컴퓨터의 동작 원리

학습목표

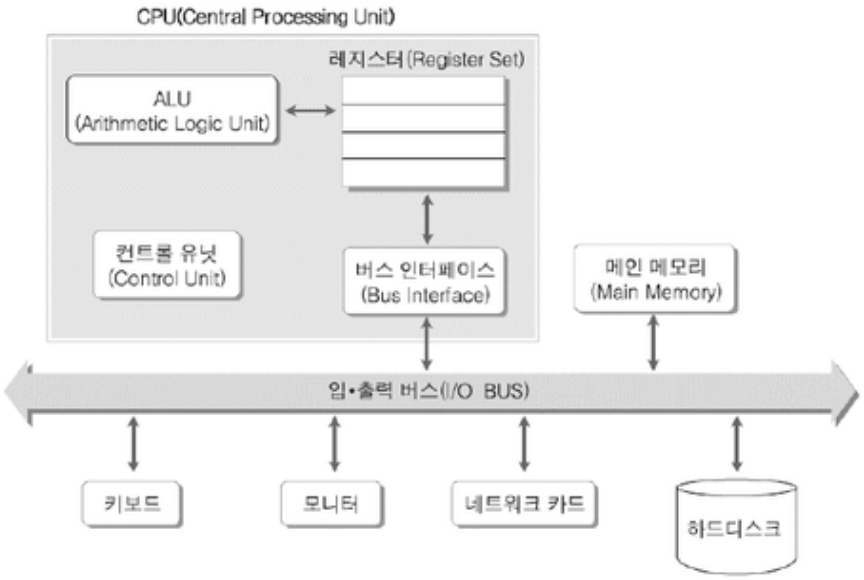
- 컴퓨터의 동작을 하드웨어와 소프트웨어 측면에서 이해할 수 있다.
- 고급 언어와 저급 언어의 차이점을 이해할 수 있다.
- 컴파일 언어와 인터프리트 언어의 차이점을 이해할 수 있다.

들어가며

개발자는 단순히 로직을 작성하지는 않는다. 현실적인 제약(컴퓨터의 물리적인 자원)을 고려하면서 문제를 해결할 줄 알아야 한다. 이를 위해 컴퓨터 과학에 대해서 공부할 필요가 있다. 그래야 여러 예외 상황을 예방할 수 있고, 효율적인 구현이 가능하기 때문이다. 이 시간에 모든 것을 기술할 순 없지만 최소한의 내용을 공부해보도록 하자.

컴퓨터의 구성 요소

현대적 컴퓨터는 매우 복잡하지만 단순하게 **메모리**(Memory), **입력과 출력**(I/O; Input and Output)장치, **CPU**로 나눌 수 있다. 그리고 이런 장치들은 **버스**(Bus)를 통해 연결된다. 각 부품은 각자만의 역할이 있으며, 이를 그림으로 나타내면 아래와 같다.



하나씩 살펴보자.

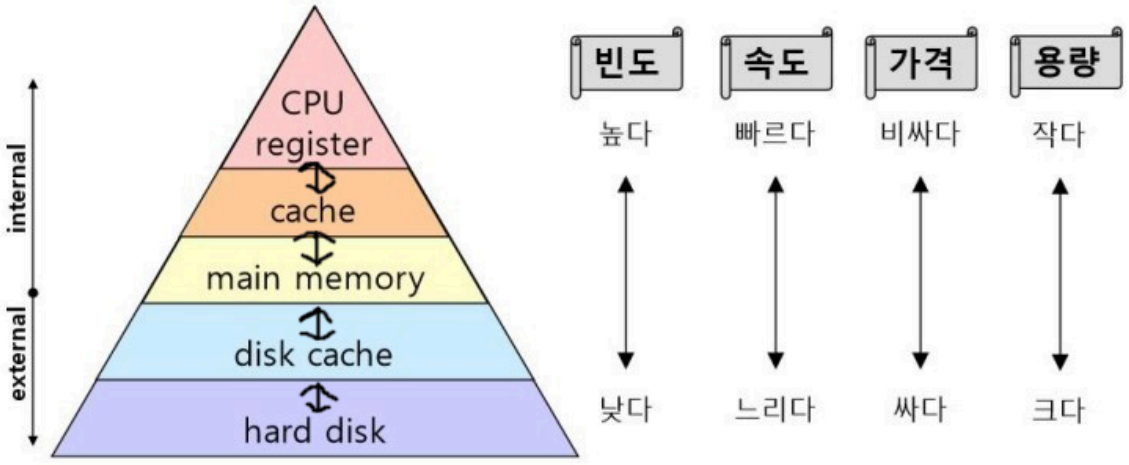
메모리

메모리(Memory)는 데이터를 저장하는 부품이다. 메모리는 여러 단독 주택이 모인 주택 복합 단지에 비유할 수 있는데, 각 집마다 **주소**(Address)가 있으며, 메모리를 사용하려면 꼭 주소를 알아야 한다.

각 집에 우리의 데이터가 저장된다.

- **메모리 계층 구조**

메모리의 종류에는 여러 가지가 있고, 각각의 쓰임새가 있다. 이를 표현한 계층도를 **메모리 계층 구조**(Memory Hierarchy)라 한다. 아래 그림에서 위에 위치할 수록 접근 빈도 수가 높으며 속도도 빠르지만, 가격이 비싸고 용량이 매우 작다.



메모리 계층 구조(memory hierarchy)

각 메모리는 윗 계층의 메모리로부터 데이터를 받아 저장하거나, 데이터를 건네준다*. 읽기 요청을 받았을 때, 그 요청을 처리할 수 있는 데이터가 존재하지 않을 경우 아래 계층의 메모리로부터 복사하며, 쓰기 요청을 받았을 때는 윗 계층으로부터 전달 받은 정보로 갱신한다. 위에서부터 차례대로 가볍게 살펴보자.

* 저장하는 것을 쓴다(Write)고 표현하고, 건네주는 것을 읽는다(Read)고 표현한다.

