

1과목 · 데이터베이스

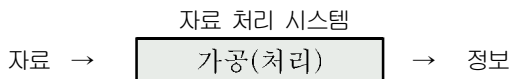
핵심 001 | 정보처리 시스템

정보시스템

- 조직체에 필요한 DATA를 수집, 저장해 두었다가 필요시에 처리해서 의사결정에 유용한 정보를 생성하고 분배하는 수단
- 사용하는 목적에 따라 경영정보 시스템, 군사 정보시스템, 인사 행정 정보 시스템, 의사 결정 지원 시스템 등으로 사용됨

정보와 자료

- 자료(Data) : 현실세계에서 관찰이나 측정을 통해 수집한 단순한 사실이나 결과값으로, 가공되지 않은 상태
- 정보(Information) : 의사 결정에 도움을 줄 수 있는 유용한 형태로, 자료를 가공(처리)해서 얻을 수 있는 결과



- 자료처리 시스템 : 정보시스템이 사용할 자료를 처리하는 정보 시스템의 서브시스템으로, 처리형태에 따라 일괄 처리 시스템, 온라인 실시간 처리 시스템, 분산 처리 시스템으로 분류됨
- 데이터 웨어 하우스(DataWare House) : 조직이나 기업체의 중심이 되는 주요 업무 시스템에서 추출되어 새로이 생성된 데이터베이스로서 의사결정지원 시스템을 지원하는 주제적, 통합적, 시간적 데이터의 집합체

핵심 002 | 데이터베이스의 정의

- 통합된 데이터(Integrated data) : 자료의 중복을 배제한 데이터의 모임
- 저장된 데이터(Stored data) : 컴퓨터가 접근할 수 있는 저장 매체에 저장된 자료
- 운영 데이터(Operational data) : 조직의 업무를 수행하는데 있어서 존재 가치가 확실하고 없어서는 안 될 반드시 필요한 자료
- 공용 데이터 : 여러 응용 시스템들이 공동으로 소유하고 유지하는 자료

핵심 003 | 데이터베이스의 특징

- 실시간 접근성 : 수시적이고 비정형적인 질의(조회)에 대하여 실시간 처리(real time processing) 응답이 가능함
- 계속적인 변화 : 새로운 데이터의 삽입(insertion), 삭제(deletion), 갱신(update)으로 항상 최신의 데이터를 유지함
- 동시 공유 : 여러 사용자가 동시에 자기가 원하는 데이터를 이용할 수 있음
- 내용에 의한 참조 : 데이터베이스에 있는 데이터를 참조할 때 데이터 주소나 위치에 의해서가 아니라 사용자가 요구하는 데이터 내용으로 데이터를 찾음

핵심 004 | 기존의 파일 처리 방식에서의 문제점

중복성으로 인한 문제점

- 중복성 : 응용프로그램과 데이터 파일이 상호 의존적인 관계

- 데이터 파일이 보조 기억 장치에 저장되는 방법이나 저장된 데이터의 접근 방법을 변경할 때는 응용프로그램도 같이 변경하여야함

중복성으로 인한 문제점

- 일관성 : 중복된 데이터간에 내용이 일치하지 않는 상황이 발생하여 일관성이 없어짐
- 보안성 : 중복되어 있는 모든 데이터에 동등의 보안수준을 유지하기가 어려움
- 경제성 : 저장 공간의 낭비와 동일한 데이터의 반복 작업으로 인한 비용의 증가
- 무결성 : 제어의 분산으로 인해 데이터의 정확성을 유지할 수 없음

핵심 005 | DBMS의 필수기능

- 정의(조직) : 데이터의 형(Type)과 구조, 데이터가 DB에 저장될 때의 제약조건 등을 명시하는 기능
- 조작 : 체계적 처리를 위한 데이터 접근 수단 등을 정하는 기능
- 제어 : 무결성, 보안 및 권한 검사, 병행수행 제어 등의 기능을 정하는 기능

핵심 006 | DBMS의 장단점

장점

- 데이터의 중복을 피할 수 있음
- 저장된 자료를 공동으로 이용할 수 있음
- 데이터의 일관성을 유지할 수 있음
- 데이터의 무결성을 유지할 수 있음
- 보안을 유지할 수 있음
- 데이터를 표준화할 수 있음
- 데이터를 통합하여 관리할 수 있음
- 항상 최신의 데이터를 유지함
- 데이터의 실시간 처리가 가능함
- 데이터의 논리적 물리적 독립성이 보장됨

단점

- 데이터베이스 전문가 부족
- 전산화 비용이 증가함
- 대용량 디스크로의 집중적인 Access로 과부하(Overhead)가 발생함
- 파일의 예비(Backup)와 회복(Recovery)이 어려움
- 시스템이 복잡함

핵심 007 | 데이터의 독립성

- 논리적 독립성 : 응용 프로그램과 데이터베이스를 독립시킴으로써, 데이터의 논리적 구조를 변경시키더라도 응용 프로그램은 변경되지 않음
- 물리적 독립성 : 응용 프로그램과 보조기억장치와 같은 물리적 장치를 독립시킴으로써, 데이터베이스 시스템의 성능 향상을 위해 새로운 디스크를 도입하더라도 응용 프로그램에는 영향을 주지 않고 데이터의 물리적 구조만을 변경함

핵심 008 | 스키마(Schema)의 정의

- 데이터베이스의 구조와 제약조건에 관한 전반적인 명세(Specification)를 기술(Description)함
- 데이터베이스를 구성하는 데이터 개체(Entity), 속성(Attribute), 관계(Relationship) 및 데이터 조작시 데이터 값들이 갖는 제약조건 등에 관해 전반적으로 정의함

핵심 009 | 스키마의 3계층

외부 스키마(External Schema) = 서브 스키마 = 사용자 뷰(View)

- 사용자나 응용프로그램이 각 개인의 입장에서 필요로 하는 데이터베이스의 논리적 구조 정의
- 전체 데이터베이스의 한 논리적인 부분으로 볼 수 있으므로 서브스키마(subschema)라고도 함
- 하나의 데이터베이스 시스템에는 여러 개의 외부 스키마가 존재할 수 있으며, 하나의 외부 스키마를 여러 개의 응용 프로그램이나 사용자가 공유할 수 있음
- 같은 데이터베이스에 대해서도 서로 다른 관점을 정의할 수 있도록 허용함

개념 스키마(Conceptual Schema) = 전체적인 뷰(View)

- 데이터베이스의 전체적인 논리적 구조로서 모든 응용 프로그램이나 사용자들이 필요로 하는 데이터를 종합한 조직 전체의 데이터베이스로 하나만 존재함
- 개념 스키마는 개체간의 관계와 제약조건을 나타내고 데이터베이스의 접근 권한, 보안 및 무결성 규칙에 관한 명세를 정의함
- 단순히 스키마(schema)라고하면 개념 스키마를 의미함
- 기관이나 조직체의 관점에서 데이터베이스를 정의한 것임

내부 스키마(Internal Schema)

- 데이터베이스의 물리적 구조
- 데이터의 실제 저장 방법 기술
- 물리적인 저장장치와 밀접한 계층
- 시스템 프로그래머나 시스템 설계자가 보는 관점의 스키마

핵심 010 | 데이터베이스 언어(Database Language)

데이터 정의 언어(DDL: Data Definition Language)

- DB 구조, 데이터 형식, 접근 방식 등 DB를 구축하거나 수정할 목적으로 사용하는 언어
- 번역한 결과가 데이터사전(Data-dictionary)이라는 특별한 파일에 여러 개의 테이블로서 저장됨
- 데이터 정의 언어의 기능
 - 외부 스키마 명세 정의
 - 데이터베이스 정의 및 수정
 - 스키마에 사용되는 제약조건에 대한 명세 정의
 - 데이터의 물리적 순서 규정

데이터 조작 언어(DML: Data Manipulation Language) = 서브 언어

- 사용자로 하여금 데이터를 처리할 수 있게 하는 도구로서 사용자(응용프로그램)와 DBMS간의 인터페이스를 제공함
- 응용 프로그램을 통하여 사용자가 DB의 데이터를 실질적으로 조작할 수 있도록 하기 위해 FORTRAN, COBOL 등의 호스트 언어에 DB 기능을

추가시켜 만든 언어

- 대표적인 데이터 조작어(DML)에는 질의어가 있으며, 질의어는 터미널에서 주로 이용하는 비절차적(procedural) 데이터 언어임

데이터 제어 언어(DCL: Data Control Language)

- 무결성, 보안 및 권한 제어, 회복을 등을 하기 위한 언어
- 데이터를 보호하고 데이터를 관리하는 목적으로 사용됨
- 데이터 제어 언어의 기능
 - 불법적인 사용자로부터 데이터를 보호하기 위한 데이터 보안(Security)
 - 데이터 정확성을 위한 무결성(Integrity)유지
 - 시스템 장애에 대비한 데이터 회복과 병행 수행

핵심 011 | DBA(DataBase Administrator)

데이터베이스 시스템의 모든 관리와 운영에 대한 책임을 지고 있는 사람이나 그룹

데이터베이스 설계와 조작에 대한 책임

- 데이터베이스 구성요소 결정
- 개념 스키마 및 내부스키마 정의
- 데이터베이스의 저장 구조 및 접근 방법 정의
- 보안 및 데이터베이스의 접근 권한 부여 정책 수립
- 장애에 대비한 예비(Back Up) 조치와 회복(Recovery)에 대한 전략 수립
- 무결성을 위한 제약 조건의 지정
- 데이터 사전의 구성과 유지관리
- 사용자의 변화 요구와 성능향상을 위한 데이터베이스의 재구성

행정책임

- 사용자의 요구와 불평의 청취 및 해결
- 데이터 표현 방법의 표준화
- 문서화에 대한 기준 설정

시스템 감시 및 성능분석

- 변화 요구에 대한 적응과 성능 향상에 대한 감시
- 시스템 감시 및 성능분석
- 자원의 사용도와 병목현상 조사
- 데이터 사용 추세, 이용 형태 및 각종 통계 등을 종합, 분석함

핵심 012 | 데이터 모델의 정의

- 현실 세계의 정보들을 컴퓨터에 표현하기 위해서 단순화, 추상화하여 체계적으로 표현한 개념적 모형임
- 현실 세계를 데이터베이스에 표현하는 중간 과정, 즉 데이터베이스 설계과정에서 데이터의 구조를 표현하기 위해 사용되는 도구임
- 데이터의 구조(Schema)를 논리적으로 묘사하기 위해 사용되는 지능적 도구임

핵심 013 | 데이터 모델의 종류

개념적 데이터 모델

- 속성들로 기술된 개체 타입과 이 개체 타입들 간의 관계를 이용하여 현실 세계를 표현하는 방법

- 종류로는 E-R 모델이 있음

논리적 데이터 모델

- 필드로 기술된 데이터 타입과 이 데이터 타입들 간의 관계를 이용하여 현실 세계를 표현하는 방법
- 단순히 데이터 모델이라고 하면 논리적 데이터 모델을 의미함
- 논리적 데이터베이스 모델은 데이터 간의 관계를 어떻게 표현하느냐에 따라 관계모델, 계층모델, 네트워크 모델로 구분함

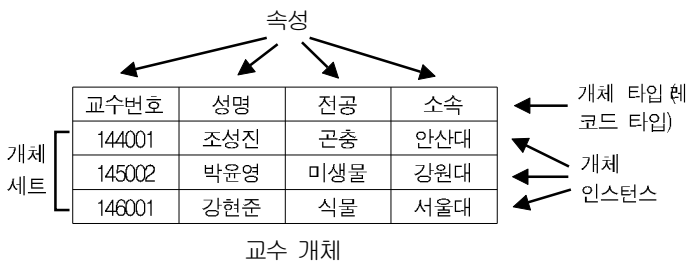
핵심 014 | 데이터 모델의 구성요소

개체(Entity)

- 데이터베이스에 표현하려는 것으로 사람이 생각하는 개념이나 정보 단위 같은 현실 세계의 대상체
- 유형, 무형의 정보로서 서로 연관된 몇 개의 속성으로 구성됨
- 파일 시스템의 레코드에 대응하는 것으로 어떤 정보를 제공하는 역할을 수행함
- 독립적으로 존재하거나 그 자체로서도 구별이 가능함

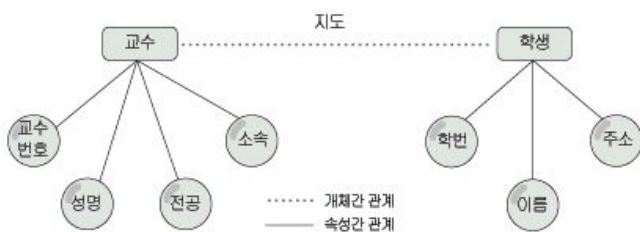
속성(Attribute)

- 데이터의 가장 작은 논리적 단위로서 파일 구조상의 데이터 항목 또는 데이터 필드에 해당함
- Entity를 구성하는 항목



관계(Relationship)

- Entity간의 관계 또는 Attribute간의 관계



- 위 그림의 관계는 교수가 학생을 지도하는 관계임

핵심 015 | 개체-관계(Entity-Relationship) 모델

- 개념적 데이터 모델의 가장 대표적인 것으로, 1976년 Peter Chen에 의해 제안됨
- 개체 타입(entity type)과 이들 간의 관계 타입(relationship type)을 이용해 현실 세계를 개념적으로 표현함
- 데이터를 개체(Entity), 관계(Relationship), 속성(Attribute)으로 묘사함
- 특정 DBMS를 고려한 것은 아님

핵심 016 | E-R 다이어그램

기호	기호이름	의미
	사각형	개체(Entity) 타입
	다이아몬드	관계(Relationship) 타입
	타원	속성(Attribute)
	밑줄 타원	기본키 속성
	복수 타원	복합 속성 (예) 성명은 성과 이름으로 구성
	관계	1:1, 1:n, n:m 등의 개체 관계에 대해 선 위에 대응수 기술
	선, 링크	개체타입과 속성을 연결

핵심 017 | 관계형 데이터 모델

- 계층 모델과 망 모델의 복잡한 구조를 단순화시킨 모델
- 표(Table)를 이용해서 데이터 상호관계를 정의하는 DB 구조
- 데이터간의 관계를 기본키(primary key)와 이를 참조하는 외래키(foreign key)로 표현함
- 대표적인 언어 : Oracle, MS-SQL, Informix
- 1:1, 1:N, M:N 관계를 자유롭게 표현할 수 있음
- 장점 : 간결하고, 보기 편리하고, 다른 데이터베이스로의 변환이 용이함
- 단점 : 성능이 다소 떨어짐

핵심 018 | 계층형 데이터 모델

- 데이터의 논리적 구조도가 트리 형태이며, 개체가 트리를 구성하는 노드 역할을 함
- 개체 집합에 대한 속성 관계를 표시하기 위해 개체를 노드로 표현하고 개체 집합들 사이의 관계를 링크로 연결함
- 개체간의 관계를 부모와 자식간의 관계로 표현함
- 개체 타입간에는 상위와 하위관계가 존재하며, 일 대 다(1:N) 대응 관계만 존재함
- 레코드 삭제시 연쇄 삭제(Triggered Delete)가 됨
- 개체 타입들간에는 사이클(cycle)이 허용되지 않음
- 계층형 모델에서는 개체(Entity)를 세그먼트(Segment)라 부름
- 대표적인 DBMS는 IMS임
- 관계의 유형
 - 속성 관계(Attribute Relation) : 세그먼트(개체)를 구성하는 속성들의 관계
 - 개체 관계(Entity Relation) : 개체와 개체간의 관계를 링크로 표시함

핵심 019 | 망(그래프)형 데이터 모델

- CODASYL이 제안한 것으로, CODASYL DBTG 모델이라고도 함
- 그래프를 이용해서 데이터 논리구조를 표현한 데이터 모델임

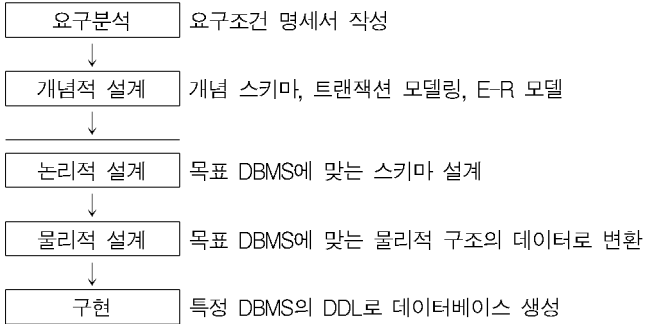
- 상위와 하위 레코드 사이에서 다 대 다(N:M) 대응관계를 만족하는 구조임
- 레코드 타입간의 관계는 1:1, 1:N, N:M이 될 수 있음
- 대표적 DBMS : DBTG, EDBS, TOTAL

- 하나의 애트리뷰트가 취할 수 있는 같은 타입의 원자(atomic) 값들의 집합
- 실제 애트리뷰트 값이 나타날 때 그 값의 합법여부를 시스템이 검사하는 데에도 이용됨

차수(Degree) : Attribute의 개수

기수(대응수; Cardinality) : Tuple의 개수

핵심 020 | 데이터베이스 설계순서



핵심 021 | 관계 데이터베이스 특징

- 1970년 IBM에 근무하는 E. F. Codd에 의해 처음 제안되었음
- 개체(Entity)나 관계(Relationship)를 모두 릴레이션(Relation)이라는 표(Table)로 표현함
- 릴레이션은 개체를 표현하는 개체 릴레이션, 관계를 나타내는 관계 릴레이션으로 구분할 수 있음
- 장점 : 간결하고 보기 편리하며, 다른 데이터베이스로의 변환이 용이함
- 단점 : 성능이 다소 떨어진다.

핵심 022 | 관계 데이터베이스의 Relation 구조

데이터들을 표(Table)의 형태로 표현한 것으로 구조를 나타내는 릴레이션 스키마와 실제 값들인 릴레이션 인스턴스로 구성됨

<학생> 릴레이션



튜플(Tuple)

- 릴레이션을 구성하는 각각의 행
- 속성의 모임으로 구성됨
- 파일 구조에서 레코드와 같은 의미

속성(Attribute, 애트리뷰트)

- 데이터베이스를 구성하는 가장 작은 논리적 단위
- 파일 구조상의 데이터 항목 또는 데이터 필드에 해당됨
- 개체의 특성을 기술함

도메인(Domain)

핵심 023 | 키(Key)의 개념 및 종류

데이터베이스에서 조건에 만족하는 튜플을 찾거나 순서대로 정렬할 때 기준이 되는 애트리뷰트

후보키 (Candidate Key)	<ul style="list-style-type: none"> • 릴레이션을 구성하는 속성들 중에서 튜플을 유일하게 식별하기 위해 사용하는 속성들의 부분집합, 즉 기본키로 사용할 수 있는 속성들을 말함 • 릴레이션에 있는 모든 튜플에 대해서 유일성과 최소성을 만족시켜야 함
기본키 (Primary Key)	<ul style="list-style-type: none"> • 후보키 중에서 선택한 주키(Main Key) • 한 릴레이션에서 특정 튜플을 유일하게 구별할 수 있는 속성 • Null 값을 가질 수 없음 • 기본키로 정의된 속성에는 동일한 값이 중복되어 저장될 수 없음
대체키 (Alternate Key)	<ul style="list-style-type: none"> • 후보키가 둘 이상일 때 기본키를 제외한 나머지 후보키들을 말함 • 보조키라고도 함
슈퍼키 (Super Key)	<ul style="list-style-type: none"> • 릴레이션에서 같은 튜플이 발생하지 않는 키를 구성할 때, 속성의 집합으로 구성하는 것을 말함 • 릴레이션을 구성하는 모든 튜플에 대해 유일성은 만족시키지만, 최소성은 만족시키지 못함
외래키 (Foreign Key)	<ul style="list-style-type: none"> • 관계(Relation)를 맺고 있는 릴레이션 R1, R2에서 릴레이션 R1이 참조하고 있는 릴레이션 R2의 기본키와 같은 R1 릴레이션의 속성 • 외래키로 지정되면 참조테이블의 기본키에 없는 값은 입력할 수 없음

※ 널 값(NULL Value) : 데이터베이스에서 아직 알려지지 않거나 모르는 값으로서 "해당없음" 등의 이유로 정보 부재를 나타내기 위해 사용하는, 이론적으로 아무것도 없는 특수한 데이터

핵심 024 | 무결성

- 개체 무결성 : 릴레이션에서 기본키를 구성하는 속성은 널(NULL) 값이나 중복값을 가질 수 없음
 예) '학생' 릴레이션에서 '학번' 이 기본키로 정의 되면 튜플을 추가할 때 '주민번호' 나 '성명' 필드에는 값을 입력하지 않아도 되지만 '학번' 속성에는 반드시 값이 입력되어야 함, 또한 '학번' 속성에는 이미 한 번 입력한 속성 값을 중복하여 입력할 수 없음
- 참조 무결성 : 외래키 값은 NULL이거나 참조 릴레이션의 기본키 값과 동일해야 함 즉, 릴레이션은 참조할 수 없는 외래키값을 가질 수 없음
 예) '수강' 릴레이션의 '학번' 속성에는 '학생' 릴레이션의 '학번' 속성에 없는 값은 입력할 수 없음

핵심 025 | 순수관계 연산자

관계 데이터베이스에 적용할 수 있도록 특별히 개발한 관계 연산자

Select

- 릴레이션에 존재하는 튜플중에서 선택조건을 만족하는 튜플의 부분 집합을 구하여 새로운 릴레이션을 만들
- 릴레이션의 행(가로)에 해당하는 튜플을 구하는 것이므로 수평연산 이라고도 함
- 연산자의 기호는 그리스문자 시그마(σ)를 사용함
- 표기 형식 : $\sigma_{\langle\text{조건}\rangle}(R)$ 단, R은 릴레이션 이름

Project

- 주어진 릴레이션에서 속성 List에 제시된 Attribute만을 추출하는 연산
- 릴레이션의 열(세로)에 해당하는 Attribute을 추출하는 것이므로 수직 연산자라고도 함
- 연산자의 기호는 그리스문자 파이(π)를 사용함
- 표기 형식 : $\pi_{\langle\text{속성리스트}\rangle}(R)$ 단, R은 릴레이션 이름

Join

- 공통 속성을 중심으로 2개의 릴레이션을 하나로 합쳐서 새로운 릴레이션을 만드는 연산
- 연산자의 기호는 그리스문자 \bowtie 를 사용함
- 표기 형식 : $R \bowtie_{\langle\text{키속성r=키속성s}\rangle} S$
 단, 키속성 r은 릴레이션 R의 속성이고, 키속성 s는 릴레이션 S의 속성임

Division

- $X \supset Y$ 인 2개의 릴레이션에서 R(X)와 S(Y)가 있을 때, R의 속성이 S의 속성값을 모두 가진 튜플에서 S가 가진 속성을 제외한 속성만을 구하는 연산임
- 표기 형식 : $R \div_{\langle\text{속성r} \div \text{속성s}\rangle} S$
 단, 속성 r은 관계 R의 속성이고 속성 s는 관계 S의 속성이며, 속성 r과 속성 s는 동일 속성값을 가지는 속성이어야 함



정규화 단계 암기 요령

정규화라는 출소자가 말했다.

