**Chapter 3 명령어**

***3-1*** **소스 코드와 명령어**

**<고급 언어와 저급 언어>**

**고급언어**: 사람이 이해하고 작성하기 쉽게 만들어진 언어

**저급언어**: 컴퓨터가 직접 이해하고 실행할 수 있는 언어

* 고급 언어로 작성된 소스 코드가 실행되려면 반드시 저급 언어, 즉 명령어로 변환되어야 함
* 저급 언어에는 **기계어**와 **어셈블리어**가 존재

**기계어**: 0과 1의 명령어 비트로 이루어진 언어

**어셈블리어**: 기계어를 읽기 편한 형태로 번역한 언어

* 하드웨어와 밀접하게 맞닿아 있는 프로그램을 개발하는 임베디드 개발자, 게임 개발자, 정보 보안 분야등의 개발자는 어셈블리어를 많이 사용

**<컴파일 언어와 인터프리터 언어>**

고급 언어를 저급 언어로 변환하는 방식은 **컴파일 방식**과 **인터프리터 방식** 두 가지

**컴파일 언어**: 컴파일러에 의해 소스 코드 전체가 저급 언어로 변환되어 실행되는 고급 언어

* 대표적인 컴파일 언어는 C
* 컴파일을 수행해 주는 도구를 **컴파일러**라고 함
* 소스 코드를 처음부터 끝까지 저급 언어로 컴파일 -> 소스 코드 내에서 오류를 하나라도 발견하면 해당 소스 코드는 컴파일 실패
* 컴파일러를 통해 저급 언어로 변환된 코드를 **목적 코드**라고 함

**인터프리터 언어**: 인터프리터에 의해 소스 코드가 한 줄씩 실행되는 고급 언어

* 대표적인 인터프리터 언어는 Python
* 소스 코드를 한 줄씩 한 줄씩 차례로 실행
* 소스 코드를 한 줄씩 저급 언어로 변환하여 실행해 주는 도구를 **인터프리터**라고 함
* 소스 코드 전체를 저급 언어로 변환하는 시간을 기다릴 필요가 없음
* 소스 코드 N번째 줄에 문법 오류가 있더라도 N-1번째 줄까지는 올바르게 수행
* 컴파일 언어보다 느림

-> 외국어 책을 번역할 때의 예시

***3-2*** **명령어의 구조**

**<연산 코드와 오퍼랜드>**

**명령어**: **연산 코드**와 **오퍼랜드**로 구성

**연산 코드(연산자)**: 명령어가 수행할 연산

* 연산 코드가 담기는 영역은 연산 코드 필드
* 기본적인 연산 코드 유형은 네 가지

1. 데이터 전송
2. 산술/논리 연산
3. 제어 흐름 변경
4. 입출력 제어

**오퍼랜드(피연산자)**: 연산에 사용할 데이터 또는 연산에 사용할 데이터가 저장된 위치

* 오퍼랜드가 담기는 영역은 오퍼랜드 필드
* 오퍼랜드 필드에는 데이터를 직접 명시하기 보다는 데이터가 저장된 위치가 주로 담김**(주소필드)**
* 오퍼랜드는 명령어 안에 0개 또는 1개 이상의 여러개가 담길 수 있음
* **0-주소 명령어**: 오퍼랜드가 하나도 없는 명령어

**1-주소 명령어**: 오퍼랜드가 하나인 명령어

**2-주소 명령어**: 오퍼랜드가 두 개인 명령어

**3-주소 명령어**: 오퍼랜드가 세 개인 명령어

**<주소 지정 방식>**

오퍼랜드 필드에 주로 데이터를 그대로 담지 않고 데이터의 주소를 이용하는 이유

-> 명령어 길이 때문. 주소를 담는다면 표현할 수 있는 데이터의 크기는 하나의 주소 공간만큼 커짐

**유효 주소**: 연산의 대상이 되는 데이터가 저장된 위치

**주소 지정 방식**: 연산에 사용할 데이터 위치(유효 주소)를 찾는 방법

1. **즉시 주소 지정 방식**

* 데이터를 오퍼랜드 필드에 직접 명시
* 표현할 수 있는 데이터의 크기가 작아지는 단점
* 데이터를 메모리나 레지스터로부터 찾는 과정이 없기에 속도가 빠름

1. **직접 주소 지정 방식**

* 오퍼랜드 필드에 유효주소를 직접적으로 명시
* 표현할 수 있는 데이터의 크기는 즉시 주소 지정 방식보다 커졌지만 여전히 유효 주소를 표현할 수 있는 범위가 연산 코드의 비트 수만큼 줄어듬

1. **간접 주소 지정 방식**

* 유효 주소의 주소를 오퍼랜드 필드에 명시
* 직접 주소 지정 방식보다 표현할 수 있는 유효 주소의 범위가 더 넒어짐
* 두 번의 메모리 접근이 필요하기 때문에 위의 방식들보다 느림

1. **레지스터 주소 지정 방식**

* 오퍼랜드 필드에 연산에 사용할 데이터가 저장된 레지스터를 명시
* 메모리 접근보다 레지스터 접근이 빠르기때문에 직접 주소 지정 방식보다 빠름
* 단점은 레지스터 크기에 제한이 생길 수 있음

1. **레지스터 간접 주소 지정 방식**

* 데이터를 메모리에 저장하고 그 주소를 저장한 레지스터를 오퍼랜드 필드에 명시
* 간접 주소 지정 방식과 다르게 메모리에 접근하는 횟수가 한 번 -> 속도 더 빠름

**<스택과 큐>**

**스택**

* 한쪽 끝이 막혀 있는 통과 같은 저장 공간
* 후입 선출(LIFO)
* 새로운 데이터를 저장하는 명령어는 **PUSH**, 저장된 데이터를 꺼내는 명령어는 **POP**

**큐**

* 양쪽이 뚫려 있는 통과 같은 저장 공간
* 선입선출(FIFO)
* 새로운 데이터를 저장하는 명령어는 **PUSH**, 저장된 데이터를 꺼내는 명령어는 **POP**