0118] 추가-레벨 TLB 12-f 및 추가-레벨 웨이 테이블 14-f의 용량은 유한하므로, 이들 테이블은 어드레스 공간의 모든 페이지에 대응하는 엔트리를 저장할 수 없다. 따라서, 필요한 경우에는 TLB 또는 웨이 테이블로부터의 엔트리가, 상이한 페이지에 대응하는 또 다른 엔트리에 대한 공간을 만들기 위해서 축출될 수 있다. 대응하는

TLB 엔트리는 메모리와 관련된 페이지 테이블로부터 페치(fetch)될 것이고, 웨이 테이블 엔트리는, 대응하는 페이지 내의 모든 메모리 어드레스와 관련된 데이터가 캐시에 저장되지 않는다는 것을 유효 정보가 나타내는 리셋 상태에 있는 웨이 테이블에 할당될 것이다.

[0119] 따라서, 데이터가 캐시에 남아 있는 동안에도 캐시에 저장된 데이터가, 추가 웨이 테이블 14-f로부터 축출된 그것의 대응하는 웨이 테이블 엔트리를 갖는 것이 가능하다. 대응하는 웨이 테이블 엔트리가 웨이 테이블 14-f에 다시 반입될 때는, 그것은 대응하는 데이터가 캐시 내에 저장되어 있더라도, 처음에는 리셋 상태에 있을 것이다. 따라서, 캐시(6)는 유효 정보가 나타내는 데이터가 캐시에 저장되어 있지 않고, 실제로도 캐시에 저장되어 있지 않다고 가정할 수 없다. 그래서, 유효 정보가, 캐시에 데이터가 저장되어 있지 않다는 것을 나타내어도, 캐시는 실제로 캐시에 데이터가 저장되는지를 식별하기 위해서 태그 어레이(22)를 이용해서 통상의 캐시 액세스를 여전히 행할 것이다. 도 7의 배치에 있어서, 웨이 정보가 단지 캐시 라인필 또는 캐시 축출시에 갱신되는 곳에서, 웨이 정보가 이미 캐시에 존재하는 데이터에 대한 웨이 테이블로 다시 반입되면, 웨이 정보를 갱신하는 것이 가능하지 않을 것이다. 이것은 캐시 액세스가 비교적 국지적이면 문제가 되지 않을 수도 있으므로, 몇 번 특정 값에 액세스한 후에는, 그 값이 다시 요구될 것 같지는 않다(예를 들면, 스트리밍 어플리케이션(streaming application)이 데이터를 캐시로 로드하고, 데이터를 재사용하며, 그 후에 다른 데이터를 이용해서 이동하므로, 첫 번째로 로드된 데이터는 다시 요구되지 않을 수도 있다).

[0120] 그렇지만, 다른 어플리케이션에 대해서는, 캐시된 데이터가 일정 시간의 기간 동안 간격을 두고 액세스되는 것이 가능할 수도 있다. 이 경우, 도 8에 나타낸 갱신 방식을 이용하는 것이 유용할 수 있는데, 여기에서는 도 7에서와 같이 라인 필 및 캐시 축출시에 웨이 정보를 갱신하는 것뿐 아니라, 데이터 액세스에 의해 요구된 데이터가 캐시(6)("웨이 테이블 미스") 내에 있지 않다는 것을 대응하는 웨이 테이블 엔트리가 나타낼 때 웨이 정보도 갱신하지만, 태그 어레이(22)의 체크시에는, 캐시(6)에 데이터가 있다("캐시 히트")는 것을 알 수 있다. 이 시나리오는 대응하는 웨이 테이블 엔트리가 최근에 웨이 테이블(14)에 복원되었던 페이지의 각 메모리 위치에 제1 액세스시에 발생할 가능성이 있다. 따라서, 웨이 테이블 미스/캐시 히트가 검출될 때 캐시가 웨이 테이블에 신호를 보낼 수 있고 실제로 어느 웨이가 데이터를 저장하는지를 나타낼 수 있다. 이에 응답하여, 웨이 테이블이 대응하는 웨이 테이블 엔트리를 갱신함으로써, 유효 정보가 지금 캐시에 데이터가 저장된다는 것을 나타내고, 웨이 정보가 지금 어느 웨이가 데이터를 저장하는지를 나타낸다. 같은 웨이 테이블 엔트리는, 그 웨이 테이블 엔트리에 대응하는 페이지 내의 상이한 어드레스에 대해서 웨이 테이블 미스/캐시 히트가 검출될 때마다 여러 번 갱신될 필요가 있다.

[0121] 웨이 테이블 미스/캐시 히트에 응답해서 어느 웨이 테이블 엔트리(14)가 갱신되어야 하는지를 식별하기 위해서, 웨이 테이블(14) 또는 TLB(12)가 데이터의 어드레스에 의거해서 검색을 행할 수 있다. 그렇지만, 이것은 여분의 에너지를 소모할 것이다. 따라서, 추가 검색을 행해야 하는 것을 피하기 위해서, 도 8의 웨이 테이블에는 가장 최근에 액세스된 웨이 테이블 엔트리의 인덱스값을 저장하는 저장 소자(110)가 설치된다. 그러므로, 캐시가 웨이 테이블 미스/캐시 히트에 응답해서 갱신이 요구된다는 신호를 보낼 때, 웨이 테이블(14)은 액세스된 데이터의 어드레스에 의거해서 테이블을 검색할 필요없이, 최종 엔트리 저장소자(110)에 표시되는 엔트리를 간단히 갱신할 수 있다. 저장소자(110)에 의거한 갱신은 단지 마이크로-웨이 테이블 14-u에 대해서만 요구될 수 있는데, 그 이유는 추가-레벨 웨이 테이블 14-f로부터의 엔트리에 액세스할 때, 그 엔트리가 마이크로-웨이 테이블 14-u에 할당될 수 있고, 웨이 정보 갱신이 추가-레벨 웨이 테이블 14-f보다 좀더 효율적으로 마이크로-웨이 테이블 14-u에서 행해질 수 있다. 대응하는 추가-레벨 웨이 테이블 14-f의 갱신은, 엔트리가 마이크로-웨이 테이블 14-u로부터 축출될 때까지 연기될 수 있다.

[0122] 선택적으로, 웨이 테이블(14)은 또한 원하는 데이터에 대해서 L1 캐시 미스가 발생하는지를 예측하기 위해서 사용될 수 있다. 주어진 메모리 어드레스에 대한 유효 정보가 캐시에 대응하는 데이터가 저장되어 있지 않다는 것을 나타내면, 이것은 L1 캐시 미스로서 해석될 수 있고, L2 캐시 액세스 또는 메모리 액세스는, L1 캐시(6)의 데이터 웨이(24) 또는 태그 어레이(22)에 액세스할 필요 없이, 다른 캐시 액세스 서비스에 사용하기 위해 이들 데이터 웨이(24) 및 태그 어레이(22)를 자유롭게 하는 일없이, 또는 에너지를 절약하기 위해서 데이터 웨이(24) 및 태그 어레이(22)가 유휴 상태로 남아 있도록 허용하는 일없이, 시작될 수 있다. 그렇지만, 이 접근법은 유효 정보가 올바르다고 보장되는 것을 요구할 것이다. 상술한 바와 같이, 이것은 대응하는 데이터가 캐시에 남아 있는 동안 웨이 테이블 엔트리가 웨이 테이블(14)로부터 축출되고, 나중에 웨이 테이블(14)에 복원되는 경우가 되지 않을 수 있다. 이 문제를 해결하기 위해서, 축출된 웨이 테이블 엔트리가 별개의 메모리에 또는 부가적인 추가-레벨 웨이 테이블에 저장될 수 있고, 리셋 상태에서의 웨이 테이블에 복원되었던 엔트리는 L1 캐시 미스 예측이 행해지지 않아도 되는 엔트리로서 표시될 수 있다. 그렇지만, 이것은 축출된 엔트리 또는 부가적인 "미스

예측(miss prediction)" 비트를 저장하기 위한 추가 회로를 필요로 할 것이고, 이 추가 회로에 의해 소모되는 에너지는, 대부분의 L1 캐시가 매우 낮은 미스율(miss rates)을 나타내기 때문에 정당화되지 않으므로, L1 미스 예측의 이점은 자주 발생하지 않을 수도 있다. 그러므로, 유효 정보에 의거한 미스 예측은 선택사항이다.

[0123] 도 9 내지 11은 캐시 액세스에 의거하여 웨이 테이블 엔트리를 갱신하는 방법을 나타낸다. 도 9는 캐시 라인필에 응답해서 웨이 테이블을 갱신하는 것을 나타낸다. 스텝 120에서는, 캐시 라인필이 검출된다. 이 캐시는 할당된 데이터와 관련된 물리 어드레스를 TLB(12) 및 웨이 테이블(14)에 제공하고, 또한 캐시 웨이가 할당되어 있는 표시에 데이터를 제공한다. TLB(12) 또는 웨이 테이블은 물리 페이지 ID(44)에 의거해 갱신되는 웨이 테이블 엔트리를 식별한다. 스텝 122에서는, 웨이 테이블(14)은 어드레스의 페이지 오프셋부(46)의 일부분을 이용하여 데이터에 대응하는 웨이 테이블 엔트리의 유효 및 웨이 정보 필드를 식별하고, 유효 정보를 갱신하여 캐시에 데이터가 저장되어 있다는 것을 나타내며, 웨이 정보를 갱신하여 어느 캐시 웨이(24)가 데이터를 저장하는지를 나타낸다.

[0124] 도 10은 캐시 라인 축출에 응답해서 웨이 테이블을 갱신하는 방법을 나타낸다. 스텝 130에서는, 캐시 라인으로부터의 데이터가 캐시로부터 축출되고, 메모리 계층의 하부 레벨(예를 들면, L2 캐시, L3 캐시, 또는 메모리)로 복귀된다. 캐시는 TLB(12) 및 웨이 테이블(14)에 축출된 데이터와 관련된 물리 어드레스를 제공한다. TLB 또는 웨이 테이블은 어드레스의 물리 페이지 ID(44)를 이용해서 갱신되어야 하는 웨이 테이블 엔트리를 배치한다. 웨이 테이블(14)은 어드레스의 페이지 오프셋부(46)를 이용해서 웨이 내에 대응하는 유효 필드를 배치한다. 스텝 132에서는, 웨이 테이블(14)은 유효 필드를 갱신해서 대응하는 데이터가 캐시에 저장되어 있지 않다는 것을 나타낸다.

[0125] 도 11은 웨이 테이블 미스/캐시 히트에 응답해서 웨이 테이블 엔트리를 갱신하는 방법을 나타낸다. 스텝 140에서 이 시나리오를 검출할 때, 캐시는 웨이 테이블 엔트리가 갱신되어야 하고, 또 액세스된 데이터를 저장하는 웨이(24)를 식별하는 것을 웨이 테이블(14)에 표시한다. 스텝 142에서는, 가장 최근에 액세스된 웨이 테이블 엔트리의 인덱스는 저장소자(110)로부터 판독된다. 스텝 144에서는, 웨이 테이블(14)이 저장소자에 나타낸 마이크로-웨이 테이블 엔트리를 갱신함으로써, 유효 정보는, 데이터가 캐시되고, 웨이 정보가 데이터를 저장하는 웨이를 표시한다는 것을 현재 나타낸다. 추가-레벨 웨이 테이블 내의 대응하는 웨이 테이블 엔트리는, 마이크로-웨이 테이블 엔트리가 나중에 마이크로-웨이 테이블 14-u로부터 축출될 때까지 갱신되지 않는다.

[0126] 후술하는 바와 같이, 캐시(6)는 같은 처리 사이클 내에서 상이한 뱅크에 지시된 다수의 요구를 서비스할 수 있다. 그렇다면, 같은 처리 사이클에서, 다수의 웨이 테이블 엔트리 또는 같은 웨이 엔트리 내의 다수의 필드를 갱신할 필요가 있다. 그러므로, 도 9 내지 11의 방법은 갱신되어야 하는 각 정보에 대해서 반복될 수 있다.

[0127] 요약하면, 웨이 정보를 저장하는 웨이 테이블을 제공하는 것에 의해, 전력 소모는 웨이 정보에 나타낸 웨이에만 액세스하고 다른 웨이(24) 또는 태그 어레이(22)에 액세스하지 않음으로써 감소될 수 있다. 메모리의 페이지에 대응하는 웨이 정보를 함께 그룹화하는 웨이 테이블 엔트리를 제공하고, 각 웨이 테이블 엔트리를 대응하는 TLB 엔트리에 연결함으로써, 이들 테이블 중의 하나의 단일 검색이 대응하는 TLB 엔트리 및 웨이 테이블 엔트리 모두를 배치하는 것이 충분하기 때문에 TLB 및 웨이 테이블을 검색하는 것이 더 효율적이게 된다.

[0128] 도 12는 특히 장치(2)가 병렬로 다수의 데이터 액세스 요구를 처리할 수 있는 방법을 좀더 상세히 나타내는 장치(2)를 도시한 것이다. 프로세서(4)는 캐시(6)에 데이터 액세스 요구를 발행할 때를 결정하기 위한 디스패치 스테이지(dispatch stage)(200)와 이슈(issue) 스테이지(202)를 갖는다. 로드 큐(load queue)(204)는 또한, 펜딩 로드 요구를 저장하기 위해서도 사용될 수 있다. 이 프로세서(4)는 프로세서(4)에 의해 발행된 데이터 액세스 요구와 관련된 가상 어드레스를 계산하기 위한 어드레스 계산 스테이지 10-0 내지 10-3을 구비한다.

[0129] 입력 벡터(210)는 캐시에 의해서 서비스하는 것을 기다리는 펜딩 요구들을 버퍼링하기 위해서 제공된다. 아비터(arbiter)(230)는 주어진 처리 사이클에서 입력 벡터(210)에 나타낸 펜딩 요구들 중의 어느 것이 서비스되어야 하는지를 선택하기 위해서 제공된다. 어드레스 계산 스테이지 10-0 내지 10-3으로부터의 로드 요구들은 캐시에 의해 서비스하는 것을 기다리기 위해서 입력 벡터(210)에 직접 배치된다. 한편, 투기적으로 발행된 스토어 요구들은, 스토어 요구들을 수행한다는 확인을 기다리기 위해서 스토어 버퍼(15)에 전달된다. 일단 저지르면, 스토어 요구들은 캐시(6)에 의해서 서비스하는 것을 기다리기 위해서 머지(merge) 버터(16)에 전송된다. 머지 버퍼(16)는 스토어 버퍼(15)로부터 다수의 스토어 요구들을 병합하여 머지 버퍼 엔트리를 형성해도 된다. 머지 버퍼 엔트리는 아비터(230)에 의한 선택을 기다리기 위해서 입력 벡터(210)에 제공된다.

