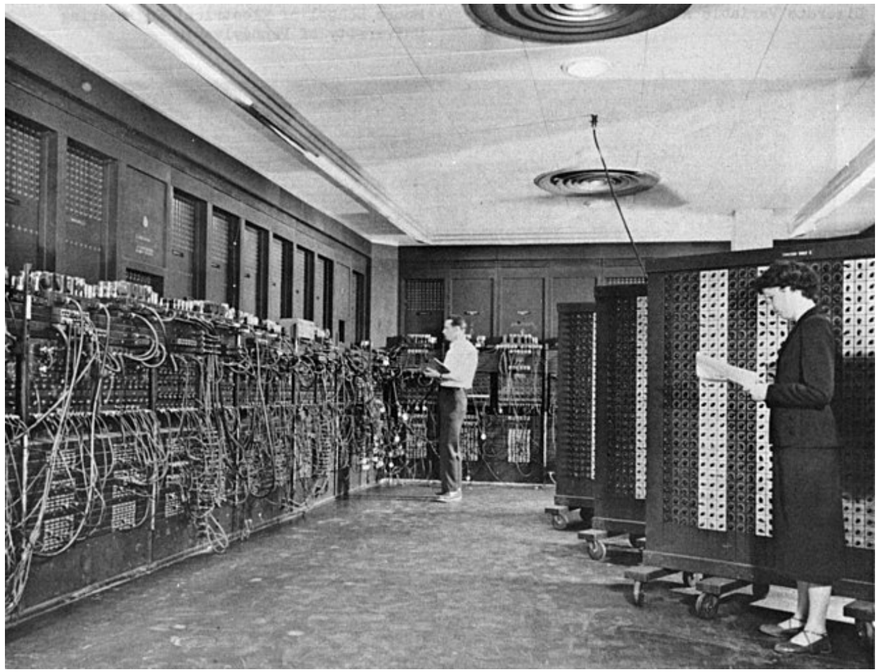
폰노이만의 컴퓨터 구조

목표

컴퓨터의 구조를 이해함으로써

전반적인 컴퓨터의 작업 플로우를 이해한다



애니악 ENIAC

계산을 할 때마다,
손으로 직접 진공관의 회로 스위치를 다시 조정하여
입력을 처리하는 하드웨어 프로그램 방식

직접 전선의 위치를 바꿔가며 프로그래밍을 수행

단점을 해결 하는 새로운 아키텍쳐의 등장

폰노이만의

컴퓨터 구조

직접 전선의 위치를 바꿀 필요 없이

소프트웨어만 교체하여 다른 연산을 수행한다.

연산 명령어

연산에 필요한 데이터

연산 결과 데이터

단점을 해결 하는 새로운 아키텍쳐의 등장

교체하려면 저장을 해야하기 때문에..

폰노이만의
컴퓨터 구조

메모리

직접 전선의 위치를 바꿀 필요 없이

소프트웨어만 교체하여 다른 연산을 수행한다.

연산 결과 데이터

주소 데이터

Memory Address	Memory Content
0000	10000000
0001	00000000
0010	00001100
0011	00000100
0100	10000001
0101	00000000
0110	00000000
0111	10001100
1000	00000000
1001	11010011
1010	00000000
1011	00110001
1100	00000000
1101	10000000
1110	00000000
1111	00100100

- 프로그램 파일은 명령어와 데이터로 이루어져 있고 메모리에 프로그램이 저장되어 있다.

- 휘발성(RAM): 주 기억장치 , 비휘발성(ROM): 보조 기억장치에 따라 종류가 나뉜다.