레지스터(Register)는 CPU 안에 존재하는 메모리로, 프로그램을 실행하는 데 필요한 값을 임시로 저장할 수 있다. 레지스터는 여러 개가 존재하며, 각기 다른 이름과 용도를 갖고 있다. **캐시**(Cache) 메모리는 주기억장치나 보조기억장치에 접근하는 시간을 줄이기 위한 임시 저장장치다. **주기억장치**(Main Memory)는 현재 실행하고 있는 프로그램에 대한 정보가 저장되는 부품이다.* 여기에는 **RAM**(Random Access Memory)과 **ROM**(Read Only Memory)이 있는데, 보통 RAM을 일컫는다.** 주기억장치의 단점은 저장 용량이 작고, 전원이 꺼지면 저장된 내용을 잃는다는 것이다.*** 그래서 이런 단점을 보완하기 위한 메모리가 **보조기억장치**(Secondary Memory)다. 하드 디스크, SSD, USB 메모리, DVD, CD-ROM 등이 여기에 속한다.

* 명령코드가 메모리에 저장되는 방식을 **프로그램 저장 방식 컴퓨터**(Stored-Program Computer)라고 한다.

** ROM에는 컴퓨터가 **부팅**(Bootstrap)되기 위한 프로그램이 저장된다.

*** 이런 성질을 **휘발성**(Volatility)이라 한다.

- **비트에 대한 이해**

살펴본 것처럼 메모리는 데이터를 저장하는 부품이다. 그런데, 데이터는 어떻게 저장이 될까? 컴퓨터가 사용하는 모든 데이터는 **비트**(Bit)로 저장된다. 비트는 Binary와 Digit의 합성어로, 2가지 상태를 나타낼 수 있는 숫자다.

비트에 저장할 수 있는 데이터는 무엇이든지 상관 없다. 그냥 수로서 0과 1이 될 수도 있고, 화재가 발생했다 / 발생하지 않았다, 왼쪽 / 오른쪽, 존재한다 / 존재하지 않는다 등 추상적인 의미도 저장할 수 있다. 즉, 메모리에는 비트가 저장된다고 할 수 있다. 하지만 비트가 나타낼 수 있는 상태는 오직 2가지이기에 비트 하나만으로는 많은 데이터를 저장할 수 없다. 그래서 컴퓨터는 비트를 여러 개 묶어 **바이트**(Byte)*라는 단위로 데이터를 다루게 된다.**

* 1바이트는 비트 8개이다. 즉 1byte = 8bit 이다.

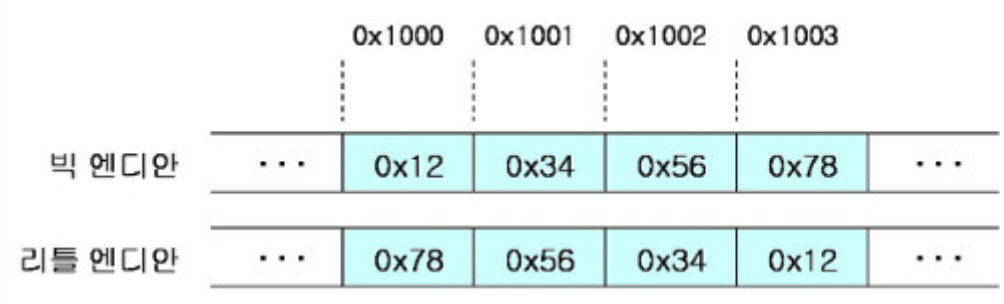
** 하지만 바이트도 사용자의 입장에서는 무척 작기 때문에 실제로는 KB, MB, GB, TB나 KiB, MiB, GiB, TiB 라는 단위를 사용한다.

간혹, **워드**(Word)라는 단위를 사용하기도 하는데, 이는 CPU가 한번에 처리할 수 있는 데이터 크기를 의미한다. CPU를 구매할 때나 프로그램을 설치할 때 32bit, 64bit 등의 용어를 확인할 수 있는데, 이것이 CPU가 한번에 처리할 수 있는 데이터 크기*라고 할 수 있다. 워드의 절반을 **하프 워드**(Half Word), 워드의 1배를 **풀 워드**(Full Word), 워드의 2배를 **더블 워드**(Double Word)라고 한다.

* 버스의 대역폭(Bandwidth)이라고 한다.

- **엔디안**

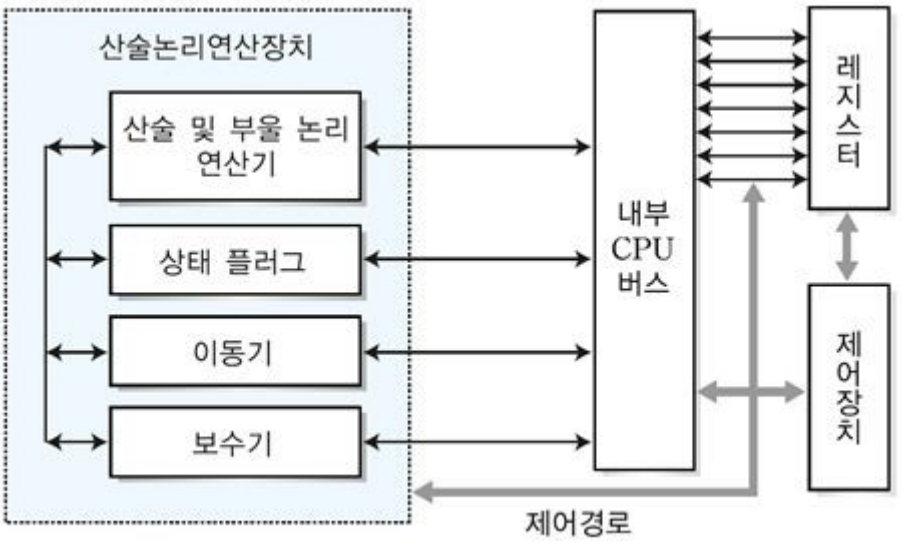
엔디안(Endian)은 비트를 저장하는 방식이다. 다른 컴퓨터로 데이터를 전송할 때는 이를 염두해 둬야 하는데, 데이터 순서가 뒤섞일 수 있기 때문이다. 첫 번째 바이트가 최하위 비트(LSB; Least Significant Bit)쪽에 위치하면 **리틀 엔디안**(little endian), 최상위 비트(MSB; Most Significant Bit)쪽에 위치하면 **빅 엔디안**(big endian)이라고 한다. 아래 그림은 같은 데이터가 엔디안에 따라 메모리에 어떻게 저장되는지를 나타낸 것이다.



