석가탄신일... 이지만

그래도 우리는 공부한다

스터디 자료

2020.04.30. 목

B반 송명훈

목 차

1. 컴퓨터의 구조

1. 컴퓨터의 구조 (Computer Architecture)

1. 개념 및 정의

1. 컴퓨터 구조는 컴퓨터 구조공학이라는 컴퓨터공학 전공과목이 존재할 정도로 양이 방대하고 깊음
2. 컴퓨터 구조공학이란? (잡글)
    1. 컴퓨터구조(computer architecture)는 컴퓨터를 구성하는 CPU, 메모리, 입출력장치(I/O 장치)의 기본이 되는 디지털 회로소자의 특성과, CPU의 동작원리와 설계방법을 익히며, CPU의 제어회로 및 마이크로프로그램 등을 연구함으로써 컴퓨터구조에 대한 원리와 특성을 익히는 학문이다. RISC(Reduced Instruction Set Computer) 프로세서 중 하나인 MIPS(Microprocessor without Inter-locked Pipeline Stages)를 예제로 하여 실제 컴퓨터의 데이터 패스와 컨트롤 로직을 설계하고 성능과 관련된 요인들을 공부한다. 구체적으로는 컴퓨터 성능 구분 요소, 이진 데이터 연산, 데이터 패스 설계, 컨트롤 로직 설계, 파이프라인 기법, 메모리 시스템 설계, 그리고 주변장치 및 멀티프로세서 시스템에 대해 연구한다.

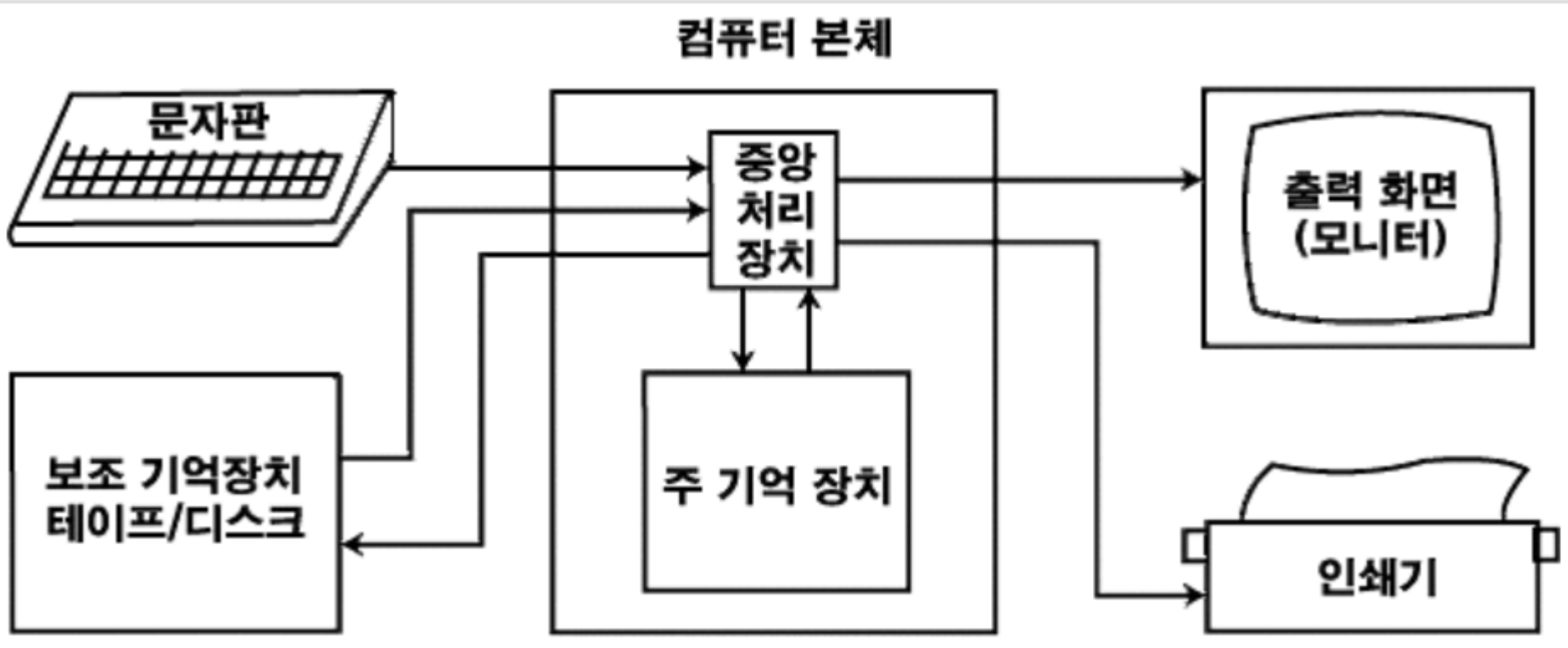
    2. 또한 MIPS CPU의 어셈블리 언어(assemble language)를 이해함으로써 CPU의 구조 및 동작원리와 파이프라인(pipeline)의 동작원리를 이해하고 기억장치 계층(memory hierarchy)의 구조를 이해함으로써 가상기억장치(virtual memory)의 동작원리를 이해하는 학문 분과이다. 컴퓨터구조의 기본 개념 및 동작원리를 이해하고 그 응용분야를 학습하고 MIPS CPU의 어셈블리 언어(assemble language) 및 기계어 명령(machine instruction)을 학습함으로써 CPU의 내부 동작원리를 이해한다. 그리고 단일 사이클 중앙처리장치(single cycle CPU)에서 데이터 경로(data path) 및 제어장치(control unit)의 동작원리를 이해한다. 또한 파이프라인(pipeline)의 동작원리를 이해함으로써 CPU 파이프라인(pipeline)을 모델링 한다.

    3. 이 과목은 프로세서와 메모리로 구성되는 폰 노이만 구조를 이해한다. 또한 레지스터, 산술 논리 연산 장치(ALU: Arithmetic and Logic Unit), 제어 유닛 등 프로세서 내부 구조에 대해 알아보며, 메인 메모리 및 보조기억장치의 원리에 대해 공부한다. 각종 입출력 장치의 구조 및 인터럽트, 직접 메모리 접근(DMA: Direct Memory Access) 등 입출력 방식을 알아보며, 운영체제, 컴퓨터시스템 설계, 마이크로프로세서 등 컴퓨터공학(-工學, computer technology) 전공 하드웨어 교육의 근간이 된다.

1. 컴퓨터의 구조 (Computer Architecture)

I. 컴퓨터 구조의 개략도

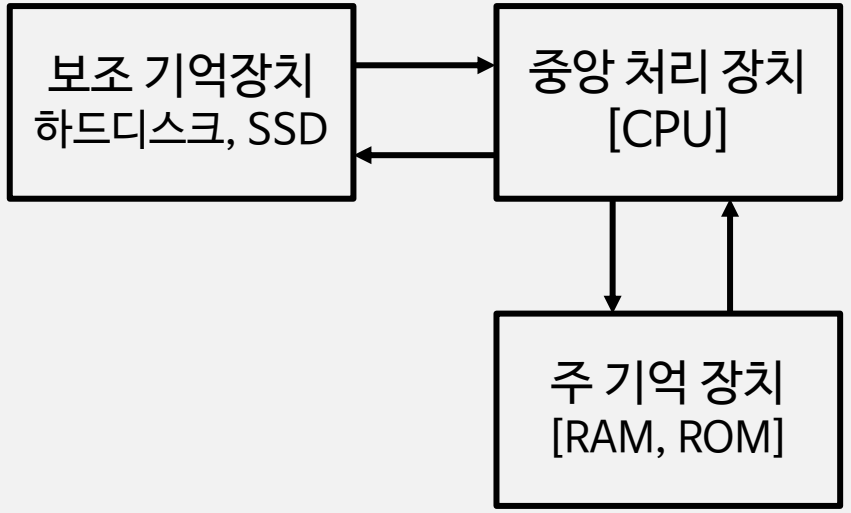
1. 컴퓨터 구조는 컴퓨터 구조공학이라는 컴퓨터공학 전공과목이 존재할 정도로 양이 방대하고 깊음



1. 컴퓨터의 구조 (Computer Architecture)

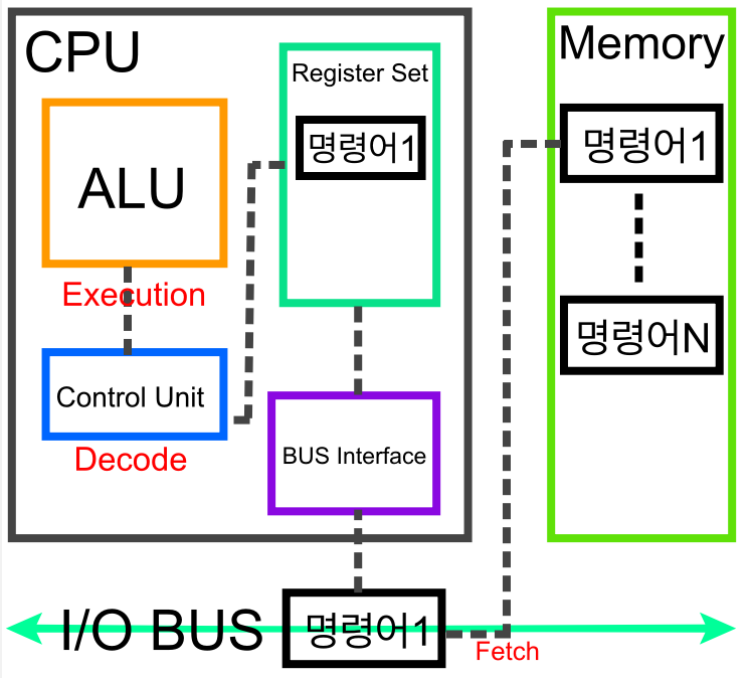
Ⅱ. 우리가 알아야할 부분들

1. 중앙처리장치(CPU, Central Processing Unit)
   1) 컴퓨터에서 구성 단위 중 기억, 해석, 연산, 제어라는 4대 기능을 종합하는 장치인 Central Processing Unit(중앙 처리 장치)의 줄임말.
   2) 컴퓨터의 정 중앙에서 모든 데이터를 처리하는 장치, 컴퓨터의 두뇌에 해당
   3) 사용자로부터 입력 받은 명령어를 해석, 연산 후 결과 출력하는 역할까지 담당.
   4) CPU 처럼 하나의 부품에 연산, 해독, 제어 장치 등이 집적되어 있는 형태를 마이크로프로세서(Micro-processor)라고 한다. CPU와 Micro-process는 거의 같은 의미로 사용됨