두부이겨다줘=도부이겔다조

도 메인이 원자값

부 분적 함수 종속 제거

이 행적 함수 종속 제거

결 정자이면서 후보키가 아닌 것 제거

다 치 종속 제거

조 인 종속성 이용

핵심 028 | SQL의 분류

DDL(데이터 정의어)

- SCHEMA, DOMAIN, TABLE, VIEW, INDEX를 정의하거나 변경 또는 삭제할 때 사용하는 언어
- 데이터베이스 관리자나 데이터베이스 설계자가 사용함
- 데이터 정의어(DDL)의 3가지 유형

명령어	기능
CREATE	Schema, Domain, Table, View, Index를 정의함
ALTER	Table에 대한 정의를 변경하는 데 사용함
DROP	Schema, Domain, Table, View, Index를 삭제함

DML(데이터 조작어)

- 데이터베이스 사용자가 응용 프로그램이나 질의어를 통하여 저장된 데이터를 실질적으로 처리하는데 사용하는 언어
- 데이터베이스 사용자와 데이터베이스 관리 시스템 간의 인터페이스 제공
- 데이터 조작어(DML)의 4가지 유형

명령어	기능
SELECT	테이블에서 조건에 맞는 튜플을 검색함
INSERT	테이블에 새로운 튜플을 삽입함
DELETE	테이블에서 조건에 맞는 튜플을 삭제함
UPDATE	테이블의 조건에 맞는 튜플의 내용을 변경함

DCL(데이터 제어어)

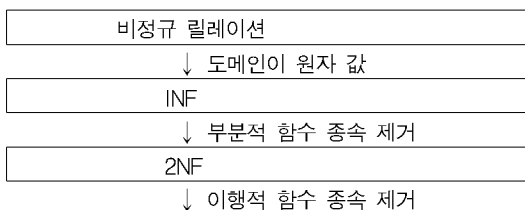
- 데이터의 보안, 무결성, 데이터 회복, 병행 수행 제어 등을 정의하는 데 사용하는 언어
- 데이터베이스 관리자가 데이터 관리를 목적으로 사용함
- 데이터 제어어(DCL)의 종류

핵심 026 | Anomaly(이상)의 개념 및 종류

이상(Anomaly) : 정규화(Normalization)를 거치지 않은 데이터베이스 내에 데이터들이 불필요하게 중복되어 릴레이션 조작시 발생하는 예기치 못한 곤란한 현상

삽입 이상 (Insertion Anomaly)	릴레이션에 데이터를 삽입할 때 의도와는 상관없이 원하지 않은 값들도 함께 삽입되는 현상
삭제 이상 (Deletion Anomaly)	릴레이션에서 한 튜플을 삭제할 때 의도와는 상관없는 값들도 함께 삭제되는 연쇄 삭제 현상
갱신 이상 (Update Anomaly)	릴레이션에서 튜플에 있는 속성값을 갱신할 때 일부 튜플의 정보만 갱신되어 정보에 모순이 생기는 현상

핵심 027 | 정규화 과정



명령어	기능
COMMIT	데이터베이스 조작 작업이 정상적으로 완료되었음을 관리자에게 알려줌
ROLLBACK	데이터베이스 조작 작업이 비정상적으로 종료되었을 때 원래의 상태로 복구함
GRANT	데이터베이스 사용자에게 사용권한을 부여함
REVOKE	데이터베이스 사용자의 사용권한을 취소함

DELETE
 FROM 테이블명
 WHERE 조건;

- 모든 레코드를 삭제할 때는 WHERE절을 생략함
- 모든 레코드를 삭제하더라도 테이블 구조는 남아 있기 때문에 디스크에서 테이블을 완전히 제거하는 DROP과는 다름

갱신문 (Update ~ Set ~)

UPDATE 테이블명
 SET 속성명 = 데이터[, 속성명=데이터]
 WHERE 조건;

핵심 029 | Select 문

테이블을 구성하는 튜플(행)들 중에서 전체 또는 조건을 만족하는 튜플(행)을 검색하여 주기억장치상에 임시 테이블로 구성시키는 명령문
일반 형식

```
SELECT predicate [테이블명]속성명1, [테이블명]속성명2,...
FROM 테이블명1, 테이블명2,...
[WHERE 조건]
[GROUP BY 속성명1, 속성명2,...]
[HAVING 조건]
[ORDER BY 속성명 [ASC | DESC]];
```

① SELECT절

- Predicate : 불러올 튜플수를 제한할 명령어를 기술함
 - ALL : 모든 튜플을 검색할 때 지정하는 것으로, 주로 생략함
 - DISTINCT : 중복된 튜플이 있으면 그 중 첫번째 한 개만 검색함
 - DISTINCTROW : 중복된 튜플을 검색하지만 선택된 속성의 값이 아닌, 튜플 전체를 대상으로 함
- 속성명 : 검색하여 불러올 속성(열) 및 수식들을 지정함
 - 기본 테이블을 구성하는 모든 속성을 지정할 때는 '*' 를 기술함
 - 두 개 이상의 테이블을 대상으로 검색할 때는 반드시 테이블명.속성명으로 표현해야 함

② FROM절 : 질의에 의해 검색될 데이터들을 포함하는 테이블명을 기술함

③ WHERE절 : 검색할 조건을 기술

④ GROUP BY절 : 특정 속성을 기준으로 그룹화하여 검색할 때 그룹화할 속성을 지정함. 일반적으로 GROUP BY절은 그룹함수와 함께 사용됨

⑤ HAVING절 : GROUP BY와 함께 사용되며, 그룹에 대한 조건을 지정함

⑥ ORDER BY절 : 특정 속성을 기준으로 정렬하여 검색할 때 사용함

- 속성명 : 정렬의 기준이 되는 속성명을 기술함
- [ASC|DESC] : 정렬방식으로 'ASC'는 오름차순, 'DESC'는 내림차순임
생략하면 오름차순으로 지정됨

핵심 030 | 삽입, 삭제, 갱신문

삽입문(Insert Into ~)

```
INSERT
INTO 테이블명(속성명1, 속성명2,...)
VALUES (데이터1, 데이터2,...);
```

- 대응하는 속성과 데이터는 개수와 data_type이 일치해야 함
 - 기본 테이블의 모든 속성을 사용할 때는 속성명을 생략할 수 있음
 - SELECT문을 사용하여 다른 테이블의 검색 결과를 삽입할 수 있음
- 삭제문 (Delete From ~)**

핵심 031 | 내장 SQL의 특징

- **내장 SQL** : 응용 프로그램 내에 SQL문장을 내포하여 프로그램이 실행될 때 함께 실행되도록 호스트 프로그램 언어에 삽입된 SQL
- 내장 SQL 실행문은 호스트 언어에서 실행문이 나타날 수 있는 곳이면 프로그램의 어느 곳에서나 사용할 수 있음
- 일반 SQL문은 수행 결과로 여러 개의 튜플을 반환하는 반면, 내장 SQL은 단 하나의 튜플만을 반환함
- 내장 SQL문에 의해 반환되는 튜플은 일반 변수를 사용하여 저장할 수 있음
- Host Program의 컴파일시 선행처리기에 의해 내장 SQL 문은 분리되어 컴파일됨
- 호스트 변수와 데이터베이스 필드의 이름은 같아도 됨
- 내장 SQL 문의 호스트 변수의 데이터 타입은 이에 대응하는 데이터베이스 필드의 SQL 데이터 타입과 일치하여야 함
- 삽입 SQL 문이 실행되면 SQL 실행의 상태가 SQL 상태 변수에 전달됨
- **호스트 언어의 실행문과 구분시키는 방법**
 - 명령문의 구분
 - C/C++에서 내장 SQL문은 \$와 세미콜론(;) 문자 사이에 기술함
 - Visual BASIC에서는 내장 SQL문 앞에 'EXEC SQL'을 기술함
 - 변수의 구분 : 내장 SQL에서 사용하는 호스트 변수는 변수 앞에 콜론(:) 문자를 붙임

핵심 032 | 시스템 카탈로그

- 시스템 그 자체에 관련이 있는 다양한 객체에 관한 정보를 포함하는 시스템 데이터베이스
- 데이터베이스에 포함되는 모든 데이터 객체에 대한 정의나 명세에 관한 정보를 유지 관리하는 시스템 테이블
- 데이터 정의어의 결과로 구성되는 기본 테이블, 뷰, 인덱스, 패키지, 접근권한 등의 데이터베이스 구조 및 통계 정보를 저장함
- 카탈로그들이 생성되면 자료 사전(Data Dictionary)에 저장되기 때문에 좁은 의미로는 카탈로그를 자료사전이라고도 함
- 카탈로그에 저장된 정보를 메타 데이터(Meta-Data)라고 함
- **카탈로그의 특징**
 - 카탈로그 자체도 시스템 테이블로 구성되어 있어 일반 이용자로 SQL을 이용하여 내용을 검색해 볼 수 있음
 - INSERT, DELETE, UPDATE문으로 갱신하는 것은 허용하지 않음

- DBMS가 스스로 생성하고, 유지함
- 카탈로그는 사용자가 SQL문을 실행시켜 기본 테이블, 뷰, 인덱스 등에 변화를 주면 시스템이 자동으로 갱신함

핵심 033 | 뷰(View)

- 사용자에게 접근이 허용된 자료만을 제한적으로 보여주기 위해 하나 이상의 기본 테이블로부터 유도된, 이름을 가지는 가상 테이블
- 권한이 있는 상태로 저장장치 내에 물리적으로 존재하지 않지만, 사용자에게는 있는 것처럼 간주됨

•뷰(View)의 특징

- 기본 테이블로부터 유도된 테이블이기 때문에 기본 테이블과 같은 형태의 구조를 가지며, 조작도 기본 테이블과 거의 같음
- 가상 테이블이기 때문에 물리적으로 구현되어 있지 않음
- 필요한 데이터만 뷰로 정의해서 처리할 수 있기 때문에 관리가 용이하고 명령문이 간단해짐
- 뷰를 통해서만 데이터에 접근하게 하면 뷰에 나타나지 않는 데이터를 안전하게 보호하는 효율적인 기법으로 사용할 수 있음
- 기본 테이블의 기본키를 포함한 속성(열) 집합으로 뷰를 구성해야만 삽입, 삭제, 갱신 연산이 가능함
- 정의된 뷰는 다른 뷰의 정의에 기초가 될 수 있음
- 하나의 뷰를 삭제하면 그 뷰를 기초로 정의된 다른 뷰도 자동으로 삭제됨

•뷰의 장점

- 논리적 데이터 독립성을 제공함
- 동일 데이터에 대해 동시에 여러 사용자의 상이한 응용이나 요구를 지원해줌
- 사용자의 데이터 관리를 간단하게 해줌
- 접근제어를 통한 자동 보안이 제공됨

•뷰의 단점

- 독립적인 인덱스를 가질 수 없음
- 뷰의 정의를 변경할 수 없음
- 뷰로 구성된 내용에 대한 삽입, 삭제, 갱신 연산에 제약이 따름

핵심 034 | 트랜잭션의 정의

- 데이터베이스에서 하나의 논리적 기능을 수행하기 위한 작업의 단위
- 데이터베이스 시스템에서 복구 및 병행 실행시 처리되는 작업의 논리적 단위
- 하나의 트랜잭션은 commit 되거나 rollback 됨
- 트랜잭션은 일반적으로 회복의 단위가 됨

핵심 035 | 트랜잭션의 특징

Atomicity (원자성)	<ul style="list-style-type: none"> ·트랜잭션의 연산은 데이터베이스에 모두 반영되든지 아니면 전혀 반영되지 않아야 함 ·트랜잭션 내의 모든 명령은 반드시 완벽히 수행되어야 하며, 모두가 완벽히 수행되지 않고 어느 하나라도 에러가 발생하면 트랜잭션 전부가 취소되어야 함
--------------------	--

Consistency (일관성)	<ul style="list-style-type: none"> ·트랜잭션이 그 실행을 성공적으로 완료하면 언제나 일관성 있는 데이터베이스 상태로 변환함 ·시스템이 가지고 있는 고정요소는 트랜잭션 수행 전과 트랜잭션 수행 완료 후의 상태가 같아야 함
Isolation (독립성, 격리성)	<ul style="list-style-type: none"> ·둘 이상의 트랜잭션이 동시에 병행 실행되는 경우 어느 하나의 트랜잭션 실행 중에 다른 트랜잭션의 연산이 끼어들 수 없음 ·수행 중인 트랜잭션은 완전히 완료될 때 까지 다른 트랜잭션에서 수행결과를 참조할 수 없음
Durability (영속성, 지속성)	성공적으로 완료된 트랜잭션의 결과는 영구적으로 반영되어야 함

핵심 036 | Commit, Rollback 연산

- COMMIT 연산** : 한 작업의 논리적 단위(트랜잭션)가 성공적으로 끝났고 데이터베이스가 다시 일관된 상태에 있으며, 이 트랜잭션이 행한 갱신 연산이 완료된 것을 트랜잭션 관리자에게 알려주는 연산
- ROLLBACK 연산** : 하나의 트랜잭션 처리가 비정상적으로 종료되어 데이터베이스의 일관성을 깨뜨렸을 때, 이 트랜잭션의 일부가 정상적으로 처리되었다라도 트랜잭션의 원자성을 구현하기 위해 이 트랜잭션이 행한 모든 연산을 취소(UNDO)시키는 연산으로 해당 트랜잭션을 재시작하거나 폐기함

핵심 037 | 트랜잭션의 상태

- Active(활동)** : 트랜잭션이 실행중에 있는 상태
- Failed(장애)** : 트랜잭션 실행에 오류가 발생하여 중단된 상태
- Aborted(철회)** : 트랜잭션이 비정상적으로 종료되어 ROLLBACK 연산을 수행한 상태
- Partially Committed(부분완료)** : 트랜잭션의 마지막 연산까지 실행했지만, COMMIT 연산이 실행되기 직전의 상태
- Committed(완료)** : 트랜잭션이 성공적으로 종료되어 COMMIT 연산을 실행한 후의 상태

핵심 038 | 회복(Recovery)의 개념

- 회복** : 트랜잭션들을 수행하는 도중 장애가 발생하여 데이터베이스가 손상되었을 때 손상되기 이전의 정상 상태로 복구시키는 작업
- 장애의 유형**
 - 트랜잭션 장애 : 입력 데이터 오류, 불명확한 데이터, 시스템 자원 요구의 과다 등 트랜잭션 내부의 비정상적인 상황으로 인하여 프로그램 실행이 중지되는 현상
 - 시스템 장애 : 데이터베이스에 손상을 입히지는 않으나 하드웨어 오동작, 소프트웨어의 손상, 교착 상태 등에 의해 모든 트랜잭션의 연속적인 수행에 장애를 주는 현상
 - 미디어 장애 : 저장장치인 디스크 블록의 손상이나 디스크 헤드의 충돌 등에 의해 데이터베이스의 일부 또는 전부가 물리적으로 손상된 상태
- 회복 관리기(Recovery Management)**
 - DBMS의 구성요소
 - 트랜잭션 실행이 성공적으로 완료되지 못하면 트랜잭션이 데이터베이스에 만들었던 모든 변화를 취소(UNDO)시키고, 트랜잭션 수행 이전의 원래 상태로 복구하는 역할을 담당함
 - 메모리 덤프, 로그(Log)를 이용하여 수행함

핵심 039 | 병행제어

- 다중프로그램의 이점을 활용하여 동시에 여러 개의 트랜잭션을 병행 수행 시킬 때, 동시에 실행되는 트랜잭션들이 데이터베이스의 일관성을 파괴하지 않도록 트랜잭션간의 상호작용을 제어하는 것

•병행제어의 목적

- 데이터베이스의 공유 최대화
- 시스템의 활용도 최대화
- 데이터베이스의 일관성 유지
- 사용자에 대한 응답시간 최소화

•병행수행의 문제점

문제점	의미
갱신 분실 (Lost Update)	2개 이상의 트랜잭션이 같은 자료를 공유하여 갱신할 때 갱신 결과의 일부가 없어지는 현상
비완료 의존성 (Uncommitted Dependency)	·임시 갱신 이라고도 함 ·하나의 트랜잭션 수행이 실패한 후 회복되기 전에 다른 트랜잭션이 실패한 갱신 결과를 참조하는 현상
모순성 (Inconsistency)	·불일치 분석(Inconsistent Analysis)이라고도 함 ·두 개의 트랜잭션이 병행수행 될 때 원치 않는 자료를 이용함으로써 발생하는 문제
연쇄 복귀 (Cascading Rollback)	병행수행되던 트랜잭션을 중 어느 하나에 문제가 생겨 ROLLBACK 하는 경우 다른 트랜잭션도 함께 ROLLBACK 되는 현상

위치 투명성 (Location Transparency)	액세스하려는 데이터베이스의 실제 위치를 알 필요없이 단지 데이터베이스의 논리적인 명칭만으로 액세스 할 수 있음
중복 투명성 (Replication Transparency)	동일 데이터가 여러 곳에 중복되어 있더라도 사용자는 마치 하나의 데이터만 존재하는 것처럼 사용하고, 시스템은 자동으로 여러 자료에 대한 작업을 수행함
병행 투명성 (Concurrency Transparency)	분산 데이터베이스와 관련된 다수의 트랜잭션들이 동시에 실행되더라도 그 트랜잭션의 결과는 영향을 받지 않음
장애 투명성 (Failure Transparency)	트랜잭션, DBMS, 네트워크, 컴퓨터 장애에도 불구하고 트랜잭션을 정확하게 처리함

핵심 042 | 분산 데이터베이스의 장.단점

장점	단점
<ul style="list-style-type: none"> ·지역 자치성이 높음 ·자료의 공유성이 향상됨 ·분산제어가 가능함 ·시스템 성능이 향상됨 ·효율성과 융통성이 높음 ·신뢰성 및 가용성이 높음 ·점증적 시스템 용량 확장이 용이함 	<ul style="list-style-type: none"> ·DBMS가 수행할 기능이 복잡함 ·데이터베이스 설계가 어려움 ·소프트웨어 개발 비용이 증가함 ·처리 비용이 증가함 ·잠재적 오류가 증가함