엔디안

CPU

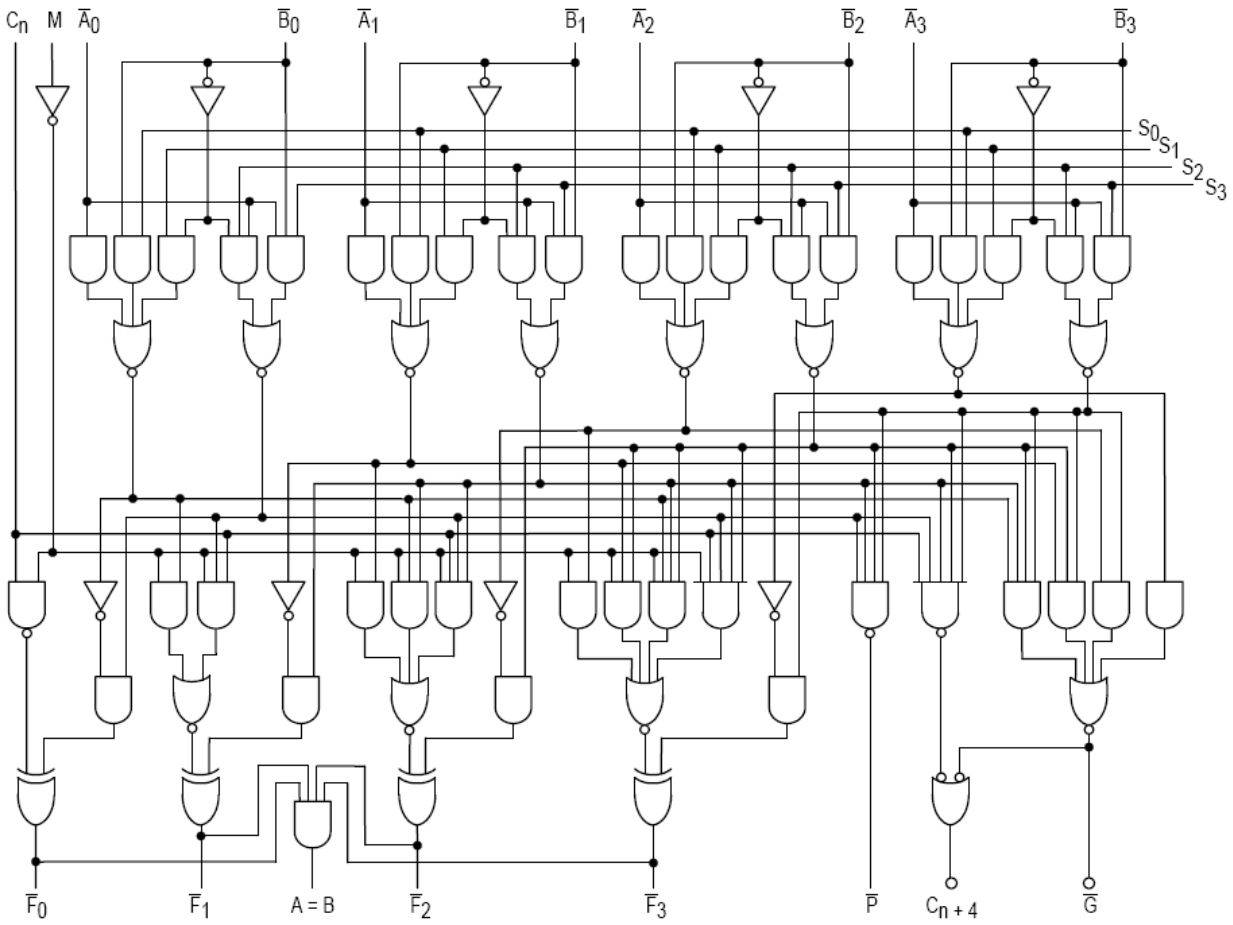
CPU(Central Processing Unit)는 우리가 입력한 명령어의 처리를 담당하는 부품이다. CPU도 컴퓨터처럼 여러 부품으로 구성되는데, 크게 산술 논리 장치와 레지스터 및 제어 장치로 구성된다.



