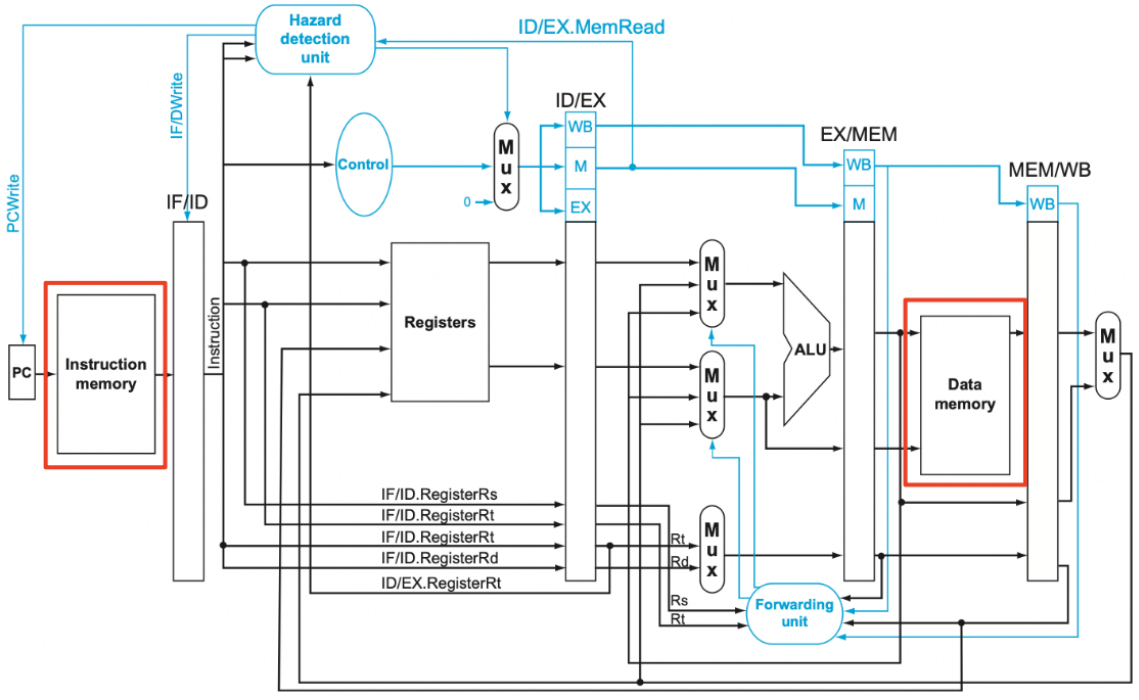
Computer Architecture (ENE1004)

▼ Lec 17

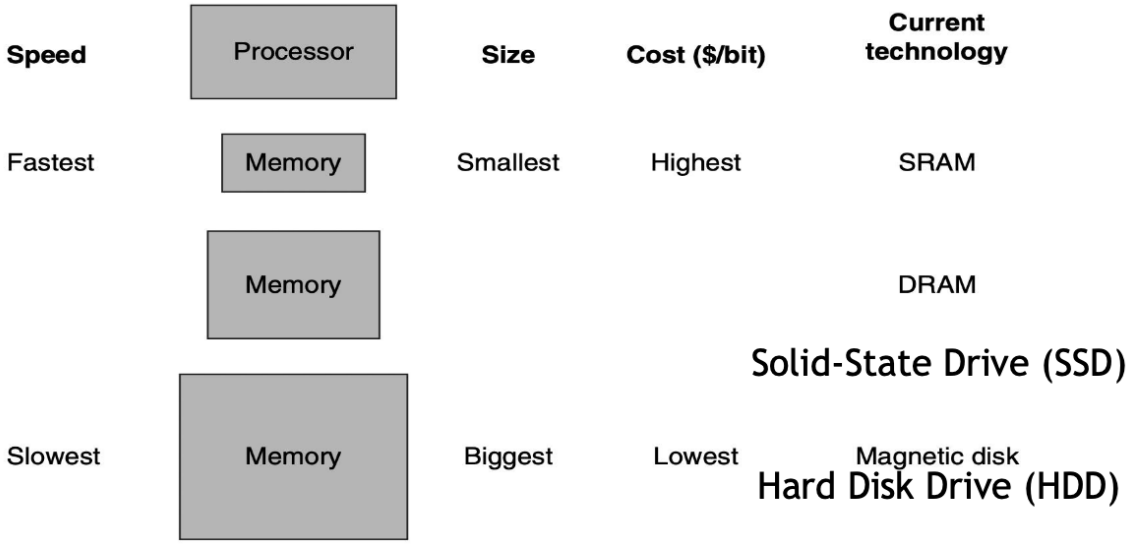
Lec 17: Large and Fast : Exploting Memory Hierarchy 1 (메모리 계층 구조 살펴보기 1)

