메모리&캐시

RAM의 특징과 종류

> 실행하는 프로그램은 모두 RAM에 저장되어 있다
>
> RAM의 하드웨어적 특성과 종류에 대해 알아보자

RAM의 특징

RAM에는 `실행할 프로그램의 명령어와 데이터가 저장`된다.

전원을 끄면 RAM에 저장된 명령어와 데이터가 모두 날아갑니다.

휘발성 저장 장치 : 전원을 끄면 저장된 내용이 사라지는 저장장치

비휘발성 저장 장치 : 전원이 꺼져도 저장된 내용이 유지되는 저장장치

하드디스크, SSD, CD-ROM, USB 메모리와 같은 보조기억장치가 여기에 속한다.

보조기억장치는 전원을 꺼도 내용이 유지되지만, **CPU는 보조기억장치에 접근할 수 없어요.**

➡️ `보조기억장치(비휘발성 저장 장치)에 "보관한 대상"을 저장`하고,

➡️ `RAM(휘발성 저장 장치)에 "실행할 대상"을 저장`합니다.

RAM의 용량과 성능

> RAM 용량은 컴퓨터 성능에 어떤 영향을 미칠까?

CPU가 실행하려는 프로그램이 보조기억장치에 있으면, RAM으로 가져와야 한다.

`그런데,` **RAM 용량이 적으면** 보조기억장치에서 실행할 프로그램을 `가져오는 일이 잦아 실행 시간이 길어진다.`

`예를 들어,` **RAM 용량이** A, B, C 중 하나의 프로그램만 저장할 수 있을 만큼 **작다면**

CPU는 다른 프로그램을 **실행하려고 할 때마다 RAM으로 가지고 와야 합니다.**

`하지만,` **RAM 용량이 충분히 크다면** 보조기억장치에서 많은 데이터를 가져와 미리 RAM에 저장할 수 있습니다. **여러번 가져오지 않아도 됩니다.**

➡️ **RAM 용량이 크면** `많은 프로그램을 동시에 빠르게 실행`할 수 있습니다.

RAM 용량이 필요 이상으로 커지면 더이상 **속도가 증가하지 않습니다.**

RAM의 종류

DRAM (Dynamic RAM)

: 저장된 데이터가 **동적으로 변하는(사라지는)** RAM으로 **시간이 지나면 저장된 데이터가 점차 사라집니다.**

➡️ **데이터 소멸을 막기 위해** `일정 주기로 데이터를 재활성화(다시 저장)`해야 합니다.

일반적으로 사용하는 RAM은 DRAM입니다.

소비전력이 비교적 낮고, 저렴하고, 집적도가 높기 때문에 **대용량을 설계하기가 용이**합니다.

SRAM (Static RAM)

: 저장된 데이터가 **변하지 않는** RAM으로 **시간이 지나도 저장된 데이터가 사라지지 않습니다.**

➡️ 주기적으로 데이터를 `재활성화할 필요가 없습니다.`

DRAM보다 일반적으로 속도가 빠릅니다.

소비전력이 크고, 가격이 비싸고, 집적도가 낮습니다.

대용량으로 만들어질 필요는 없지만 속도가 빨라야 하는 저장 장치에서 사용합니다.

`(=캐시메모리)`

SDRAM (Synchronous Dynamic RAM)

: **클럭 신호와 동기화된**, 발전된 형태의 DRAM으로 클럭 타이밍에 맞춰서 CPU와 정보를 주고받을 수 있습니다.

DDR SDRAM (Double Data Rate SDRAM)

: 대역폭을 넓혀 **속도를 빠르게 만든 SDRAM**으로 **한 클럭당 두 번씩** CPU와 데이터를 주고 받을 수 있습니다.

최근 가장 흔히 사용되는 RAM입니다.

SDRAM에 비교해 DDR SDRAM은 **너비가 두배인 도로**와 같습니다.