CPU의 구성 요소

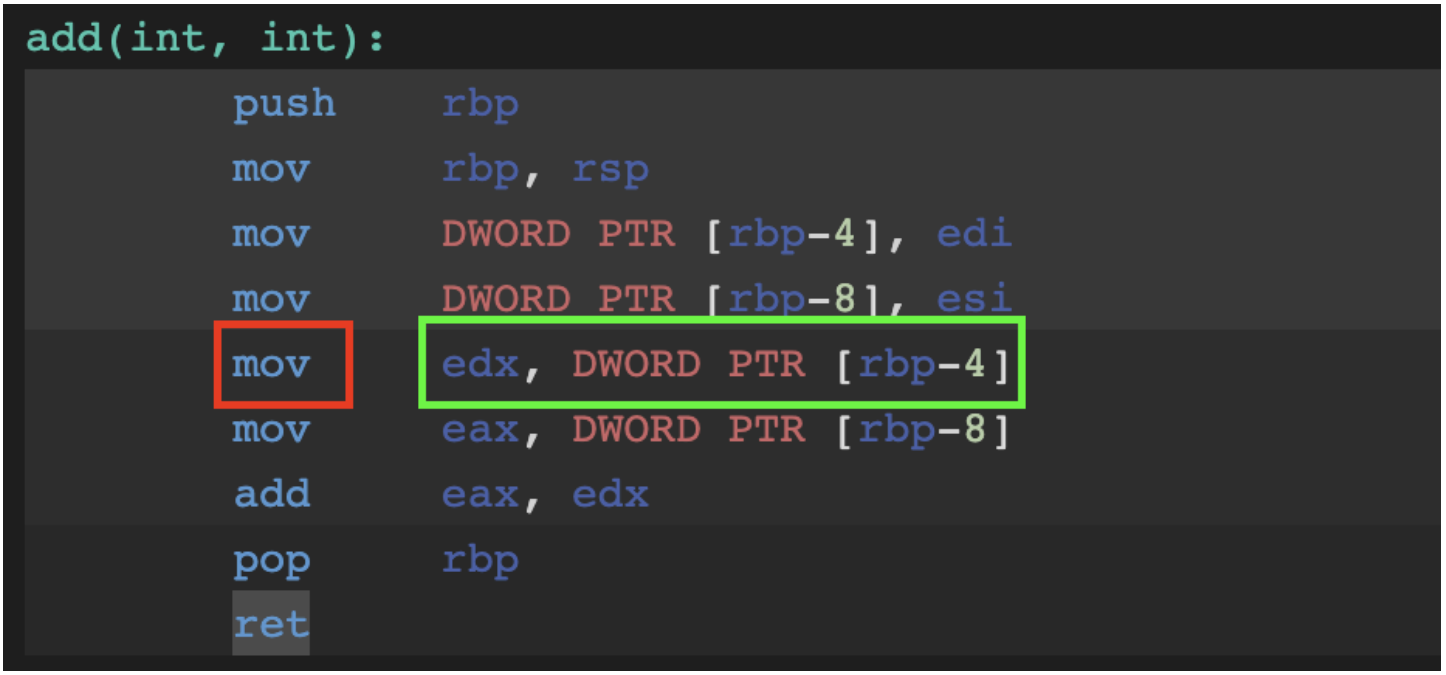
- **산술논리장치**

산술논리장치(ALU; Arithmetic Logic Unit)는 CPU의 핵심 부품으로 연산을 담당하는 장치다. 산술논리장치 내부에는 연산을 수행할 수 있는 여러 가지 논리 회로가 존재한다. 덧셈을 위한 가산기, 뺄셈을 위한 보수기, 시프트 연산을 위한 시프터 등 이러한 논리 회로 등이 얽히고 섞혀 연산을 수행하게 되는 것이다.



산술논리장치는 실제로 이런 회로다.

산술논리장치는 연산을 의미하는 **명령코드**(Opcode; Operation Code)와 **피연산자**(Operand)를 갖고 결과를 산출하게 된다. 명령코드는 CPU의 종류에 따라 다른데 이는 미리 정의되어 있다. 명령코드의 집합을 **명령어 집합**(Instruction Set)이라고 하며, 우리가 흔히 쓰는 x86이란 x86 명령어 집합 아키텍처를 일컫는 것이다.



빨간색이 명령코드고 초록색이 피연산자다.

x86 아키텍처의 명령어 개수는 매우 많고 복잡하다. 그래서 이런 컴퓨터를 **CISC**(Complex Instruction Set Computer)라고 한다. 반대로 모바일 칩셋에 들어가는 ARM 아키텍처의 경우 CISC에서 주로 사용되는 명령어만 남겼는데, 이를 **RISC**(Reduced Instruction Set Computer)*라고 한다.

* 명령어의 처리 방식이 메모리에서 데이터를 불러와(Load) 연산 후 다시 저장하는(Store) 절차로 이뤄져 Load-Store 아키텍처라고도 한다.

- **레지스터**

레지스터는 상술했듯 CPU에 있는 메모리다. ALU가 사용할 데이터, 명령코드, 결과 데이터 등 모두 레지스터에 저장된다. 다시 말해 메모리에 있는 데이터를 조작하기 위해서는 그 데이터를 레지스터로 가져와야 한다는 것이며, 메모리에 데이터를 저장하고 싶을 때도 레지스터에서 메모리로 보내는 과정이 필요하다.

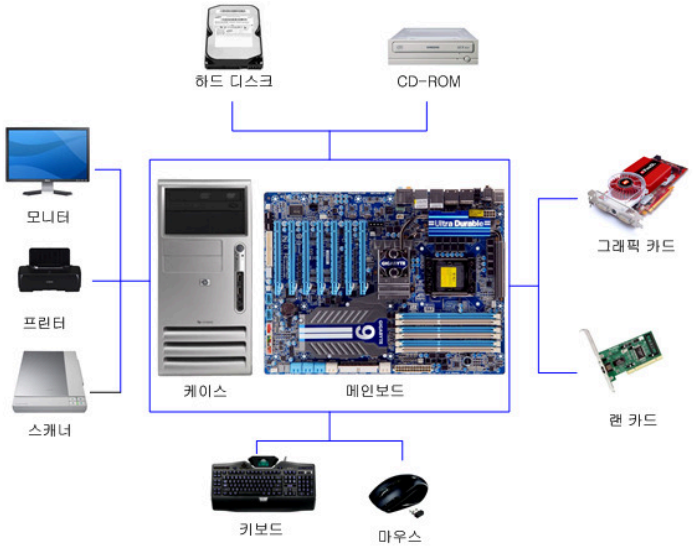
- **제어 장치**

컴퓨터의 내부 부품들은 멋대로 동작하지 않는다. 복잡한 교차로에서 경찰이 도로의 질서를 정리하는 것처럼 컴퓨터 또한 이런 경찰의 역할을 하는 부품이 필요하다. 이런 신호를 **제어 신호**(Control Signal)라고 하며, **제어 장치**(Control Unit)*는 제어 신호를 내보내, 메모리에서 명령코드*와 피연산자들을 가져와서 ALU에게 어떤 연산을 수행할지 알려주고, 결과를 메모리에 돌려준다. 제어 장치 회로의 예시는 여기서 확인하자.

* **실행 장치**(Execution Unit)라고도 한다.

입출력 장치

컴퓨터는 계산하다는 의미의 Compute와 행위자를 뜻하는 er의 합성어로 본래 계산을 위해 만들어진 기계이다. 그럼 컴퓨터를 사용하기 위해 우리의 데이터를 넣어주는 방법과 그 결과를 보는 방법이 필요할 것이다. 컴퓨터로 데이터를 주는 것을 **입력**(Input)이라고 하며, 컴퓨터가 우리에게 데이터를 주는 것을 **출력**(Output)이라고 한다. 입출력 장치는 여러 가지가 있으며 우리가 그 중 흔히 사용하는 입력 장치는 키보드 및 마우스, 출력 장치는 모니터라고 할 수 있다.