1. 컴퓨터의 구조 (Computer Architecture)

1. CPU의 구성



1) 산술 논리 연산장치 (Arithmetic Logic Unit, ALU)
   - CPU에서 연산을 수행하는 장치. 덧셈 뺄셈과 같은 산술 연산과 AND, OR 등의 논리 연산의 2가지 방식으로만 연산함

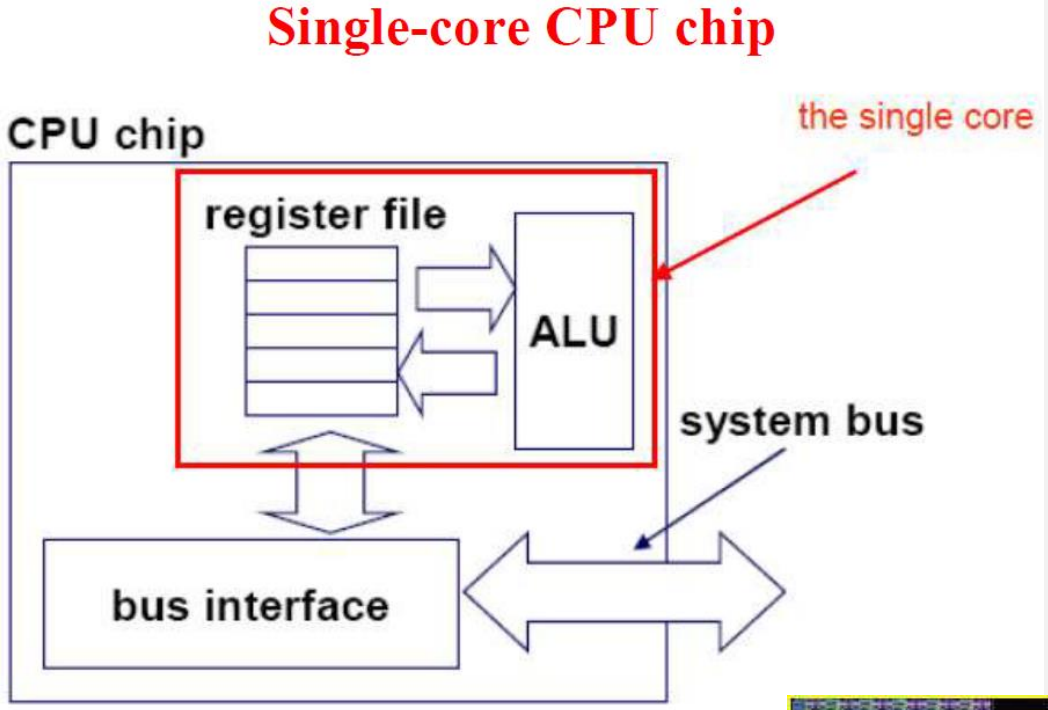
2) 레지스터 (Register Set)
   - CPU에서 연산을 하기 위해서 데이터를 임시 저장하는 공간
   - 연산 후에 나오는 임시 데이터를 저장하거나, CPU의 상태, 다음 명령의 메모리 번지수 등이 이 레지스터에 기억됨. 다양한 종류가 존재.

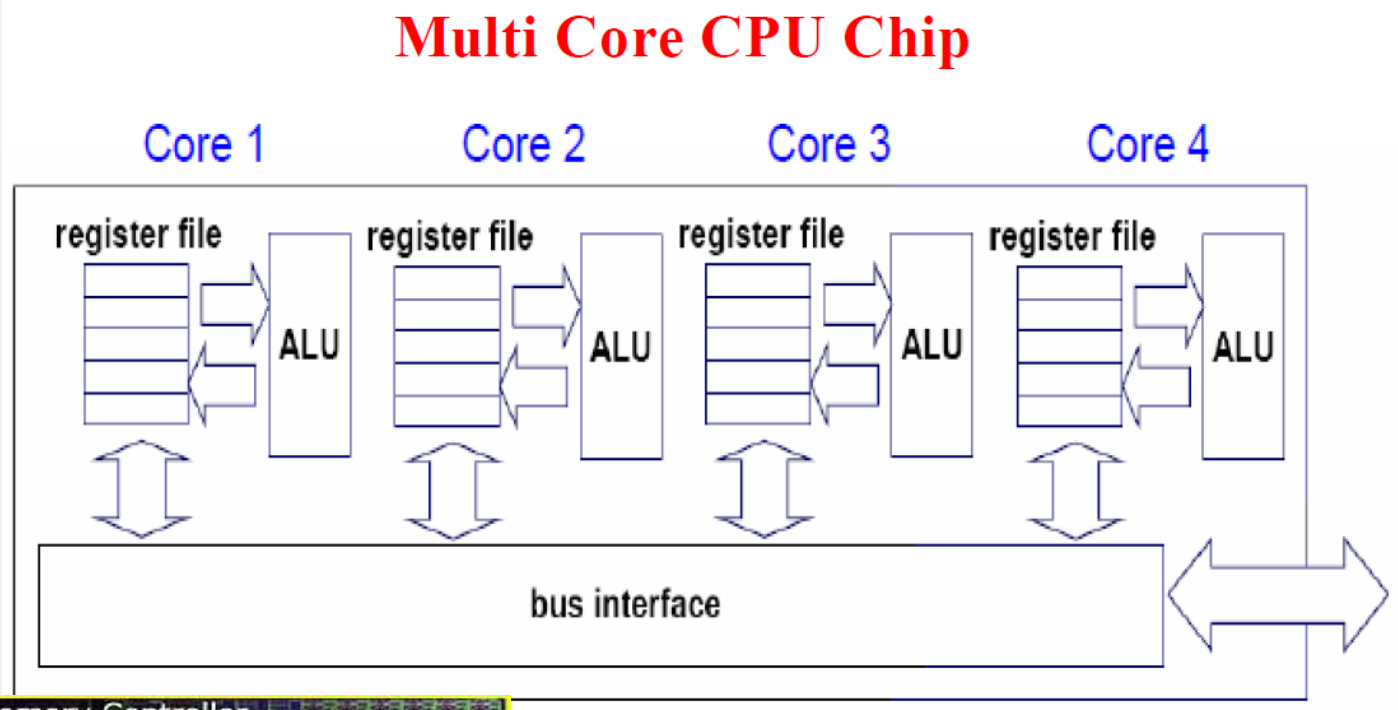
3) 제어부 (Control Unit)
   - 컨트롤 유닛은 CPU를 제어하는 역할 담당. 컨트롤 유닛이 바이너리 코드를 해석(Decode)하고 이를 바탕으로 레지스터와 ALU 사이의 명령 흐름을 전기적으로 제어
   - ALU는 연산만 담당할 뿐, ALU로 들어오는 명령어를 해석하지는 못하며 컨트롤 유닛이 명령어를 해석하고 연산을 수행하게끔 전기적 신호로 ALU에 명령을 내린다. 이후 결과에 따라 다른 영역으로 제어 신호 보냄

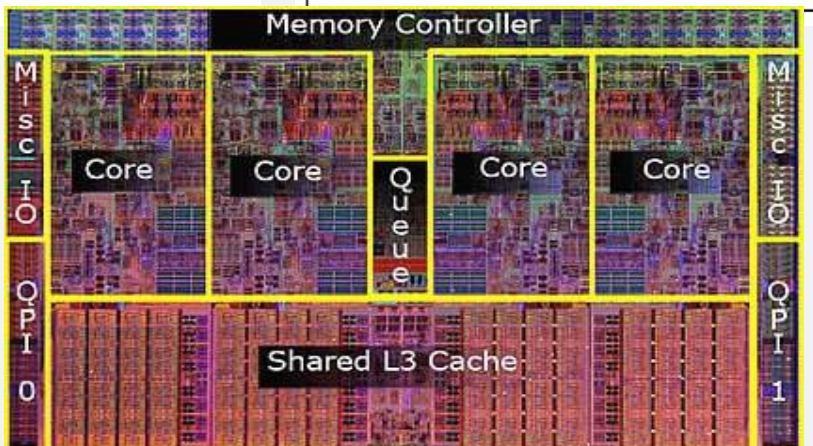
4) 버스 인터페이스 (Bus Interface)
   - 데이터가 이동하는 통로(회로)를 버스(Bus)라고 한다. CPU 외부와 내부사이에서 버스들과 데이터를 송수신하는데 이 때 CPU 외부와 데이터를 주고 받는 통로가 버스 인터페이스

1. 컴퓨터의 구조 (Computer Architecture)

2. 멀티 코어 프로세서 (multi-core processor)







1. 컴퓨터의 구조 (Computer Architecture)

2. 코어의 성능을 표시하는 Clock(클럭)

공식 스피드

실제 측정(동작) 스피드
(=CPU Measured Speed)

1.8 GHz란?

Giga = 10^9 (10억)

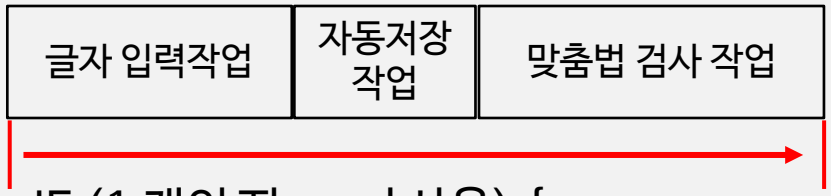
Hz(헤르쯔): 1초 당 진동수, 1 Hz = $\frac{1\text{회}}{1\text{초}}$

왜 여기서 진동수가?