> SDR SDRAM (Single Data Rate SDRAM)
>
> : **한 클럭당 하나씩** 데이터를 주고 받을 수 있는 SDRAM
>
> **DDR2 SDRAM**
>
> : DDR SDRAM보다 대역폭이 두 배 넓은 SDRAM
>
> **DDR4 SDRAM**을 최근에 가장 많이 사용합니다.

메모리의 주소 공간

> 💡 주소에는 물리 주소와 논리 주소가 있어요

물리 주소와 논리 주소

주소에는 메모리가 사용하는 **물리 주소**, CPU와 실행 중인 프로그램이 사용하는 **논리 주소**가 있다.

물리 주소 : 정보가 실제로 저장된 하드웨어 상의 주소

논리 주소 : 실행 중인 프로그램 각각에게 부여된 0번지부터 시작되는 주소

`현재 메모리`에 메모장, 게임, 인터넷 브라우저 **프로그램이 적재되어 있다**고 가정해 봅시다.

새로운 프로그램이 **언제든 메모리에 적재**될 수 있고, **언제든 프로그램이 메모리에서 사라질 수** 있기 때문에 `물리적인 주소를 알 필요가 없다.`

`그리고,` 각각의 **프로그램에는 자신만을 위한 논리 주소**를 가지고 있다. CPU는 이 주소를 받아들인다.

`그런데!` CPU가 메모리와 상호작용하려면 `논리 주소와 물리 주소 간의 변환이 필요`해요.

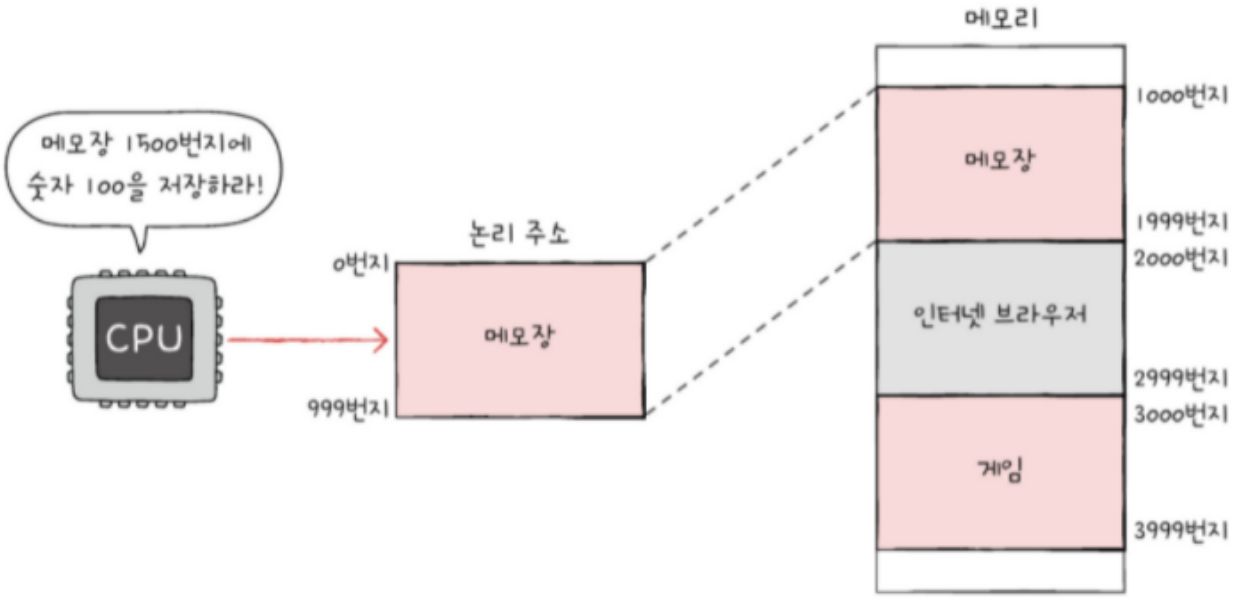
➡️ CPU와 주소 버스 사이에 위치한 `메모리 관리 장치(MMU - Memory Managemet Unit)`라는 하드웨어에 의해 수행된다.

