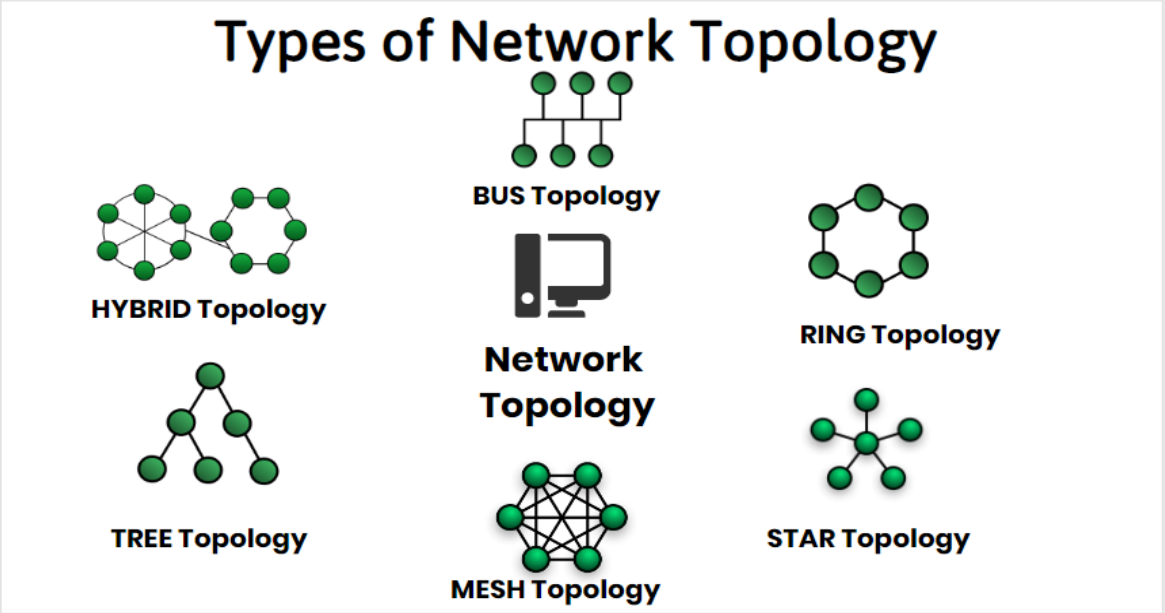
▣ Network Topology

2020.08.01




MCU DATASHEET를 보면 BUS란 단어가 꼭 등장한다.
BUS란 무엇일까?
우선, 저런 구조로 장치들이 연결되어 있다고 할때 장단점에 대하여 생각해보자.

MESH : 1:1로 모든 장치들이 연결되어있다.
하지만 돈이 많이들고 공간적인 문제 발생한다.

RING : 돈이 적게들고 공간도 많이 사용가능하나 속도가 느리다.

STAR : 중간의 중계기를 통해 연결된다. 중계기에 과도한 부담이 걸릴 수 있으며
병목현상이 발생할 수 있다.

BUS : 위 모든것들의 절충안으로 공간적인 면에서 속도측면에서 모두 무난하다.

우리가 일반적으로 사용하는 MCU는 위 BUS구조를 사용한다

▣ 메모리와 메모리주소 그리고 용량

2020.07.30 KKS

1) 32bit? 4GB?

32bit CPU에 4GB램이 조립되어있다고 할때.
PC가 사용가능한 숫자의 범위는 2^32일까? 아니면 4GB→3.436*10^10 일까?
이 물음에 대한 해답을 알아가보자.

2) 32bit? 64bit?

cpu에서 한번에 처리할 수 있는 데이터 크기에 따라서 32bit 64bit 구분
즉 32bit로 설계된 프로그램을 64bit환경에서 사용하면 남은 32bit는 논다.