[0130] 대표적으로, 머지 버퍼(16)는, 입력 벡터(210)에 머지 버퍼 엔트리를 발행하지 않으면 새로운 요구를 받을 때까

지 그렇지 않은 넘칠 때까지 입력 벡터(210)에 머지 버퍼 엔트리를 보내는 것을 연기할 것이다. 즉, 머지 버퍼(16)가 어떤 현재의 머지 버퍼 엔트리에 의해 표현되지 않는 영역을 어드레스하는 스토어 버퍼(15)로부터 스토어 요구를 수신하면, 그리고 어떤 자유로운 머지 버퍼 엔트리도 없으면, 머지 버퍼(16)는 현재의 머지 버퍼 엔트리를 축출할 것이고 그것을 입력 벡터(210)에 보낼 것이다. 가능한 한 머지 버퍼 엔트리의 축출을 지연시킴으로써, 이것은 다수의 스토어 요구들이 같은 사이클에서 병합되어 처리될 수 있는 기능성을 증가시킴으로써, 캐시를 좀더 효율적으로 이용할 수 있다.

[0131] 도 13은 입력 벡터의 예를 나타낸다. 입력 벡터(210)는 펜딩 액세스 요구를 수신하기 위한 다수의 벡터 엔트리(212)를 구비한다. 벡터 엔트리(212)는 소정의 우선순위를 가지므로, 아비터(230)가 예를 들면, 엔트리 #7 내의 데이터 액세스 요구보다도 더 높은 확률을 갖는 엔트리 #0으로부터 데이터 액세스 요구를 선택할 수 있다. 이 예에 있어서, 머지 버퍼(16)로부터의 머지 버퍼 엔트리는 벡터 엔트리 #0에 배치되고, 어드레스 계산 스테이지 10-0 내지 10-3으로부터 수신된 판독 액세스 요구들이 벡터 엔트리 #4 내지 #7에 배치되며, 3개의 벡터 엔트리 #1 내지 #3은 이전의 클럭 사이클로부터 처리되지 않았던 남아 있는 펜딩 액세스 요구들을 저장하기 위해서 사용된다. 그러므로, 도 13의 예에 있어서, 머지 버퍼 엔트리(MBE)는 이전의 사이클로부터 오래된 판독 액세스 요구보다 먼저 우선순위가 정해지고, 오래된 판독 액세스 요구들은 좀더 새로운 판독 액세스 요구보다 먼저 우선순위가 정해진다. 다른 예에 있어서는, 우선 순위는, 예를 들면 오래된 판독 액세스 요구가 머지 버퍼 엔트리보다 더 높은 우선순위를 갖도록 변할 수 있다.

[0132] 각 벡터 엔트리는 이 엔트리가 유효 요구를 나타내는지 여부를 표시하는 유효 필드(232)를 구비한다. 예를 들면, 어드레스 계산 스테이지 10-0 내지 10-3의 몇몇은 주어진 사이클에서 활성화하지 않거나, 펜딩 머지 버퍼 엔트리 #0가 없거나 이전의 사이클로부터 남은 충분한 요구가 없어서, 백터 엔트리 중의 몇몇은 유효하지 않을 수도 있다. 각 벡터 엔트리는 또한 데이터 액세스 요구와 관련된 가상 어드레스를 저장하기 위한 어드레스 필드(234)와, 엔트리의 타입(예를 들면, 엔트리가 로드(load) 요구 또는 머지 버퍼 엔트리에 대응하는지 여부)를 나타내는 타입 필드(236)도 갖는다. 각 벡터 엔트리는 또한 같은 타입의 상이한 액세스 요구들을 구별하는 연령(age) 식별자를 저장하는 연령 식별자 필드(238)(예를 들면, 리오더(reorder) 버퍼 식별자 또는 ROB ID 필드)도 갖는다. 연령 식별자는, 예를 들면, 한 개의 요구가 또 다른 요구보다 더 오래되었는지 여부의 판정을 허용하기 위해서, 순차적으로 할당되어서 사용될 수 있다. 다른 정보 또는 다른 연령 식별자가 벡터 엔트리에도 저장될 수 있다는 것을 알 수 있을 것이다.

[0133] 입력 버퍼(210)는 모든 벡터 엔트리(212)에 대해서 저장 위치를 구비할 필요가 없다는 점에 유념한다. 예를 들면, 머지 버퍼 엔트리 및 새로운 요구들은 어드레스 계산 스테이지(10) 및 머지 버퍼(16)로부터 수신된 와이어(wires)로부터 직접 판독될 수 있으므로, 입력 버퍼(210)는 실제로 이들 요구에 대해서 어떤 데이터도 저장할 필요가 없다. 한편, 이전의 사이클에서 서비스되지 않았던 좀더 오래된 요구들을 저장하기 위한 스토리지가 제공될 수 있다. 또한, 다른 예에 있어서, 모든 요구들이 입력 버퍼(210) 내에 설치된 스토리지에서 버퍼링될 수 있다.

[0134] 도 12로 되돌아가서, 각 사이클에 있어서, 입력 벡터(210)의 제1 유효 벡터 엔트리의 가상 어드레스의 페이지 ID가 물리 페이지 ID로 변환되도록 TLB(12)에 제공된다. 그러므로, 벡터 엔트리 #0에 펜딩 머지 버퍼 엔트리가 있으면, 머지 버퍼 엔트리의 가상 페이지 ID가 TLB(12)에 제공되지만, 펜딩 머지 버퍼 엔트리가 없으면, 유효 벡터 엔트리가 발견될 때까지 그 엔트리가 유효 요구를 포함하는 경우에 엔트리 #1로부터의 어드레스가 제공된다. 제1 유효 벡터 엔트리에 저장된 요구는 "주요한(primary)" 요구라고 칭할 수 있다. TLB(120)는 주요한 요구의 가상 페이지 ID를 물리 페이지 ID로 변환하고, 물리 페이지 ID를 중재유닛(230)에 제공한다. 또한, TLB는 웨이 테이블(14)을 제어해서 상기 설명한 방식으로 대응하는 웨이 테이블을 검색하고, 웨이 테이블 엔트리를 중재유닛(30)에 제공한다. 중재유닛(230)은 유효한 경우, 선택된 요구를 서비스할 때 사용하기 위한 캐시(6)에 물리 페이지 ID와 웨이 정보를 제공한다.

[0135] 도 14는 중재유닛(230)의 일부의 예를 나타낸다. 중재유닛(230)은 입력 벡터(210)를 수신하고, 주요한 액세스 요구의 가상 페이지 ID(즉, TLB에 전해진 가상 페이지 ID)를 입력 벡터(210)에서의 다른 요구들의 가상 페이지 ID와 비교하기 이한 비교기(240)를 갖는다. 가상 페이지 ID가 주요한 요구 자체를 포함하는, 주요한 요구의 가상 페이지 ID와 매치하는 어떤 펜딩 요구들이, 아비터(230)에 의해 후보 요구들로서 선택되어, 캐시의 4개의 뱅크(20)에 대응하는 4개의 뱅크 비교기 250-0 내지 250-3에 전달된다. 주요한 요구와 같은 가상 페이지 ID를 갖지 않는 요구들은 후보로서 선택되지 않으므로, 서비스되어야 하는 다음의 사이클까지 기다려야 한다.

[0136] 각 뱅크 비교기(250)는 뱅크 식별자(252)와 캐시 라인 식별자(254)를 갖는다. 뱅크 식별자 252-0는 비교기(24

0)에 의해 선택된 각 후보 요구의 어드레스를 검사하고, 어드레스가 뱅크 0에 대응하는지 여부를 어드레스의 일부(대표적으로, 페이지 오프셋부(46)의 일부)로부터 판정한다. 뱅크 0에 대응하는 어떤 후보 액세스 요구가 있는 경우, 이들 요구들의 첫 번째(예를 들면, 가장 높은 우선순위 벡터 엔트리에서의 요구)가 선택되고, 뱅크 0을 겨냥하는 어떤 다른 요구는 캐시 라인 식별자(254)에 제공된다. 뱅크 0을 겨냥하는 제1 후보 요구가 로드 요구이면, 캐시 라인 식별자(254)는, 뱅크 0을 겨냥하는 다른 후보 요구가 뱅크 0을 겨냥하는 제1 후보 요구와 같은 캐시 라인을 겨냥하는지 여부를 체크한다. 만일 그렇다면, 이들 요구가 뱅크 0에 대한 제1 후보 요구와 병합되고, 병합된 요구는 셀렉터(260)에 제공된다. 뱅크 0에 대한 제1 후보 요구와 같은 캐시 라인을 겨냥하는 어떤 다른 후보 요구도 없으면, 또는 제1 후보 요구가 머지 버퍼 엔트리이면, 뱅크 0에 대해서 선택된 제1 후보 요구가 셀렉터(260)에 제공된다(머지 버퍼 엔트리가 로드 요구와 병합될 수 없다).

[0137] 반면, 다른 뱅크 비교기 250-1 내지 250-3은 뱅크 1, 2,3에 관해서는 뱅크 비교기 250-0과 같은 동작을 행한다. 셀렉터(260)는 각 뱅크 비교기 250-0 내지 250-3에 의해 선택된 요구들(한 개의 요구 또는 병합된 요구일 수 있다)을 수신하고 그 후에 각 뱅크(20)에 대한 최대 한 개의 요구를 캐시에 의해 서비스하기 위해서 발행한다. 그리고나서, 선택된 요구는 로드 요구에 응답해서 하나의 값을 돌려보내거나, 머지 버퍼 엔트리에 나타낸 스토어 요구에 응답해서 기록 값을 캐시에 돌려보내는, 요구를 서비스하는 캐시(6)에 제공된다. 그 다음 액세스 요구들의 결과는 도 12에 나타낸 바와 같이 버스(280)를 통해서 프로세서에 다시 기록된다. 캐시(6)는 캐시 액세스를 행할 때, 상술한 방식으로 웨이 테이블(14)로부터 제공된 웨이 정보를 이용해서 할 수만 있으면 감소된 캐시 액세스를 행한다.

[0138] 그러므로, 도 12 및 도 14에 나타낸 바와 같이, 아비터(230)는 입력 벡터(10)로부터 선택된 주요한 액세스 요구와 같은 페이지에 액세스하는 이들 요구만 후보 요구로서 선택한다. 주요한 요구는 예를 들면 가장 오래된 요구, 가장 높은 우선순위 요구 또는 주어진 타입의 가장 높은 우선순위 요구일 수 있다. 그리고나서, 아비터(230)는 캐시(6)의 4개의 뱅크(20) 전역에 걸쳐 액세스가 분산되는 방식으로 서비스되는 요구를 후보 요구 중에서 선택하고, 같은 캐시 라인을 겨냥하는 요구들을 병합한다. 이렇게 함으로써, 후보 요구들 중에서, 가장 효율적인 캐시의 이용을 달성할 수 있다.

[0139] 셀렉터(260)가 같은 사이클에서 캐시(6)에 제공될 수 있는 총 요구의 수를 제한할 수 있다는 점에 유념한다. 프로세서(4)에 캐시(6)를 접속하는 결과 버스(280)의 개수가 한정될 수 있고, 그래서 많은 요구들이 아비터(230)에 의해 병합될 수 있는 경우에도, 셀렉터(260)는 총 서비스된 요구의 수를 결과 버스(280)의 수로 제한해야 한다. 예를 들면, 도 12에 있어서는, 4개의 결과 버스가 있으므로 셀렉터(260)는 사이클당 최대 4개의 판독 요구를 선택할 수 있다. 결과 버스(280)를 필요로 하지 않는 1개 이상의 머지 버퍼 엔트리는 또한 캐시(6)에 의해 같은 사이클에서 서비스될 수 있는 요구의 개수에 의존해서 선택될 수 있다. 서비스될 요구의 선택은 서로 다른 방식으로 이루어질 수 있다. 예를 들면, 셀렉터(260)는 입력 벡터(210) 내의 그들의 위치에 따라 로드의 우선순위를 지정할 수 있다.

[0140] 각 뱅크 비교기(250)의 캐시 라인 식별자(254)가 같은 뱅크를 겨냥하는 모든 다른 후보 요구들과 각 뱅크에 대해서 선택된 제1 후보 요구를 비교하는 것이 가능하고, 도 14의 예에 있어서는, 뱅크 비교기(250)만이 그 뱅크에 대한 제1 후보 요구와 3개 이상의 후보 요구를 비교한다. 병합될 요구를 식별하기 위해서 이루어질 수 있는 비교의 개수를 제한함으로써, 비교기 회로의 양이 감소될 수 있다. 실제로, 입력 벡터로부터의 4개 이상의 액세스 요구가 같은 사이클 내에서 같은 캐시 및 같은 뱅크를 겨냥할 것 같지 않으므로, 전체 영역 및 에너지 효율은 비교기의 수를 줄임으로써 향상될 수 있다. 4개 이상의 액세스 요구가 같은 사이클 내에서 같은 캐시 및 같은 뱅크를 겨냥하는 경우는 매우 드물기 때문에, 요구의 몇몇은 다음 싸이클까지 지연될 수 있다. 또한, 같은 캐시 라인에 대한 대부분의 액세스가 연속되는 경향이 있기 때문에, 제한된 수의 연속적인 요구만을 비교해서 병합될 요구들을 식별함으로써 효율적으로 비교를 구현할 수 있다. 그래서, 비교기(250)는 어떤 추가 요구도 비교하지 않고, 첫 번째 후보 요구와 다음 3개의 연속 요구를 비교할 수 있다.

[0141] 도 12 내지 14는 한 개의 페이지를 겨냥하는 액세스 요구만이 같은 사이클에서 서비스될 수 있는 예를 나타낸다. 그렇지만, 한 개 이상의 페이지가 같은 사이클에서, 예를 들면 2개의 페이지 어드레스에서 처리되도록 허용하는 것이 가능하다. 이와 같이, TLB(12) 및 웨이 테이블(14)은 같은 사이클에서 2개의 TLB 엔트리 및 웨이 테이블 엔트리에 액세스해야 할 것이다. 따라서, 여분의 TLB 포트는 두 번째의 가상 페이지 어드레스를 수신하고 2개의 동시 TLB 테이블 검색을 허용하기 위해서 제공될 수 있다. 아비터(230)는 페이지 비교기(240)와 뱅크 비교기 250-0 내지 250-3에 의해 행해진 동작의 많은 예를 병렬로 수행할 것이고, 각 가상 페이지 ID가 동시에 처리되는 한 예를 수행할 것이다. 셀렉터(260)는 각 가상 페이지 ID에 대해서 각 뱅크 비교기 250-0 내지 250-3로부터 발행된 요구들을 수신할 것이고, 그 후에 이들 요구들 중에서 선택해서 캐시(6)에 서비스될 뱅크

(20)당 1개의 요구를 발행할 것이다.

[0142] 같은 사이클 내에서 2개의 상이한 페이지 어드레스를 처리하기 위한 회로를 제공하는 것은, 몇몇 어플리케이션이 한 개의 메모리 위치로부터 데이터가 판독되어 상이한 메모리 위치에 기록되는 메모리 카피 동작을 필요로 할 수 있기 때문에 유용할 수 있다. 그러한 동작은 메모리의 2개의 상이한 페이지와 관련되어 행해지는 데이터 액세스 요구들을 종종 필요로 하는 경우가 있어, 이들 동작이 메모리의 서로 다른 페이지를 겨냥하는 동시 액세스 요구를 가능하게 함으로써 같은 사이클 내에서 병렬로 발생하는 것을 허용하는 것이 더 효율적일 수 있다. 그럼에도 불구하고, 한 개의 사이클에서 처리될 수 있는 최대 서로 다른 페이지의 수를 2개로 한정함으로써, 회로는 임의의 수의 서로 다른 페이지 어드레스를 지정하는 액세스 요구가 동시에 처리될 수 있는 경우보다 여전히 더 효율적일 수 있다. 일반적으로, 본 기술은 같은 사이클에서 처리될 수 있는 상이한 페이지 어드레스의 최대 수가 캐시에 의해 병렬로 처리될 수 있는 액세스 요구의 총 수보다 적은 경우에도 적용가능하다.