핵심 043 | 자료구조의 분류

- 선형구조 : 선형리스트, 연결리스트, 스택, 큐, 데크
- 비선형 구조 : 트리, 그래프

핵심 044 | Stack

- 리스트의 한쪽 끝으로만 자료의 삽입, 삭제 작업이 이루어지는 자료구조
- 가장 나중에 삽입된 자료가 가장 먼저 삭제되는 후입선출(LIFO: Last-In, First-Out) 방식으로 자료를 처리함
- TOP : Stack으로 할당된 기억공간에 가장 마지막으로 삽입된 자료가 기억된 공간을 가리키는 요소, 스택 포인터라고도 함
- Bottom : 스택의 가장 밑바닥임
- Stack의 용도
 - 부프로그램 호출시 복귀주소를 저장할 때
 - 인터럽트가 발생하여 복귀주소를 저장할 때
 - 후위표기법(Postfix Notation)으로 표현된 산술식을 연산할 때
 - 0 주소 지정방식 명령어의 자료지장소
 - 재귀(RECURSIVE)프로그램의 순서제어
 - 컴파일러를 이용한 언어번역시

핵심 045 | 큐(Queue)

- 선형 리스트의 한쪽에서는 삽입작업이 이루어지고 다른 쪽에서는 삭제작업이 이루어지도록 구성된 자료구조
- 가장 먼저 삽입된 자료가 가장 먼저 삭제되는 선입선출(FIFO: First-In, First-Out) 방식으로 처리함
- Queue를 이용하는 예
 - 창구업무처럼 서비스 순서를 기다리는 등의 대기행렬의 처리에 사용함
 - 운영체제의 작업 스케줄링에 사용함

핵심 040 | 암호화 기법

개인키 암호방식(Private Key Encryption)=비밀키 암호방식

- 동일한 키로 데이터를 암호화 하고 복호화 함
- 데이터베이스 생성자는 평문의 정보 M을 암호화 알고리즘 E와 공용 키 K를 이용하여 암호문 C로 바꾸어 저장시켜놓고, 사용자가 그 데이터베이스에 접근하려면 복호화 알고리즘 D와 공용키 K를 이용하여 평문의 정보로 바꾸어 이용하는 방법임
- 대칭 암호 방식 또는 단일키 암호화 기법이라고도 함
- 비밀키는 제3자에게는 노출시키지 않고 데이터베이스 사용 권한이 있는 사용자만 나누어 가짐
- 종류 : 전위 기법, 대체기법, 대수기법, 합성기법(DES, LUGIFER)
- 장점 : 암호화/복호화 속도가 빠르며, 알고리즘이 단순하고 파일 크기가 작음
- 단점 : 사용자의 증가에 따라 관리해야 할 키의 수가 상대적으로 많아짐

공개키 암호 방식(Public key encryption)

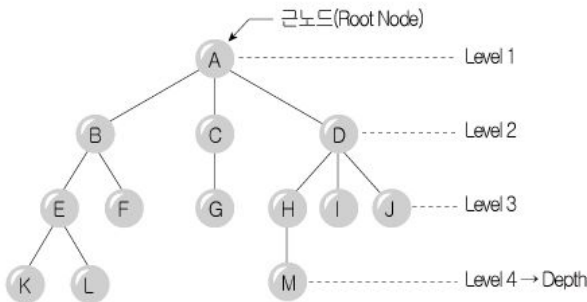
- 서로 다른 키로 데이터를 암호화하고 복호화 함
- 데이터를 암호화할 때 사용하는 키(공개키, Public key)는 데이터베이스 사용자에게 공개하고, 복호화할 때의 키(비밀키, Secret key)는 관리자가 비밀리에 관리하는 방법
- 비대칭 암호방식이라고도 하며, 대표적으로 RSA(Rivest Shamir Adleman)가 있음
- 장점 : 키의 분배가 용이하고, 관리해야 할 키의 개수가 적음
- 단점 : 암호화/복호화 속도가 느리며, 알고리즘이 복잡하고 파일 크기가 크다.

핵심 041 | 분산 데이터베이스의 목표

핵심 046 | 데크(Deque)

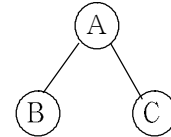
- 삽입과 삭제가 리스트의 양쪽 끝에서 모두 발생할 수 있는 자료구조
- Stack과 Queue의 장점만 따서 구성한 것임
- 입력이 한쪽에서만 발생하고 출력은 양쪽에서 일어날 수 있는 입력 제한과, 입력은 양쪽에서 일어나고 출력은 한곳에서만 이루어지는 출력제한이 있음
- 입력 제한 데크 : Scroll
- 출력 제한 데크 : Shelf

핵심 047 | 트리(Tree) 관련 용어

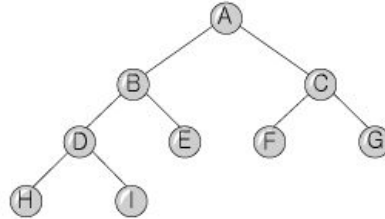


- 노드(Node) : 하나의 자료항목과 다른 항목에 대한 가지(Branch)를 합친 것
(예) A, B, C, D, E, F, G, H, I, J, K, L, M
- 근노드(Root Node) : 트리의 맨 위에 있는 노드 (예) A
- 디그리(Degree, 차수) : 각 노드에서 뻗어 나온 가지의 수
(예) A=3, B=2, C=1, D=3
- 트리의 디그리 : 노드들의 디그리 중에서 가장 많은 수
(예) 노드 A나 D가 3개의 디그리를 가지므로 위 트리의 디그리는 3임
- 단말노드(Terminal Node)=잎노드(Leaf Node) : 자식이 하나도 없는 노드, 즉 Degree가 0인 노드
(예) K, L, F, G, M, I, J
- 비단말 노드(Non-Terminal Node) : 자식이 하나라도 있는 노드, 즉 Degree가 0이 아닌 노드
(예) A, B, C, D, E, H
- 자노드(Son Node) : 어떤 노드에 연결된 다음 레벨의 노드들
(예) D의 자노드 : H, I, J
- 부노드(Parent Node) : 어떤 노드에 연결된 이전 레벨의 노드
(예) E, F의 부모는 B
- 형제노드(Brother Node, Sibling) : 동일한 부모를 갖는 노드들
(예) H의 형제노드는 I, J
- Level : 근노드의 Level을 1로 가정한 후 어떤 Level이 L이면 자식 노드는 L+1
(예) H의 레벨은 3
- 깊이(Depth, Height) : 어떤 Tree에서 노드가 가질 수 있는 최대의 레벨
(예) 위 트리의 깊이는 4

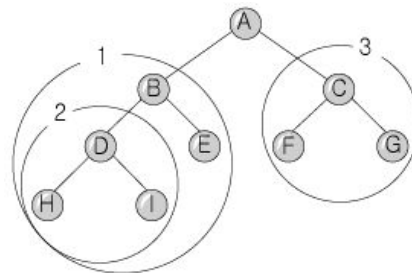
핵심 048 | 이진트리의 운행법(Traversal)



- Preorder 운행 : Root → Left → Right 순으로 운행함 A B C
- Inorder 운행 : Left → Root → Right 순으로 운행함 B A C
- Postorder 운행 : Left → Right → Root 순으로 운행함 B C A
(예) 다음 트리를 Inorder, Preorder, Postorder 방법으로 운행했을 때 각 노드를 방문한 순서는?



Preorder 운행법의 방문순서



- 서브 트리를 하나의 노드로 생각할 수 있도록 위 그림과 같이 서브 트리단 위로 묶습니다. Preorder, Inorder, Postorder 모두 공통으로 사용함
- Preorder는 Left → Root → Right이므로 A130이 됨
- 1은 B2E이므로 AB2E30이 됨
- 2는 DH1이므로 ABDH1E30이 됨
- 3은 CFG이므로 ABDH1ECFG가 됨 방문순서 : ABDH1ECFG

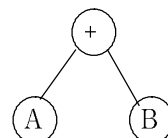
Inorder 운행법의 방문순서

- Inorder는 Left → Root → Right 이므로 1A30이 됨
- 1은 2BE이므로 2BEA30이 됨
- 2는 HD1이므로 HD1BEA30이 됨
- 3은 FCG이므로 HD1BEAFCG가 됨 방문순서 : HD1BEAFCG

Postorder

- Postorder는 Left → Right → Root이므로 13A가 됨
- 1은 2EB이므로 2EB3A가 됨
- 2는 HD1이므로 HD1EB3A가 됨
- 3은 FCG이므로 HD1EBFGCA가 됨 방문순서 : HD1EBFGCA

핵심 049 | 수식의 표기법



- 전위표기법(PreFix) : 연산자 → Left → Right, +AB

- 중위표기법(InFix) : Left → 연산자 → Right, A+B
- 후위표기법(PostFix) : Left → Right → 연산자, AB+

핵심 050 | 정렬 방식

내부정렬

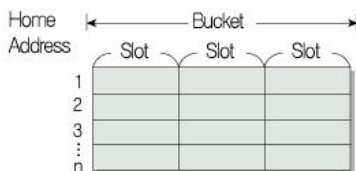
- 소량의 데이터에 대하여 주기억장치 내에만 기억시켜서 정렬하는 방식
- 종류 : 히프정렬, 삽입정렬, 버블정렬, 선택정렬, 퀵정렬, 2-Way Merge Sort, 기수정렬(=Radix Sort)

외부정렬

- 대량의 데이터에 대하여 보조기억장치에 기억시켜서 정렬하는 방식으로, 대부분 합병정렬(Merge Sort) 기법으로 처리
- 종류 : 밸런스 병합정렬, 캐스케이드 병합정렬, 폴리파즈 병합정렬, 오실레이팅 병합정렬

핵심 051 | 해싱 (Hashing)

- Hash Table이라는 기억공간을 할당하고, 해시함수(Hash Function)를 이용하여 레코드 키에 대한 Hash Table 내의 Home Address를 계산한 후 주어진 레코드를 해당 기억장소에 저장하거나 검색작업을 수행하는 방식
- DAM(직접접근) 파일을 구성할 때 해싱이 사용되며, 접근 속도는 빠르나 기억공간이 많이 요구됨
- 검색 속도가 가장 빠름
- 삽입, 삭제 작업의 빈도가 많을 때 유리한 방식
- 해시 테이블(Hash Table)
- 레코드를 1개 이상 보관할 수 있는 Home Bucket들로 구성된 기억공간으로, 보조기억장치에 구성할 수도 있고 주기억장치에 구성할 수도 있음



- 버킷(bucket)** : 하나의 주소를 갖는 파일의 한 구역을 의미하며, 버킷의 크기는 같은 주소에 포함될 수 있는 레코드 수를 의미함
- 슬롯(slot)** : 한 개의 레코드를 저장할 수 있는 공간으로 n개의 슬롯이 모여 하나의 버킷을 형성함
- Collision(충돌현상)** : 서로 다른 두 개 이상의 레코드가 같은 주소를 갖는 현상
- Synonym** : 같은 Home Address를 갖는 레코드들의 집합
- Overflow** : 계산된 Home Address의 Bucket 내에 저장할 기억공간이 없는 상태(Bucket을 구성하는 Slot이 여러 개일 때는 Collision은 발생해도 Overflow는 발생하지 않을 수 있음)

핵심 052 | 순차 파일(Sequential File) = 순서 파일

- 입력되는 데이터들을 논리적인 순서에 따라 물리적 연속 공간에 순차적으로 기록하는 방식
- 급여 관리 등과 같이 변동 사항이 크지 않고 기간 별로 일괄 처리를

주로 하는 경우에 적합함

- 주로 순차 접근이 가능한 자기 테이프에서 사용됨

순차 파일의 장점

- 기록 밀도가 높아 기억 공간을 효율적으로 사용할 수 있음
- 매체 변환이 쉬워 어떠한 매체에도 적용할 수 있음
- 레코드를 기록할 때 사용한 키 순서대로 레코드를 처리하는 경우, 다른 편성법보다 처리 속도가 빠르다.

순차 파일의 단점

- 파일에 새로운 레코드를 삽입·삭제하는 경우 파일 전체를 복사해야 하므로 시간이 많이 소요됨
- 데이터 검색시 처음부터 순차적으로 하기 때문에 검색 효율이 낮음

핵심 053 | 색인 순차 파일(Indexed Sequential File)

- 순차 처리와 랜덤 처리가 모두 가능하도록 레코드들을 키 값순으로 정렬(Sort)시켜 기록하고, 레코드의 키 항목만을 모은 색인을 구성하여 편성하는 방식
- 색인을 이용한 순차적인 접근 방법을 제공하여 ISAM(Index Sequential Access Method)라고도 함
- 레코드를 참조하는 경우 색인을 탐색한 후 색인이 가리키는 포인터(주소)를 사용하여 직접 참조할 수 있음
- 일반적으로 자기 디스크에 많이 사용되며, 자기 테이프에서는 사용할 수 없음
- 색인 순차 파일의 구성**
 - 기본 구역(Prime Area) : 실제 레코드들을 기록하는 부분으로, 각 레코드는 키 값순으로 저장됨
 - 색인 구역(Index Area) : 기본 구역에 있는 레코드들의 위치를 찾아가는 색인이 기록되는 부분으로, 트랙 색인 구역, 실린더 색인 구역, 마스터 색인 구역으로 구분할 수 있음
 - 오버플로우 구역(Overflow Area) : 기본 구역에 빈 공간이 없어서 새로운 레코드의 삽입이 불가능할 때를 대비하여 예비적으로 확보해둔 부분임

실린더 오버플로우 구역(Cylinder Overflow Area)	각 실린더마다 만들어지는 오버플로우 구역으로, 해당 실린더의 기본 구역에서 오버플로우된 데이터를 기록함
독립 오버플로우 구역(Independent Overflow Area)	실린더 오버플로우 구역에 더 이상 오버플로우된 데이터를 기록할 수 없을 때 사용할 수 있는 예비 공간으로, 실린더 오버플로우 구역과는 별도로 만들어진다.

핵심 054 | VSAM 파일

- 동적 인덱스 방법을 이용한 색인 순차 파일
- 제어 구간, 제어 구역, 순차 세트, 인덱스 세트로 구성됨

제어 구간(Control Interval)	데이터 레코드가 저장되는 부분
제어 구역(Control Area)	몇 개의 제어 구간을 모아놓은 것
순차 세트(Sequence Set)	제어 구역에 대한 인덱스를 저장한 것
인덱스 세트(Index Set)	순차 세트의 상위 인덱스








- 기본 구역과 오버플로우 구역을 구분하지 않음
- 레코드를 삭제하면 그 공간을 재사용할 수 있음
- 제어 구간에 가변 길이 레코드를 쉽게 수용할 수 있음


2과목 · 전자계산기 구조

핵심 055 | 불 대수의 기본 공식

- 교환법칙 : $A+B=B+A$, $AB=BA$
- 결합법칙 : $A+(B+C)=(A+B)+C$, $A(BC)=(AB)C$
- 분배법칙 : $A(B+C)=AB+AC$, $A+BC=(A+B)(A+C)$
- 역등법칙 : $A+A=A$, $AA=A$
- 보수법칙 : $A+A'=1$, $AA'=0$
- 항등법칙 : $A+0=A$, $A+1=1$, $A0=0$, $A1=A$
- 콘센서스 : $AB+BC+CA=AB+CA$, $(A+B)(B+C)(C+A)=(A+B)(C+A)$
- 드모르강 : $A+B'=(A'B)'$, $A'B=(A+B)'$
- 복원법칙 : $A''=A$

핵심 056 | 논리 게이트

게이트	기호	의미	진리표	논리식															
AND		입력신호가 모두 1일 때 1 출력	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	0	0	1	0	1	0	0	1	1	1	$Y=A \cdot B$ $Y=AB$
A	B	Y																	
0	0	0																	
0	1	0																	
1	0	0																	
1	1	1																	
OR		입력신호 중 1 개만 1이어도 1 출력	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	1	$Y=A+B$
A	B	Y																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	1																	
NOT		입력된 정보를 반대로 변환하 여 출력	<table><tr><th>A</th><th>Y</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	Y	0	1	1	0	$Y=A'$ $Y=\bar{A}$									
A	Y																		
0	1																		
1	0																		
BUFFER		입력된 정보를 그대로 출력	<table><tr><th>A</th><th>Y</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	A	Y	0	0	1	1	$Y=A$									
A	Y																		
0	0																		
1	1																		
NAND		NOT + AND, 즉 AND의 부정	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	Y	0	0	1	0	1	1	1	0	1	1	1	0	$Y=\overline{A \cdot B}$
A	B	Y																	
0	0	1																	
0	1	1																	
1	0	1																	
1	1	0																	
NOR		NOT + OR, 즉 OR의 부정	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	0	$Y=\overline{A+B}$
A	B	Y																	
0	0	1																	
0	1	0																	
1	0	0																	
1	1	0																	
XOR		입력되는 값이 모두 같으면 0, 한개라도 틀리 면 1출력	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	0	$Y=A \oplus B$ $Y=A'B+AB'$ $Y=(A+B)(A'+B')$ $Y=(A+B)(AB)'$
A	B	Y																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	0																	

XNOR		NOT + XOR, 즉 XOR의 부정	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	1	$Y=A \odot B$ $Y=X \oplus Y'$ $Y=AB+A'B'$ $Y=(A'+B)(A+B')$ $Y=(AB)(A+B)'$
			A	B	Y														
			0	0	1														
			0	1	0														
			1	0	0														
1	1	1																	

핵심 057 | 반가산기 (HA ; Half Adder)

1Bit짜리 2진수 2개를 덧셈한 합(S)과 자리올림 수(C)를 구하는 회로

0	0	1	1	A
+ 0	+ 1	+ 0	+ 1	+ B
0 0	0 1	0 1	1 0	C S

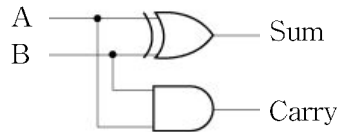
· S : 합
· C : 자리 올림

진리표

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

논리식 : $C=AB$ $S=A \oplus B = A \oplus B$

논리회로



핵심 058 | 전가산기 (FA ; Full Adder)

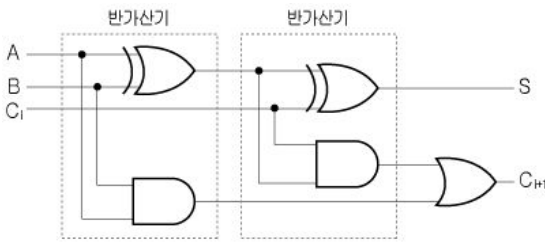
자리 올림수(C_i)를 포함하여 1Bit 크기의 2진수 3자리를 더하여 합(S_i)과 자리 올림수(C_{i+1})를 구하는 회로

논리식

$$\begin{aligned}
 C_{i+1} &= A'BC_i + AB'C_i + ABC_i \\
 &= (A'B+AB')C_i + ABC_i \\
 &= (A \oplus B)C_i + AB \leftarrow A'B+AB' = A \oplus B, C_i' + C_i = 1 \\
 S_i &= A'B'C_i + A'BC_i' + AB'C_i' + ABC_i \\
 &= (A'B'+AB)C_i + (A'B+AB')C_i' \\
 &= (A \oplus B)'C_i + (A \oplus B)C_i' \leftarrow A'B'+AB = (A \oplus B)', A'B+AB' = A \oplus B \\
 &= (A \oplus B) \oplus C_i \leftarrow A \oplus B \text{를 } X \text{라 하면 } X'C_i + XC_i' = X \oplus C_i \\
 C_i &= (A \oplus B) \oplus C_i
 \end{aligned}$$

회로

전가산기는 2개의 반가산기(HA)와 1개의 OR Gate로 구성된다.



