Slide 0

5장. 마이크로프로세서의 캐쉬

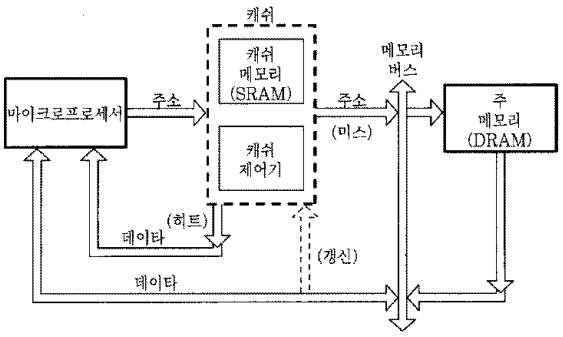
개요

Slide 1

- 캐쉬의 목적 : 빠른 CPU 와 느린 주메모리 사이에 있는 고 속의 RAM 메모리
- 캐쉬 히트 (cache hit) : 액세스하려는 명령어/데이타 가 캐쉬내에 있을 때
  → 고 속의 캐쉬를 액세스
- 캐쉬 미스 (cache miss) : 액세스하려는 명령어/데이타 가 캐쉬내에 없을 때
  → 주 메모리 액세스 → 캐쉬 갱신
- 캐쉬의 구성 (그림 5.1)

개요 (Cont'd)

Slide 2



개요 (Cont'd)

- 캐쉬의 장점
  - 빠른 읽기 (읽기 싸이클 향상)
  - 빠른 쓰기 (캐쉬가 주메모리에 쓸 때 마이크로프로세서 수행됨)
  - 버스 활용도 개선 (외부 버스 싸이클의 필요 감소, 특히 멀티 프로세스 시스템에서)

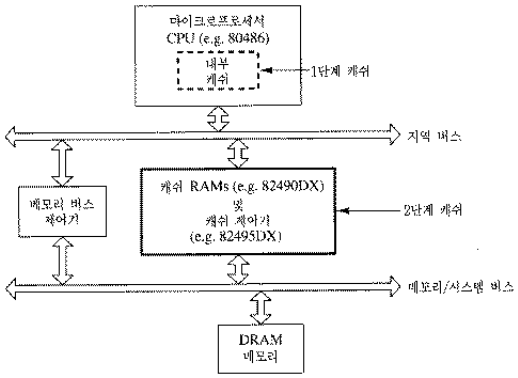
Slide 3

- 캐쉬의 원리 : 사용할 가능성이 높은 것을 고 속의 메모리에 저장
  - 공간적 지역성 (spatial locality) : 현재 수행중인 프로그램 주변에 있는 프로그램은 다음에 액세스될 가능성이 높다.
  - 시간적 지역성 (temporal locality) : 최근에 사용했던 프로그램은 재 사용될 가능성이 높다. (예, 루프)
- 캐쉬의 성능 (히트율 (hit rate))
  - $hit\ rate\ =\ \frac{cache\ hits}{total\ memory\ requests}$
  - $miss\ rate\ =\ 1\ -\ hit\ rate$

시스템에 관한 문제

Slide 4

- 통합 내부 캐쉬 : 명령어/데이타를 캐쉬함 (80486) (그림 5.2 d)



(d) 통일형 (1단계) 캐쉬가 있는 마이크로프로세서 ; 외부캐쉬는 2단계 캐쉬임.

  - 보통 내부캐쉬는 분리형 캐쉬 방식임
  - 일반 사용자의 액세스 불가
  - 슈퍼바이저/시스템 모드로 동작시 시스템 코드만이 액세스

시스템에 관한 문제 (Cont'd)

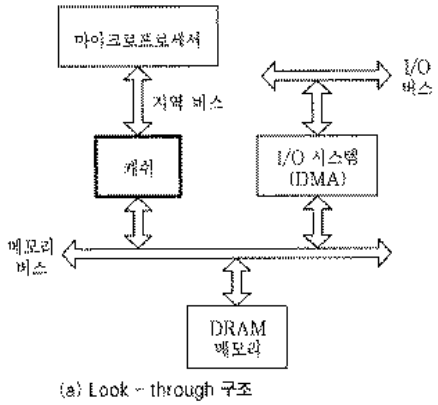
Slide 5

- 물리적 캐쉬 (real cache) : MMU로 부터 변환된 물리주소를 받는 캐쉬 (68040, 80486 →real on-chip cache)
- 가상 캐쉬 (virtual cache) : 변환되지 않은 가상의 주소를 받는 캐쉬 (68020/30, I860, MIPS R4000 →virtual on-chip cache)
  - 가상캐쉬의 문제점 : 동의어 (synonym) 문제

시스템에 관한 문제 (Cont'd)

- Look-through : 직렬 방식 (그림 5.3, 5.4)

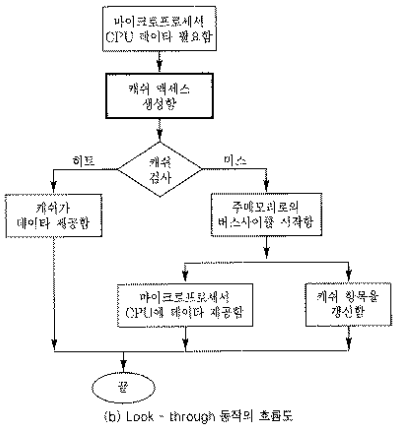
Slide 6



(a) Look - through 구조

시스템에 관한 문제 (Cont'd)