Ideal Memory System (이상적인 메모리 시스템)



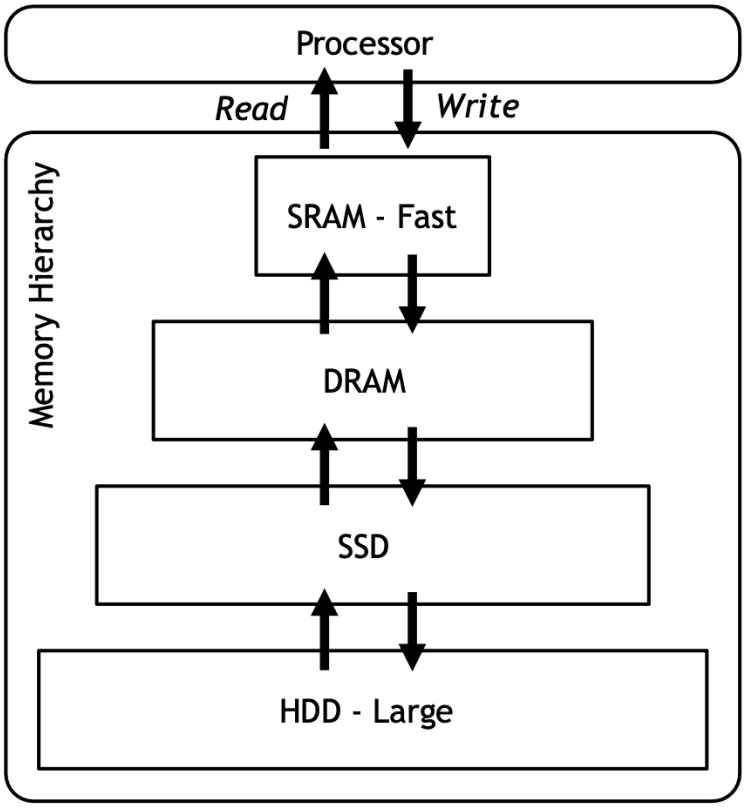
- All the data (program, data) are kept in the memory (모든 데이터(프로그램, 데이터)는 메모리에 보관된다.)
    - Load and store instructions access (read from and write into) the memory (메모리 접근 (읽기 및 쓰기)를 위한 명령어 Load 및 store)
    - From the earliest days, programmers have wanted “unlimited amounts of fast memory” (초창기부터 프로그래머들은 “무제한의 빠른 메모리”를 원했다)
    - Fast memory (performance angle) (빠른 메모리 (성능 각도))
    - Large memory (capacity angle) (대용량 메모리 (용량 각도))

Current Memory Technologies



- Comparison: SRAM vs DRAM vs Solid-State Drive (SSD) vs Hard Disk Drive (HDD)
  - Cost ($/bit) comparison: SRAM > DRAM > SSD > HDD
  - Performance (read/write speed) comparison: SRAM > DRAM > SSD > HDD
  - Capacity (size) comparison: SRAM < DRAM < SSD < HDD
- There is NO single memory technology that is both large and fast (용량이 크고 빠른 단일 메모리 기술은 없다.)
  - However, modern computers provide an “illusion” of the large and fast memory (그러나 최신 컴퓨터는 크고 빠른 메모리에 대한 “환상”을 제공한다.)