- 1초 당 18억 Clock 을 처리한다는 뜻
- 여기서 64비트 프로세서는 1개 코어가 1 클럭에 처리할 수 있는 정보의 크기
- 64 bit = 8 byte
  = 0부터 18,446,744,073,709,551,615까지의 정수
- 쿼드(4) 코어는 1초당 18억 X 4 X 8 바이트의 정보를 처리할 수 있다!

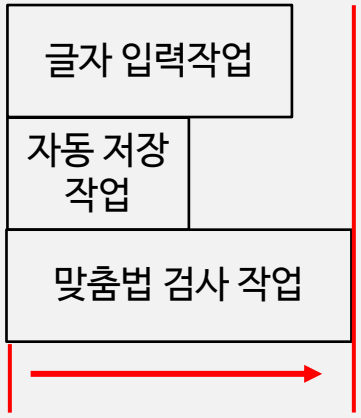
1. 연습문제 6-1 에서 조사한 쓰레드란?

1 개의 프로세스 (Ex] MS Word)



IF (1 개의 Thread 사용) {
    1개의 Thread가 순차적 작업 진행
}

1 개의 프로세스 (Ex] MS Word)



IF (3 개의 Thread를 사용) {
    각 Thread 개별적 작업 진행
}
⇨ Multithread 작업방식

일반적으로 스레드 (Thread) 란?

사전적 의미

- "프로세스 내에서 실행되는 여러 흐름의 단위"
- 프로세스의 특정 수행 경로
- 프로세스가 할당 받은 자원을 이용하는 실행의 단위

그럼 프로세스(Process) 란?

사전적 의미

- "컴퓨터에서 연속적으로 실행되고 있는 프로그램"
  Ex] MS Word. (다수의 프로세스로 실행되는 프로그램도 존재)
- 운영체제로부터 시스템 자원(CPU 시간, 운영에 필요한 주소공간, 독립된 메모리 영역)을 할당 받는 작업의 단위
- 메모리 영역은 Code, Data, Stack, Heap 구조
  (이에 대한 추가적인 설명은 생략)
- 즉, 동작하고 있는 프로그램을 의미
- 기본적으로 프로세스당 최소 1개의 스레드 (메인 스레드)를 가짐

⇨∴ 따라서 Thread는 Process의 구성 단위.

1. 연습문제 6-1에서 조사한 쓰레드란?

자바에서 스레드 란?

- 일반적인 thread와 큰 차이가 없음
- JVM가 운영체제 역할 수행
- java에는 process 존재하지 않고 thread만 존재
- java thread는 JVM에 의해 예정된 실행 단위 코드 블록
- 대부분 JVM이 관리 및 조정
- 개발자는 java thread로 작동할 thread 코드를 작성하고 실행 시작을 JVM에 요청

java.lang.Thread class

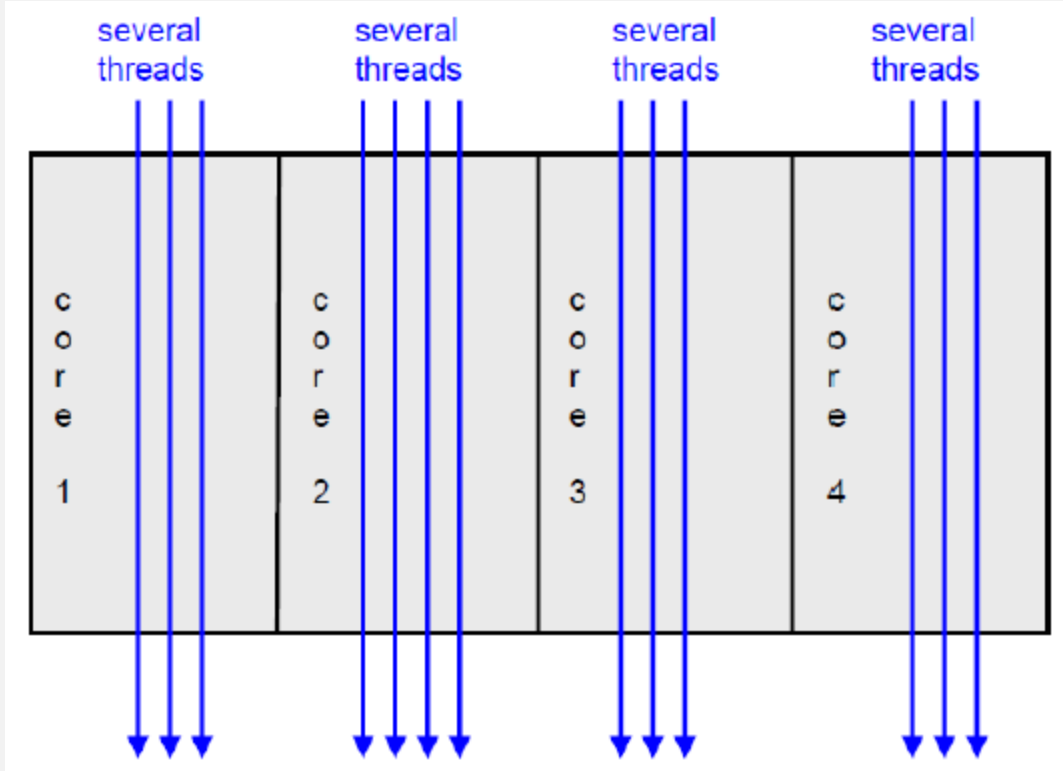
- Thread class는 thread 사용을 조작하기 위한 코드를 작성할 때 사용하는 class
- 수업 중 사용한 String 자료형은 실제로는 String Class 생성자로 생성된 String 인스턴스
  - 개발자는 java.lang.String class로 String 객체를 생성하고 String class 내부 메소드를 호출하여 기능 사용 가능. (Ex] strName.split(" ") 등)
- String 처럼 Thread class를 이용해 Thread 인스턴스와 메소드를 생성 및 사용 가능.
  이를 통해 java에서 thread의 작동 방식을 조절할 수 있는 코드를 작성할 수 있다.

Thread.sleep(long millis)

- Java.lang.Thread class의 메소드 중 하나
- argument로 입력 받은 millis 값에 해당하는 밀리 초(millisecond, ms)만큼 현재 Thread의 작업을 중지시킴
- 3 초를 정지시키기 위해서는 Thread.sleep 메소드의 argument에 3000을 입력해주면 된다.

1. 컴퓨터의 구조 (Computer Architecture)

1. 멀티 쓰레드 (Multi-thread)
   1) 코어와 쓰레드
      A. 쓰레드는 코어와 마찬가지로, CPU를 여러 개로 분리해주는 것
      B. 코어는 CPU를 물리적으로 구별한 것이고, 쓰레드는 CPU를 논리적으로 구별한 것
         ① 코어는 사람, 쓰레드는 팔이라고 생각하면 됨 (단순히 코어와 쓰레드 관계에서만 해당)
         ② 예전에는 코어 하나 당 쓰레드가 하나였으나 (1코어 1쓰레드), 인텔이 하이퍼쓰레딩(Hyper Threading)라는, 1개 코어를 반으로 쪼개서 사용하는 기술을 개발하여 현재 하이퍼 쓰레딩이 적용되는 CPU의 경우 1코어 2쓰레드 또는 그 이상의 구조를 갖는다.
         ③ 최신 CPU라 해도 저사양 CPU는 1코어 1쓰레드가 적용된다. 기본적으로 1코어 1쓰레드.
      C. 멀티 코어 프로세서의 경우 코어가 4개 이므로 최소 4개의 쓰레드를 갖게 됨. 이를 멀티 쓰레드라 한다.
      D. 현재 가장 많은 코어와 많은 쓰레드를 가진 CPU는 96코어 384 쓰레드인 ARM의 서버용 CPU



1. 컴퓨터의 구조 (Computer Architecture)