입출력 장치는 다양하다.

- **폴링 레이트**

그러면 컴퓨터는 입력을 어떻게 감지할까? 즉, 마우스의 버튼이 눌렸는지, 키보드의 버튼이 눌렸는지 등을 어떻게 알 수 있을까? 각 입력 장치는 특정 주기를 가지고, 입력 여부를 감지해 컴퓨터로 보낸다. 이를 **폴링 레이트**(Polling Rate)라고 한다. **폴링**(Polling)은 주기적으로 무언가를 처리하는 기법을 의미하는데, 예를 들어 어떤 입력 장치의 폴링 레이트가 1000Hz라고 한다면 1000분의 1초마다 입력을 감지한다는 것을 의미한다.

- **리프레시 레이트**

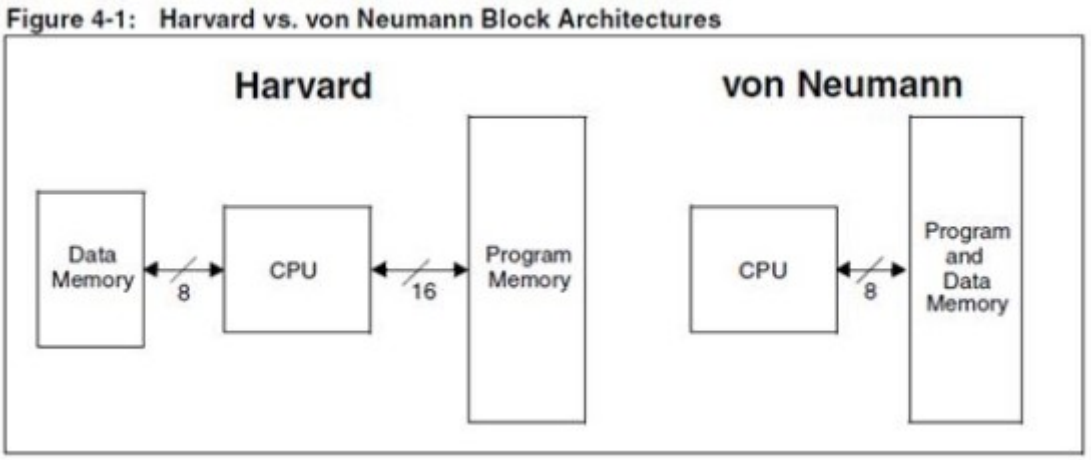
그러면 모니터는 어떻게 화면을 그리게 될까? 폴링 레이트와 비슷하게 **리프레시 레이트**(Refresh Rate)가 있다. 모니터는 특정 주기마다 새로운 화면을 그리는 데, 이를 **리프레시**(Refresh)라고 한다. 어떤 모니터의 리프레시 레이트가 75Hz라면 1초에 75번 화면을 그려주는 것이다. 헌데, 여기서 문제가 생긴다. 게임의 프레임과 모니터의 리프레시 레이트가 일치하지 않는다면 **테어링***(Tearing) 현상이나 **스터터링****(Stuttering) 현상이 발생할 수 있다. 이를 해결하기 위한 방법이 **수직 동기화**(Vertical Synchronization)이다.

* 테어링 예시는 여기에서 확인할 수 있다.

** 스터터링 예시는 여기에서 확인할 수 있다.

컴퓨터 아키텍처

컴퓨터 아키텍처(computer architecture)는 컴퓨터의 여러 구성요소를 배치하는 방법을 말한다. 가장 흔한 컴퓨터 구조는 **폰 노이만**(Von Neumann) 구조와 **하버드**(Harvard) 구조다. 두 구조는 메모리 배열 외에는 모두 동일한 구조로 하버드 구조가 명령어와 데이터를 동시에 가져올 수 있어 좀 더 빠르지만 두 번째 메모리를 처리하기 위한 버스가 더 필요하다.



하버드 구조와 폰 노이만 구조

운영체제

자, 컴퓨터를 조립하는 방법까지 알았으니 이제는 정말 사용할 수 있는 것일까? 그렇지 않다. 각 부품을 전체적으로 관리해줄 프로그램이 필요하다. 이를 **운영체제**(Operating System)라고 한다. 운영체제는 프로그램을 실행하기 위해서 메모리, CPU와 같은 여러 물리적인 자원을 관리한다. 모든 응용 프로그램은 운영체제 위에서 동작한다는 것을 잊지 말자.