핵심 059 | 자료 구성의 단위

비트(Bit, Binary Digit)	<ul style="list-style-type: none"> · 자료(정보) 표현의 최소 단위 · 두 가지 상태(0과 1)를 표시하는 2진수 1자리
니블(Nibble)	<ul style="list-style-type: none"> · 4개의 비트(Bit)가 모여 1개의 Nibble을 구성함 · 4비트로 구성되며 16진수 1자리를 표현하기에 적합함
바이트(Byte)	<ul style="list-style-type: none"> · 문자를 표현하는 최소 단위로, 8개의 비트(Bit)가 모여 1Byte를 구성함 · 1Byte는 $256(2^8)$가지의 정보를 표현할 수 있음 · 주소 지칭의 단위로 사용됨
워드(Word)	<ul style="list-style-type: none"> · CPU가 한 번에 처리할 수 있는 명령 단위 · 반워드(Half Word) : 2Byte · 풀워드(Full Word) : 4Byte · 더블워드(Double Word) : 8Byte
필드(Field)	<ul style="list-style-type: none"> · 파일 구성의 최소 단위 · 의미 있는 정보를 표현하는 최소 단위
레코드(Record)	<ul style="list-style-type: none"> · 하나 이상의 관련된 필드가 모여서 구성됨 · 컴퓨터 내부의 자료 처리 단위로서, 일반적으로 레코드는 논리 레코드(Logical Record)를 의미함
블록(Block) 물리 레코드(Physical Record)	<ul style="list-style-type: none"> · 하나 이상의 논리 레코드가 모여서 구성됨 · 각종 저장매체와의 입·출력 단위를 의미하며, 일반적으로 물리 레코드(Physical Record)라고 함
파일(File)	프로그램 구성의 기본 단위로, 여러 레코드가 모여서 구성됨
데이터베이스(Database)	여러 개의 관련된 파일(File)의 집합

핵심 060 | 보수

컴퓨터가 기본적으로 수행하는 가산을 이용하여 뺄셈을 수행하기 위해 사용함

r의 보수	<ul style="list-style-type: none"> · 10진법에는 10의 보수가 있고, 2진법에는 2의 보수가 있음 · 보수를 구할 숫자의 자리 수만큼 0을 채우고 가장 왼쪽에 1을 추가하여 기준을 만듦 예) 33의 10의 보수는? $33+X=100 \rightarrow X=100-33 \rightarrow X=67$ 예) 10101의 2의 보수는? $10101+X=100000 \rightarrow X=100000-10101 \rightarrow X=01011$
r-1의 보수	<ul style="list-style-type: none"> · 10진법에는 9의 보수가 있고, 2진법에는 1의 보수가 있음 · 10진수 N에 대한 9의 보수는 주어진 숫자의 자리 수만큼 9를 채워 기준을 만듦 예) 33의 9의 보수는? $33+X=99 \rightarrow X=99-33 \rightarrow X=66$ · 2진수 N에 대한 1의 보수는 주어진 숫자의 자리 수만큼 1을 채워 기준을 만듦 예) 10101의 1의 보수는? $10101+X=11111 \rightarrow X=11111-10101 \rightarrow X=01010$

핵심 061 | 2진 연산

- 정수 값을 2진수로 변환하여 표현하는 방식
- 표현할 수 있는 범위가 작지만 연산 속도가 빠름

종류	표현방법	비고
부호화 절대치법 (Signed Magnitude)	양수표현에 대하여 부호 Bit의 값만 0을 1로 바꾼다	2가지 형태의 0 존재(+0, -0)
부호화 1의 보수법 (Signed 1's Complement)	양수 표현에 대하여 1의 보수를 취함	
부호화 2의 보수법 (Signed 2's Complement)	양수 표현에 대하여 2의 보수를 취함	한 가지 형태의 0 만 존재(+0)

표현범위

종류	범위	n=8	n=16	n=32
부호화 절대치법	$-2^{n-1}+1 \sim +2^{n-1}-1$	-127 ~ +127	-32767 ~ +32767	$-2^{31}+1 \sim +2^{31}-1$
부호화 1의 보수법	$-2^{n-1} \sim +2^{n-1}-1$	-128 ~ +127	-32768 ~ +32767	$-2^{31} \sim +2^{31}-1$
부호화 2의 보수법	$-2^{n-1} \sim +2^{n-1}-1$	-128 ~ +127	-32768 ~ +32767	$-2^{31} \sim +2^{31}-1$

핵심 062 | 자료의 외부적 표현

BCD(Binary Coded Decimal, 2진화 10진 코드)	<ul style="list-style-type: none"> · 6Bit 코드로 IBM에서 개발 · 1 개의 문자를 2개의 Zone 비트와 4개의 Digit 비트로 표현함 · 6Bit는 2^6개를 표현할 수 있으므로 64개의 문자를 표현할 수 있음 · 1Bit의 Parity Bit를 추가하여 7Bit로 사용함 · 영문 소문자를 표현하지 못함
ASCII 코드 (American Standard Code for Information Interchange)	<ul style="list-style-type: none"> · 7Bit 코드로 미국 표준협회에서 개발 · 1개의 문자를 3개의 Zone 비트와 4개의 Digit 비트로 표현함 · $2^7=128$가지의 문자를 표현할 수 있음 · 1Bit의 Parity Bit를 추가하여 8Bit로 사용함 · 통신 제어용 및 마이크로컴퓨터에서 사용함
EBCDIC(Extended BCD Interchange Code, 확장 2진화 10진 코드)	<ul style="list-style-type: none"> · 8Bit 코드로 IBM에서 개발 · 1개의 문자를 4개의 Zone 비트와 4개의 Digit 비트로 표현함 · $2^8=256$가지의 문자를 표현할 수 있음 · 1Bit의 Parity Bit를 추가하여 9Bit로 사용함 · 대형 기종의 컴퓨터에서 사용함

핵심 063 | 기타 자료의 표현방식

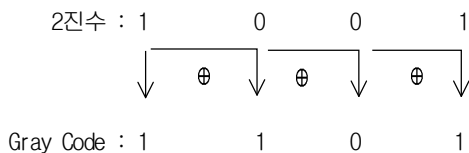
BCD 코드	<ul style="list-style-type: none"> · 10진수 1자리의 수를 2진수 4Bit로 표현함 · 4Bit의 2진수 각 Bit가 $8(2^3)$, $4(2^2)$, $2(2^1)$, $1(2^0)$의 자리 값을 가지므로 8421코드라고도 함 · 대표적인 가중치 코드 · 문자코드인 BCD에서 Zone 부분을 생략한 형태임 · 10진수 입·출력이 간편함
--------	--

Excess-3 코드 (3초과 코드)	<ul style="list-style-type: none"> BCD + 3, 즉 BCD 코드에 3(0011)을 더하여 만든 코드임 대표적인 자보수 코드이며, 비가중치 코드임
Gray 코드	<ul style="list-style-type: none"> BCD 코드의 인접하는 비트를 X-OR 연산하여 만든 코드 입출력장치, D/A변환기, 주변장치 등에서 숫자를 표현할 때 사용 1Bit만 변화시켜 다음 수치로 증가시키기 때문에 하드웨어적인 오류가 적음
패리티 검사 코드	<ul style="list-style-type: none"> 코드의 오류를 검사하기 위해서 데이터비트 외에 1Bit의 패리티 체크 비트를 추가하는 것으로 1Bit의 오류만 검출할 수 있음 Odd Parity : Odd 패리티는 코드에서 1인 Bit의 수가 홀수가 되도록 0이나 1을 추가함 Even Parity : Even 패리티는 코드에서 1인 Bit의 수가 짝수가 되도록 0이나 1을 추가함
해밍 코드	<ul style="list-style-type: none"> 오류를 스스로 검출하여 교정이 가능한 코드 1Bit의 오류만 교정할 수 있음 데이터 비트 외에 에러 검출 및 교정을 위한 잉여 비트가 많이 필요함 해밍코드 중 1, 2, 4, 8, 16 2^n 번째 비트는 오류 검출을 위한 패리티 비트임

핵심 064 | 그레이 코드 변환

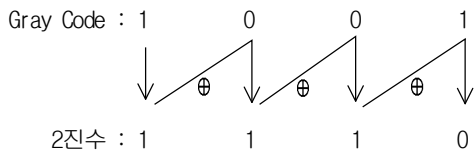
2진수를 Gray Code로 변환하는 방법

- 2진수의 첫 번째 비트는 그대로 내려쓴다.
 - 2번째 Gray Bit 부터는 변경할 2진수의 해당 번째 비트와 그 왼쪽의 비트를 X-OR 연산하여 씀
- (예) 2진수 1001을 Gray Code로 변환하시오.



Gray Code를 2진수로 변환하는 방법

- 그레이 코드의 첫 번째 비트는 그대로 내려쓴다.
 - 2번째 2진수 비트부터는 왼쪽에 구한 그레이 비트와 변경할 해당 번째 2진수 비트를 X-OR 연산하여 씀
- (예) Gray Code 1001을 2진수로 변환하시오.



핵심 065 | 코드의 분류

분류	코드 종류
가중치 코드 (Weight Code)	BCD(8421), 2421, 84-2-1, Biquinary(5043210), 51111, Ring-Counter(9876543210)
비가중치 코드 (Non-Weight Code)	3초과(Excess-3), Gray, Jonson, 2-out-of-5, 3-out-of-5

자보수 코드 (Self-Complement Code)	Excess-3, 2421, 51111, 84-2-1
오류 검출용 코드	해밍코드, 패리티 검사 코드, Biquinary, Ring-Counter, 2-out-of-5, 3-out-of-5

핵심 066 | 중앙처리장치의 구성 요소

제어 장치	<ul style="list-style-type: none"> 컴퓨터에 있는 모든 장치들의 동작을 지시하고 제어하는 장치 주 기억장치에서 읽어 들인 명령어를 해독하여 해당하는 장치에게 제어 신호를 보내 정확하게 수행하도록 지시함 프로그램 카운터(PC), 명령어 레지스터(IR), 부호기(제어신호 발생기, 명령어 해독기, 번지 해독기 등으로 구성되어 있음)
연산 장치	<ul style="list-style-type: none"> 제어장치의 명령에 따라 실제로 연산을 수행하는 장치 연산장치가 수행하는 연산에는 산술연산, 논리연산, 관계연산, 이동(Shift) 등이 있음 가산기, 누산기(AC ; Accumulator), 보수기, 데이터 레지스터, 오버플로우 검출기, Shift Register 등으로 구성되어 있음
레지스터	<ul style="list-style-type: none"> CPU 내부에서 처리할 명령어나 연산의 중간 결과값 등을 일시적으로 기억하는 임시 기억 장소 플립플롭(Flip-Flop)이나 래치(Latch)들을 병렬로 연결하여 구성함 메모리 중에서 가장 속도가 빠름

핵심 067 | 주요 레지스터

레지스터	기능
프로그램 카운터, 프로그램 계수기 (PC, Program Counter)	다음 번에 실행할 명령어의 번지를 기억하는 레지스터
명령 레지스터 (IR, Instruction Register)	현재 실행 중인 명령의 내용을 기억하는 레지스터
누산기 (AC, Accumulator)	연산된 결과를 일시적으로 저장하는 레지스터로 연산의 중심임
상태 레지스터 (Status Register) PSWR (Program Status Word Register)	<ul style="list-style-type: none"> 시스템 내부의 순간순간의 상태가 기록된 정보를 PSWR라고 함 오버플로, 언더플로, 자리올림, 인터럽트 등의 PSW를 저장하고 있는 레지스터
메모리 주소 레지스터 (MAR, Memory Address Register)	기억장치를 출입하는 데이터의 번지를 기억하는 레지스터
메모리 버퍼 레지스터 (MBR, Memory Buffer Register)	기억장치를 출입하는 데이터가 잠시 기억되는 레지스터
인덱스 레지스터 (Index Register)	주소의 변경이나 프로그램에서의 반복연산의 횟수를 계수하는 레지스터
데이터 레지스터 (Data Register)	연산에 사용될 데이터를 기억하는 레지스터
Shift Register	<ul style="list-style-type: none"> 저장된 값을 왼쪽 또는 오른쪽으로 1Bit씩 자리 이동시키는 레지스터 2배 길이 레지스터라고도 함
Major Status Register	CPU의 메이저 상태를 저장하고 있는 레지스터

핵심 068 | 버스

CPU, 메모리, I/O장치 등과 상호 필요한 정보를 교환하기 위해 연결하는 공동의 전송선

전송하는 정보에 따른 분류	<ul style="list-style-type: none"> · 번지 버스(Address Bus) : CPU가 메모리나 입출력 기기의 번지를 지정할 때 사용하는 단방향 전송선 · 자료 버스(Data Bus) : CPU와 메모리 또는 입출력 기기 사이에서 데이터를 전송하는 양방향 버스 · 제어 버스(Control Bus) : CPU의 현재 상태나 상태 변경을 메모리 또는 입출력에 알리는 제어신호를 전송하는 선
버스 위치에 따른 분류	<ul style="list-style-type: none"> · 내부 버스 : CPU 및 메모리 내에 구성된 Bus · 외부 버스 : 주변 입출력장치에 구성된 Bus

핵심 069 | 명령어의 구성

Operation Code, 연산자 부	자료부, Operand
-----------------------	--------------

연산자 부(Operation Code부)

- 수행해야 할 동작에 맞는 연산자를 표시함, 흔히 OP-Code부라고 함
- 연산자부의 크기(비트수)는 표현할 수 있는 명령의 종류를 나타내는 것으로, nBit면 최대 2ⁿ개의 명령어를 사용할 수 있음

주소부(Operand부)

- 실제 데이터에 대한 정보를 표시하는 부분임
- 기억장소의 주소, 레지스터 번호, 사용할 데이터 등을 표시함
- 주소부의 크기는 메모리의 용량과 관계가 있음

핵심 070 | 연산자(Operation Code)의 기능

합수 연산 기능	산술 연산 : ADD, SUB, MUL, DIV, 산술 Shift 등 논리 연산 : NOT, AND, OR, XOR, 논리적 Shift, Rotate, Complement, Clear 등
자료 전달 기능	<ul style="list-style-type: none"> · CPU와 기억장치 사이에서 정보를 교환하는 기능 · Load : 기억장치에 기억되어 있는 정보를 CPU로 꺼내오는 명령 · Store : CPU에 있는 정보를 기억장치에 기억시키는 명령 · Move : 레지스터간에 자료를 전달하는 명령 · Push : 스택에 자료를 저장하는 명령 · Pop : 스택에서 자료를 꺼내오는 명령
제어 기능	<ul style="list-style-type: none"> · 프로그래머가 명령어 실행 순서를 변경시키는 기능 · 무조건 분기 명령 : GOTO, Jump(JMP) 등 · 조건 분기 명령 : IF 조건, SPA, SNA, SZA 등 · Call : 부프로그램 호출 · Return : 부프로그램에서 메인 프로그램으로 복귀
입·출력 기능	<ul style="list-style-type: none"> · CPU와 I/O장치, 또는 메모리와 I/O장치 사이에서 자료를 전달하는 기능 · INPUT : 입출력 장치의 자료를 주기억장치로 입력하는 명령 · OUTPUT : 주기억 장치의 자료를 입출력 장치로 출력하는 명령

핵심 071 | 피연산자의 수에 따른 연산자의 분류

단항연산자 (Unary Operator)	NOT, Complement, Shift, Rotate, MOVE 등
이항연산자 (Binary Operator)	사칙연산, AND, OR, XOR, XNOR

핵심 072 | 연산

AND (Masking Operation)	<ul style="list-style-type: none"> · 특정 문자 또는 특정 Bit를 삭제(Clear)시키는 명령으로 Masking 명령이라고도 함 · 삭제할 부분의 Bit를 0과 AND시켜서 삭제하는데, 대응시키는 0인 Bit를 Mask Bit라고 함
OR (Selective Set)	<ul style="list-style-type: none"> · 특정 문자를 삽입하거나 특정 Bit에 1을 세트시키는 명령으로 Selective Set 연산이라고도 함 · 삽입하거나 세트 시킬 Bit에 삽입할 문자코드 또는 1을 OR 연산시킴
XOR : 비교 (Compare) 명령	<ul style="list-style-type: none"> · 2개의 데이터를 비교하거나, 특정 비트를 반전시킬 때 사용함 · 2개의 데이터를 XOR 연산하여 결과에 1Bit라도 1이 있으면 서로 다른 데이터임 · 반전시킬 때는 반전시킬 비트와 1을 XOR 시킴
NOT(Complement, 보수)	각 비트의 값을 반전시키는 연산으로 보수를 구할 때 사용함
논리 Shift	<ul style="list-style-type: none"> · 왼쪽 또는 오른쪽으로 1Bit씩 자리를 이동시키는 연산으로 데이터의 직렬전송(Serial Transfer)에 사용함 · 삽입되는 자리는 무조건 0임
Rotate	<ul style="list-style-type: none"> · Shift에서 밀려 나가는 비트의 값을 반대편 값으로 입력하는 연산임 · 문자 위치를 변환할 때 이용
산술 Shift	<ul style="list-style-type: none"> · 부호(Sign)를 고려하여 자리를 이동시키는 연산으로, 2ⁿ으로 곱하거나 나눌 때 사용함 · 왼쪽으로 n Bit Shift하면 원래 자료에 2ⁿ을 곱한 값과 같다. · 오른쪽으로 n Bit Shift 하면 원래 자료를 2ⁿ으로 나눈 값과 같음 · 홀수를 오른쪽으로 한 번 Shift하면 0.5의 오차가 발생함

핵심 073 | 명령어 형식

3 번지 명령어	<ul style="list-style-type: none"> · Operand부가 3개로 구성되는 명령어형식으로 여러 개의 범용 레지스터(GPR)를 가진 컴퓨터에서 사용함 · 연산의 결과는 Operand3에 기록됨 · 연산시 원시 자료를 파괴하지 않음 · 다른 형식의 명령어를 이용하는 것보다 프로그램 전체의 길이를 짧게 할 수 있음 · 전체 프로그램 실행시 명령 인출을 위하여 주기억장치를 접근하는 횟수가 줄어들어 프로그램 실행속도를 단축시킴 · 명령어 한 개의 길이가 너무 길어짐
2 번지 명령어	<ul style="list-style-type: none"> · Operand부가 2개로 구성되는 명령어 형식으로 가장 일반적으로 사용되는 명령어 형식 · 여러 개의 범용레지스터를 가진 컴퓨터에서 사용함 · 3 주소 명령에 비해 명령어의 길이가 짧음 · 연산의 결과는 주로 Operand1에 저장되므로 Operand1에 있던 원시자료가 파괴됨 · 전체 프로그램의 길이가 길어짐
1 번지 명령어	<ul style="list-style-type: none"> · Operand부가 1개로 구성되어 있음 · AC(Accumulator ; 누산기)를 이용하여 명령어를 처리함
0 번지 명령어	<ul style="list-style-type: none"> · Operand부 없이 OP Code부만으로 구성됨 · 모든 연산은 STACK 메모리의 Stack Pointer 가 가리키는 Operand를 이용하여 수행함 · 모든 연산은 스택에 있는 자료를 이용하여 수행하기 때문에 스택머신(Stack Machine)이라고도함 · 원래의 자료가 남지 않음

핵심 074 | 주소지정방식(Addressing Mode)의 종류

암시적 주소 지정 방식 (Implied Mode)	주소를 지정하는 필드가 없는 0 번지 명령어에서 Stack의 SP가 가리키는 Operand를 암시하여 이용함
즉치(즉시)적 주소지정방식 (Immediate Mode)	<ul style="list-style-type: none"> 명령어 자체에 오퍼랜드(실제 데이터)를 내포하고 있는 방식 별도의 기억장소를 액세스하지 않고 CPU에서 곧바로 자료를 이용할 수 있어서 실행속도가 빠르다는 장점이 있음 명령어의 길이에 영향을 받으므로 표현할 수 있는 데이터 값의 범위가 제한적임
직접 주소 지정방식 (Direct Mode)	<ul style="list-style-type: none"> 명령의 주소부(Operand)가 사용할 자료의 번지를 표현하고 있는 방식 명령의 Operand부에 표현된 주소를 이용하여 실제 데이터가 기억된 기억장소에 직접 사상시킬 수 있음 기억용량이 2ⁿ개의 Word인 메모리 시스템에서 주소를 표현하려면 n 비트의 Operand부가 필요함
간접 주소 지정방식 (Indirect Mode)	<ul style="list-style-type: none"> 명령어에 나타낸 주소가 명령어 내에서 데이터를 지정하기 위해 할당된 비트(Operand 부의 비트) 수로 나타낼 수 없을 때 사용하는 방식 명령의 길이가 짧고 제한되어 있어도 긴 주소에 접근 가능함 명령어 내의 주소부에 실제 데이터가 저장된 장소의 번지를 가진 기억장소의 번지를 표현함으로써, 최소한 주기억장치를 두 번 이상 접근하여 데이터가 있는 기억장소에 도달함
계산에 의한 주소지정 방식	<ul style="list-style-type: none"> 상대 주소 지정방식 : 명령어의 주소부분 + PC Base Register Mode : 명령어의 주소부분 + Base Register Index Register Mode : 명령어의 주소부분 + Index Register

핵심 075 | 마이크로 오퍼레이션(Micro Operation)의 정의

- Instruction을 수행하기 위해 CPU내의 레지스터와 플래그가 의미 있는 상태 변환을 하도록 하는 동작
- 레지스터에 저장된 데이터에 의해 이루어지는 동작
- 한 개의 Clock 펄스 동안 실행되는 기본 동작
- 마이크로 오퍼레이션의 순서를 결정하기 위하여 제어장치가 발생하는 신호를 제어신호라고 함
- 한 개의 Instruction은 여러 개의 Micro Operation이 동작되어 실행됨
- Micro Cycle Time : 한 개의 Micro Operation을 수행하는데 걸리는 시간