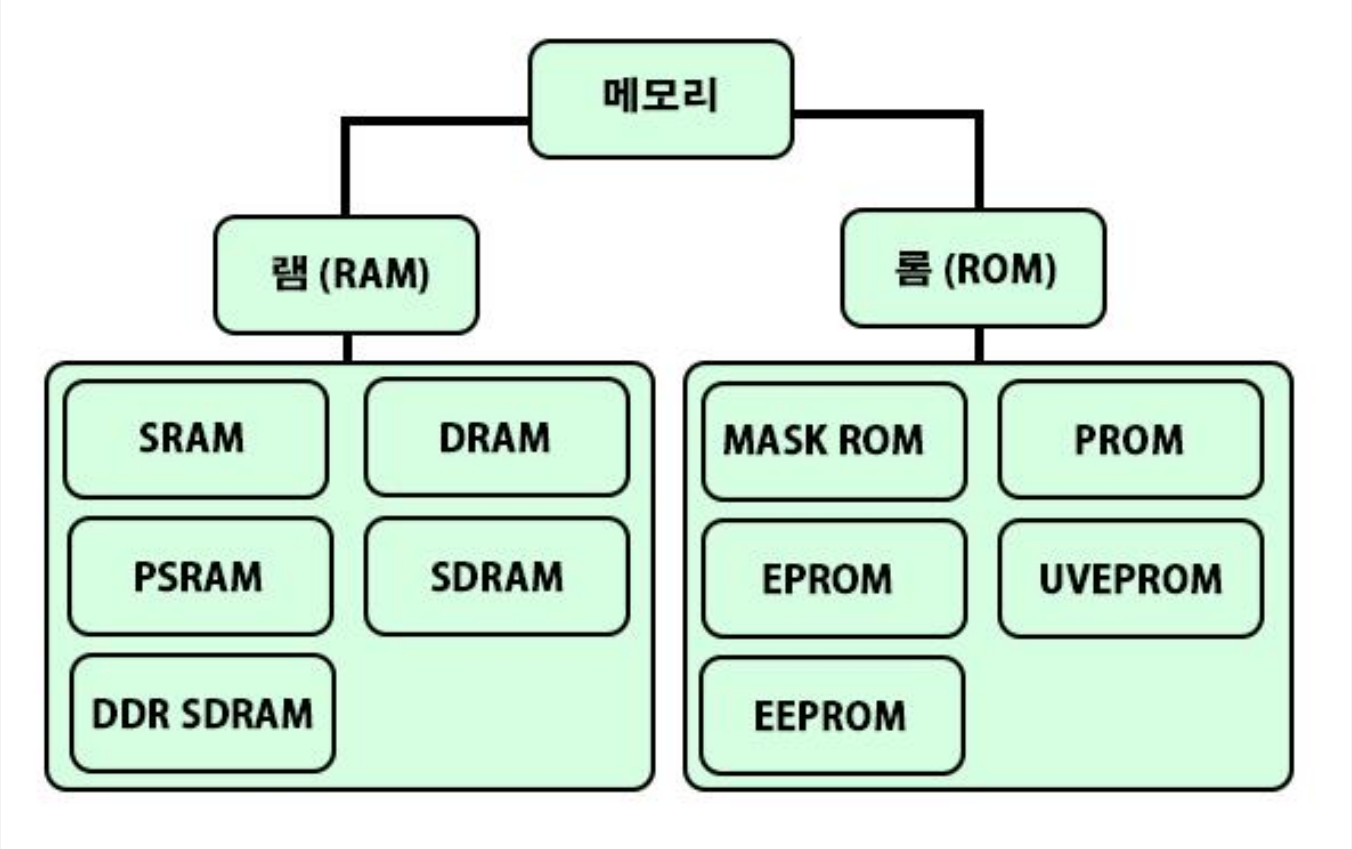
2. 기억 장치 = 메모리 (Memory)
   1) 저장 방식 : 반도체 (Semi-conductor)
      A. RAM, ROM, DRAM 등 : 주기억 장치에 사용가능
         컴퓨터 시스템 내에는 분명 ROM이 탑재되어 있으며, ROM 역시 주기억장치의 일부이다.
         ① 램디스크 (보조기억장치로 사용가능)
         ② 옵테인 메모리 (보조기억장치로 사용가능)
      B. 플래시 메모리 :보조 기억 장치에 사용 가능
         A. USB 메모리, SD카드(MMC)
         B. eMMC (embeded Multi Media Card)
         C. SSD(Solid State Disk)
         D. UFS(삼성 갤럭시 등 안드로이드계열), NVMe(아이폰6s ~) 등
   2) 자기장(Magnetic)을 이용한 방식
      A. 플로피 디스크, 하드 디스크, 자기 테이프(비디오 테이프)
   3) 광학(Optical) 방식
      A. CD(예전엔 CD-ROM이라고 불렀음), DVD(HD-DVD), Blu-ray(or Ultra HD Blu-ray) 등

1. 컴퓨터의 구조 (Computer Architecture)

1. 주 기억 장치
   A. Ramdom Access Memory (RAM)
      ① 전원이 끊어지면 휘발유처럼 기록된 정보도 날아 감 → 휘발성 메모리(Volatile Memory)라고도 함
      ② 자기를 이용한 기존의 자기 코어와 달리 TR(트랜지스터)를 직접한 IC를 이용해 기억소자로 사용하고, 읽을 때 순차적이 아닌 랜덤하게 읽을 수 있기 때문에 읽기와 쓰기 속도가 매우 빠르다. 이러한 특성 때문에 컴퓨터의 주기억장치, 응용 프로그램의 로딩, 데이터의 일시적 저장 등에 사용
      ③ 기록과 해독의 두 회로가 있어 정보의 기록, 해독이 가능
   B. Read-Only Memory (ROM)
      ① ROM에 데이터를 (반영구적으로) 저장한 후 지속적으로 사용. 컴퓨터를 구동하기 위한 기본적인 정보가 담겨 있음 (컴퓨터의 BIOS, 운영체제 구동 정보 등)
      ② 일반적인 ROM은 데이터를 한 번 저장하면 지울 수 없어 계속 사용 해야 하지만, PROM(1번 다시 쓰기 가능), EPROM(무한), EEPROM(무한)은 특수한 방법을 통해 데이터를 삭제한 후 다시 기록 가능
      ③ 가장 기본적인 ROM은 MASK ROM으로써 제조사가 ROM 제조시에 데이터를 미리 저장해주며 데이터를 변경할 수 없다. 용량이 램에 비해 적음
      ④ 가장 최근 개발된 ROM은 낸드 플래시 메모리이며 보조 기억 장치인 SSD, USB가 플래시 메모리로 구성 되어 있다.

1. 컴퓨터의 구조 (Computer Architecture)

램 (RAM)	롬 (ROM)
읽기 쓰기 가능	읽기만 가능 (특수 케이스 제외)
빠름	비교적 느림
휘발성 메모리	비 휘발성 메모리

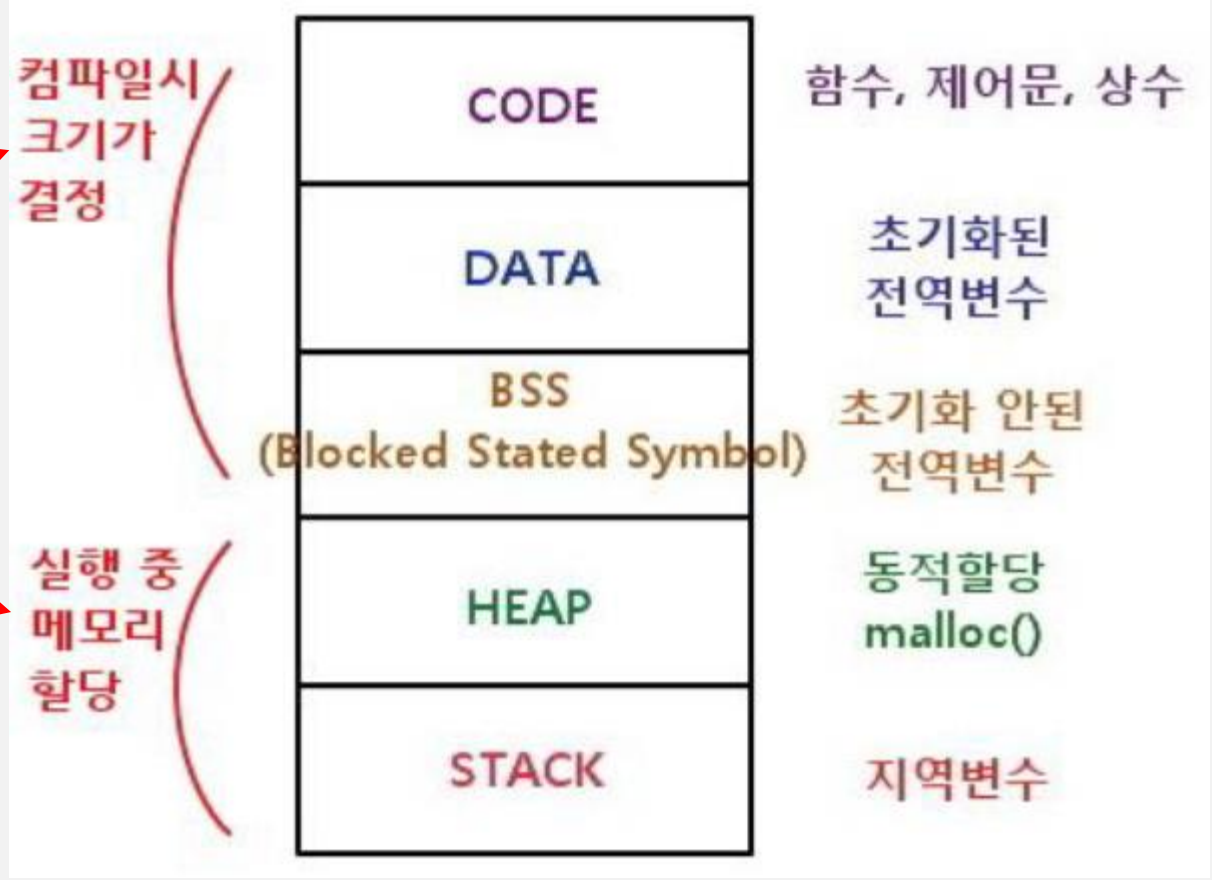


1. 컴퓨터의 구조 (Computer Architecture)

A. 우리가 프로그래밍에서 사용하는 메인 메모리는 RAM을 의미한다.

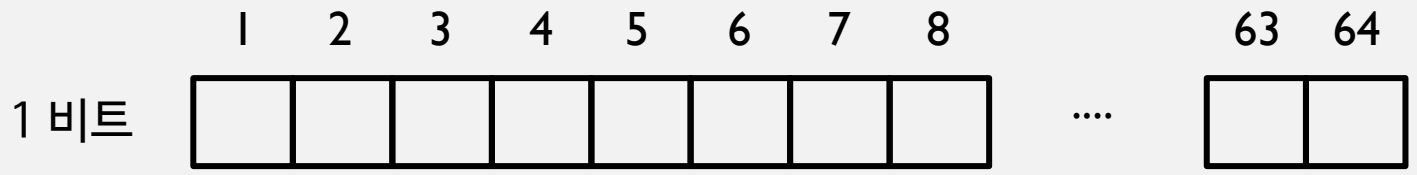
	syntax error	runtime error	logic error
원인	코드의 형식적 오류	실행 중에 예상외의 사태가 발생하여 동작이 중지됨	기술한 처리 내용에 논리적인 오류가 있음
알아 채는 방법	컴파일하면 에러 남	실행하면 도중에 강제 종료 됨	실행하면 예상외의 값이 나옴
해결 방법	컴파일러의 지적을 보고 수정	에러	원인을 스스로 찾아서 해결해야 함

Java 15장 예외. 교안에서 발췌



Java 9장 클래스. 교안에서 발췌

1. 컴퓨터의 구조 (Computer Architecture)



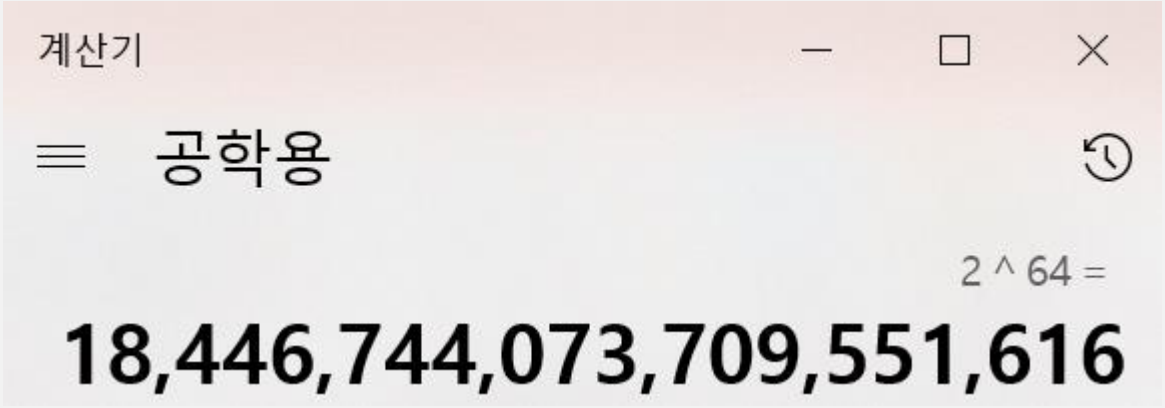
0 0 0 0 0 0 0 0 0 0

| | | | | | | | | | |

경우의 수 계산: 2 4 8 16 32 .....

