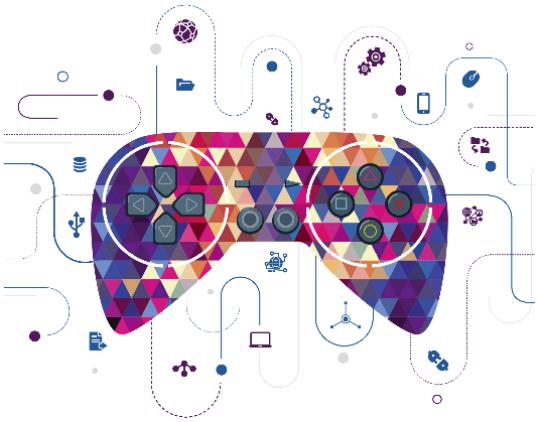


그림으로 배우는 구조와 원리

운영체제 개정 3판

Chatpter 01
컴퓨터 시스템의 소개

01 컴퓨터 하드웨어의 구성

02 컴퓨터 시스템의 동작

성적 : 중간고사 +기말고사 + 출석 + 레포트
E-mail : hkjung@pcu.ac.kr(010-5430-9099)

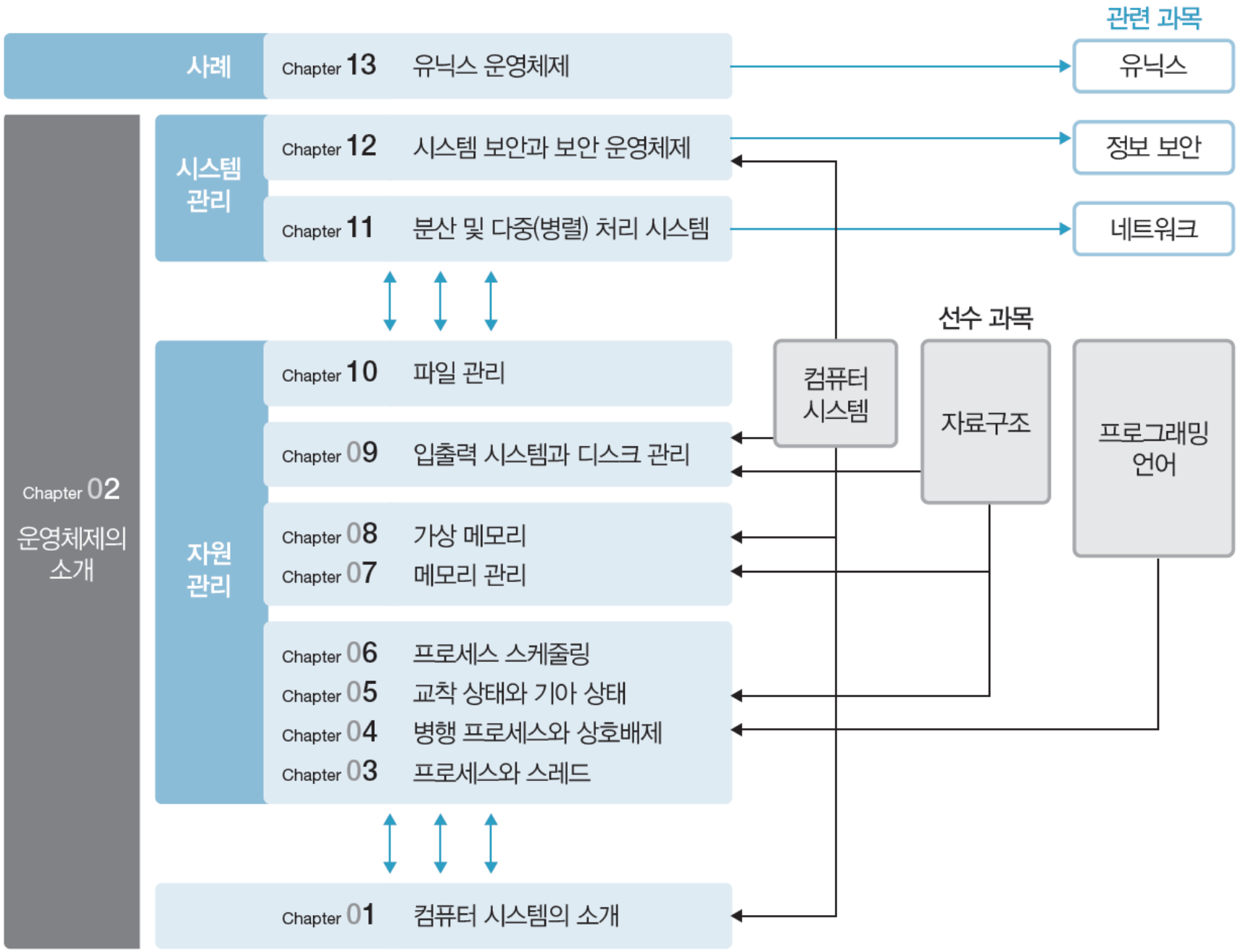
장별 요약

- **1~2장 : 컴퓨터 시스템과 운영체제 개요**
  - 프로세스를 중심으로 컴퓨터 시스템의 전반적인 요소를 다루며, 운영체제의 개념과 역할, 기능 등을 정리
- **3~6장 : 프로세스 관리**
  - 프로세스를 주제로 프로세스의 상태와 변환 관련 기술과 제어, 스레드, 병행 프로세스(상호배제 및 동기화), 교착 상태 등을 소개한다. 이어서 프로세스 스케줄링도 설명
- **7~8장 : 메모리 관리**
  - 메모리 관리 전략, 메모리 할당 방법, 가상 메모리의 개념과 요구 페이징, 페이지 대치 알고리즘을 설명
- **9~10장 : 장치 관리 및 파일 관리**
  - 입출력 관리와 주변장치(디스크)의 공간 할당, 디스크 스케줄링을 설명한다. 아울러 파일 관리 시스템의 구성, 디렉터리, 파일의 보조기억장치 할당을 설명
- **11~12장 : 분산 및 보호**
  - 분산 운영체제와 다중 처리, 클라이언트/서버, 클러스터를 살펴보고, 컴퓨터 보안과 보호, 신뢰 시스템을 설명
- **13장 : 유닉스 운영체제**
  - 운영체제의 한 예인 유닉스의 구조, 설계 목표를 설명

강의 계획표

주	해당 장/주제	주제
1	1장	과목 소개, 컴퓨터 시스템의 소개
2	2장	운영체제의 소개
3	3장	프로세스와 스레드
4	4장	병행 프로세스와 상호배제
5	5장	교착 상태와 기아 상태
6	6장	프로세스 스케줄링
7	7장	메모리 관리
8	필기	중간고사
9	8장	가상 메모리
10	9장	입출력 시스템과 디스크 관리
11	10장	파일 관리
12	11장	분산 및 다중(병렬) 처리 시스템
13	12장	시스템 보안과 보안 운영체제
14	13장	유닉스 운영체제
15	필기	기말고사

학습 로드맵


관련 과목
사례
Chapter 13 유닉스 운영체제
유닉스
시스템 관리
Chapter 12 시스템 보안과 보안 운영체제
정보 보안
Chapter 11 분산 및 다중(병렬) 처리 시스템
네트워크
선수 과목
Chapter 02 운영체제의 소개
자원 관리
Chapter 10 파일 관리
Chapter 09 입출력 시스템과 디스크 관리
Chapter 08 가상 메모리
Chapter 07 메모리 관리
Chapter 06 프로세스 스케줄링
Chapter 05 교착 상태와 기아 상태
Chapter 04 병행 프로세스와 상호배제
Chapter 03 프로세스와 스레드
Chapter 01 컴퓨터 시스템의 소개
컴퓨터 시스템
자료구조
프로그래밍 언어

- 컴퓨터 하드웨어를 구성하는 프로세서, 메모리, 주변장치, 시스템 버스
- 메모리의 역할과 계층 구조
- 컴퓨터 시스템의 동작 원리
- 프로그램의 기본 단위인 명령어의 구성과 그 실행 주기
- 인터럽트의 개념과 처리 과정

Section 01 컴퓨터 하드웨어의 구성

컴퓨터 시스템

- 데이터를 처리하는 물리적인 기계장치인 하드웨어hardware와 어떤 작업을 지시하는 명령어로 작성한 프로그램인 소프트웨어software로 구성

컴퓨터 하드웨어

- 하드웨어는 프로세서, 메모리(기억장치), 주변장치로 구성되고, 이들은 시스템 버스로 연결

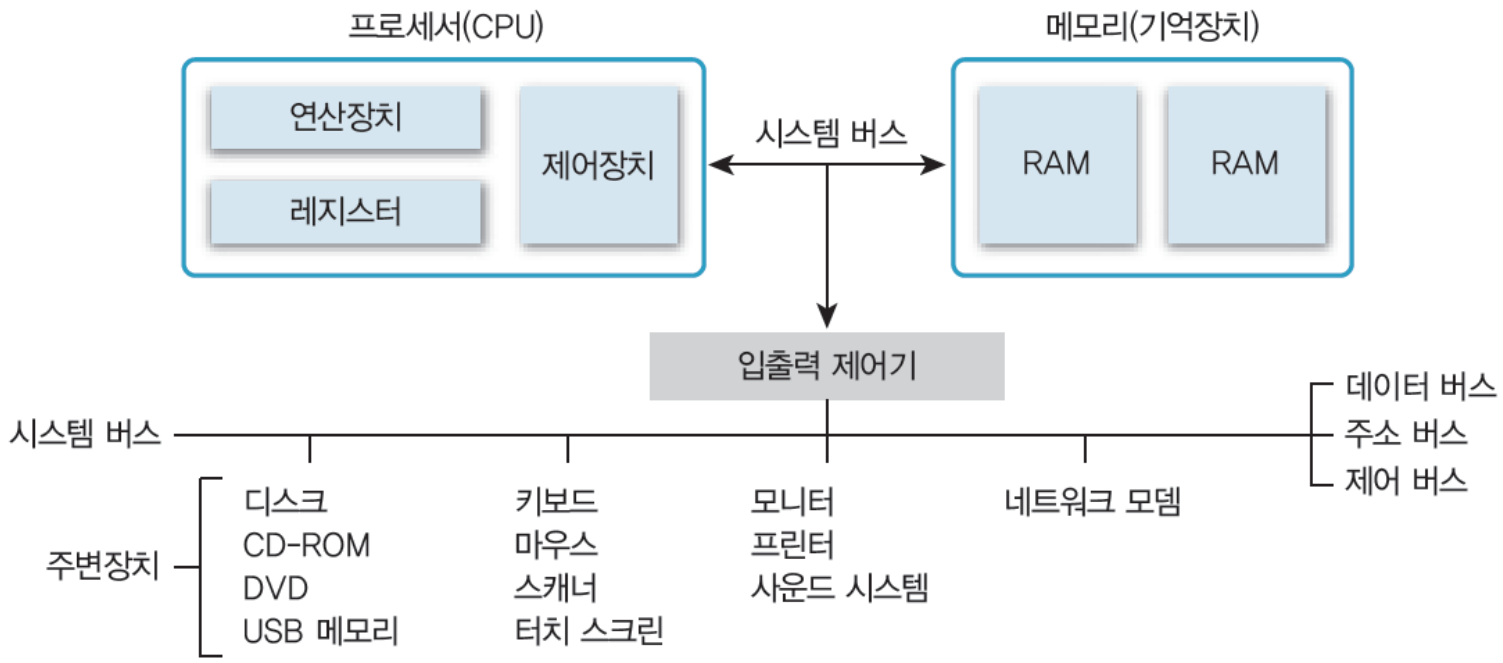


그림 1-1 컴퓨터 하드웨어의 구성

1. 프로세스

■ 프로세서CPU(중앙처리장치)