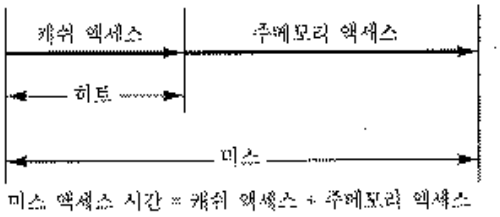
Slide 7



(b) Look - through 동작의 흐름도

시스템에 관한 문제 (Cont'd)

Slide 8



(c) Look - through의 시간 지연

시스템에 관한 문제 (Cont'd)

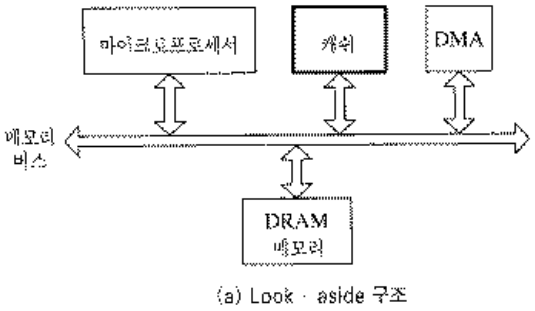
Slide 9

- 캐쉬용 분리된 버스 사용
- 캐쉬 먼저 액세스
- 미스이면 주메모리 액세스
- 단점
  - 두 개의 버스 사용 → 복잡
  - 미스시 버스싸이클이 길어짐
- 장점
  - 캐쉬액세스시 다른 버스 매스터가 주 메모리 액세스 가능

시스템에 관한 문제 (Cont'd)

Slide 10

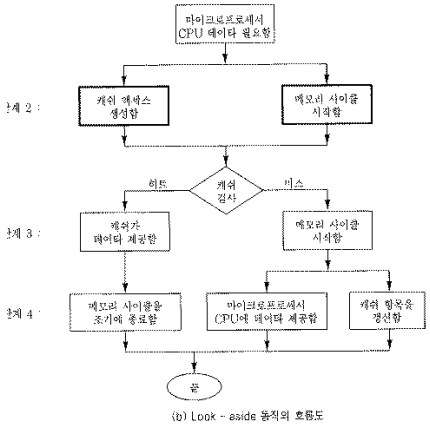
- Look-aside : 병렬 방식 (그림 5.5)



(a) Look - aside 구조

시스템에 관한 문제 (Cont'd)

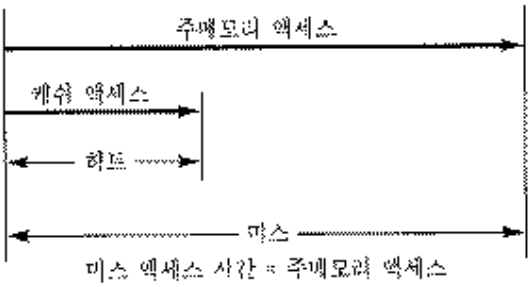
Slide 11



(b) Look - aside 동작의 흐름도

시스템에 관한 문제 (Cont'd)

Slide 12



(c) Look - aside의 시간 작면

- 캐쉬와 주 메모리 가 동일 버스사용
- 캐쉬와 주메모리 액세스 동시 시작
- 캐쉬 히트이면 캐쉬에서 읽고 메모리 동작 중단
- 캐쉬 미스이면 주메모리에서 읽어온 것 사용, 캐쉬 갱신
- 단점 (버스 사용도가 높다)

시스템에 관한 문제 (Cont'd)

Slide 13

  * 캐쉬액세스시 다른 버스 매스터가 주 메모리 액세스 불가능
  * 다른 버스 매스터가 주 메모리 액세스시 캐쉬 사용 불가능
- 장점
  * 한 개의 버스 사용 → 간단
  * 선택사용으로 설계될수 있고, 시스템 재설계없이 캐쉬 제거 가능

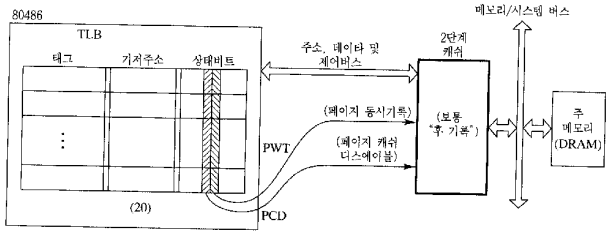
시스템에 관한 문제 (Cont'd)

Slide 14

- 다단계 캐쉬 : 1단계, 2단계 캐쉬 사용
  - 1단계 캐쉬 : 보통 소형 (보통 4K 64K) →내부 캐쉬가 없는 프로세서의 외부 캐쉬
  - 2단계 캐쉬 : 1단계에 비하여 대형 (보통 64K 512K) →내부캐쉬가 있는 프로세서의 외부 캐쉬
  - 80486 에서의 다단계 캐쉬 (그림 5.6)

시스템에 관한 문제 (Cont'd)

Slide 15



PWT = 1: 후기록인 2단계 캐쉬를 동시기록으로 만든다.