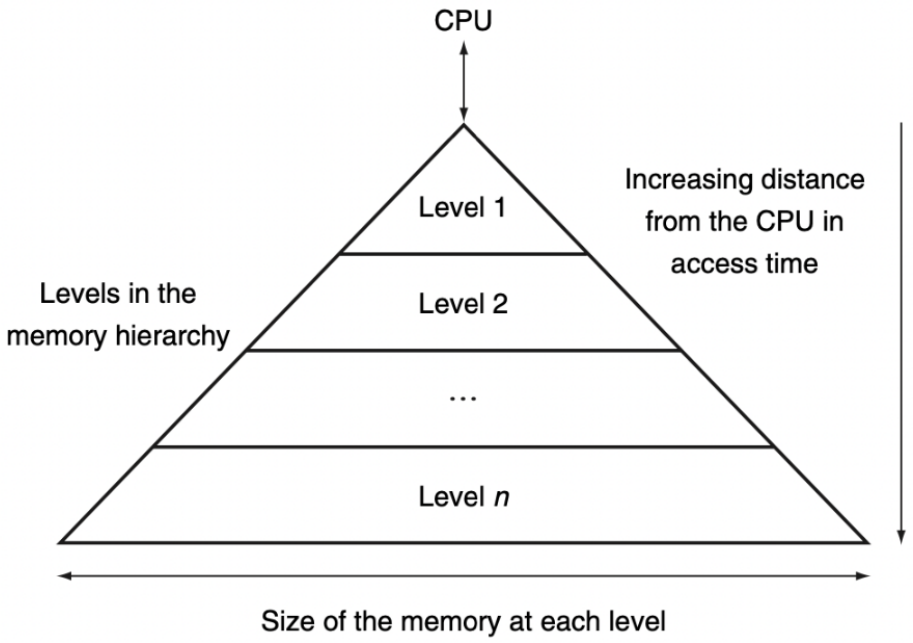
Memory Hierarchy (1) : 메모리 계층 구조



- Memory system includes different memory technologies in a hierarchical fashion (메모리 시스템에는 계층적 방식으로 다양한 메모리 기술이 포함되어 있다.)
    - At the lowest level, the largest/slowest memory (가장 낮은 레벨에서는 가장 큰 / 가장 느린 메모리)
    - At the first level, the fastest/smallest memory (첫 번째 레벨에서는 가장 빠른 / 가장 작은 메모리)
    - As the distance from the processor increases, both the size and the access time increase (프로세스와의 거리가 멀어질수록 크기와 엑세스 시간이 모두 증가한다.)
- The data is similarly hierarchical (데이터도 마찬가지로 계층 구조가 있다.)
    - All the data is stored at the lowest level (모든 데이터는 가장 낮은 레벨에 저장된다.)
    - A level closer to the processor is generally a “subset” of any level further away (프로세스에 더 가까운 레벨은 일반적으로 더 먼 레벨의 “하위 집합”이다.)

- The subset can change over time by sending data upwards/downwards between any two level (하위 집합은 두 레벨 간의 데이터를 위/아래로 전송하여 시간이 지남에 따라 변경될 수 있다.)
- The smallest subset is stored at the first level (가장 작은 하위 집합은 첫 번째 레벨에 저장된다.)
- Processor can directly access only the subset data at the first level (프로세스는 첫 번째 레벨의 하위 집합 데이터에만 직접 엑세스 할 수 있다.)

Memory Hierarchy (2)

