Chapter 02

컴퓨터의 구조와 성능 향상

Contents

01 컴퓨터의 기본 구성

02 CPU와 메모리

03 컴퓨터 성능 향상 기술

04 병렬 처리

05 [심화학습] 무어의 법칙과 암달의 법칙

학습목표

- 컴퓨터를 구성하는 하드웨어의 특성을 이해한다.
- CPU와 메모리의 구성 및 동작 방식을 이해한다.
- 컴퓨터의 성능을 향상하는 기술을 알아본다.
- 병렬 처리의 개념을 이해하고 병렬 처리 기법을 알아본다.

1 하드웨어의 구성

컴퓨터의 구성

- 필수장치 : 중앙처리장치, 메인메모리
- 주변장치 : 입력장치, 출력장치, 저장장치

그림 2-1 컴퓨터를 구성하는 장치

1 하드웨어의 구성

용어 통일

- 메인메모리 → **메모리**
- 보조저장장치 → **저장장치**
- 중앙처리장치 → **CPU**

CPU와 메모리

- CPU : 명령어를 해석하여 실행하는 장치로 인간으로 치면 두뇌에 해당
- 메모리
  - 작업에 필요한 프로그램과 데이터를 저장하는 장소
  - 바이트 단위로 분할되어 있으며 분할 공간마다 주소(**address**)로 구분

입출력장치

- 입력장치 : 외부의 데이터를 컴퓨터에 입력하는 장치
- 출력장치 : 컴퓨터에서 처리한 결과를 사용자가 원하는 형태로 출력하는 장치

1 하드웨어의 구성

저장장치

- 메모리보다 느리지만 저렴하고 용량이 큼
- 전원의 온·오프와 상관없이 데이터를 영구적으로 저장
- 느린 저장장치를 사용하는 이유는 저장 용량에 비해 가격이 싸기 때문
- 종류
  - 자성을 이용하는 장치 : 카세트테이프, 플로피디스크, 하드디스크 등
  - 레이저를 이용하는 장치 : CD, DVD, 블루레이디스크 등
  - 메모리를 이용하는 장치 : USB 드라이버, SD 카드, CF 카드, SSD 등

(a) 자성을 이용하는 장치 (b) 레이저를 이용하는 장치 (c) 메모리를 이용하는 장치

그림 2-2 다양한 저장장치

1 하드웨어의 구성

메인보드

- CPU와 메모리 등 다양한 부품을 연결하는 커다란 판
- 전력이 공급되면 버스(**bus**)로 연결된 부품이 작동
- 그래픽카드, 사운드카드, 랜카드 등이 기본으로 장착되어 있기도 하고, 성능을 향상하기 위해 따로 장착하기도 함

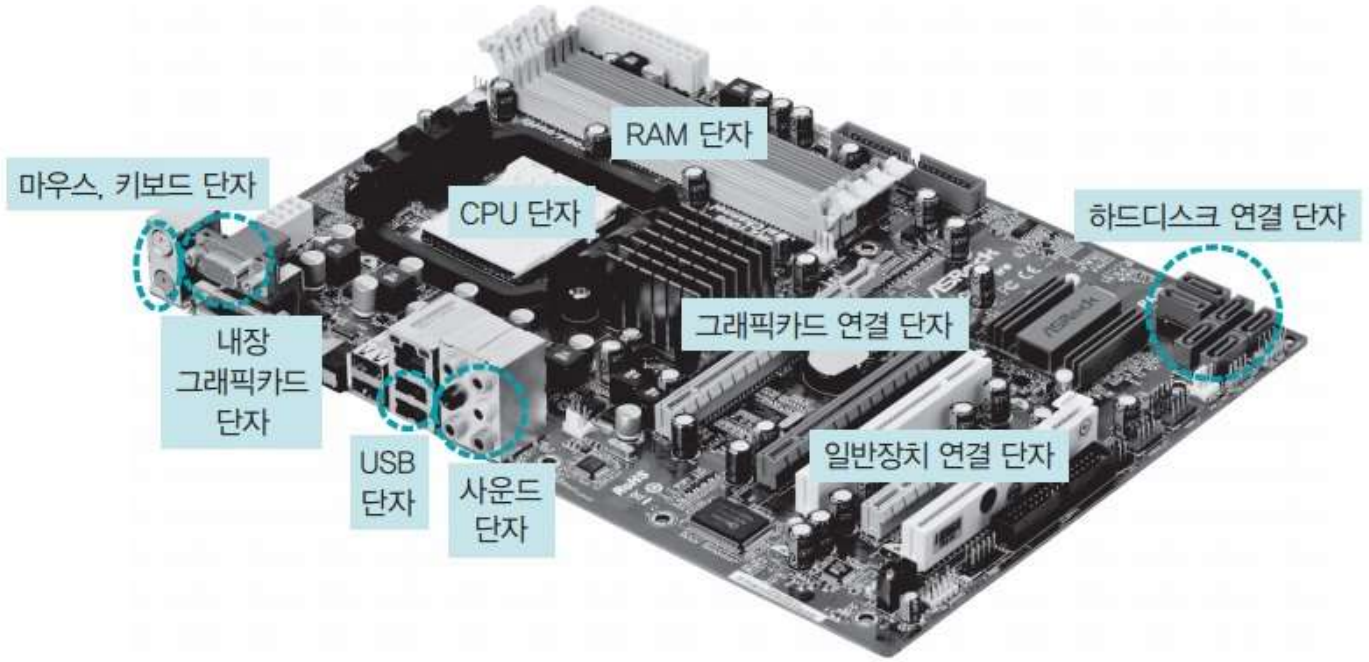


그림 2-3 메인보드

2 폰노이만 구조

폰노이만 구조

- CPU, 메모리, 입출력장치, 저장장치가 버스로 연결되어 있는 구조
- 하드웨어는 그대로 둔 채 작업을 위한 프로그램만 교체하여 메모리에 올리는 방식

정의 2-1 폰노이만 구조

모든 프로그램은 메모리에 올라와야 실행할 수 있다.

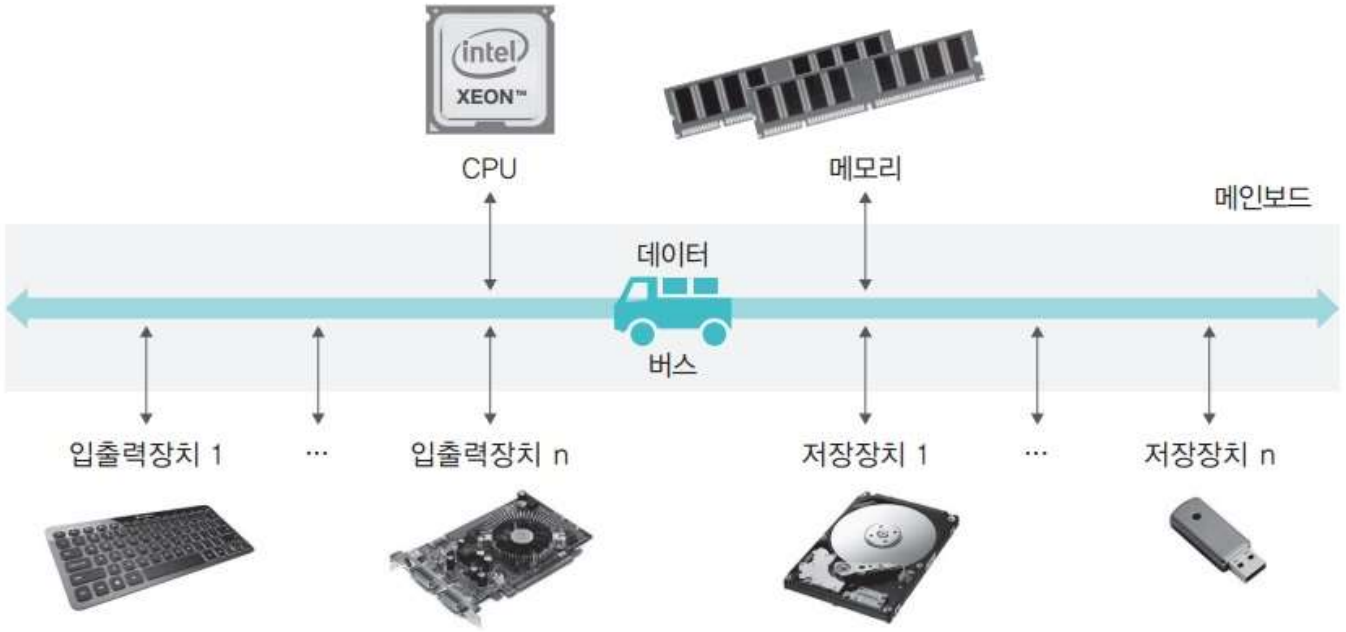


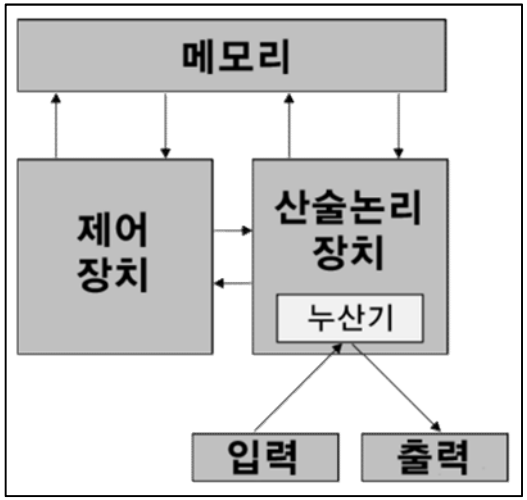
그림 2-4 폰노이만 구조

2 폰노이만 구조

폰노이만 구조

- 현대의 컴퓨터들은 폰 노이만 구조를 통해서 발전함
- CPU와 메모리 그리고 입/출력장치로 이루어진 구조
- CPU는 메모리에 내장된 프로그램을 불러와 수행 (프로그램 내장 방식)