[0143] 도 15는 도 12에 나타낸 장치를 이용해서 데이터를 처리하는 방법을 나타낸다. 스텝 300에서는, 다음 처리 사이클이 시작된다. 스텝 302에서는, 어드레스 계산 스테이지 10-0 내지 10-3으로부터의 새로운 요구가 입력 벡터(210)에 입력된다. 또한, 머지 버퍼(16)로부터 축출될 준비가 되어 있는 머지 버퍼(16)에 머지 버퍼 엔트리가 있으면, 이것은 입력 벡터(210)에도 입력된다. 또한, 프로세서에 의해 발행된 새로운 스토어는 머지 버퍼 엔트리(16)를 형성하기 위해서 나중에 병합되어도 되는 스토어 버퍼(15)에 기록된다.

[0144] 스텝 304에서는 (주요한 액세스 요구를 포함하는) 제1 유효 벡터 엔트리에 의해 지정된 가상 페이지 ID는 TLB(12)에 전달된다. 스텝 306에서는, TLB(12)는 제1 유효 벡터 엔트리의 가상 페이지 ID를 수신하고 대응하는 TLB 엔트리를 배치한다. TLB는 TLB 엔트리를 이용해서 가상 페이지 ID를 물리 페이지 ID로 변환하고, 물리 페이지 ID를 중재유닛(230)으로 돌려보낸다. 또한, 스텝 308에서는, 웨이 테이블(14)은 대응하는 웨이 테이블 엔트리에 액세스하고, 웨이 테이블(14)은 웨이 테이블 엔트리에 포함된 웨이 정보를 중재유닛(230)에 보낸다. 서비스하기 위해서 캐시(6)에 요구가 전달되면, 중재유닛(230)은 요구들을 서비스할 때 사용하기 위한 캐시(6)에 대응하는 물리 페이지 ID와 웨이 정보를 제공한다. 같은 사이클에서 서비스된 모든 요구가 같은 페이지 ID를 공유하기 때문에, 같은 물리 페이지 ID 및 웨이 테이블 엔트리가 모든 요구에 대해서 사용될 수 있다.

[0145] 한편, 스텝 310에서는, 중재유닛(230)은 TLB(12)에 전해진 가상 페이지 ID와 같은 가상 페이지 ID를 갖는 입력 벡터(210)에서의 로드 요구 또는 머지 버퍼 엔트리 중의 어느 것인가를 후보 액세스 요구로서 선택한다. 서로 다른 페이지 ID를 갖는 어떤 요구도 선택되지 않는다.

[0146] 스텝 320에서는, 뱅크 비교기 250-0은 스텝 310에서 선택된 후보 액세스 요구들 중의 어느 것이 뱅크 0에의 액세스를 필요로 하는지를 확인한다(이들 요구는 "뱅크 0 후보 액세스 요구"라고 칭한다). 적어도 한 개의 뱅크 0 후보 액세스 요구가 있으면, 스텝 322에서는 뱅크 비교기 250-0는 이들 요구 중 하나가 머지 버퍼 엔트리인지 여부를 확인한다. 만약 그렇다면, 스텝 324에서는, 그 머지 버퍼 엔트리가 캐시에 의해 서비스되도록 선택된다. 어떠한 뱅크 0 후보 머지 버퍼 엔트리도 없으면, 스텝 324에서는 뱅크 비교기 250-0가 뱅크 0을 겨냥하는 제1 유효 후보 로드 요구를 선택한다. 스텝 326에서는, 캐시 라인 식별자 250-0는 뱅크 0을 겨냥하는 다른 후보 로드를 검사하고, 스텝 324에서 선택된 제1 뱅크 0 후보 로드와 같은 캐시 라인을 겨냥하는 어떤 다른 뱅크 0 후보 로드가 있는지를 확인한다. 그러한 다른 로드가 있으면, 스텝 328에서는 같은 캐시 라인을 겨냥하는 로드가 병합되고, 셀렉터(260)는 캐시에 의해 서비스하기 위한 병합된 요구를 선택한다. 제1 뱅크 0 후보 로드와 같은 캐시 라인을 겨냥하는 어떤 다른 뱅크 0 후보 로드 요구도 없으면, 스텝 330에서는 제1 뱅크 0 후보 로드가 캐시에 의해 서비스하기 위해서 선택된다.

[0147] 한편, 스텝 340, 350, 및 360에서, 다른 뱅크 비교기 250-1, 250-2, 250-3은 뱅크 1, 2, 및 3을 각각 겨냥하는 스텝 310에서 선택된 후보 요구에 대해서 스텝 322 내지 330과 같은 스텝을 행한다. 그러므로, 각 뱅크에 대해서, 그 뱅크를 겨냥하는 적어도 한 개의 후보 요구가 있으면, 그 뱅크에 대해서 한 개의 요구가 선택될 것이다. 모든 선택된 요구들은 캐시(6)에 전해지고, 스텝 306, 308에서 취득된 웨이 테이블 엔트리와 물리 페이지 ID를 이용해서 스텝 380에서 캐시(6)에 의해 서비스된다. 스텝 382에서는, 웨이 테이블 정보의 갱신이 요구되면(이전에 설명한 바와 같이, 캐시 액세스들 중 하나가 캐시 라인 필, 축출, 또는 마이크로 웨이 테이블 미스/캐시 히트를 발생하면) 캐시(6)가 웨이 테이블(14)에 신호를 보내고, 그에 따라서 웨이 테이블(14)은 웨이 테이블 엔트리의 대응하는 필드를 갱신한다. 스텝 384에서는 이 사이클에서 캐시(6)에 의해 서비스되지 않았던 입력 벡터(210)에 남아 있는 어떤 액세스 요구들이 다음 사이클에서 선택하기 위한 도 13에 나타낸 것과 같은 벡터 엔트리 #1, #2, 또는 #3에 저장된다. 그러므로, 이들 요구는 새로운 요구보다 먼저 우선적으로 처리될 것이다. 스텝 300에서는, 다음 사이클이 개시되고, 도 15의 방법은 한 번 더 개시된다.

[0148] 스토어 버퍼(15) 및 머지 버퍼(16)는 1개의 가상 페이지 어드레스만이 같은 사이클 내에서 처리되는 것을 허용하는 실시예의 맥락에서 더 최적화될 수 있다. 사이클당 4개의 로드까지 전체 연관 검색을 허용하기 위해서, 스토어 버퍼(15) 및 머지 버퍼(16)를 검색하기 위한 대응하는 비교기 어레이가 보통 4번 복제될 필요가 있을 것이다. 그렇지만, 한 사이클 내에서 서비스된 모든 요구가 같은 페이지에 액세스해야 하는 경우에는, 어드레스 비교에 책임이 있는 비교기 어레이가 남아 있는 어드레스 필드에 대한 4개의 독립된 비교기와 가상 페이지 어드레스에 대한 1개의 공통 비교기로 분리될 수 있다. 32비트 어레이와 4K바이트 페이지를 이용하는 시스템에 있어서는, 이것에 의해서 3개의 20비트 비교가 절약되어서, 에너지가 절약된다.

[0149] 요약하면, 본 기술은 아비터(230) 또는 캐시(6)에서와 같이 시스템의 다른 부분에서 제한된 수의 비교기와 TLB 변환을 수행하기 위한 제한된 수의 TLB 포트만으로 서로 병렬로 다수의 캐시 액세스를 서비스할 수 있다. 이것에 의해 더 낮은 에너지 소모를 가진 좀더 효율적인 시스템이 가능하게 된다.

[0150] 본 기술에 대한 추가 정보가 이하에 제공된다.

[0151] **1. 도입**

[0152] 현대의 비순차 슈퍼스칼라(out-of-order superscalar) 프로세서는 ILP(instruction-level parallelism)을 이용하기 위해서 사행 실행(speculative execution), 와이드 이슈 윈도우(wide issue windows) 및 다수의 데이터 패스에 의존하고, 메모리 액세스 레이턴시 및 스루풋을 향상시키기 위해서 빠른 온-칩 SRAM 캐시로 이루어진 메모리 계층에 의존한다. 모든 명령의 높은 퍼센티지는 메모리 참조이므로(ARMv7에서 작동되고 있는 SPEC CPU2000에 대해서 평균적으로 40%), 그것은 병렬로 다수의 액세스를 지지하기 위해서 제1 레벨 데이터 캐시에 대해서 필수적이다(세션 2 참조). 이 문제는 최신의 마이크로프로세서에서 상당히 공통되는 개선된 메모리 추론 및 명확화 기술(memory speculation and disambiguation techniques)에 의해, 그리고 SIMD 계산 패러다임에 대한 지원을 제공하는 Intel AVX 및 ARM NEON과 같은, 벡터 확장에 의해 악화되고, 메모리 서브시스템에 대한 훨씬 더 많은 요구 사항을 소개할 수 있다.

[0153] 그렇지만, 반도체 기술 스케일링에서의 한정된 에너지원 및 제약은 모바일 기기에 대한 고정된 에너지 비용을 초래하고, 또한, 쿨링(cooling) 및 전기 비용이 데스크탑 및 서버 세그먼트에 있어서 상당히 중요해지고 있다. 그 결과, 에너지 효율이 마이크로프로세서 디자인에 대한 결정 요인 및 성능 향상에 대한 주요 장애물 중 하나가 되었다. 캐시가 온 칩 전력 소모에의 주요한 기여자 중 하나이므로, 캐시 레벨 최적화는 에너지 효율이 저하된 증가된 성능을 트레이드 오프(trade-off)할 필요가 있다.

[0154] 이 기술은 에너지 효율적인 방식으로 다수의 액세스 L1 데이터 캐시를 구현하는 문제를 해결한다. 인텔의 Sandy Bridge 및 AMD의 Bulldozer와 같은 커런트 하이 엔드 마이크로아키텍처(current high end microarchitectures)는 사이클당 2개의 128비트 로드 및 1개의 128비트 스토어까지 허용한다. 양쪽은 물리적 멀티포팅(multi-porting) 및 캐시 뱅킹에 의존한다. 전자의 기술은 다수의 포트를 가진 변경된 SRAM 셀에 의존하고; 그것은 낮은 액세스 레이턴시를 허용하지만, 큰 에너지와 면적 페널티를 도입한다. 이에 반하여, 뱅킹은, 캐시 라인의 서브 세트를 각각 홀드하는, 몇 개의 작은 구조를 이용함으로써, 액세스당 에너지 소모를 효율적으로 줄인다. 뱅크들은 다수의 메모리 참조를 서비스하도록 독립적으로 액세스될 수 있지만, 같은 뱅크에의 액세스 맵핑은 직렬화되어야 한다(뱅크 충돌). 새로 나올 프로세서 세대는 개선된 벡터 확장뿐만 아니라 공격적인 메모리 추론 및 명확화 기술을 처리하기 위해서 훨씬 더 정교한 캐시 인터페이스를 필요로 한다(예를 들면, 비단위 보폭 로드(non-unit strided loads)를 지지하는 인텔의 AVX2).

[0155] 다수의 동시 메모리 액세스를 처리하는 문제는 벡터 머신의 측면에서 이미 분석되었다. Tarantula, CODE 및 VT 구조는 정교한 벡터 구조의 예이다. 그렇지만, 이들 디자인은 상당한 양의 전용 하드웨어와, 구현된 솔루션을 필요로 하고, 풍부한 데이터-레벨 병행성(parallelism)을 가진 효과적인 온 워크로드(effective on workloads)는 범용 마이크로프로세서에 적합하지 않다. 현대의 캐시의 주요한 특징은 큰 워킹 세트(working-sets)를 수용하여 미스율(miss rates)을 줄이기 위한 높은 용량 및 세트 결합(set-associativity)이다. 그렇지만, 특정 데이터는 단지 하나의 웨이에 배치될 수 있지만, n-웨이 세트 결합 캐시의 검색은 n개의 태그 비교 및 n개의 데이터-어레이 액세스를 필요로 한다. 리던던트 액세스(redundant accesses)를 피함으로써 에너지를 절약하려고 하는 기술은 "웨이 예측", "웨이 추정" 및 "웨이 판정" 방식으로서 분류될 수 있다. 제1 그룹은 MRU 통계에 의거해서 웨이를 예측한다. 이 개념은 구현이 간단하지만, 거짓 예측은 이전에 파기된 웨이 내에서 원하는 데이터를 찾기 위해서 제2 캐시 액세스를 필요로 한다. 다른 방식은 선택적인 다이렉트 맵핑과 웨이 예측의 결합에 의거해서 예측 정확도를 증가시킴으로써 이 문제를 완화하려고 시도한다. 웨이 추정 기술은 한 개의 웨이 대신에 한 세트의 웨이를 전달한다. 원하는 데이터가 캐시 내에 존재하면, 거기에서 발견되는 것을 보장한다. 그 결과, 캐

시 액세스는 2개의 사이클밖에 필요로 하지 않지만, 몇 개의 리던던트 태그 비교를 위한 에너지를 소모할 수도 있다. 대체가능한 기술의 그룹은 웨이를 예측하기보다는 결정한다. 웨이 결정 유닛(WDU;way determination unit)은 작은 버퍼에 최근에 사용된 한 세트의 캐시 라인에 대한 웨이 정보를 저장한다. 각 라인은 정확히 하나의 웨이와 관련되어 있고, 거기를 히트(hit)하거나 전체 캐시를 미스(miss)하도록 보장된다. WDU는 병렬 액세스당 1개의 포트를 포함하는 전체 연관 검색 구조를 필요로 하므로, 에너지 효율이 그것의 사이즈에 반비례하게 된다. 그 결과, 그것은 작은 수의 로드를 서비스하기 위해 설계된 캐시 액세스의 높은 잠정 구역성(temporal locality)을 갖는 시스템에 제한되며 병렬로 저장한다.

[0156] 이 기술에서 제안된 MALEC(Multiple Access Low Energy Cache)는 연속적인 메모리 참조가 동일한 페이지에 액세스할 가능성이 매우 높은 관측에 근거한다. 그 결과, 그것은 다수의 액세스 사이의 메모리 어드레스 변환 결과를 공유하고 슈퍼스칼라 비순차 프로세서(즉, 스토어 및 머지 버퍼)에서 공통인 특정 구성요소의 검색 구조를 단순화한다. 페이지 기반 액세스 그룹화는 또한 작은 세트의 비교기의 어플리케이션이 같은 캐시 라인에 액세스하는 로드를 식별하는 것을 허용한다. 이들 로드 중에서 단 한 개의 캐시 액세스로부터 수신된 데이터를 공유하면 뱅크 충돌의 수가 효율적으로 감소된다. 또한, MALEC는 로드에 웨이 정보를 동시에 제공하고 같은 페이지에의 액세스를 저장하는 새로운 웨이 결정 방식을 도입한다. 자신의 검색 메카니즘을 간소화하기 위해서 TLB 검색에 필요한 어드레스 비교를 재이용하는 것이 가능하다. 웨이 결정을 위한 유효 정보의 추가에 의해, 다수의 메모리 참조가 태그 비교를 완전히 피하고 원하는 캐시 라인에 직접 액세스하는 것이 가능하다. 이것은 적어도 한 개의 태그 비교로 그들의 결과를 검증할 필요가 있는 D-캐시에 대한 다른 예측 방식과 MALEC를 구별한다.