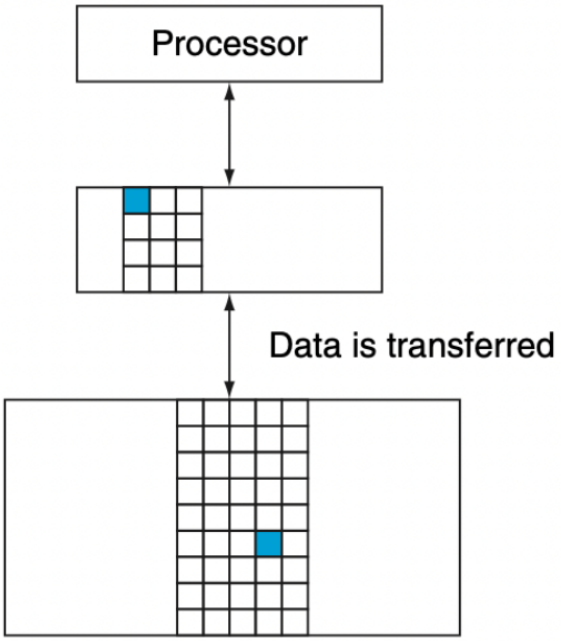


- Processor accesses only the level-1 memory (프로세서가 레벨 1 메모리에만 접근)
  - Processor feels like it works with the fastest memory (프로세서가 가장 빠른 메모리에서 작동하는 것처럼 느껴짐)
- Processor can access all the data set in the level-n memory (프로레서는 레벨 n 메모리에 있는 모든 데이터 세트에 접근할 수 있음)
  - Any data can be sent upwards to the level-1 memory from level-n memory (모든 데이터를 레벨 n 메모리에서 레벨 1 메모리로 전송할 수 있음)
  - Processor feels like it works with the largest memory (프로세스가 가장 큰 메모리에서 작동하는 것처럼 느껴짐)

Memory Hierarchy (3)

- A level closer to the processor is generally a “subset” of any level further away (프로세스에 더 가까운 레벨은 일반적으로 더 멀리 있는 레벨의 “하위 집합” 개념이다.)
- This concept is based on the principle of locality in the memory accesses (이 개념은 메모리 접근의 지역성 원칙에 기반한다.)
    - Two different types of locality are observed in real-world applications (실제 애플리케이션에서는 두 가지 유형의 지역성이 관찰된다.)
- Temporal locality (locality in time) (시간적 지역성 (시간 내 지역성))
    - If a data item is referenced (accessed), it will tend to be referenced again soon (데이터 항목이 참조(접근)되면 곧 다시 참조되는 경향이 있다.)
    - So, if such a data item is stored in an upper level, we can reduce the access times to it (따라서 이러한 데이터 항목이 상위 레벨에 저장되어 있으면 접근 시간을 줄일 수 있다.)
    - In a for loop, the variable for iterations is referenced again in a short period of time (for 루프에서 반복을 위한 변수는 짧은 시간 내에 다시 참조된다.)
- Spatial locality (locality in space) (공간적 지역성 (공간 내 지역성))
    - If an item is referenced, items whose addresses are close by will tend to be referenced soon (항목이 참조되면 주소가 가까운 항목이 곧 참조되는 경향이 있다.)
    - So, if data items in the neighborhood are stored in an upper level, we can reduce the access times to them (따라서 이웃에 있는 데이터 항목을 상위 레벨에 저장하면 해당 항목에 대한 접근 시간을 줄일 수 있다.)
    - In an array, neighboring elements are usually referenced together in a short period of time (배열에서 인접한 요소는 일반적으로 짧은 시간 내에 함께 참조된다.)

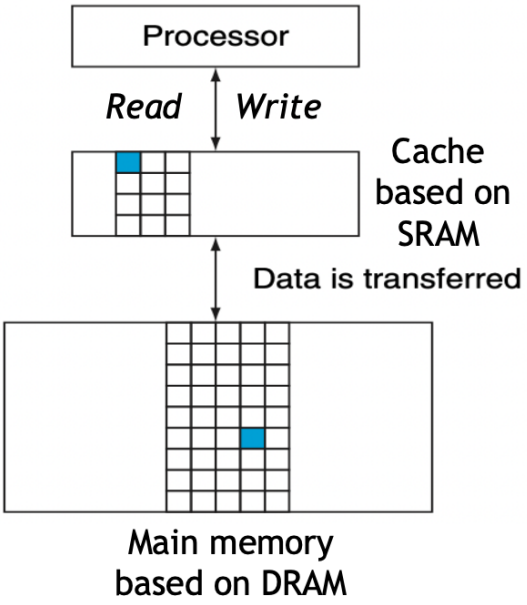
In Every Pair of Levels (모든 레벨 쌍)