- 프로그램 내장 방식: 전자식 기억장치에 프로그램 명령어를 저장하는 컴퓨터



3 요리사 모형

폰노이만 구조와 요리사 모형

- 운영체제의 여러 가지 현상에 대한 이해를 돕기 위한 것
- 요리사 → CPU, 도마 → 메모리, 보관 창고 → 저장장치에 비유

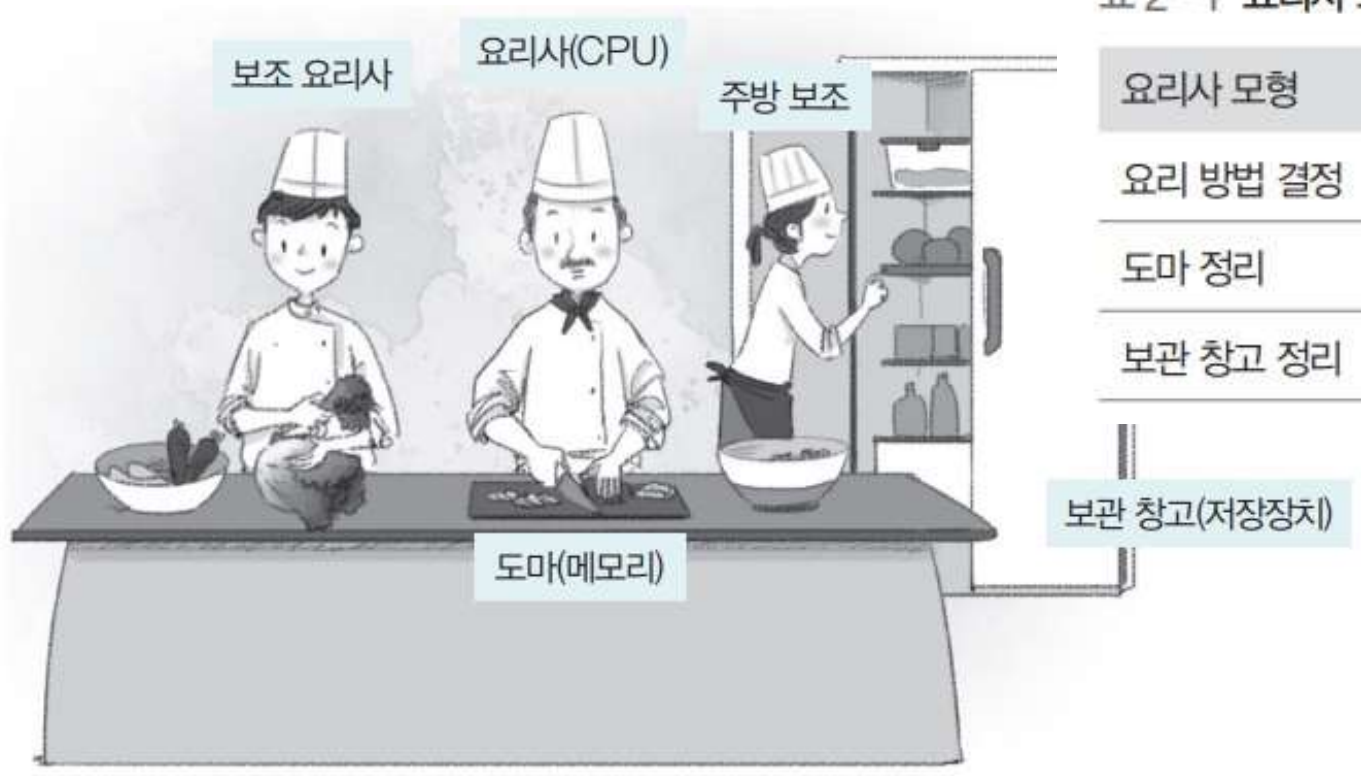


그림 2-5 요리사 모형

표 2-1 요리사 모형과 운영체제 작업의 비교

요리사 모형	운영체제 작업
요리 방법 결정	프로세스 관리
도마 정리	메모리 관리
보관 창고 정리	저장장치 관리

3 요리사 모형

도마의 크기(메모리의 크기)와 작업 속도

- 도마가 크면 재료를 모두 가져다놓고 요리할 수 있지만 도마가 작으면 재료를 모두 가져올 수 없음
- 도마가 작으면 재료 하나를 다듬어 보관 창고에 가져다놓은 뒤 다른 재료를 가져와야 하므로 재료를 손질하는 시간보다 보관 창고로 옮기는 시간이 많이 걸려 작업 속도가 떨어짐
- 도마의 크기가 전체 재료를 놓을 수 있을 만큼 충분히 크다면 작업 속도에 영향을 미치지 않음

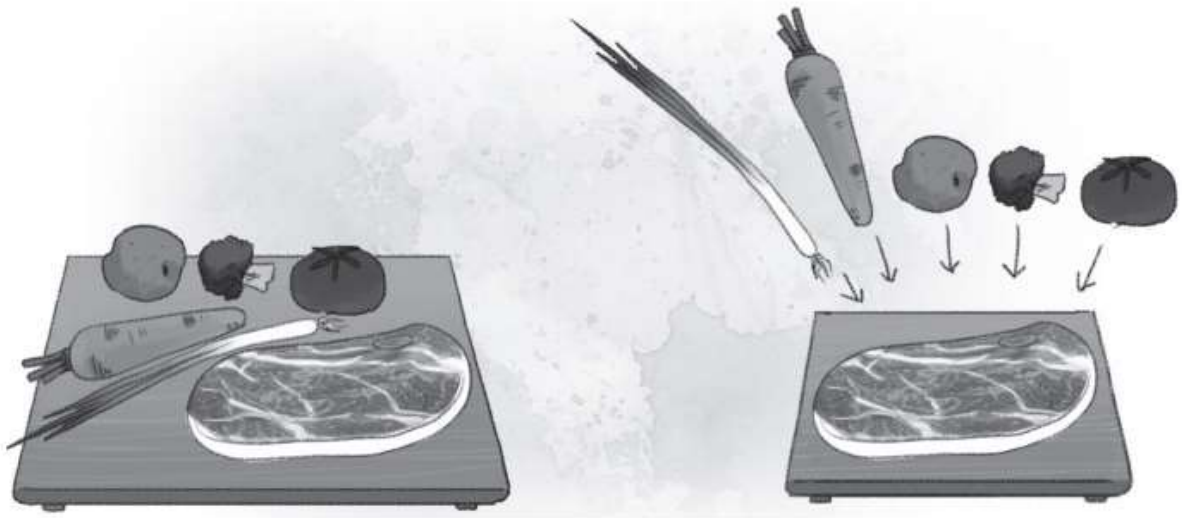


그림 2-6 도마의 크기와 작업 속도

4 하드웨어 사양 관련 용어

클록clock

- CPU의 속도와 관련된 단위
- 클록이 일정 간격으로 틱tick을 만들면 거기에 맞추어 CPU 안의 모든 구성 부품이 작업함
- 틱은 펄스pulse 또는 클록틱clock tick이라고도 부름

헤르츠(Hz)

- 클록 틱이 발생하는 속도를 나타내는 단위
- 1초에 클록틱이 한 번이면 1Hz, 1,000번이면 1kHz(1,000Hz)
- 3.4GHz는 1초에 클록틱이 3,400,000,000(3.4×10^9)번 발생하여 **CPU가 1초에 약 34억 번의 연산**(작업)을 할 수 있음을 의미

표 2-2 하드웨어 사양의 예

부품	사양
CPU	인텔 코어 i7(4코어, 3.4GHz, 캐시 4MB)
메인보드	FSB 1,333MHz
메모리	DDR3 SDRAM 4GB(1,333MHz)
그래픽카드	라데온 8500(1GB)
하드디스크(HDD)	1TB, 7200rpm, 32MB
광학디스크(ODD)	DVD 레코더(DVD: 22X, CD-R: 48X)

4 하드웨어 사양 관련 용어

시스템 버스

- 메모리와 주변장치를 연결하는 버스로 FSB[Front-Side Bus], 즉 전면 버스라고 함

CPU 내부 버스

- CPU 내부에 있는 장치를 연결하는 버스로 BSB[Back Side Bus], 즉 후면 버스라고 함

CPU와 메모리의 속도

- CPU는 CPU 내부 버스의 속도로 작동하고 메모리는 시스템 버스의 속도로 작동

1 CPU의 구성과 동작

산술논리 연산장치

- 데이터의 덧셈, 뺄셈, 곱셈, 나눗셈 같은 산술 연산과 AND, OR 같은 논리 연산을 수행

제어장치

- CPU에서 작업을 지시

레지스터

- CPU 내에 데이터를 임시로 보관

요리: 산술논리 연산장치 작업 지시: 제어장치 재료 임시 보관: 레지스터

그림 2-7 CPU의 구성 요소

1 CPU의 구성과 동작

CPU의 명령어 처리 과정

01 : int D2=2, D3 = 3, sum;
02 : sum = D2 + D3;

소스코드 2-2 어셈블리어로 변환한 덧

```
01  LOAD mem(100), register 2;
02  LOAD mem(120), register 3;
03  ADD register 5, register 2, r
04  MOVE register 5, mem(160);
```

- 01행 : 메모리의 100번지(D
- 02행 : 메모리의 120번지(D
- 03행 : 레지스터 2와 레지스
- 04행 : 레지스터 5의 값을 I