  Random Access Memory — DRAM, SRAM
  Read Only Memory — HDD, SDD

- 성능: SRAM (플립플롭 회로로 구성) 〉 DRAM 〉 SDD 〉 HDD

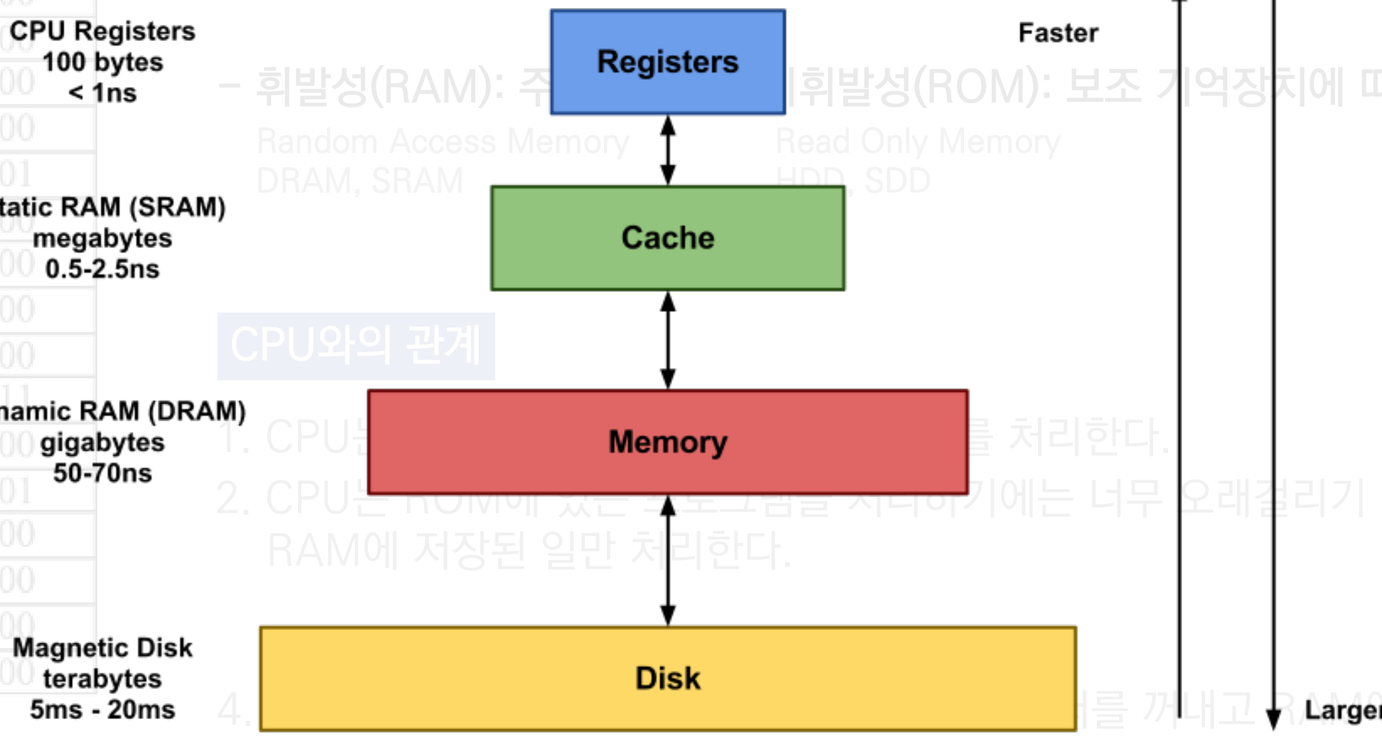
CPU와의 관계

1. CPU는 RAM에 저장된 데이터나 명령어를 처리한다.
2. CPU는 ROM에 있는 프로그램을 처리하기에는 너무 오래걸리기 때문에 RAM에 저장된 일만 처리한다.

+ 컴퓨터는 RAM이외에도 레지스터, 캐시 메모리 총 세가지 기억장치를 사용하여 프로그램을 실행

메모리 계층 구조

레지스터, 캐시메모리 등 더 많은 개념이 있지만..