PCD = 1: 내부캐쉬는 디스에이블되고, 따라서 2단계 캐쉬도 디스에이블됨. (공유 혹은 캐쉬불가능 데이타인 경우)

PCD = 0: 그 페이지로부터의 데이타 캐싱을 허용함 (비공유 혹은 캐쉬가능 데이타인 경우)

쓰기 방식

Slide 16

- 쓰기 방식
  - 동시기록 (write-through) : 캐쉬 히트로 캐쉬에 데이터를 쓸 때 주 메모리도 갱신
  - 장점
    1) 캐쉬 일관성 (cache coherency) 이 유지됨
    2) 액세스가 간단 (다른 버스 매스터에서)
    3) 구현 쉽고, 신뢰도 높음
  - 단점
    1) 주메모리의 버스 싸이클이 빈번 (버스 사용도 감소, 특히 멀티 프로세스 시스템에서)
    2) 다중 프로세서 구성에서 모든 프로세서의 캐쉬는 해당 캐쉬라인을 무효화 시키거나 새로운 데이터로 갱신 시켜야함
  - 버퍼된 동시 기록 (Buffered write-through) : 캐쉬히트로 캐쉬와 메모리에 데이터를 쓸 때 주 메모리가 사용중이면 내부 버퍼에 쓰고, 버스가 사용되지 않을 때 주 메모리에 씀 (예, 80486 4개의 쓰기 버퍼를 가짐)

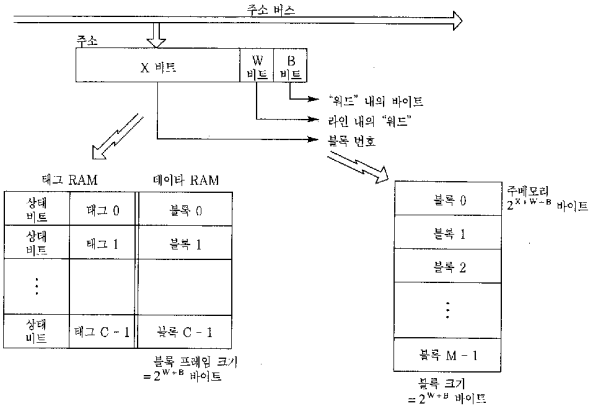
쓰기 방식 (Cont'd)

Slide 17

  - 후기록 (write-back) : 캐쉬 히트로 캐쉬에 데이터를 쓸 때 캐쉬에만 쓰고 나중에 주 메모리 갱신
    * 캐쉬내 해당 라인에 수정비트 (modified bit) 나 오염비트 (dirty bit) 를 세팅함
    * 주 메모리에 쓰는 경우 →쓴후 수정비트 소거됨
      1) 캐쉬미스에 의하여 이 라인이 교체되어야할
      2) 버스가 사용되지 않을 때
      3) 문맥 (context) 교환에 의하여 캐쉬 비우기 (cache flushing) 할 때
    * 장점
      1) 주 메모리 버스 사용이 줄어듦 (버스 사용도 증가)
    * 단점
      1) 캐쉬 일관성이 유지 되지 않음 (다른 버스 매스터가 주메모리 읽을 때 수정비트를 검사해야함) →주메모리 먼저 갱신되고 수정비트 소거후 다른 버스 매스터가 읽기 가능
    * 페이지 별로 기록 방식을 선택하는 프로세서 (68040)
  - 공유 (shared) 메모리는 동시기록을 사용 (캐쉬 일관성 유지)
  - 전용 (private) 메모리는 후기록 방식을 사용

캐쉬의 구성 (그림 5.8)

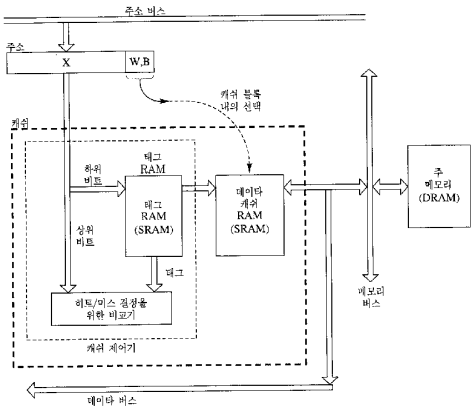
Slide 18



(a) 주메모리의 블록 분할과 캐쉬메모리의 동일크기 블록 프레임 분할

캐쉬의 구성 (그림 5.8) (Cont'd)

Slide 19



(b) 캐쉬의 블록도

캐쉬의 구성 (그림 5.8) (Cont'd)

Slide 20

- 메모리 블록 : 바이트의 주메모리는 바이트의 M개로 분할됨
- 메모리 액세스
  - X 개의 최대 유효 비트 : 메모리 블록번호
  - W 개의 비트 : 블록내의 워드
  - B 개의 비트 : 워드내의 바이트
- 캐쉬
  - C 개의 라인
  - 각 라인당 필드: 캐쉬태그 (태그 RAM에) 와 캐쉬 블록 (캐쉬 데이터 RAM에)
  - 캐쉬 블록의 크기 : 바이트 (주메모리 블록과 같음)
  - 메모리 블록 →캐쉬 블록 으로 매핑됨
  - 캐쉬의 크기 : (데이타 부분만 나타냄)
  - X개의 최대유효 비트 는 캐쉬의 태그 필드와 비교됨
  - 일치 →캐쉬 히트 →캐쉬로 부터 인출
  - W와 B를 이용 데이터 액세스

캐쉬의 구성 (그림 5.8) (Cont'd)

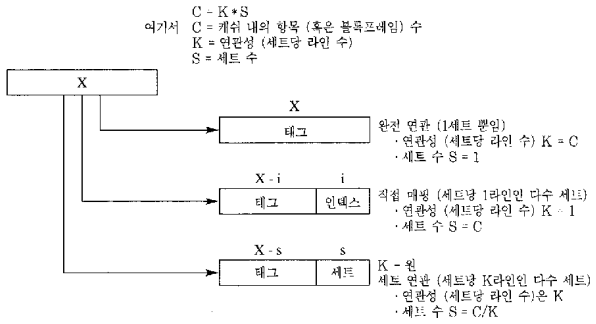
Slide 21

- 불일치 →캐쉬 미스 →캐쉬할당 (갱신) (보통 캐쉬라인 정체가 갱신됨)
- 캐쉬 블록의 크기 > 데이터 버스 넓이 (버스트 모드로 읽음)
- 라인수 : $C = \frac{2^d}{2^e} = 2^{d-e}$ (2^d : 총용량, 2^e : 블록크기)
- 관계식 : $C = K \times S$ (K: 세트당라인수, S: 세트수)