[0157] ## 2. 모티베이션(MOTIVATION)

[0158] 슈퍼스칼라 프로세서의 주요한 성능 미터법(metric)은 사이클당 실행된 명령의 개수이다. SPEC CPU2000 및 MediaBench2 벤치마크 세트의 우리의 분석은 이 미터법으로 메모리 참조의 심각한 영향을 보여주었다. 실제로, 그들은 모든 명령의 40%(로드:스토어의 비=2:1)을 구성한다. 스칼라보다는 오히려 벡터에 대해서 동작하는 SIMD 확장은 강력한 캐시 인터페이스에 대한 요구를 강화한다.

[0159] 도 16(i)은 L1 D-캐시에의 연속적인 판독 액세스들 간의 관계를 나타낸다. 그 결과는 10억개의 명령의 간격에 대한 Simpoint v3.0에 의해 식별된 각 벤치마크의 가장 대표적인 실행 단계를 분석하기 위해서 2진 계측도구를 이용해서 취득되었다. 특히, 도 16(i)은 연속적인 판독의 전체 67% 뒤에 동일 페이지에 대해서 적어도 한 개의 판독이 이어지고, 즉, 판독의 15%, 14%, 12%, 6% 및 20% 뒤에 동일한 페이지에 대해서 각각 1, 2-3, 4-7, 8-15, 및 15 이상 판독이 이어진다는 것을 나타낸다. 도 16(ii)은 상이한 페이지에 한 개의 중간 액세스를 허용함으로써 이 분석을 확장함으로써 판독의 83% 뒤에 동일한 페이지에 대해서 적어도 한 개의 액세스가 이어진다. 2개 및 3개의 중간 액세스를 허용하면 이 수가 각각 90% 및 95%로 더 증가된다(도 16에는 도시되어 있지 않음). 그 결과, 로드의 대부분이 세션 4에 소개된 페이지 기반 메모리 액세스 그룹화에 적합하다. 스토어에 관한 비슷한 분석은 훨씬 더 높은 등급의 어드레스 구역성을 나타낸다. 그렇지만, 스토어 및 머지 버퍼-세션 3에 기재된-가 대응하는 캐시 액세스 패턴을 상당히 변경하기 때문에, 여기에서는 그들에 대해서 더 이상 설명하지 않는다.

[0160] 도 16(iii)은 페이지 기반 메모리 액세스 그룹화보다는 오히려 라인에 대한 기회를 조사한다. 그것은 로드의 46% 뒤에 동일한 캐시 라인에 액세스하는 적어도 한 개의 다른 로드가 이어진다는 것을 나타낸다. MALEC는 4개의 로드까지 한 개의 특정 캐시 라인으로부터 판독된 결과를 공유하는 것을 허용함으로써 이 관측을 활용한다. 세션 6은 특정한 MALEC 구성에 대해서 공유하는 D-캐시 라인 및 페이지 기반 메모리 액세스 그룹화의 성능 영향을 분석한다.

[0161] ## 3. 기본 프로세서 캐시 인터페이스

[0162] MALEC는 비순차(out-of-order) 슈퍼스칼라 프로세서를 겨냥하기 때문에, 이들 디자인과 관련된 다양한 복합한 구성소자를 고려하는 것이 중요하다. 도 17은 간단한 형태의 정교한 L1 캐시 인터페이스를 나타낸다. 프로세서에 의해 발행된 각 로드 및 스토어는 어드레스 계산, 어드레스 변환, 데이터 액세스 및 필요한 경우 하부 레벨 캐시 액세스를 겪는다. 어드레스 계산 회로는 메모리 계층 액세스 전에 물리 어드레스로 변환될 필요가 있는 어플리케이션 특정 가상 어드레스를 생성한다. MALEC는 물리적으로 인덱스된, 물리적으로 태그된 L1 데이터 캐시라고 가정하고, 현 상황에서는 오늘의 마이크로프로세서에 있어서의 공통 디자인 선택이라고 가정한다는 점에 유념한다. 어드레스 변환은 페이지 단위에 의거해서 행해진다(예를 들면, 메모리 어드레스 공간의 4K바이트).

그들은 TLBs(Translation Lookaside Buffers) 및 페이지 테이블의 계층을 포함한다. TLBs는 프로세스에 가장 가깝고 작은 세트의 가장 최근에 액세스된 페이지에 정보를 홀드하여 변환 레이턴시 및 에너지 소모를 줄인다. 마이크로 TLBs(uTLBs)는 서브 세트의 TLB 엔트리만을 포함으로써 데이터 구역성을 추가로 활용한다. 또한, 도 17의 부분은 이 예에서 사용된 메모리 어드레스 비트 필드에 대한 명명 규칙(naming conventions)이다. 비트 필드 사이즈는 세션 6에 기술된 디자인 파라미터, 즉 64바이트 와이드 라인 내의 데이터의 32K바이트를 홀딩하는 4웨이 세트 결합 캐시, 4K바이트 페이지, 및 32비트 시스템 버스에 대응한다. 이들 파라미터의 대부분은 고성능 프로세서에 대해 실현 가능하고, 32비트 시스템 버스의 선택은 이용된 시뮬레이션 프레임워크에 의해 지시된다. 그렇지만, 물리 어드레스 공간이 더 넓은 구조(40-48 어드레스 비트)는 MALEC에 의해 소개된 에너지 최적화로부터 훨씬 더 혜택을 받을 것이다(세션 4 참조).

[0163] 도 17에 나타낸 스토어 버퍼(SB)는 스토어의 사행 실행을 허용하기 위해서 비순차 머신에서 사용된다. 그것은 인플라이트(in flight) 스토어에 정보를 홀드하고, 의존적인 로드에 데이터를 전달하는 것을 허용한다. 도 17의 부분이 아니라 나중에 관련된 부분은 로드에 대한 SB의 동등한 것이다. 로드 큐는 인플라이트 로드를 홀드하고, 데이터 의존 위반을 삭제하기 위해서 사용된다. 스토어가 수용될 때, 해당 SB 엔트리가 축출되어 MB(Merge Buffer)에 전달된다. MB는 다수의 스토어로부터의 데이터를 동일한 어드레스 영역과 병합으로써 L1 캐시 액세스의 수를 줄이려고 하는 작은 메모리 구조이다. 새롭게 수용된 스토어가 프리 엔트리에 병합되거나 할당될 수 없는 경우에 한해, 가장 오래된 MB 엔트리가 축출되어 L1 캐시에 기록된다. 로드는 통상적으로 추가 레이턴시를 피하기 위해서 병렬로 SB, MB 및 L1 캐시에 액세스한다. 멀티플렉서는 MB 위의 SB 및 L1 캐시 위의 MB의 우선순위를 매기는 모든 3개의 소스로부터 수신된 결과를 결합한다.

[0164] 도 2는 4-웨이 세트 결합 캐시의 예를 나타낸다. 이 디자인은 특정 어드레스 영역, 예를 들면 뱅크 0 짝수 라인, 뱅크 1 홀수 라인에 대응하는 데이터를 각각 홀딩하는 2개의 비의존 캐시 뱅크를 이용한다. 뱅크들은 전체 캐시보다 작기 때문에, 그들은 보다 빨리 더 에너지 효율적으로 액세스될 수 있다. 하나의 뱅크는 메모리 어드레스의 인덱스 필드 내의 LSB(least significant bit)에 의해 식별된다(도 17 참조). 인덱스 필드의 나머지를 이용해서 특별한 뱅크 내의 캐시 라인을 선택한다. 각 라인은 그것의 메모리 위치 및 데이터 어레이 엔트리를 명확하게 식별하기 위해 태그 어레이 엔트리로 구성된다. 도 2의 뱅크들은 또한 각 라인이 4개의 서로 다른 위치 내에 캐시되는 것을 허용하는 4-웨이 세트 결합이다. 이것은 히트율(hit rates)을 상당히 향상시키지만, 4개의 가능한 위치의 검색은 액세스당 필요한 에너지를 증가시킨다. 종래의 캐시는 태그들을 매치해서 해당 데이터를 선택하기 위해서 병렬로 모든 4개의 태그 및 데이터 어레이 쌍에 액세스해야 한다.

[0165] **4. 페이지 기반 액세스 그룹화**

[0166] 도 12는 병렬로 4개의 메모리 요구까지 서비스하도록 설계된 제한된 다수의 액세스 로우 에너지 캐시 인터페이스의 예를 나타낸다. WT 및 uWT는 세션 5에서의 웨이 결정 방법의 일부로서 설명될 것이다. 그들은 MALEC의 동작에 대해서 강제적이지는 않다. 이 장치는 4개의 병렬 메모리 액세스까지 지원한다. 이 능력을 달성하기 위한 직관적 접근법은 해당 큐의 포트의 수를 간단히 증가시키는 것이고 어드레스 계산 및 우선순위 다중화를 위한 회로를 복제하는 것이다.

[0167] 어드레스 계산을 끝내는 스토어들은 사이클당 어드레스 변환의 개수를 줄이기 위해서 SB에 직접 전달된다. 이것은 스토어들이 L1 대신에 MB에 수용됨에 따라, 중요한 성능 페널티를 부과하지 않는다. 이에 반하여, 어드레스 계산을 끝내는 로드들은 소위 입력 벡터의 8개의 엔트리 중에서 4개를 대표한다. 나머지 엔트리는 이전 사이클에서 서비스될 수 없는 3개의 로드까지이며, 또 1개의 MBE(Merge Buffer Entry)가 된다. 순차적으로, 즉 홀수 로드, 신규 로드 및 축출된 MBE의 순으로 엔트리들의 우선순위를 정한다. 축출된 MBE의 우선순위가 낮은 이유는, MBE로 나타낸 스토어들이 이미 수용되어 더 이상 시간이 중요하지 않다고 하는 사실이다. 각 사이클의 시작부분에서는, 가장 높은 우선순위 입력 벡터 엔트리의 가상 페이지 ID(vPageID, 도 17 참조)는 종래의 어드레스 변환을 위한 uTLB에 전달된다. 동시에, 이 vPageID는 모든 나머지의 현재 유효한 엔트리와 비교된다. 모든 매칭 엔트리는 중재유닛에 전달되고, 여기 각 뱅크에 대해서는 우선순위가 가장 높은 액세스가 식별되고,-하나의 로드의 경우에는- 그것을 따르는 3개의 연속 로드까지 그룹화된다. vPageID와 나머지 어드레스 비트의 비교분할에 의해 중재유닛의 비교기의 복잡성이 감소된다. 모든 입력 벡터 엔트리 대신에 3개의 연속 로드만을 비교함으로써 이 복잡성을 더 줄이기 위한 방법은 세션 2로 나타낸 것처럼, 대부분의 그룹화 가능한 로드를 포함시킨다. 다음에, 중재유닛은 4개의 가장 높은 우선순위 로드를 선택함으로써, 이용가능한 결과 버스의 수로 로드의 수를 제한한다. 대체 시스템은 가장 적은 수의 캐시 액세스를 필요로 하는 로드의 결합을 결정할 수 있다.

그렇지만, 그러한 회로의 복잡성이 증가하면 사이클당 더 많은 에너지와 시간이 필요하게 된다.

[0168] 중재유닛에 의해 선택된 메모리 액세스는 L1 캐싱 전달되고, 로드의 경우에는 SB 및 MB에도 전달된다. 캐시 자체는 높게 최적화된 디자인의 구현을 허용하도록 변경되지 않는다. 특별한 경우는 더 작은 비의존 뱅크(통상 128비트 와이드)에서의 데이터 어레이를 분할함으로써 에너지를 절약하려고 하는 서브 뱅크드(sub-banked) 캐시이다. MALEC는 1개의 서브 뱅크를 초과하는 것들만 대신에, 모든 판독 액세스에 대한 2개의 서브 뱅크로부터 데이터를 리턴하기 위해서 이들 디자인을 필요로 한다. 이것은 캐시로부터 판독한 결과를 공유할 수 있도록 로드에 대한 확률을 효율적으로 두 배로 한다. SB 및 MB의 디자인은 병렬로 4개의 로드까지 서비스하는데 필요한 추가 포트의 에너지 영향을 줄이기 위해 약간 변경된다. 특히, 그들의 검색 구조는 2개의 세그먼트로 분할된다. 하나는 모든 4개의 로드 중에서 공유된 vPageID에 대응하는 어드레스 비트를 검색하는 것이다. 두 번째는 액세스 특정 어드레스 영역(즉, 캐시 라인/서브 뱅크)을 식별하는 나머지 비트들을 비교하는 것이다. 양쪽 세그먼트가 동시에 검색됨에 따라, MB 및 SB 에너지 요건이 추가 레이턴시를 도입하지 않고 감소된다. SB 및 MB는 가상으로 태그된 L1 캐시와 대조적이기 때문에, vPage 검색은 실제로 어드레스 변환 전에 행해질 수 있다는 점에 유념한다. 그렇지만, 이것은 고성능 프로세스에 대해서 공통적이므로, 분리된 파이프라인 스테이지에서 어드레스 변환 및 데이터 액세스의 실행을 복잡하게 만들 것이다.

[0169] 요약하면, MALEC는 캐시 뱅킹 및 머지 버퍼와 같은 기술을 이용해서 병렬로 다수의 명령을 서비스하려고 한다. 또한, 그것은 로드가 L1 캐시 라인으로부터 판독한 데이터를 공유하는 것을 허용하고 에너지 효율을 향상시키기 위한 메카니즘을 도입한다. 페이지 기반 메모리 액세스 그룹화를 이용해서 페이지 변환 결과를 재이용하고 중재 유닛, SB 및 MB 내에서의 어드레스 비교를 간소화한다. 중재 유닛은 특정 은행에 대해서 가장 높은 우선순위 로드를 따르는 3개의 연속 액세스로의 제한에 의해 더 간소화된다.

[0170] MALEC의 중요한 관심은 그것의 구성요소에 의해 도입된 레이턴시이다. 이것을 해결하기 위해서, 입력 벡터는 uTLB/TLB 액세스에 대해서 동시에 그들을 수행함으로써 vPageID 비교의 레이턴시를 숨긴다. 같은 캐시 라인에 액세스하는 로드들을 식별하기 위해 나머지 어드레스 비트들 간을 비교하는 것은 중재 유닛 내에서 병렬로 행해진다. 그 결과, 단위 전체 레이턴시는 한 개의 좁은 비교기 및 몇몇 추가 제어회로와 같다. 또 다른 관심은 MALEC의 확장성이다. 실제로, 그것의 효율성은 프로세서에 의해 생성된 병렬 메모리 요구들의 수에 비례한다. 이것에 대한 이유는 어드레스 변환, 웨이 정보 및 - 같은 캐시 라인에 대한 로드의 경우에는- L1으로부터 판독한 데이터를 공유할 수 있는 입력 벡터 내에 다수의 액세스를 가질 가능성의 증가이다. 또한, MALEC는, 입력 벡터 내의 vPage 비교에 필요한 추가 에너지가 감소된 수의 어드레스 변환 및 간소화된 SB 및 MB 검색에 의한 높은 절약보다 더 크기 때문에, 40비트 또는 48비트의 어드레스 공간을 가진 시스템에 훨씬 더 유용한다. 또한, 더 큰 L1 태그 어레이는 MALEC의 웨이 결정 방식의 효율을 더 향상시킨다(세션 5 참조). 최종적으로, 비순차 프로세서에 대한 중요한 관심은 정확한 예외의 처리이다. MALEC 자체에 의해 홀드된 투기 상태의 양은 입력 벡터 및 중재 유닛 내의 로드로 한정된다. 예를 들면 uWT/WT 엔트리 또는 축출된 엔트리 형태의 모든 다른 정보는 비투기적이므로 어떤 회복 메카니즘에 대해서도 신경 쓸 필요가 없다.