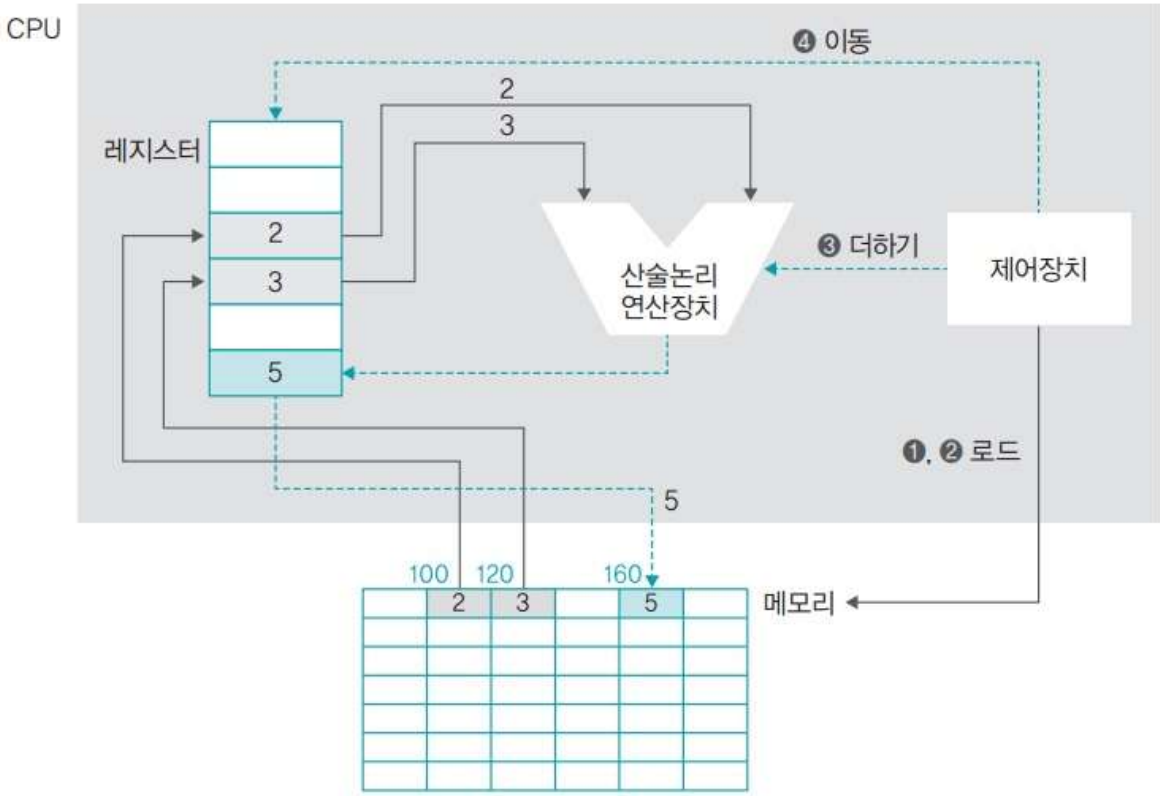


그림 2-8 CPU의 명령어 처리 과정

1 CPU의 구성과 동작

표 2-3 주요 레지스터의 종류와 특징

레지스터		특징
사용자 가시 레지스터	데이터 레지스터(DR)	CPU가 명령어를 처리하는 데 필요한 일반 데이터를 임시로 저장하는 범용 레지스터이다.
	주소 레지스터(AR)	데이터 또는 명령어가 저장된 메모리의 주소를 저장한다.
사용자 불가시 레지스터	프로그램 카운터(PC)	다음에 실행할 명령어의 위치 정보(코드의 행 번호, 메모리 주소)를 저장한다.
	명령어 레지스터(IR)	현재 실행 중인 명령어를 저장한다.
	메모리 주소 레지스터(MAR)	메모리 관리자가 접근해야 할 메모리의 주소를 저장한다.
	메모리 버퍼 레지스터(MBR)	메모리 관리자가 메모리에서 가져온 데이터를 임시로 저장한다.
	프로그램 상태 레지스터(PSR)	연산 결과(양수, 음수 등)를 저장한다.

1 CPU의 구성과 동작

‘LOAD mem(100), register 2;’의 실행 과정

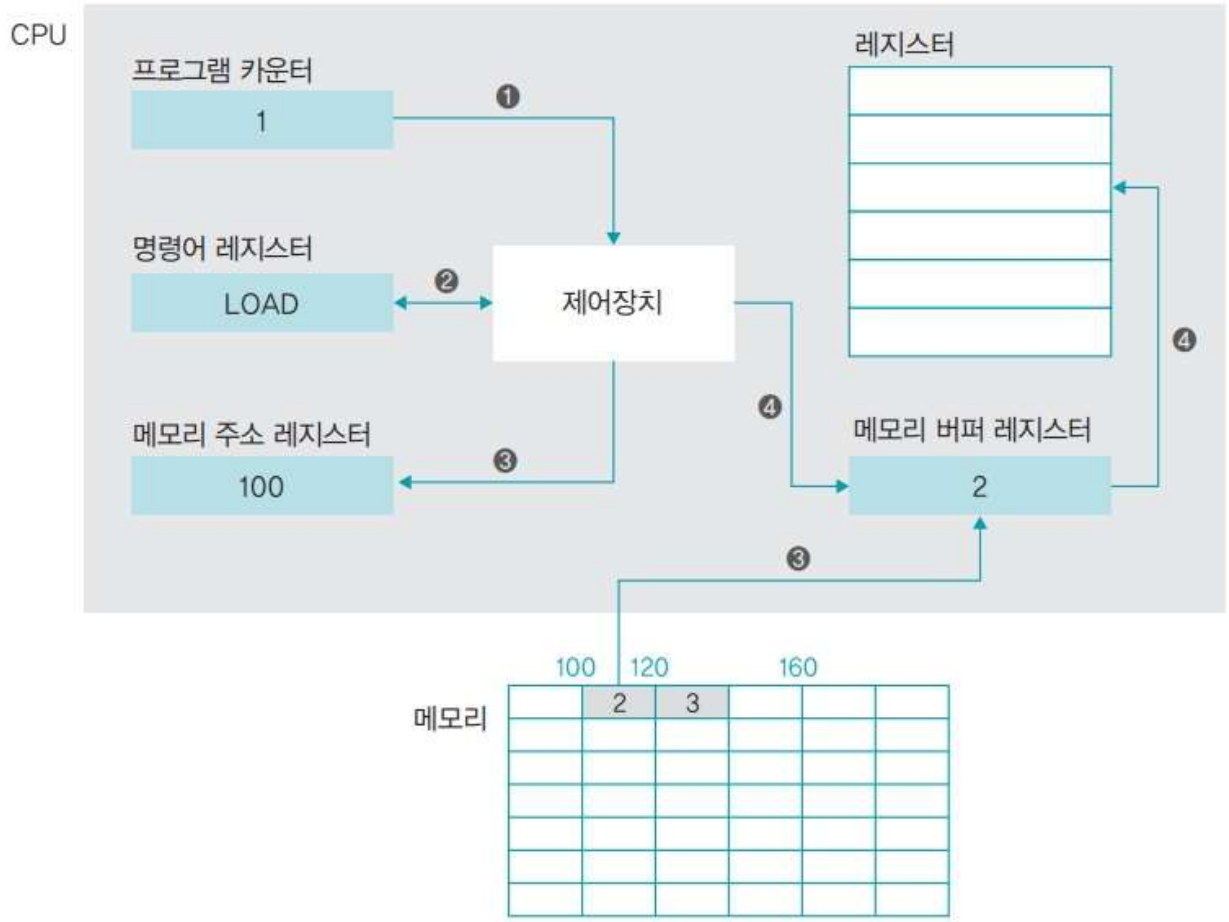


그림 2-9 'LOAD mem(100), register 2;'의 실행 과정

1 CPU의 구성과 동작

프로그램 상태 레지스터의 역할

- 연산 결과가 양수인지, 음수인지, 0이 아닌지, 자리 올림이 있는지 등 프로그램의 상태를 저장

소스코드 2-3 분기 조건이 있는 프로그램

```
01  if(D2 - D3 > 0)
02      goto 100;
03  else
04      goto 200;
```

- D2-D3의 결과를 임시로 저장하고 있다가 음수인지 양수인지를 제어 장치에 알려주어 다음에 몇 번 행으로 이동할지를 결정

1 CPU의 구성과 동작

버스의 종류

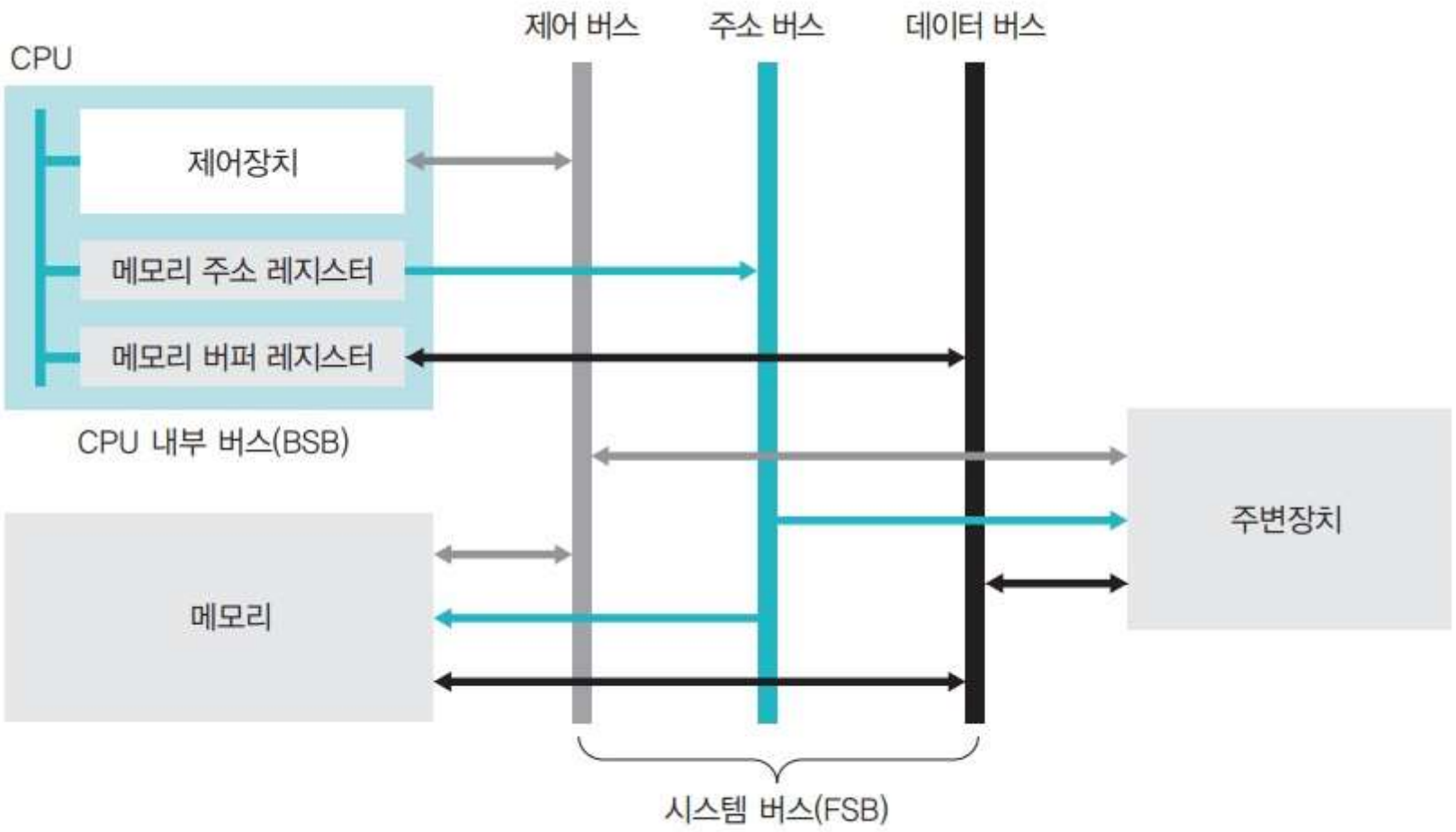


그림 2-10 **버스의 종류와 구조**

1 CPU의 구성과 동작

버스의 대역폭(bandwidth)