캐쉬의 매핑방법

Slide 22

- 3가지 매핑 방법 (그림 5.9)

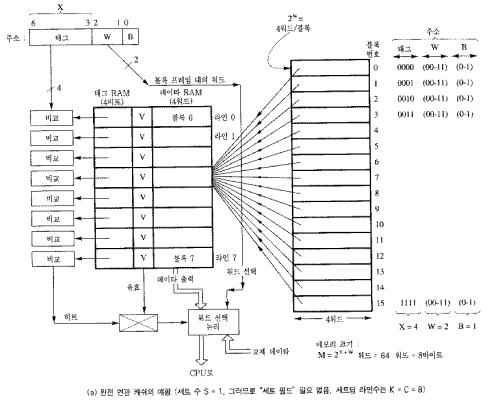


1) $S = 1 \rightarrow K = C$: 완전연관 매핑 (fully associative mapping)
2) $S = C \rightarrow K = 1$: 직접 매핑 (direct mapping)
3) $1 < S < C \rightarrow K = C/S$: K-원 세트연관 매핑 (K-way set-associative mapping)

캐쉬의 매핑방법 (Cont'd)

Slide 23

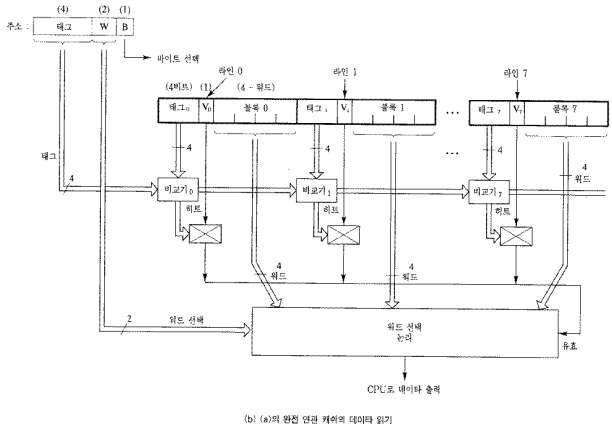
- 완전연관 매핑 (fully associative mapping) (그림 5.10)



(a) 완전 연관 캐쉬의 매핑 (세트 수 S = 1, 그러므로 "세트 필드" 필요 없음, 세트당 라인수는 K = C = 8)

캐쉬의 매핑방법 (Cont'd)

Slide 24



(b) (a)의 완전 연관 캐쉬의 데이타 읽기

- $S = 1, K = C$ →세트 필드 필요 없음
- X개의 최대유효 비트 →태그 필드로 해석됨
- 태그가 일치하고 유효비트가 유효이면 →캐쉬 히트
- 태그가 불일치하거나 유효비트가 무효이면 →캐쉬 미스

캐쉬의 매핑방법 (Cont'd)

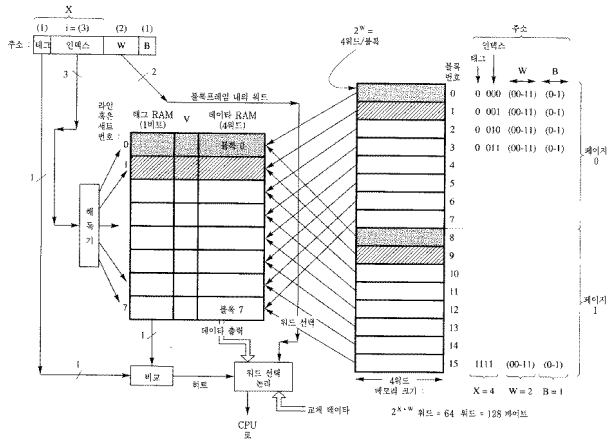
Slide 25

- 장점
  - 모든 태그는 동시에 비교됨
  - 주메모리의 임의의 블록도 캐쉬의 임의의 블록으로 매핑됨
  - 외부해독 회로가 필요없다 (임의로 저장됨으로)
  - 다양한 교체알고리즘 사용가능
  - 쓰래싱 (thrashing) 이 일어나지 않음
- 단점
  - 비교기가 많이 듦으로 비용이 높음
  - 소형 캐쉬에 적합 특수소자를 사용하여 구현됨
  - 정보비트 대 캐쉬라인 전체비트율이 낮음

캐쉬의 매핑방법 (Cont'd)

Slide 26

- 직접매핑 (direct mapping) (그림 5.11)



  - K =1, S=C →각라인은 한세트
  - 주메모리 블록 d 는 캐쉬블록 f 로만 가능 (f = d mod C)

캐쉬의 매핑방법 (Cont'd)

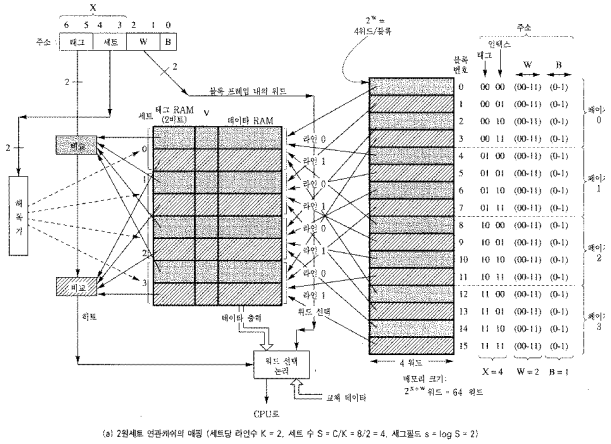
Slide 27

  - 태그 비교기가 하나만 필요하다
  - 장점
    * 비교기가 하나면 됨으로 비용이 적음
    * 교체알고리즘이 간단
  - 단점
    * 교대로 사용되는 두 개의 메모리 블록이 같은 캐쉬블록에 매핑될 때 →thrashing 일어남
    * 미스율이 높다