[0171] ## 5. 페이지 기반 웨이 결정

[0172] 이 기술에 제안된 웨이 결정 방식의 주요한 구성요소는 소위 WT(Way Table) 및 uWT(Micro Way Table)이다(도 12 참조). 그들은 TLB 및 uTLB와 심하게 뒤섞여 있다. 실제로, TLB 및 WT는 캐시의 태그 어레이 및 데이터 어레이와 비슷하게 행동한다. TLB 히트는 어드레스 변환 결과뿐 아니라 해당 WT 엔트리를 리턴한다. 그 결과, 페이지 크기의 어드레스 검색에 대한 에너지는 양 구성소자로 분할된다. 각 WT 엔트리는 1페이지 내의 모든 캐시 라인에 대한 웨이 및 유효 정보를 포함한다. 4K바이트 페이지 및 64바이트 캐시 라인을 가진 시스템에 대해서는, 각 WT 엔트리가 4096/64=64 라인에 정보를 홀드한다. WT 엔트리 내의 데이터의 위치는 절대적으로 그것과 관련된 라인 어드레스를 인코딩하고, 즉 첫 번째 및 마지막 비트 필드는 라인 0 및 63을 각각 식별한다. 라인당 저장된 비트의 수는 L1 캐시 결합, 예를 들면 4웨이에 대하여 2비트에 비례한다. 4개의 뱅크로 이루어지는 캐시는 개개의 뱅크 내에 라인 1..3을 홀드하고, 같은 뱅크 내에 라인 0,4,8,..60을 홀드한다(세션 3 참조)는 점에 유념한다. 라인 0..3에 대해서 웨이 0이 무효라고 여기고, 라인 4..7에 대해서는 웨이 1이 무효라고 여김으로써, MALEC는 각 라인과 관련된 2비트 내에 유효 정보를 포함한다. 그러므로, WT 엔트리당 크기는 64*2bit=128bit가 된다. 이것은 L1 교체 정책의 성능에 크게 영향을 주지 않는다. 특히, 기술된 캐시 인터페이스, LRU 교체 정책 및 세션 6에 소개된 벤치마크 세트에 의거한 시뮬레이션은 L1 히트율의 어떤 측정 가능한 감소도 보이지 않았다.

[0173] 중재 유닛은 원하는 캐시 라인의 그룹과 웨이를 관련짓고 그들을 해당 캐시 뱅크에 전달함으로써 WT 엔트리를 평가한다. MALEC는 도 17의 이전에 소개된 캐시 디자인에 대하여 2개의 상이한 액세스 모드를 지원함으로써 웨이 정보를 이용한다.

[0174] 종래의 캐시 액세스(미공지된 웨이):

[0175] · 모든 태그 어레이 및 모든 데이터 어레이에의 병렬 액세스

[0176] · 매칭 태그와 관련된 데이터를 선택

[0177] 감소된 캐시 액세스(공지된 유효한 웨이)

[0178] · 액세스된 어떤 태그 어레이도 없다

[0179] · 한 개의 특정 데이터 어레이에만 액세스

[0180] 감소된 캐시 액세스에 대한 전제조건은 웨이 정보의 정확성이다. uWT 및 WT의 갱신은, 어떤 uWT 엔트리도 발견되지 않았으면 WT만 액세스되는, 각 캐시 라인 필 및 축출에 대해 행해진다.

[0181] uWT 및 WT의 동기화는 uTLB 갱신 시에 전달된 모든 엔트리에 근거한다. 캐시에 의해 생성된 갱신 정보가 물리적으로 어드레스됨에 따라, uTLB 및 TLB가 가상 PageID뿐 아니라 물리적 PageID에 의거한 검색을 허용하도록 변경되어야 한다. 또한, 한정된 수의 TLB 엔트리는 캐시 내에 해당 라인을 여전히 가지고 있는 페이지의 축출을 필요로 할 수 있다. 이들 라인 중 하나가 나중에 다시 액세스되어야 하므로, 새로운 WT 엔트리가 할당되고 모든 웨이 정보가 무효화된다. 정보의 손실을 보상하기 위해서, 마지막 판독 uWT 엔트리는 필요하다면 다음 캐시 히트에 대해서 갱신된 얼라이브 및 웨이 정보를 홀드한다.

[0182] 웨이 결정 방식에 대한 중요한 파라미터는 에너지 소모, 레이턴시 및 확장성이다. 중재 유닛 내부에서의 uWT/WT 액세스 및 어드레스 비교가 병렬로 처리됨에 따라, 양 구성요소에 의해 도입된 레이턴시가 오버랩될 수 있다. 이 방식은 한 페이지 내의 모드 라인에 대한 웨이 예측을 동시에 전달하도록 설계되어 있으므로, MALEC의 실제의 계산 성능에는 의존하지 않는다. 이 방식은, WT 엔트리의 사이즈가 감소되고 재사용된 TLB 어드레스 검색에 의한 절약이 증가되기 때문에, 보다 넓은 L1 캐시 라인 및 어드레스 버스(예를 들면, 48비트)에 대해서 더 에너지 효율적이 된다. 비록 더 높은 캐시 결합방법이 더 넓은 WT 엔트리를 필요로 하지만, 웨이 결정에 의해 절약된 추가 에너지는 실제로 그 방식의 효율을 증가시킨다. 최종적으로, 큰 페이지(예를 들면, 64K)가 WT 엔트리당 홀드된 라인의 수를 상당히 증가시킴에 따라, 이 방식은 uTLB를 입력할 때 4K바이트 세그먼트로 양자화될 TLB 엔트리를 필요로 한다. WT 자체는 각각이 4K바이트 어드레스 공간에 대응하는 데이터를 나타내는, 소수의 덩어리로 세그먼트화될 수 있다. FIFO 또는 LRU 방식으로 덩어리를 할당해서 교체함으로써, 그들의 수가 전체 페이지를 나타내는데 필요한 것보다 더 작을 수 있다.

[0183] ## 6. 평가 및 결과

[0184] 성능 및 에너지 소모에 대한 MALEC의 영향을 평가하고 그것을 현존하는 방식과 비교하기 위해서, gem5 시뮬레이터 시스템(Simulator System)은 사이클 레벨 정확성으로 이 연구에 수반된 마이크로 구조의 관점을 모델링할 수 있는 개선된 프로세서 캐시 인터페이스를 지지하도록 확장되었다. gem5로부터 취득된 액세스 통계와, CACTI v.6.5를 이용해서 산출된 에너지 추정을 결합해서 정적 및 동적 구성요소 양자를 포함하는, 데이터 캐시 서브시스템의 에너지 소모를 결정한다. 특히, 평가는 다음의 구조: L1 데이터 캐시(태그&데이터 SRAM 어레이 및 제어 로직을 포함한다), uTLB+uWT 및 TLB+WT의 에너지 기여를 포함한다. 모델링된 L1 캐시 인터페이스는 LQ, SB 및 MB와 같은 다른 구조를 포함하고, 전체 에너지 소모에 대한 그들의 기여도는 2가지의 이유로 고려되지 않는다. 첫 번째로, L1, uWT 및 WT는 L1 인터페이스의 다수의 트랜지스터와 그것의 리키지 전력을 설명한다. 두 번째로, LQ, SB 및 MB와 같은 다른 구성요소에 의해 기부된 에너지는 MALEC와 분석된 베이스라인 사이에서 매우 유사하다. 우리의 시뮬레이션은, 이것이 또한 하부 메모리 레벨, 즉 L2 캐시 및 메인 메모리의 경우이지만, MALEC가 L2 액세스의 타이밍을 변경함에 따라, 그들의 개수 또는 미스율에 상당히 영향을 받지 않는다는 것을 보여준다.

[0185] 표 1:

	Address computations per cycle	uTLB/TLB ports	Cache Ports per Bank
Base1ldst	1 ld/st	1 rd/wt	1 rd/wt
Base2ld1st	2 ld + 1 st	1 rd/wt + 2 rd	1 rd/wt + 1 rd
MALEC	1 ld + 2 ld/st	1 rd/wt	1 rd/wt

[0186]

[0187] 테이블 1은 uTLB, TLB 및 캐시 포트(rd..read, wt..write, rd/wt..read or write)의 개수뿐만 아니라, 사이클당 포텐셜(potential) 어드레스 계산(ld..load, st..store, ld/st..laod or store)에 관하여 분석된 베이스라인 및 선택된 MALEC 구성을 특징으로 한다. Base1ldst는 사이클당 한 개의 로드 또는 스토어로 제한되고, Base2ldlst는 병렬로 2개의 로드 및 1개의 스토어까지 허용하는 고성능 구성을 나타낸다. 시뮬레이션된 프로세서가 Base2ldlst에 대해서 최적화됨에 따라, 세션 4에서 도입된 MALEC 구성은 유사한 성능을 달성하기 위해서 스케일 다운(scale down)된다. 이것은 특히 Base2ldlst와 MALEC 사이의 공정한 비교를 허용한다.

[0188] 표 2:

Component	Parameter
Processor	Single-core, out-of-order, 1 GHz clock, 168 ROB entries, 6 element fetch&dispatch width, 8 element issue width
L1 interface	64 TLB entries, 16 uTLB entries, 40 LQ entries, 24 SB entries, 4 MB entries
L1 D-cache	32 Kbyte capacity, 64 byte line size, 2 cycle latency, 4 independent banks, 4-way set-associative, physically indexed, physically tagged
L2 cache	1 MByte capacity, 16-way set associative
CACTI	32nm technology, design objective low dynamic power, cell type low standby power for data & tag arrays and high performance for peripherals, L1 with ECC

[0189]

[0190] 시뮬레이션된 시스템의 구성은 1GHz에서 동작하는 ARMv7 호환이 되는 싱글 코어 비순차 프로세서에 근거한다. 분석된 프로세서 캐시 인터페이스의 관련 구성 파라미터들은 테이블 2에 요약되어 있다. 이 연구에서 이용된 벤치마크 세트, MediaBench 2(MB2), SPEC CPU2000 Int 및 FP는 다수의 상이한 메모리 액세스 행위를 갖는 한 세트의 워크로드(workloads)를 나타낸다. MB2의 경우에, NEON SIMD 엔진을 이용해서 자동 백터화를 가능하게 하여, 이들 벤치마크에 의해 L1 인터페이스에 대한 압력을 증가시켰다. 시뮬레이션 시간을 줄이기 위해서, SimPoint v.3.1은 각 벤치마크의 가장 대표적인 실행 단계를 식별하기 위해서 사용되었다. 각 단계는 해당 참조 워킹 세트의 10억개의 명령을 포함한다.

[0191] **6.1 성능 평가**

[0192] 도 18은 베치마크당 필요한 CPU 사이클에 관하여 Base1ldst에 대한 MALEC 및 Base2ldlst의 성능을 나타낸다. 그래프는 또한 분석된 벤치마크 전역 및 각 세트에 대한 연산수단도 포함한다. MALEC가 Base1ldst와 유사한 단일 포트(single ported) uTLB, TLB 및 L1을 이용하지만, 이것은 9%의 평균 성능 개선을 달성하는 것이 관측될 수 있다. 이것은 멀티 포트(multi-ported)되어야 하는 이들 구성요소를 필요로 하는 Base2ldlst보다 단지 1% 적다. SPEC-Int, SPEC-FP 및 MB2 평균을 비교함으로써 7%, 10% 및 16%의 성능 향상이 산출된다. SPEC-Int에 대한 SPEC-FP의 증가된 혜택 이유는 메모리 참조에 비해서 적은 수의 제어 명령이고, 즉 SPEC-FP 벤치마크는 집중 제어 대신에 더 많은 데이터이다. MB2는 그것의 미디어 커널(media kernels)이 빈번한 고도로 구조화된 메모리 액세스에 의존해서 광범위한 데이터 세트에 대한 기본 동작을 계산하기 때문에 훨씬 더 유용한다. 특히 mcf 및 art는 Base1ldst에 대해 거의 어떤 개선도 보여주지 않는다. 이 이유는 더 빠른 L1 액세스로부터 이득을 받지 않는 높은 미스율로 이어지는 낮은 구역성과 결합된 큰 워킹 세트이다. 이에 반하여, djpeg 및 h263dec는 높은 액세스 구역성을 보이고 병렬로 메모리 액세스를 실행하려고 함으로써, MALEC에 대해서 대략 30%의 속력 증가가 초래된다.

[0193] Base1ldst에 대한 MALEC에 의해 허락된 성능 이득은, 동일한 캐시 라인으로 로드를 그룹화하고 병렬로 다수의

캐시 뱅크에 액세스하는 2개의 메카니즘에서 비롯된다. 도 20은 로드를 그룹화(MALEC NoLd-Grouping)하는 능력이 없는 구성 및 MALEC의 결과를 나타낸다. 평균적으로, 로드의 그룹화는 MALEC의 전체 성능 향상에 대략 20%을 기여한다. 그렇지만, 특별한 벤치마크에 대해서는 이 값은 예를 들면 gap에 대해서 56%, mpeg4enc에 대해서 45%로 상당히 높다. 최종적으로, 도 18에 있어서의 MALEC와 Base2ldlst 사이의 딱 1%의 전체 성능 차이는 동일한 페이지에 액세스하는 한 사이클에서 이들 명령만을 처리하는 데에 충분하다는 것을 암시하는 세션 2에서 취득된 결과를 보여준다.

[0194] **6.2 에너지 평가**

[0195] 도 19는 이전의 세션에서 분석된 베이스라인의 동적 및 전체 에너지 계산을 나타낸다. Base2ldlst에 의한 동적 에너지 소모의 41% 증가는 그것의 uTLB, TLB 및 L1의 추가 포트에 액세스하는 데에 필요한 에너지가 원인이다. 이에 반하여, MALEC는 페이지 기반 액세스 그룹화 및 웨이 결정을 이용해서 싱글 포트 구성요소에서 동작하고, 적은 에너지 집중적인 "감소된 캐시 액세스"를 수행함으로써, Base1ldst에 비해서 동적 에너지의 33%를 절약한다(세션 5 참조). mcf에 대한 MALEC의 비정상적으로 높은 절약은 벤치마크의 유난히 높은 미스율(전체 평균 대략 7번)에서 유래한다는 점에 유념한다. MALEC가 동일한 캐시 라인에의 로드 중에서 L1 데이터를 공유하려고 시도함에 따라, 캐시에 액세스해서 미스(missing)하는 효율적인 로드의 수가 감소된다. 이 능력 없이, MALEC는 실제로 mcf에 대해서 51% 적은 동적 에너지 대신에 5% 이상을 소비할 것이다.