- 한 번에 전달할 수 있는 데이터의 최대 크기
- CPU가 한 번에 처리할 수 있는 데이터의 크기(**word**)와 같음
- 32bit CPU는 메모리에서 데이터를 읽거나 쓸 때 한 번에 최대 32bit를 처리할 수 있으며, 이 경우 레지스터의 크기도 32bit, 버스의 대역폭도 32bit
- 버스의 대역폭, 레지스터의 크기, 메모리에 한 번에 저장할 수 있는 데이터의 크기는 항상 같음

2 메모리의 종류와 부팅

메모리의 종류

그림 2-11 메모리의 종류

2 메모리의 종류와 부팅

메모리 보호의 필요성

- 현대의 운영체제는 시분할 기법을 사용하여 여러 프로그램을 동시에 실행하므로 사용자 영역이 여러 개의 작업 공간으로 나뉘어 있음
- 메모리가 보호되지 않으면 어떤 작업이 다른 작업의 영역을 침범하여 프로그램을 파괴하거나 데이터를 지울 수도 있으며, 최악의 경우 운영체제 영역을 침범하면 시스템이 멈출 수도 있음

메모리 보호 방법

❶ 작업의 메모리 시작 주소를 경계 레지스터(**base register**)에 저장 후 작업

❷ 작업이 차지하고 있는 메모리의 크기, 즉 마지막 주소까지의 차이를 한계 레지스터(**limit register**)에 저장

❸ 사용자의 작업이 진행되는 동안 이 두 레지스터의 주소 범위를 벗어나는지 하드웨어적으로 점검

그림 2-13 메모리 보호

2 메모리의 종류와 부팅

부팅

- 컴퓨터를 켰을 때 운영체제를 메모리에 올리는 과정

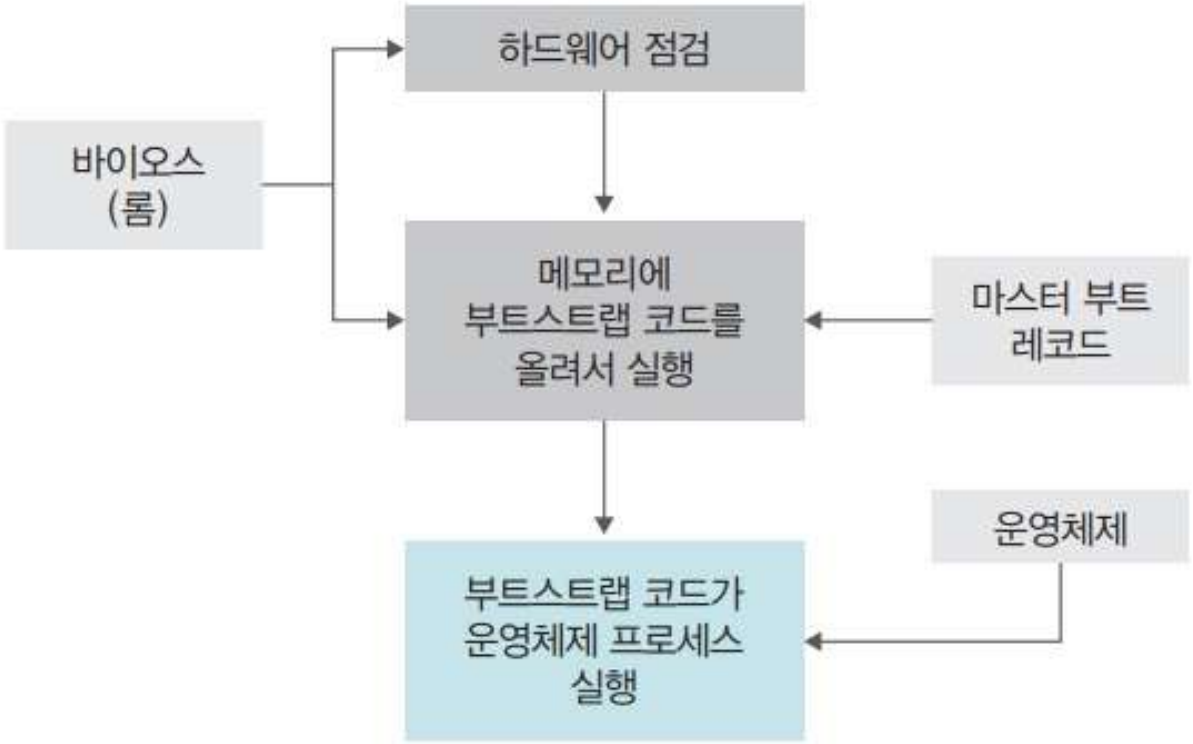


그림 2-14 부팅 과정

1 버퍼

버퍼(buffer)

- 속도에 차이가 있는 두 장치 사이에서 그 차이를 완화하는 역할을 하는 장치
- 일정량의 데이터를 모아 옮김으로써 속도의 차이를 완화
- 하드디스크 메모리 버퍼 : 1TB, 7200rpm, 32MB
- 동영상 스트리밍 버퍼

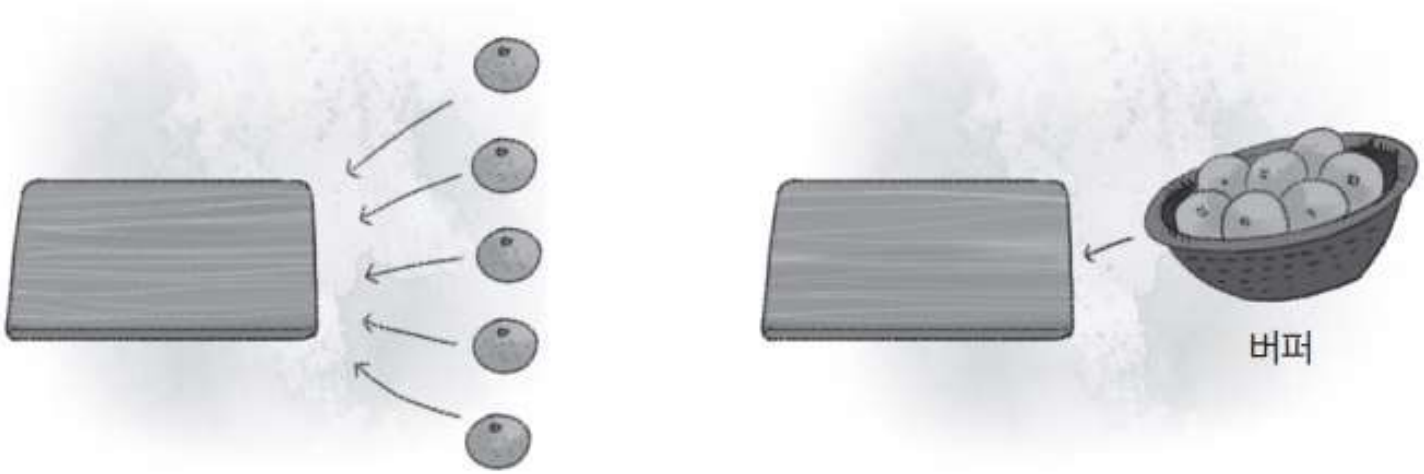


그림 2-15 버퍼의 개념

1 버퍼

스풀(spool)

- CPU와 입출력장치가 독립적으로 동작하도록 고안된 소프트웨어적인 버퍼
- [예] 스풀러
  - 인쇄할 내용을 순차적으로 출력하는 소프트웨어로 출력 명령을 내린 프로그램과 독립적으로 동작
  - 인쇄물이 완료될 때까지 다른 인쇄물이 끼어들 수 없으므로 프로그램 간에 배타적임

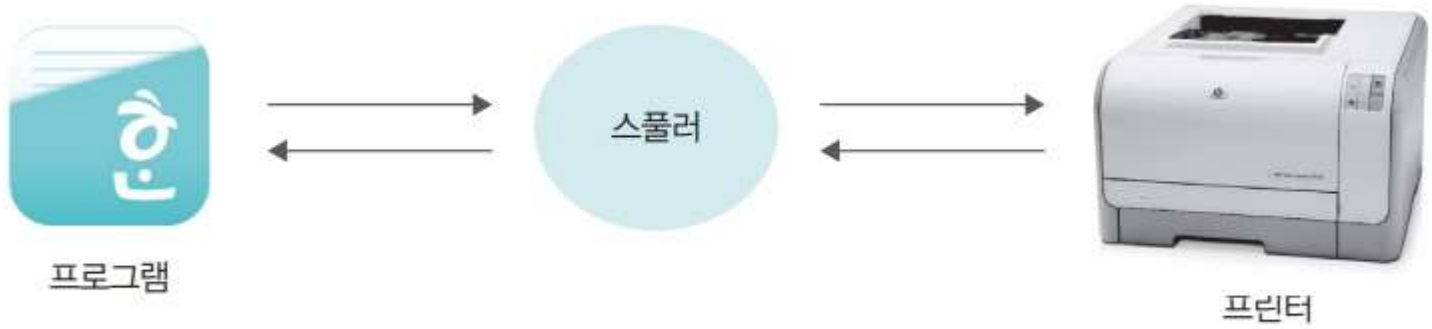


그림 2-16 스풀러의 동작 원리

2 캐시

캐시(cache)

- 메모리와 CPU 간의 속도 차이를 완화하기 위해 메모리의 데이터를 미리 가져와 저장해두는 임시 장소
- 필요한 데이터를 모아 한꺼번에 전달하는 버퍼의 일종으로 CPU가 앞으로 사용할 것으로 예상되는 데이터를 미리 가져다 놓음(**prefetch**)
- CPU는 메모리에 접근해야 할 때 캐시를 먼저 방문하여 원하는 데이터가 있는지 찾아봄