- 컴퓨터 하드웨어 구성 요소 중 운영체제와 가장 밀접한 부분으로, 컴퓨터의 모든 장치의 동작을 제어하고 연산 수행

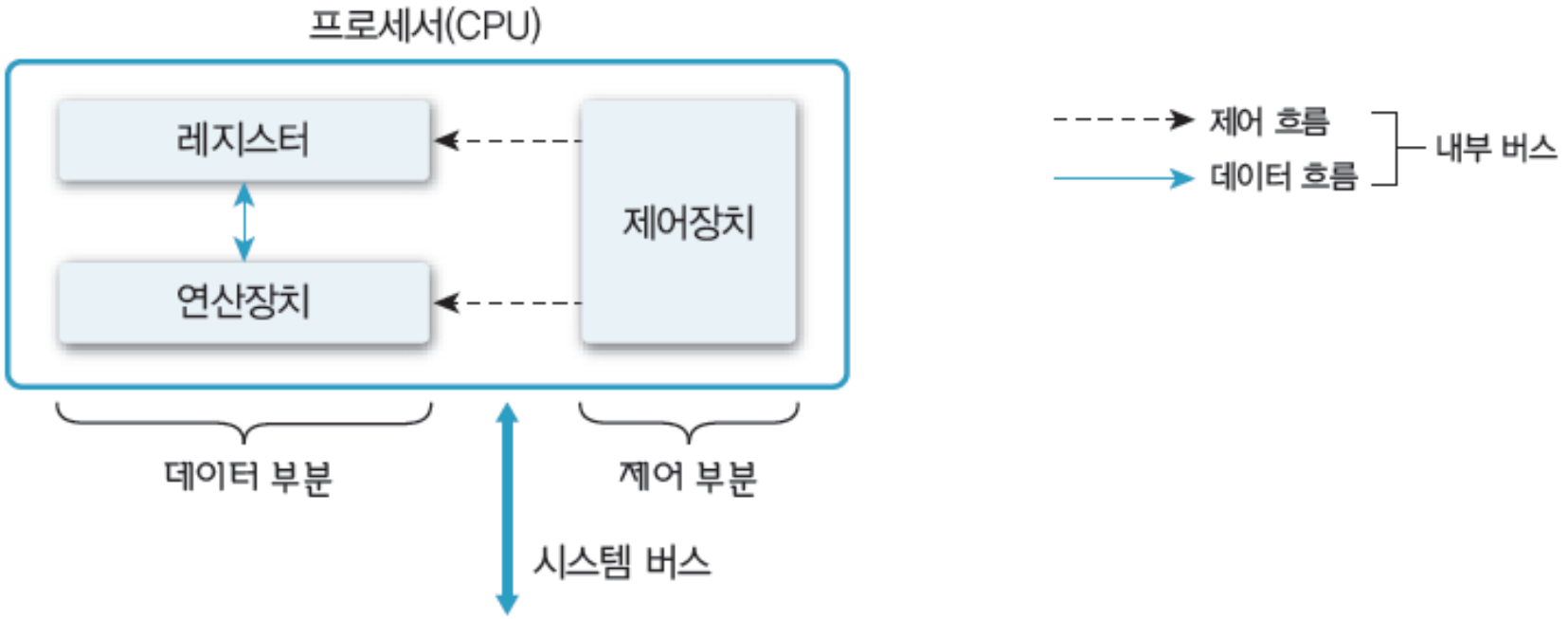


그림 1-2 프로세서의 구성

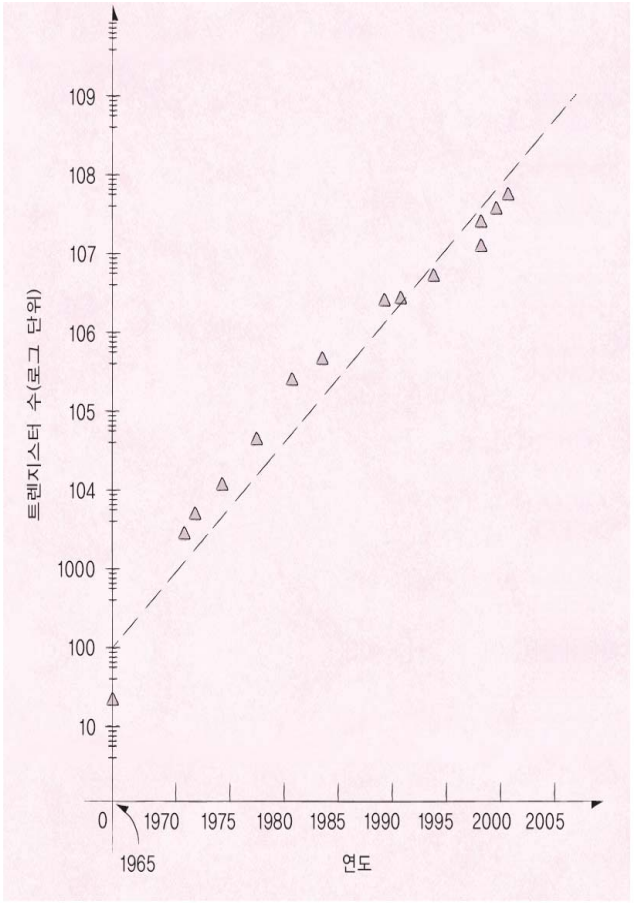
1. 프로세스

대부분의 운영체제는 하드웨어 구성에 의존하지 않게 구현

- 특정 장치를 대상으로 입출력 시 디바이스 드라이버 사용
  - ex) 플러그 앤 플레이 장치 - 컴퓨터에 연결 시 자신이 어떤 장치인지 운영체제에 알림

무어의 법칙Moore's law

- 인텔 프로세서의 연도별 트랜지스터 수
- 프로세서의 트랜지스터 수가 해마다 2배씩 증가하는 법칙
- 해마다 CPU의 속도가 24개월마다 2배 빨라진다는 내용
- 초기의 CPU에만 적용되며 지금은 그렇지 않음



암달의 법칙Amdahl's law

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

1. 프로세스

프로세스의 레지스터

- 용도에 따른 분류
  - 전용 레지스터
  - 범용 레지스터

- 사용자가 정보 변경 가능 여부에 따른 분류
  - 사용자 가시user-visible 레지스터
  - 사용자 불가시userinvisible 레지스터

- 저장하는 정보의 종류에 따른 분류
  - 데이터 레지스터
  - 주소 레지스터
  - 상태 레지스터

1. 프로세스

■ 사용자 가시 레지스터

- 사용자가 운영체제와 사용자 프로그램을 이용해 정보 변경 가능

표 1-1 사용자 가시 레지스터

종류	설명	
데이터 레지스터 DR, Data Register	함수 연산에 필요한 데이터를 저장한다. 값, 문자 등을 저장하므로 산술 연산이나 논리 연산에 사용하며, 연산 결과로 플래그 값을 저장한다.	
주소 레지스터 AR, Address Register	주소나 유효 주소를 계산하는 데 필요한 주소의 일부분을 저장한다. 주소 레지스터에 저장한 값(값 데이터)을 사용하여 산술 연산을 할 수 있다.	
	기준 주소 레지스터	프로그램을 실행할 때 사용하는 기준 주소 값을 저장한다. 기준 주소는 하나의 프로그램이나 일부처럼 서로 관련 있는 정보를 저장하며, 연속된 저장 공간을 지정하는 데 참조할 수 있는 주소이다. 따라서 기준 주소 레지스터는 페이지나 세그먼트처럼 블록화된 정보에 접근하는 데 사용한다.
	인덱스 레지스터	유효 주소를 계산하는 데 사용하는 주소 정보를 저장한다.
	스택 포인터 레지스터	메모리에 프로세서 스택을 구현하는 데 사용한다. 많은 프로세서와 주소 레지스터를 데이터 스택 포인터와 큐 포인터로 사용한다. 보통 반환 주소, 프로세서 상태 정보, 서브루틴의 임시 변수를 저장한다.

1. 프로세스

■ 사용자 불가시 레지스터

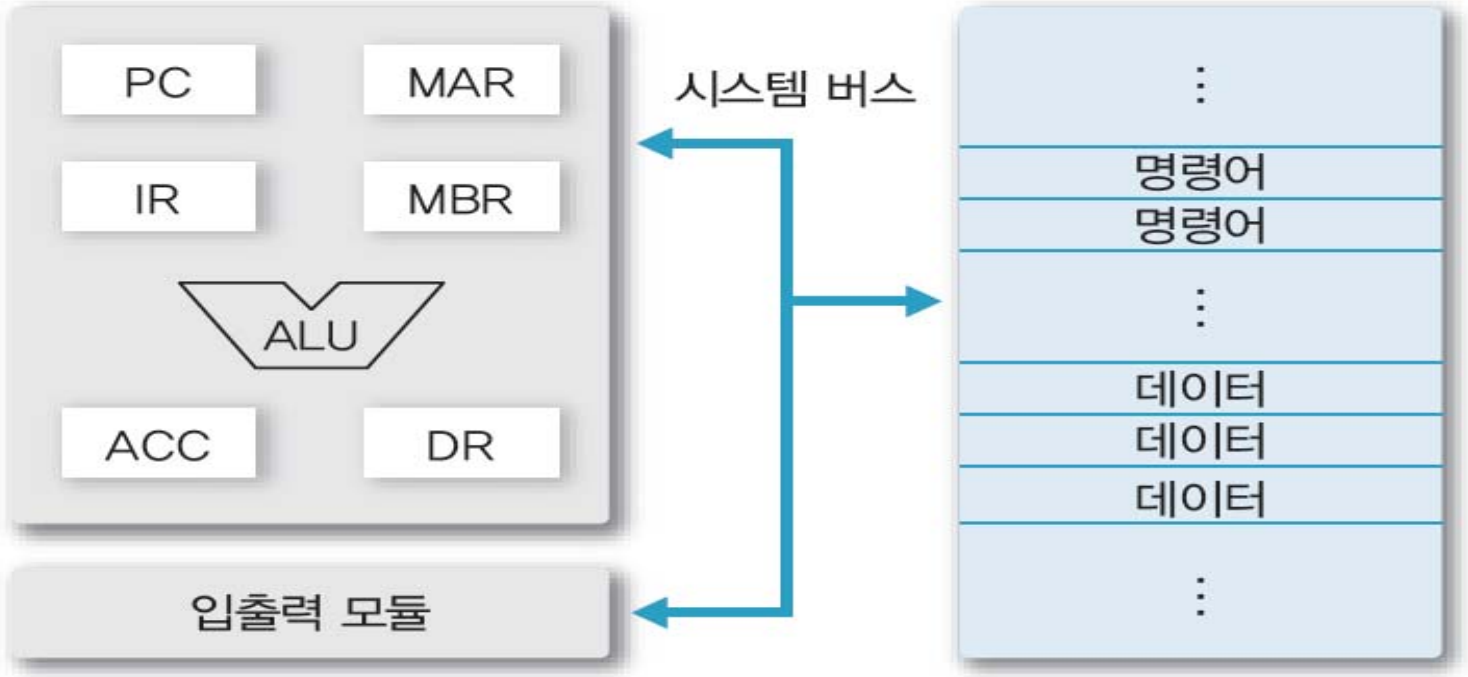
- 사용자가 정보를 변경할 수 없는 레지스터이다. 프로세서의 상태와 제어를 관리

표 1-2 사용자 불가시 레지스터

종류	설명
프로그램 카운터PC, Program Counter	다음에 실행할 명령어의 주소를 보관하는 레지스터이다. 계수기로 되어 있어 실행할 명령어를 메모리에서 읽으면 명령어의 길이만큼 증가하여 다음 명령어를 가리키며, 분기 명령어는 목적 주소로 갱신할 수 있다.
명령어 레지스터IR, Instruction Register	현재 실행하는 명령어를 보관하는 레지스터이다.
누산기ACC, ACCumulator	데이터를 일시적으로 저장하는 레지스터이다.
메모리 주소 레지스터MAR, Memory Address Register	프로세서가 참조하려는 데이터의 주소를 명시하여 메모리에 접근하는 버퍼 레지스터이다.
메모리 버퍼 레지스터MBR, Memory Buffer Register	프로세서가 메모리에서 읽거나 메모리에 저장할 데이터 자체를 보관하는 버퍼 레지스터이다. 메모리 데이터 레지스터MDR, Memory Data Register라고도 한다.

