

컴퓨터의 구조

컴퓨터의 구성

컴퓨터가 가지는 구성에 대해 알아보자

컴퓨터 시스템은 크게 하드웨어와 소프트웨어로 나누어진다.

하드웨어 : 컴퓨터를 구성하는 기계적 장치

소프트웨어 : 하드웨어의 동작을 지시하고 제어하는 명령어 집합

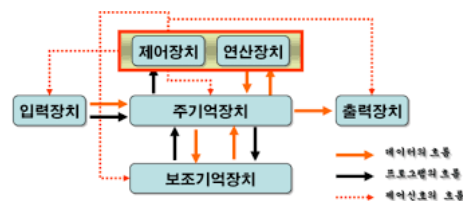
하드웨어

- 중앙처리장치(CPU)
- 기억장치 : RAM, HDD
- 입출력 장치 : 마우스, 프린터

소프트웨어

- 시스템 소프트웨어 : 운영체제, 컴파일러
- 응용 소프트웨어 : 워드프로세서, 스프레드시트

먼저 하드웨어부터 살펴보자



하드웨어는 중앙처리장치(CPU), 기억장치, 입출력장치로 구성되어 있다.

이들은 시스템 버스로 연결되어 있으며, 시스템 버스는 데이터와 명령 제어 신호를 각 장치로 실어나르는 역할을 한다.

중앙처리장치(CPU)

인간으로 따지면 두뇌에 해당하는 부분

주기억장치에서 프로그램 명령어와 데이터를 읽어와 처리하고 명령어의 수행 순서를 제어할 중앙처리장치는 비교와 연산을 담당하는 **산술논리연산장치(ALU)**와 명령어의 해석과 실행을 담당하는 **제어장치**, 속도가 빠른 데이터 기억장소인 **레지스터**로 구성되어있음

개인용 컴퓨터와 같은 소형 컴퓨터에서는 CPU를 마이크로프로세서라고도 부름

기억장치

프로그램, 데이터, 연산의 중간 결과를 저장하는 장치

주기억장치와 보조기억장치로 나누어지며, RAM과 ROM도 이곳에 해당함. 실행중인 프로그램과 같은 프로그램에 필요한 데이터를 일시적으로 저장한다.

보조기억장치는 하드디스크 등을 말하며, 주기억장치에 비해 속도는 느리지만 많은 자료를 영구적으로 보관할 수 있는 장점이 있다.

입출력장치

입력과 출력 장치로 나누어짐.

입력 장치는 컴퓨터 내부로 자료를 입력하는 장치 (키보드, 마우스 등)

출력 장치는 컴퓨터에서 외부로 표현하는 장치 (프린터, 모니터, 스피커 등)

시스템 버스


하드웨어 구성 요소를 물리적으로 연결하는 선

각 구성요소가 다른 구성요소로 데이터를 보낼 수 있도록 통로가 되어줌
용도에 따라 데이터 버스, 주소 버스, 제어 버스로 나누어짐

데이터 버스

중앙처리장치와 기타 장치 사이에서 데이터를 전달하는 통로


기억장치와 입출력장치의 명령어와 데이터를 중앙처리장치로 보내거나, 중앙처리장치의 연산 결과를 기억장치와 입출력장치로 보내는 '양방향' 버스임



주소 버스

데이터를 정확히 실어나르기 위해서는 기억장치 '주소'를 정해주어야 함.

주소버스는 중앙처리장치가 주기억장치나 입출력장치로 기억장치 주소를 전달하는 통로이기 때문에 '단방향' 버스임



제어 버스

주소 버스와 데이터 버스는 모든 장치에 공유되기 때문에 이를 제어할 수단이 필요함

제어 버스는 중앙처리장치가 기억장치나 입출력장치에 제어 신호를 전달하는 통로임

제어 신호 종류 : 기억장치 읽기 및 쓰기, 버스 요청 및 승인, 인터럽트 요청 및 승인, 클락, 리셋 등

제어 버스는 읽기 동작과 쓰기 동작을 모두 수행하기 때문에 '양방향' 버스임

컴퓨터는 기본적으로 읽고/처리한 뒤 저장하는 과정으로 이루어짐

(READ → PROCESS → WRITE)

이 과정을 진행하면서 끊임없이 주기억장치(RAM)과 소통한다. 이때 운영체제가 64bit라면, CPU는 RAM으로부터 데이터를 한번에 64비트씩 읽어온다.

