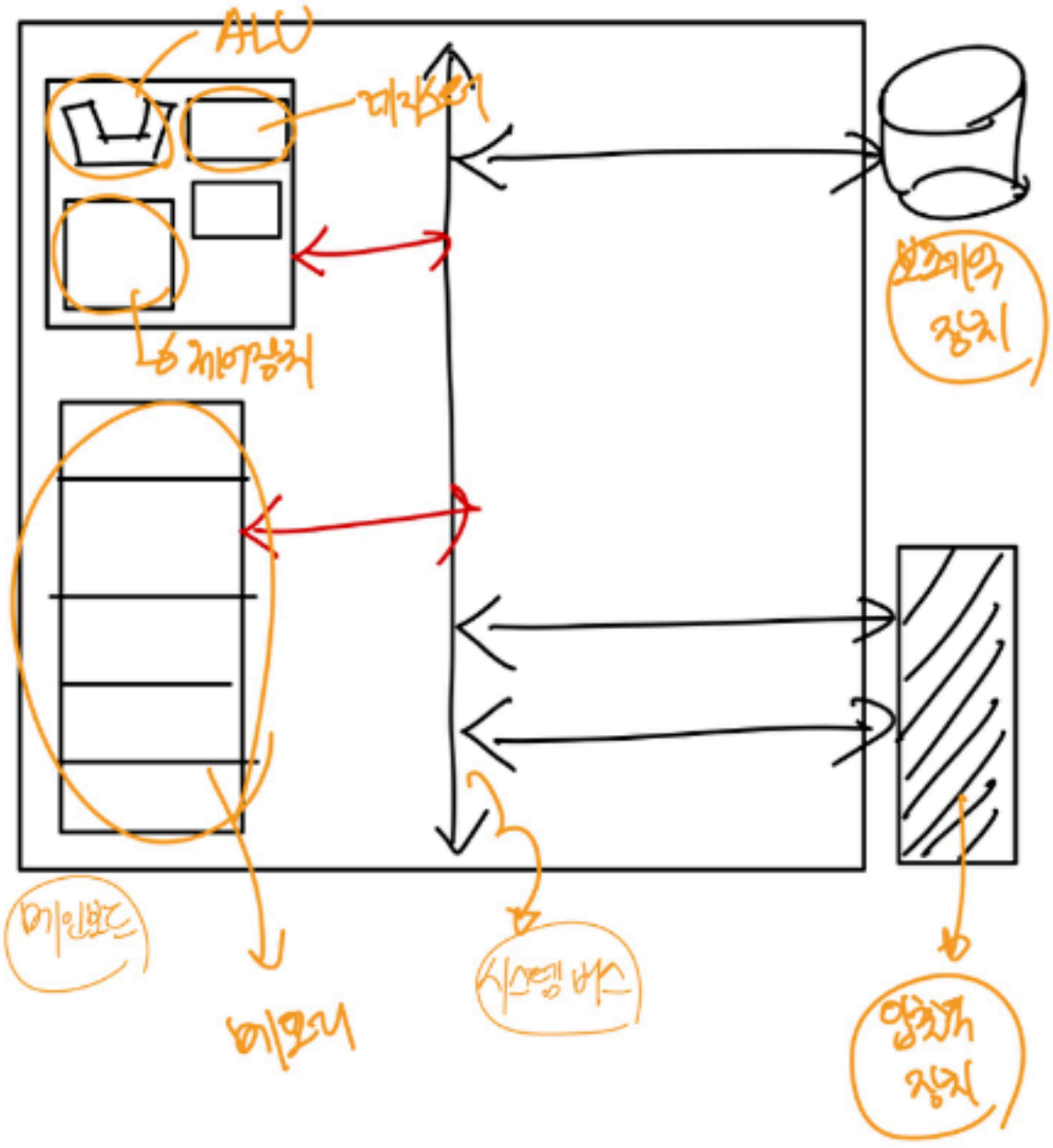
혼공컴운 1주차 발표

#스터디/CS스터디/혼공컴운

챕터1) 컴퓨터 구조 시작하기

컴퓨터의 4가지 핵심부품

- **중앙처리장치** (CPU: Central Processing Unit)
- **주기억장치** (메모리: Main Memory)
- **보조기억장치** (Secondary Storage)
- **입출력장치** (I/O Device)