3) RAM(메모리)용량이 가지는 의미

32bit cpu의 경우 2^32 = 4,294,967,296 42억개의 숫자 사용 가능
(이 이상 숫자는 CPU가 한번에 못 알아들음)
즉, CPU가 받아들일 수 있는 주소의 개수가 42억개라는 의미다.
이때 사용할 수 있는 32bit 시스템에서 사용할 수 있는 주소값은 마찬가지로
0x00000000 ~ 0xFFFFFFFE 이 주소만 사용할 수 있다.
이때 RAM이 갖는 주소하나의 크기는 1byte이며 이는 8bit이다.
따라서 32bit → 4,294,967,296 주소공간
RAM 주소의 크기 → 1Byte 1Byte X 4,294,967,296 = 약 4GB

~

| 11bit | 10bit | 9bit | 8bit | 7bit | 6bit | 5bit | 4bit | 3bit | 2bit | 1bit | 0bit |
|---|---|---|---|---|---|---|---|---|---|---|---|

메모리 주소 1개

(실제 RAM 메모리는 이런셀이 3.436*10^10개 쭉 늘어져있다고 생각하자)
하드디스크, SDD등에서 1TB씩 사용이 가능한것은 데이터가 메모리에
로드된 이후 사용하기 때문이다. 즉 하드→램→CPU

더 공부 할 것.

Q1. pointer 변수가 int가 아닌 float으로 선언되면 메모리 주소는 더 커진다.(4Byte)
이때 사용할 수 있는 RAM용량은 커지는 것인가?

Q2. 가상메모리란 무엇인가?

▣ CACHE MEMORY

2020.07.30 KKS

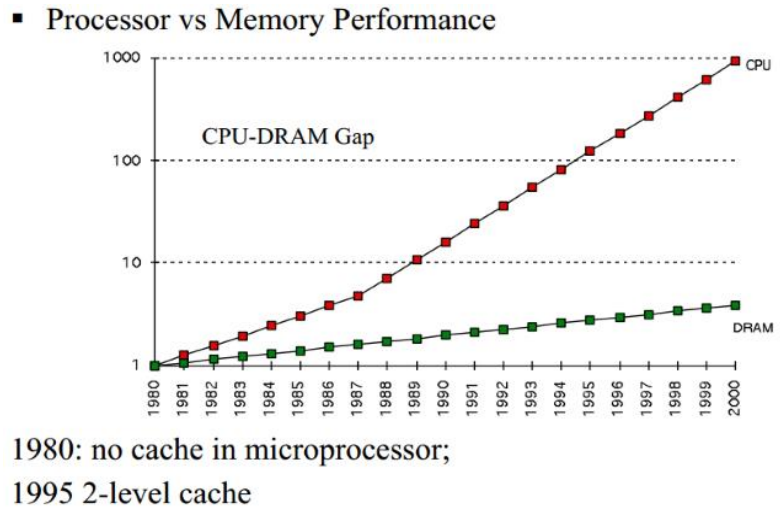
1) CACH MEMORY는 왜 사용할까?

**AMD Ryzen™ 9 3900X**

| | | |
|---|---|---|
| **CPU 코어 수:** 12 | **스레드 수:** 24 | **기본 클럭:** 3.8GHz |
| **최대 부스트 클럭:** 최대 4.6GHz | **총 L1 캐시:** 768KB | **총 L2 캐시:** 6MB |
| **총 L3 캐시:** 64MB | **언락 기능:** 예 | **CMOS:** TSMC 7nm FinFET |
| **패키지:** AM4 | **PCI Express 버전:** PCIe 4.0 x16 | **방열 솔루션 (PIB):** Wraith Prism with RGB LED |
| **방열 솔루션 (MPK):** Wraith PRISM | **기본 TDP/TDP:** 105W | **최대 온도:** 95°C |

(출처 : AMD KOREA 홈페이지)
최신 CPU 사양을 확인하면 언제나 캐시 메모리를 확인 할 수 있다.
또한 그 캐시 메모리는 L1, L2, L3등으로 나뉜다.
왜 별도의 캐시 메모리를 사용하는 것이며 L1,L2,L3가 의미하는바는 무엇일까?