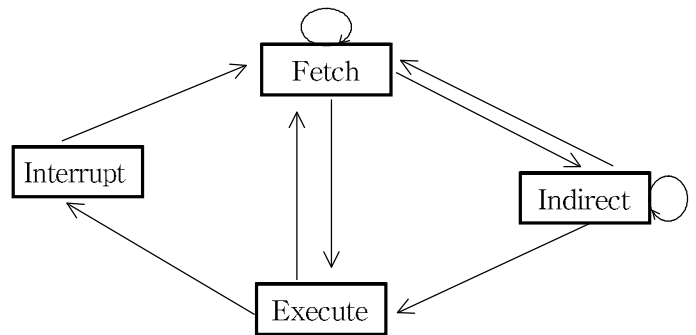
핵심 076 | Micro Cycle Time 부여 방식

동기 고정식 (Synchronous fixed)	<ul style="list-style-type: none"> 모든 마이크로 오퍼레이션의 동작시간이 같다고 가정하여 CPU Clock의 주기를 Micro Cycle Time과 같도록 정의하는 방식 모든 마이크로 오퍼레이션 중에서 수행시간이 가장 긴 마이크로 오퍼레이션의 동작시간을 Micro Cycle Time으로 정함 모든 마이크로 오퍼레이션의 동작시간이 비슷할 때 유리한 방식임 장점 : 제어기의 구현이 단순함 단점 : CPU의 시간 낭비가 심함
----------------------------	--

동기 가변식(Synchronous Variable)	<ul style="list-style-type: none"> 수행시간이 유사한 Micro Operation끼리 그룹을 만들어, 각 그룹별로 서로 다른 Micro Cycle Time을 정의하는 방식 동기 고정식에 비해 CPU 시간 낭비를 줄일 수 있는 반면 제어기의 구현은 조금 복잡함 마이크로 오퍼레이션의 동작시간이 차이가 날 때 유리함(정수배)
비동기식 (Asynchronous)	<ul style="list-style-type: none"> 모든 마이크로 오퍼레이션에 대하여 서로 다른 Micro Cycle Time을 정의하는 방식 CPU의 시간 낭비는 전혀 없으나, 제어기가 매우 복잡해지기 때문에 실제로는 거의 사용되지 않음

핵심 077 | 메이저 스테이트

- 현재 CPU가 무엇을 하고 있는가를 나타내는 상태로서 fetch, in direct, execute, interrupt 이렇게 4개의 상태가 있음
- CPU는 메이저 스테이트의 4가지 단계를 반복적으로 거치면서 동작을 수행함
- 메이저 스테이트는 메이저 스테이트 레지스터를 통해서 알 수 있음
- Major Cycle 또는 Machine Cycle라고도 함
- 메이저 스테이트의 변천 과정



핵심 078 | 인출 단계

- 명령어를 주기억장치에서 중앙처리장치의 명령레지스터로 가져와 해석하는 단계
- 읽어와 해석된 명령어가 1 Cycle 명령이면 이를 수행한 후 다시 Fetch Cycle 사이클로 변천함
- 1 Cycle 명령이 아니면, 해석된 명령어의 모드비트에 따라 직접주소와 간접 주소를 판단함

제어신호	Micro Operation	의미
C ₀ t ₀	MAR ← PC	PC에 있는 번지를 MAR에 전송시킴
C ₀ t ₁	MBR ← M[MAR], PC ← PC + 1	<ul style="list-style-type: none"> 메모리에서 MAR이 지정하는 위치의 값을 MBR에 전송함 다음에 실행할 명령의 위치를 지정하기 위해 PC의 값을 1 증가시킴.
C ₀ t ₂	IR ← MBR[OP], I ← MBR[I]	<ul style="list-style-type: none"> 명령어의 OP-code 부분을 명령레지스터에 전송함 ※ 현재 MBR에는 주기억 장치에서 읽어온 명령이 들어있음 명령어의 모드비트를 플립플롭 I에 전송함
C ₀ t ₃	F ← 1 또는 R ← 1	I가 0이면 F 플립플롭에 1을 전송하여 Execute 단계로 변천하고, I가 1이면 R 플립플롭에 1을 전송하여 Indirect 단계로 변천함

핵심 079 | 간접 단계(Indirect cycle)

- Fetch 단계에서 해석된 명령의 주소부가 간접주소인 경우 수행됨
- Fetch 단계에서 해석한 주소를 읽어온 후 그 주소가 간접주소이면 유효주소를 계산하기 위해 다시 Indirect 단계를 수행 함
- 간접 주소가 아닌 경우에는 명령어에 따라서 Execute 단계 또는 Fetch 단계로 이동할지를 판단함

제어신호	Micro Operation	의미
C ₁₀	MAR ← MBR[AD]	MBR에 있는 명령어의 번지 부분을 MAR에 전송함
C ₁₁	MBR ← M[MAR]	메모리에서 MAR이 지정하는 위치의 값을 MBR에 전송함
C ₁₂	No Operation	동작없음
C ₁₃	F ← 1, R ← 0	F에 1, R에 0을 전송하여 Execute 단계로 변천함

핵심 080 | 실행 단계(Execute Cycle)

- Fetch 단계에서 인출하여 해석한 명령을 실행하는 단계
- 플래그 레지스터의 상태 변화를 검사하여 Interrupt 단계로 변천할 것인지를 판단함
- Interrupt 요청신호를 나타내는 플래그 레지스터의 변화가 없으면 Fetch 단계로 변천함
- ADD 연산을 수행하는 Execute 단계

제어신호	Micro Operation	의미
C ₂₀	MAR ← MBR[AD]	MBR에 있는 명령어의 번지 부분을 MAR에 전송함
C ₂₁	MBR ← M[MAR]	메모리에서 MAR이 지정하는 위치의 값을 MBR에 전송함
C ₂₂	AC ← AC + MBR	누산기의 값과 MBR의 값을 더해 누산기에 전송함 ※ 실질적인 ADD 연산이 이루어는 부분임
C ₂₃	F ← 0 또는 R ← 1	F에 0을 전송하면 F=0, R=0이 되어 Fetch 단계로 변천하고, R에 1을 주면 F=1, R=1이 되어 Interrupt 단계로 변천함

핵심 081 | 인터럽트 단계(Interrupt Cycle)

- 인터럽트 발생시 복귀주소(PC)를 저장시키고, 제어순서를 인터럽트 처리 프로그램의 첫 번째 명령으로 옮기는 단계
- 인터럽트 단계를 마친 후에는 항상 Fetch 단계로 변천함

제어신호	Micro Operation	의미
C ₃₀	MBR[AD] ← PC, PC ← 0	PC가 가지고 있는, 다음에 실행할 명령의 주소를 MBR의 주소 부분으로 전송함 · 복귀 주소를 저장할 0번지를 PC에 전송함
C ₃₁	MAR ← PC, PC ← PC + 1	PC가 가지고 있는, 값 0번지를 MAR에 전송함 · 인터럽트 처리 루틴으로 이동할 수 있는 인터럽트 벡터의 위치를 지정하기 위해 PC의 값을 1 증가 시켜 1로 세트시킴

C ₂₂	M[MAR] ← MBR, IEN ← 0	· MBR이 가지고 있는, 다음에 실행할 명령의 주소를 메모리의 MAR이 가리키는 위치(0 번지)에 저장함 · 인터럽트 단계가 끝날 때까지 다른 인터럽트가 발생하지 않게 IEN에 0을 전송함
C ₂₃	F ← 0, R ← 0	F에 0, R에 0을 전송하여 Fetch 단계로 변천함

핵심 082 | 주요 명령의 마이크로 오퍼레이션

ADD : AC ← AC + M[AD]

제어신호	Micro Operation	의미
C ₂₀	MAR ← MBR[AD]	MBR에 있는 명령어의 번지 부분을 MAR에 전송함
C ₂₁	MBR ← M[MAR]	메모리에서 MAR이 지정하는 위치의 값을 MBR에 전송함
C ₂₂	AC ← AC + MBR	누산기의 값과 MBR의 값을 더해 누산기에 전송함
C ₂₃	IEN F ← 0	F에 0을 전송하면 F=0, R=0이 되어 Fetch 단계로 변천함
	IEN R ← 1	R에 1을 전송하면 F=1, R=1이 되어 Interrupt 단계로 변천함

LDA(Load to AC) : AC ← M[AD]

제어신호	Micro Operation	의미
C ₂₀	MAR ← MBR[AD]	MBR에 있는 명령어의 번지 부분을 MAR에 전송함
C ₂₁	MBR ← M[MAR] AC ← 0	· 메모리에서 MAR이 지정하는 위치의 값을 MBR에 전송함 · AC에 0을 전송하여 AC를 초기화 함
C ₂₂	AC ← AC + MBR	· 메모리에서 가져온 MBR과 AC를 더해 AC에 전송함 · 초기화된 AC에 더해지므로 메모리의 값을 AC로 불러오는 것과 같다.
C ₂₃	IEN F ← 0	F에 0을 전송하면 F=0, R=0이 되어 Fetch 단계로 변천함
	IEN R ← 1	R에 1을 전송하면 F=1, R=1이 되어 Interrupt 단계로 변천함

STA(Store AC) : M[AD] ← AC

제어신호	Micro Operation	의미
C ₂₀	MAR ← MBR[AD]	MBR에 있는 명령어의 번지 부분을 MAR에 전송함
C ₂₁	MBR ← AC	AC의 값을 MBR에 전송함
C ₂₂	M[MAR] ← MBR	MBR의 값을 메모리의 MAR이 지정하는 위치에 전송함
C ₂₃	IEN F ← 0	F에 0을 전송하면 F=0, R=0이 되어 Fetch 단계로 변천함
	IEN R ← 1	R에 1을 전송하면 F=1, R=1이 되어 Interrupt 단계로 변천함

BUN(Branch unconditionally)

BUN은 PC에 특정한 주소를 전송하여 실행명령의 위치를 변경하는 무조건 분기 명령임

제어신호	Micro Operation	의미
C ₂ t ₀	PC ← MBR[AD]	<ul style="list-style-type: none"> MBR에 있는 명령어의 번지 부분을 PC에 전송함 ※다음에 실행할 명령의 주소를 갖는 PC의 값이 변경되었으므로 변경된 주소에서 다음 명령이 실행됨
C ₂ t ₁	no Operation	동작없음
C ₂ t ₂	no Operation	동작없음
C ₂ t ₃	IEN' F ← 0	F에 0을 전송하면 F=0, R=0이 되어 Fetch 단계로 변천함
	IEN R ← 1	R에 1을 전송하면 F=1, R=1이 되어 Interrupt 단계로 변천함

핵심 083 | 제어 데이터

제어장치가 제어신호를 발생하기 위한 자료로서, CPU가 특정한 메이저 상태와 타이밍 상태에 있을 때 제어자료에 따른 제어규칙에 의해 제어신호가 발생함

- ① 메이저 스테이트 사이의 변천을 제어하는 데이터
- ② 중앙처리장치의 제어점을 제어하는 데이터
- ③ 인스트럭션의 수행 순서를 결정하는데 필요한 제어 데이터

핵심 084 | 제어장치의 비교

제어장치는 필요한 마이크로 연산들이 연속적으로 수행할 수 있도록 제어 신호를 보내는 역할을 함

	고정배선 제어장치	마이크로 프로그래밍 기법
반응속도	고속	저속
회로 복잡도	복잡	간단
경제성	비경제적	경제적
융통성	없음	있음
구성	하드웨어	소프트웨어

핵심 085 | 입출력장치의 구성

입 · 출력 제어 장치	<ul style="list-style-type: none"> 입 · 출력 장치와 컴퓨터 사이의 자료전송을 제어하는 장치 데이터 버퍼 레지스터를 이용하여 두 장치간의 속도차이를 조절함 제어신호의 논리적, 물리적 변환 그리고 에러를 제어함 종류 : DMA, 채널, 입 · 출력 프로세서, 입 · 출력 컴퓨터
입 · 출력 인터페이스	<ul style="list-style-type: none"> 동작방식이나 데이터 형식이 서로 다른 컴퓨터 내부의 주기억 장치나 CPU의 레지스터와 외부 입 · 출력 장치간의 이진 정보를 원활하게 전송하기 위한 방법을 제공함 컴퓨터와 각 주변 장치와의 다음과 같은 차이점을 해결하는 것이 목적임 <ul style="list-style-type: none"> - 전자기 혹은 기계적인 주변장치와 전기적인 CPU나 메모리 간의 동작방식의 차이 - 주변 장치와 CPU간의 데이터 전송속도의 차이 - 주변 장치의 데이터 코드와 CPU나 메모리의 워드 형식의 차이 - 동작 방식이 서로 다른 주변 장치들의 간섭 없는 제어
입 · 출력 버스	<ul style="list-style-type: none"> 주기억장치와 입 · 출력장치 사이의 데이터 전송을 위해 모든 주변 장치의 인터페이스에 공통으로 연결된 버스 데이터버스, 주소버스, 제어 버스로 구성되어 있음

핵심 086 | 기억장치와 입출력 장치 동작의 차이

기억장치는 처리속도가 nano(10^{-9})의 단위인 전자적인 장치이고 입출력 장치는 milli(10^{-3})의 단위인 기계적인 장치이므로 동작방식에는 많은 차이가 있음

비교항목	입 · 출력 장치	기억 장치
동작의 속도	느리다	빠르다
동작의 자율성	타율/자율	타율
정보의 단위	Byte(문자)	Word
착오 발생률	많다	적다

핵심 087 | 스푼링(spooling)

- 다중 프로그래밍 환경 하에서 용량이 크고 신속한 액세스가 가능한 디스크를 이용하여 각 사용자 프로그램이 입 ·출력할 데이터를 직접 I/O 장치로 보내지 않고 디스크에 모았다가 나중에 한꺼번에 입 ·출력함으로써 입 ·출력 장치의 공유 및 상대적으로 느린 입 ·출력 장치의 처리 속도를 보완하는 기법
- 스푼링은 고속의 CPU와 저속의 입 ·출력장치가 동시에 독립적으로 동작하게 하여 높은 효율로 여러 작업을 병행 작업할 수 있도록 해줌으로써 다중 프로그래밍 시스템의 성능 향상을 가져올 수 있음
- 스푼링은 디스크 일부를 매우 큰 버퍼처럼 사용하는 방법임

핵심 088 | 입 · 출력 방식

Program med I/O	<ul style="list-style-type: none"> 원하는 I/O가 완료되었는지의 여부를 검사하기 위해서 CPU가 상태 Flag를 계속 조사하여 I/O가 완료 되었으면 MDR (MBR)과 AC 사이의 자료전송도 CPU가 직접 처리하는 I/O 방식 입 ·출력에 필요한 대부분의 일을 CPU가 해주므로 Interface는 MDR, Flag, 장치번호 디코더로만 구성하면 됨 I/O 작업시 CPU는 계속 I/O 작업에 관여해야 하기 때문에 다른 작업을 할 수 없다는 단점이 있음
Interrupt I/O	<ul style="list-style-type: none"> 입 ·출력을 하기 위해 CPU가 계속 Flag를 검사하지 않고, 데이터를 전송할 준비가 되면 입 ·출력 인터페이스가 컴퓨터에게 알려 입 ·출력이 이루어지는 방식임 입 ·출력 인터페이스는 CPU에게 인터럽트 신호를 보내 입 ·출력이 있음을 알림 CPU는 작업을 수행하던 중 입 ·출력 인터럽트가 발생하면 수행중인 프로그램을 중단하고 입 ·출력을 처리한 후 원래의 작업으로 돌아와 작업을 계속 수행함 CPU가 계속 Flag를 검사하지 않아도 되기 때문에 Programmed I/O 보다 효율적임
DMA (Direct Memory Access)에 의한 I/O	<ul style="list-style-type: none"> 입 ·출력장치가 직접 주기억장치를 접근(Access)하여 Data Block을 입 ·출력하는 방식으로 입 ·출력 전송이 CPU의 레지스터를 경유하지 않고 수행됨 CPU는 I/O에 필요한 정보를 DMA제어기에 알려져서 I/O동작을 개시 시킨 후 I/O동작에 더 이상 간섭하지 않고 다른 프로그램을 할당하여 수행함 입 ·출력 자료 전송시 CPU를 거치지 않기 때문에 CPU의 부담없이 보다 빠른 데이터의 전송이 가능함 인터럽트 신호를 발생시켜 CPU에게 입 ·출력 종료를 알림 Cycle Steal 방식을 이용하여 데이터를 전송함

Channel에 의한 I/O	<ul style="list-style-type: none"> · CPU를 대신하여 주기억장치와 입·출력장치 사이에서 입·출력을 제어하는 입·출력 전용 프로세서(IOP)임 · 채널 제어기는 채널명령어로 작성된 채널 프로그램을 해독하고 실행하여 입·출력 동작을 처리함 · CPU로부터 입·출력 전송을 위한 명령어를 받으면 CPU와는 독립적으로 동작하여 입·출력을 완료함 · CPU와 인터럽트로 통신함 · 채널의 종류 <ul style="list-style-type: none"> - Selector Channel : 고속 입·출력장치(자기디스크, 자기테이프, 자기드럼)1개와 입·출력하기 위해 사용함 - Multiplexer Channel : 저속 입·출력장치(카드리더, 프린터)를 여러개를 동시에 제어하는 채널 - Block Multiplexer Channel : 동시에 여러 개의 고속 입·출력장치를 제어함
-----------------	---

핵심 089 | 인터럽트의 정의

- 프로그램을 실행하는 도중에 예기치 않은 상황이 발생할 경우 현재 실행 중인 작업을 즉시 중단하고, 발생한 상황을 우선 처리한 후 실행 중이던 작업으로 복귀하여 계속 처리하는 것, 일명 "끼어들기"라고도 함
- 외부 인터럽트, 내부 인터럽트, 소프트웨어 인터럽트로 분류하는데, 외부나 내부 인터럽트는 CPU의 하드웨어에서의 신호에 의해 발생하고 소프트웨어 인터럽트는 명령어의 수행에 의해 발생함

핵심 090 | 인터럽트의 종류 및 발생원인

외부 인터럽트	<ul style="list-style-type: none"> · 전원 이상 인터럽트 (Power Fail Interrupt) : 정전이 되거나 전원 이상이 있는 경우 · 기계 착오 인터럽트 (Machine Check Interrupt) : CPU의 기능적인 오류동작이 발생한 경우 · 외부 신호 인터럽트 (External Interrupt) <ul style="list-style-type: none"> - 타이머에 의해 규정된 시간(Time Slice)을 알리는 경우 - 키보드로 인터럽트 키를 누른 경우 - 외부장치로부터 인터럽트 요청이 있는 경우 · 입·출력 인터럽트 (Input-Output Interrupt) <ul style="list-style-type: none"> - 입·출력 Data의 오류나 이상 현상이 발생한 경우 - 입·출력장치가 데이터의 전송을 요구하거나 전송이 끝났음을 알릴 경우
내부 인터럽트	<ul style="list-style-type: none"> · 잘못된 명령이나 데이터를 사용할 때 발생하며, 트랩(Trap)이라고도 부름 · 명령어 잘못에 의한 인터럽트 : 프로그램에서 명령어를 잘못 사용한 경우 · 프로그램 인터럽트 (Program Interrupt) : 0으로 나누거나 Over flow 또는 Underflow가 발생한 경우
소프트웨어 인터럽트	<ul style="list-style-type: none"> · 프로그램 처리 중 명령의 요청에 의해 발생하는 것으로, 가장 대표적인 형태는 감시 프로그램을 호출하는 SVC(Supervisor Call) 인터럽트가 있음 · SVC (Supervisor Call) 인터럽트 : 사용자가 SVC 명령을 써서 의도적으로 호출한 경우

핵심 091 | 인터럽트 발생시 CPU가 확인할 사항

- 프로그램 카운터의 내용
- 사용한 모든 레지스터의 내용
- 상태 조건의 내용(PSW)

핵심 092 | 인터럽트의 동작원리

- ① 인터럽트 요청 신호 발생
- ② 프로그램 실행을 중단함 : 현재 실행 중이던 명령어(Micro Instruction)는 끝까지 실행함
- ③ 현재의 프로그램 상태를 보존함 : 프로그램 상태는 다음에 실행할 명령의 번지로서 PC가 가지고 있음
- ④ 인터럽트 처리 루틴을 실행함 : 인터럽트를 요청한 장치를 식별함
- ⑤ 인터럽트 서비스 루틴을 실행함 : 실질적인 인터럽트를 처리함
- ⑥ 상태복구: 인터럽트 요청신호가 발생했을 때 보관한 PC의 값을 다시 PC에 저장함
- ⑦ 중단된 프로그램 실행 재개 : PC의 값을 이용하여 인터럽트 발생 이전에 수행 중이던 프로그램을 계속 실행함

핵심 093 | 인터럽트 우선순위

- 목적 : 여러 장치에서 동시에 인터럽트가 발생하였을 때 먼저 서비스할 장치를 결정하기 위해서임
- 우선 순위(높음>낮음) : 전원 이상(Power Fail) > 기계착오(Machine Check) > 외부신호(External) > 입출력(I/O) > 명령어 잘못 > 프로그램(Program Check) > SVC(Supervisor Call)