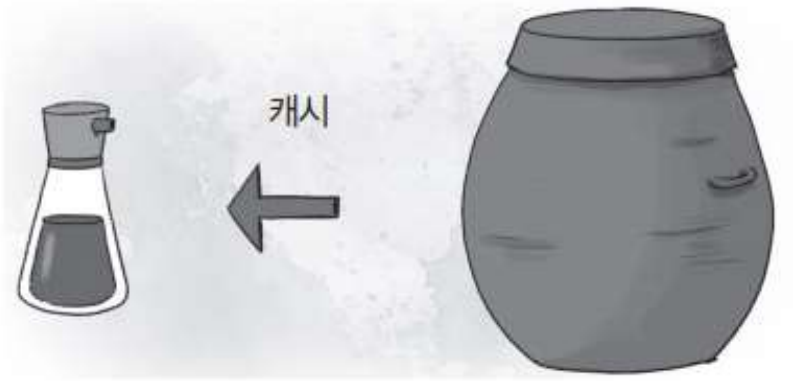


그림 2-17 캐시의 개념

2 캐시

캐시의 구조

- 캐시 히트cache hit : 캐시에서 원하는 데이터를 찾는 것으로, 그 데이터를 바로 사용
- 캐시 미스cache miss : 원하는 데이터가 캐시에 없으면 메모리로 가서 데이터를 찾음
- 캐시 적중률cache hit ratio
  - 캐시 히트가 되는 비율로, 일반적인 컴퓨터의 캐시 적중률은 약 90%

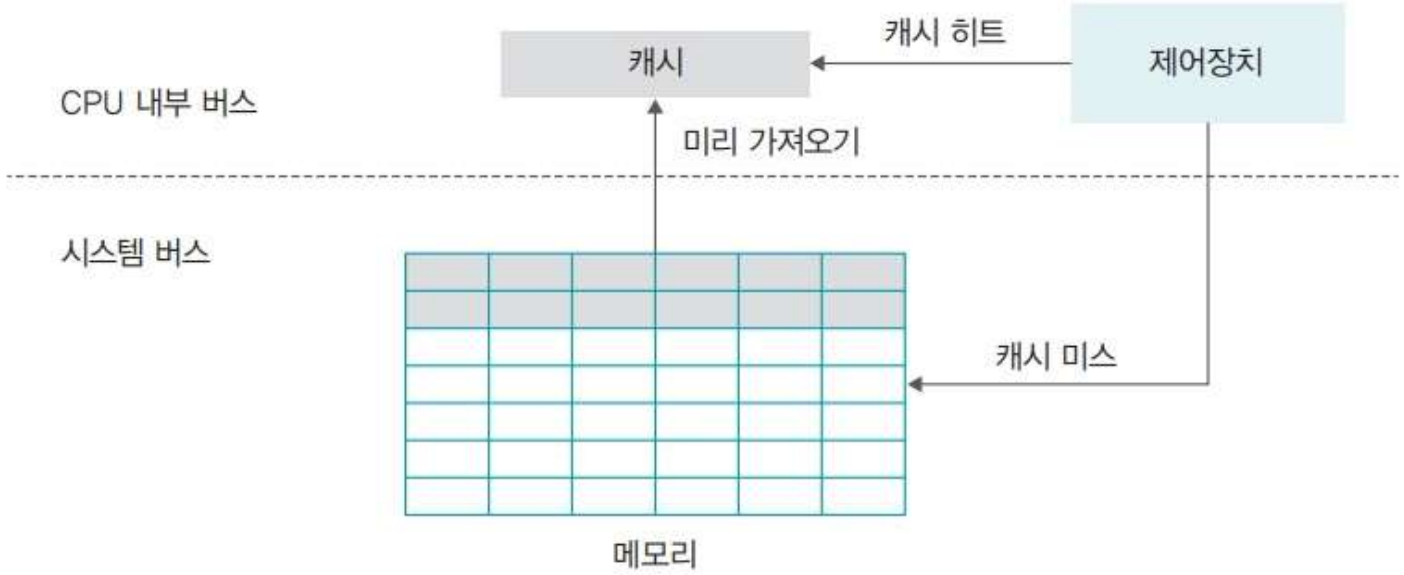


그림 2-18 캐시의 구조

2 캐시

즉시 쓰기write through

- 캐시에 있는 데이터가 변경되면 이를 즉시 메모리에 반영하는 방식
- 메모리와의 빈번한 데이터 전송으로 인해 성능이 느려짐
- 메모리의 최신 값이 항상 유지되기 때문에 급작스러운 정전에도 데이터를 잃어버리지 않음

지연 쓰기write back

- 캐시에 있는 데이터가 변경되면 이를 즉시 메모리에 반영하는 것이 아니라 변경된 내용을 모아서 주기적으로 반영하는 방식
- 카피백copy back이라고도 함
- 메모리와의 데이터 전송 횟수가 줄어들어 시스템의 성능을 향상할 수 있음
- 메모리와 캐시된 데이터 사이의 불일치가 발생할 수도 있음

2 캐시

L1 캐시와 L2 캐시

- 캐시는 명령어와 데이터의 구분 없이 모든 자료를 가져오는 일반 캐시, 명령어와 데이터를 구분하여 가져오는 특수 캐시로 구분
- 일반 캐시
  - 메모리와 연결되기 때문에 L2(Level 2) 캐시라고 부름
- 특수 캐시
  - CPU 레지스터에 직접 연결되기 때문에 L1(Level 1) 캐시라고 부름

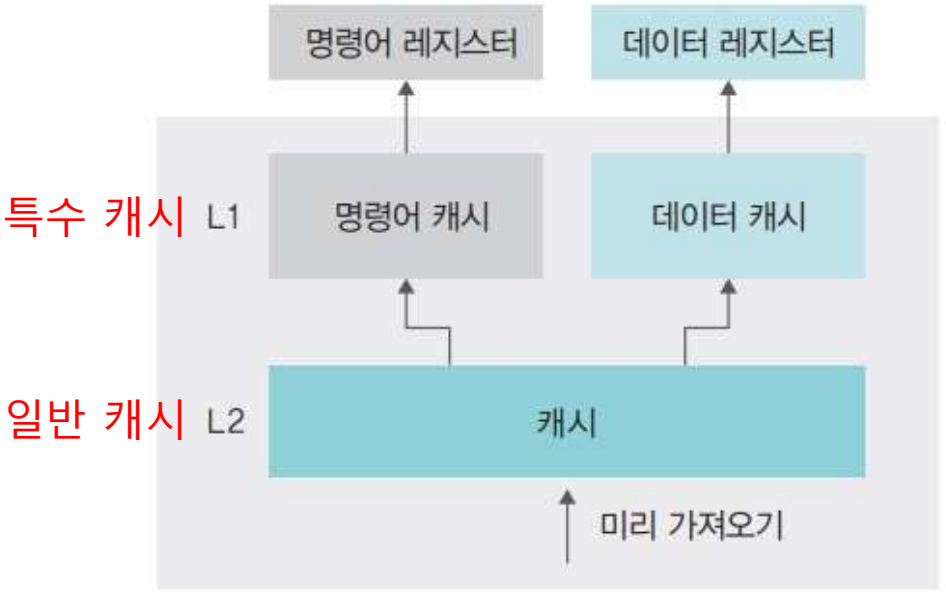


그림 2-19 레벨별 캐시의 구조

3 저장장치의 계층 구조

저장장치의 계층 구조

- 개념
  - 속도가 빠르고 값이 비싼 저장장치를 CPU 가까운 쪽에 두고, 값이 싸고 용량이 큰 저장장치를 반대쪽에 배치하여 적당한 가격으로 빠른 속도와 큰 용량을 동시에 얻는 방

- 이점
  - CPU와 가까
    수 있음
  - 메모리에서
    영구적으로

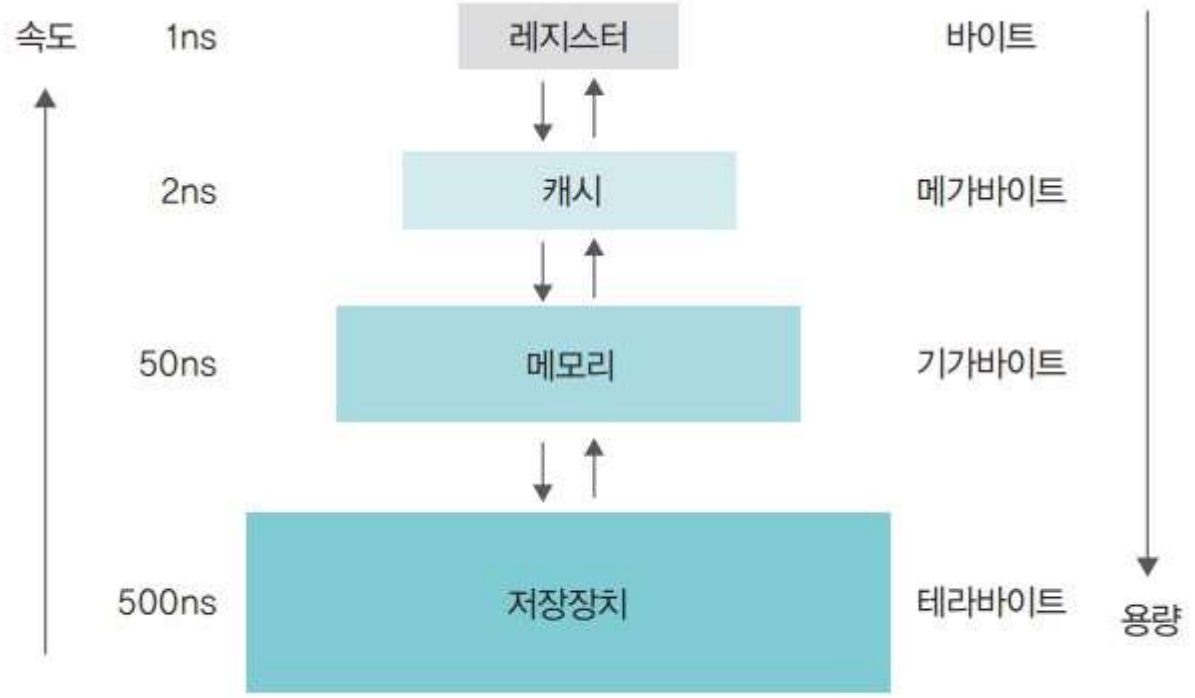


그림 2-20 저장장치의 계층 구조

4 인터럽트

폴링 방식polling

- **CPU가 직접 입출력장치에서 데이터를 가져오거나 내보내는 방식**
- CPU가 입출력장치의 상태를 주기적으로 검사하여 일정한 조건을 만족할 때 데이터를 처리
- CPU가 명령어 해석과 실행이라는 본래 역할 외에 모든 입출력까지 관여해야 하므로 작업 효율이 떨어짐