[0196] 리키지(leakage)가 분석된 32nm 기술 라이브러리에서 전체 에너지 소모에 대략 50%를 기여함에 따라, 그것을 설명하는 것이 중요하다. 도 19는 Base2ldlst의 평균 에너지 소모가 실제로 Base1ldst보다 높은 52%라는 것을 보여준다. 이 이유는 감소된 계산 시간으로 인해 절약을 더 중요하게 생각하는 그것의 부가적인 uTLB, TLB 및 L1 포트에 의해 도입된 리키지 전력이다. 예를 들면, 부가적인 판독 포트는 대략 80% L1 리키지 전력을 증가시키지만, 평균 계산 시간은 단지 10% 감소된다(세션 6.1 참조). 비슷한 효과가 MALEC에 대해서 관찰될 수 있다. 그것은 Base1ldst와 같은 수의 uTLB, TLB 및 L1 포트를 보유하지만, 그것의 변경된 캐시 모델 및 신규로 도입된 uWT 및 WT는 부가적인가 리키지 전력을 소개한다. 그 결과, 그것의 전체 에너지 절약이 22%로 감소된다. 그렇지만, 그것을 MALEC와 비슷한 성능을 달성하는 Base2ldlst와 비교하면, 그것은 48% 절약한다.

[0197] 대체 웨이 결정 방식은 MALEC의 리키지 전력을 줄이는 것으로 생각될 수 있다. 여기에서 제안된 웨이 테이블과 유사하게, 웨이 결정 유닛(WDU)(세션 1 참조)은 세션 5에서 소개된 것처럼 "감소된 캐시 액세스"를 지지하도록 구현될 수 있다. 그것은 작은 버퍼 구조에 최근에 액세스된 캐시 라인의 웨이 정보를 홀드한다. 도 21은 웨이 테이블(두 번째 좌측 열)과, 8, 16, 32 엔트리를 홀드하는 WDUs(좌측에서 우측으로 다른 3개의 열)과, 웨이 결정(좌측 열) 없이 MALEC 구현을 비교한다. 최상의 WDU 셋업(16 엔트리)의 평균 동적 에너지 소모는 대략 제안된 웨이 테이블 방식보다 2% 높다. 이것에 대한 이유는 2가지가 있다. 첫 번째로, 단일 포트 웨이 테이블에 반해서, WDU는 분석된 MALEC 구현에 의해 병렬로 처리된 3개의 요구까지 서비스하도록 3개의 포트를 필요로 한다. 그것이 태그 크기의 어드레스 필드의 전체 연관 검색을 행함에 따라, WDU 액세스당 에너지가 uWT 액세스와 비슷한다(16 엔트리 WDU에 대해서 딱 40% 더 적다). 두 번째로, 모든 WDU 구성은 웨이 테이블보다 상당히 적은 메모리 액세스를 커버하므로(도 22 참조), "감소된 캐시 액세스" 대신에 "기존의" 증가된 수의 에너지 패널티의 고통을 받는다(세션 5 참조). 이에 반하여, 그것의 작은 사이즈로부터 비롯되는, 방식의 낮은 리키지 전력을 설명하면서, 그것의 전체 에너지 소모가 디폴트 MALEC 구성에 바싹 접근한다(도 21에서 1% 차이보다 적다). 그렇지만, 웨이 테이블의 에너지 소모가 원하는 수의 병렬 메모리 액세스에 의해 광범위하게 영향을 받지 않고, WDU는 잘 확장(scale)되지 않는다. 그러므로, 좀더 공격적인 슈퍼스칼라 프로세서 또는 정교한 벡터 확장은 명확히 WDU에 대해서 웨이 테이블을 선호할 것이다.

[0198] MALEC의 리키지 전력 소모를 직접적으로 향상시키는 하나의 방법은 uTLB 엔트리의 수를 줄여서 uWT 사이즈를 줄이는 것이다. 그렇지만, 이것은 더 많은 WT 액세스로 인해 소모되는 동적 에너지를 증가시키는, uWT 커버리지를 효율적으로 줄인다. 우리의 시뮬레이션은 분석된 MALEC 구현에 대해서 이 트레이드 오프(trade-off)가 uTLB 크기에 관계없이 광범위하게 전체 에너지 소모로 이어진다는 것을 보여준다. 즉, 4, 8, 16 및 32 엔트리 사이의 에너지 차는 단지 1%에 지나지 않는다. 낮은 리키지에 유리한 WT를 완전히 포기하면, 웨이 결정에 의해 커버되는 L1 액세스의 수가 상당히 감소된다. 16 엔트리 uWT를 이용하고 WT를 이용하지 않는 시뮬레이션은 대략 70%의 평균 커버리지를 달성해서, MALEC의 에너지 소모를 5% 증가시켰다.

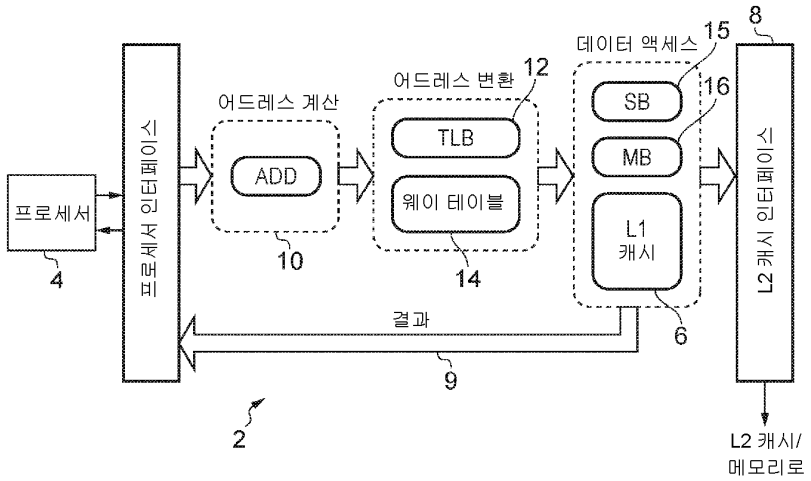
[0199] **7. 결론**

[0200] 이 기술은 고성능 비순차 슈퍼스칼라 프로세서에 대한 L1 캐시 인터페이스에서의 높은 에너지 소모의 문제를 해결한다. MALEC(Multiple Access Low Energy Cache)는 연속적인 메모리 참조가 같은 페이지에 액세스할 가능성이 매우 높다는 관측에 근거한다. 그것은 다수의 로드와 스토어 사이에서의 메모리 어드레스 변환 결과를 공유하고, 스토어 및 머지 버퍼 검색 구조를 간소화하며, 같은 캐시 라인에 액세스하는 로드 중에서 L1 데이터를 공유한다. 또한, 그것은 한 개의 페이지와 맵핑하는 모든 캐리 라인에 대한 웨이 정보를 동시에 제공하는 새로운 웨이 결정 방식을 소개한다. MALEC는 SPEC CPU2000 및 Media-Bench2 벤치마크를 실행하기 위해서 공격적인 비순차 프로세서 및 64Byte 라인을 갖는 33KByte, 4웨이 세트 결합 L1 데이터 캐시를 이용하는 32nm 구현의 시뮬레이션에 의거해서 평가된다. 선택된 MALEC 구성은, 사이클당 1로드 또는 스토어를 서비스하는 것이 가능한, 기본 캐시 인터페이스와 비교해서, 12% 적은 에너지를 이용해서 9% 속도증가를 달성한다. 이에 반하여, MALEC보다 약간 더 높은 성능만을 달성하는 종래의 인터페이스는 베이스라인보다 12% 적은 에너지 대신에 51% 더 많이 소비한다.

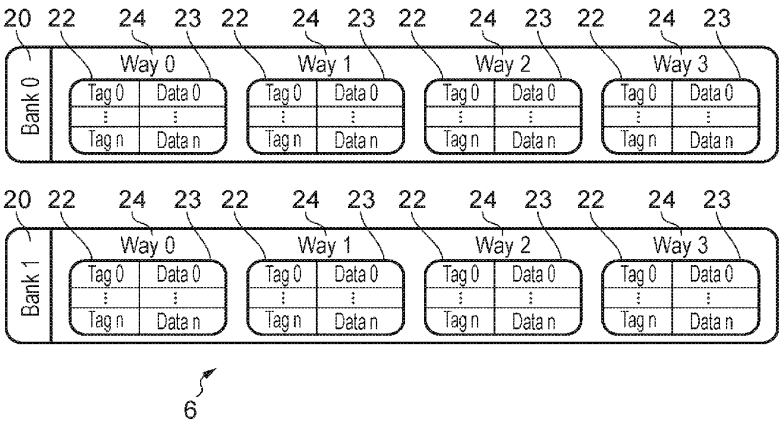
[0201] 본 발명의 예시적인 실시예는 여기서 첨부도면을 참조하여 상세히 설명되었지만, 본 발명은 이들 정확한 실시예에 한정되지 않고, 다양한 변경 및 변형이 첨부된 청구항에 의해 정의된 것처럼 본 발명의 범위 및 정신으로부터 벗어나지 않고 본 발명이 속하는 기술분야의 당업자에 의해 달성될 수 있다는 것을 알 수 있을 것이다.

도면

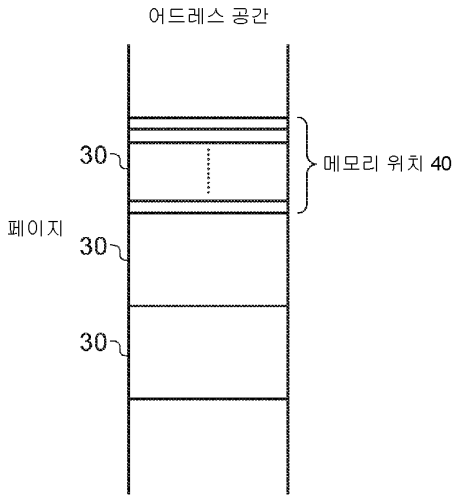
도면1



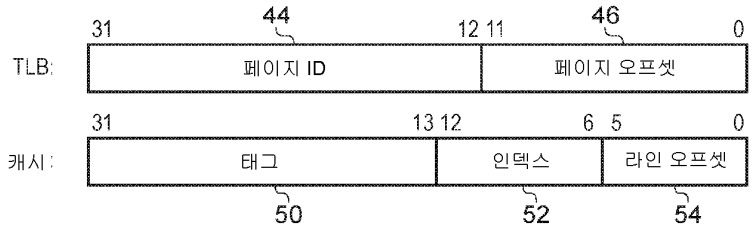
도면2



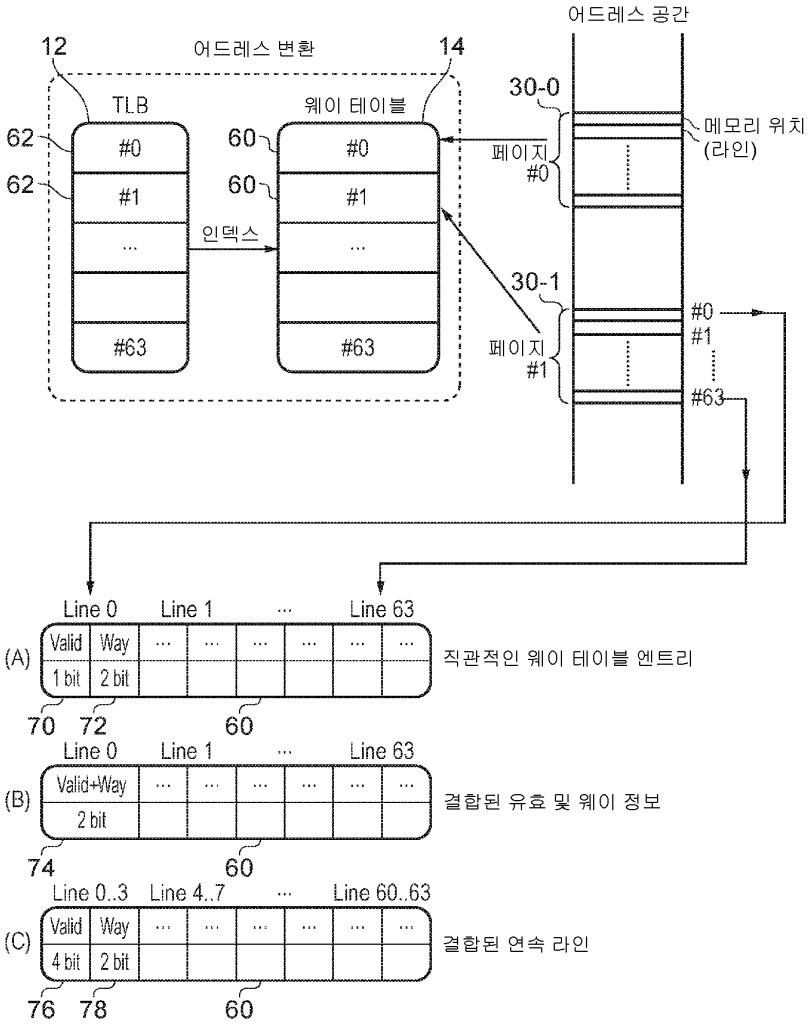
도면3


어드레스 공간
30
30
30
메모리 위치 40
페이지

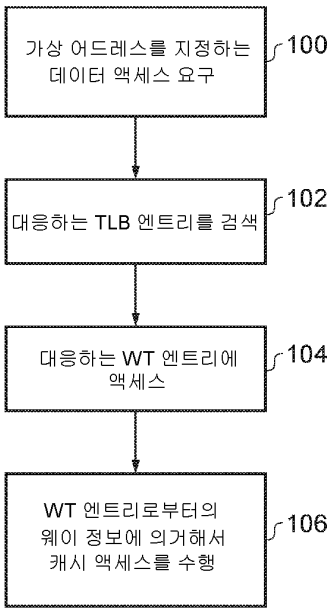
도면4


31
44
12 11
46
0
TLB:
페이지 ID
페이지 오프셋
31
13 12
6 5
0
캐시:
태그
인덱스
라인 오프셋
50
52
54

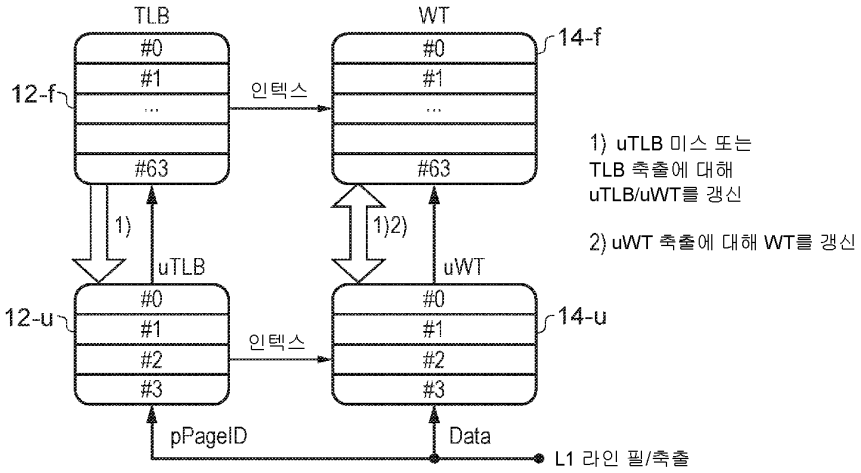
도면5


어드레스 공간
12
어드레스 변환
14
TLB
웨이 테이블
62
62
60
60
#0
#1
...
#63
인덱스
30-0
30-1
메모리 위치
(라인)
페이지
#0
페이지
#1
Line 0
Line 1
Line 63
(A)
Valid
Way
1 bit
2 bit
직관적인 웨이 테이블 엔트리
70
72
60
(B)
Valid+Way
2 bit
결합된 유효 및 웨이 정보
74
Line 0..3
Line 4..7
Line 60..63
(C)
4 bit
결합된 연속 라인
76
78