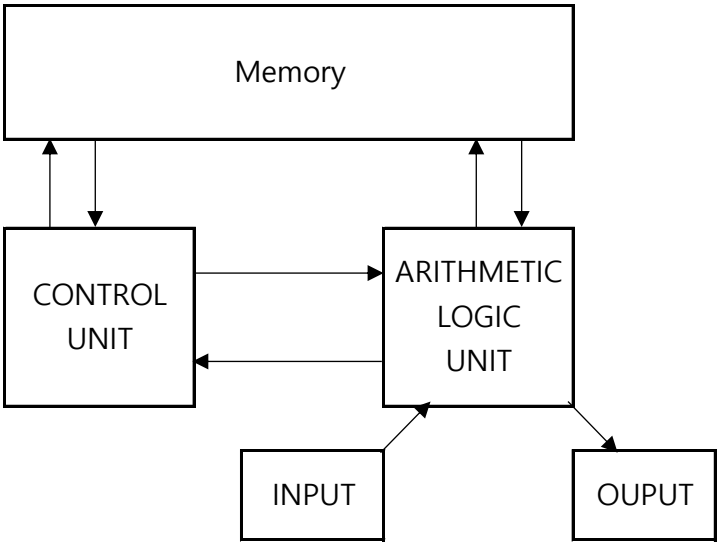
2) 캐시메모리는 속도가 빠르다.



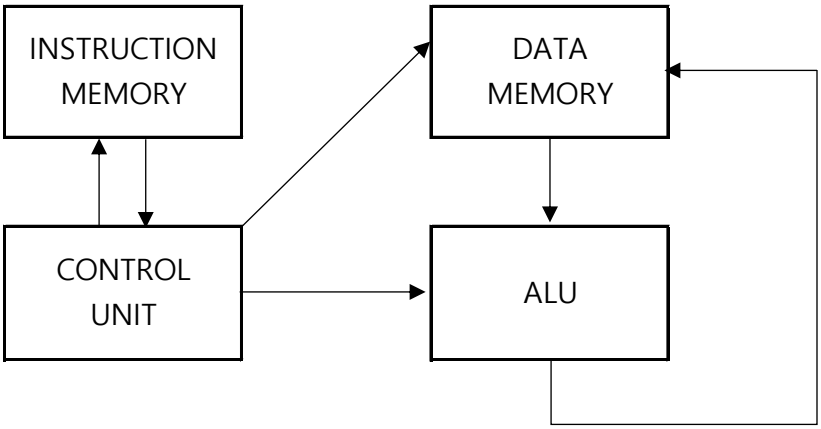

위 표는 시간이 지남에 따라 CPU의 속도와 DRAM의 속도 차이이다.
보이듯이 DRAM의 속도가 CPU의 속도를 따라가지 못하고 있다.
이처럼 CPU와 메모리간에 속도차이가 발생하면 병목현상 또한 가속화 되므로
이를 해결하기 위해 CACHE MEMORY를 도입하였다.
CPU외부 DRAM보다 물리적인 거리도 가까워서 더 빠르다.

3) 캐시메모리는 CPU내부적으로 하버드 구조로 사용이 가능하다.
일전에 배웠던 하버드 구조와 폰노이만 구조에서 폰노이만 구조에서
파이프라이닝을 원할히 수행하기 위하여 폰노이만 구조에서 하버드 구조가
파생되었다고 확인하였다.(각각 장단점이 있다.)
이때 캐시메모리는 두 구조의 완충형태를 위한것이라 볼 수 있다.
CPU내부에 CACHE MEMORY(속도가 빠름)을 두어 하버드 구조를 채택하고
자주 사용하는 데이터의 경우 CACHE에 저장한다.
CACHE는 INSTRUCION MEMORY와 DATA MEMORY로 구분되어 있다.
CPU외적으로는 RAM과 폰노이만 구조로 되어있다.

