*** RAM은 어떤 단어들의 약자이며, 왜 RAM이라고 부르게 되었을까?**

Random Access Memory, RAM은 어느 위치에 저장된 데이터든지 접근(읽기 및 쓰기)하는 데 동일한 시간이 걸리는 메모리이기에 '랜덤(Random, 무작위)'이라는 명칭이 주어진다.

*** 하버드 구조와 폰 노이만 구조의 한계는 각각 무엇이며,**

>>하버드 구조 단점
1. 버스 시스템이 복잡하여 설계가 어렵다
2. 프로그램 중에 데이터가 함께 혼합된 경우, 예를 들어 폰트 데이터나 표시할 문장 데이터, 메뉴 데이터 등은 프로그램과 함께 존재해야 하는데, 프로세서가 이러한 데이터를 읽어들이려고 시도하면 엄연히 '데이터'를 읽으려 하므로 데이터 메모리에서 이 내용을 찾으려 한다. 그런데 이 데이터들은 명령(프로그램) 메모리에 존재하기 때문에 명령(프로그램) 메모리로부터 데이터를 읽을 수 있는 특수 상황의 명령어가 필요하고, 더구나 명령(프로그램) 메모리와 데이터 메모리의 비트수가 다른 경우 주소를 변환하는 과정을 거쳐야 한다.

>>폰노이만 구조 단점
1. 명령어를 읽을 때 데이터를 읽을 수 없다
2. 명령어가 200개 수준으로 너무 많다

*** 지금은 어떻게 구성된 모습으로 사용되고 있을까?**

최근에 CPU의 성능은 메모리의 속도와 비교해 크게 향상했다. 성능을 높이려면 주 메모리의 접근 횟수를 줄이는 노력이 필요하다. 명령을 처리할 때마다 주 메모리에 접근할 필요가 있다고 한다면 성능 향상은 전망할 수 없다. 이른바 메모리 속박 문제이다.

메모리는 속도가 올라가면 값이 비싸질 수 있다. 이를 해결하는 방법은 캐시로 불리는 작은 규모의 고속의 메모리를 준비하는 것이다. CPU는 필요로 하는 메모리의 내용이 캐시에 존재하면 성능이 향상된다. 하지만 반대로 필요한 메모리의 내용이 캐시에 없으면, 메모리로부터 캐시에 내용을 가져오는 것이다. 이 캐시의 조정이 컴퓨터의 설계로 중요한 관점이 되고 있다.

최신의 성능이 좋은 CPU 설계에서는 하버드와 폰 노이만 두 쪽 모두의 아키텍처를 도입하고 있다. 캐시 메모리 장치는 보통 명령용과 데이터용으로 분리되어 있다. 하버드 아키텍처는 CPU와 캐시의 관계에 활용되고 있다. 캐시에 오류가 일어나면 주 메모리로부터 데이터를 가져 오고, 명령 캐시나 데이터 캐시에 저장한다. 따라서, 폰 노이만 구조는 CPU 외부에 적용된다.

+ 파이프라인 프로세서

폰노이만 구조는 stored program concept 으로 파이프라인 스테이지(fetch, decode, execute, store)가 분절되지 않은 프로세서이다. 즉 한 명령어가 끝나기 전에는 다른 명령어가 파이프라인에 투입될 수 없다. 다시 말해 위의 네 단계에 해당하는 하드웨어가 모두 명령어 하나에 매달려, 그 명령어가 끝나기만을 기다린다.

1클럭 -> ① :: ○○○○ ::
2클럭 -> ② :: ①○○○ ::
3클럭 -> ② :: ○①○○ ::
4클럭 -> ② :: ○○①○ ::
5클럭 -> ② :: ○○○① ::
6클럭 -> ③ :: ②○○○ :: ①
7클럭 -> ③ :: ○②○○ :: ①
8클럭 -> ③ :: ○○②○ :: ①
9클럭 -> ③ :: ○○○② :: ①
10클럭 -> ④ :: ③○○○ :: ②①

반면 위의 네 단계가 각각의 파이프라인 스테이지로 분절된 '파이프라이닝된' 프로세서의 경우, 어떤 명령어가 fetch 스테이지를 마치고 decode 스테이지로 넘어가면 fetch 스테이지는 놀고 있는 것이 아니라 바로 그 다음 명령어를 fetch하는 작업에 착수할 수 있다.

1클럭 -> ① :: ○○○○ ::
2클럭 -> ② :: ①○○○ ::
3클럭 -> ③ :: ②①○○ ::
4클럭 -> ④ :: ③②①○ ::
5클럭 -> ⑤ :: ④③②① ::
6클럭 -> ⑥ :: ⑤④③② :: ①
7클럭 -> ⑦ :: ⑥⑤④③ :: ②①
8클럭 -> ⑧ :: ⑦⑥⑤④ :: ③②①
9클럭 -> ⑨ :: ⑧⑦⑥⑤ :: ④③②①
10클럭 -> ⑩ :: ⑨⑧⑦⑥ :: ⑤④③②①

단, 파이프라이닝의 약점은 파이프라인 버블에 있다. 명령어가 특정 파이프라인 스테이지를 마치지 못해 공백이 되는 경우를 파이프라인 버블이라 하고 버블이 있는 경우 그 버블이 채워질 때까지 파이프라인으로의 투입은 순연된다.

1클럭 -> ① :: ○○○○ ::
2클럭 -> ② :: ①○○○ ::
3클럭 -> ② :: ○①○○ :: <- 여기서 명령어 2 버블 발생 (fetch 단계)
4클럭 -> ③ :: ②○①○ ::
5클럭 -> ④ :: ③②○① ::
6클럭 -> ⑤ :: ④③②○ :: ①
7클럭 -> ⑥ :: ⑤④③② :: ①
8클럭 -> ⑦ :: ⑥⑤④③ :: ②①
9클럭 -> ⑧ :: ⑦⑥⑤④ :: ③②①
10클럭 -> ⑨ :: ⑧⑦⑥⑤ :: ④③②①

*클럭 : https://ko.wikipedia.org/wiki/%ED%81%B4%EB%9F%AD_%EC%8B%A0%ED%98%B8