입력 →

중앙 처리 장치 (CPU)

제어장치 (Control Unit)	명령을 해석하고, 명령을 실행하기 위한 제어 신호를 발생시킨다.
산술 연산 장치 (Arithmetic Logic Unit)	CU의 신호에 따라 실질적인 산술, 논리 연산이 일어난다.
레지스터 (Register)	제어, 연산 등에서 사용하는 임시 기억 장치이다.

→ 출력

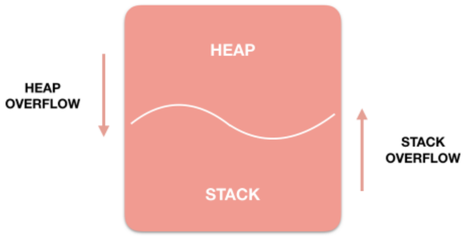
명령어 읽기,
데이터 읽고 쓰기

↑↓

버스 (BUS)
둘을 연결한다

메모리 (Momory Unit)

코드 영역	실행할 프로그램의 코드가 저장되는 영역. CPU는 코드 영역에 저장된 명령어를 처리한다.
데이터 영역	프로그램의 전역 병수와 정적 변수가 저장되는 영역. 프로그램의 시작과 함께 할당된다.
스택 영역	함수의 호출과 관계되는 지역 변수와 매개변수가 저장되는 영역. 함수의 호출과 함께 할당되며, 함수의 호출이 완료되면 소멸한다.
힙 영역	사용자가 직접 관리할 수 있는 '그리고 해야만 하는' 메모리 영역. 사용자에 의해 메모리 공간이 동적으로 할당되고 해제된다.



명령어 사이클

인간이 사고하는 과정과 유사
CPU가 한개의 명령어를 실행하는 전체 처리과정

1. 명령어 인출 (Instruction Fetch) 사이클

CPU에서 명령어를 읽어오는 단계

명령어 인출 (Fetch)

* PC에 저장된 명령어 주소를 * MAR로 전송한다.
MAR에 저장된 주소에 있는 명령어를 * MBR로 전송한 뒤, PC가 다음 명령어 주소를 가르키도록 준비한다.
인출한 명령어를 * IR로 전송한다.

* PC: 다음에 실행할 명령어에 대한 메모리 주소를 추적하는 명령어 계수기 (한 번 읽힐 때 마다 값이 1이 증가하여, 순차적으로 다음 메모리에 접근이 가능하다)
* MAR: CPU 내부에 존재하는 회로로 다음에 실행할 명령어의 메모리에 있는 주소를 보관하는 레지스터
* MBR (메모리 버퍼) : 메모리로 부터 Fetch 되어 cpu가 처리할 준비가 된 데이터나 메모리에 저장되어 대기 중인 데이터를 보관하는 양방향 레지스터
* IR: 메모리로부터 Fetch되는 (현재 실행할) 명령어를 일시적으로 보관하는 레지스터

2. 명령어 실행 (Instruction Fetch) 사이클

CPU가 명령어를 실행하는 단계

명령어 해석 (Decode)

인출된 명령어를 해석한다.

피연산자 인출 (Fetch)

명령어 실행 (Excute)

명령어 실행에 필요한 CPU 내외부 제어신호를 발생시킨다.

일반적인 명령어 기본 사이클

3. 인터럽트 (Interrupt) 사이클

CPU에게 처리할 예외 상황에 대해 알려주는 것

프로그램 동작 Flow

마우스 클릭으로 실행

보조기억장치에 저장 되어 있는 프로그램을 메모리에 올린다.

Loader: 외부 기억장치로부터 주 기억 장치로 옮기기 위하여 메모리 할당 및 연결, 재배치, 적재를 담당한다.
Loader를 통해 메모리에 올라간 프로그램을 프로세스라고 부른다.

EX) 저장된 명령어의 주소가 700~ 799이라고 가정

명령어 인출

주소	
700	PC
701	
...	

→

주소	
700	MAR 레지스터
701	
...	

PC에서 저장된 주소를 MAR로 전달

→

주소	
700	MBR 레지스터
701	PC
...	