경우의 수 계산: 2 2^2 2^3 2^4 2^5 2^6 ..... 2^{63} 2^{64}

2^{64} = ?



1. 컴퓨터의 구조 (Computer Architecture)

김유두 교수님, 소프트웨어코딩 수업, C언어 기본 문법과 변수

자료형	크기	범위	비고
char signed char	1바이트, 8비트	-128~127	
unsigned char	1바이트, 8비트	0~255	
short short int	2바이트, 16비트	-32,768~32,767	int 생략 가능
unsigned short unsigned short int	2바이트, 16비트	0~65,535	int 생략 가능
int signed int	4바이트, 32비트	-2,147,483,648~ 2,147,483,647	
unsigned unsigned int	4바이트, 32비트	0~4,294,967,295	int 생략 가능
long long int signed long signed long int	4바이트, 32비트	-2,147,483,648~ 2,147,483,647	int 생략 가능
unsigned long unsigned long int	4바이트, 32비트	0~4,294,967,295	int 생략 가능
long long long long int signed long long signed long long int	8바이트, 64비트	-9,223,372,036,854,775,808~ 9,223,372,036,854,775,807	int 생략 가능
unsigned long long unsigned long long int	8바이트, 64비트	0~18,446,744,073,709,551,615	int 생략 가능

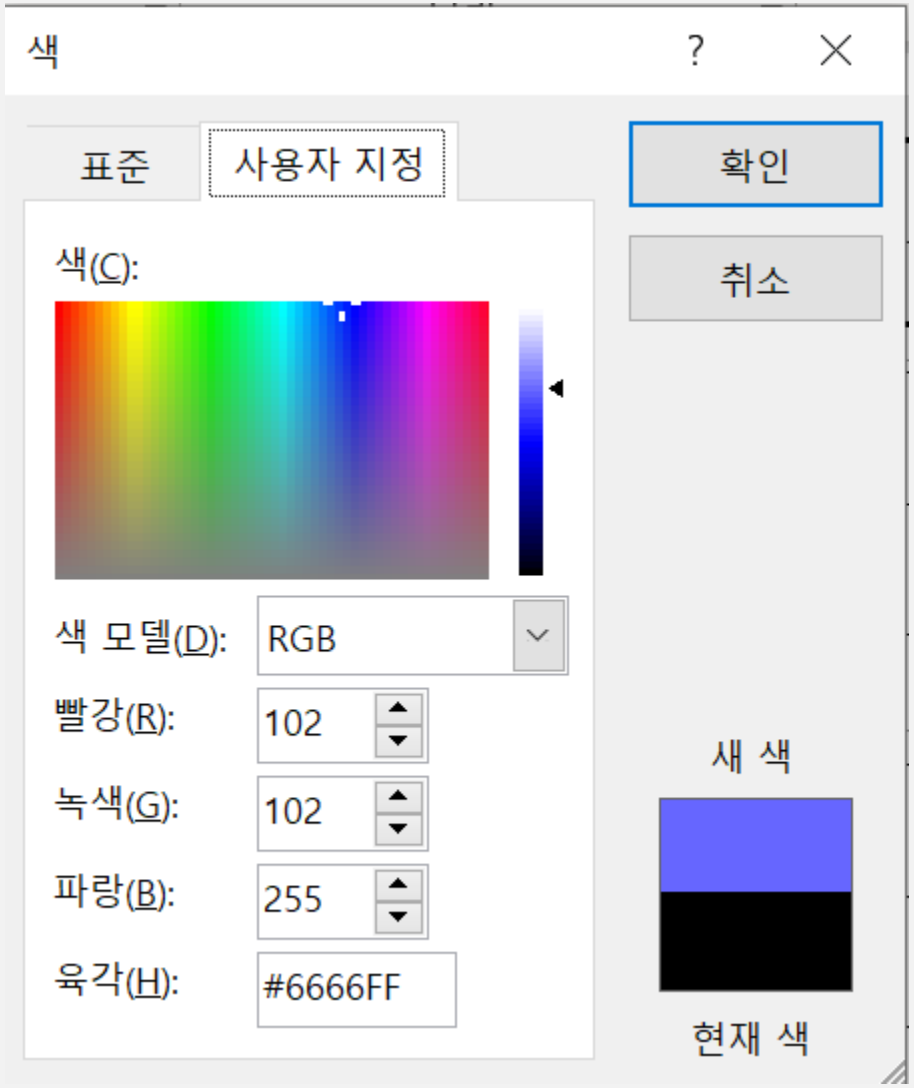
1. 컴퓨터의 구조 (Computer Architecture)

컴퓨터는 그림파일을 어떻게 저장할까?

해상도와 픽셀, RGB와 Color code, 4K? UHD?
저장 형식에 따라 다름 (BMP, JPG, JPEG, PNG...)
압축 알고리즘 문제!

그럼 동영상은??

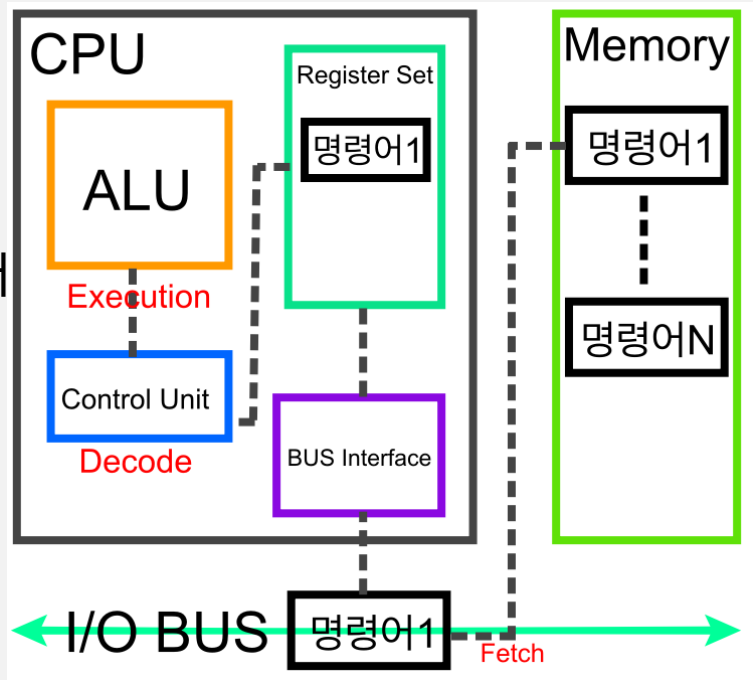
동영상 화질과 시간 정보
(encoding, decoding)



2. 컴퓨터의 작동방식

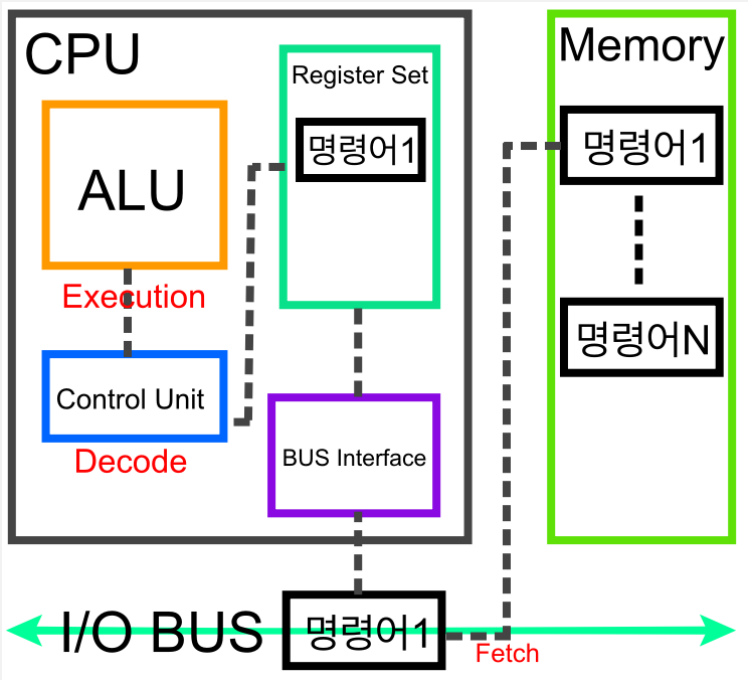
2. 컴퓨터의 작동 방식

1. CPU 명령어의 실행 순서
   1) Fetch
      - 실행할 명령어를 가져온다. 이후 단계들도 그렇지만 한번에 보통 4개 정도를 처리함. 슈퍼스칼라 라는 기술은 이렇게 한 사이클에 여러 명령어를 처리하는 기술을 의미함
   2) Decode
      - 컨트롤 유닛이 이진수로 구성된 명령을 해석하고 명령어 종류를 구분
   3) Rename
      - 명령어가 가리키는 레지스터들을 실제 물리적인 레지스터와 맵핑하는 과정
   4) Dispatch
      - 명령어가 실행하기 위해 기다리는 대기열(레지스터)에 명령어를 넣는 과정
   - Issue
      - 대기열에 있는 명령어가 실행 가능하면, 그 명령을 실행하는 장치(계산 명령의 경우 ALU, 기록 명령의 경우 Cache)로 보내는 것. 프로그램에서 시간상 뒤의 명령어가 앞의 명령어보다 먼저 Issue될 수 있음.