MMU는 CPU가 요청한 `논리 주소에 베이스 레지스터 값을 더하여` **물리 주소로 변환**합니다.

베이스 레지스터 : 프로그램의 가장 작은 물리 주소. **프로그램의 첫 물리 주소**

논리 주소 : 프로그램의 **시작점으로부터 떨어진 거리**

메모리 보호 기법



위와 같은 명령어가 실행되어도 안전할까?

❌ 프로그램의 `논리 주소 영역을 벗어났기 때문`에 실행되어서는 안됩니다 ❌

이 명령이 실행 되면 인터넷 브라우저 프로그램에 숫자 100을 저장하게 됩니다.

다른 프로그램의 영역을 침범할 수 있는 명령어는 위험하기 때문에 `논리 주소 범위를 벗어나는 명령어 실행을 방지`하고, 실행 중인 프로그램이 `다른 프로그램에 영향을 받지 않도록 보호`해야 합니다.

➡️ **한계 레지스터**가 담당합니다.

한계 레지스터 : **논리 주소의 최대 크기**를 저장한다.

CPU가 접근하려는 논리 주소는 `한계 레지스터 값보다 커서는 안됩니다.`

`따라서` **CPU**는 **메모리에 접근하기 전**에 논리 주소가 **한계 레지스터 값보다 작은지를 항상 검사**한다.

논리 주소가 **한계 레지스터 값보다 크면** `인터럽트(트랩)을 발생`시켜 **실행을 중단**한다.

캐시 메모리

> 💡 CPU는 프로그램을 실행하면서 메모리에 저장된 데이터를 빈번하게 사용합니다.
>
> `하지만` CPU가 메모리에 접근하는 시간은 **CPU의 연산 속도보다 느립니다.**
>
> 이를 극복하기 위한 저장 장치가 `캐시 메모리` 입니다.