1. 프로세스

■ 프로세스의 기본 레지스터



- ALU Arithmetic Logic Unit : 산술 · 논리 연산장치

그림 1-3 프로세서의 기본 레지스터

2. 메모리

■ 메모리 계층 구조

- 1950~1960년대 너무 비싼 메인 메모리의 가격 문제 때문에 제안한 방법
- 메모리를 계층적으로 구성하여 비용, 속도, 용량, 접근시간 등을 상호 보완

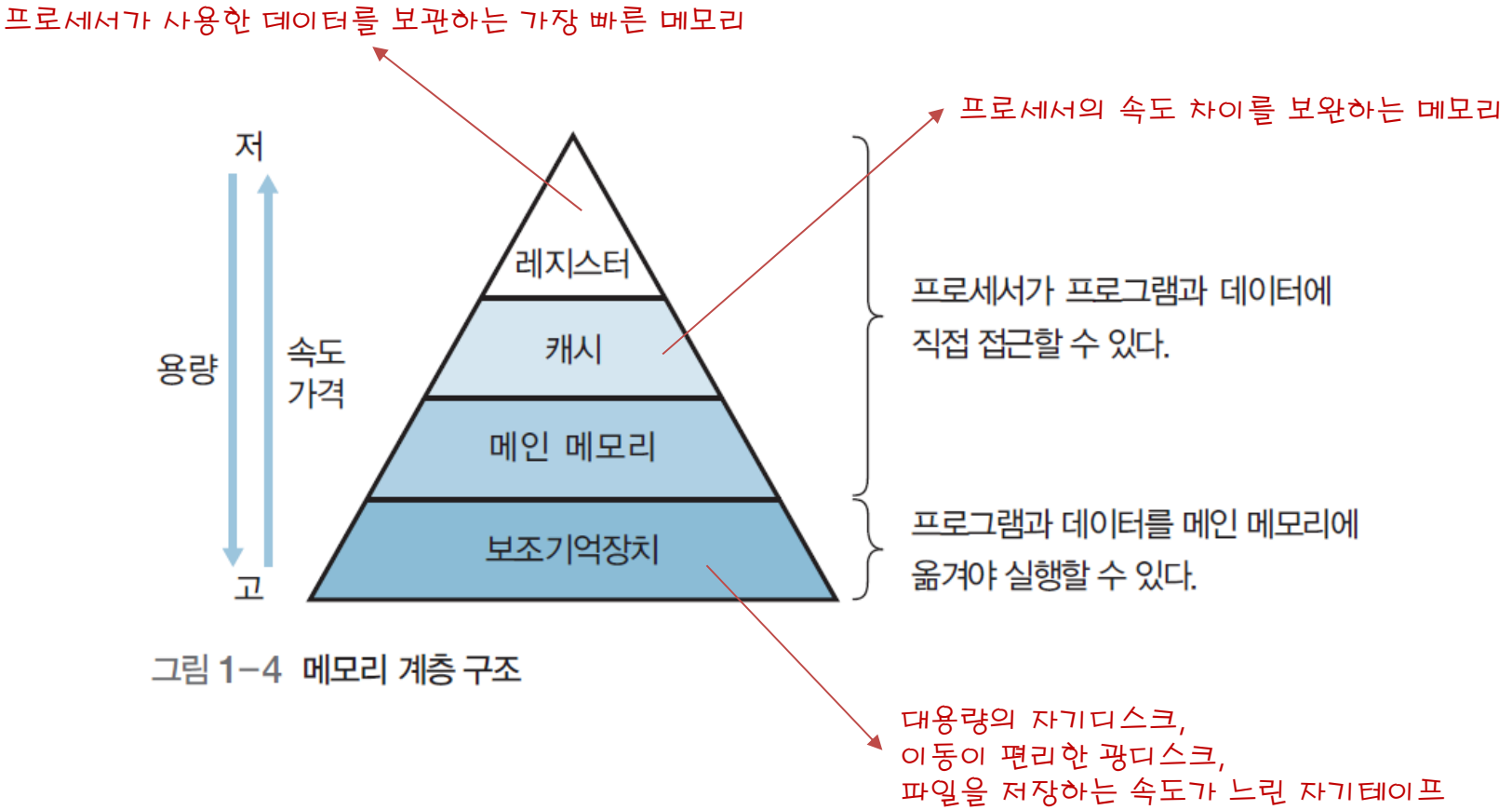


그림 1-4 메모리 계층 구조

2. 메모리

레지스터

- 프로세서 내부에 있으며, 프로세서가 사용할 데이터를 보관하는 가장 빠른 메모리

메인 메모리

- 프로세서 외부에 있으면서 프로세서에서 수행할 프로그램과 데이터를 저장하거나 프로세서에서 처리한 결과 저장
- 주기억장치 또는 1차 기억장치라고도 한다. 저장 밀도가 높고 가격이 싼 DRAM(Dynamic RAM)을 많이 사용

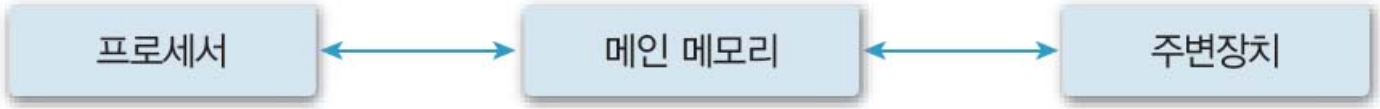


그림 1-5 메인 메모리의 역할 1

2. 메모리

메인 메모리

- 다수의 셀cell로 구성되며, 각 셀은 비트로 구성
- 셀이 K비트이면 셀에 2^K 값 저장 가능
- 메인 메모리에 데이터를 저장할 때는 셀 한 개나 여러 개에 나눠서 저장
- 셀은 주소로 참조하는데, n비트이라면 주소 범위는 $0 \sim 2^{n-1}$

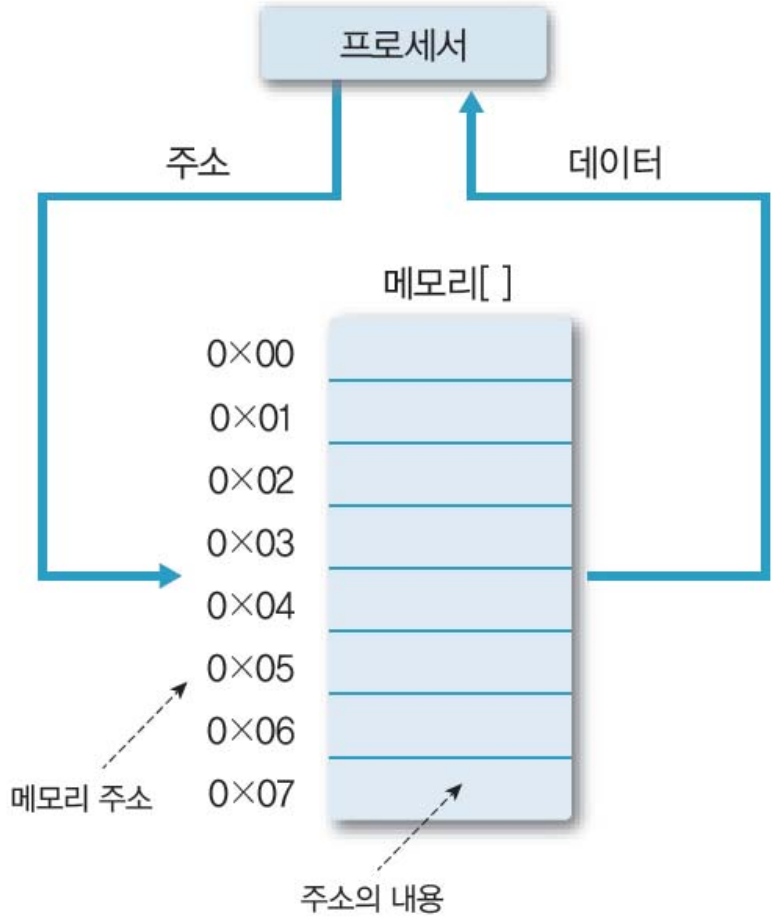


그림 1-6 메인 메모리의 주소 지정

2. 메모리

■ 메인 메모리

- 프로세서와 보조기억장치 사이에 있으며, 여기서 발생하는 디스크 입출력 병목 현상을 해결하는 역할 수행
- 프로세서와 메인 메모리 간에 속도 차이의 부담을 줄이려고 프로세서 내부나 외부에 캐시를 구현하기도 함

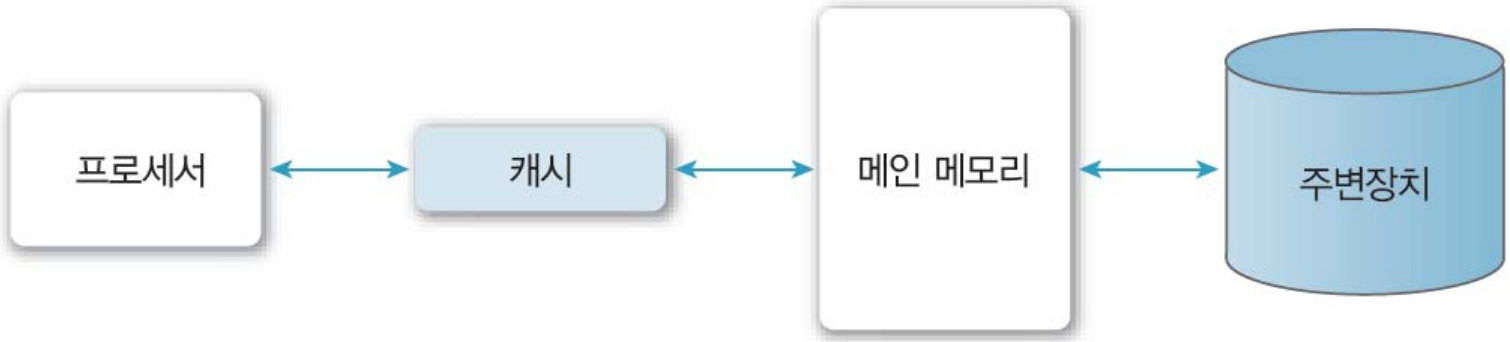


그림 1-9 메인 메모리의 역할 2 : 캐시 추가

2. 메모리

■ 메인 메모리

▪ 메모리 매핑mapping : 사상

- 컴파일로 논리적 주소를 물리적 주소로 변환하는 과정

그림 1-7 메모리 매핑

▪ 메모리 속도

- 메모리 접근시간과 메모리 사이클 시간으로 표현

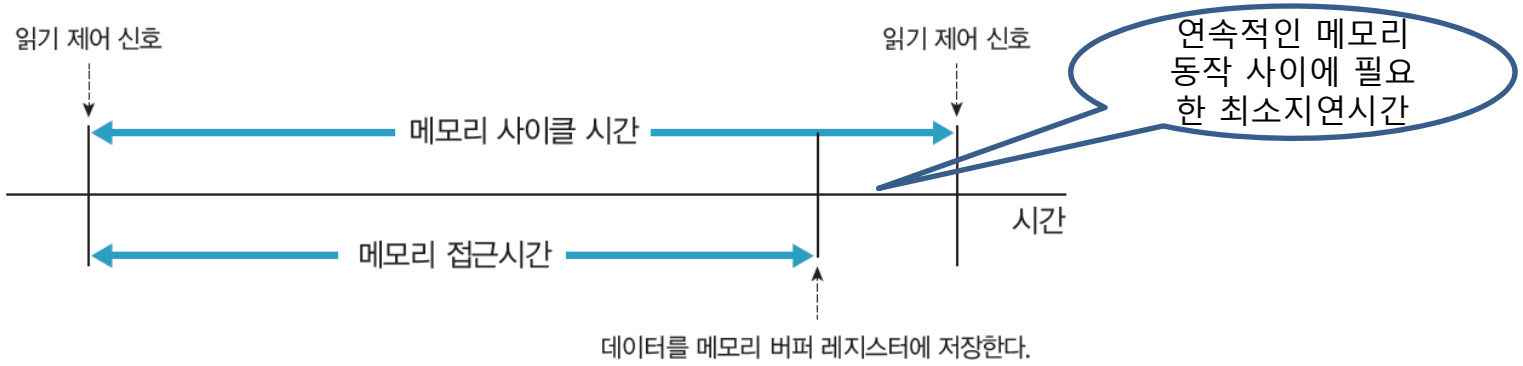


그림 1-8 메모리 접근시간과 메모리 사이클 시간

2. 메모리

■ 캐시

- 프로세서 내부나 외부에 있으며, 처리 속도가 빠른 프로세서와 상대적으로 느린 메인 메모리의 속도 차이를 보완하는 고속 버퍼
  - 메인 메모리에서 데이터를 **블록 단위**로 가져와 프로세서에 **워드 단위**로 전달하여 속도를 높임
  - 데이터가 이동하는 통로(대역폭)를 확대하여 프로세서와 메모리의 속도 차이를 줄임