2. 컴퓨터의 작동 방식

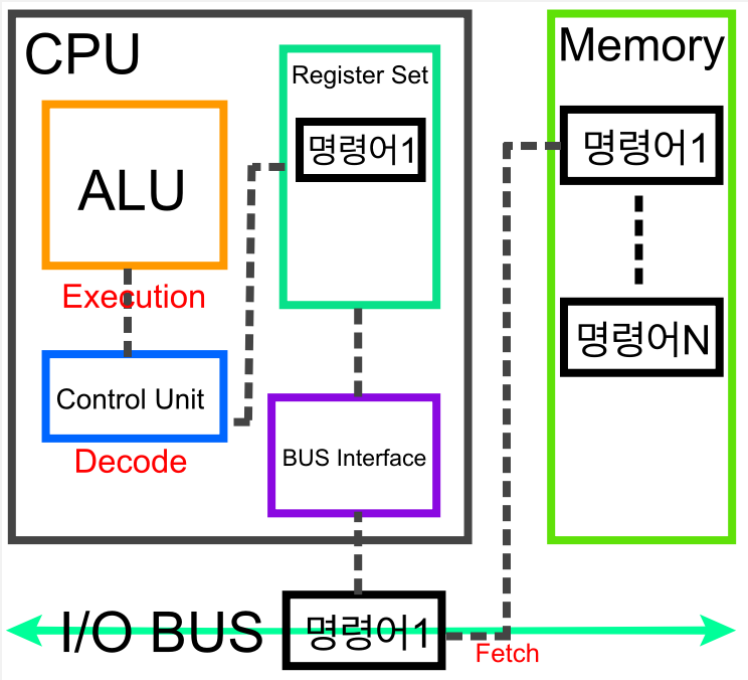
1. CPU 명령어의 실행 순서
   6) Execute
      - Issue된 명령을 실행하는 단계
   7) Writeback
      - 실행된 결과 값을 레지스터에 임시 저장할 필요가 있다면 저장하고, 결과 값을 기다리고 있던 명령어가 있다면 결과가 나왔다고 알려주는 것
   8) Commit
      - 명령어 수행을 완료하고, 명령어 실행을 위해 할당 받은 자원을 모두 초기화해 새로운 명령 수행을 위해 준비하는 것.
      - 명령어의 실행 결과를 사용자에게 노출시키며 (Commit 전에는 노출 안됨), 이후 명령어의 실행을 취소할 수 없다. 최종 과정
   9) CPU는 기본적으로 위 8개 과정을 반복한다.
   10) 위 과정 외에도 몇몇 핵심 기술들이 필요하다.
      - Cache (캐시메모리) : 주 메모리 값을 임시 저장하는 작고 빠른 메모리. 주 메모리 접근은 CPU 입장에서는 100 Cycle 정도 걸리는 매우 느린 동작이므로 Cache가 없다면 명령을 빠르게 실행하고 싶어도 가져온 명령이 없어 불가능하다. 이를 위해 CPU 내부에 임시로 저장하는 메모리 공간



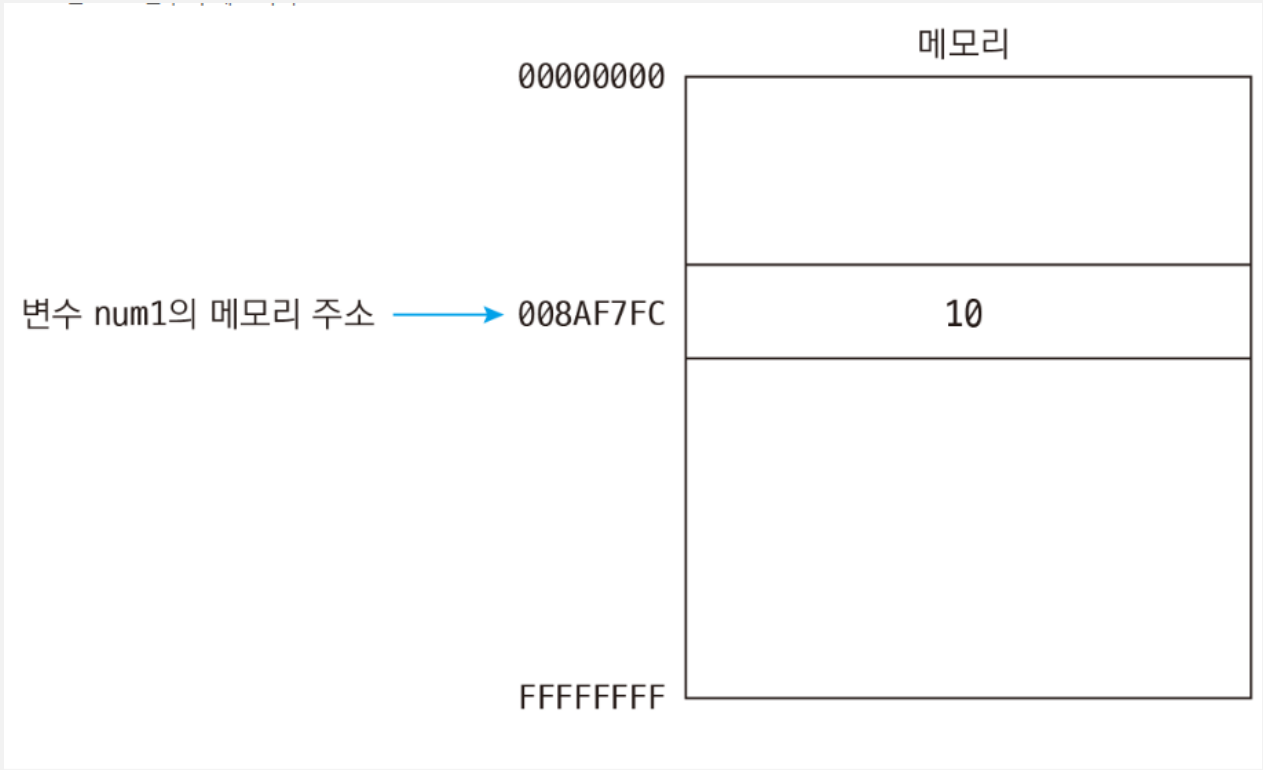
2. 컴퓨터의 작동 방식

- Branch prediction(분기예측) : Fetch를 매 사이클 마다 하기 위해서는 다음에 실행할 명령어가 저장된 메인 메모리의 주소를 알아야 한다. 분기(Branch) 명령어는 어떤 조건이 맞으면 다음에 실행할 명령어의 위치를 임의 지정할 수 있게 하는 명령어 이다.(조건문) 그런데 분기 명령의 결과가 나오려면 시간이 꽤 걸린다. 프로그램마다 차이는 있지만, 명령어의 약 30% 정도는 분기이다. 따라서 분기 명령의 결과를 과거의 기록을 기반으로 예측하는 기술이 분기예측기술이다.