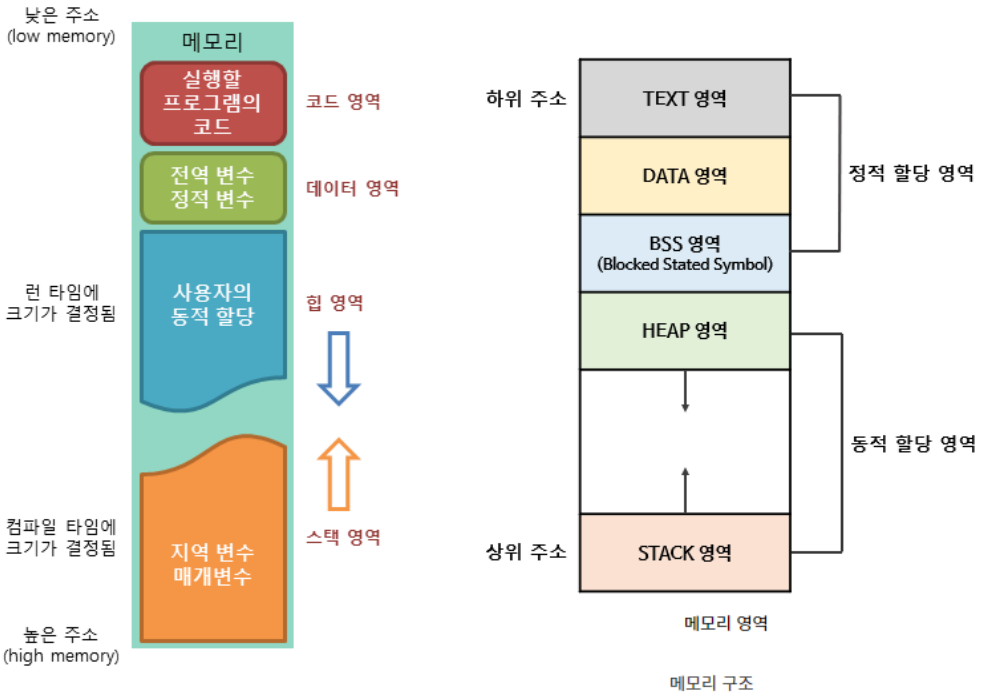
프로세스

그럼 프로그램을 실행하면 무슨 일이 일어날까? 프로그램을 실행하면 **프로세스**(Process)가 된다. 프로세스는 운영체제로부터 자원을 할당 받은 개체*를 의미한다. 프로세스는 프로그램을 실행하기 위한 여러 가지 데이터를 갖고 있으며, 아래와 같이 4가지 영역이 존재한다.

- 코드(Code) : 명령어가 저장되는 공간이다.
- 데이터(Data) : 정적** 데이터가 저장되는 공간으로 BSS 세그먼트와 데이터 세그먼트로 나뉜다.
- 힙(Heap) : 동적** 할당 영역으로 하위 주소부터 시작한다.
- 스택(Stack) : 함수를 동작하기 위한 데이터가 저장되는 동적 할당 영역으로 상위 주소부터 시작한다.

* 쉽게 얘기하면 실행되고 있는 프로그램이다.

** 컴퓨터 공학에서 정적(Static)과 동적(Dynamic)은 자주 등장하는 용어인데, 보통 정적은 프로그램 실행 전에 무언가를 한다는 것이고, 동적은 프로그램 실행 시간에 무언가를 한다는 것이다.



메모리 구조

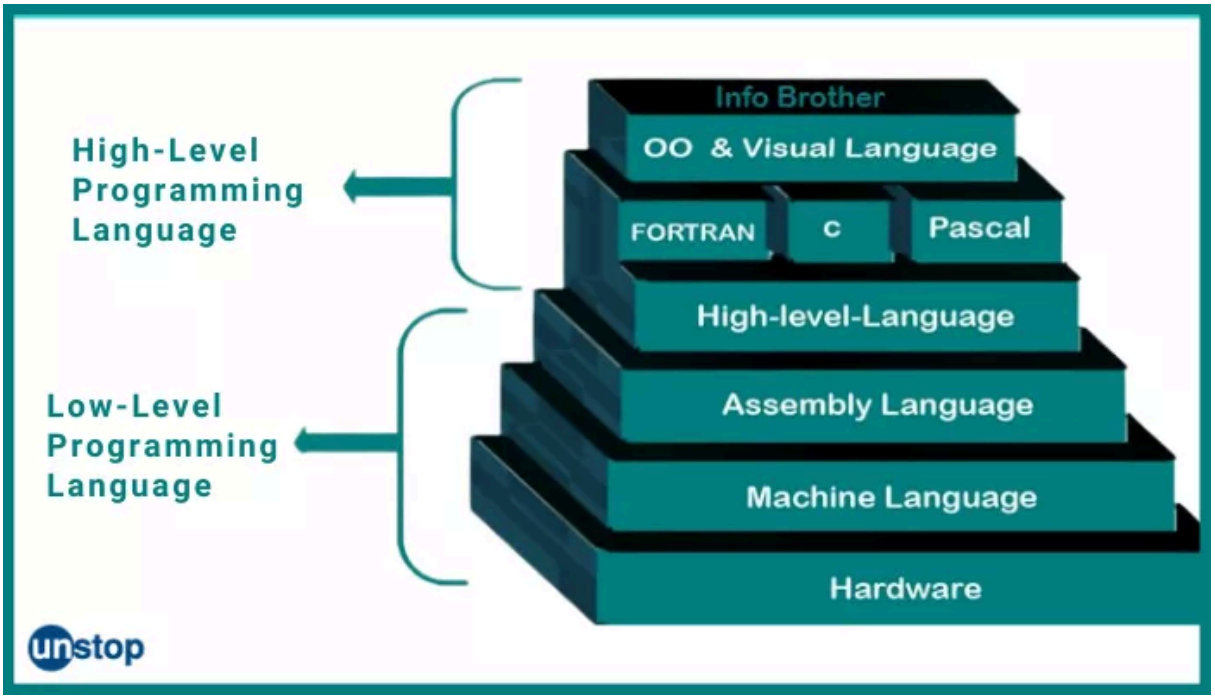
[우리의 프로그램은 메모리를 위와 같이 사용한다.]

스레드

스레드(Thread)는 프로세스 내에서 실행 흐름의 단위를 말한다. 프로세스는 데이터만 관리하고, 이 데이터를 갖고 실행을 담당하는 건 스레드이다. 프로세스는 하나 이상의 스레드를 가질 수 있는데, 이를 **멀티스레드**(Multi-thread)라고 한다. 스레드가 여러 개면 여러 개의 실행 흐름을 가지므로, 여러 가지 일을 동시에 병렬적으로 처리할 수 있다.