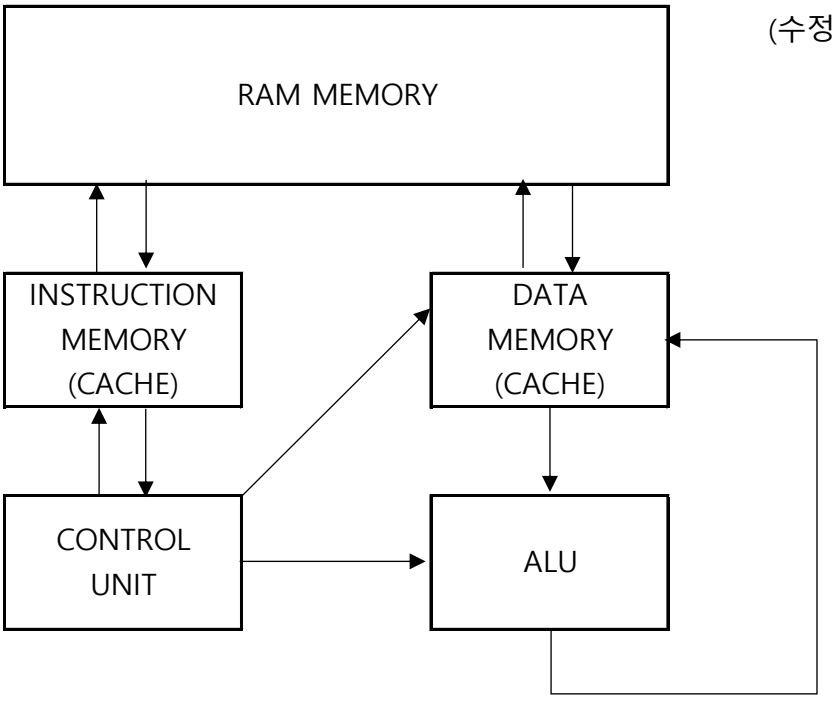
(폰노이만 구조)



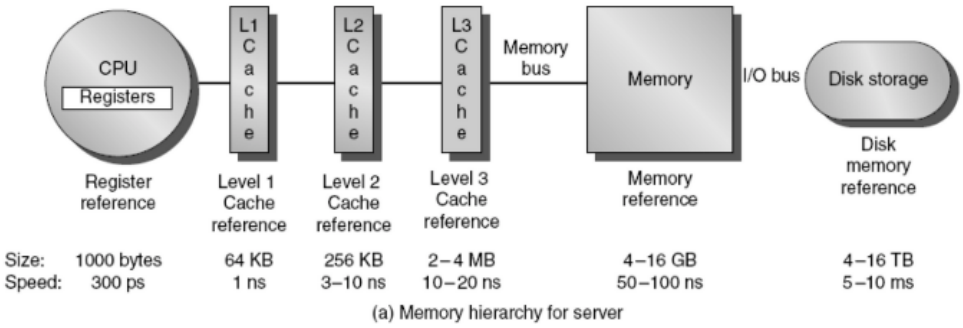

(하버드 구조)




(수정된 하버드 구조)

