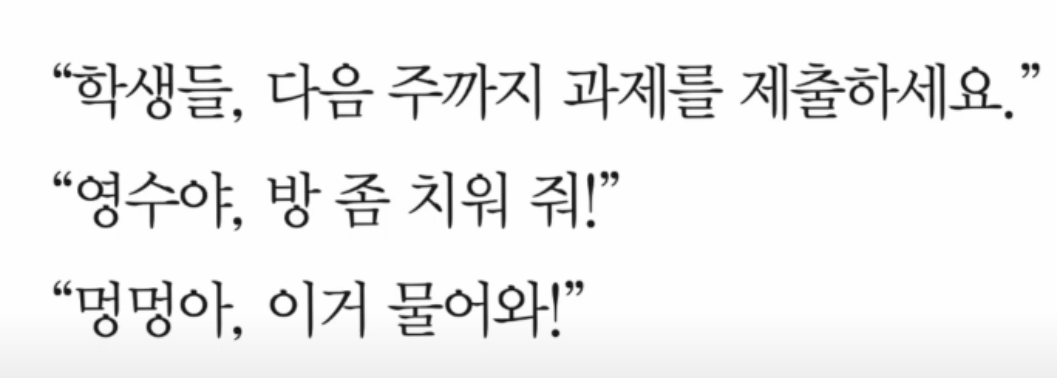
7강. 명령어의 구조와 주소 지정 방식

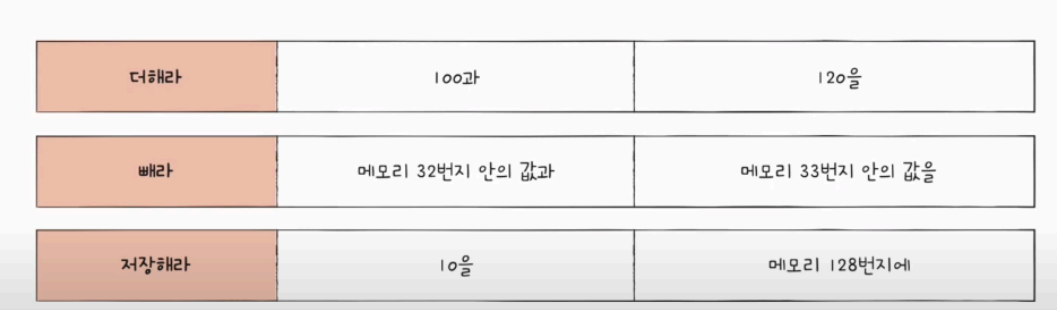
명령어의 구조

명령어의 구조는 일상 언어로는 다음과 같이 구성됩니다.

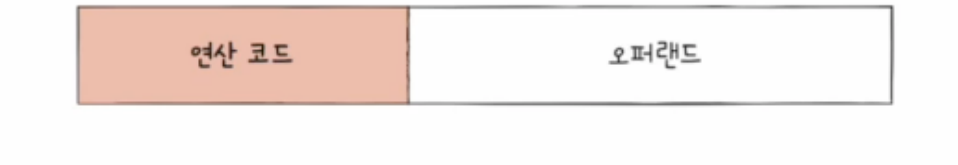


- 무엇을 대상으로, 무엇을 수행하라!

조금 더 컴퓨터가 이해할 수 있는 명령어의 구조는 다음과 같습니다.