메모리

- 현재 실행되는 프로그램의 명령어와 데이터를 저장하는 부품
- 프로그램이 실행되려면 반드시 메모리에 저장되어야 함
- 메모리에 저장된 값에 빠르고 효율적인 접근 ➜ **주소(address)**

CPU (Central Processing Unit)

- **컴퓨터의 두뇌**
- 메모리에 저장된 명령어를 읽어 들이고, 읽어 들인 명령어를 해석하고 실행하는 부품
- 연산 장치
- **ALU, 레지스터, 제어장치**
  - ALU: 계산기 (연산 담당)
  - 레지스터: CPU 내부의 임시 저장 장치 (굉장히 비싼 자원, 여러개지만 많지않음, 고유이름)
  - 제어장치: 제어신호라는 전기신호를 내보내고 명령어를 해석하는 장치
    - 메모리 읽기와 메모리 쓰기

보조기억장치

- **메모리는 가격이 비싸 저장 용량이 적음. 그리고 전원이 꺼지면 저장된 내용을 잃음.**
- ex. 워드에서 문서 작성하다가 컴퓨터가 꺼지면 내용이 다 날라감
  (물론 지금은 시대가 좋아서 복구가 됨)
- 그래서, **메모리보다 크기가 크고 전원이 꺼져도 저장된 내용을 잃지 않는 메모리를 보조할 저장장치**가 필요했는데 이게 보조기억장치이다.

- **하드디스크, SSD, USB 메모리, DVD, CD-ROM**과 같은 저장장치를 의미한다.
- 메모리는 현재 `실행되는` 프로그램을 저장한다면, 보조기억장치는 `보관할` 프로그램을 저장

입출력장치

마이크, 스피커, 프린터 등 컴퓨터 외부에 연결되어 컴퓨터 내부와 정보를 교환하는 장치

- 보조기억장치로 알고 있었던 **하드디스크, USB 메모리** 등도 **"컴퓨터 외부에 연결되어 컴퓨터 내부와 정보를 교환할 수 있는 장치"** 로 볼 수 있지 않음? 그러면 결국에는 입출력 장치 아닌지?
  - 관점에 따라 입출력장치의 일종으로 볼 수 있음.
  - 실제로 보조기억장치와 입출력장치를 `주변장치`라고 통칭하기도 한다.
  - 다만, 일반적인 입출력장치에 비해 메모리를 보조한다는 이유가 있어서 보조기억장치라고 칭했다.