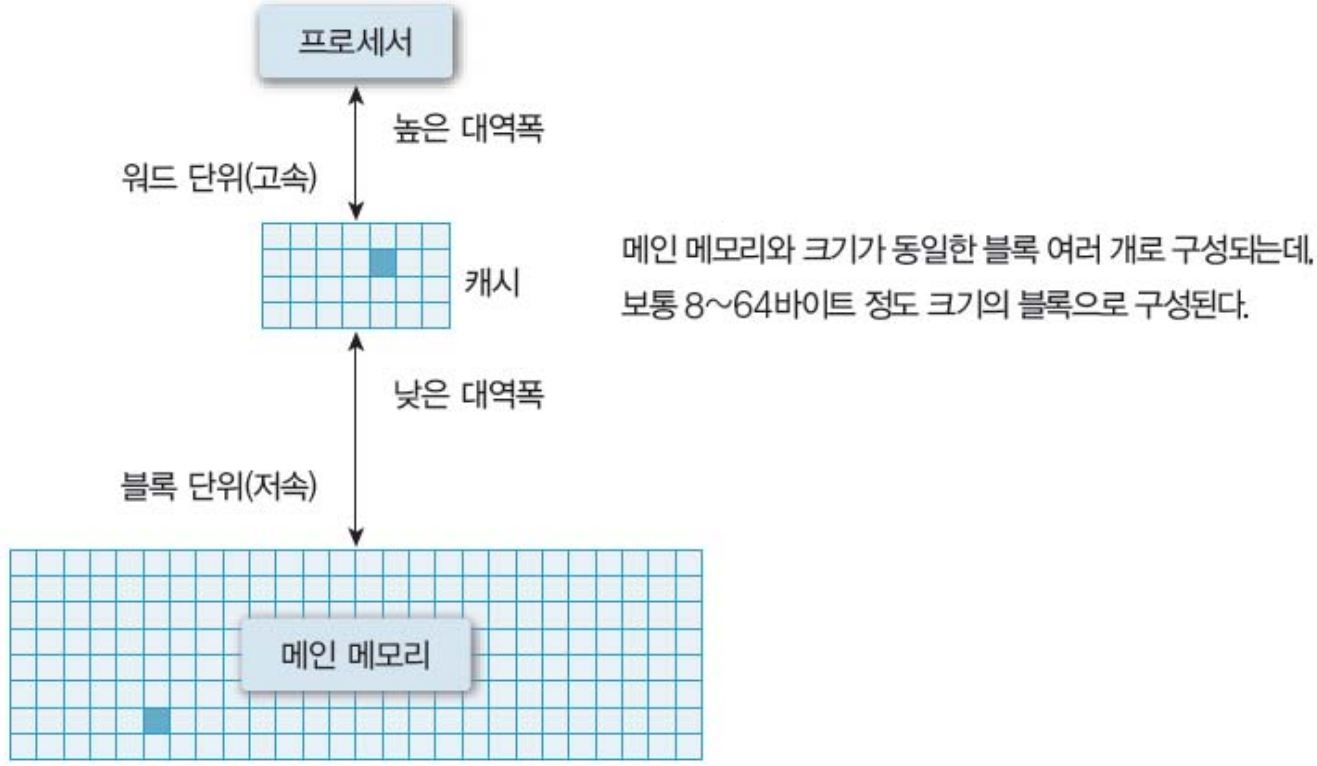


그림 1-10 캐시의 역할

2. 메모리

■ 캐시

▪ 캐시의 기본 동작

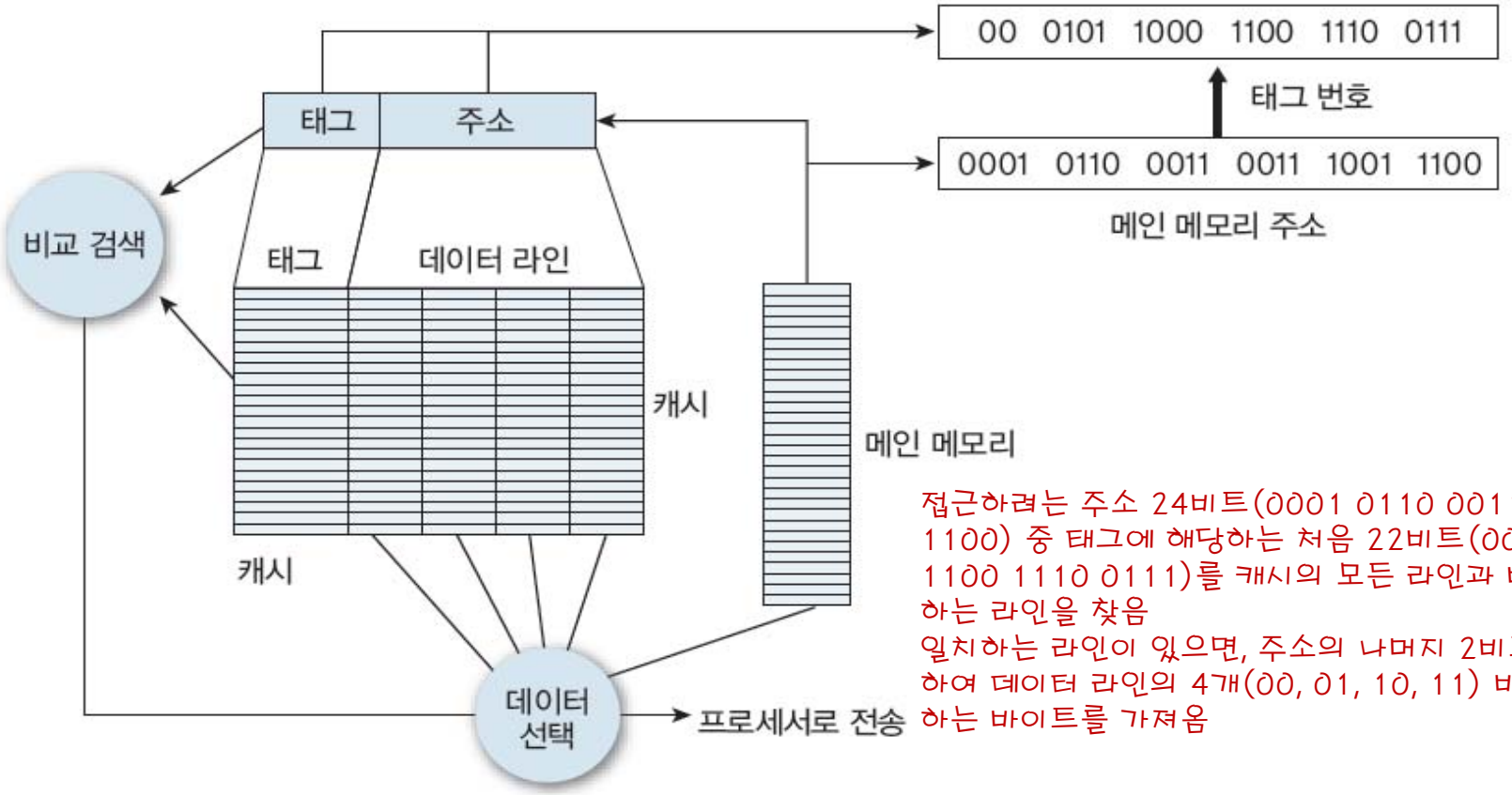


그림 1-11 캐시의 기본 동작

접근하려는 주소 24비트(0001 0110 0011 0011 1001 1100) 중 태그에 해당하는 처음 22비트(00 0101 1000 1100 1110 0111)를 캐시의 모든 라인과 비교하여 일치하는 라인을 찾음
일치하는 라인이 있으면, 주소의 나머지 2비트(00)를 이용하여 데이터 라인의 4개(00, 01, 10, 11) 바이트 중 해당하는 바이트를 가져옴

2. 메모리

캐시

- 캐시의 성능은 작은 용량의 캐시에 프로세서가 이후 참조할 정보가 얼마나 들어 있느냐로 좌우됨
  - 캐시 적중cache hit (캐시 히트) : 프로세서가 참조하려는 정보가 있을 때
  - 캐시 실패cache miss (캐시 미스) : 프로세서가 참조하려는 정보가 없을 때

- 블록의 크기는 캐시의 성능으로 좌우되는데, 실제 프로그램을 실행할 때 참조한 메모리에 대한 공간적 지역성(국부성)과 시간적 지역성이 있기 때문
  - 공간적 지역성spatial locality : 대부분의 프로그램이 참조한 주소와 인접한 주소의 내용을 다시 참조하는 특성
  - 시간적 지역성temporal locality : 한 번 참조한 주소를 곧 다시 참조하는 특성

2. 메모리

캐시

공간적 지역성과 시간적 지역성의 발생 원인

- 프로그램이 명령어를 순차적으로 실행하는 경향이 있어 명령어가 특정 지역 메모리에 인접해 있다.
- 순환(단일 순환, 중첩 순환 등) 때문에 프로그램을 반복하더라도 메모리는 일부 영역만 참조한다.
- 대부분의 컴파일러를 메모리에 인접한 블록에 배열로 저장한다. 따라서 프로그램이 배열 원소에 순차적으로 자주 접근하므로 지역적인 배열 접근 경향이 있다.

→ 지역성은 블록이 크면 캐시의 히트율이 올라갈 수 있음을 의미하지만, 블록이 커지면 이에 따른 전송 부담과 캐시 데이터 교체 작업이 자주 일어나므로 블록 크기를 무작정 늘릴 수는 없음

2. 메모리

보조기억장치

- 주변장치 중 프로그램과 데이터를 저장하는 하드웨어
- 2차 기억장치 또는 외부기억장치라고도 함
- 자기디스크, 광디스크, 자기테이프 등이 있음

3. 시스템 버스

■ 시스템 버스system bus

- 하드웨어를 물리적으로 연결하여 서로 데이터를 주고받을 수 있게 하는 통로
- 컴퓨터 내부의 다양한 신호(데이터 입출력 신호, 프로세서 상태 신호, 인터럽트 요구와 허가 신호, 클록clock 신호 등)를 시스템 버스로 전달
- 기능에 따라 데이터 버스, 주소 버스, 제어 버스로 구분

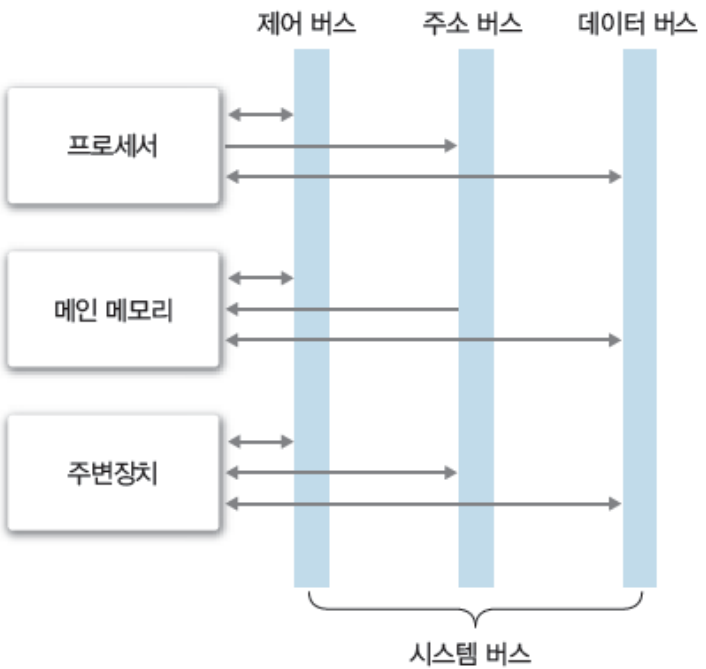


표 1-3 시스템 버스의 종류

종류	설명
데이터 버스	프로세서와 메인 메모리, 주변장치 사이에서 데이터를 전송한다. 데이터 버스를 구성하는 배선 수는 프로세서가 한 번에 전송할 수 있는 비트 수를 결정하는데, 이를 워드라고 한다.
주소 버스	프로세서가 시스템의 구성 요소를 식별하는 주소 정보를 전송한다. 주소 버스를 구성하는 배선 수는 프로세서와 접속할 수 있는 메인 메모리의 최대 용량을 결정한다.
제어 버스	프로세서가 시스템의 구성 요소를 제어하는 데 사용한다. 제어 신호로 연산장치의 연산 종류와 메인 메모리의 읽기나 쓰기 동작을 결정한다.