- Every pair of levels in the hierarchy can be thought of as having an upper and lower level (계층 구조의 모든 레벨 쌍은 상위 레벨과 하위 레벨이 있는 것으로 생각할 수 있다.)
    - Data is copied between only two adjacent levels at a time (데이터는 한 번에 인접한 두 개의 레벨 사이에만 복사된다.)
- The minimum unit of information that can be either present or not present in the two levels is called “block” or “line” (두 레벨에 존재하거나 존재하지 않을 수 있는 최소 정보 단위를 “블록” 또는 “라인”이라고 한다.)
- If the data requested by the processor appears in some block in the upper level, the request is called a “hit” (프로세서가 요청한 데이터가 상위 레벨의 일부 블록에 나타나면 해당 요청을 “히트”라고 한다.)
    - It can be serviced quickly from the upper-level faster memory (상위 레벨의 빠른 메모리에서 빠르게 서비스 할 수 있다.)
    - The time to access the upper level memory is “hit time” (상위 레벨 메모리에 접근하는 시간을 “히트 시간”이라고 한다.)
- If the data is not found in the upper level, it is called “miss” (상위 레벨에서 데이터를 찾지 못하면 “미스”라고 한다.)
    - The data is copied from the lower level to the upper level; then is read (데이터는 하위 레벨에서 상위 레벨로 복사된 다음 읽는다.)
    - The total time is “miss penalty” (총 시간은 “미스 패널티”이다.)

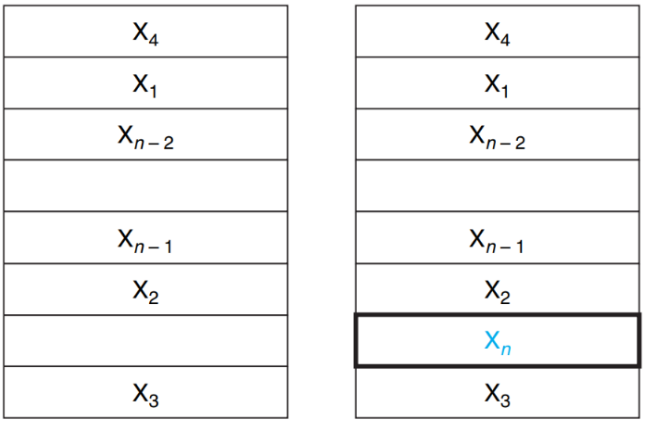
- "Hit ratio" is the fraction of accesses found in the upper level ("적중률"은 상위 수준에서 발견된 엑세스의 비율이다.)
  - The higher the hit ratio, the higher the performance (적중률이 높을수록 성능이 높다.)
  - "Miss ratio" = 1 - "hit ratio" ("미스 비율" = 1 - "적중률")

Basics of Caches (1)



- In this course, we'll focus on the most upper level two memory technologies (이 강의에서는 가장 상위 레벨의 두 가지 메모리 기술에 초점을 맞출 것이다.)
  - SRAM in upper level is called "cache" (상위 레벨의 SRAM을 "캐시"라고 한다.)
  - DRAM in lower level is called "main memory" (하위 레벨의 DRAM은 "메인 메모리"라고 한다.)
  - Processor always works with the cache (프로세서는 항상 캐시와 함께 작동한다.)
  - Data set in the cache changes by transferring data between the cache and the main memory (캐시와 메인 메모리 간에 데이터를 전송하여 캐시의 데이터 세트가 변경된다.)