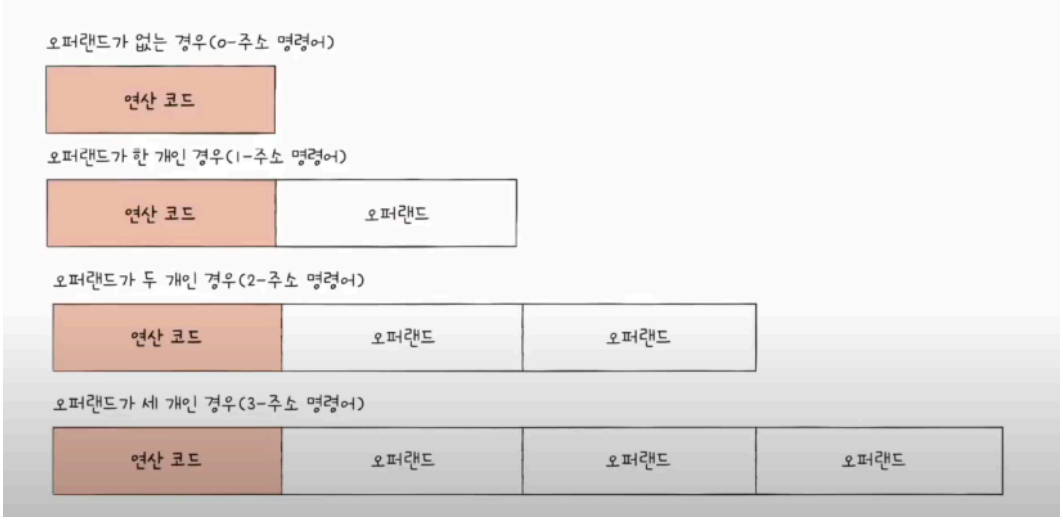


이를 더욱 구조화 시키면 다음과 같이 나타낼 수 있습니다.



오퍼랜드(operand)는 연산에 사용될 데이터 또는 연산에 사용될 데이터가 저장된 위치가 될 수 있습니다.

오퍼랜드



연산코드

연산 코드는 다음과 같이 구성되어 있습니다.

- 데이터 전송
- 산술 / 논리 연산
- 제어 흐름 변경
- 입출력 제어

대표적인 연산 코드의 종류

- 스택 (Stack)
- 큐 (Queue)

산술/논리 연산

- 사칙연산 (ADD, SUBTRACT, MULTIPLY, DIVIDE)
- 논리 연산자 (AND, OR ...)
- 비교 (Compare)

제어 흐름 변경

- JUMP : 특정 주소로 실행 순서를 옮겨라
- CONDITION JUMP : 조건에 부합할 때 특정 주소로 실행 순서를 옮겨라
- HALT : 프로그램의 실행을 멈춰라

- CALL : 되돌아올 주소를 저장한 채 특정 주소로 실행 순서를 옮겨라
- RETURN : CALL을 호출할 때 저장했던 주소로 돌아가라

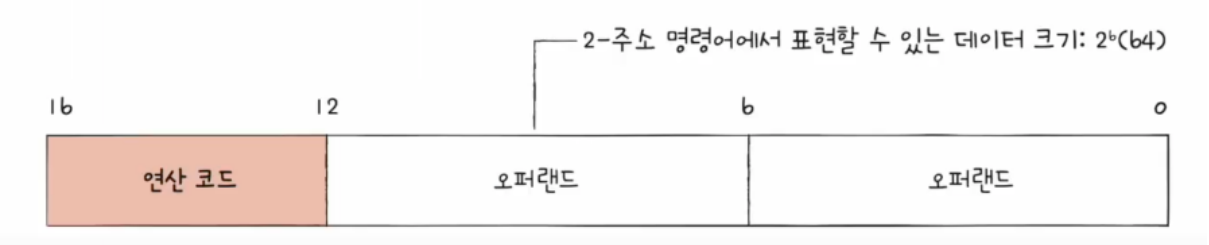
입출력 제어

- READ(INPUT) : 특정 입출력 장치로부터 데이터를 읽어라
- WRITE(OUTPUT) : 특정 입출력 장치로 데이터를 써라
- START IO : 입출력 장치를 시작하라
- TEST IO : 입출력 장치의 상태를 확인하라

명령어 주소 지정 방식

앞서 오퍼랜드는 연산에 사용될 데이터 그 자체가 될 수도 있고, 데이터가 저장된 위치가 될 수도 있습니다.

현대 컴퓨터 구조에서는 일반적으로 데이터 자체보다는 위치(주소)를 더 많이 사용하는데, 그 이유는 다음 세가지와 같습니다.