중앙처리장치(CPU) 작동 원리

CPU는 컴퓨터에서 가장 핵심적인 역할을 수행하는 부분. '인간의 두뇌'에 해당

크게 연산장치, 제어장치, 레지스터 3가지로 구성됨

• 연산 장치

산술연산과 논리연산 수행 (따라서 산술논리연산장치라고도 불림)

연산에 필요한 데이터를 레지스터에서 가져오고, 연산 결과를 다시 레지스터로 보냄

• 제어 장치

명령어를 순서대로 실행할 수 있도록 제어하는 장치

주기억장치에서 프로그램 명령어를 꺼내 해독하고, 그 결과에 따라 명령어 실행에 필요한 제어 신호를 기억장치, 연산장치, 입출력장치로 보냄

또한 이들 장치가 보낸 신호를 받아, 다음에 수행할 동작을 결정함

• 레지스터

고속 기억장치임

명령어 주소, 코드, 연산에 필요한 데이터, 연산 결과 등을 임시로 저장

용도에 따라 범용 레지스터와 특수목적 레지스터로 구분됨

중앙처리장치 종류에 따라 사용할 수 있는 레지스터 개수와 크기가 다름

- 범용 레지스터 : 연산에 필요한 데이터나 연산 결과를 임시로 저장

- 특수목적 레지스터 : 특별한 용도로 사용하는 레지스터

특수 목적 레지스터 중 중요한 것들

- MAR(메모리 주소 레지스터) : 읽기와 쓰기 연산을 수행할 주기억장치 주소 저장
- PC(프로그램 카운터) : 다음에 수행할 명령어 주소 저장
- IR(명령어 레지스터) : 현재 실행 중인 명령어 저장
- MBR(메모리 버퍼 레지스터) : 주기억장치에서 읽어온 데이터 or 저장할 데이터 임시 저장
- AC(누산기) : 연산 결과 임시 저장

CPU의 동작 과정

1. 주기억장치는 입력장치에서 입력받은 데이터 또는 보조기억장치에 저장된 프로그램 읽어옴
2. CPU는 프로그램을 실행하기 위해 주기억장치에 저장된 프로그램 명령어와 데이터를 읽어와 처리하고 결과를 다시 주기억장치에 저장
3. 주기억장치는 처리 결과를 보조기억장치에 저장하거나 출력장치로 보냄
4. 제어장치는 1~3 과정에서 명령어가 순서대로 실행되도록 각 장치를 제어

명령어 세트란?

CPU가 실행할 명령어의 집합

연산 코드(Operation Code) + 피연산자(Operand)로 이루어짐

연산 코드 : 실행할 연산

피연산자 : 필요한 데이터 or 저장 위치

연산 코드는 연산, 제어, 데이터 전달, 입출력 기능을 가짐

피연산자는 주소, 숫자/문자, 논리 데이터 등을 저장

CPU는 프로그램 실행하기 위해 주기억장치에서 명령어를 순차적으로 인출하여 해독하고 실행하는 과정을 반복함

CPU가 주기억장치에서 한번에 하나의 명령어를 인출하여 실행하는데 필요한 일련의 활동을 '명령어 사이클'이라고 말함

명령어 사이클은 인출/실행/간접/인터럽트 사이클로 나누어짐

주기억장치의 지정된 주소에서 하나의 명령어를 가져오고, 실행 사이클에서는 명령어를 실행함. 하나의 명령어 실행이 완료되면 그 다음 명령어에 대한 인출 사이클 시작