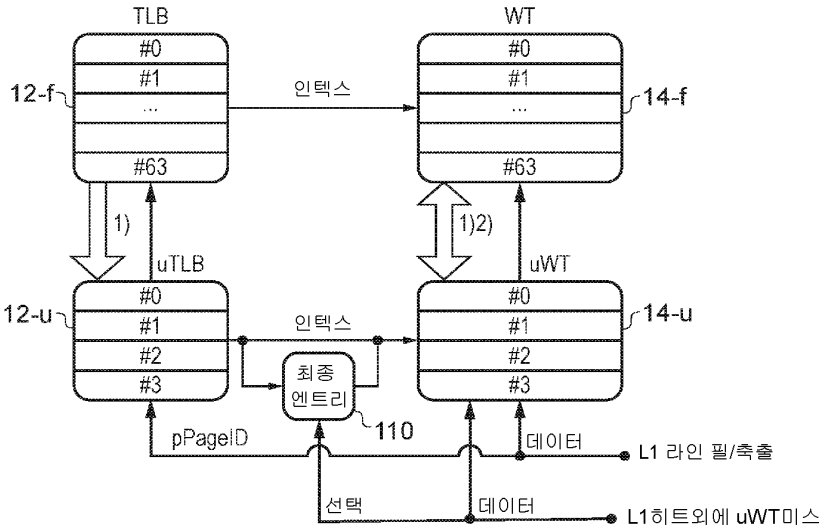
도면6


가상 어드레스를 지정하는
데이터 액세스 요구
100
대응하는 TLB 엔트리를 검색
102
대응하는 WT 엔트리에
액세스
104
WT 엔트리로부터의
웨이 정보에 의거해서
캐시 액세스를 수행
106

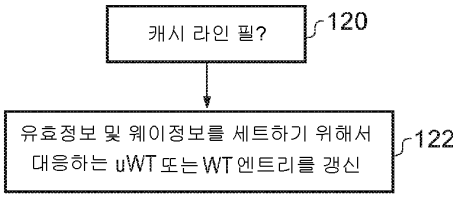
도면7


TLB
WT
#0
#1
...
#63
12-f
14-f
인텍스
1)
1)2)
uTLB
uWT
12-u
14-u
#2
#3
pPageID
Data
L1 라인 필/축출
1) uTLB 미스 또는
TLB 축출에 대해
uTLB/uWT를 갱신
2) uWT 축출에 대해 WT를 갱신

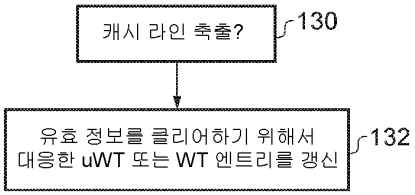
도면8



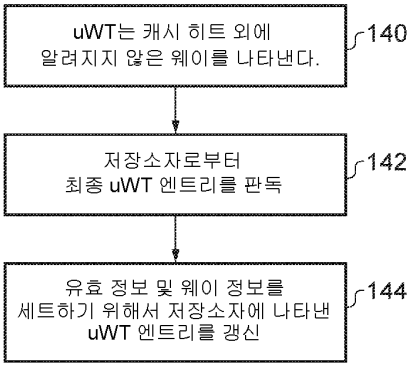
도면9



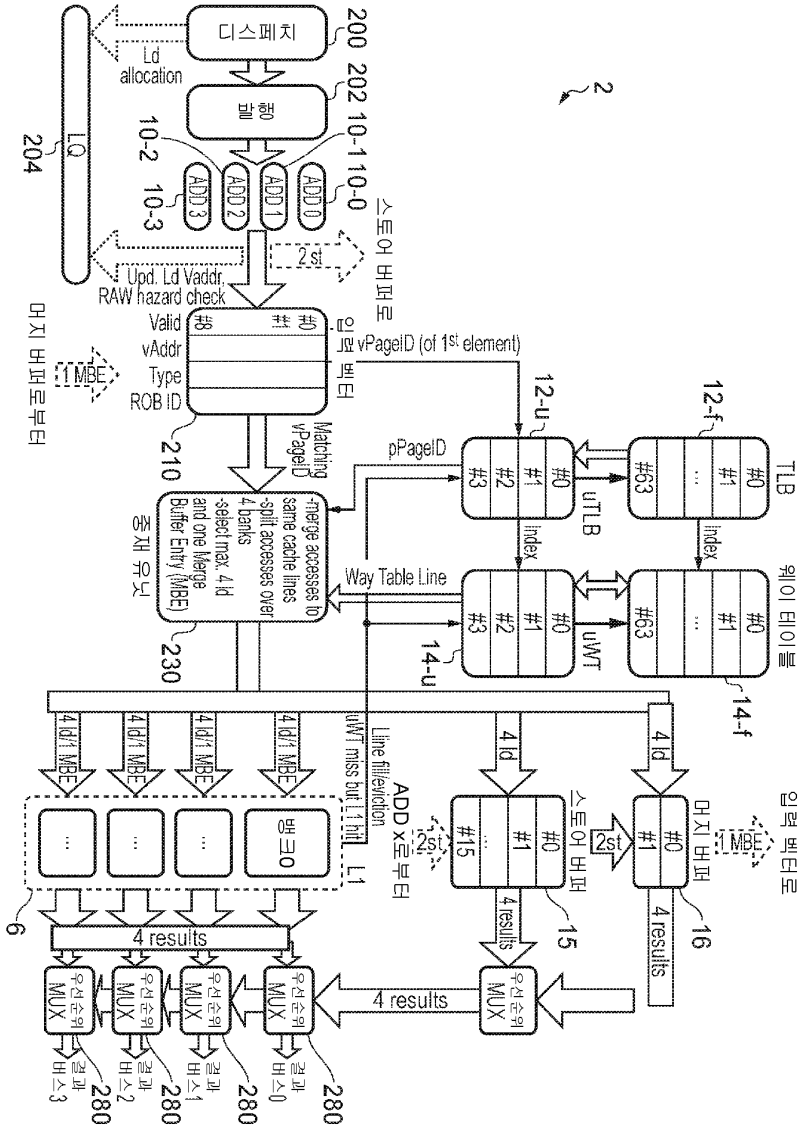
도면10



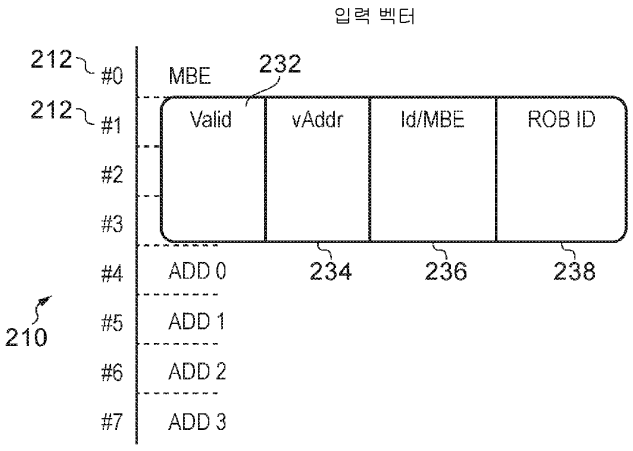
도면11



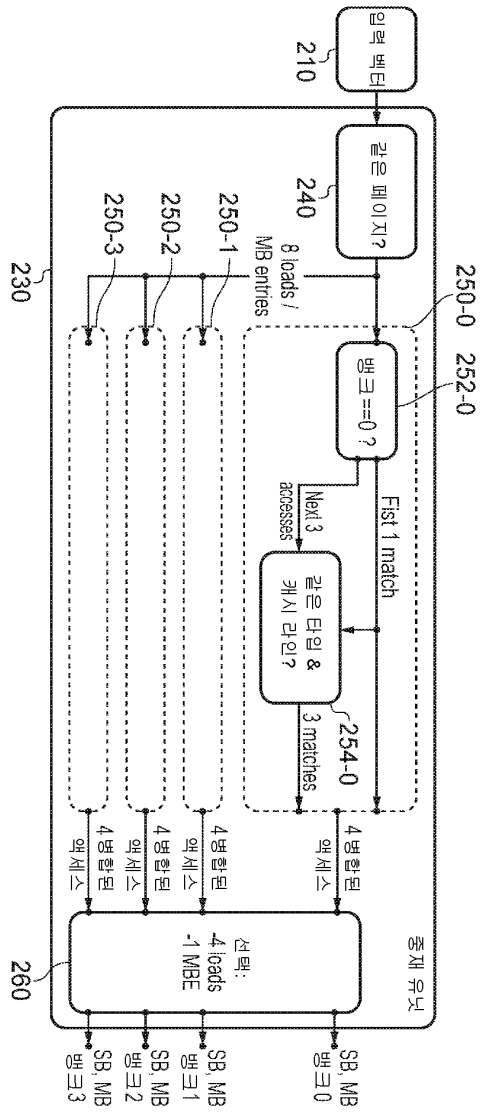
도면12


2
200
디스패치
Ld allocation
202
발행
10-1
10-0
10-2
10-3
ADD 3
ADD 2
ADD 1
ADD 0
스토어 버퍼로
2st
204
LQ
Upd. Ld Vaddr, RAW hazard check
머지 버퍼로부터
Valid
vAddr
Type
ROB ID
1 MBE
#0
#1
#2
#3
입력 벡터
210
Matching vPageID
입력 vPageID (of 1st element)
12-u
12-f
pPageID
uTLB
TLB
#63
Index
웨이 테이블 액세스 유닛
merge accesses to same cache lines
split accesses over 4 banks
select max. 4 ld and one Merge Buffer Entry (MBE)
230
Way Table Line
14-u
uWT
14-f
웨이 테이블
4 ld/1 MBE
uWT miss but L1 hit
Line fill/eviction
L1
4 ld
ADD x로부터
스토어 버퍼
15
머지 버퍼
16
입력 벡터로
6
4 results
우선순위 MUX
결과 버스3
결과 버스2
결과 버스1
결과 버스0
280

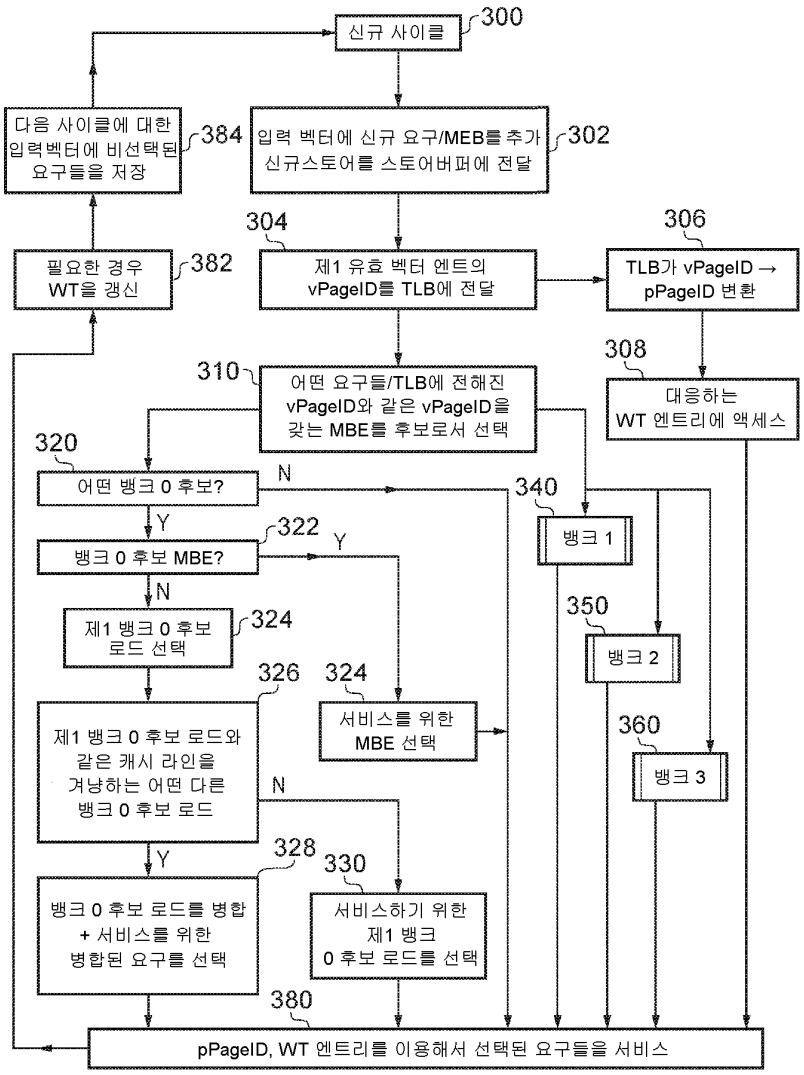
도면13


입력 벡터
212
#0
MBE
232
#1
Valid
vAddr
Id/MBE
ROB ID
#2
#3
#4
ADD 0
234
236
238
210
#5
ADD 1
#6
ADD 2
#7
ADD 3

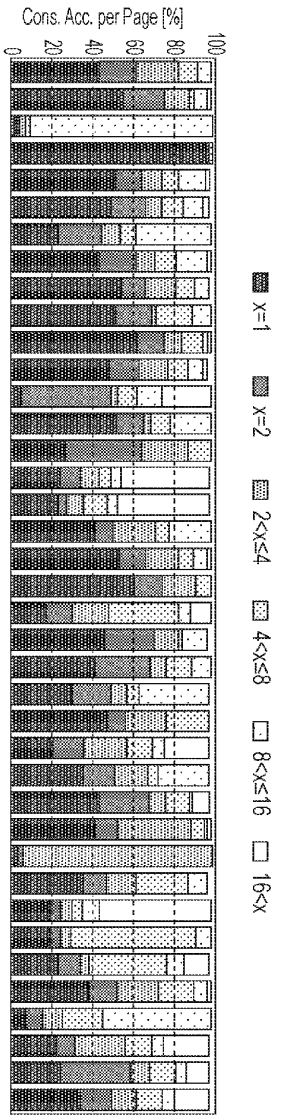
도면14


입력 벡터
210
같은 페이지?
240
8 loads / MB entries
250-1
250-2
250-3
230
250-0
252-0
뱅크 ==0 ?
Next 3 accesses
Fist 1 match
같은 타입 & 캐시 라인?
3 matches
254-0
4 병합 액세스
4 병합 액세스
4 병합 액세스
4 병합 액세스
선택: 4 loads -1 MBE
260
중재 유닛
SB, MB 뱅크 0
SB, MB 뱅크 1
SB, MB 뱅크 2
SB, MB 뱅크 3