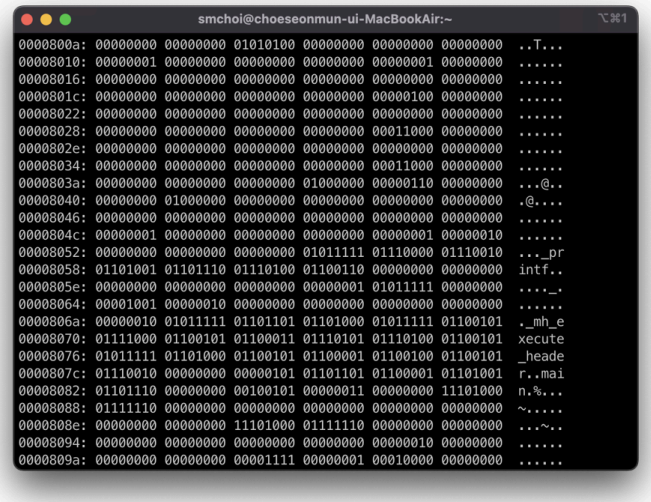
고급 언어와 저급 언어

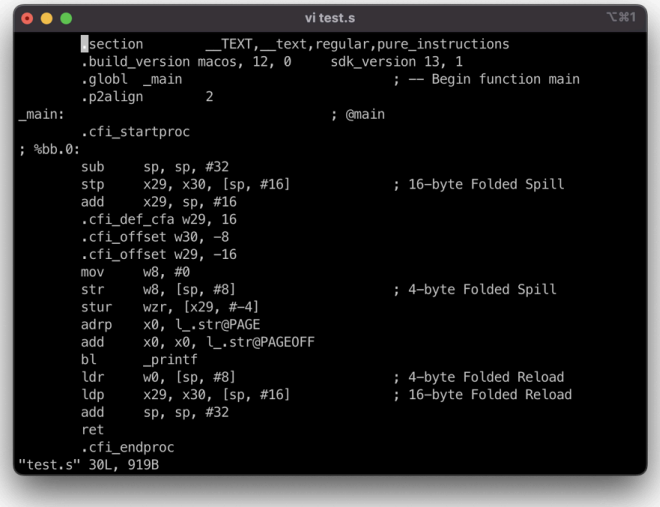
이번에는 프로그래밍 언어의 종류에 대해서 알아보자. 프로그래밍 언어에는 **저급 언어**(Low-level Programming Language)와 **고급 언어**(High-level Programming Language)가 있는데, 기계에 가까울수록 저급이라고 표현하며, 사람에게 가까울수록 고급이라고 표현한다. 다시 말해 저급 언어는 컴퓨터가 이해하기 쉬운 언어이며, 고급 언어는 사람이 이해하기 쉬운 언어라고 할 수 있다.



컴퓨터와 가깝냐, 사람과 가깝냐에 따라 프로그래밍 언어의 수준을 나눈다.

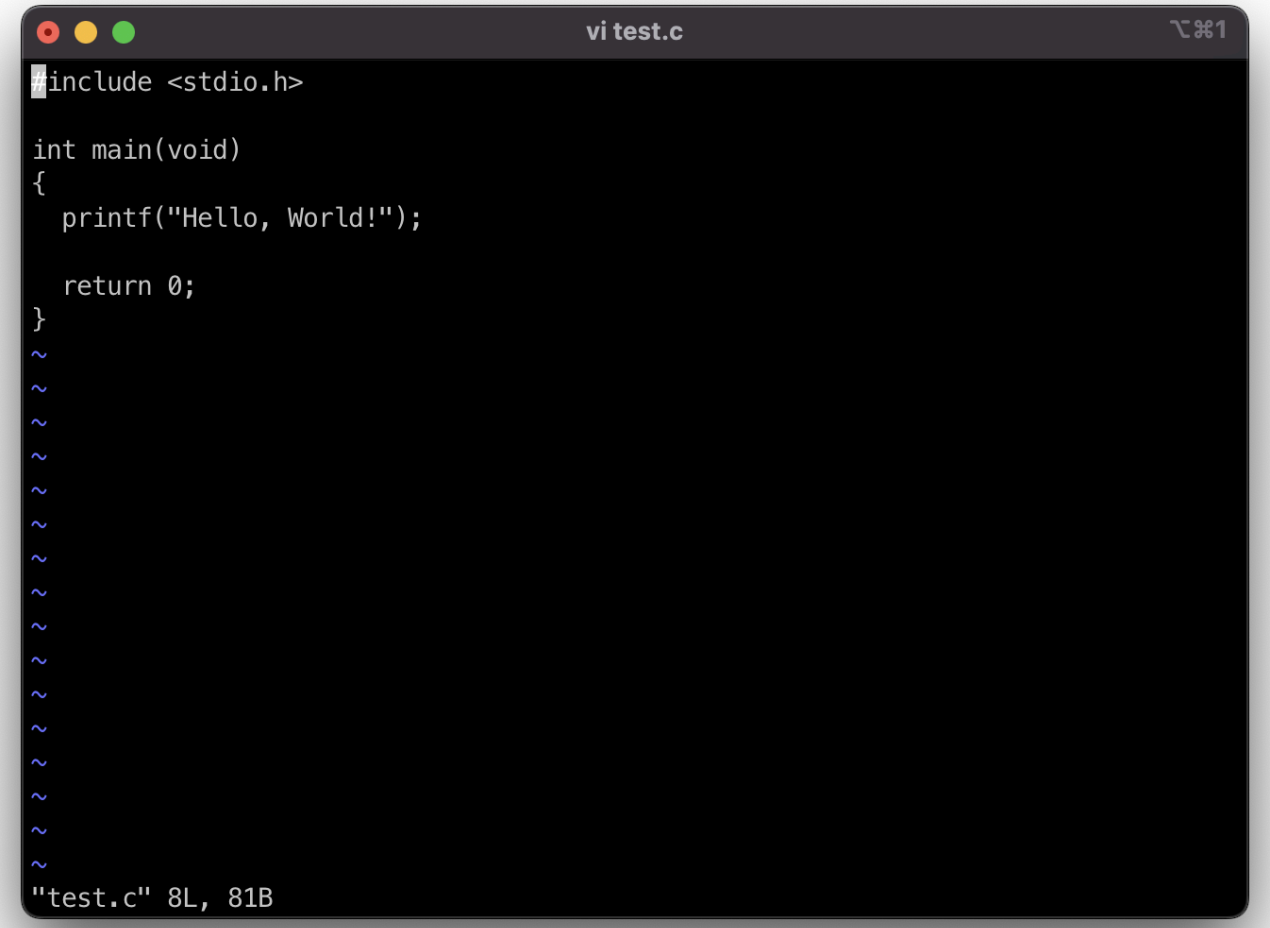
프로그램의 명령어는 메모리에 저장된다. 즉, 명령어도 0과 1로 이루어져 있으며 이를 **기계어**(Machine Language)라고 한다. 하지만 사람이 기계어를 읽기에는 굉장히 힘들기 때문에 이와 1:1로 대응되는 **어셈블리어**(Assembly Language)가 있다. 이 두 언어는 저급 언어에 속하며, 이 외의 C, C++, C#, Java, Javascript, Python 등의 언어는 고급 언어다. 예시를 하나 보도록 하자.