그림 2-21 CPU가 주변장치를 직접 관리하는 폴링 방식

4 인터럽트

인터럽트 방식interrupt

- **입출력 관리자가 대신 입출력을 해주는 방식**
- CPU의 작업과 저장장치의 데이터 이동을 독립적으로 운영함으로써 시스템의 효율을 높임
- 데이터의 입출력이 이루어지는 동안 CPU가 다른 작업을 할 수 있음

그림 2-22 입출력 관리자에게 데이터 전송을 지시하는 인터럽트 방식

4 인터럽트

인터럽트

- 입출력 관리자가 CPU에 보내는 완료 신호

인터럽트 번호interrupt number

- 많은 주변장치 중 어떤 것의 작업이 끝났는지를 CPU에 알려주기 위해 사용하는 번호
- IRQ(Interrupt ReQuest) : 키보드 IRQ 1, 마우스 IRQ 12, HDD IRQ 14 등

인터럽트 벡터interrupt vector

- 여러 개의 입출력 작업을 관리하기 위해 여러 개의 인터럽트를 하나의 배열로 만든 것

4 인터럽트

인터럽트 방식의 동작 과정

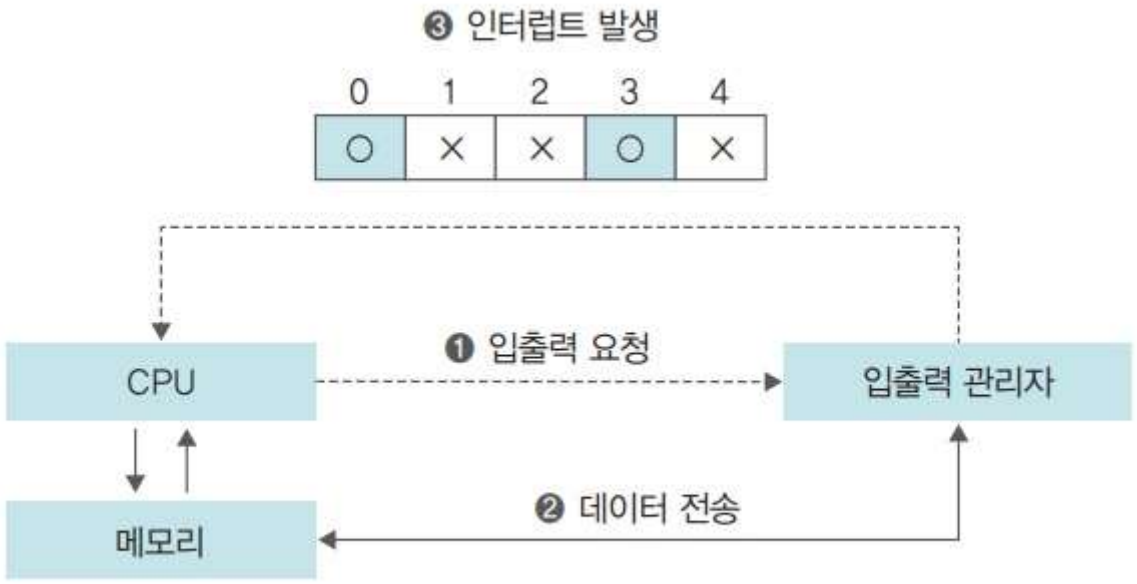


그림 2-23 인터럽트 처리 과정

❶ CPU가 입출력 관리자에게 입출력 명령을 보낸다.

❷ 입출력 관리자는 명령받은 데이터를 메모리에 가져다놓거나 메모리에 있는 데이터를 저장장치로 옮긴다.

❸ 데이터 전송이 완료되면 입출력 관리자는 완료 신호를 CPU에 보낸다.

4 인터럽트

직접 메모리 접근Direct Memory Access, DMA

- 입출력 관리자에게는 필요한 CPU의 허락 없이 메모리에 접근할 수 있는 권한
- 데이터 전송을 지시받은 입출력 관리자는 직접 메모리 접근 권한이 있어야 CPU의 관여 없이 작업을 완료할 수 있음

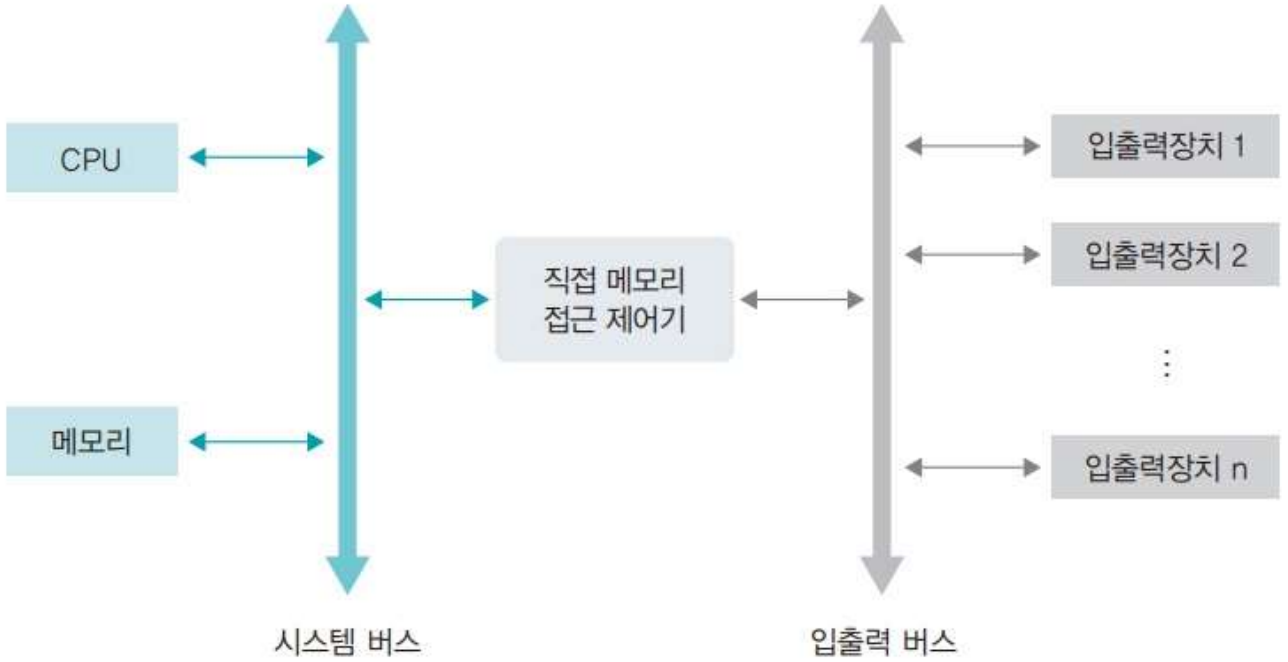


그림 2-24 직접 메모리 접근 제어기의 구조

4 인터럽트

메모리 매핑 입출력 Memory Mapped I/O, MMIO

- 메모리의 일정 공간을 입출력에 할당하는 기법

사이클 훔치기 Cycle Stealing

- 입출력 채널(**I/O channel**) : CPU 대신 입출력 동작을 수행하는 장치
- CPU와 직접 메모리 접근이 동시에 메모리에 접근하면 보통 CPU가 메모리 사용 권한을 양보
- CPU의 작업 속도보다 입출력장치의 속도가 느리기 때문에 직접 메모리 접근에 양보하는 것으로, 이러한 상황을 사이클 훔치기라고 함