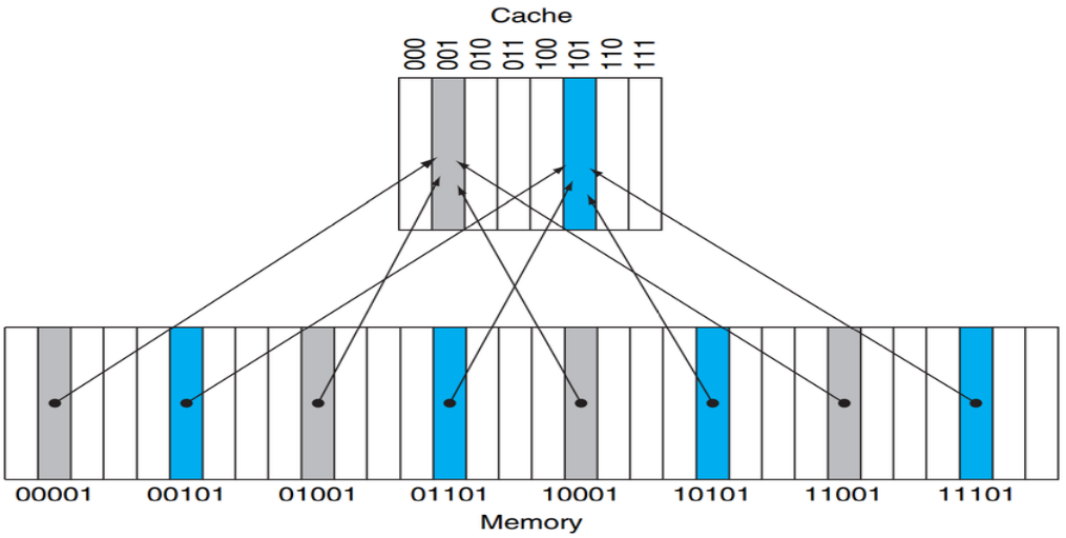
Basics of Caches (2)



a. Before the reference to X_n b. After the reference to X_n

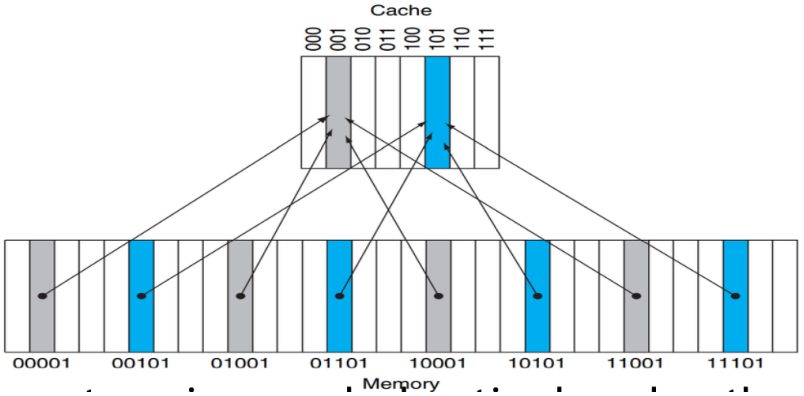
- An example of simple cache (간단한 캐시의 예)
  - The cache can hold up to 8 data items; currently, it holds 6 data items and two slots are empty (캐시는 최대 8개의 데이터 항목을 저장할 수 있으며, 현재 6개의 데이터 항목이 저장되어 있고 2개의 슬롯이 비어 있다.)
  - Given a new access to one of the six items, the data is provided from the cache to the processor (6개의 항목 중 하나에 대한 새로운 접근이 주어지면, 데이터는 캐시에서 프로세서로 제공된다.)
  - Given a new item Xn, it is brought from the main memory and accommodated in the cache (새로운 항목 Xn이 주어지면 메인 메모리에서 가져와서 캐시에 수용한다.)
- Here, one may have the following important questions (여기서 다음과 같은 중요한 질문이 있을 수 있다.)
  - (1) How do we know if a data item is in the cache? (데이터 항목이 캐시에 있는지 어떻게 알 수 있는가?)
  - (2) If it is, how do we find it? (만약 있다면 어떻게 찾을 수 있는가?)