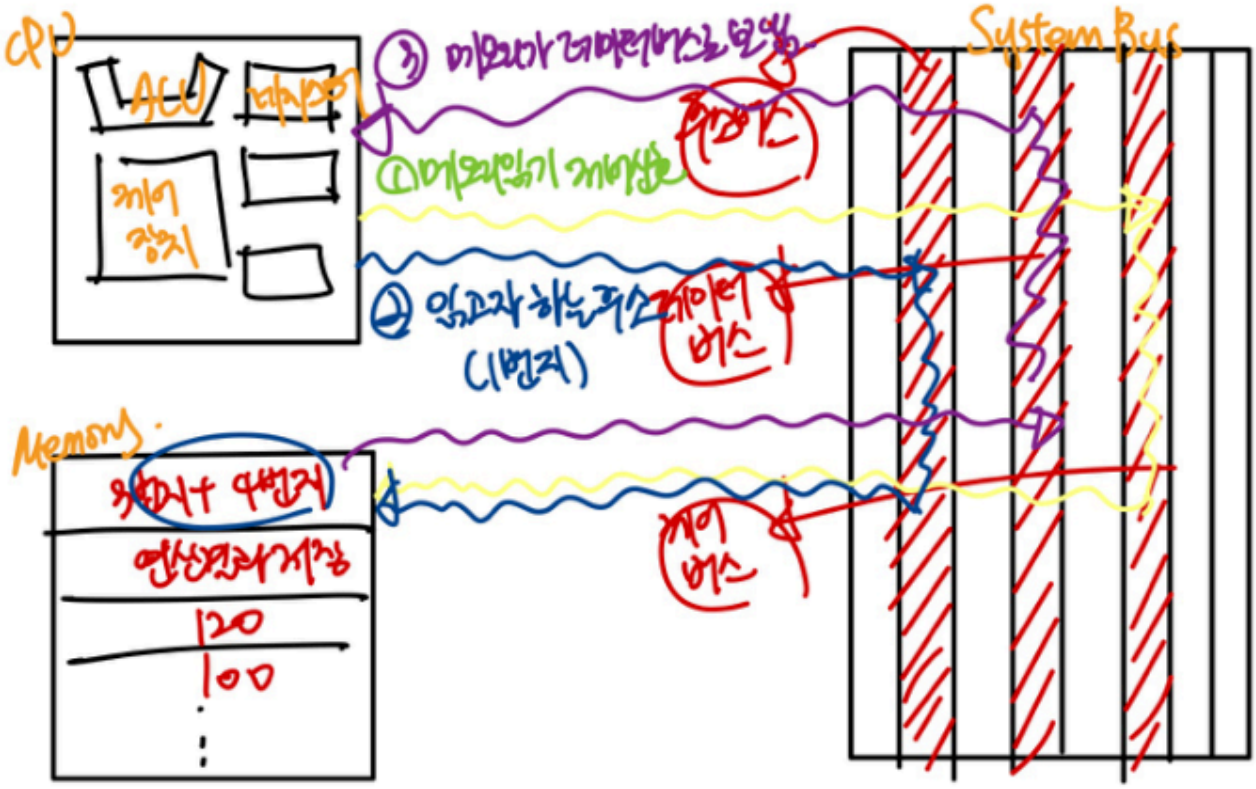
메인보드와 시스템 버스

- 컴퓨터의 핵심 부품들은 메인보드라는 판에 연결됨.
- **메인보드 = 마더보드**
- 메인보드에는 부품을 비롯한 여러 컴퓨터 부품을 부착할 수 있는 슬롯과 연결단자가 있음.

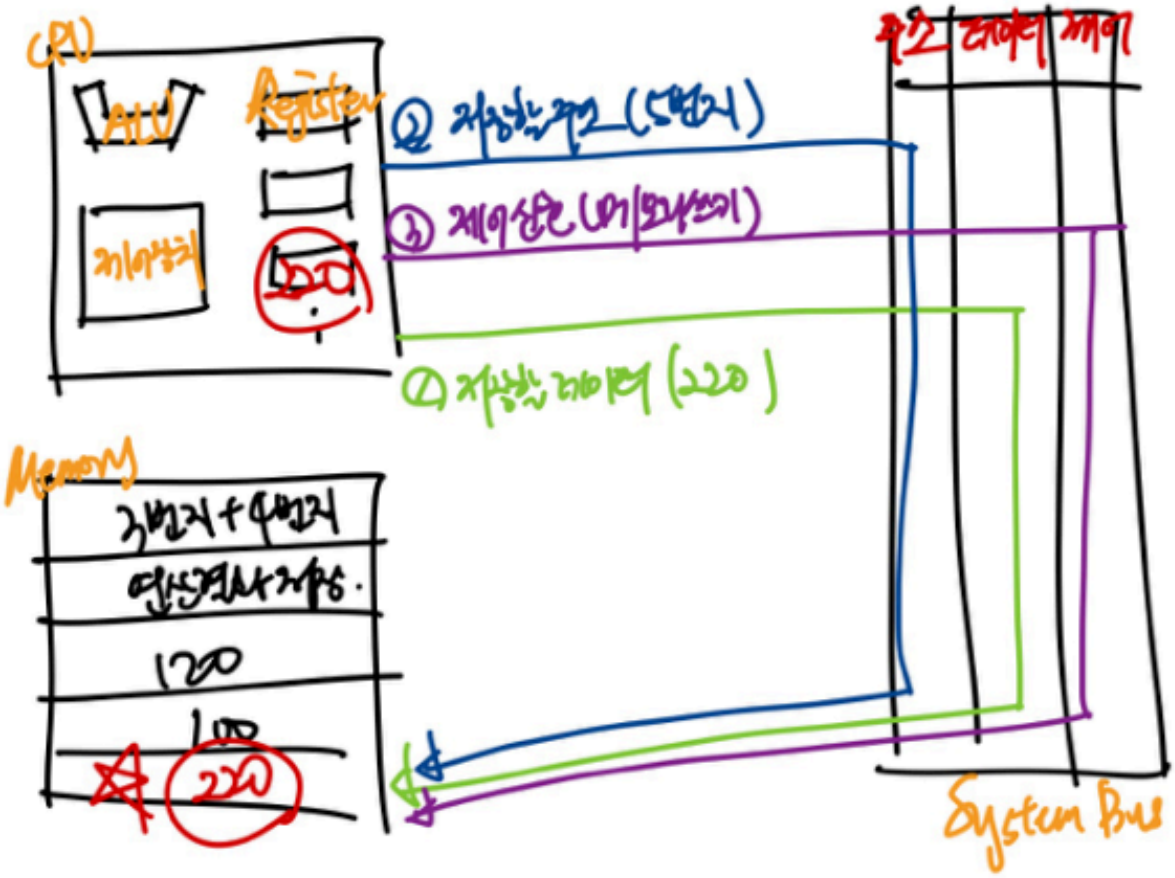
- 메인보드에 연결된 부품들은 서로 정보를 주고 받을 수 있음 ➜ 메인보드 내부에 **버스**라는 통로가 있어서!
- 컴퓨터 내부에는 다양한 버스가 있는데 컴퓨터의 4가지 핵심 부품을 연결하는 가장 중요한 버스가 `시스템 버스` 이다.
- 즉, 컴퓨터의 4가지 핵심 부품이 서로 정보를 주고받는 통로가 `시스템 버스` 이다.