그림 2-25 **메모리 매핑 입출력**

1 병렬 처리의 개념

병렬 처리parallel processing

- 동시에 여러 개의 명령을 처리하여 작업의 능률을 올리는 방식

볶음밥 조리 예

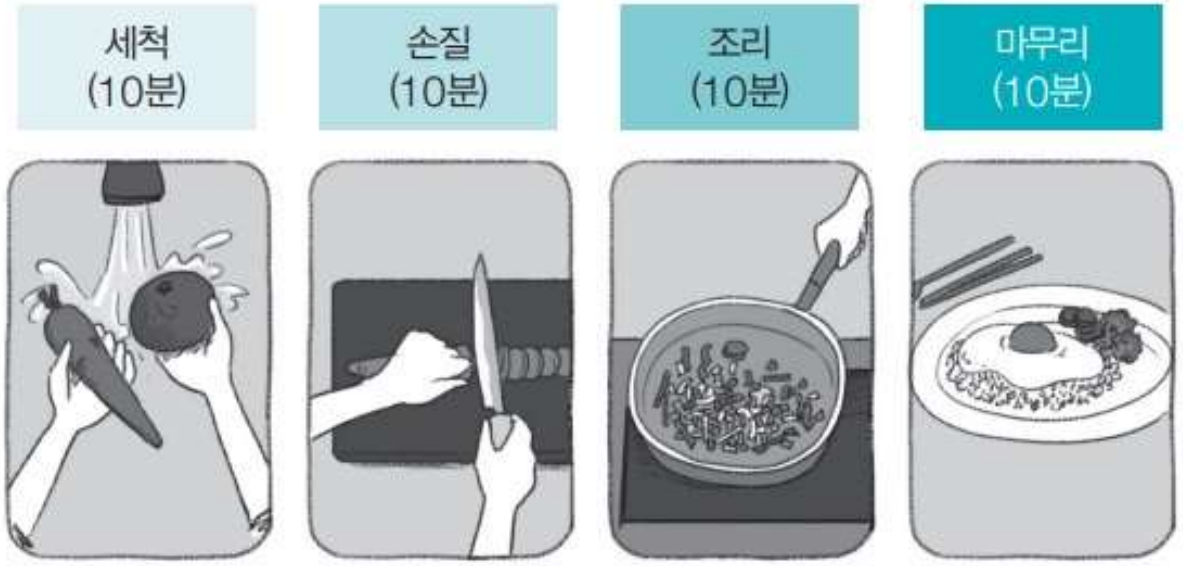


그림 2-26 볶음밥 조리 과정

1 병렬 처리의 개념

볶음밥 조리의 병렬 처리

그림 2-27 볶음밥 조리의 병렬 처리

파이프라인 기법

- 하나의 코어에 여러 개의 스레드thread를 이용하는 방식

1 병렬 처리의 개념

슈퍼스칼라 기법

- 듀얼코어 CPU를 이용해 2개의 작업을 동시에 처리하는 방식

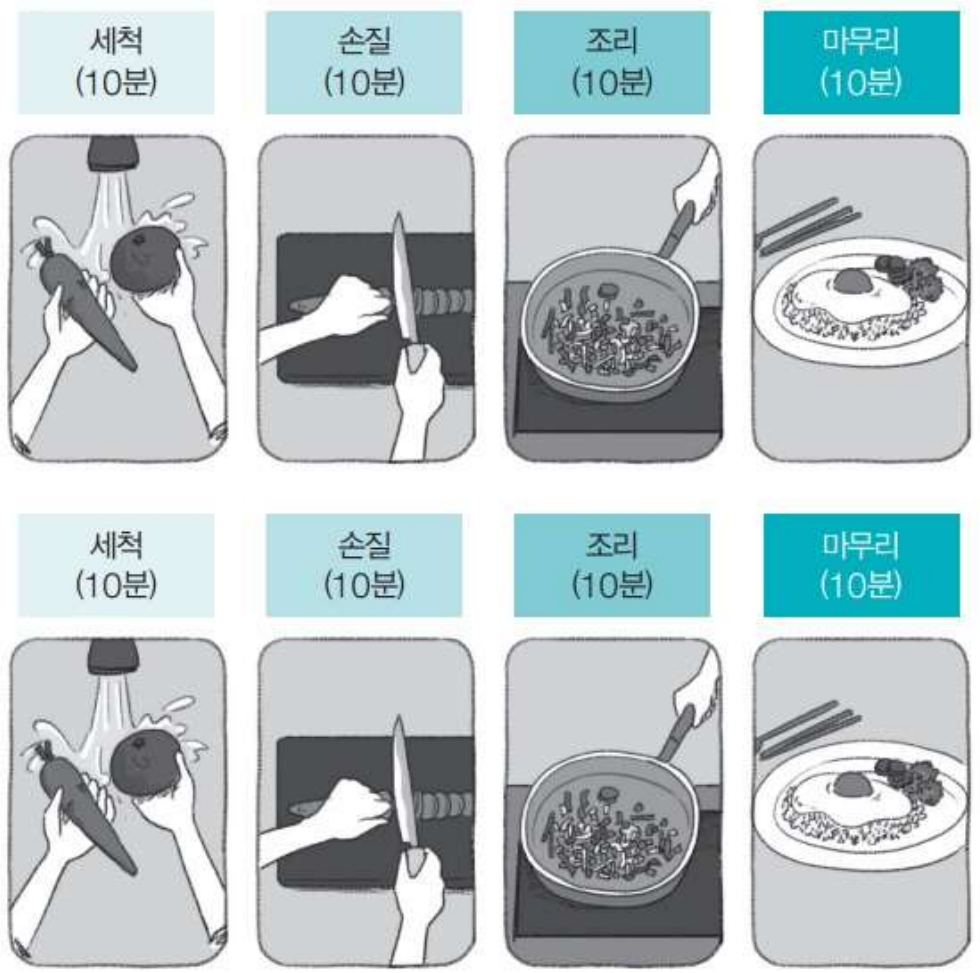


그림 2-28 조리 시설이 2개인 경우의 병렬 처리

2 병렬 처리 시 고려 사항

상호 의존성이 없어야 병렬 처리가 가능

- 각 명령이 서로 독립적이고 앞의 결과가 뒤의 명령에 영향을 미치지 않아야 함

각 단계의 시간을 일정하게 맞춰야 병렬 처리가 원만하게 이루어짐

- 오랜 시간이 걸리는 작업 때문에 진행이 전반적으로 밀려서 전체 작업 시간이 늘어나므로 단계별 시간의 차이가 크면 병렬 처리의 효과가 떨어짐

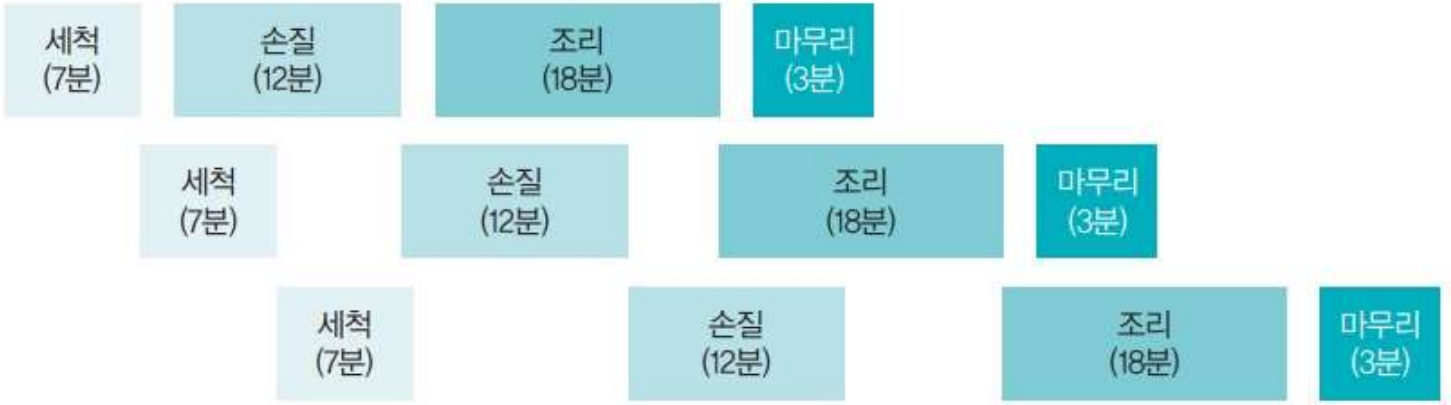


그림 2-29 단계별 시간이 다른 경우의 병렬 처리

2 병렬 처리 시 고려 사항

전체 작업 시간을 몇 단계로 나눌지 잘 따져보아야 함

- 병렬 처리의 깊이 N은 동시에 처리할 수 있는 작업의 개수를 의미
- 이론적으로는 N이 커질수록 동시에 작업할 수 있는 작업의 개수가 많아져서 성능이 높아지지만, 작업을 너무 많이 나누면 각 단계마다 작업을 이동하고 새로운 작업을 불러오는 데 시간이 너무 많이 걸려서 오히려 성능이 떨어짐

그림 2-30 병렬 처리의 깊이가 2인 경우

3 병렬 처리 기법

CPU에서 명령어가 실행되는 과정

❶ 명령어 패치(IF) : 다음에 실행할 명령어를 명령어 레지스터에 저장

❷ 명령어 해석(ID) : 명령어 해석

❸ 실행(EX) : 해석한 결과를 토대로 명령어 실행

❹ 쓰기(WB) : 실행된 결과를 메모리에 저장

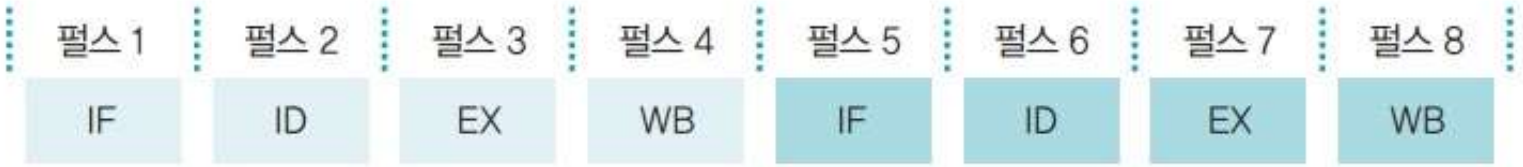


그림 2-31 명령어 처리 단계

3 병렬 처리 기법

파이프라인 기법

- CPU의 사용을 극대화하기 위해 명령을 겹쳐서 실행하는 방법

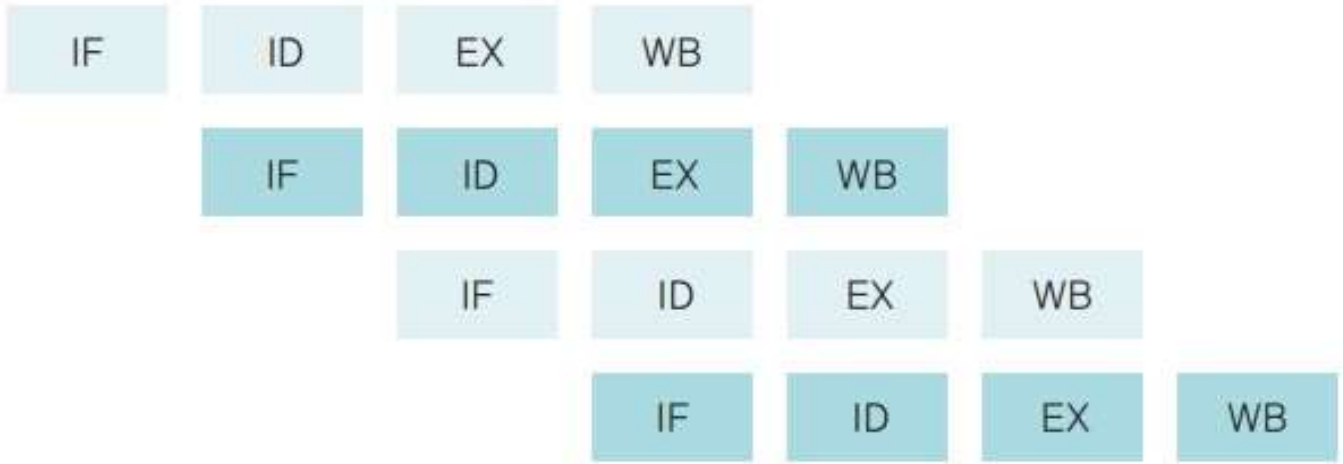


그림 2-32 파이프라인 기법의 동작 과정

3 병렬 처리 기법

파이프라인의 위험

- 데이터 위험Data hazard
  - 데이터의 의존성 때문에 발생하는 문제
  - 데이터 A를 필요로 하는 두 번째 명령어는 앞의 명령어가 끝날 때까지 동시에 실행되어서는 안 됨
  - 데이터 위험은 파이프라인의 명령어 단계를 지연하여 해결