핵심 094 | 인터럽트 우선순위 판별 방법

소프트웨어적인 인터럽트 우선순위 판별 방법 : Polling	<ul style="list-style-type: none"> · Interrupt 발생시 가장 높은 우선순위의 인터럽트 자원(Source)부터 차례로 검사해서, 우선순위가 가장 높은 Interrupt 자원(Source)를 찾아내어 이에 해당하는 인터럽트 서비스 루틴을 수행하는 방식 · 소프트웨어적인 방식을 폴링이라고 함 · 많은 인터럽트가 있을 때 그들을 모두 조사하는 데 많은 시간이 걸려 반응시간이 느리다는 단점이 있음 · 회로가 간단하고 융통성이 있으며 별도의 하드웨어가 필요 없으므로 경제적인
하드웨어적인 인터럽트 우선순위 판별 방법	<ul style="list-style-type: none"> · CPU와 Interrupt를 요청할 수 있는 장치 사이에 장치 번호에 해당하는 버스를 병렬이나 직렬로 연결하여 요청장치의 번호를 CPU에 알리는 방식 · 장치 판별 과정이 간단해서 응답속도가 빠름 · 회로가 복잡하고 융통성이 없으며, 추가적인 하드웨어가 필요하므로 비경제적임 · 직렬(Serial) 우선순위 부여방식 : 데이지 체인(Daisy-Chain) <ul style="list-style-type: none"> - 인터럽트가 발생하는 모든 장치를 1개의 회선에 직렬로 연결함 - 우선순위가 높은 장치를 선두에 위치시키고 나머지를 우선순위에 따라 차례로 연결함 - 직렬 우선순위 부여방식을 데이지 체인 방식이라고 함 · 병렬(Parallel) 우선순위 부여방식 <ul style="list-style-type: none"> - 인터럽트가 발생하는 각 장치를 개별적인 회선으로 연결함 - 각 장치의 인터럽트 요청에 따라 각 Bit가 개별적으로 Set될 수 있는 Mask Register를 사용함 - 우선순위는 Mask Register의 Bit위치에 의해서 결정됨 - 우선순위가 높은 Interrupt는 낮은 Interrupt가 처리되는 중에도 우선 처리됨

핵심 095 | 기억장치의 특성을 결정하는 요소

기억용량	기억장치는 무조건 기억용량이 큰 것을 사용한다고 해서 좋은 것이 아니라, 사용목적에 따라 성능당 경비 비율이 적은 것을 사용하는 것이 바람직함
Access Time	<ul style="list-style-type: none"> 기억장치에 읽기요청이 발생한 시간부터 요구한 정보를 꺼내서 사용 가능할 때까지의 시간 한 Word단위의 정보를 읽거나 기록하는데 걸리는 시간 Access time = Seek Time + Latency Time(또는 Serach Time) + Transmission Time
Cycle Time	<ul style="list-style-type: none"> 기억장치에 읽기신호를 보낸 후 다시 읽기 신호를 보낼 수 있을 때까지의 시간 간격 Cycle Time ≥ Access Time
Bandwidth (대역폭, 전송률)	<ul style="list-style-type: none"> 메모리로부터 또는 메모리까지 1초 동안 전송되는 최대한의 정보량으로 기억장치의 자료처리 속도를 나타내는 단위 메모리 워드의 길이가 작을수록 대역폭이 좋음

핵심 096 | 기억장치의 구분

구분 방식	구분
내용의 보존 여부	<ul style="list-style-type: none"> 파괴성 메모리(Destructive Memory) : 판독 후 저장된 내용이 파괴되는 메모리로, 파괴된 내용을 재생시키기 위한 재 저장 시간(Restoration Time)이 필요함 (예) 자기 코어 비파괴성 메모리 : 판독 후에도 저장된 내용이 그대로 유지됨 (예) 자기코어를 제외한 모든 기억장치
전원단절시 내용 소멸 여부	<ul style="list-style-type: none"> 휘발성 메모리(Volatile Memory) : 전원이 단절되면 모든 정보가 지워지는 메모리 (예) RAM 비휘발성 메모리 : 전원이 단절되더라도 기억된 정보가 보존되는 메모리 (예) ROM, 자기코어, 보조기억장치
재충전 (Refresh) 여부	<ul style="list-style-type: none"> 정적메모리(SRAM) : 전원이 공급되는 한 기억된 내용이 계속 유지되는 메모리 동적메모리(DRAM) : 전원이 공급되어도 일정시간이 지나면 내용이 지워지므로 재충전을 해야 하는 메모리
접근 방식	<ul style="list-style-type: none"> 직접접근방식(SASD, Sequential Access Storage Device) : 자료가 저장된 위치에 접근할 때, 처음부터 순서대로 접근하여 원하는 위치를 검색하는 메모리 (예) 자기테이프 직접접근방식(DASD, Direct Access Storage Device) : 순서를 거치지 않고 자료가 저장된 위치를 직접 접근할 수 있는 메모리 (예) 자기 테이프를 제외한 모든 기억장치

핵심 097 | ROM(Read Only Memory)

- 기억된 내용을 읽을 수만 있는 기억장치로서 일반적으로 쓰기는 불가능함
- 전원이 꺼져도 기억된 내용이 지워지지 않는 비휘발성 메모리
- 실제로 ROM은 주기억장치로 사용하기 보다는 주로 기본 입.출력시스템(BIOS), 자가 진단 프로그램(POST) 같은 변경가능성이 희박한 시스템 소프트웨어를 기억시키는데 이용함

•ROM의 종류와 특징

종 류	특 징
Mask ROM	제조공장에서 프로그램화하여 생산한 ROM으로, 사용자가 내용을 변경시킬 수 없음
PROM (Programmable ROM)	PROM 프로그램장치라는 특수장치를 이용하여 비어 있는 ROM에 사용자가 한번만 내용을 기입할 수 있으며, 이후엔 읽기만 가능함

EPROM (Erasable PROM)	<ul style="list-style-type: none"> 자외선을 쏘여서 기입한 내용을 지울 수도 있고, PROM프로그램장치로 기입할 수도 있음 사용자가 여러 번 반복해서 지우거나 기입할 수 있음
EAROM (Erasable Alterable ROM)	전기적 특성을 이용하여 기록된 정보의 일부를 바꿀 수 있는 ROM
EEPROM (Electronic EPROM)	전기적인 방법을 이용하여 기록된 내용을 여러 번 수정하거나 새로운 내용을 기록할 수 있는 ROM

핵심 098 | RAM(Random Access Memory)

- 자유롭게 읽고 쓸 수 있는 기억장치로, RMM(Read Write Memory)라고도 함
- RAM에는 현재 사용 중인 프로그램이나 데이터가 저장되어 있음
- 전원이 꺼지면 기억된 내용이 모두 사라지는 휘발성 메모리임
- 일반적으로 '주기억장치' 또는 메모리라고 하면 램을 의미함
- 정보가 저장된 위치는 주소로 구분함

•SRAM/DRAM의 특징

	동적 램(DRAM)	정적 램(SRAM)
구성 소자	콘덴서	플립플롭
특징	전원이 공급되어도 일정 시간이 지나면 전하가 방전되므로 주기적인 재충전(Refresh)이 필요함	전원이 공급되는 동안에는 기억 내용이 유지됨
전력 소모	적음	많음
접근 속도	느림	빠름
집적도(밀도)	높음	낮음
가격	저가	고가
용도	일반적인 주기억장치	캐시 메모리

핵심 099 | 자기 코어

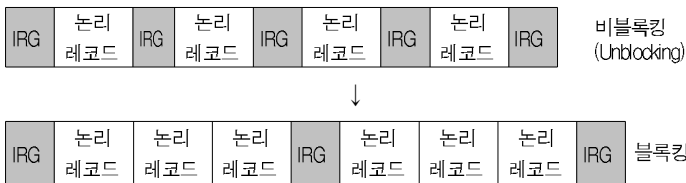
- 전류 일치 기술(coincident-current technique)에 의하여 기억장소를 선별함
- 데이터를 읽으면 읽은 내용이 지워지는 파괴메모리(destructive memory) 이므로 내용을 읽은 후 지워진 내용을 기록하기 위한 재 저장(restoration time)시간이 필요함
- 자기 코어는 중심을 통과하는 전선에 흐르는 전류의 방향에 따라 1 혹은 0의 값을 갖음
- 자기 코어는 부피에 비해 용량이 작고 가격이 비싸 현재는 거의 사용하지 않음
- 자기코어에는 4개의 선이 있음
 - 구동선(X, Y) 2개 : 번지 선택선
 - 센스선 1개 : 자기코어의 상태 검출
 - 금지선 1개 : 불필요하게 자화되었을 때 -1/2 금지 전류를 흘려 자화를 소거시키는 선

핵심 100 | 보조기억장치

자기 테이프	<ul style="list-style-type: none"> • 순차처리(SASD)만 할 수 있는 대용량 저장매체 • 가격이 저렴하고 용량이 커서 자료의 백업용으로 많이 사용함 • 자성 물질이 코팅된 얇은 플라스틱 테이프를 동그란 릴에 감아 놓은 형태 • 테이프의 시작과 끝부분을 알리는 은박지 사이의 정보 저장 부분을 7~9 트랙으로 구성함
자기 디스크 (Magnetic Disk)	<ul style="list-style-type: none"> • 자성 물질을 입힌 금속 원판을 여러 장 겹쳐서 만든 기억매체로 용량이 크고 접근 속도가 빠름 • 순차, 비순차(직접) 처리가 모두 가능한 DASD(Direct Access Storage Device) 방식으로 데이터를 처리함 • 트랙(Track) : 디스크 표면에서 회전축(스핀들 모터)을 중심으로 데이터가 기록되는 동심원 • 섹터(Sector) : Track들을 일정한 크기로 구분한 부분이며, 정보 기록의 기본 단위임 • 실린더(Cylinder) : 서로 다른 면들에 있는 동일 위치의 Track들의 모임으로 실린더의 수는 한 면의 트랙 수와 동일함
자기드럼 (Magnetic Drum)	<ul style="list-style-type: none"> • 원통 표면에 Track과 Sector를 구성하고, 각 Track마다 고정된 R/W Head를 두고 있어 자기디스크에 비해 속도가 빠름 • 순차, 비순차(직접) 처리가 모두 가능한 DASD(Direct Access Storage Device) 방식으로 데이터를 처리함 • 크기에 비해 용량이 적어 현재는 거의 사용하지 않음

핵심 101 | 블로킹(Blocking)

- 1개 이상의 논리적 레코드를 묶어서 테이프에 기록하는 방식



- 하나의 블록을 구성하는 논리레코드의 개수를 블록화 인수(BF ; Blocking Factor)라고 함
- 블로킹을 하면 블로킹을 하지 않았을 때에 비해 IRG의 수가 줄어들므로 다음과 같은 장점이 있음
 - 기억공간의 낭비가 줄어듦
 - Access Time이 감소함
 - 입·출력 횟수가 감소함

핵심 102 | 디스크의 Access Time(이동 Head)

- 디스크 시스템은 디스크 번호, 디스크 표면 번호, 트랙 번호, 섹터 번호를 표현하는 번지 Bit를 가지고 디스크의 기억공간을 Access함
- Access Time = Seek Time + Latency Time + Transmission Time
- Seek Time(탐색 시간) : R/W Head가 특정 트랙까지 이동하는데 걸리는 시간
- Latency Time(회전 지연 시간) 또는 Search Time : R/W Head가 특정 트랙까지 이동한 후 디스크가 회전하여 트랙에 포함되어 있는 특정 섹터가 R/W Head까지 도달하는데 걸리는 시간
- Transmission Time(전송 시간) : R/W Head가 Access한 Sector와 주 기억장치 간의 자료 전송에 걸리는 시간

핵심 103 | 특수기억장치

연관기억 장치 (Associative Memory)	<ul style="list-style-type: none"> • 기억장치에서 자료를 찾을 때 주소에 의해 접근하지 않고, 기억된 내용의 일부를 이용하여 Access할 수 있는 기억장치로 CAM(Content Addressable Memory)이라고도 함 • 주소에 의해서만 접근이 가능한 기억장치보다 정보검색이 신속함 • 캐시메모리나 가상메모리관리 기법에서 사용하는 Mapping Table에 사용됨 • 외부의 인자와 내용을 비교하기 위한 병렬 판독 논리 회로를 갖고 있기 때문에 하드웨어 비용이 증가함
복수 모듈 기억장치 (Memory Interleaving)	<ul style="list-style-type: none"> • 독자적으로 데이터를 저장할 수 있는 기억장치 모듈을 여러 개 가진 기억장치 • 주기억장치와 CPU의 속도차의 문제점을 개선함 • 기억장치 버스를 시분할하여 사용함 • 기억장소의 접근을 보다 빠르게 함 • 복수 모듈 기억장치에 사용되는 각각의 기억장치는 자체의 어드레스 레지스터와 버퍼 레지스터를 가지고 독자적으로 데이터를 저장할 수 있음 • 인터리빙 기법에 의해 기억장치를 구성하는 모듈 수만큼의 단어(Word)들에 동시 접근이 가능함
캐시 메모리 (Cache Memory)	<ul style="list-style-type: none"> • CPU의 속도와 메모리의 속도 차이를 줄이기 위해 사용하는 고속 Buffer Memory임 • 캐시는 주기억장치와 CPU 사이에 위치함 • 캐시 메모리는 메모리 계층 구조에서 가장 빠른 소자이며, 처리속도가 거의 CPU의 속도와 비슷할 정도임 • 캐시를 사용하면 기억장치의 접근(access) 시간이 줄어들므로 컴퓨터의 처리속도가 향상됨 • 캐시는 수십 Kbyte~수백 Kbyte의 용량을 사용함
가상기억장치 (Virtual Memory)	<ul style="list-style-type: none"> • 기억용량이 작은 주기억장치를 마치 큰 용량을 가진 것처럼 사용할 수 있도록 하는 운영체제의 메모리 운영 기법 • 가상기억장치의 목적은 주기억장치의 용량 확보임 • 가상 기억장치는 하드웨어적으로 실제로 존재하는 것이 아니고 소프트웨어적인 방법으로 보조기억 장치를 주기억장치처럼 사용하는 것임 • 사용자 프로그램을 여러 개의 작은 블록으로 나누어서 보조기억장치 상에 보관해놓고 프로그램 실행시 필요한 부분들만 주기억장치에 적재함 • 주기억장치의 이용률과 다중 프로그래밍의 효율을 높일 수 있음 • 가상기억장치 기법에서 사용하는 보조기억장치는 디스크 같은 DASD 장치이어야 함

핵심 104 | 병렬컴퓨터의 개요

병렬 처리 개념

병렬처리는 폰 노이만 컴퓨터 구조의 순차처리에 반대되는 구조로 I/O 채널 또는 Processor와 같은 다수의 Processor(처리기)에서 동시에 여러 작업(Process)을 처리하는 것을 말함

병렬 처리 대상

병렬 처리 컴퓨터는 일상적인 작업처리 보다는 다음과 같은 특수한 업무에 적용됨

- 짧은 시간에 정확하고 많은 양의 처리해야 하는 일기예보
- 인공지능 분야에서의 음성, 화상, 자연언어 처리
- 역학계산, 자원탐사, 핵반응 연구를 위한 모의실험
- 유도탄 등의 첨단 군장비

3과목 · 운영체제

핵심 105 | 시스템 소프트웨어의 구성

- **제어 프로그램(Control Program)** : 시스템 전체의 작동 상태 감시, 작업의 순서 지정, 작업에 사용되는 데이터 관리 등의 역할을 수행하는 프로그램

감시(Supervisor) 프로그램	각종 프로그램의 실행과 시스템 전체의 작동 상태를 감시 · 감독하는 프로그램
작업 제어(Job Control) 프로그램	어떤 업무를 처리하고 다른 업무로의 이행을 자동으로 수행하기 위한 준비 및 그 처리에 대한 완료를 담당하는 프로그램
자료 관리(Data Management) 프로그램	주기억장치와 보조기억장치 사이의 데이터 전송과 보조기억장치의 자료 갱신 및 유지 보수 기능을 수행하는 프로그램

- **처리 프로그램** : 제어 프로그램의 지시를 받아 사용자가 요구한 문제를 해결하기 위한 프로그램

언어 번역(Language Translate) 프로그램	원시 프로그램을 기계어 형태의 목적 프로그램으로 번역하는 프로그램(어셈블러, 컴파일러, 인터프리터)
서비스(Service) 프로그램	컴퓨터를 효율적으로 사용할 수 있는 사용 빈도가 높은 프로그램
문제(Problem) 프로그램	특정 업무 및 해결을 위해 사용자가 작성한 프로그램

핵심 106 | 운영체제 정의/목적/기능

정의	컴퓨터 시스템의 자원들을 효율적으로 관리하며, 사용자가 컴퓨터를 편리하고 효과적으로 사용할 수 있도록 환경을 제공하는 여러 프로그램의 모임
목적	<ul style="list-style-type: none"> · 사용자와 컴퓨터 간의 인터페이스 제공 · 자원의 효율적인 운영 및 자원 스케줄링 · 데이터 공유 및 주변 장치 관리 · 처리 능력 및 신뢰도 향상, 사용 가능성도 향상 · 응답(반응) 시간 단축, 반환 시간 등의 단축
기능	<ul style="list-style-type: none"> · 프로세서, 기억장치, 입 · 출력장치, 파일 및 정보 등의 자원 관리 · 자원의 스케줄링 기능 제공 · 사용자와 시스템 간의 편리한 인터페이스 제공 · 시스템의 각종 하드웨어와 네트워크 관리 · 제어 · 시스템의 오류 검사 및 복구, 데이터 관리, 데이터 및 자원 공유 · 자원 보호 기능 제공 · 가상 계산기 기능 제공

핵심 107 | 운영체제 운용 기법 및 발달과정

- **일괄 처리(Batch Processing) 시스템** : 초기의 컴퓨터 시스템에서 사용된 형태로, 일정량 또는 일정 기간 동안 데이터를 모아서 한꺼번에 처리하는 방식
- **다중 프로그래밍(Multi Programming) 시스템** : 하나의 CPU와 주기억장치를 이용하여 여러 개의 프로그램을 동시에 처리하는 방식
- **시분할(Time Sharing) 시스템** : 여러 명의 사용자가 사용하는 시스템에서 컴퓨터가 사용자들의 프로그램을 번갈아가며 처리해줌으로써 각 사용자에게 각자 독립된 컴퓨터를 사용하는 느낌을 주는 방식(라운드 로빈 방식)

- **다중 처리(Multi Processing) 시스템** : 여러 개의 CPU와 하나의 주기억장치를 이용하여 여러 개의 프로그램을 동시에 처리하는 방식
- **실시간 처리(Real Time Processing) 시스템** : 데이터 발생 즉시, 또는 데이터 처리 요구가 있는 즉시 처리하여 결과를 산출하는 방식
- **다중 모드 처리(Multi Mode Processing)** : 일괄 처리 시스템, 시분할 시스템, 다중 처리 시스템, 실시간 처리 시스템을 한 시스템에서 모두 제공하는 방식
- **분산 처리(Distributed Processing) 시스템** : 여러 개의 컴퓨터(프로세서)를 통신 회선으로 연결하여 하나의 작업을 처리하는 방식
- **발달 과정** : 일괄 처리 시스템 → 다중 프로그래밍, 다중 처리, 시분할, 실시간 처리 시스템 → 다중 모드 → 분산 처리 시스템

핵심 108 | 컴파일러와 인터프리터

컴파일러	<ul style="list-style-type: none"> · 고급 언어로 작성된 프로그램 전체를 목적 프로그램으로 번역한 후, 링킹 작업을 통해 컴퓨터에서 실행 가능한 실행 프로그램 생성 · 번역 과정이 번거롭고, 번역 시간이 오래 걸리지만 실행 속도가 빠름 · FORTRAN, COBOL, PASCAL, C, C++, PL/1 등이 컴파일러를 사용함
인터프리터	<ul style="list-style-type: none"> · 고급 언어로 작성된 프로그램을 한 줄 단위로 받아들여 번역하고, 번역과 동시에 프로그램을 한 줄 단위로 즉시 실행시키는 프로그램 · 줄 단위로 번역 · 실행되기 때문에 시분할 시스템에 유용함 · 프로그램이 직접 실행되므로 목적 프로그램이 생성되지 않음 · 번역 속도는 빠르지만 실행 속도는 느림 · BASIC, SNOBOL, LISP, APL 등이 인터프리터를 사용함

핵심 109 | 매크로와 매크로 프로세서

- **매크로** : 프로그램 작성시 한 프로그램 내에서 동일한 코드가 반복될 경우 반복되는 코드를 한번만 작성하여 특정 이름으로 정의한 후 그 코드가 필요할 때마다 정의된 이름을 호출하여 사용하는 것
- **매크로 프로세서** : 원시 프로그램에 존재하는 매크로 호출 부분에 매크로(Macro) 프로그램을 삽입하여 확장된 원시 프로그램을 생성하는 시스템 소프트웨어
- **매크로 프로세서 처리 과정** : 매크로 정의 인식 → 매크로 정의 저장 → 매크로 호출 인식 → 매크로 확장 및 인수(매개 변수) 치환