캐쉬의 매핑방법 (Cont'd)

Slide 28

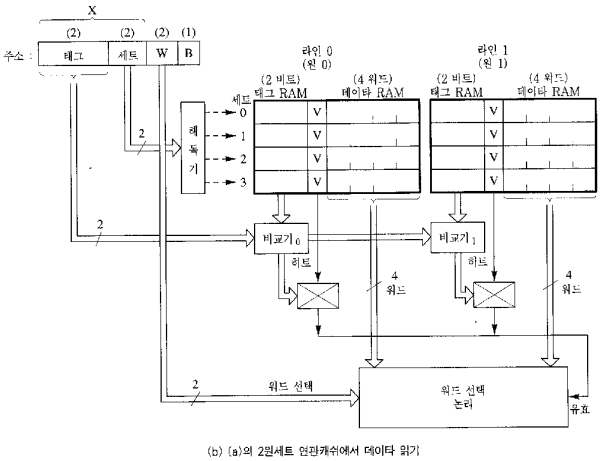
- 세트 연관 매핑 (set associative mapping) (그림 5.12)



(a) 2원세트 연관캐쉬의 매핑 (세트당 라인수 K = 2, 세트 수 S = C/K = 8/2 = 4, 세그필드 s = log S = 2)

캐쉬의 매핑방법 (Cont'd)

Slide 29



(b) (a)의 2원세트 연관캐쉬에서 데이타 읽기

- $1 < K < C$, $S = C/K$
- 최신프로세서에서 많이 사용
- 주메모리 블록 d 는 캐쉬세트 f 로만 가능 (f = d mod S)

캐쉬의 매핑방법 (Cont'd)

Slide 30

- K 개의 라인중 임의의 라인에 저장 가능
- 완전연관 매핑과 직접매핑의 혼합형

캐쉬의 매핑방법 (Cont'd)

Slide 31

- 캐쉬라인의 상태 비트
  - 라인유효 비트 (Line valid bit) : 무효는 주 메모리는 변경, 캐쉬 데이터는 변경 안되었을 때
  - →(메모리 버스를 감시 (snoopy) 하여 해당 라인을 무효화 시키는 회로 필요)
  - 워드 유효 비트 (Word valid bit) : 라인 유효비트대신 블록내의 각 워드 혹은 이중 워드에 대하여 별도의 워드 유효 비트를 두는 프로세서가 있음
    * 태그 히트, 워드 무효 →해당 워드만 단일모드로 갱신 (버스 사이클을 줄임)
  - "워드" 오염/수정 비트 (word dirty/modified bits) : 캐쉬내의 데이터가 오염 (바뀜) 되었음을 표시
    * 데이터 캐쉬에만 사용 : 명령어는 write 되지 않음으로
    * 읽기에서는 무시 : 올바른 데이터가 있음으로
    * 후기록 방식 지원 (나중에 메모리를 갱신할 때 참조할수 있음으로)
  - LRU (least recently used) 필드 : 캐쉬 히트가 있을때마다 갱신됨

캐쉬의 매핑방법 (Cont'd)

Slide 32

- 배타적 소유 비트 (Exclusive ownership bits) : 캐쉬의 데이터가 자신에게만 있는지 아니면 다른 캐쉬에도 있는지 표시
  - EU (exclusive unmodified) : 나만이 갖고 있고 수정되지 않음
  - EM (exclusive modified) : 나만이 갖고 있고 수정됨
  - SU (shared unmodified) : 공유하고 있고 수정되지 않음
  - SM (shared modified) : 공유하고 있고 수정됨
    →캐쉬 일관성에 사용됨
- 액세스권 및 프로세스-ID 필드
  - 액세스권 : 코드/데이타, 사용자/슈퍼바이저 액세스 공간 구별
  - 프로세스-ID : 다른 프로세서 사이에 보호를 제공
- 패리티 비트 : 태그 RAM의 각 바이트마다 패리티를 갖는 프로세서도 있음

캐쉬 갱신 방식

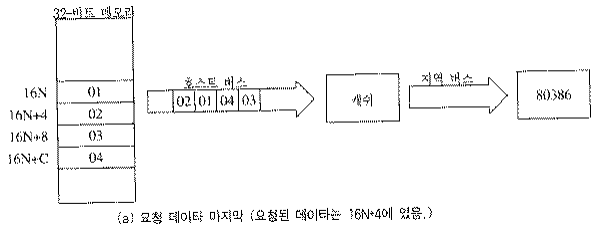
Slide 33

- 완전연관 캐쉬 : 모든 블록으로 할당 가능
  - 무효라인을 교체
  - 모두 유효하면 알고리즘 사용 교체
- 직접 매핑 캐쉬 : 주메모리 블록 d 는 캐쉬의 특정 블록 f 로만 할당 가능 (f = d mod C)
  - 교체 알고리즘 간단
- 세트 연관 캐쉬 : 주 메모리 블록 d는 캐쉬의 특정 세트 s안에 임의의 블록으로 할당 가능 (s = d mod S)
- 캐쉬 할당 : 캐쉬 갱신을 의미
  - 캐쉬 히트시 발생 안함
  - 캐쉬 일기 미스시 항상 수행됨
  - 캐쉬 쓰기 미스시는 대부분의 프로세서가 할당하지 않음 (일부 하는 프로세서도 있음)
- LRU 교체 알고리즘 (그림 5.14)
  - 완전 연관 캐쉬와 세트 연관 캐쉬에서 모든 라인이 유효할 때 사용

캐쉬 갱신 방식 (Cont'd)

Slide 34

- 캐쉬 라인 채우기 기법
  - 요청된 데이터 마지막 (data requested last) (그림 5.15 a): 요청된 데이터를 마지막으로 읽음



(a) 요청 데이타 마지막 (요청된 데이타는 16N+4에 있음.)