인출 사이클과 실행 사이클에 의한 명령어 처리 과정

인출 사이클에서 가장 중요한 부분은 PC(프로그램 카운터) 값 증가

- PC에 저장된 주소를 MAR로 전달
- 저장된 내용을 토대로 주기억장치의 해당 주소에서 명령어 인출
- 인출한 명령어를 MBR에 저장
- 다음 명령어를 인출하기 위해 PC 값 증가시킴
- 메모리 버퍼 레지스터(MBR)에 저장된 내용을 명령어 레지스터(IR)에 전달

```
T0 : MAR ← PC
T1 : MBR ← M[MAR], PC ← PC+1
T2 : IR ← MBR
```

여기까지는 인출하기까지의 과정

인출한 이후, 명령어를 실행하는 과정

ADD addr 명령어 연산

```
T0 : MAR ← IR(Addr)
T1 : MBR ← M[MAR]
T2 : AC ← AC + MBR
```

이미 인출이 진행되고 명령어만 실행하면 되기 때문에 PC를 증가할 필요x

IR에 MBR의 값이 이미 저장된 상태를 의미함

따라서 AC에 MBR을 더해주기만 하면 됨

캐시메모리

정의 - 속도 차이에 따른 병목 현상을 줄이기 위한 메모리

장점 - 속도 빠름

단점 - 용량 적음, 비용 비쌈

캐시 메모리(Cache Memory)

속도가 빠른 장치와 느린 장치에서 속도 차이에 따른 병목 현상을 줄이기 위한 메모리를 말한다.

ex1) CPU 코어와 메모리 사이의 병목 현상 완화
ex2) 웹 브라우저 캐시 파일은, 하드디스크와 웹페이지 사이의 병목 현상을 완화

CPU가 주기억장치에서 저장된 데이터를 읽어들 때, 자주 사용하는 데이터를 캐시 메모리에 저장한 뒤, 다음에 이용할 때 주기억장치가 아닌 캐시 메모리에서 먼저 가져오면서 속도를 향상시킨다.

속도라는 장점을 얻지만, 용량이 적기도 하고 비용이 비싼 점이 있다.

CPU에는 이러한 캐시 메모리가 2~3개 정도 사용된다. (L1, L2, L3 캐시 메모리라고 부른다)

속도와 크기에 따라 분류한 것으로, 일반적으로 L1 캐시부터 먼저 사용된다. (CPU에서 가장 빠르게 접근하고, 여기서 데이터를 찾지 못하면 L2로 감)

듀얼 코어 프로세서의 캐시 메모리: 각 코어마다 독립된 L1 캐시 메모리를 가지고, 두 코어가 공유하는 L2 캐시 메모리가 내장됨

만약 L1 캐시가 128kb면, 64/64로 나누어 64kb에 명령어를 처리하기 직전의 명령어를 임시 저장하고, 나머지 64kb에는 실행 후 명령어를 임시저장한다. (명령어 세트로 구성, I-Cache - D-Cache)

- L1 : CPU 내부에 존재
- L2 : CPU와 RAM 사이에 존재
- L3 : 보통 메인보드에 존재한다고 함

캐시 메모리 크기가 작은 이유는, SRAM 가격이 매우 비쌈

디스크 캐시: 주기억장치(RAM)와 보조기억장치(하드디스크) 사이에 존재하는 캐시

캐시 메모리 작동 원리

- 시간 지역성

for나 while 같은 반복문에 사용하는 조건 변수처럼 한번 참조된 데이터는 잠시후 또 참조될 가능성이 높음

- 공간 지역성

A[0], A[1]과 같은 연속 접근 시, 참조된 데이터 근처에 있는 데이터가 잠시후 또 사용될 가능성이 높음

이처럼 참조 지역성의 원리가 존재한다.

캐시에 데이터를 저장할 때는, 이러한 참조 지역성(공간)을 최대한 활용하기 위해 해당 데이터뿐만 아니라, 옆 주소의 데이터도 같이 가져와 미래에 쓰일 것을 대비한다.