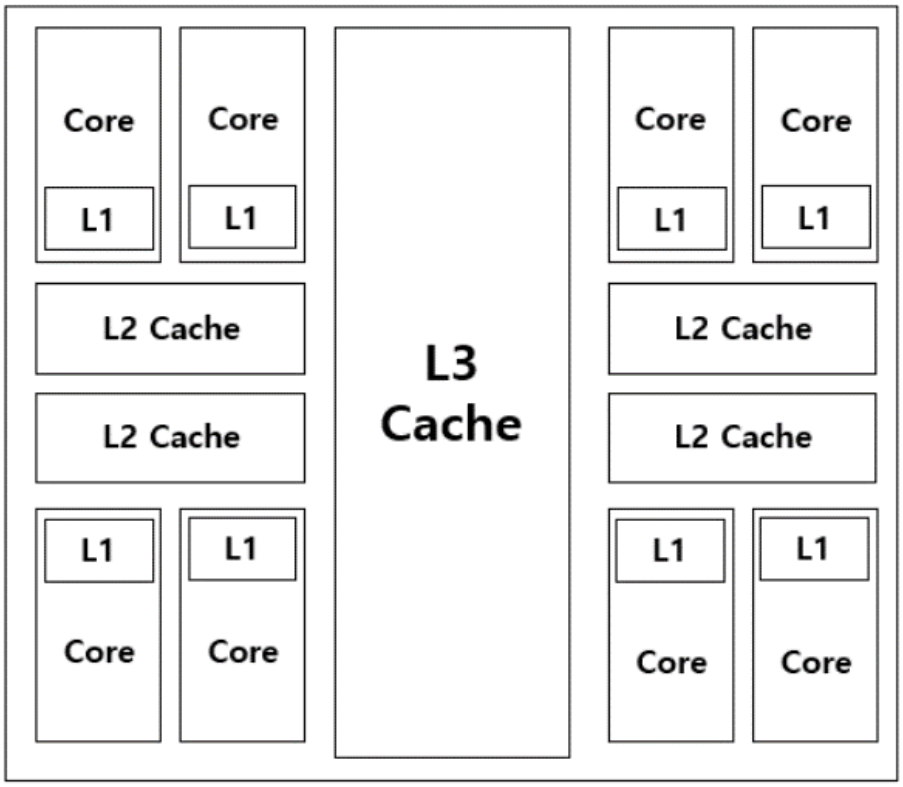


4) L1, L2, L3 는 무슨 차이?



(a) Memory hierarchy for server

위 그림에서 보듯 L1이 L2보다, L2는 L3보다 빠르다.
당연히 L1이 L2보다 L2가 L3보다 비싸다.

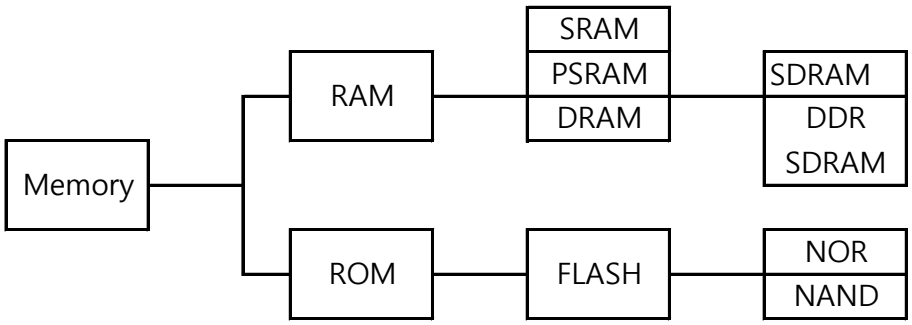



그림에서 보듯이 L1은 CORE와 직접 1:1 매칭되며
L1→L2→L3순으로 데이터를 검색하게 된다.