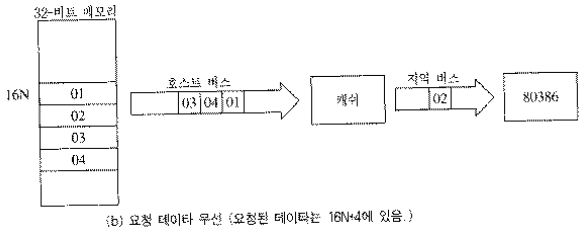
예) 16N+4 를 요청한 16바이트 캐쉬에서 읽기 순서: 16N+8, 16N+C, 16N, 16N+4

* 장점 : 구현이 쉽다.
* 단점 : 라인채우기가 완료될때까지 프로세서 기다림 →미스 벌칙이 높음

캐쉬 갱신 방식 (Cont'd)

Slide 35

  - 요청된 데이터 우선 (data requested first) (그림 5.15 b): 요청된 데이터를 먼저 읽음



(b) 요청 데이타 우선 (요청된 데이타는 16N+4에 있음.)

예) 16N+4 를 요청한 16바이트 캐쉬에서 읽기 순서: 16N+4, 16N, 16N+C, 16N+8

* 장점 : 라인채우기 완료전 프로세서가 동작 가능 →미스벌칙이 감소됨
* 단점 : 구현이 어렵다 →라인의 나머지 채울 때 제2의 읽기 처리 어려움

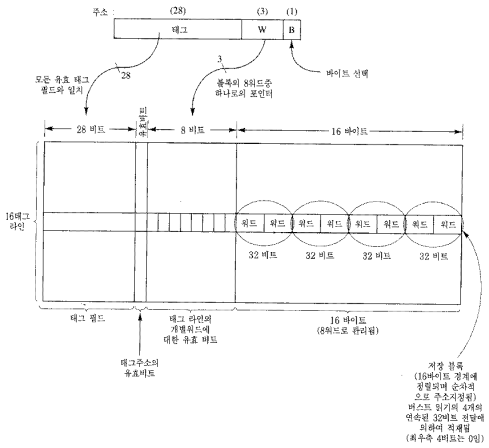
  - 라인 버퍼 캐싱 (line buffer caching) : 버퍼를 사용 프로세서가 바로 사용 가능하게 함

    예) 16N+4, 16N+C 를 읽을 때 16N+C 를 라인버퍼에서 읽기 가능

마이크로프로세서의 캐쉬 예

Slide 36

- 완전연관 매핑 : Zilog Z80000 (그림 5.16)



- 대부분의 프로세서에서는 사용되지 않음
- 읽기 동작

마이크로프로세서의 캐쉬 예 (Cont'd)

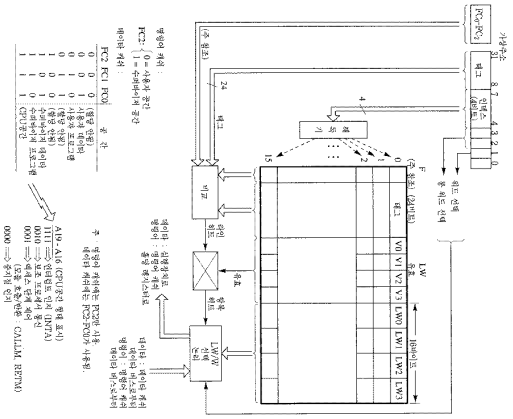
Slide 37

  - * 태그 히트, 워드 히트 →동일칩 캐쉬 액세스
  - * 이외 →주 메모리 액세스
  - * 태그 미스 →버스트 모드 액세스 LRU 알고리즘으로 교체
  - * 태그 히트, 워드 미스 →해당 워드 읽어옴
- 쓰기 동작
  - * 태그히트 →동시 기록
  - * 태그 미스 →주 메모리만 갱신

마이크로프로세서의 캐쉬 예 (Cont'd)

Slide 38

- 직접 매핑 : Motorola 68030 (그림 5.17)



  - 가상 256 바이트 분리된 직접 매핑 캐쉬
  - 24 비트의 태그 필드
  - 4개의 이중워드 유효비트

마이크로프로세서의 캐쉬 예 (Cont'd)