CPU가 요청한 데이터가 캐시에 있으면 'Cache Hit', 없어서 DRAM에서 가져오면 'Cache Miss'

캐시 미스 경우 3가지

1. Cold miss

해당 메모리 주소를 처음 불러서 나는 미스

2. Conflict miss

캐시 메모리에 A와 B 데이터를 저장해야 하는데, A와 B가 같은 캐시 메모리 주소에 할당되어 있어서 나는 미스 (direct mapped cache에서 많이 발생)

항상 핸드폰과 열쇠를 오른쪽 주머니에 넣고 다니는데, 잠깐 친구가 준 물건을 받느라 손에 들고 있던 핸드폰을 가방에 넣었음. 그 이후 핸드폰을 찾으려 오른쪽 주머니에서 찾는데 없는 상황

3. Capacity miss

캐시 메모리의 공간이 부족해서 나는 미스 (Conflict는 주소 할당 문제, Capacity는 공간 문제)

캐시 크기를 키워서 문제를 해결하려하면, 캐시 접근속도가 느려지고 파워를 많이 먹는 단점이 생김

구조 및 작동 방식

• Direct Mapped Cache

ARM은 칩의 기본 설계 구조만 만들고, 실제 기능 추가와 최적화 부분은 개별 반도체 제조사의 영역으로 맡긴다. 따라서 물리적 설계는 같아도, 명령 집합이 모두 다르기 때문에 서로 다른 칩이 되기도 하는 것이 ARM.

소비자에게는 칩이 논리적 구조인 명령 집합으로 구성되면서, 이런 특성 때문에 물리적 설계 베이스는 같지만 용도에 따라 다양한 제품군을 만날 수 있는 특징이 있다.

아무래도 아키텍처는 논리적인 명령 집합을 물리적으로 표현한 것이므로, 명령어가 많고 복잡해질수록 실제 물리적인 칩 구조도 크고 복잡해진다.

하지만, ARM은 RISC 설계 기반으로 '단순한 명령집합을 가진 프로세서가 복잡한 것보다 효율적'임을 기반하기 때문에 명령 집합과 구조 자체가 단순하다. 따라서 ARM 기반 프로세서가 더 작고, 효율적이며 상대적으로 느린 것이다.

단순한 명령 집합은, 적은 수의 트랜지스터만 필요하므로 간결한 설계와 더 작은 크기를 가능케 한다. 반도체 기본 부품인 트랜지스터는 전원을 소비해 다이의 크기를 증가시키기 때문에 스마트폰이나 태블릿PC를 위한 프로세서에는 가능한 적은 트랜지스터를 가진 것이 이상적이다.

따라서, 명령 집합의 수가 적기 때문에 트랜지스터 수가 적고 이를 통해 크기가 작고 전원 소모가 낮은 ARM CPU가 스마트폰, 태블릿PC와 같은 모바일 기기에 많이 사용되고 있다.

ARM의 장점은?

소비자에 있어 ARM은 '생태계'의 하나라고 생각할 수 있다. ARM을 위해 개발된 프로그램은 오직 ARM 프로세서가 탑재된 기기에서만 실행할 수 있다. (즉, x86 CPU 프로세서 기반 프로그램에서는 ARM 기반 기기에서 실행할 수 없음)

따라서 ARM에서 실행되던 프로그램을 x86 프로세서에서 실행되도록 하려면 (혹은 그 반대로) 프로그램에 수정이 가해져야만 한다.

하지만, 하나의 ARM 기기에 동작하는 OS는 다른 ARM 기반 기기에서도 잘 동작한다. 이러한 장점 덕분에 수많은 버전의 안드로이드가 탄생하고 있으며 또한 HP나 블랙베리의 태블릿에도 안드로이드가 탑재될 수 있는 가능성이 생기게 된 것이다.

(하지만 애플사는 iOS 소스코드를 공개하지 않고 있기 때문에 애플 기기는 불가능하다)

ARM을 만드는 기업들은 통해 전력 소모를 줄이고 성능을 높이기 위해 설계를 개선하며 노력하고 있다.

[참고 자료]

- [링크](#)