왼쪽은 기계어, 오른쪽은 어셈블리어다.

왼쪽 이미지와 오른쪽 이미지 모두 같은 프로그램이다. 기계어는 아예 사람이 읽을 수 없고, 어셈블리어는 그나마 기계어보다는 낫지만 그래도 꽤 복잡하다.



C언어로 작성된 프로그램

위 프로그램은 아까 본 기계어, 어셈블리어로 작성된 프로그램과 똑같은 동작을 한다. 확실히 저급 언어에 비해 작성과 읽기가 굉장히 쉽다는 것을 알 수 있다.

컴파일과 인터프리트

현대의 프로그램은 거의 대부분 고급 언어를 이용해 작성되지만*, 프로그램을 실행하려면 결국 고급 언어로 작성된 코드를 저급 언어로 변환하는 과정이 필요하다. 고급 언어를 저급 언어로 변환하는 방법에는 2가지가 있다. 바로

컴파일(Compile)과 **인터프리트**(Interpret)다. 컴파일은 코드** 전체를 기계어로 변환하는 것이며, 인터프리트는 코드를 실행하는 동시에 한 줄씩 기계어로 변환하는 것이다.*** 둘의 차이점을 표로 나타내면 아래와 같다.

* 임베디드 개발자, 게임 개발자, 정보 보안 등 하드웨어와 가까운 프로그래머는 저급 언어를 사용하기도 한다. 왜냐하면 어셈블리어를 읽으면 컴퓨터가 프로그램을 어떤 과정으로 실행하는지 가장 근본적인 단계에서부터 하나하나 추적하고 관찰할 수 있어 고성능 프로그램이나 찾기 어려운 오류를 발견할 수 있기 때문이다.

** 작성된 파일을 소스 코드(Source Code) 혹은 소스 파일(Source Flie)이라 한다.

*** 컴파일이 있는 언어를 컴파일 언어(Compiled Language), 그 외를 인터프리트 언어(Interpreted Language)라 한다.

구분	컴파일(Compile)	인터프리트(Interpret)
변환 주체	컴파일러(Compiler)	인터프리터(Interpreter)
변환 시점	실행 전	실행 중
변환 단위	코드 전체	코드 한 줄
결과물	실행 파일(.exe, .app 등) 혹은 라이브러리 파일(.lib, .dll 등)	X(인터프리터가 코드를 읽어서 실행하므로 결과물이 필요 없음)
오류 처리	오류를 코드 실행 전 발견 가능	오류를 코드 실행 중 발견 가능
실행 속도	빠름	느림
플랫폼 독립성	낮음(OS에 맞는 실행 파일 필요)	높음(인터프리터만 있으면 실행 가능)
코드 보호	기계어이므로 원본 소스 코드 파악이 어려움	원본 소스 코드가 그대로 노출됨
대표 언어	C++, C#, Java, Go 등	Python, Javascript, Ruby 등

즉, 컴파일과 인터프리트의 가장 큰 차이는 변환이 일어나는 시점과 실행 단위라 할 수 있다. C#, Java와 같은 일부 언어는 두 과정이 동시에 존재하기도 한다. 이는 추후 더 알아본다.

더 나아가기

1. 컴퓨터는 어떤 부품으로 구성되어 있으며, 각 부품은 무슨 역할을 하나요?
2. CPU를 구성하는 세 가지 주요 부품(ALU, 레지스터, 제어 장치)을 나열하고, 각 부품의 역할을 간략히 서술하세요.
3. 교안에 제공된 부품 외에는 어떤 부품이 있나요? 그리고 그것의 역할은 무엇인가요?
4. 컴퓨터의 메모리 계층 구조를 속도, 가격, 용량을 기준으로 설명하고, 각 계층에서 CPU에 가장 가까운 두 가지 메모리(가장 상위)를 나열해 보세요.

5. 비트에 담을 수 있는 데이터를 5가지 나열해보세요.
6. 컴퓨터 아키텍처엔 무엇이 있고, 어떤 차이점이 있는지 서술해주세요.
7. 테어링과 스터터링의 차이점은 무엇일까요?
8. 여러분이 사용하고 있는 입출력 장치의 폴링 레이트, 리프레시 레이트는 어떻게 되나요?
9. 운영체제의 종류로는 어떤 것이 있나요?
10. 프로세스와 스레드의 차이는 무엇인가요?
11. 프로세스의 네 가지 메모리 영역(코드, 데이터, 힙, 스택) 중 동적 할당 영역 두 가지의 시작 주소 방향(상위 주소 또는 하위 주소)을 구분하여 설명해 주세요.
12. 아래의 레지스터에 대해 알아보세요.
    12.1. 프로그램 카운터(PC; Program Counter)
    12.2. 스택 포인터(SP; Stack Pointer)
    12.3. 명령어 레지스터(IR; Instruction Register)
    12.4. 메모리 주소 레지스터(MAR; Memory Address Register)
    12.5. 메모리 버퍼 레지스터(MBR; Memory Buffer Register)
    12.6. 플래그 레지스터(Flag Register)
13. 엔디안이 무엇인가요?
14. CISC와 RISC의 차이점은 무엇인가요?
15. 고급 언어와 저급 언어의 차이점은 무엇인가요?
16. 컴파일 방식과 인터프리트 방식을 '변환 시점'과 '변환 단위'를 중심으로 비교하여 설명하세요.

참고자료

- 한 권으로 읽는 컴퓨터 구조와 프로그래밍 - YES24
- Introduction to Operating Systems
- 혼자 공부하는 컴퓨터 구조+운영체제 | 강민철 - 모바일교보문고