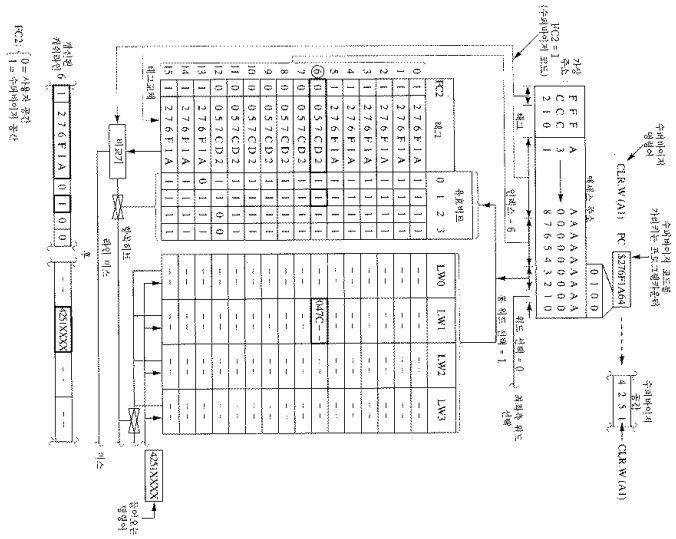
Slide 39

  - 4개의 이중워드로 (주소 4비트 필요) 구성된 16개의 라인 (주소 4비트 필요)
  - 명령어 캐쉬 : 액세스 필드 (FC2 만 사용 : 사용자 코드/슈퍼바이저 코드 구별)
  - 데이터 캐쉬 : 액세스 필드 (FC0 FC2 사용 : 슈퍼바이저 데이터공간 (101) 사용자 데이터공간 (001) 구별
  - 캐쉬히트
    * 액세스 공간 히트
    * 태그 히트
    * 워드 히트
  - 캐쉬 미스
    * 액세스 공간 미스
    * 태그 미스
    * 워드 미스
  - 동시기록 캐쉬
    * CDIS# (Cache disable) 입력핀 : 두 캐쉬를 모두 사용안함
    * CIIN# (Cache inhibit) 입력핀 : 캐싱 금지 핀
    * COUT# (Cache inhibit out) 출력핀 : 외부캐쉬 금지 시킴

마이크로프로세서의 캐쉬 예 (Cont'd)

Slide 40

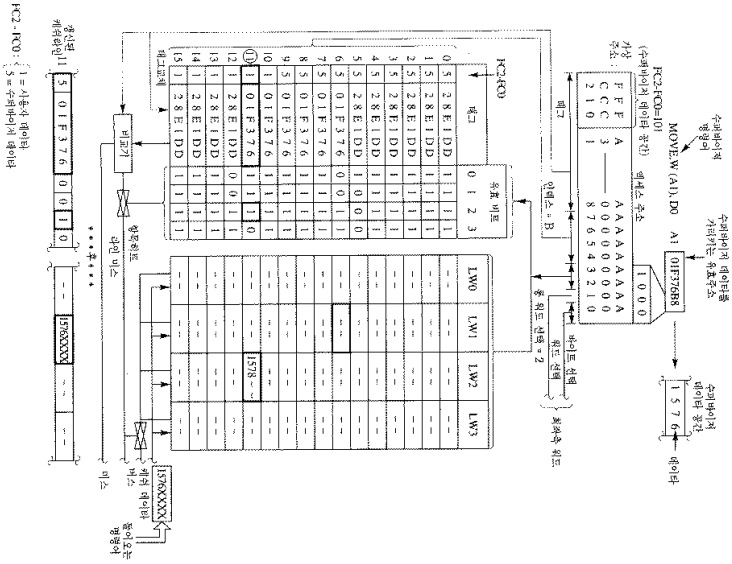
* 명령어 캐쉬의 예 (그림 5.18)



마이크로프로세서의 캐쉬 예 (Cont'd)

Slide 41

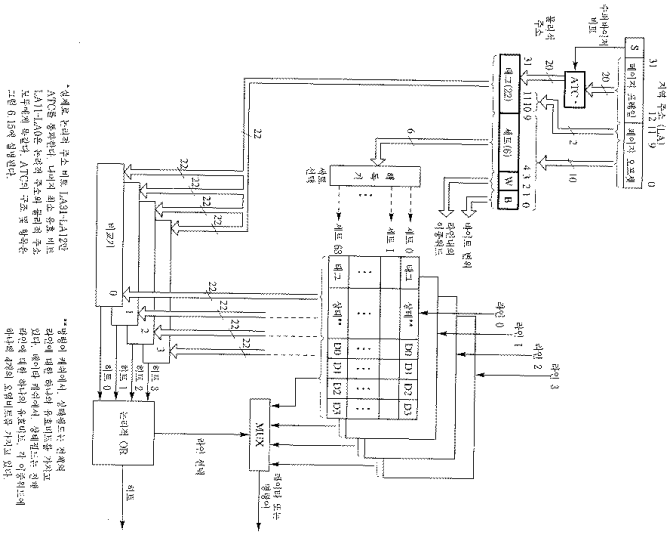
* 데이터 캐쉬의 예 (그림 5.19)



마이크로프로세서의 캐쉬 예 (Cont'd)

Slide 42

- 세트 연관 매핑 : Motorola 68040 (그림 5.21)



마이크로프로세서의 캐쉬 예 (Cont'd)

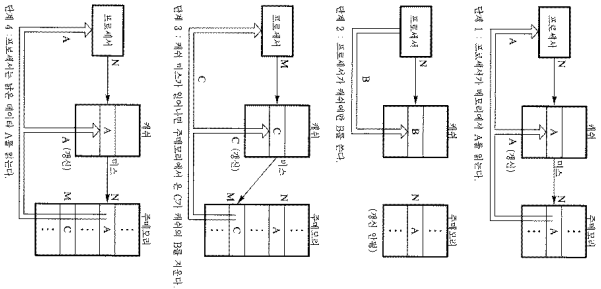
Slide 43

- 물리 4원 4K 분리된 세트 연관 캐쉬
- 22비트 태그
- 명령어 캐쉬 : 액세스 필드 (FC2 만 사용 : 사용자 코드/슈퍼바이저 코드 구별)
- 데이터 캐쉬 : 액세스 필드 (FC0 FC2 사용 : 슈퍼바이저 데이터공간 (101) 사용자 데이터공간 (001) 구별
- 라인 유효 비트 만 사용 →캐쉬 미스시 라인 전체 갱신
- 데이터 캐쉬에 4개의 오염 비트 사용
- 동일칩 버스 감시회로 →캐쉬 일관성 유지
- 동시기록모드 및 후기록 모드 지원
- 데이터 캐쉬는 페이지별로 두 모드 중 하나 세팅 가능

