13강.명령어 병렬 처리 기법

명령어 파이프라인

명령어 파이프라인은 CPU가 명령어를 더 빠르게 처리하기 위해 여러 단계를 겹쳐서 실행하는 기법입니다.

전공서마다 단계 구분이 다르지만, 크게 두 가지 방식으로 나눌 수 있습니다.

- 단순화 : 인출 → 실행
- 세분화 : 명렁어 인출 → 명령어 해석 → 명령어 실행 → 명령어 접근 → 결과 저장

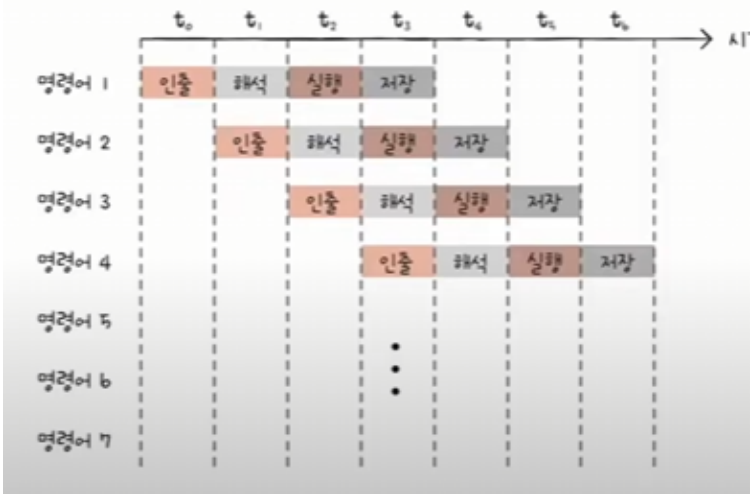
해당 강의에서는 다음과 같이 파이프라인을 정의합니다.

1. 명령어 인출 (Instruction Fetch)
    - 메모리에서 명령어를 가져오는 단계입니다.
    - 프로그램 카운터(PC)에 지정된 주소를 이용해 명령어를 읽어옵니다.
2. 명령어 해석 (Instruction Decode)
    - 가져온 명령어를 해석하고, 필요한 피연산자(레지스터, 메모리)를 확인합니다.
3. 명령어 실행 (Execute Instruction)
    - ALU 또는 실행 장치에서 실제 연산을 수행합니다.
4. 결과 저장 (Write Back)
    - 연산 결과를 레지스터나 메모리에 기록하는 단계입니다.

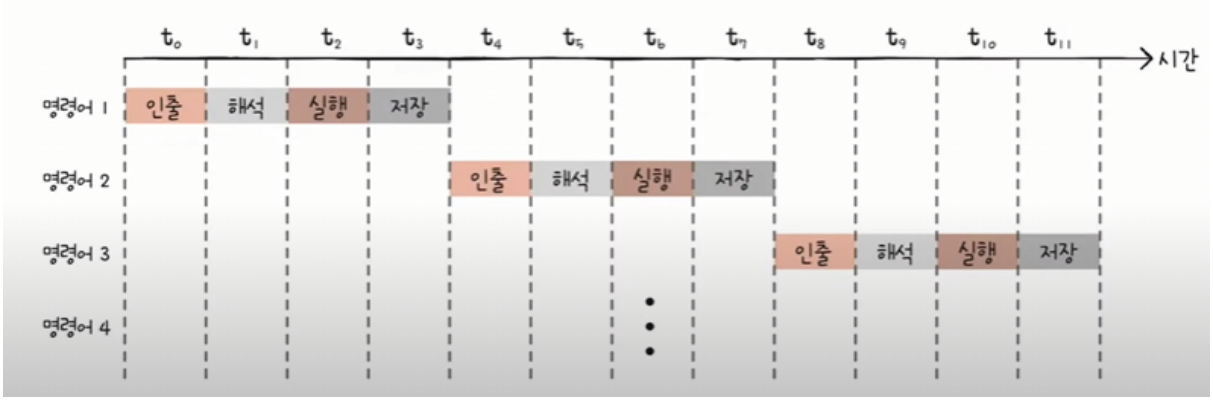
명령어 파이프라인의 효과

파이프라인을 사용하면, 같은 단계가 겹치지만 않는다면 여러 명령어가 동시에 실행 됩니다.

다음 그림과 같이, `명령어1`이 실행되는 동안 `명령어2` 는 해석되고, `명령어3` 은 인출될 수 있습니다.