- Speculative memory disambiguation



2. 컴퓨터의 작동 방식



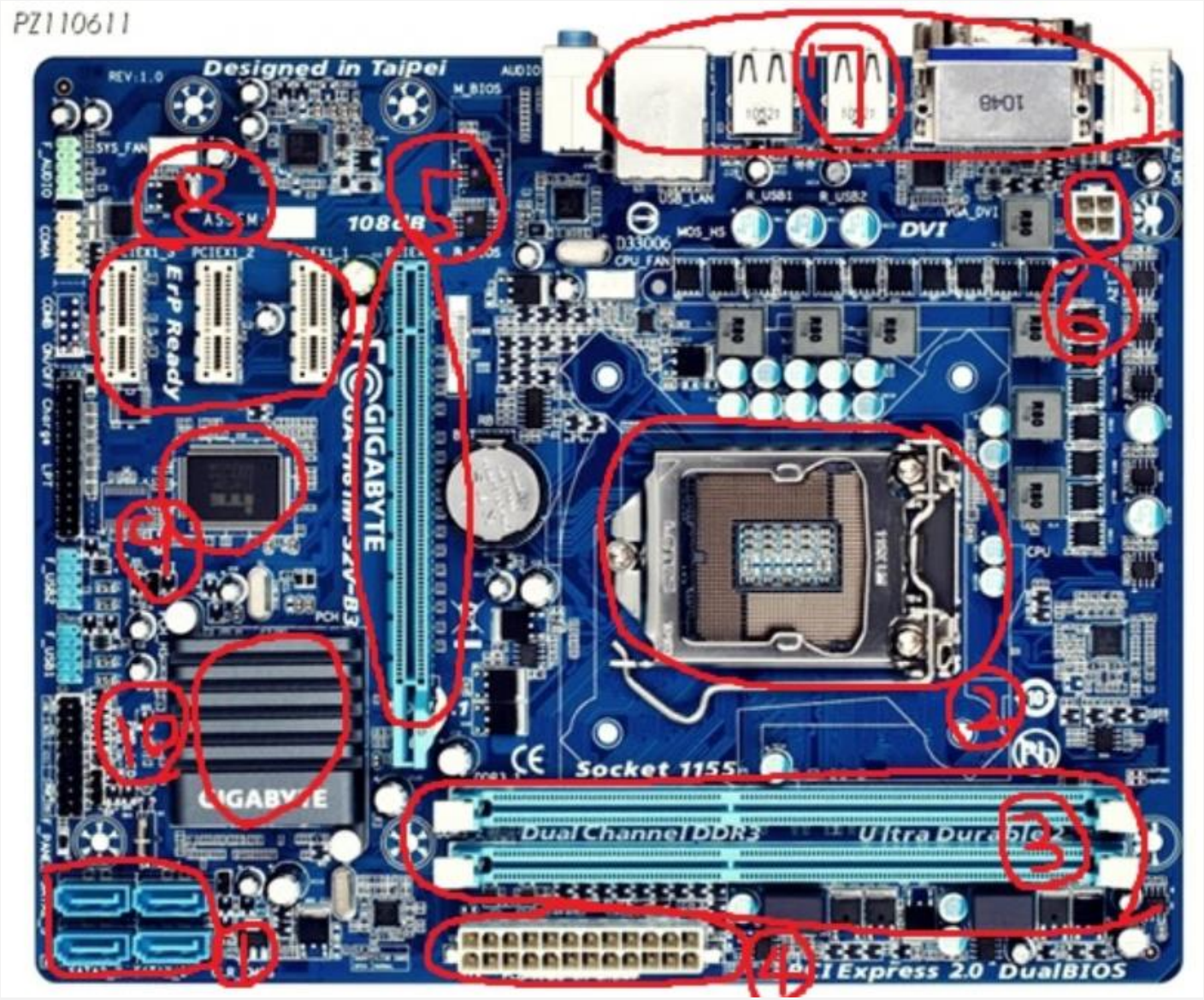
참고 | **32비트, 64비트와 메모리 주소**

다음과 같이 시스템이 32비트인지 64비트인지에 따라 메모리 주소의 범위도 달라집니다.

- 32비트: 16진수 8자리
  - 0x00000000 ~ 0xFFFFFFFF
  - 예) 0x008AF7FC
- 64비트: 16진수 16자리
  - 0x0000000000000000 ~ 0xFFFFFFFFFFFFFFFF
  - 예) 0x00000000008AF7FC
  - 64비트 메모리 주소는 0x00000000`00000000처럼 8자리씩 끊어서 `를 붙이기도 합니다.

앞으로 16진수 8자리 또는 16자리로 된 숫자가 나오면 일단 메모리 주소라 생각하면 됩니다. 이제 16진수 표기는 두려워하지 마세요.

2. 컴퓨터의 작동 방식


PZ110611
Designed in Taipei
DVI
Socket 1155
Dual Channel DDR3
Ultra Durable 2
Express 2.0 DualBIOS
GIGABYTE

3. 컴퓨터가 내 소스 코드를 어떻게 읽나요?

3. 컴퓨터가 내 소스코드를 어떻게 읽냐고!

2588번 | 제출 | 맞은 사람 | 숏코딩 | 재채점/수정 | 채점 현황 | 내 소스 | 강의 | 질문 검색

곱셈 성공 출처

☆

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
1 초	128 MB	44246	23907	21647	55.940%

문제

(세 자리 수) × (세 자리 수)는 다음과 같은 과정을 통하여 이루어진다.

```
      4 7 2 ····· (1)
    × 3 8 5 ····· (2)
-----------
      2 3 6 0 ····· (3)
    3 7 7 6   ····· (4)
1 4 1 6       ····· (5)
-----------
1 8 1 7 2 0 ····· (6)
```

(1)과 (2)위치에 들어갈 세 자리 자연수가 주어질 때 (3), (4), (5), (6)위치에 들어갈 값을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 (1)의 위치에 들어갈 세 자리 자연수가, 둘째 줄에 (2)의 위치에 들어갈 세자리 자연수가 주어진다.

출력

첫째 줄부터 넷째 줄까지 차례대로 (3), (4), (5), (6)에 들어갈 값을 출력한다.

예제 입력 1 복사

```
472
385
```

예제 출력 1 복사

```
2360
3776
1416
181720
```

3. 컴퓨터가 내 소스코드를 어떻게 읽냐고!

```java
import java.util.Scanner;
public class Main {
    public static int[] divided(int a) {
        int[] intArray = new int[3];
        for (int i = 0; i < 3; i++) {
            intArray[i] = (a / (int) Math.pow(10, 2 - i));
            a = a % (int) Math.pow(10, 2 - i);
        }
        return intArray;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a = sc.nextInt();
        int b = sc.nextInt();
        int[] intB = divided(b);
        int result = 0;
        for (int i = 0; i < 3; i++) {
            System.out.println(a * intB[2 - i]);
            result += a * intB[2 - i] * Math.pow(10, i);
        }
        System.out.println(result);
    }
}
```

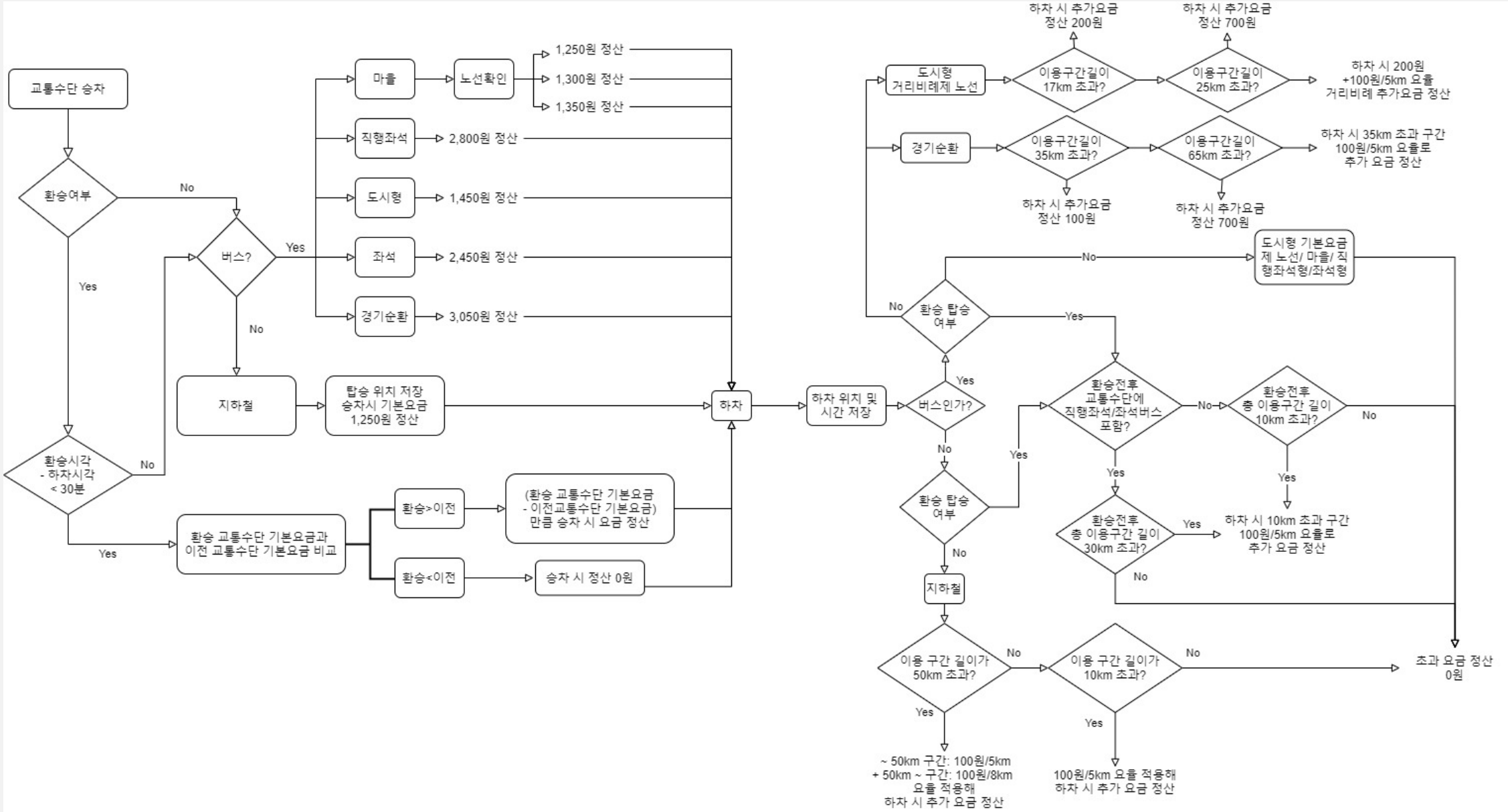
4. 알고리즘 대체 뭔데

4. 알고리즘? 먹는건가요?

https://full-of-dream.tistory.com/7

https://www.zdnet.co.kr/view/?no=20150622080223

4. 알고리즘? 먹는건가요?