시스템 버스

주소 버스, 데이터 버스, 제어 버스로 구성되어 있음.
- 주소 버스: 주소를 주고받는 통로
- `데이터 버스`: 명령어와 데이터를 주고받는 통로
- 제어 버스: 제어 신호를 주고받는 통로



- CPU가 메모리를 읽을 땐 제어 신호만 보내지 않는다.
- (1) CPU가 `제어 버스`로 `메모리 읽기` 제어 신호를 보냄
- (2) CPU가 `주소 버스`로 읽고자 하는 주소를 내보낸다.
- (3) 메모리는 `데이터 버스`로 CPU가 요청한 주소에 있는 내용을 보낸다.



참고로, 컴퓨터의 4가지 핵심 부품은 **메인보드**에 연결되어 **시스템 버스**를 통해 서로 정보 또는 데이터를 주고받는다.

챕터2) 데이터

- **컴퓨터는 0또는 1밖에 이해하지 못함 → `비트(bit)`**
- 예를 들어, 1byte라고 한다면 8bit이고 이는 2^8로 256개의 정보를 표현할 수 있음.
- **꿀팁) 4bit → 8421**

이진수의 음수 표현 → 보수 사용

보수의 개념(2진법과 비트, 바이트, 1의 보수, 2의 보수)

보수(Complement)

- 컴퓨터는 오로지 덧셈만을 이용해서 연산을 처리함
- 이 과정에서 음수를 표현하거나 음수 연산할 때 보수라는 개념이 필요함.
- **즉, 덧셈을 이용해서 뺄셈을 구현하는 방식.**

이진수에서의 1의 보수 표현법

- **1은 0으로, 0은 1로만 바꿔주면 된다.**
- 즉, **XOR (Exclusive OR) 연산**을 진행하면 된다.
- e.g) 0b1010의 1의 보수 ➜ 1010 + 0000 (XOR 연산) ➜ **0101**

1의 보수를 활용한 연산

- 10진수 13과 10진수 10을 빼보자. (13 - 10) ➜ 13 + (-10)
- 1101 - 1010 = **이진수 0011**

- 컴퓨터에서 -10을 표현하기 위해서는 1의 보수를 사용해야 한다.
- 10은 1010 인데 -10은 1의 보수인 0101이다.
- 따라서 1101 + 0101을 하면 **10010** 이 나온다.
- 근데 왜 이진수 0011일까?
  - **1의 보수 연산에서는 연산 결과가 한 자리 더 길어지면 캐리 값을 최하위 비트에 더해줘야 한다.**
  - **따라서 0010 + 1을 하게 되면 0011이 되는 것이다.**

1의 보수의 문제점

- **+0은 0000 0000 으로 나타낼 수 있음.**
- **그런데 -0은 1111 1111로 1의 보수로 인해 0을 표현하는 방법이 2개가 존재하는 것.**
- 그래서 이런 문제를 해결하기 위해 `2의 보수`이다.