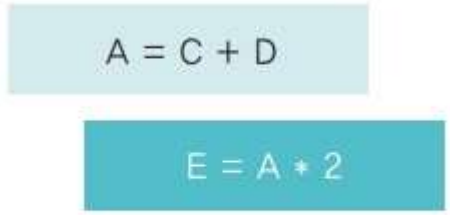


그림 2-33 데이터 위험의 예

3 병렬 처리 기법

파이프라인의 위험

- 제어 위험control hazard
  - 프로그램 카운터 값을 갑자기 변화시켜 발생하는 위험
  - 첫 명령어를 실행하고 보니 goto 문이어서 다음 문장이 아니라 다른 문장으로 이동하게 되면 현재 동시에 처리되고 있는 명령어들이 쓸모없어짐
  - 제어 위험은 분기 예측이나 분기 지연 방법으로 해결

그림 2-34 제어 위험의 예

3 병렬 처리 기법

파이프라인의 위험

- 구조 위험structural hazard
  - 서로 다른 명령어가 같은 자원에 접근하려 할 때 발생하는 문제
  - 명령어 A가 레지스터 RX를 사용하고 있는데 병렬 처리되는 명령어 B도 레지스터 RX를 사용해야 한다면 서로 충돌
  - 구조 위험은 해결하기 어렵다고 알려져 있음

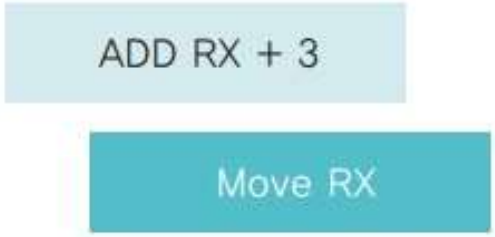


그림 2-35 구조 위험의 예

3 병렬 처리 기법

슈퍼스칼라 기법super-scalar

- 파이프라인을 처리할 수 있는 코어를 여러 개 구성하여 복수의 명령어가 동시에 실행되도록 하는 방식
- 대부분은 파이프라인 기법과 동일하지만 코어를 2개 구성하여 각 단계에서 동시에 실행되는 명령어가 2개라는 점이 다름

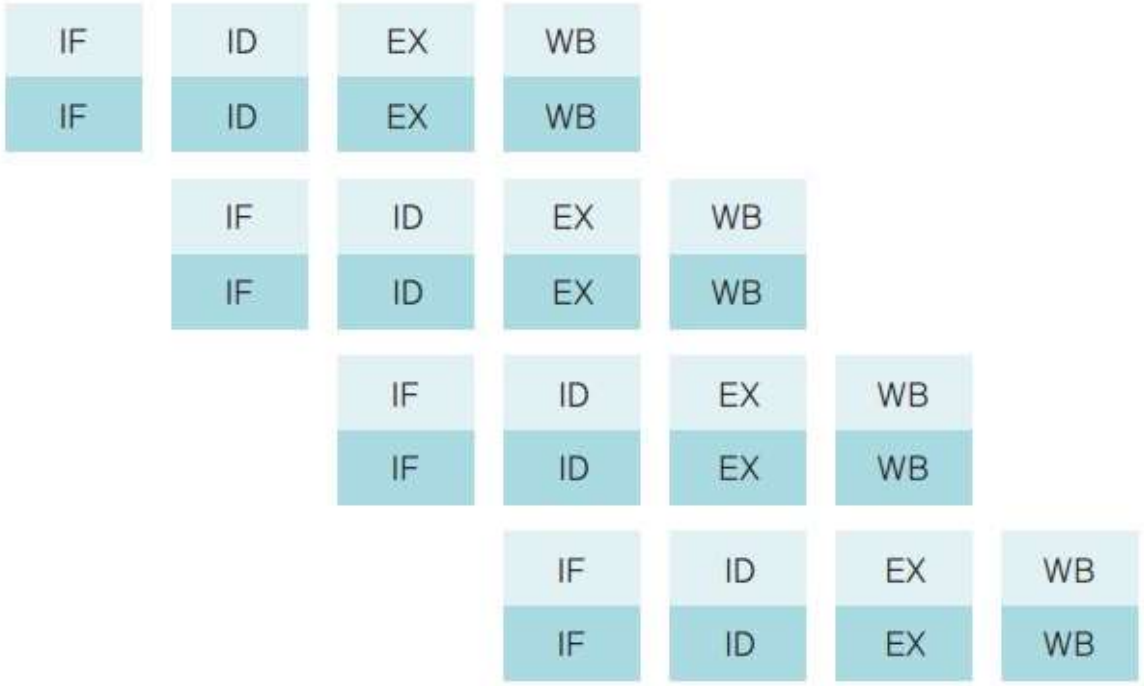


그림 2-36 슈퍼스칼라 기법의 동작 과정

3 병렬 처리 기법

슈퍼파이프라인 기법super-pipeline

- 파이프라인의 각 단계를 세분하여 한 클록 내에 여러 명령어를 처리
- 한 클록 내에 여러 명령어를 실행하면 다음 명령어가 빠른 시간 안에 시작될 수 있어 병렬 처리 능력이 높아짐

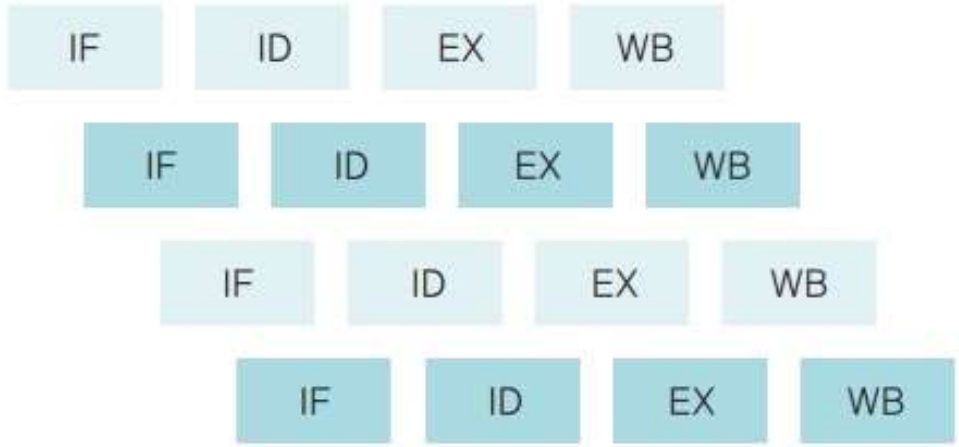


그림 2-37 슈퍼파이프라인 기법의 동작 과정

3 병렬 처리 기법

슈퍼파이프라인 슈퍼스칼라 기법 super-pipelined super-scalar

- 슈퍼파이프라인 기법을 여러 개의 코어에서 동시에 수행하는 방식

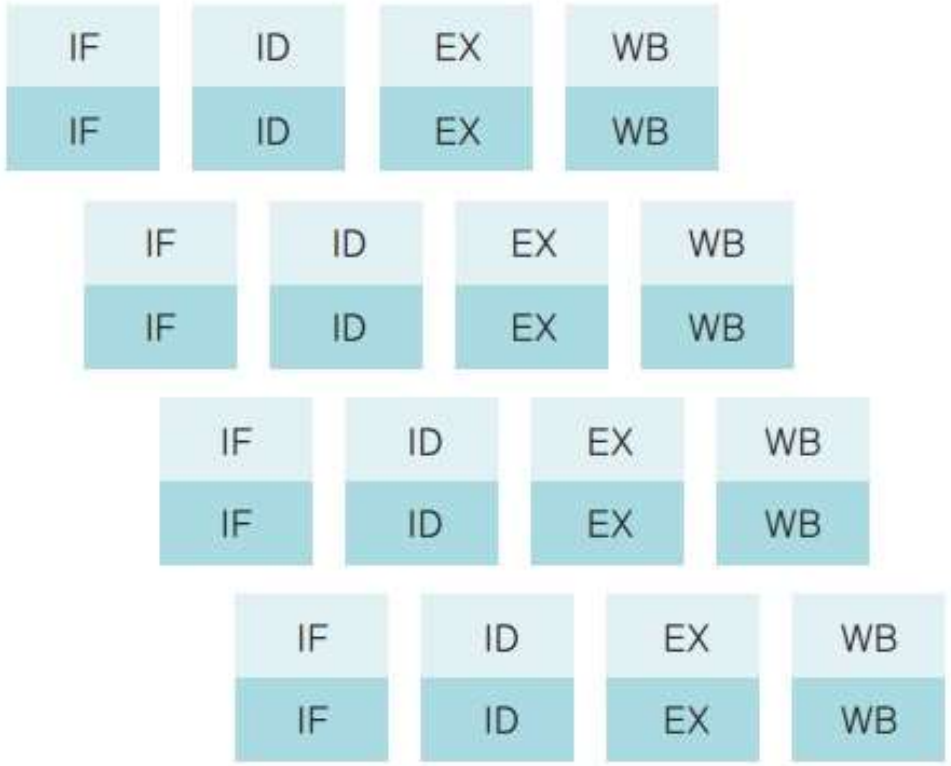


그림 2-38 슈퍼파이프라인 슈퍼스칼라 기법의 동작 과정

3 병렬 처리 기법

VLIW 기법 Very Long Instruction Word

- CPU가 병렬 처리를 지원하지 않을 경우 소프트웨어적으로 병렬 처리를 하는 방법
- 동시에 수행할 수 있는 명령어들을 컴파일러가 추출하고 하나의 명령어로 압축하여 실행
- CPU가 병렬 처리를 지원하지 않을 때 사용하는 방법이므로 앞의 병렬 처리 기법들에 비해 동시에 처리하는 명령어의 개수가 적음
- 컴파일 시 병렬 처리가 이루어짐

1 무어의 법칙과 암달의 법칙

무어의 법칙Moore's law

- **CPU의 속도가 24개월마다 2배 빨라진다**
- 초기의 CPU에만 적용되며 지금은 그렇지 않음

암달의 법칙Amdahl's law

- 컴퓨터 시스템의 일부를 개선할 때 전체 시스템에 미치는 영향과의 관계를 수식으로 나타낸 법칙
- 이 법칙에 따르면 **주변장치의 향상 없이 CPU의 속도를 2GHz에서 4GHz로 늘리더라도 컴퓨터의 성능이 2배 빨라지지 않음**

Homework

2장 연습문제

- P. 123 연습문제(단답형)
- P. 125 심화문제(서술형)
- 제출기한 : 9월 27일(일) 23시
- 제출방법 : hwp/word로 작성한 후 pdf로 변환하여 eclass 과제함
- 파일이름 : **학번(이름)-2장**.pdf

Thank You