4. 주변장치

주변장치

- 프로세서와 메인 메모리를 제외한 나머지 하드웨어 구성 요소
- 크게 입력장치, 출력장치, 저장장치로 구분
  - 입력장치
    - 컴퓨터에서 처리할 데이터를 외부에서 입력하는 장치
  - 출력장치
    - 입력장치와 반대로 컴퓨터에서 처리한 데이터를 외부로 보내는 장치
  - 저장장치
    - 메인 메모리와 달리 거의 영구적으로 데이터를 저장하는 장치. 데이터를 입력하여 저장하며, 저장한 데이터를 출력하는 공간이므로 입출력장치에 포함하기도 함

Section 02 컴퓨터 시스템의 동작

컴퓨터 시스템의 작업 처리 순서

❶ 입력장치로 정보를 입력받아 메모리에 저장

❷ 메모리에 저장한 정보를 프로그램 제어에 따라 인출하여 연산장치에서 처리

❸ 처리한 정보를 출력장치에 표시하거나 보조기억장치에 저장

명령어와 데이터

- 입력장치로 컴퓨터에 유입되는 정보
- 명령어는 실행할 산술·논리 연산의 동작을 명시하는 문장으로, 어떤 작업을 수행하는 명령어 집합이 프로그램
- 프로그램은 컴파일러 등을 이용하여 0과 1로 이진화된 기계 명령어로 변환해야 컴퓨터가 이해할 수 있음

1. 명령어의 구조

■ 명령어의 기본 구조

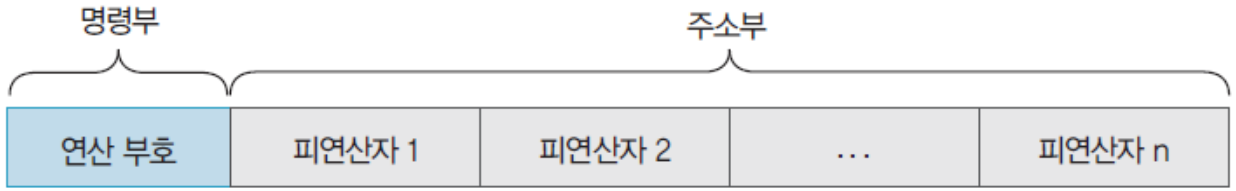


그림 1-13 명령어의 기본 구조

- 연산 부호OPcode, OPeration code (오피코드)
  - 프로세서가 실행할 동작인 연산 지정
  - 산술 연산(+, -, *, /), 논리 연산(AND, OR, NOT), 시프트shift, 보수 등 연산 정의
  - 연산 부호가 n비트이면 최대 2^n개 연산이 가능
- 피연산자operand (오퍼랜드)
  - 연산할 데이터 정보 저장
  - 데이터는 레지스터나 메모리, 가상 기억장치, 입출력장치 등에 위치할 수 있는데 보통 데이터 자체보다는 데이터의 위치 저장

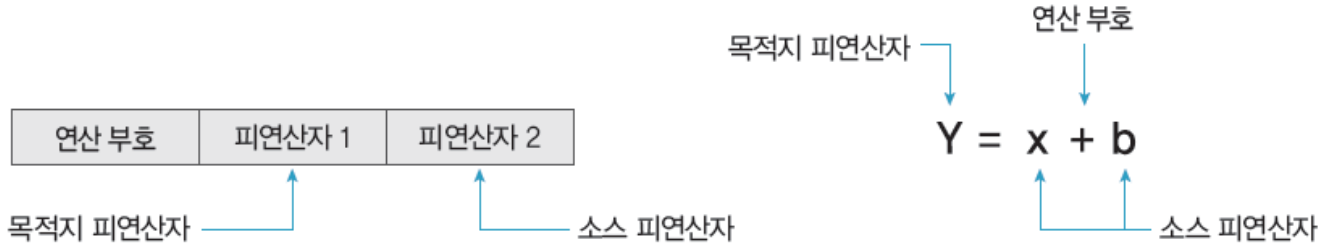


그림 1-14 소스 피연산자와 목적지 피연산자

1. 명령어의 구조

■ 메인 메모리에 저장된 명령어 예

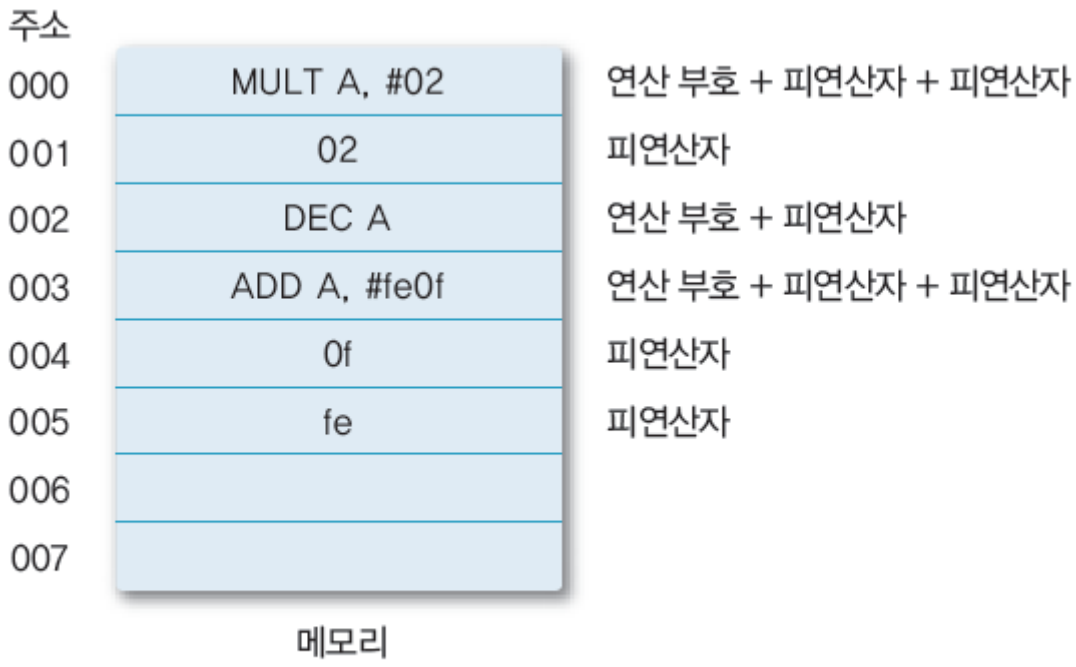


그림 1-15 메인 메모리에 저장된 명령어 예

1. 명령어의 구조

■ 직접 주소와 간접 주소

- 피연산자의 위치를 명시하는 방법(직접 주소 또는 간접 주소)을 나타내는 모드 비트mode bit를 추가하거나, 다음 명령어의 위치를 나타내는 주소를 추가 가능

- 직접 주소direct address
  - 피연산자에 데이터가 있는 레지스터나 메모리 주소 지정

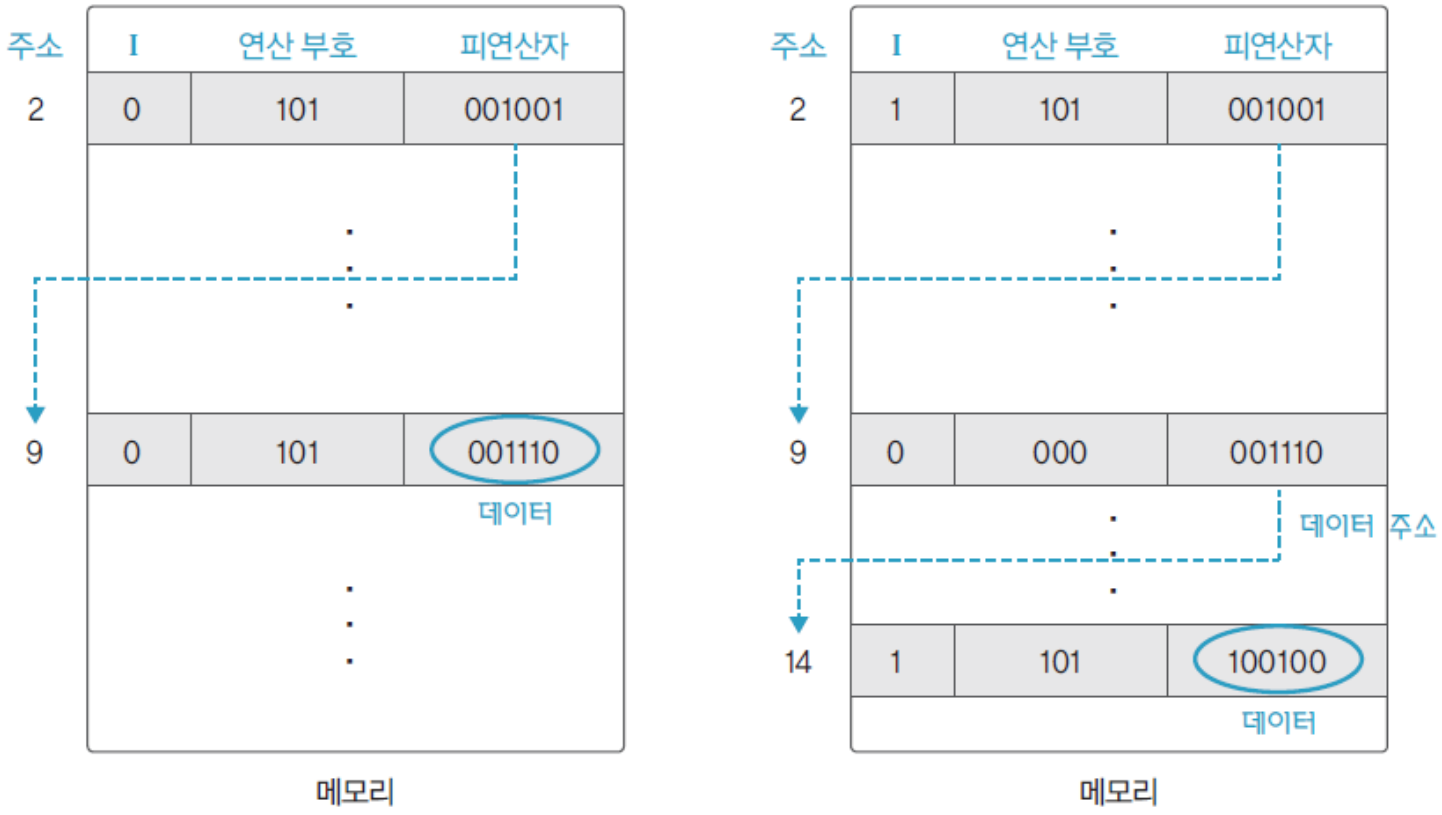
- 간접 주소indirect address
  - 레지스터나 메모리 주소 정보 지정