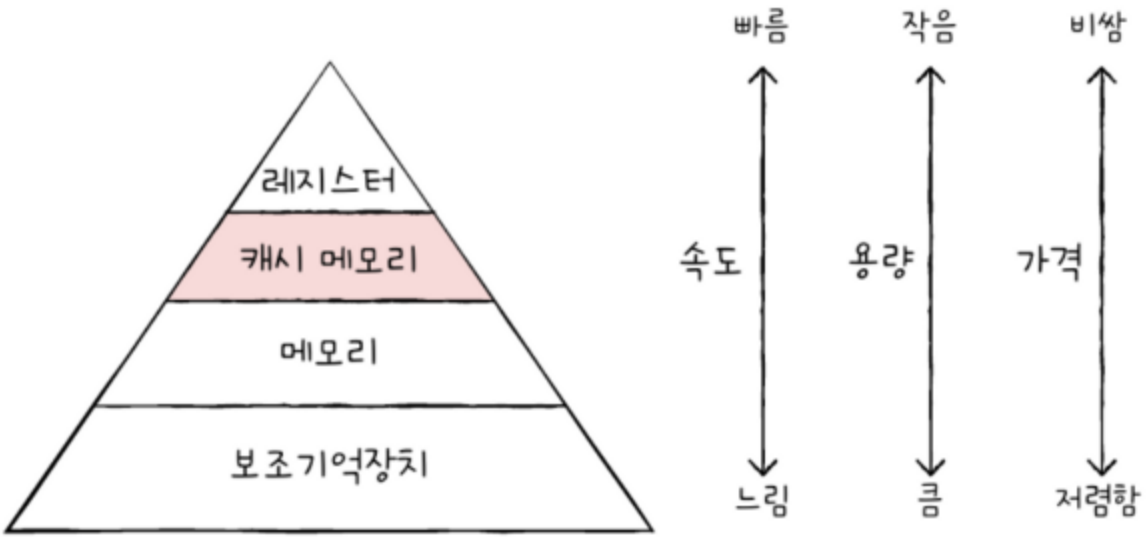
저장 장치 계층 구조

사용자들은 **빠르면서 용량이 큰 저장 장치**를 원합니다. `하지만` **둘 다 만족하기는 어렵습니다.**

- **CPU**와 가까운 저장 장치는 빠르고, 멀리 있는 저장 장치는 느립니다.
- **속도**가 빠른 저장 장치는 저장 용량이 작고, 가격이 비쌉니다.

컴퓨터가 사용하는 저장 장치들은 `'CPU에 얼마나 가까운가'`를 기준으로 계층으로 나타낼 수 있다.

➡️ **저장 장치 계층 구조**



캐시 메모리

: **CPU와 메모리 사이에 위치**하고, **레지스터보다 용량이 크고 메모리보다 빠른** SRAM 기반의 저장 장치

캐시 메모리는 CPU의 연산 속도와 메모리 접근 `속도의 차이를 줄이기 위해 탄생`했다.

CPU가 `매번 메모리에 접근 하면 오래` 걸리기 때문에 `사용할 일부 데이터를 미리 캐시 메모리에 가지고 와서 활용` 하는 것입니다.

컴퓨터 내부에는 **여러 개의 캐시 메모리**가 있다. **CPU와 가까운 순서대로 계층을 구성**한다.

코어에 가장 가까운 캐시를 `L1 캐시`라고 한다.

➡️ **L1, L2 캐시는 코어 내부**에. **L3 캐시는 코어 외부**에 위치한다.

용량은 L1, L2, L3 순으로 커지고, 가격은 L3, L2, L1 순으로 비쌉니다.

`멀티 코어 프로세서`에서는 **L1 캐시와 L2 캐시는 코어마다 할당**되고, **L3 캐시는 여러 코어가 공유**하는 형태로 사용합니다.

> 💡 분리형 캐시
>
> L1 캐시에서 `접근 속도를 빠르게 만들기 위해` 명령어만 저장하는 L1I 캐시와 데이터만 저장하는 L1D캐시로 분리합니다.

참조 지역성 원리

캐시 메모리는 메모리보다 용량이 작습니다. 따라서 메모리에 있는 모든 내용을 저장할 수 없습니다.

그렇다면 **캐시 메모리에 무엇을 저장**해야 할까요?

➡️ `CPU가 사용할 법한 대상을 예측하여 저장` 합니다.

캐시 히트 (cache hit)

: 자주 사용될 것으로 예측한 데이터가 **실제로 들어맞아 캐시 메모리 내 데이터가 CPU에 활용**될 경우

캐시 미스 (cache miss)

: 자주 사용될 것으로 예측해 캐시 메모리에 저장했지만 **예측이 틀려서 메모리에서 데이터를 가져와야 하는** 경우

캐시 적중률 (cache hit ratio)

: 캐시가 히트되는 비율

```
캐시 히트 횟수 / (캐시 히트 횟수 + 캐시 미스 횟수)
```

➡️ **캐시 적중률이 높으면 CPU의 메모리 접근 횟수를 줄일 수 있다.**

> CPU가 사용할 법한 데이터를 어떻게 알 수 있지?

캐시 메모리는 `참조 지역성의 원리에 따라` 가져올 데이터를 결정합니다.

이는 **CPU가 메모리에 접근할 때의 주된 경향**을 바탕으로 만들어진 원리이다.

- **CPU는 최근에 접근했던 메모리 공간에 다시 접근하려는 경향이 있다.**

  ➡️ 시간 지역성(temporal locality)

- **CPU는 접근한 메모리 공간 근처를 접근하려는 경향이 있다.**

  ➡️ 공간 지역성(spatial locality)