MAR에서 저장된 주소를 통해 명령어를 찾고 이를 MBR에 저장한다.

→

주소	
700	IR 레지스터
701	PC
...	

명령어를 IR레지스터에 저장

주소	
700	IR 레지스터
701	PC
...	

Decode 명령어를 해석한다. (어떤 연산을 하는가)

↓

Fetch 연산에 필요한 피연산자를 레지스터 파일이나 메모리에서 읽음

↓

Excute 실제 계산을 수행한다.
명령어를 실행한 결과를 잠시 누산기에 저장한다.

* 누산기: 연산장치에서 중요한 역할을 하는 레지스터로 연산등의 중간 결과를 기억한다.

↓

다음 명령어 인출

문제점 폰노이만 병목 현상

CPU에서 아무리 계산을 빠르게 처리할 수 있었도 메모리에서 불러오는 속도가 느리면 전체적인 속도가 느려진다.
즉, 계산 속도가 기억장치 속도에 영향을 받는다.

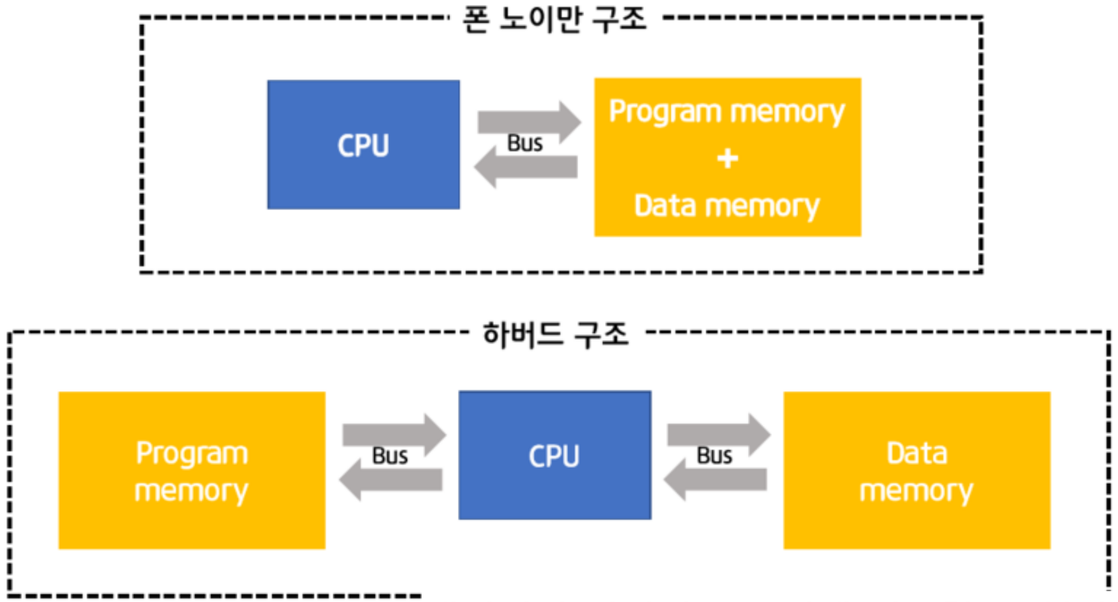
근본적인 원인
CPU 연산속도가 너무 빠른 것에 비해 다른 장치들에서 CPU까지의 전달 속도가 너무 느리다.

그래서.. 하버드 구조

현대에 이르러서 CPU 외부적으로는 폰 노이만 구조를
내부적으로는 하버드 구조를 적용하여 병목 현상이 어느정도 해결 되었음

- 캐시 메모리 장치: 하버드 구조

- 주 메모리 장치: 폰 노이만 구조



데이터와 명령어를 동시에 접근할 수 있다.

전기 회로가 많이 필요하고 두개의 버스와 메모리를 가지기 때문에 CPU 코어에서 공간을 많이 차지하게 된다.

끝