그림 1-16 모드 비트를 추가한 명령어 형식

1. 명령어의 구조

■ 직접 주소와 간접 주소 사용 예

- 모드가 1비트, 연산 부호가 3비트, 피연산자가 6비트인 명령어



(a) 직접 주소 명령어 (b) 간접 주소 명령어

그림 1-17 직접 주소와 간접 주소를 사용한 명령어 예

2. 명령어의 실행

■ 명령어의 실행 과정

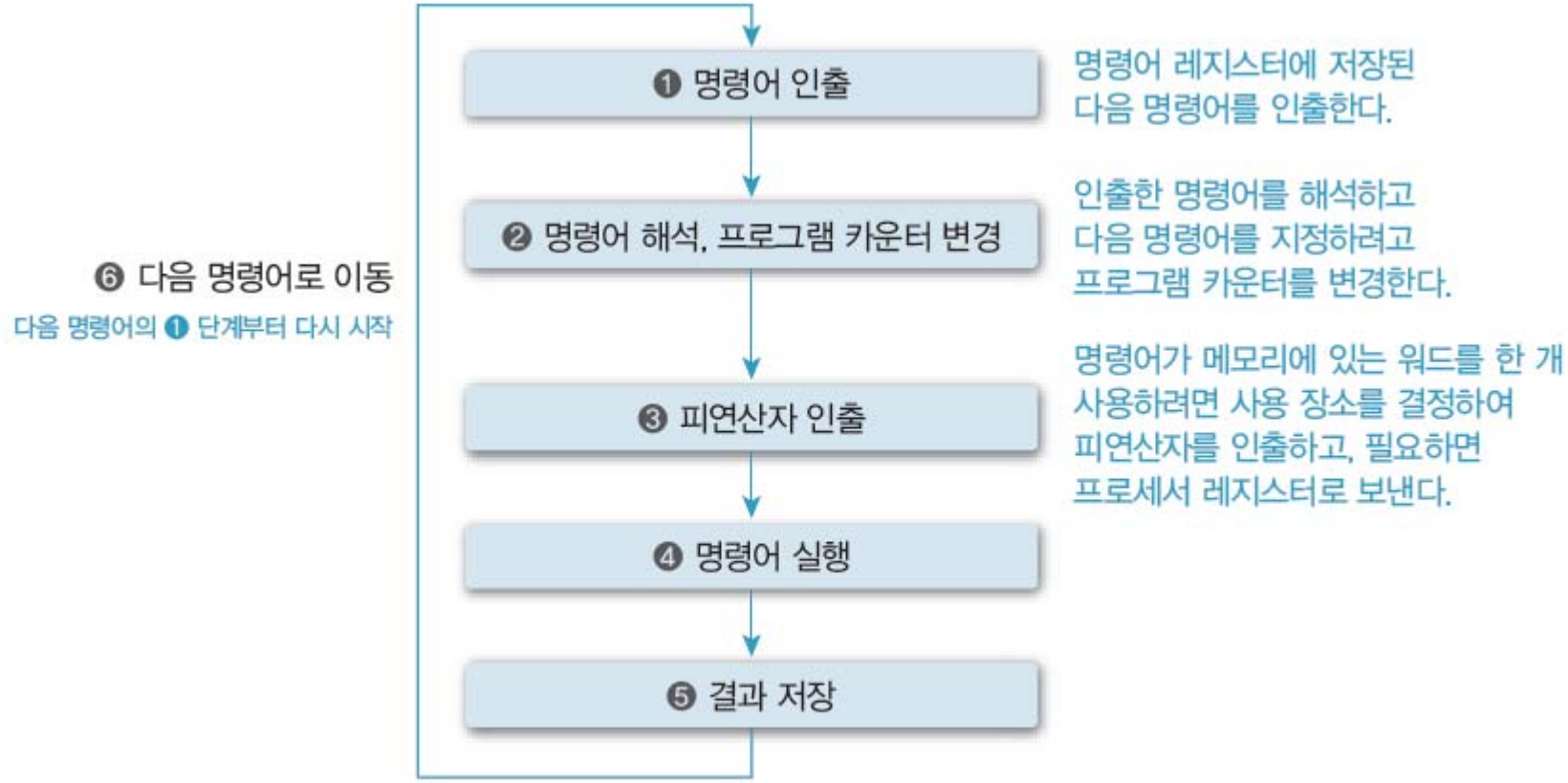
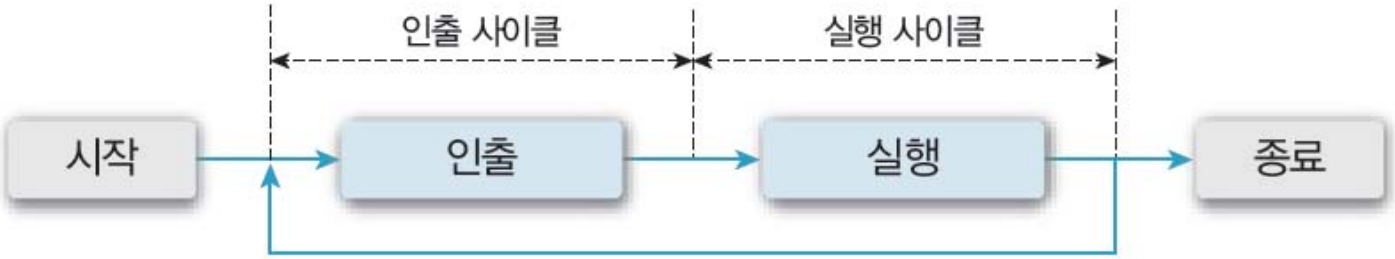


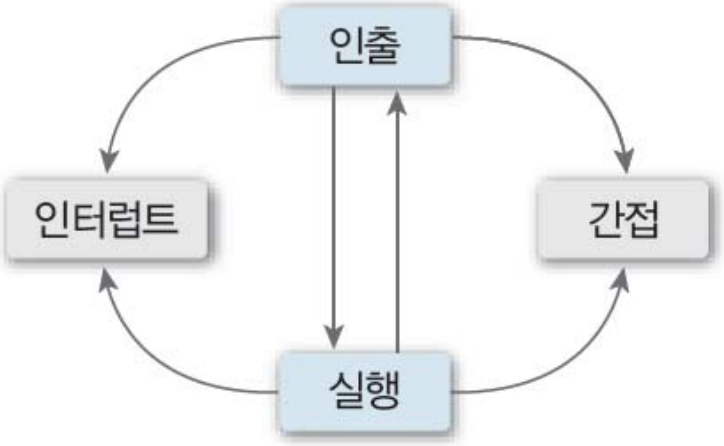
그림 1-18 명령어 실행 과정

2. 명령어의 실행

■ 명령어의 실행 사이클



(a) 일반적인 명령어 사이클



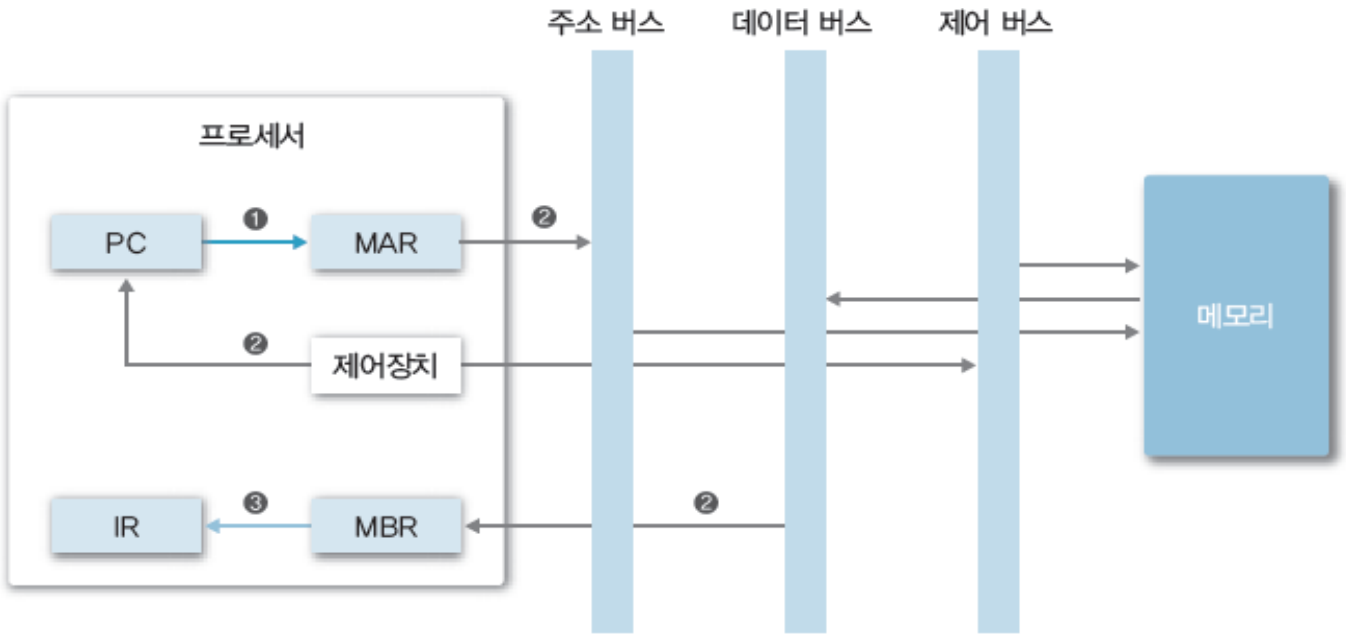
(b) 세분화된 명령어 사이클

그림 1-19 명령어 실행 사이클

2. 명령어의 실행

- **인출 사이클fetch cycle**
  - 메모리에서 명령어를 읽어 명령어 레지스터에 저장하고, 다음 명령어를 실행하려고 프로그램 카운터를 증가시킴
  - 인출 사이클에 소요되는 시간을 명령어 인출 시간이라고 함
- **실행 사이클execution cycle**
  - 인출한 명령어를 해독하고 그 결과에 따라 제어 신호를 발생시켜 명령어 실행
  - 실행 사이클에서 소비되는 시간을 실행 시간이라고 함
- **간접 사이클indirect cycle**
  - 간접 주소 지정 방법을 사용하는 실행 사이클은 명령어를 수행하기 전에 실제 데이터가 저장된 주기억장치의 주소인 유효 주소를 한 번 더 읽어 옴
- **인터럽트 사이클 interrupt cycle**
  - 인터럽트는 프로세서가 프로그램을 수행하는 동안 컴퓨터 시스템의 내부나 외부에서 발생하는 예기치 못한 사건
  - 프로세서는 실행 사이클을 완료한 후 인터럽트 요구가 있는지 검사. 인터럽트 요구가 없으면 다음 명령어를 인출하고, 인터럽트 요구가 있으면 현재 수행 중인 프로그램의 주소(프로그램 카운터) 값을 스택이나 메모리의 0번지와 같은 특정 장소에 저장. 프로그램 카운터에는 인터럽트 처리 루틴의 시작 주소를 저장해 두었다가 인터럽트 처리를 완료하면 중단된 프로그램으로 복귀하여 계속 수행

2. 명령어의 실행

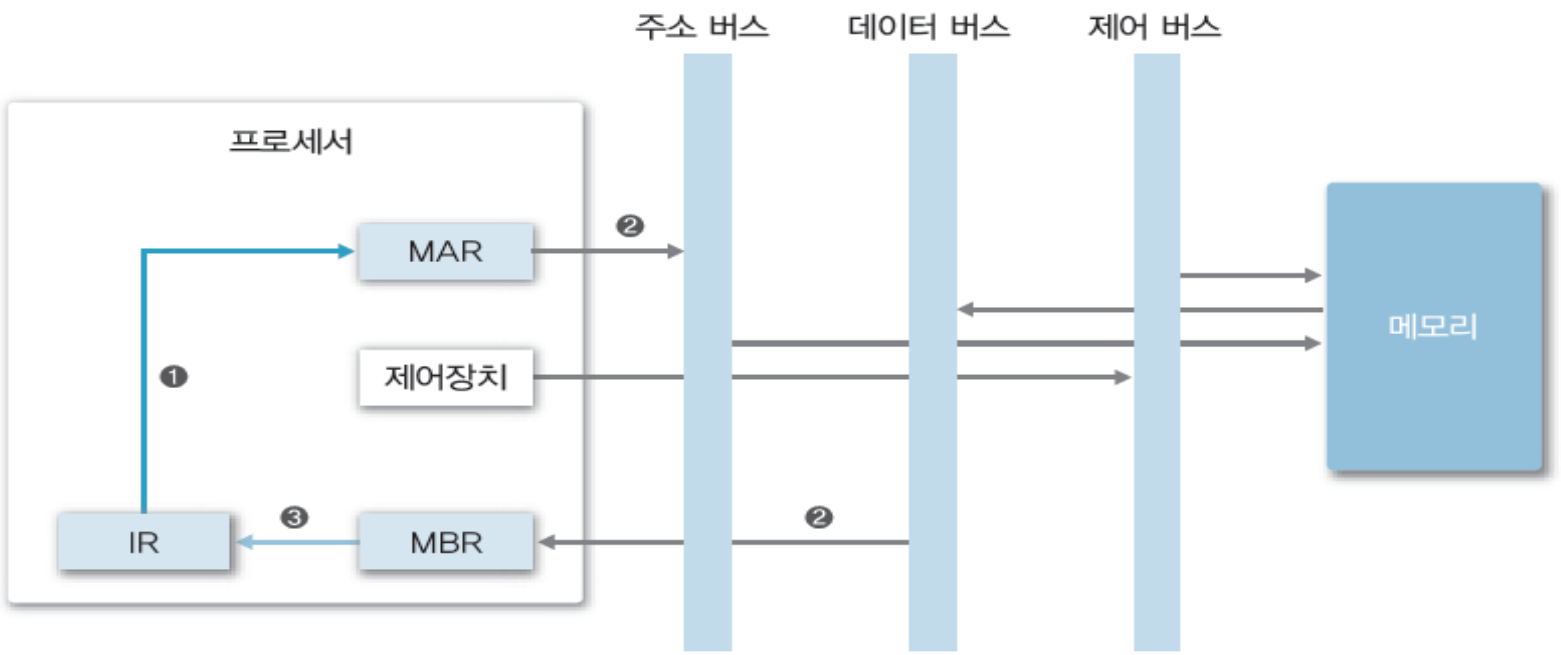


시간	레지스터 동작	설명
❶	PC → MAR	PC에 저장된 주소를 프로세서 내부 버스를 이용하여 MAR에 전달한다.
❷	$Memory^{MAR}$ → MBR	MAR에 저장된 주소에 해당하는 메모리 위치에서 명령어를 인출한 후 이 명령어를 MBR에 저장한다. 이때 제어장치는 메모리에 저장된 내용을 읽도록 제어 신호를 발생시킨다.
	PC + 1 → PC	다음 명령어를 인출하려고 PC를 증가시킨다.
❸	MBR → IR	MBR에 저장된 내용을 IR에 전달한다.