만약 파이프라인이 없다면, 하나의 명령어가 완전히 끝난 후에야 다음 명령어가 시작되므로 시간이 오래걸리고 병목 현상이 발생합니다.



파이프라인 위험

파이프라인이 항상 이상적으로 동작하는 것은 아닙니다. 동시에 처리할 수 없는 상황을 파이프라인 위험이라고 합니다. 종류에 따라 크게 세 가지로 나눌 수 있습니다.

데이터위험 (Data Hazzard)

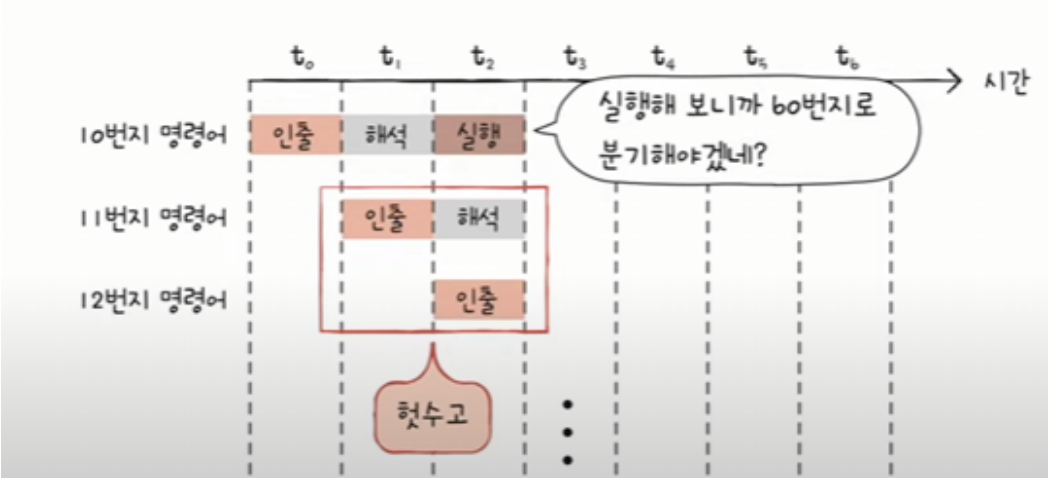
명령어 간 데이터 의존성에 의해 야기됩니다.

```
I1 : R1 ← R2 + R3
I2 : R4 ← R1 + R5 // I1이 끝나야 I2 실행 가능
```

위 예시에서는 `I1`이 `R1` 을 갱신하기 전에 `I2`가 `R1`을 읽으려고 하면 문제가 발생합니다.

제어 위험

분기(Branch), 점프(Jump) 같은 제어 흐름 변경으로 발생합니다.




```
IF (조건) JUMP L1
```

분기 여부가 확정되기 전까지 다음 명령어를 가져올 수 없습니다.

구조적 위험

서로 다른 명령어가 동시에 같은 하드웨어 자원을 사용하려 할 때 발생합니다.

예시로 메모리 접근을 동시에 요구하거나, ALU가 한 번에 하나 밖에 못쓰일 때로 들 수 있습니다.

슈퍼스칼라(Superscalar)

CPU에 여러 개의 파이프라인을 두어 동시에 여러 명령어를 병렬 처리하는 구조입니다.

이론적으로는 파이프라인 개수에 비례하여 성능이 향상될 수 있습니다. 하지만 실제로는 파이프라인 위험 증가로 인해 비례하게 성능이 좋아지진 않습니다.

비순차적 명령어 처리 (Out-of-Order Execution)

현대 CPU 성능 향상에 크게 기여한 기술입니다. 순차적 실행 원칙을 깨고, 서로 의존성이 없는 명령어는 순서를 바꿔서 먼저 실행하는 방식입니다.

간단한 예시로는 합법적인 새치기라고 생각할 수 있습니다.

예시 1. 순차적 처리

```
I1: R1 ← R2 + R3
I2: R4 ← R1 + R5
I3: R6 ← R7 + R8
```

- 순서대로 실행하면, `I2` 는 `I1` 이 끝날 때까지 기다려야 합니다.

예시 2. 비순차적 처리

```
I1: R1 ← R2 + R3
I3: R6 ← R7 + R8   // I1과 독립적이므로 먼저 실행 가능
I2: R4 ← R1 + R5
```

- `I3`은 `I1`과 의존성이 없으므로 `I2`보다 먼저 실행됩니다.
- 전체 실행 시간이 단축되는 효과를 얻습니다.