▣ DRAM & SRAM

2020.08.01 KKS

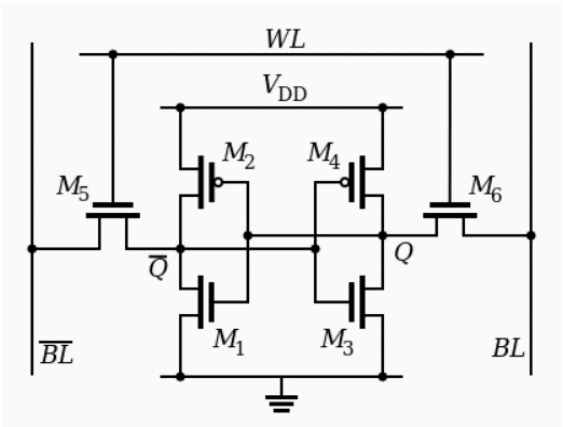
1. MEMORY의 종류



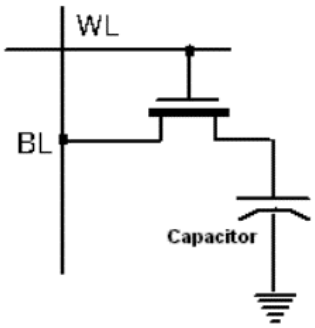

메모리의 종류는 매우 다양하다.
일반적으로 말하는 메모리는 DRAM을 이야기 한다.
캐시메모리는 SRAM이라 한다.

2.,DRAM과 SRAM의 차이

SRAM 회로




DRAM 회로




DRAM에 비하여 SRAM의 회로 딱봐도 복잡하다.
SRAM의 경우 데이터를 write할때 별도의 충방전 시간이 필요하지 않다.
하지만 DRAM의 경우 위 회로에 보이는 커패시터를 충전시켜줘야 한다.
또한 주기적으로 CAP을 REFRESH 시켜야 한다.(자연상태에서는 방전되기 때문)
결론적으로 SRAM의 속도가 DRAM보다 빠르다.

CPU내부에 캐시메로리(SRAM)을 두어 자주 사용하는 데이터를 저장한다.
이를 통해 DRAM에 접근하는것보다(물리적 거리도 길다) 캐시메모리에 접근할경우
빠른속도로 데이터 값을 쓸 수 있다.

또한 DRAM의 경우 WRITE가 READ보다 늦는 이유또한 CAP을 충전해야 하기 때문이다.
(값을 읽을때는 별도의 충전 필요치 않다)

▣ 폰 노이만 구조 VS 하버드 구조(추가)

2020-08-01 KKS

1. CPU내부는 하버드 구조 외부는 폰 노이만 구조?

→ 수정된 하버드 구조

위에서 말했던 CPU내부에 캐시메모리를 내장하는 구조

이때 캐시 메모리를 하버드 구조로 설계하여 병목현상을 줄인다.

2. 프로세서가 하는 일

프로세서 = CPU

위에서 설명했던 것  처럼 DATA에 저장된 명령어를 가져오고

해석하고 연산하고 Data에 저장하는 일련의 과정을 수행한다.

3. 파이프라인 구조란 무엇인가?

위에서 말했듯이 일정시간안에 더 많은 일들을 수행하기 위해서

동작을 겹치지 않게 쌓아서 수행하는 것을 이야기 한다.

위에서 설명한 한 클럭에 한가지 기능수행 (Fectch, Decode, Excute, Store)는

아주 간단한 예시이고

이 외에도 슈퍼 스칼라방식이 있다. 명령어를 병렬로 한번에 처리한다.

| | | | | | | |
|---|---|---|---|---|---|---|
| FETCH | DECODE | EXCUTE | STORED | - | - | - |
| FETCH | DECODE | EXCUTE | STORED | - | - | - |
| - | FETCH | DECODE | EXCUTE | STORED | - | - |
| - | FETCH | DECODE | EXCUTE | STORED | - | - |
| - | - | FETCH | DECODE | EXCUTE | STORED | - |
| - | - | FETCH | DECODE | EXCUTE | STORED | - |

(슈퍼스칼라)

4. RISC = HARVAD? CISC= VON NEUMAN?

전혀 상관 없음.

RISC 구조에 폰노이만 아키텍쳐를 채택 할 수 있고

CISC구조에 하버드 구조를 채택 할  수 있다.

5. CISC 방식 파이프라이닝 불가능?

CISC방식에도 파이프라인 사용가능.

하지만 RISE방식이 한 클럭에 한가지 명령을 완료하기에

파이프라인을 구성하는데 있어서 더 효율적임.