핵심 110 | 로더

정의	컴퓨터 내부로 정보를 들여오거나 로드 모듈을 디스크 등의 보조 기억장치로부터 주기억장치에 적재하는 시스템 소프트웨어
기능	<ul style="list-style-type: none"> · 할당(Allocation) : 실행 프로그램을 실행시키기 위해 기억장치 내에 옮겨놓을 공간을 확보하는 기능 · 연결(Linking) : 부프로그램 호출시 그 부프로그램이 할당된 기억 장소의 시작 주소를 호출한 부분에 등록하여 연결하는 기능 · 재배치(Relocation) : 디스크 등의 보조기억장치에 저장된 프로그램이 사용하는 각 주소들을 할당된 기억 장소의 실제 주소로 배치시키는 기능 · 적재>Loading) : 실행 프로그램을 할당된 기억공간에 실제로 옮겨주는 기능

종류	<ul style="list-style-type: none"> • Compile And Go 로더 : 별도의 로더없이 언어 번역 프로그램이 로더의 기능까지 수행하는 방식(할당, 재배치, 적재 작업을 모두 언어 번역 프로그램이 담당) • 절대 로더(Absolute Loader) : 목적 프로그램을 기억 장소에 적재시키는 기능만 수행하는 로더(할당 및 연결은 프로그래머가 재배치는 언어 번역 프로그램이 담당) • 직접 연결 로더(Direct Linking Loader) : 일반적인 기능의 로더로, 로더의 기본 기능 4가지를 모두 수행하는 로더 • 동적 적재 로더(Dynamic Loading Loader) : 프로그램을 한꺼번에 적재하는 것이 아니라 실행시 필요한 일부분만을 적재하는 로더
----	---

핵심 111 | 링커

- 언어 번역 프로그램이 생성한 목적 프로그램들과 라이브러리, 또 다른 실행 프로그램(로드 모듈) 등을 연결하여 실행 가능한 로드 모듈을 만드는 시스템 소프트웨어
- 연결 기능만 수행하는 로더의 한 형태로, 링커에 의해 수행되는 작업을 링킹(Linking)이라 함

핵심 112 | 프로세스의 여러 가지 정의

- 실행중인 프로그램, PCB를 가진 프로그램, 실기억장치에 저장된 프로그램
- 프로세서가 할당되는 실체, 프로시저가 활동 중인 것,
- 비동기적 행위를 일으키는 주체, 지정된 결과를 얻기 위한 일련의 계통적 동작
- 목적 또는 결과에 따라 발생하는 사건들의 과정

핵심 113 | PCB

- PCB : 운영체제가 프로세스에 대한 중요한 정보를 저장해 놓는 곳
- PCB에 저장되어 있는 정보 : 프로세스의 현재 상태, 포인터, 프로세스 고유 식별자, 스케줄링 및 프로세서의 우선 순위, CPU 레지스터 정보, 주기억장치 관리 정보, 입·출력 상태 정보, 계정 정보

핵심 114 | 프로세스 상태 전이

- 프로세스의 주요 상태

준비(Read)	프로세스가 프로세서를 할당받기 위해 기다리고 있는 상태
실행(Run)	<ul style="list-style-type: none"> • 준비상태 큐에 있는 프로세스가 프로세서를 할당받아 실행되는 상태 • 프로세스 수행이 완료되기 전에 프로세스에게 주어진 프로세서 할당 시간이 종료(Time Run Out)되면 프로세스는 준비 상태로 전이됨 • 실행중인 프로세스에 입·출력(I/O) 처리가 필요하면 실행 중인 프로세스는 대기 상태로 전이됨
대기(Wait), 보류, 블록(Block)	프로세스에 입·출력 처리가 필요하면 현재 실행중인 프로세스가 중단되고, 입·출력 처리가 완료될 때까지 대기하고 있는 상태

• 프로세스 상태 전이 관련 용어

Dispatch	준비 상태에서 대기하고 있는 프로세스 중 하나가 프로세서를 할당 받아 실행 상태로 전이되는 과정
Wake-Up	입·출력 작업이 완료되어 프로세스가 대기 상태에서 준비 상태로 전이되는 과정

핵심 115 | 스케줄링

- 정의 : 프로세스가 생성되어 실행될때 필요한 시스템의 여러 자원을 해당 프로세스에게 할당하는 작업
- 목적 : 공정성, 처리율 증가, CPU 이용률 증가, 우선 순위 제도, 오버헤드 최소화, 응답 시간 최소화, 반환 시간 최소화, 대기 시간 최소화, 균형 있는 자원의 사용, 무한 연기 회피

핵심 116 | 프로세서 스케줄링의 종류

비선점(Non-preemptive) 스케줄링	<ul style="list-style-type: none"> • 이미 할당된 CPU를 다른 프로세스가 강제로 빼앗아 사용할 수 없는 스케줄링 기법 • 프로세스가 CPU를 할당 받으면 해당 프로세스가 완료될 때까지 CPU를 사용함 • 모든 프로세스에 대한 요구를 공정하게 처리할 수 있음 • 프로세스 응답 시간의 예측이 용이하며, 일괄 처리 방식에 적합함 • 중요한 작업(짧은 작업)이 중요하지 않은 작업(긴 작업)을 기다리는 경우가 발생할 수 있음 • 종류 : FCFS, SJF, 우선 순위, HRN, 기한부 등의 알고리즘
선점(Preemptive) 스케줄링	<ul style="list-style-type: none"> • 하나의 프로세스가 CPU를 할당받아 실행하고 있을 때 우선 순위가 높은 다른 프로세스가 CPU를 강제로 빼앗아 사용할 수 있는 스케줄링 기법 • 우선 순위가 높은 프로세스를 빠르게 처리할 수 있음 • 주로 빠른 응답 시간을 요구하는 대화식 시분할 시스템에 사용됨 • 종류 : SRT, 선점 우선 순위, Round Robin, 다단계 큐, 다단계 피드백 큐 등의 알고리즘

핵심 117 | 비선점 스케줄링의 종류

FCFS (First-Come First-Service)	<ul style="list-style-type: none"> • 준비상태 큐에 도착한 순서에 따라 차례로 CPU를 할당하는 기법 • 먼저 도착한 것이 먼저 처리되어 공정성은 유지되지만 짧은 작업이 긴 작업을, 중요한 작업이 중요하지 않은 작업을 기다리게 됨
SJF (Shortest Job First)	<ul style="list-style-type: none"> • 실행 시간이 가장 짧은 프로세스에 먼저 CPU를 할당하는 기법 • 가장 적은 평균 대기 시간을 제공하는 최적 알고리즘
HRN(Highest Response-ratio Next)	<ul style="list-style-type: none"> • 실행 시간이 긴 프로세스에 불리한 SJF 기법을 보완하기 위한 것으로, 대기 시간과 서비스(실행) 시간을 이용하는 기법 • 우선순위 계산 공식 = $\frac{\text{대기 시간} + \text{서비스(실행) 시간}}{\text{서비스(실행) 시간}}$

기한부 (Deadline)	<ul style="list-style-type: none"> 프로세스에게 일정한 시간을 주어 그 시간 안에 프로세스를 완료하도록 하는 기법 시스템은 프로세스에게 할당할 정확한 시간을 추정해야 하며, 이를 위해서 사용자는 시스템이 요구한 프로세스에 대한 정확한 정보를 제공해야 함
우선순위 (Priority)	<ul style="list-style-type: none"> 준비상태 큐에서 기다리는 각 프로세스마다 우선 순위를 부여하여 그 중 가장 높은 프로세스에게 먼저 CPU를 할당하는 기법

핵심 118 | 에이징(Aging) 기법

- 시스템에서 특정 프로세스의 우선 순위가 낮아 무한정 기다리게 되는 경우, 한번 양보하거나 기다린 시간에 비례하여 일정 시간이 지나면 우선 순위를 한 단계씩 높여 가까운 시간 안에 자원을 할당받도록 하는 기법
- SJF나 우선 순위 기법에서 발생할 수 있는 무한 연기 상태, 기아 상태를 예방할 수 있음

핵심 119 | 선점 스케줄링의 종류

선점 우선 순위	준비상태 큐의 프로세스들 중에서 우선 순위가 가장 높은 프로세스에게 먼저 CPU를 할당하는 기법
SRT(Shortest Remaining Time)	비선점 기법인 SJF 알고리즘을 선점 형태로 변형한 기법으로, 현재 실행중인 프로세스의 남은 시간과 준비상태 큐에 새로 도착한 프로세스의 실행 시간을 비교하여 가장 짧은 실행 시간을 요구하는 프로세스에게 CPU를 할당하는 기법
RR (Round Robin)	<ul style="list-style-type: none"> 시분할 시스템(Time Sharing System)을 위해 고안된 방식으로, FCFS 알고리즘을 선점 형태로 변형한 기법 FCFS 기법과 같이 준비상태 큐에 먼저 들어온 프로세스가 먼저 CPU를 할당받지만 각 프로세스는 할당된 시간(Time Slice, Quantum) 동안만 실행한 후 실행이 완료되지 않으면 다음 프로세스에게 CPU를 넘겨주고 준비상태 큐의 가장 뒤로 배치됨 할당되는 시간이 클 경우 FCFS 기법과 같아지고, 할당되는 시간이 작을 경우 문맥교환 및 오버헤드가 자주 발생됨
다단계 큐 (Multi level Queue)	프로세스를 특정 그룹으로 분류할 수 있을 경우 그룹에 따라 각기 다른 준비상태 큐를 사용하는 기법
다단계 피드백 큐(Multi level Feedback Queue)	특정 그룹의 준비상태 큐에 들어간 프로세스가 다른 준비상태 큐로 이동할 수 없는 다단계 큐 기법을 준비상태 큐 사이를 이동할 수 있도록 개선한 기법

핵심 120 | 임계 구역(Critical Section)

- 다중 프로그래밍 운영체제에서 여러 개의 프로세스가 공유하는 데이터 및 자원에 대하여 어느 한 시점에서는 하나의 프로세스만 자원 또는 데이터를 사용하도록 지정된 공유 자원(영역)
- 임계 구역에는 하나의 프로세스만 접근할 수 있으며, 해당 프로세스가 자원을 반납한 후에만 다른 프로세스가 자원이나 데이터를 사용할 수 있음

핵심 121 | 상호 배제(Mutual Exclusion)

- 특정 프로세스가 공유 자원을 사용하고 있을 경우 다른 프로세스가 해당 공유 자원을 사용하지 못하게 제어하는 기법
- 여러 프로세스가 동시에 공유 자원을 사용하려 할 때 각 프로세스가 번갈아 가며 공유 자원을 사용하도록 하는 것으로 임계 구역을 유지하는 기법

핵심 122 | 세마포어(Semaphore)

- 각 프로세스에 제어 신호를 전달하여 순서대로 작업을 수행하도록 하는 기법
- E.J.Dijkstra가 제안하였으며, P와 V라는 2개의 연산에 의해서 동기화를 유지시키고, 상호 배제의 원리를 보장함
- S는 P와 V 연산으로만 접근 가능한 세마포어 변수로, 공유 자원의 개수를 나타내며 0과 1 혹은 0과 양의 값을 가질 수 있음
- P 연산** : 자원을 사용하려는 프로세스들의 진입 여부를 자원의 개수(S)를 통해 결정하는 것으로, wait 동작이라 함
- V 연산** : 대기 중인 프로세스를 깨우는 신호(Wake Up)로서, signal 동작이라 함

핵심 123 | 모니터(Monitor)

- 동기화를 구현하기 위한 특수 프로그램 기법으로 특정 공유 자원을 프로세스에게 할당하는데 필요한 데이터와 이 데이터를 처리하는 프로시저로 구성됨
- 자료 추상화와 정보 은폐 개념을 기초로 하며 공유 자원을 할당하기 위한 병행성 구조로 이루어져 있음
- 모니터 내의 공유 자원을 사용하려면 프로세스는 반드시 모니터의 진입부를 호출해야 함
- 외부의 프로시저는 직접 액세스할 수 없으며, 모니터의 경계에서 상호 배제가 시행됨
- 한순간에 하나의 프로세스만 진입하여 자원을 사용할 수 있음

핵심 124 | 교착 상태(Deadlock)

정의	상호 배제에 의해 나타나는 문제점으로, 둘 이상의 프로세스들이 자원을 점유한 상태에서 서로 다른 프로세스가 점유하고 있는 자원을 요구하며 무한정 기다리는 현상
필요충분조건	<ul style="list-style-type: none"> 상호배제(Mutual Exclusion) : 한 번에 한 개의 프로세스만이 공유 자원을 사용할 수 있어야 함 점유와 대기(Hold & Wait) : 최소한 하나의 자원을 점유하고 있으면서 다른 프로세스에 할당되어 사용되고 있는 자원을 추가로 점유하기 위해 대기하는 프로세스가 있어야 함 비선점(Non-preemptive) : 다른 프로세스에 할당된 자원은 사용이 끝날 때까지 강제로 빼앗을 수 없어야 함 환형대기(Circular Wait) : 공유 자원과 공유 자원을 사용하기 위해 대기하는 프로세스들이 환형으로 구성되어 있어 자신에게 할당된 자원을 점유하면서 앞이나 뒤에 있는 프로세스의 자원을 요구해야 함

핵심 125 | 교착 상태 해결 방법

- **예방 기법(Prevention)** : 교착 상태가 발생되지 않도록 사전에 시스템을 제어하는 방법으로, 교착 상태 발생의 4가지 조건 중에서 상호 배제를 제외한 어느 하나를 제거(부정)함으로써 수행됨

점유 및 대기 부정	프로세스가 실행되기 전 필요한 모든 자원을 할당하여 프로세스 대기를 없애거나 자원이 점유되지 않은 상태에서 자원 요구하도록 함
비선점 부정	자원을 점유하고 있는 프로세스가 다른 자원을 요구할 때 점유하고 있는 자원을 반납하고, 요구한 자원을 사용하기 위해 기다리게 함
환형 대기 부정	자원을 선형 순서로 분류하여 고유 번호를 할당하고, 각 프로세스는 현재 점유한 자원의 고유 번호보다 앞이나 뒤 어느 한 쪽 방향으로만 자원을 요구하도록 하는 것

- **회피 기법(Avoidance)** : 교착 상태가 발생할 가능성을 배제하지 않고, 교착 상태가 발생하면 적절히 피해나가는 방법으로, 주로 은행원 알고리즘(Banker's Algorithm)이 사용됨

은행원 알고리즘	<ul style="list-style-type: none"> · Dijkstra가 제안한 것으로, 은행에서 모든 고객의 요구가 충족되도록 현금을 할당하는데서 유래한 기법 · 각 프로세스에게 자원을 할당하여 교착 상태가 발생하지 않으며 모든 프로세스가 완료될 수 있는 상태를 안전 상태, 교착 상태가 발생할 수 있는 상태를 불안전 상태라고 함
----------	---

- **발견 기법(Detection)** : 시스템에 교착 상태가 발생했는지 점검하여 교착 상태에 있는 프로세스와 자원을 발견하는 것

- **회복 기법(Recovery)** : 교착 상태를 일으킨 프로세스를 종료하거나 교착상태의 프로세스에 할당된 자원을 선점하여 프로세스나 자원을 회복하는 것

핵심 126 | 기억장치 관리 전략

- **반입(Fetch) 전략** : 보조기억장치에 보관 중인 프로그램이나 데이터를 언제 주기억장치로 적재할 것인지를 결정하는 전략

요구 반입	실행 중인 프로그램이 특정 프로그램이나 데이터 등의 참조를 요구할 때 적재하는 방법
예상 반입	실행 중인 프로그램에 의해 참조될 프로그램이나 데이터를 미리 예상하여 적재하는 방법

- **배치(Placement) 전략** : 새로 반입되는 프로그램이나 데이터를 주기억장치의 어디에 위치시킬 것인지를 결정하는 전략

최초 적합 (First Fit)	프로그램이나 데이터가 들어갈 수 있는 크기의 빈 영역 중에서 첫 번째 분할 영역에 배치시키는 방법
최적 적합 (Best Fit)	프로그램이나 데이터가 들어갈 수 있는 크기의 빈 영역 중에서 단편화를 가장 작게 남기는 분할 영역에 배치시키는 방법
최악 적합 (Worst Fit)	프로그램이나 데이터가 들어갈 수 있는 크기의 빈 영역 중에서 단편화를 가장 많이 남기는 분할 영역에 배치시키는 방법

- **교체(Replacement) 전략** : 주기억장치의 모든 영역이 이미 사용중인 상태에서 새로운 프로그램이나 데이터를 주기억장치에 배치하려고 할 때, 이미 사용되고 있는 영역 중에서 어느 영역을 교체하여 사용할 것인지를 결정하는 전략으로, FIFO, OPT, LRU, LFU, NUR, SCR 등이 있음

핵심 127 | 단일 분할 할당 기법

- **주기억장치를 운영체제 영역과 사용자 영역으로 나누어 한 순간에는 오직 한 명의 사용자만이 주기억장치의 사용자 영역을 사용하는 기법**
- **오버레이(Overlay) 기법** : 주기억장치보다 큰 사용자 프로그램을 실행하기 위한 기법으로, 보조기억장치에 저장된 하나의 프로그램을 여러 개의 조각으로 분할한 후 필요한 조각을 차례로 주기억장치에 적재하여 프로그램을 실행함
- **스와핑(Swapping) 기법** : 하나의 프로그램 전체를 주기억장치에 할당하여 사용하다 필요에 따라 다른 프로그램과 교체하는 기법

핵심 128 | 다중 분할 할당 기법

- **고정 분할 할당** : 프로그램을 할당하기 전에 운영체제가 주기억장치의 사용자 영역을 여러 개의 고정된 크기로 분할하고, 준비상태 큐에서 준비 중인 프로그램을 각 영역에 할당하여 수행하는 기법
- **가변 분할 할당** : 고정 분할 할당 기법의 단편화를 줄이기 위한 것으로, 미리 주기억장치를 분할해 놓는 것이 아니라 프로그램을 주기억장치에 적재하면서 필요한 만큼의 크기로 영역을 분할하는 기법

핵심 129 | 단편화(Fragmentation)

- **분할된 주기억장치에 프로그램을 할당하고 반납하는 과정을 반복하면서 사용되지 않고 남는 기억장치의 빈 공간 조각**
- **내부(Internal) 단편화** : 분할된 영역이 할당될 프로그램의 크기보다 크기 때문에 프로그램이 할당된 후 사용되지 않고 남아 있는 빈 공간
- **외부(External) 단편화** : 분할된 영역이 할당될 프로그램의 크기보다 작기 때문에 프로그램이 할당될 수 없어 사용되지 않고 빈 공간으로 남아 있는 분할된 전체 영역

핵심 130 | 단편화 해결 방법

- **통합(Coalescing) 기법** : 주기억장치 내에 인접해 있는 단편화된 공간을 하나의 공간으로 통합하는 작업
- **압축(Compaction) 기법, 집약** : 주기억장치 내에 분산되어 있는 단편화된 빈 공간을 결합하여 하나의 큰 가용 공간을 만드는 작업으로, 여러 위치에 분산된 단편화된 빈 공간을 주기억장치의 한쪽 끝으로 옮겨서 큰 기억 공간을 만듦

핵심 131 | 가상기억장치 구현 기법

페이징 (Paging) 기법	<ul style="list-style-type: none"> · 가상기억장치에 보관되어 있는 프로그램과 주기억장치의 영역을 동일한 크기로 나눈 후 나뉜 프로그램(페이지)을 동일하게 나뉜 주기억장치의 영역(페이지 프레임)에 적재시켜 실행하는 기법 · 외부 단편화는 발생하지 않으나 내부 단편화는 발생할 수 있음
세그먼트이션 (Segmentation) 기법	<ul style="list-style-type: none"> · 가상기억장치에 보관되어 있는 프로그램을 다양한 크기의 논리적인 단위로 나눈 후 주기억장치에 적재시켜 실행시키는 기법 · 프로그램을 배열이나 함수 등과 같은 논리적인 크기로 나눈 단위를 세그먼트라고 하며, 각 세그먼트는 고유한 이름과 크기를 갖고 있음 · 다른 세그먼트에게 할당된 영역을 침범할 수 없으며, 이를 위해 기억장치 보호키(Storage Protection Key)가 필요함