이진수에서의 2의 보수 표현법

2의 보수 (two's complement)

- **모든 0과 1을 뒤집음 (1의 보수)**
- **1의 보수에 1을 더해준다.**

13 - 10 = 3 을 구해보자.

```
(1) 13의 이진수 -> 1110
(2) 10의 이진수 -> 1010
(3) -10이기 때문에 10의 2의 보수를 구해보자
   (3-1) 1과 0을 뒤집음 -> 0101
   (3-2) 1을 더함 -> 0101 + 1 -> 0110
(4) 1101과 0110을 더한다
1101
0110
----
10011

(5) 2의 보수에서 캐리가 발생한 경우 캐리를 그냥 버린다 -> 10011 -> 0011
```

1의 보수의 문제점을 해결한 2의 보수 표현법

> 2의 보수에서 0을 표현하는 방법은 **1개**이다.
>
> - **+0: 0000 0000**
> - **-0: 0000 0000**
> - 왜냐하면 2의 보수를 취하는 과정에서 캐리가 발생했고 이를 버렸기 때문이다.
> - 실제로 캐리를 버리는 것은 아니고 CPU 레지스터 중 Flag 레지스터에 저장한다.
> - **캐리가 발생하면 음수 표현!**

2의 보수 표현의 한계

> - 10진수 8을 2진수로 표현 ➜ 1000
> - 8을 음수로 표현 ➜ 0111 + 1 ➜ 1000
> - **즉, 음수로 표현한 값이 양수로 표현한 값과 같음.**
> - **다시 말해서 보수를 취했을 때 값이 자기 자신이 되어버리면 해결할 수 있는 방법이 없음.**
> - **그래서 n비트로는 -2^n과 2^n 이라는 수를 동시에 표현할 수 없다.**

Flag 사용에 대한 고찰

> 음수 데이터의 표시에 관하여 - 인프런 | 커뮤니티 질문&답변

> CPU 레지스터 자체에 음수와 양수를 구분하기 위한 플래그가 있음.
> **그러면 그냥 보수 개념이 아니라 Flag 만 사용해도 되는거 아니야?**

- 2의 보수 표현에서는 덧셈과 뺄셈 연산이 더욱 단순해지는데, 음수 플래그를 사용하면 덧셈과 뺄셈에 대한 별도의 로직이 필요하기 때문에 번거롭다.

16진수와 2진수의 변환

> - 16진수를 사용하는 이유?
>   - 2진수로만 모든 데이터를 표현하려면 문자 길이가 엄청 길어짐
>   - 2진수를 16진수로, 16진수를 2진수로 변환하는 과정이 쉽기 때문
> - **참고로, 하드웨어와 밀집하게 맞닿아있는 개발 분야에서는 코드에 16진수도 직접 쓰는 경우가 많으니 참고하자.**

16진수를 2진수로 변환하기

- **16비트 1글자 ➜ 4bit** (0-F)
- `0x1A2B` ➜ `0b0001 / 0b1010 / 0b0010 / 0b1011`

- `0001101000101011(2)` 가 `0x1A2B`를 2진수로 표현한 값이다.

2진수를 16진수로 변환하기

> - 2진수 숫자를 4개씩 끊는 것이 제일 중요하다!
> - **8421**을 꼭 외우도록 하자.

- `0b11010101` ➜ `0b1101 / 0b0101`
- `0b1101 -> D`
- `0b0101 -> 5`
- 이 둘을 이어붙이면 `0xD5` 가 되는 것이다.

챕터3) 명령어

고급어와 저급어, 그리고 컴파일과 인터프리트

고급어

> - 사람이 쉽게 이해하기 만들어진 언어
> - 우리가 알고있는 대부분의 프로그래밍 언어

저급어

> - 크게 `기계어`와 `어셈블리어`가 있다.
> - 우리가 기계어를 보고 어떻게 동작하는지 파악할 수 있을까? 절대못함
> - **CPU(Machine)가 쉽게 읽기 위한 언어라고 생각하자.**

고급언어가 어떻게 저급언어로 변환될까?