교통수단 승차
환승여부
No
Yes
버스?
Yes
No
마을
노선확인
1,250원 정산
1,300원 정산
1,350원 정산
직행좌석
2,800원 정산
도시형
1,450원 정산
좌석
2,450원 정산
경기순환
3,050원 정산
지하철
탑승 위치 저장
승차시 기본요금
1,250원 정산
환승시각
- 하차시각
< 30분
No
Yes
환승 교통수단 기본요금과
이전 교통수단 기본요금 비교
환승>이전
(환승 교통수단 기본요금
- 이전교통수단 기본요금)
만큼 승차 시 요금 정산
환승<이전
승차 시 정산 0원
하차
하차 위치 및
시간 저장
버스인가?
Yes
No
환승 탑승
여부
No
Yes
도시형
거리비례제 노선
이용구간길이
17km 초과?
하차 시 추가요금
정산 200원
이용구간길이
25km 초과?
하차 시 추가요금
정산 700원
하차 시 200원
+100원/5km 요율
거리비례 추가요금 정산
경기순환
이용구간길이
35km 초과?
하차 시 추가요금
정산 100원
이용구간길이
65km 초과?
하차 시 추가요금
정산 700원
하차 시 35km 초과 구간
100원/5km 요율로
추가 요금 정산
환승전후
교통수단에
직행좌석/좌석버스
포함?
No
Yes
환승전후
총 이용구간 길이
10km 초과?
No
Yes
하차 시 10km 초과 구간
100원/5km 요율로
추가 요금 정산
환승전후
총 이용구간 길이
30km 초과?
Yes
No
도시형 기본요금
제 노선/ 마을/ 직
행좌석형/좌석형
지하철
이용 구간 길이가
50km 초과?
No
Yes
~ 50km 구간: 100원/5km
+ 50km ~ 구간: 100원/8km
요율 적용해
하차 시 추가 요금 정산
이용 구간 길이가
10km 초과?
No
Yes
100원/5km 요율 적용해
하차 시 추가 요금 정산
초과 요금 정산
0원

5. 알고리즘 문제 풀이 접근 방법

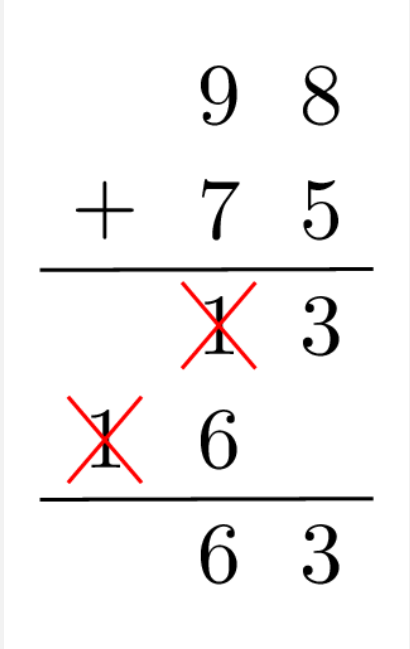
5. 전체 문제

paiza - C061 : 문제

아이들은 덧셈을 할 때 손가락으로 세어 덧셈하기 때문에 반복 오름의 계산을 할 수 없습니다.
정수가 2개 입력 받아 아이들의 계산방법으로 계산한 결과를 출력하는 프로그램을 작성해주세요. 단, 입력 예2와 같이 선두의 수가 0이 된 경우에도 생략하지 않고 0 을 출력하세요. 예를 들어 예1에서는 아이들은 그림과 같이 계산합니다.

1의 자리는 8+ 5를 계산결과 13이 나와도 올림을 무시해 3, 10의 자리는 9+7을 계산하면 16이 나오지만 올림을 무시해 6이 됩니다. 따라서, 계산의 결과는 98 +75 = 63 이 됩니다. 따라서 결과를 출력하면 63이 됩니다.

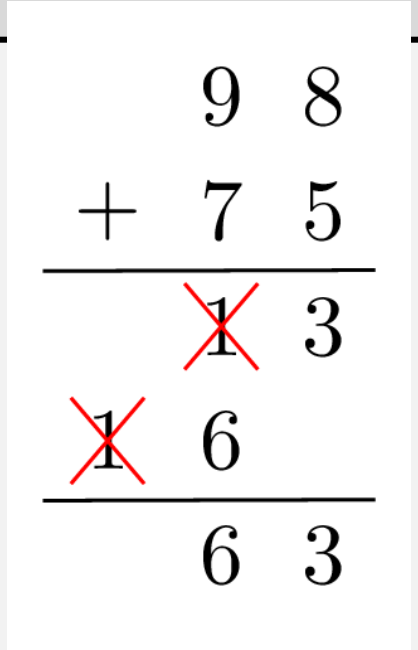
- 입력되는 값
  덧셈에 사용하는 정수 A, B가 공백으로 구분하여 입력
  입력 한 줄이 되어, 끝에 줄 바꿈이 하나 들어갑니다.
  각각의 값은 문자열로 표준 입력으로 전달됩니다.
- 기대하는 출력
  반복 올림을 무시하고 A와 B를 더한 결과를 출력합니다.
  출력은 A와 B의 큰 자릿수에 맞추어주세요. 끝에 줄 바꿈을 넣고 불필요한 문자나 빈 행을 포함하지 마십시오.
- 조건: 1 ≦ A, B ≦ 999



입력 예 1	입력 예 2	입력 예 3
98 75	274 840	624 58
출력 예 1	출력 예 2	출력 예 3
63	014	672

5.1). 문제 분석

paiza - C061 : 문제



입력부분
정수가 2개 입력 받아 아이들의 계산방법으로 계산한 결과를 출력하는 프로그램 작성 (1 ≦ A, B ≦ 999)
↳ 입력 받는 정수가 한자리, 두자리, 세자리 수인지 모른다.
↳ 각 정수 A, B는 공백으로 구분되어 한 줄로 입력되고 끝에 줄 바꿈(엔터)가 하나 들어간다.
↳ 각각의 값은 문자열 형태로 표준 입력으로 전달된다.

연산부분
아이들은 올림 계산을 할 수 없습니다
↳ 올림을 하지 않는다. → 덧셈 결과 10 이상이면 10의 자리 숫자를 버리고 1의 자리 숫자만 계산 결과로 저장한다.
1의 자리는 8+ 5를 계산결과 13이 나와도 올림을 무시해 3, 10의 자리는 9+7을 계산하면 16이 나오지만 올림을 무시해 6.
따라서, 계산의 결과는 98 +75 = 63, 따라서 결과를 출력하면 63이 됩니다.

출력부분
출력은 A와 B의 큰 자릿수에 맞추어주세요.
끝에 줄 바꿈을 넣고 불필요한 문자나 빈 행을 포함하지 마십시오.
단, 입력 예2와 같이 선두의 수가 0이 된 경우에도 생략하지 않고 0 을 출력하세요.

입력 예 1	입력 예 2	입력 예 3
98 75	274 840	624 58
출력 예 1	출력 예 2	출력 예 3
63	014	672

5.2). 각 부분 코드로 구현 (입력 & 연산부)

- 입력부분

```
public static void main(String[] args) {
    // 입력 부분
    Scanner sc = new Scanner(System.in);
    String a = sc.next();    // 98
    String b = sc.next();    // 75
```

- 문제에서 문자열 형태로 입력 받는다고 했으므로 next() 또는 nextLine()을 사용하여 입력 받는다.
- next()가 사용하기 편하다.

- 연산부분

```
    /* 연산 부분
     * 각 자리 수 별로 덧셈하므로 1,10,100 자리 숫자를 각각 쪼개주어야 한다. */
    String[] strArrayA = a.split( regex: "");    // 98 -> 9 , 8
    String[] strArrayB = b.split( regex: "");    // 75 -> 7 , 5

    /* 정수 A, B의 자리수가 다를 수가 있는 부분을 고려해야 한다.
     * Ex] A = 15, B = 203 -> A의 length = 2, B의 length = 3 */
    int aLength = strArrayA.length;
    int bLength = strArrayB.length;
    int length = Math.max(aLength,bLength); // 자리수가 큰 값의 자리수 값 확보
```

5.2). 각 부분 코드로 구현 (연산부)

- 연산부분

```
String compareLength = "a>b";   // aLength > bLength
int diffLength = Math.abs(aLength - bLength); // 자리 수의 차이를 계산
if (aLength < bLength) {    // aLength < bLength
    compareLength = "a<b";
}

/* List의 경우 index = 0일 때 각 정수의 자리수가 가장 큰 숫자가 포함 된다.
 * 따라서 length가 큰 정수를 기준으로 하여 자리수가 차이나는 경우
 * 작은 값의 list의 앞 부분에 자리 수 차이만큼 0을 넣어준다.*/
List<Integer> intListA = new ArrayList<>();
List<Integer> intListB = new ArrayList<>();
if (compareLength.equals("a>b")) {
    // diffLength >= 1
    for (int i = 0; i < diffLength; i++) {  // if diffLength == 1
        intListB.add(0);                    // for문은 i == 0인 경우 1바퀴 돈다
    }                                       // Length 차이만큼 앞 자리에 0을 넣는다.
} else if (compareLength.equals("a<b")) {
    for (int i = 0; i < diffLength; i++) {
        intListA.add(0);
    }
}
```

5.2). 각 부분 코드로 구현 (연산부)

- 연산부분

```
/*문자 타입은 숫자 덧셈이 안되므로 문자열 List에 들어있는
 *각 문자형 숫자를 각각 int형으로 type casting*/
for (String s : strArrayA) {
    intListA.add(Integer.parseInt(s));
}
for (String s : strArrayB) {
    intListB.add(Integer.parseInt(s));
}

// 올림 무시 덧셈 연산 수행

List<Integer> resultSum = new ArrayList<>(); // 연산 결과를 저장할 List 선언
/* length = aLength = bLength
 * Line 16 ~ 45 에서 자리 수가 같도록 설정해주었음 */
for (int i = 0; i < length; i++) {
    resultSum.add(intListA.get(i) + intListB.get(i));     // 올림을 무시하므로 큰 자리수부터 계산해도 무방
    if ((resultSum.get(i) / 10) == 1) {                   // 덧셈 결과 올림이 존재하면
        resultSum.set(i, resultSum.get(i) % 10);          // i번째 자리에 올림을 제외한 나머지만 덮어씌운다
    }                                                     // (올림을 무시)
}
```

5.2). 각 부분 코드로 구현 (출력부)

- 출력부분

```
        // 출력부분
        for (int i = 0; i < resultSum.size(); i++) {
            System.out.print(resultSum.get(i));
        }
    }
}
```

- 결과

제출 코드 결과 상세

테스트 번호	입력 케이스 번호	심판 결과	실행 시간
테스트 1	케이스 1 (기본 데이터)	통과	0.08 초
테스트 2	케이스 1 (기본 데이터)	통과	0.08 초
테스트 3	케이스 1 (기본 데이터)	통과	0.07 초
테스트 4	케이스 1 (기본 데이터)	통과	0.08 초
테스트 5	케이스 1 (기본 데이터)	통과	0.08 초
테스트 6	케이스 1 (기본 데이터)	통과	0.08 초
테스트 7	케이스 1 (기본 데이터)	통과	0.08 초
테스트 8	케이스 1 (기본 데이터)	통과	0.09 초
테스트 9	케이스 1 (기본 데이터)	통과	0.08 초
테스트 10	케이스 1 (기본 데이터)	통과	0.08 초


Thank you!
Human!