- PC : 프로그램 카운터
- MBR : 메모리 버퍼 레지스터
- MAR : 메모리 주소 레지스터
- IR : 명령어 레지스터

그림 1-20 인출 사이클 과정

2. 명령어의 실행

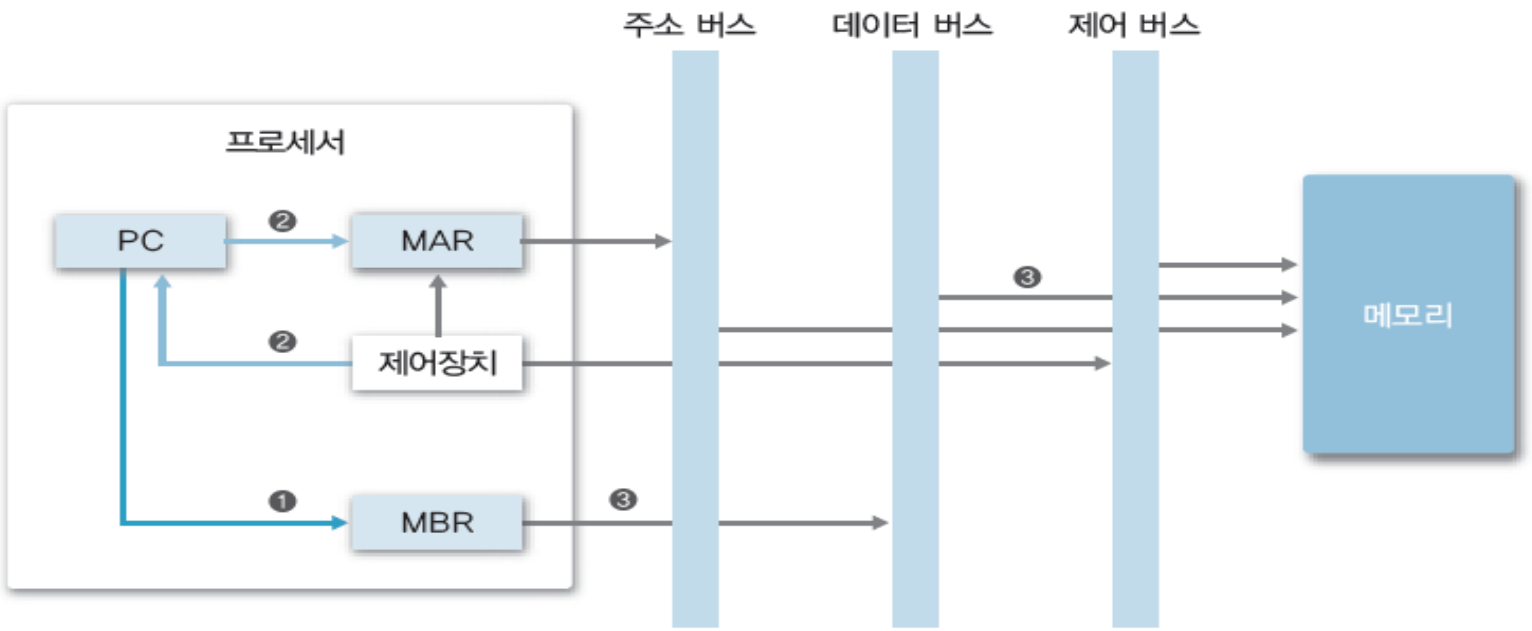


시간	레지스터 동작	설명
❶	$IR^{addr} \rightarrow MAR$	IR에 저장된 명령어의 피연산자(주소부)를 MAR에 전달한다.
❷	$Memory^{MAR} \rightarrow MBR$	MAR에 저장된 주소에 해당하는 메모리 위치에서 데이터를 인출한 후 이 데이터를 MBR에 저장한다. 이때 제어장치는 메모리에 저장된 내용을 읽도록 제어 신호를 발생시킨다.
❸	$MBR \rightarrow IR^{addr}$	MBR에 저장된 내용을 IR에 전달한다.

- IR : 명령어 레지스터
- MAR : 메모리 주소 레지스터
- MBR : 메모리 버퍼 레지스터

그림 1-21 **간접 사이클 과정**

2. 명령어의 실행



시간	레지스터 동작	설명
❶	PC → MBR	PC의 내용을 MBR에 저장한다.
❷	IntRoutine_Address → PC	인터럽트 루틴 주소를 PC에 저장한다.
	Save_Address → MAR	PC에 저장된 인터럽트 루틴 주소를 MAR에 저장한다.
❸	MBR → Memory^{MAR}	MBR의 주소에 있는 내용을 지시된 메모리 셀로 이동한다.

- PC : 프로그램 카운터
- MAR : 메모리 주소 레지스터
- MBR : 메모리 버퍼 레지스터

그림 1-22 인터럽트 사이클 과정

2. 명령어의 실행

인터럽트 명령어

- 현재 실행 중인 프로그램을 중단하고 다른 프로그램의 실행을 요구하는 명령어
- 시스템의 처리 효율을 향상시키며, 프로그램이 실행 순서를 바꿔 가면서 처리하여 다중 프로그래밍에 사용
- 컴퓨터에 설치된 입출력장치나 프로그램 등에서 프로세서로 보내는 하드웨어 신호로 인터럽트를 받은 프로그램은 실행을 중단하고 다른 프로그램을 실행
- 단일 프로세서의 컴퓨터는 명령어를 한 번에 한 개만 수행할 수 있지만, 인터럽트를 이용하면 중간에 다른 프로그램이나 명령어를 수행할 수 있음
- 예상치 못한 사용자 입력, 갑작스런 정전, 컴퓨터 시스템에서 긴급 요청, 잘못된 명령어 수행, 입출력 작업 완료와 같은 상황을 시스템이 적절히 처리하는 데 필요
- 프로그램의 정상 실행을 일시 중단했다 다시 재개하는 과정이지만, 사용자가 별도로 인터럽트 조치를 할 필요가 없고 프로세서와 운영체제가 처리
- 외부장치의 동작과 자신의 동작을 조정하는 수단으로 사용
- 인터럽트 목적으로 사용하는 제어 버스는 인터럽트 요청 회선IRQ, Interrupt ReQuest line

2. 명령어의 실행

■ 인터럽트 요청 회선

- 키보드에서 입력이 발생했을 때만 프로세서에 통보하여 처리하므로, 프로세서가 이벤트 발생 여부를 일일이 감시하지 않아도 됨
- 프로세서가 외부장치의 상태를 직접 점검할 필요가 없어 이 시간 동안 다른 연산을 수행하여 프로세서의 효율을 높일 수 있음
- 인터럽트 요청 신호에 따라 인터럽트 처리 프로그램(인터럽트 서비스 루틴) 수행
- 단일 회선과 다중 회선으로 연결
  - 단일 회선 : 인터럽트 요청이 가능한 모든 장치를 공통의 단일 회선으로 프로세서에 연결하는 방법. 회선 하나에 장치를 여러 개 연결하여 인터럽트를 요청한 장치를 판별하는 기능이 필요.
  - 다중 회선 : 모든 장치를 서로 다른 고유의 회선으로 프로세서와 연결하는 방법. 인터럽트를 요청한 장치를 바로 판별할 수 있음

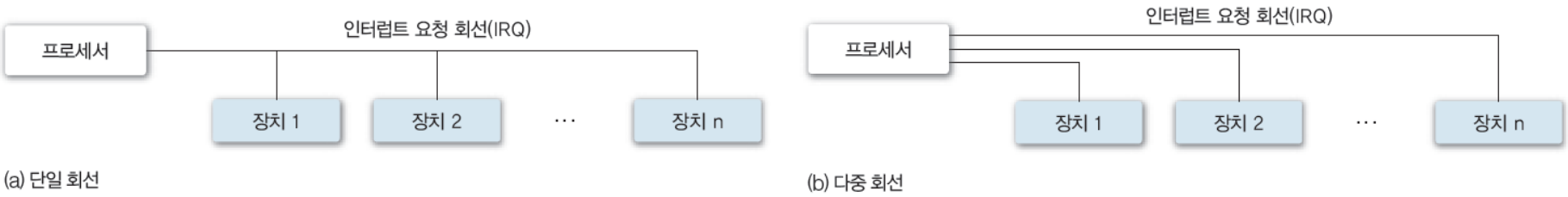
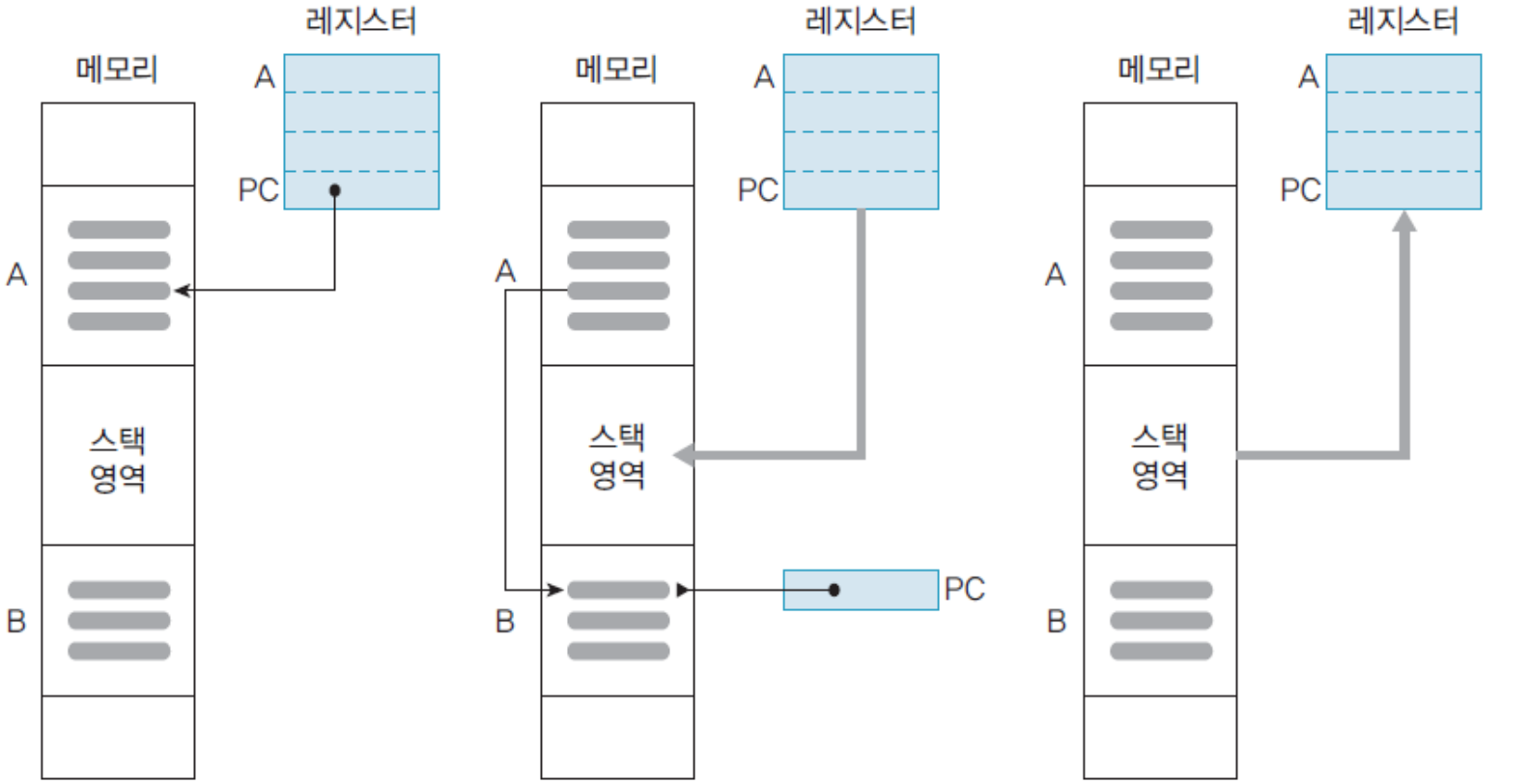