캐쉬 일관성 문제

Slide 44

- 발생 : 주메모리와 하나 이상의 메모리에 동시에 존재하는 데이터
  - 다른 버스 매스터가 주메모리만 변경
  - 프로세서가 자기 캐쉬만 변경 (후기록 방식)
  - 두 개의 캐쉬가 다른 데이터를 가지고 있을 때 (다중 프로세서 시스템에서)
    * 공유 쓰기 가능 메모리를 매스터가 액세스 할수 있을 때
    * 읽기 전용이나 비공유 자료는 안심하고 캐쉬할수 있음
  - 낡은 데이터 문제 (그림 5.22)



캐쉬 일관성 문제 (Cont'd)

Slide 45

* 단일 프로세서나 다중 프로세서에서 모두 발생
* 캐쉬에만 쓸 때 (후 기록 방식) →주 메모리 낡은 데이터
* 단일 프로세서에서 피하는 방법 : 오염비트를 사용하거나 동시기록 방법을 사용
* 다중 프로세서에서 피하는 방법
  · 소프트웨어적인 해결
    1) 가장 쉬운 방법 : 공유된 메모리는 캐쉬되지 않도록함, 페이지마다 공유/비공유 속성을 붙임
    2) 읽기 전용 공유 페이지 만 캐쉬
    3) 디렉토리 구현 : 모든 캐쉬의 블록상태를 기록
  · 하드웨어적인 해결
    1) 캐쉬 가능/불가능 메모리 : 특정 영역을 캐쉬 불가능 으로 정하여 캐쉬하지 않음, 외부 해독논리회로 필요 (그림 5.23)
    2) 공유캐쉬 이용 : 모든 버스 매스터가 같은 캐쉬를 사용하여 메모리 액세스 (그림 5.24) 적은 매스터 수일 때 사용

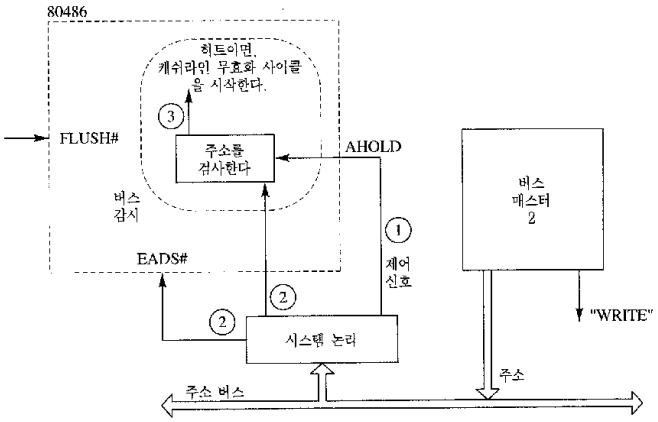
캐쉬 일관성 문제 (Cont'd)

Slide 46

3) 버스 감시 (bus snooping) : 대형 다중 프로세서에서 사용
   1. (i) 시나리오
      (a) 후기록 모드 지역캐쉬에 기록
      (b) 캐쉬 제어기는 메모리/시스템 버스 읽기 감시
      (c) 그 주소가 캐쉬 주소와 일치 하면 메모리 읽기 금지하고 자신이 데이터 공급
      (d) 주 메모리 갱신
   2. (ii) 시나리오
      (a) 동시기록 모드 지역캐쉬에 기록
      (b) 캐쉬 제어기는 메모리/시스템 버스 쓰기 감시
      (c) 그 주소가 캐쉬 주소와 일치하면 캐쉬의 해당 라인 무효화 혹은 갱신
   3. (iii) 80486 의 동일칩 감시 하드웨어 (그림 5.25) , 입출력되는 주소 버스

캐쉬 일관성 문제 (Cont'd)

Slide 47



① AHOLD (주소 홀드) : 80486은 다음 클락 사이클에서 자신의 주소출력을 디스에이블 시킨다. (AHOLD신호가 있는 동안 정상적인 데이타 활동이 계속될 수 있다.)

② 시스템 논리는 80486에 주소를 보내고 EADS# (외부주소)를 보낸다.

③ 마이크로프로세서 (캐쉬 제어기)는 주소를 캐쉬 태그와 비교하여, 일치하면 캐쉬라인을 무효화 시킨다.

캐쉬 일관성 문제 (Cont'd)

Slide 48

4) M.E.S.I 하드웨어 캐쉬 일관성 프로토콜

1. 인텔 채용 방법

(a) M (Modified) : 이 캐쉬에만 사용가능한 라인이 수정되었음

(b) E (Exclusive) : 수정되지 않은 배타적 라인

(c) S (Shared) : 다른 캐쉬와 공유되는 라인

→동시기록 방법을 수행

→다른 캐쉬의 해당 라인을 무효화 시킴

(d) I (Invalid) : 캐쉬의 라인이 무효함 →읽기 캐쉬 미스, 쓰기 동시 기록

(e) 기본 상태 전이 (그림 5.26)

캐쉬 일관성 문제 (Cont'd)

Slide 49

현재 상태	동 작	새 상태	메모리 버스 활동
M	읽기	M	없음
	쓰기	M	없음
	감시	S	후기록
E	읽기	E	없음
	쓰기	M	없음
	감시	S	없음
S	읽기	S	없음
	쓰기	E	동시기록
	감시	S	없음
I	읽기	E	라인채우기
	쓰기	I	동시기록
	감시	I	없음

(a) 표 [21] © 1992. IEEE.