Direct Mapped Cache (1)



- Answer: If each data item can go in exactly on place in the cache, then it is straightforward to find the data item if it is in the cache (답변 : 각 데이터 항목이 캐시의 제 위치에 정확히 들어갈 수 있다면, 데이터 항목이 캐시에 있는 경우 해당 데이터 항목을 쉽게 찾을 수 있다.)
- Direct-mapped cache: each memory location is mapped directly to one location in the cache (직접 매핑 캐시 : 각 메모리 위치가 캐시의 한 위치에 직접 매핑된다.)
    - It assigns a location in the cache for each data item (각 데이터 항목에 대해 캐시 내 위치를 할당한다.)
    - The assignment of a cache location based on the address of the data item in memory (메모리에 있는 데이터 항목의 주소를 기반으로 캐시 위치를 할당한다.)
- Here, a cache location is assigned to multiple data items (여기서 캐시 위치는 여러 데이터 항목에 할당된다.)
    - 8 cache blocks should be assigned to 32 data items (32개의 데이터 항목에 8개의 캐시 블록을 할당해야 한다.)
    - 4 different data items can be placed in a single cache block (하나의 캐시 블록에 4개의 서로 다른 데이터 항목을 배치할 수 있다.)

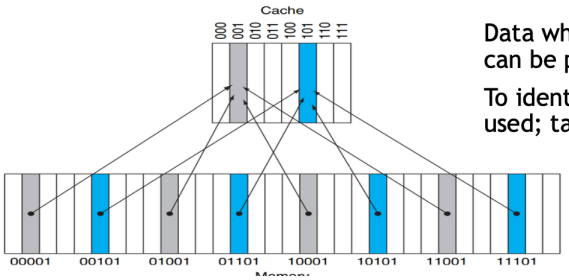
Direct Mapped Cache (2) - Cache Location



- One simple way to assign a cache location based on the address of the data item
  (데이터 항목의 주소를 기반으로 캐시 위치를 할당하는 간단한 방법 중 하나는 다음과 같다.)
  - (Block address) modulo (Number of blocks in the cache)
    ((블록 주소) modulo (캐시 내 블록수))
- In the example, the number of blocks in the cache is 8(2^3)
  (이 예제에서 캐시의 블록 수는 8(2^3)이다.)
  - Memory address 1_ten (00001_two) is mapped to cache block 1_ten (001_two); 1 module 8 = 1
    (메모리 주소 1_ten(00001_two)은 캐시 블록 1_ten(001_two)에 매핑되며 , 1 module 8 = 1 이다.)
  - Memory addresses 9_ten (01001_two), 17_ten (10001_two), and 25_ten (11001_two) are also mapped to cache block 1_ten (001_two); 9 module 8 = 1, 17 module 8 = 1, and 25 module 8 =1
    (메모리 주소 9_ten(01001_two), 17_ten(10001_two) 및 25_ten(11001_two)도 캐시 블록 1_ten(001_two)에 매핑된다; 9 module 8 = 1, 17 module 8 = 1, 25 module 8 = 1이다.)
  - Memory addresses 5_ten (00101_two), 13_ten (01101_two), 21_ten (10101_two), and 29_ten (11101_two) are all mapped to cache block 5_ten (101_two)
    (메모리 주소 5_ten(00101_two), 13_ten (01101_two), 21_ten (10101_two), 29_ten (11101_two)은 모두 캐시 블록 5_ten(101_two)에 매핑된다.)
- Simply, check the last 3 bits of the memory address; the size of cache (2^3)!

(간단히 메모리 주소의 마지막 3비트, 즉 캐시 크기 (2^3)를 확인하면 된다.)

Direct Mapped Cache (3) - Tag



Data whose addresses are 00001, 01001, 10001, and 11001 can be placed in cache @ 001; indices are the same!