그림 1-23 인터럽트 요청 회선 연결 방법

2. 명령어의 실행

■ 인터럽트 처리 과정



(a) 인터럽트 발생 전

프로그램 A를 실행, 프로그램 카운터PC는 현재 명령어를 가리킴

(b) 인터럽트 발생

현재 명령어를 종료, 레지스터의 모든 내용을 스택 영역(또는 프로세스 제어 블록)에 보내고 pc에는 인터럽트 처리 프로그램(프로그램 B)의 시작 위치를 저장하고 제어를 넘긴 프로그램 B 실행

(c) 인터럽트 처리 후

스택 영역에 있던 내용을 레지스터에 다시 저장하며, 프로그램 A가 다시 시작하는 위치를 저장하고 중단했던 프로그램 A를 재실행

그림 1-24 인터럽트 처리 과정

사이클 훔치기

사이클 훔치기(cycle stealing)

- 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

병렬 처리의 개념

볶음밥 조리의 병렬 처리

그림 2-27 볶음밥 조리의 병렬 처리

파이프라인 기법

- 하나의 코어에 여러 개의 스레드thread를 이용하는 방식

병렬 처리의 개념

슈퍼스칼라 기법

- 듀얼코어 CPU를 이용해 2개의 작업을 동시에 처리하는 방식

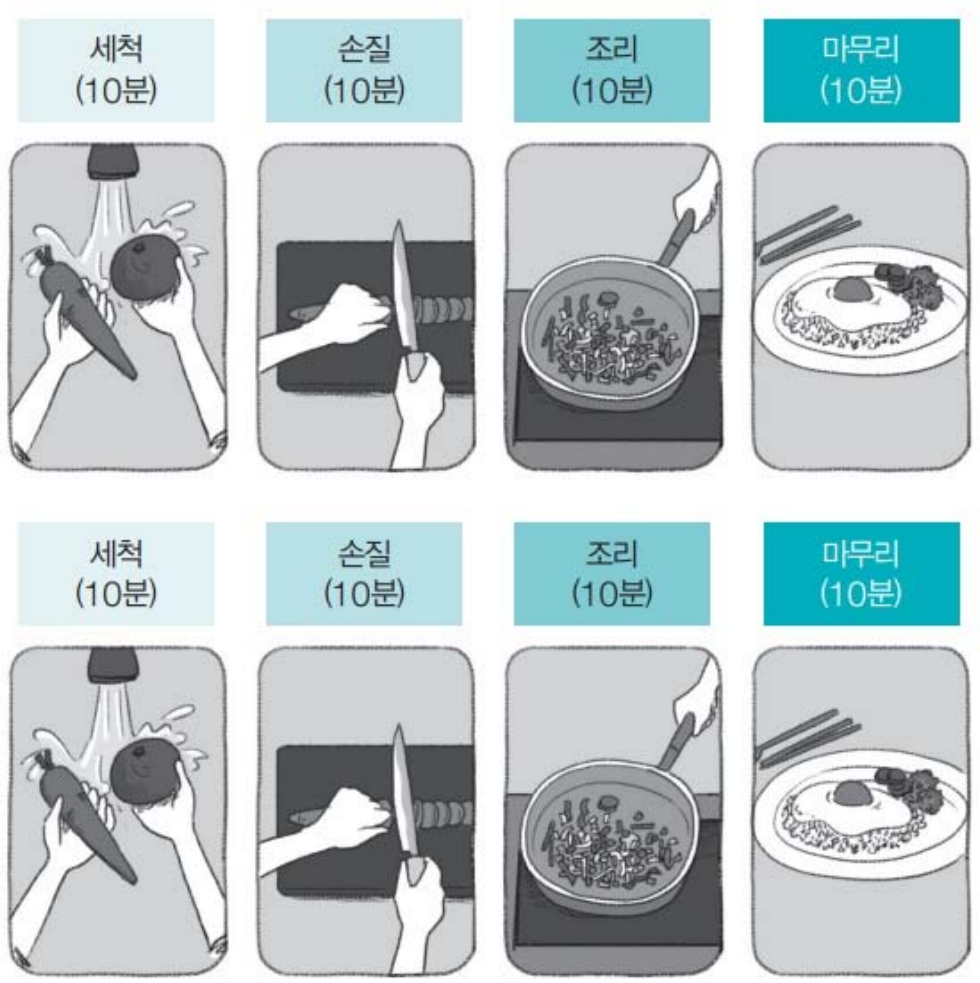


그림 2-28 조리 시설이 2개인 경우의 병렬 처리

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

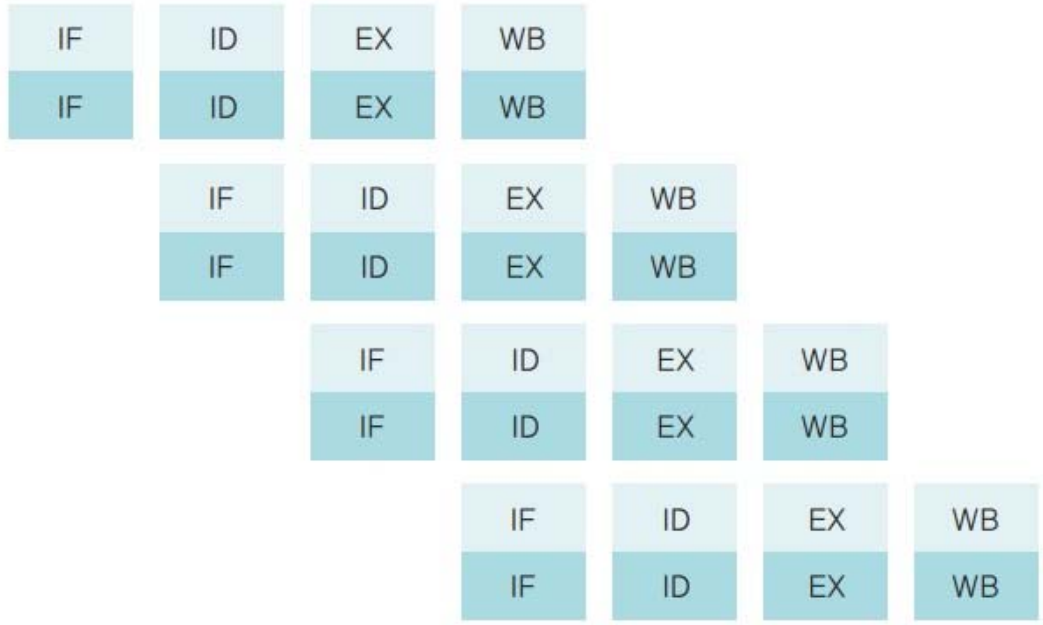


그림 2-36 슈퍼스칼라 기법의 동작 과정

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

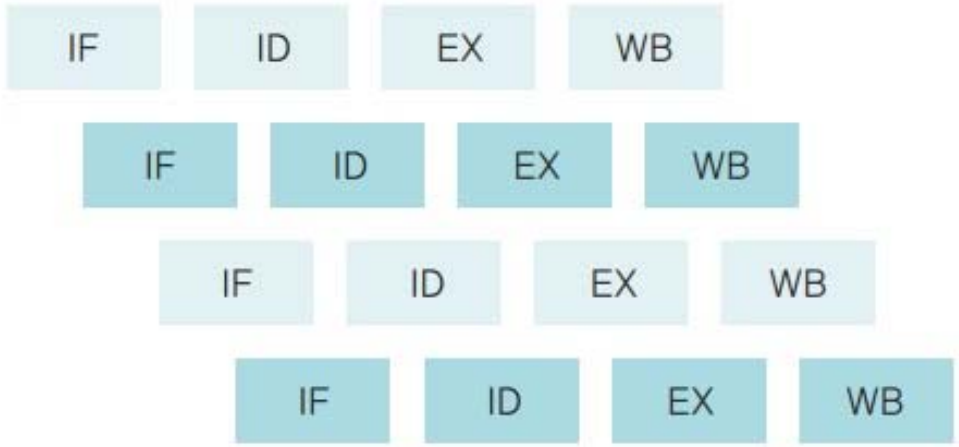


그림 2-37 슈퍼파이프라인 기법의 동작 과정

병렬 처리 기법

슈퍼파이프라인 슈퍼스칼라 기법super-pipelined super-scalar

- 슈퍼파이프라인 기법을 여러 개의 코어에서 동시에 수행하는 방식

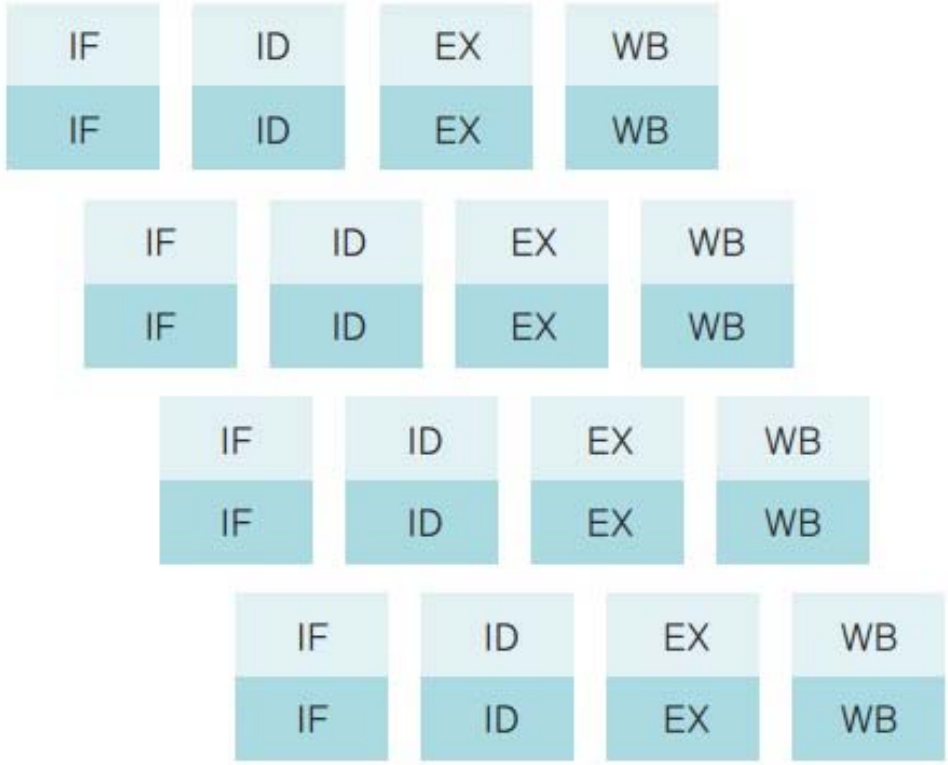


그림 2-38 슈퍼파이프라인 슈퍼스칼라 기법의 동작 과정

병렬 처리 기법

VLIW 기법Very 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