> - 크게 **컴파일** 방식과 **인터프리트** 방식이 있음.

컴파일 방식

> - **컴파일**: 컴파일 언어(ex. C언어)로 작성된 **소스코드 전체가 저급 언어로 변환**되는 과정
> - 그 컴파일 과정을 도와주는 도구가 **컴파일러**
> - **개발자가 작성한 소스코드 전체를 훑으면서 문법적인 오류는 없는지, 실행 가능한 코드인지 등을 따지고 저급언어로 컴파일한다.**
> - 변환된 코드를 `목적코드` 라고 함.

인터프리트 방식

> - **인터프리트**: 인터프리트 언어(ex. Python)로 작성된 소스코드가 1줄씩 저급언어로 변환해주는 과정
> - **컴퓨터와 대화하듯 소스코드를 한줄씩 실행하기 때문에 소스코드 전체를 저급언어로 변환하는 시간을 기다릴 필요가 없다.**
> - **컴파일 방식과는 다르게 소스코드 N번째 줄에 문법 오류가 있어도 N-1번째 줄까지는 수행이 된다.**

주소 지정 방식

> **현대 CPU는 다양한 주소 지정 방식을 사용하고 있음.**

- **즉시 주소 지정 방식**: 연산에 사용할 데이터
- **직접 주소 지정 방식**: 유효 주소 (메모리 주소)
- **간접 주소 지정 방식**: 유효 주소의 주소
- **레지스터 주소 지정 방식**: 유효 주소 (레지스터 이름)
- **레지스터 간접 주소 지정 방식**: 유효 주소를 저장한 레지스터

문제)

컴퓨터 구조

컴퓨터의 4가지 메인 구성 요소를 말해주세요.

- CPU
- 보조기억장치 (HDD, SDD, USB 등등)
- 메모리 (주기억장치)
- 입출력장치

보수 연산 문제

아래 2개의 항등식을 1의 보수를 사용해서 표현해보자.

- **19 - 3 = 16**
  - 19: **10011**
  - 3 : 00011 → 1의 보수로 변환 → **11111100**
  - **00010011 + 11111100 → 100001111**
  - Carry가 발생했기 때문에 Carry를 최하위로 내려서 더해준다.
    - 00001111 + 1 → 10000 → **16**
- **10 - 13 = -3** (10 + (-13) = -3)
  - 10: 1010

    - 13: 1101
    - **13의 1의 보수는 0010**
    - **1010 + 0010 = 1100** (12)
        - 캐리가 발생하지 않았음.
        - 그러면 해당 결과값에서 다시 1의 보수를 구하고 음수기호를 붙여주면 됨
    - **1100 → 0011 → -0011 = -3**

아래 항등식을 2의 보수를 사용해서 표현해보자.

- **10 - 13 = -3**
    - 10: 1010
    - 13: 1101
    - 13의 2의 보수: 0010 + 1 → 0011
    - 1010 + 0011 → **1101**
    - 1101 → 0010 + 1 → -0011 → -3

0xF1AB를 2진수로 표현하면 어떻게 될까요?

- 0b/1111/0001/1010/1011

0b11111111을 16진수로 표현하면 어떻게 될까요?

- 0b1111 / 0b1111 → 0xFF

컴파일과 인터프리트 문제

- **`Java`는 컴파일 언어? 인터프리트 언어?**
    - 둘다 맞음
- **`JavaScript`는 컴파일 언어? 인터프리트 언어?**
    - 둘다 맞음!
    - **그렇지만 순수한 JS언어만 보면 인터프리트 언어인 것 같음**
- **컴파일 언어와 인터프리터 언어 중에 어떤 언어가 속도가 느릴까?**
    - 인터프리터 언어

세룡 질문

(1) 현재 가장 많이 사용되는 인코딩 방식은?

- UTF-8
- 그게 정확히 무엇?

- **효율적인 인코딩 방식이라고 했는데, 어떤 부분이 효율적인 건지?**

(2) null과 undefined의 차이점?

- `null` : 의도적으로 값이 없음을 표현하기 위함
- `undefined` : 변수가 선언되었지만 값이 할당되지 않았을 때 시스템에서 자동으로 설정되는 값