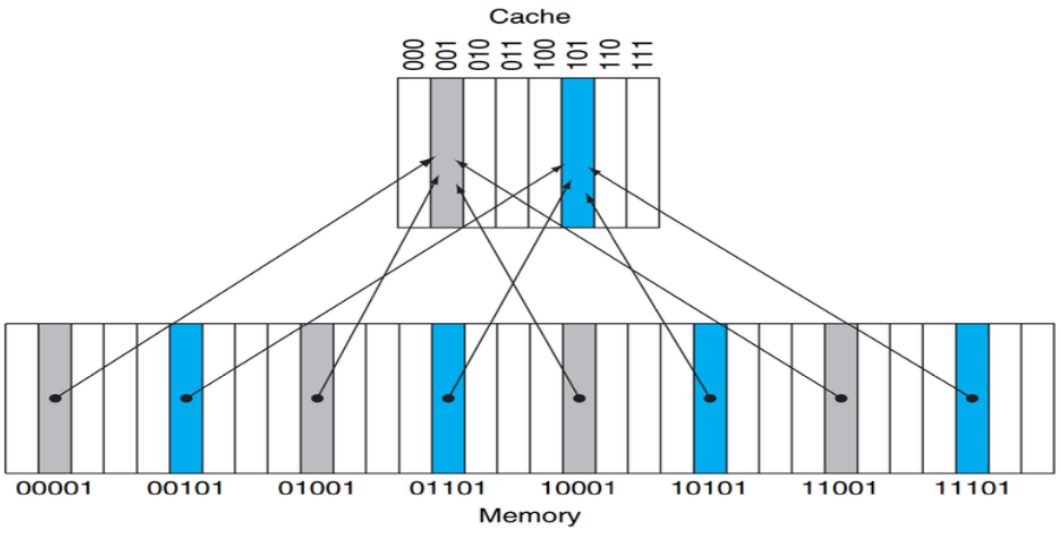
To identify one of the four, tags (00, 01, 10, 11) can be used; tags are all different!

Data whose addresses are 00001, 01001, 10001, and 11001 can be placed in cache @ 001; indices are the same! (주소가 00001, 01001, 10001, 11001인 데이터는 캐시 @001에 저장할 수 있으며 인덱스는 동일하다!)

To identify one of the four, tags (00, 01, 10, 11) can be used; tags are all different! (네 가지 중 하나를 식별하기 위해 태그 (00, 01, 10, 11)를 사용할 수 있으며, 태그는 모두 다르다!)

- A cache location can contain the data items of different memory locations (캐시 위치는 서로 다른 메모리 위치의 데이터 항목을 포함할 수 있다.)
    - Question: How do we know whether the requested data corresponds to the data in the cache? (질문 : 요청된 데이터가 캐시에 있는 데이터와 일치하는지 어떻게 알 수 있는가?)
    - We can answer the question by adding “tag” to each cache location (각 캐시 위치에 “태그”를 추가하여 질문에 답할 수 있다.)
- Memory address is divided and used for two different purposes (메모리 주소는 두 가지 용도로 나뉘어 사용된다.)
    - The lower bits are used to identify the cache location (called “index”) (하위 비트는 캐시 위치(”인덱스”)를 식별하는 데 사용된다.)
    - The upper bits are used to identify a specific data (called “tag”) (상위 비트는 특정 데이터(”태그”)를 식별하는데 사용된다.)
    - (b4 b3 b2 b1 b0) is divided; the lower 3 bits are “index”, which is used for a cache location; the upper 2 bits are “tag”, which is used for a specific data
      ((
      b4 b3 b2 b1 b0)로 나뉘며, 하위 3비트는 캐시 위치에 사용되는 “인덱스”이고, 상위 2비트는 특정 데이터에 사용되는 “태그”이다.)

Direct Mapped Cache (4) - Valid Bit



- A cache block does not always hold a valid data (캐시 블록이 항상 유효한 데이터를 보유하는 것은 아니다.)
  - The cache block can be empty, if none of the four data exist in the cache (네 가지 데이터 중 하나라도 캐시에 존재하지 않으면 캐시 블록이 비어 있을 수 있다.)
  - When a processor starts up, the cache does not have any data (프로세서가 시작될 때 캐시에 데이터가 없다.)
- We can address the above by adding a "valid bit" to each cache location (각 캐시 위치에 "유효한 비트"를 추가하여 위의 문제를 해결할 수 있다.)
  - If it is set, the cache location has a valid data; then, refer to the tag if it is the data what you want (설정되어 있으면 캐시 위치에 유효한 데이터가 있는 것이고, 원하는 데이터인 경우 태그를 참조한다.)
  - If it is not set, you do not have to check the tag, since any information in the cache location is not valid (설정되어 있지 않으면 캐시 위치의 정보가 유효하지 않으므로 태그를 확인할 필요가 없다.)