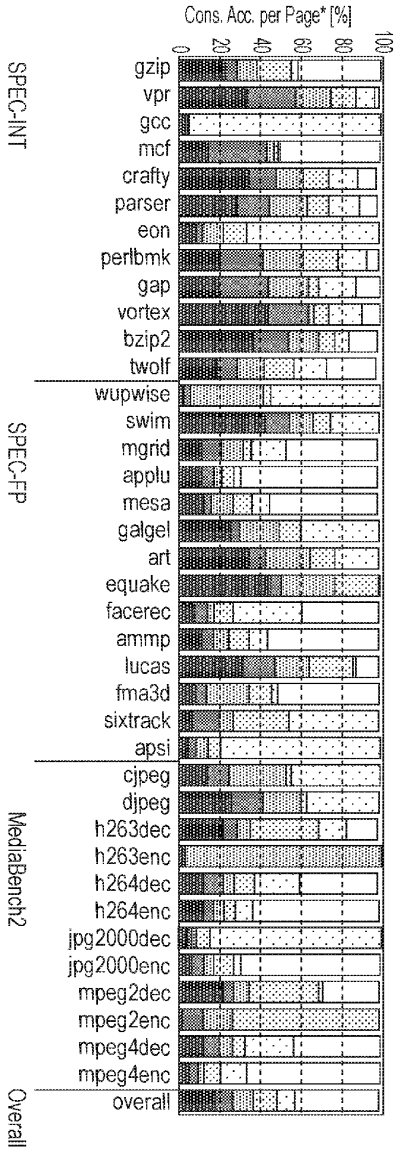
도면15


신규 사이클
300
다음 사이클에 대한 입력벡터에 비선택된 요구들을 저장
384
입력 벡터에 신규 요구/MEB를 추가 신규스토어를 스토어버퍼에 전달
302
304
306
필요한 경우 WT을 갱신
382
제1 유효 벡터 엔트의 vPageID를 TLB에 전달
TLB가 vPageID → pPageID 변환
308
310
어떤 요구들/TLB에 전해진 vPageID와 같은 vPageID을 갖는 MBE를 후보로서 선택
대응하는 WT 엔트리에 액세스
320
어떤 뱅크 0 후보?
N
340
Y
뱅크 1
322
Y
뱅크 0 후보 MBE?
N
350
제1 뱅크 0 후보 로드 선택
324
뱅크 2
326
324
360
서비스를 위한 MBE 선택
제1 뱅크 0 후보 로드와 같은 캐시 라인을 겨냥하는 어떤 다른 뱅크 0 후보 로드
뱅크 3
N
328
330
Y
뱅크 0 후보 로드를 병합 + 서비스를 위한 병합된 요구를 선택
서비스하기 위한 제1 뱅크 0 후보 로드를 선택
380
pPageID, WT 엔트리를 이용해서 선택된 요구들을 서비스

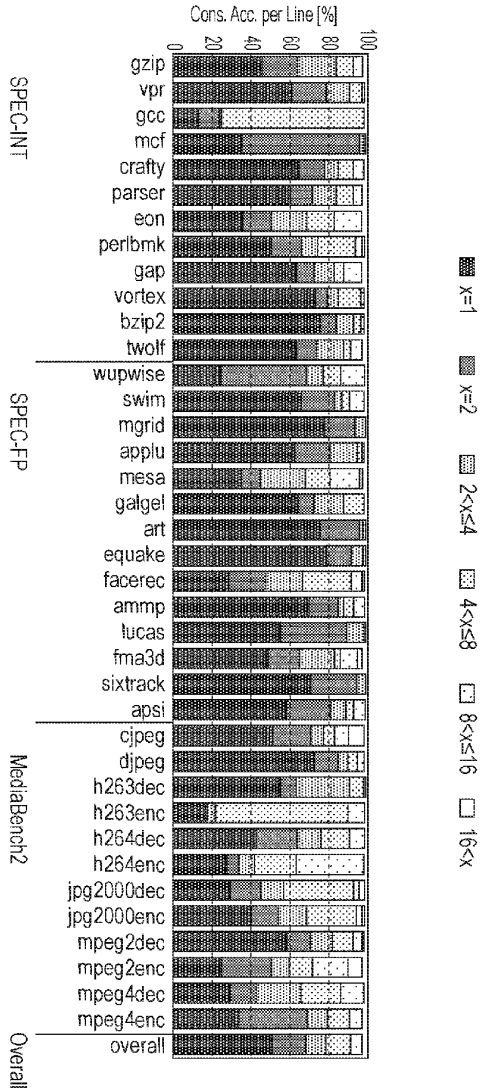
도면16a



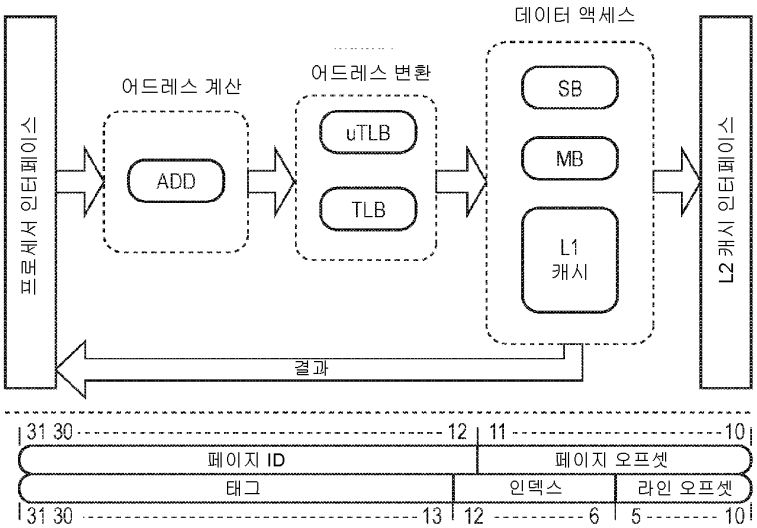
도면16b


Cons. Acc. per Page* [%]
0
20
40
60
80
100
SPEC-INT
gzip
vpr
gcc
mcf
crafty
parser
eon
perlbmk
gap
vortex
bzip2
twolf
SPEC-FP
wupwise
swim
mgrid
applu
mesa
galgel
art
equake
facerec
ammp
lucas
fma3d
sixtrack
apsi
MediaBench2
cjpeg
djpeg
h263dec
h263enc
h264dec
h264enc
jpg2000dec
jpg2000enc
mpeg2dec
mpeg2enc
mpeg4dec
mpeg4enc
Overall
overall

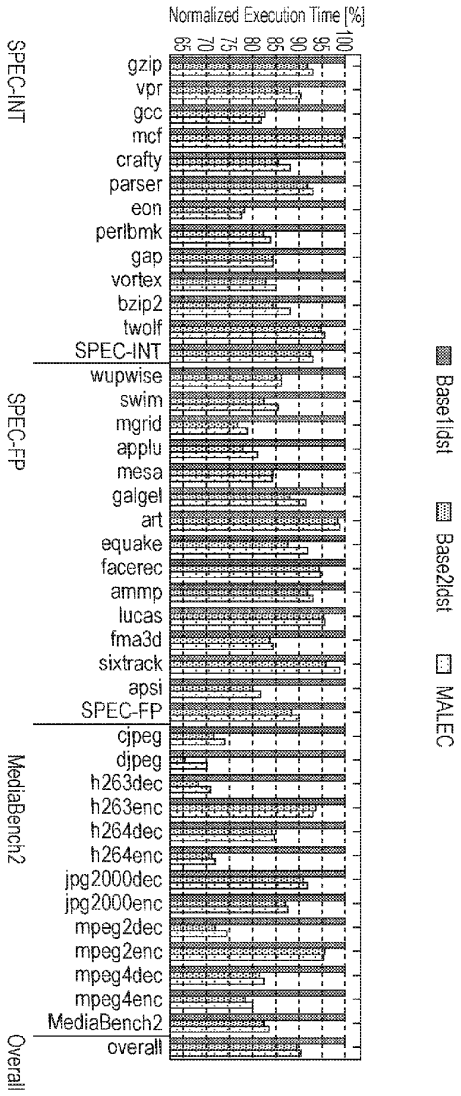
도면16c


Cons. Acc. per Line [%]
0
20
40
60
80
100
SPEC-INT
gzip
vpr
gcc
mcf
crafty
parser
eon
perlbmk
gap
vortex
bzip2
twolf
SPEC-FP
wupwise
swim
mgrid
applu
mesa
galgel
art
equake
facerec
ammp
lucas
fma3d
sixtrack
apsi
MediaBench2
cjpeg
djpeg
h263dec
h263enc
h264dec
h264enc
jpg2000dec
jpg2000enc
mpeg2dec
mpeg2enc
mpeg4dec
mpeg4enc
Overall
overall
x=1
x=2
2<x≤4
4<x≤8
8<x≤16
16<x

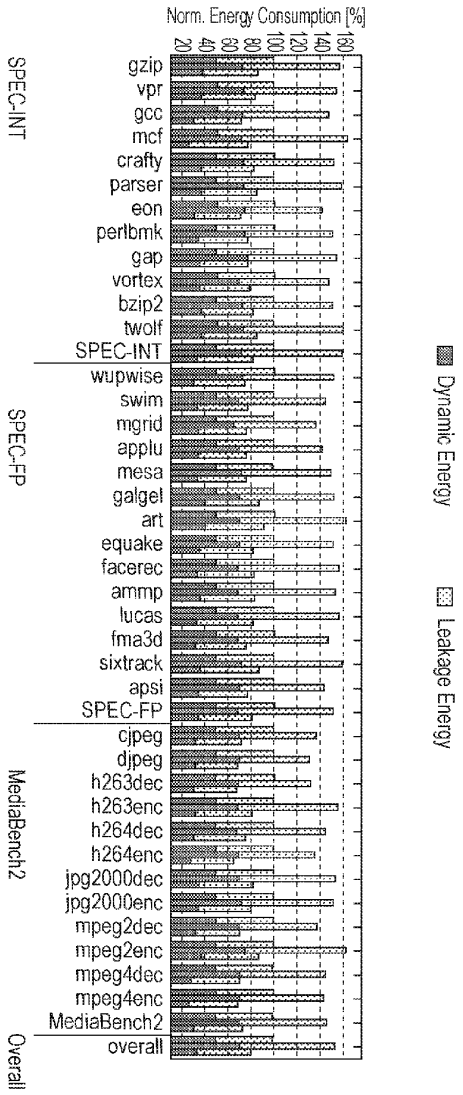
도면17


프로세서 인터페이스
어드레스 계산
ADD
어드레스 변환
uTLB
TLB
데이터 액세스
SB
MB
L1
캐시
L2 캐시 인터페이스
결과
31 30
12
11
10
페이지 ID
페이지 오프셋
태그
인덱스
라인 오프셋
31 30
13
12
6
5
10

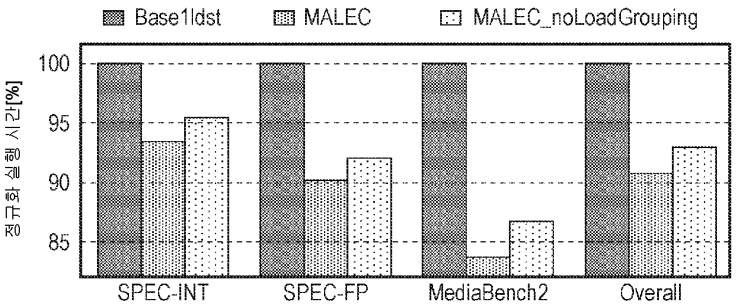
도면18


Normalized Execution Time [%]
65
70
75
80
85
90
95
100
SPEC-INT
SPEC-FP
MediaBench2
Overall
gzip
vpr
gcc
mcf
crafty
parser
eon
perlbmk
gap
vortex
bzip2
twolf
SPEC-INT
wupwise
swim
mgrid
applu
mesa
galgel
art
equake
facerec
ammp
lucas
fma3d
sixtrack
apsi
SPEC-FP
cjpeg
djpeg
h263dec
h263enc
h264dec
h264enc
jpg2000dec
jpg2000enc
mpeg2dec
mpeg2enc
mpeg4dec
mpeg4enc
MediaBench2
overall
Base1ldst
Base2ldst
MALEC

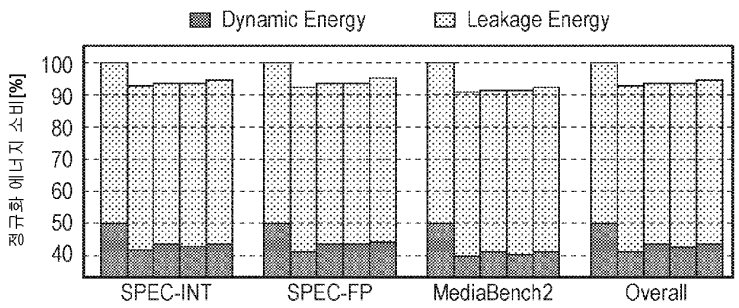
도면19


Norm. Energy Consumption [%]
20
40
60
80
100
120
140
160
SPEC-INT
gzip
vpr
gcc
mcf
crafty
parser
eon
perlbmk
gap
vortex
bzip2
twolf
SPEC-INT
SPEC-FP
wupwise
swim
mgrid
applu
mesa
galgel
art
equake
facerec
ammp
lucas
fma3d
sixtrack
apsi
SPEC-FP
MediaBench2
cjpeg
djpeg
h263dec
h263enc
h264dec
h264enc
jpg2000dec
jpg2000enc
mpeg2dec
mpeg2enc
mpeg4dec
mpeg4enc
MediaBench2
Overall
overall
Dynamic Energy
Leakage Energy

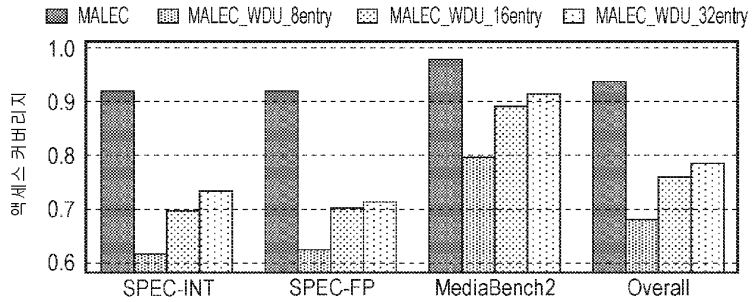
도면20


Base1ldst
MALEC
MALEC_noLoadGrouping
100
95
90
85
SPEC-INT
SPEC-FP
MediaBench2
Overall

도면21


Dynamic Energy
Leakage Energy
정규화 에너지 소비[%]
100
90
80
70
60
50
40
SPEC-INT
SPEC-FP
MediaBench2
Overall

도면22


MALEC
MALEC_WDU_8entry
MALEC_WDU_16entry
MALEC_WDU_32entry
액세스 커버리지
1.0
0.9
0.8
0.7
0.6
SPEC-INT
SPEC-FP
MediaBench2
Overall