1. 명령어 비트 수의 제약

예를 들어, 하나의 명령어가 16비트로 구성되어 있다고 합시다.

이 안에서 연산 코드와 여러 개의 오퍼랜드를 표현해야 하므로, 각 오퍼랜드에 할당할 수 있는 비트 수는 제한적입니다.

위 그림처럼 오퍼랜드가 2개라면, 각 오퍼랜드가 6비트씩 배정될 수 있습니다. 그렇다면 표현할 수 있는 값의 범위는 $2^6 = 64$개 뿐입니다.

2. 데이터 크기의 다양성

실제로 연산에 쓰이는 데이터는 매우 크거나 다양한 범위를 가질 수 있습니다. 하지만 명령어의 오퍼랜드 필드 크기는 제한적이기 때문에, 모든 데이터를 직접 표현하는 것은 불가능합니다.

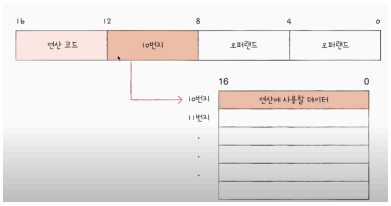
3. 위치(주소)를 사용 → 확장성

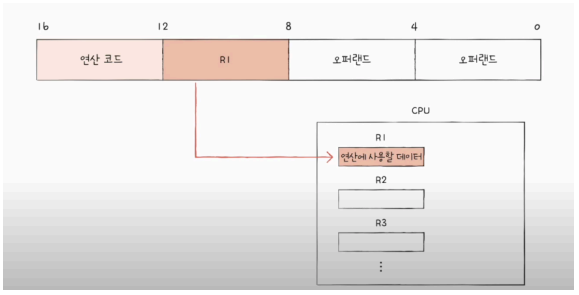
따라서 데이터 자체를 넣는 대신, 데이터가 저장된 위치를 넣습니다.

이렇게 하면, 명령어 크기와 상관없이 메모리 어딘가에 존재하는 훨씬 큰 범위의 데이터를 자유롭게 참조할 수 있습니다.

즉, 작은 비트 수로도 큰 공간의 데이터를 다룰 수 있는 유연성을 얻게됩니다.

이를 그림으로 표현하면 다음과 같습니다.





연산에 사용할 데이터가 저장된 위치를 찾는 방법

유효주소 (effective address)는 연산에 사용할 데이터가 저장된 위치입니다.

명령어 주소를 지정하는 방식에는 다음과 같습니다.

- 연산에 사용할 데이터가 저장된 위치를 찾는 방법
- 유효 주소를 찾는 방법
- 다양한 명령어 주소 지정 방식들

즉시 주소 지정 방식 (immediate addressing mode)

- 연산에 사용할 데이터를 오퍼랜드 필드에 직접 명시합니다.
- 가장 간단한 형태의 주소 지정 방식입니다.
- 연산에 사용할 데이터의 크기가 작아질 수 있지만, 빠르다는 장점을 갖고있습니다.

직접 주소 지정 방식 (direct addressing mode)

- 오퍼랜드 필드에 유효 주소를 직접적으로 명시하는 방법입니다.
- 유효 주소를 표현할 수 있는 크기가 연산 코드만큼 줄어듭니다.

간접 주소 지정 방식 (indirect addressing mode)

- 오퍼랜드 필드에 유효 주소의 주소를 명시하는 방법입니다.

- 앞선 주소 지정 방식들에 비해 속도가 느립니다.

레지스터 주소 지정 방식 (register addressing mode)

- 연산에 사용할 데이터가 저장된 레지스터를 명시하는 방법입니다.
- 메모리에 접근하는 속도보다 레지스터에 접근하는 것이 더 빠릅니다.

레지스터 간접 주소 지정 방식 (register indirect addressing mode)

- 연산에 사용할 데이터를 메모리에 저장하는 방식입니다.
- 그 주소를 저장한 레지스터를 오퍼랜드 필드에 명시합니다.