핵심 132 | 페이지 교체 알고리즘

OPT (Optimal Replacement, 최적 교체)	<ul style="list-style-type: none"> 앞으로 가장 오랫동안 사용하지 않을 페이지를 교체하는 기법 각 페이지의 호출 순서와 참조 상황을 미리 예측해야 하므로 실현 가능성이 희박함
FIFO(First In First Out)	<ul style="list-style-type: none"> 각 페이지가 주기억장치에 적재될 때마다 그때의 시간을 기억시켜 가장 먼저 들어와서 가장 오래 있었던 페이지를 교체하는 기법 이해하기 쉽고, 프로그래밍 및 설계가 간단하며, 벨레이디의 모순(Belady's Anomaly) 현상이 발생함
LRU(Least Recently Used)	<ul style="list-style-type: none"> 최근에 가장 오랫동안 사용하지 않은 페이지를 교체하는 기법 각 페이지마다 계수기나 스택을 두어 현 시점에서 가장 오랫동안 사용하지 않은 쪽, 가장 오래 전에 사용된 페이지를 교체함
LFU(Least Frequently Used)	<ul style="list-style-type: none"> 사용 빈도가 가장 적은 페이지를 교체하는 기법 프로그램 실행 초기에 많이 사용된 페이지가 그 후로 사용되지 않을 경우에도 프레임에 계속 차지할 수 있음
NUR (Not Used Recently)	<ul style="list-style-type: none"> 최근에 사용하지 않은 페이지를 교체하는 기법 최근의 사용 여부를 확인하기 위해서 각 페이지마다 참조 비트(Reference Bit)와 변형 비트(Modified Bit, Dirty Bit)가 사용됨
SCR(Second Chance Replacement)	가장 오랫동안 주기억장치에 있던 페이지 중 자주 사용되는 페이지의 교체를 방지하기 위한 것으로, FIFO 기법의 단점을 보완하는 기법

핵심 133 | 페이지 크기

페이지가 작을 경우	<ul style="list-style-type: none"> 페이지의 단편화가 감소되고, 한 개의 페이지를 주기억장치로 이동하는 시간이 줄어듦 프로세스(프로그램) 수행에 필요한 내용만 주기억장치에 적재할 수 있고, Locality(국부성)에 더 일치할 수 있기 때문에 기억장치 효율이 높아짐 페이지 정보를 갖는 페이지 맵 테이블의 크기가 커지고, 맵핑 속도가 늦어짐 디스크 접근 횟수가 많아져서 전체적인 입·출력 시간은 늘어남
페이지가 클 경우	<ul style="list-style-type: none"> 페이지 정보를 갖는 페이지 맵 테이블의 크기가 작아지고, 맵핑 속도가 빨라짐 디스크 접근 횟수가 줄어들어 전체적인 입·출력의 효율성이 증가됨 페이지의 단편화가 증가되고, 한 개의 페이지를 주기억장치로 이동하는 시간이 늘어남 프로그램 수행에 불필요한 내용까지도 주기억장치에 적재될 수 있음

핵심 134 | 국부성(Locality, 구역성)

- 프로세스가 실행되는 동안 주기억장치를 참조할 때 일부 페이지만 집중적으로 참조하는 성질이 있다는 이론
- 스래싱을 방지하기 위한 워킹 셋 이론의 기반이 됨
- 프로세스가 집중적으로 사용하는 페이지를 알아내는 방법 중 하나로, 가상 기억장치 관리의 이론적인 근거가 됨
- Locality 종류

시간 구역성(Temporal Locality)	<ul style="list-style-type: none"> 프로세스가 실행되면서 하나의 페이지를 일정 시간 동안 집중적으로 액세스하는 현상 시간 구역성이 이루어지는 기억 장소 : Loop(반복, 순환), 스택(Stack), 부프로그램(Sub Routine), Counting(1씩 증감, Totalling(집계)에 사용되는 변수(기억 장소))
공간 구역성(Spatial Locality)	<ul style="list-style-type: none"> 프로세스 실행시 일정 위치의 페이지를 집중적으로 액세스하는 현상 공간 구역성이 이루어지는 기억 장소 : 배열 순회(Array Traversal), 순차적 코드의 실행, 프로그래머들이 관련된 변수(데이터를 저장할 기억주소)들을 서로 근처에 선언하여 할당되는 기억주소, 같은 영역에 있는 변수를 참조할 때 사용

핵심 135 | 워킹 셋/페이지 부재 빈도

- 워킹 셋(Working Set)** : 프로세스가 일정 시간 동안 자주 참조하는 페이지들의 집합으로, 자주 참조되는 워킹 셋을 주기억장치에 상주 시킴으로써 페이지 부재 및 페이지 교체 현상을 줄임
- 페이지 부재 빈도(PFF, Page Fault Frequency)** : 페이지 부재가 일어나는 횟수
- 페이지 부재 빈도 방식** : 페이지 부재율(Page Fault Rate)에 따라 주기억장치에 있는 페이지 프레임의 수를 늘리거나 줄여 페이지 부재율을 적정 수준으로 유지하는 방식

핵심 136 | 스래싱(Thrashing)

- 프로세스의 처리 시간보다 페이지 교체 시간이 더 많아지는 현상
- 다중 프로그래밍 시스템이나 가상기억장치를 사용하는 시스템에서 하나의 프로세스 수행 과정 중 자주 페이지 부재가 발생함으로 인해 나타나는 현상으로 전체 시스템의 성능이 저하됨
- 다중 프로그래밍의 정도가 높아짐에 따라 CPU의 이용율은 어느 특정 시점까지는 높아지지만 다중 프로그래밍의 정도가 더욱 커지면 스래싱이 나타나고, CPU의 이용율은 급격히 감소됨
- CPU 이용율을 높이고, 스래싱 현상을 방지하려면 다중 프로그래밍의 정도를 적정 수준으로 유지하고, 페이지 부재 빈도(Page Fault Frequency)를 조절하여 사용하며, Working Set을 유지해야 함

핵심 137 | 디스크 스케줄링

FCFS (First-Come First-Serve)	<ul style="list-style-type: none"> 가장 간단한 스케줄링으로, 디스크 대기 큐에 가장 먼저 들어온 트랙에 대한 요청을 먼저 서비스하는 기법 디스크 대기 큐에 들어온 순서대로 서비스하기 때문에 더 높은 우선 순위의 요청이 입력되어도 순서가 바뀌지 않아 공정성이 보장됨
SSTF (Shortest-Seek-Time-First)	<ul style="list-style-type: none"> 탐색 거리가 가장 짧은 트랙에 대한 요청을 먼저 서비스하는 기법 현재 헤드 위치에서 가장 가까운 거리에 있는 트랙으로 헤드를 이동시킴 FCFS보다 처리량이 많고, 평균 탐색 시간이 짧음 처리량이 많은 일괄 처리 시스템에 유용함
SCAN	<ul style="list-style-type: none"> SSTF가 갖는 탐색 시간의 편차를 해소하기 위한 기법 현재 헤드의 위치에서 진행 방향이 결정되면 탐색 거리가 짧은 순서에 따라 그 방향의 모든 요청을 서비스하고, 끝까지 이동한 후 역방향의 요청 사항을 서비스함

C-SCAN (Circular SCAN)	<ul style="list-style-type: none"> · 항상 바깥쪽에서 안쪽으로 움직이면서 가장 짧은 탐색 거리를 갖는 요청을 서비스하는 기법 · 헤드는 트랙의 바깥쪽에서 안쪽으로 한 방향으로만 움직이며 서비스하여 끝까지 이동한 후, 안쪽에 더 이상의 요청이 없으면 헤드는 가장 바깥쪽의 끝으로 이동한 후 다시 안쪽으로 이동하면서 요청을 서비스함
N-step SCAN	SCAN 기법을 기초로 하며, 어떤 방향의 진행이 시작될 당시에 대기 중이던 요청들만 서비스하고, 진행 도중 도착한 요청들은 한데 모아서 다음의 반대 방향 진행 때 서비스하는 기법

핵심 138 | 파일 시스템의 기능

- 사용자와 보조기억장치 사이에서 인터페이스 제공
- 사용자가 파일을 생성, 수정, 제거할 수 있도록 함
- 적절한 제어 방식을 통해 타인의 파일을 공동으로 사용할 수 있도록 함
- 사용자가 적합한 구조로 파일을 구성할 수 있도록 함
- 불의의 사태를 대비하여 파일의 예비와 복구 등의 기능을 제공함

핵심 139 | 파일 디스크립터(File Descriptor)

- 파일을 관리하기 위한 시스템(운영체제)이 필요로 하는 파일에 대한 정보를 갖고 있는 제어 블록(파일 제어 블록, FCB)
- 보통 파일 디스크립터는 보조기억장치 내에 저장되어 있다가, 해당 파일이 Open될때 주기억장치로 옮겨짐
- 파일 시스템이 관리하므로 사용자가 직접 참조할 수 없음
- **파일 디스크립터의 정보** : 파일 이름, 보조기억장치에서의 파일 위치, 파일 구조, 보조기억장치의 유형, 액세스 제어 정보, 파일 유형, 생성 날짜와 시간, 제거 날짜와 시간, 최종 수정 날짜 및 시간, 액세스한 횟수

핵심 140 | 순차 파일(Sequential File), 순차 접근 방식

- 레코드를 논리적인 처리 순서에 따라 연속된 물리적 저장 공간에 기록하는 것
- 주로 순차 접근이 가능한 자기 테이프에서 사용
- **장점** : 파일의 구성 용이, 기억 공간의 이용 효율 높음, 접근 속도 빠름
- **단점** : 파일에 새로운 레코드를 삽입하거나 삭제하는 경우 시간이 많이 걸림, 검색 효율이 낮음

핵심 141 | 직접 파일(Direct File), 직접 접근 방식

- 파일을 구성하는 레코드를 임의의 물리적 저장 공간에 기록하는 것
- 레코드에 특정 기준으로 키가 할당되며, 해싱 함수(Hashing Function)를 이용하여 이 키에 대한 보조기억장치의 물리적 상대 레코드 주소를 계산한 후 해당하는 주소에 레코드를 저장함
- 레코드는 해싱 함수에 의해 계산된 물리적 주소를 통해 접근 가능
- 임의의 접근이 가능한 자기 디스크나 자기 드럼 사용
- **장점** : 파일의 각 레코드에 직접 접근하거나 기록할 수 있음, 접근 시간이 빠르고, 레코드의 삽입, 삭제, 갱신이 용이함
- **단점** : 레코드의 주소 변환 과정이 필요하며, 이 과정으로 인해 시간이 소요됨, 기억 공간의 효율이 저하됨

핵심 142 | 색인 순차 파일(Indexed Sequential File)

- 순차 파일과 직접 파일에서 지원하는 편성 방법이 결합된 형태
- 각 레코드를 키값 순으로 논리적으로 저장하고, 시스템은 각 레코드의 실제 주소가 저장된 색인을 관리함
- 레코드를 참조하려면 색인을 탐색한 후 색인이 가리키는 포인터(주소)를 사용하여 직접 참조할 수 있음
- 기본 영역, 색인 영역, 오버 플로우 영역으로 구성되며, 색인 영역은 트랙 색인 영역, 실린더 색인 영역, 마스터 색인 영역으로 분류됨
- **장점** : 순차 처리와 임의 처리가 모두 가능, 효율적인 검색 기능, 삭제, 삽입, 갱신이 용이함
- **단점** : 기억 공간이 필요함, 접근 시간이 직접 파일보다 느림

핵심 143 | 디렉토리의 구조

1단계 디렉토리	<ul style="list-style-type: none"> · 가장 간단하고, 모든 파일이 하나의 디렉토리 내에 위치하여 관리되는 구조 · 모든 파일들이 유일한 이름을 가지고 있어야 함 · 모든 파일이 같은 디렉토리내에 유지되므로 이해가 용이하지만, 파일의 수나 사용자의 수가 증가하면 파일 관리가 복잡해짐
2단계 디렉토리	<ul style="list-style-type: none"> · 중앙에 마스터 파일 디렉토리가 있고, 그 아래에 사용자별로 서로 다른 파일 디렉토리가 있는 2 계층 구조 · 마스터 파일 디렉토리는 사용자 파일 디렉토리를 관리하고, 사용자 파일 디렉토리는 사용자별 파일을 관리함 · 서로 다른 디렉토리에서는 동일한 파일 이름을 사용할 수 있음
트리 디렉토리	<ul style="list-style-type: none"> · 하나의 루트 디렉토리와 여러 개의 종속(서브) 디렉토리로 구성된 구조 · DOS, Windows, UNIX 등의 운영체제에서 사용되는 디렉토리 구조 · 동일한 이름의 파일이나 디렉토리를 생성할 수 있음 · 디렉토리의 생성과 파괴가 비교적 용이함
비순환 그래프 디렉토리	<ul style="list-style-type: none"> · 하위 파일이나 하위 디렉토리를 공동으로 사용할 수 있는 것으로, 사이클이 허용되지 않는 구조 · 디스크 공간을 절약할 수 있음 · 하나의 파일이나 디렉토리가 여러 개의 경로 이름을 가질 수 있음 · 공유된 파일을 삭제할 경우 고아 포인터(Dangling Pointer)가 발생할 수 있음
일반적인 그래프 디렉토리	<ul style="list-style-type: none"> · 트리 구조에 링크(Link)를 첨가시켜 순환을 허용하는 그래프 구조 · 디렉토리와 파일 공유에 완전한 융통성이 있음 · 불필요한 파일을 제거하여 사용 공간을 늘리기 위하여 참조 계수기가 필요함

핵심 144 | 디스크 공간 할당 방법

- **연속 할당** : 파일을 디스크의 연속된 기억 공간에 할당하는 방법으로, 생성되는 파일 크기만큼의 공간을 있어야 함
- **불연속 할당** : 파일의 크기가 변경될 경우 구현이 어려운 연속 할당의 단점을 보완하기 위한 것으로, 디스크 공간을 일정 단위로 나누어 할당하는 기법

섹터 단위 할당	하나의 파일이 디스크의 섹터 단위로 분산되어 할당되는 방법으로, 하나의 파일에 속하는 섹터들이 연결 리스트(Linked List)로 구성되어 있음
----------	---

블록 단위 할당	<ul style="list-style-type: none"> 하나의 파일이 연속된 여러 개의 섹터를 묶은 블록 단위로 할당되는 방법 블록 단위 할당에는 블록 체인 할당, 색인 블록 체인 할당, 블록 지향 파일 사상 기법이 있음
----------	---

핵심 145 | 자원 보호 기법

- **접근 제어 행렬(Access Control Matrix)** : 자원 보호의 일반적인 모델로, 객체에 대한 접근 권한을 행렬로써 표시한 기법
- **전역 테이블(Global Table)** : 가장 단순한 구현 방법으로, 3개의 순서쌍인 영역, 객체, 접근 권한의 집합을 목록 형태로 구성한 기법
- **접근 제어 리스트(Access Control List)** : 접근 제어 행렬에 있는 각 열, 즉 객체를 중심으로 접근 리스트를 구성한 기법
- **권한(자격) 리스트(Capability List)** : 접근 제어 행렬에 있는 각 행, 즉 영역을 중심으로 권한 리스트를 구성한 기법

핵심 146 | 파일 보호 기법

- **파일의 명명(Naming)** : 접근하고자 하는 파일 이름을 모르는 사용자를 접근 대상에서 제외시키는 기법
- **비밀번호(Password, 암호)** : 각 파일에 판독 암호와 기록 암호를 부여하여 암호를 아는 사용자에게만 접근을 허용하는 기법
- **접근 제어(Access Control)** : 사용자에게 따라 공유 데이터에 접근할 수 있는 권한을 제한하는 방법

핵심 147 | 보안 유지 기법

외부 보안	<ul style="list-style-type: none"> 시설 보안 : 천재 지변이나 외부 침입자로부터의 보안 운용 보안 : 전산소 관리 및 경영자들의 정책과 통제에 의해 이루어지는 보안
사용자 인터페이스 보안	운영체제가 사용자의 신원을 확인한 후 권한이 있는 사용자에게만 시스템의 프로그램과 데이터를 사용할 수 있게 하는 보안 기법
내부 보안	하드웨어나 운영체제의 내장된 보안 기능을 이용하여 시스템의 신뢰성을 유지하고, 보안 문제를 해결하는 기법

핵심 148 | 암호화 기법

- 데이터를 보낼 때 송신자가 지정한 수신자 이외에는 그 내용을 알 수 없도록 평문을 암호문으로 변환하는 것
- **비밀키 시스템(Private Key System, 개인키 시스템)** : 동일한 키로 데이터를 암호화하고, 해독(복호화)하는 대칭 암호화 기법으로, 대표적인 암호화 방식에는 DES가 있음
- **공개키 시스템(Public Key System, 공개키 시스템)** : 서로 다른 키로 데이터를 암호화하고, 해독하는 비대칭 암호화 기법으로, 대표적인 암호화 방식에는 RSA가 있음

핵심 149 | 프로세서 연결 방식

하이퍼 큐브	<ul style="list-style-type: none"> 다수의 프로세서들을 연결하는 방식으로 비교적 경제적인 방식 다수의 프로세서를 연결할 수 있으며, 확장성이 좋음 하나의 프로세서에 연결되는 다른 프로세서의 수(연결점)가 n개 일 경우 프로세서는 총 2ⁿ개가 필요함
--------	---

시분할 및 공유 버스	<ul style="list-style-type: none"> 프로세서, 주변장치, 기억장치 등의 각종 장치들간을 '버스'라는 단일 경로로 연결한 방식 버스에 이상이 발생하면 전체 시스템이 가동되지 않음 장치 연결이 단순하고, 경제적이며, 융통성이 있음 장치 추가가 용이하지만 버스에 이상이 발생하면 전체 시스템이 가동되지 않음
크로스바 교환 행렬	<ul style="list-style-type: none"> 시분할 및 공유 버스 구조에서 버스의 수를 기억장치 수만큼 증가시켜 연결한 방식 각 기억장치마다 다른 경로를 사용할 수 있음
다중 포트 기억장치	<ul style="list-style-type: none"> 시분할 및 공유 버스 방식과 크로스바 교환 행렬을 혼합한 형태의 방식 많은 수의 프로세서를 쉽게 연결할 수 있음 다양한 연결이 가능하며, 전송 시간이 비교적 느림

핵심 150 | 다중 처리기의 운영체제 구조

주 / 종 처리기	<ul style="list-style-type: none"> 하나의 프로세서를 Master(주프로세서)로 지정하고, 나머지는 Slave(종프로세서)로 지정하는 구조 주프로세서가 고장나면 전체 시스템이 다운됨 주프로세서 : 입·출력과 연산 담당, 운영체제 수행 종프로세서 : 연산만 담당
분리 실행 처리기	<ul style="list-style-type: none"> 주/종 처리기의 비대칭성을 보완하여 각 프로세서가 독자적인 운영체제를 가지고 있도록 구성한 구조 각 프로세서에서 발생하는 인터럽트는 해당 프로세서에서 해결 각 프로세서가 독자적인 운영체제를 가지고 있기 때문에 한 프로세서가 고장나더라도 전체 시스템이 다운되지 않음
대칭적 처리기	<ul style="list-style-type: none"> 여러 프로세서들이 완전한 기능을 갖춘 하나의 운영체제를 공유하여 수행하는 구조 가장 복잡한 구조를 가지고 있으나 가장 강력한 시스템임 여러 개의 프로세서가 동시에 수행할 수 있고, 시스템의 전반적인 정보를 통일적이고 일관성 있게 운영함

핵심 151 | 프로세서의 결합도

약결합 (Loosely Coupled) 시스템	<ul style="list-style-type: none"> 각 프로세서마다 독립된 메모리를 가진 시스템으로, 분산 처리 시스템이라고도 함 둘 이상의 독립된 컴퓨터 시스템을 통신망(통신 링크)을 통하여 연결한 시스템 각 시스템마다 독자적인 운영체제를 가지고 있음 각 시스템은 독립적으로 작동할 수도 있고, 필요한 경우에는 상호 통신할 수도 있음 프로세서 간의 통신은 메시지 전달이나 원격 프로세서 호출을 통해서 이루어짐 각 시스템마다 독자적인 운영이 가능하므로 CPU 간의 결합력이 약함
강결합 (Tightly Coupled) 시스템	<ul style="list-style-type: none"> 동일 운영체제 하에서 여러 개의 프로세서가 하나의 메모리를 공유하여 사용하는 시스템으로, 다중(병렬) 처리 시스템이라고도 함 하나의 운영체제가 모든 프로세서와 시스템 하드웨어를 제어함 프로세서 간의 통신은 공유 메모리를 통해서 이루어짐 하나의 메모리를 사용하므로 CPU 간의 결합력이 강함

핵심 152 | 분산 처리 시스템의 목적/장·단점

- **목적** : 자원 공유, 연산 속도 향상, 신뢰도 향상, 컴퓨터 통신
- **장점** : 통신 용이, 장치 공유, 데이터 공유, 중앙 컴퓨터 과부하 줄임, 컴퓨터의 위치를 몰라도 자원 사용 가능, 시스템의 점진적 확장 가능 등
- **단점** : 중앙 집중형 시스템에 비해 소프트웨어 개발이 어려움, 보안 문제 발생, 설계 복잡 등

핵심 153 | 분산 처리 시스템의 투명성

- **투명성(Transparency)** : 사용자가 분산된 시스템에 위치한 여러 자원을 사용할 때 각 자원의 위치 정보를 알지 못하고 마치 하나의 커다란 시스템을 사용하는 것처럼 인식하도록 하는 것
- 여러 유형의 투명성을 통해 자원의 위치나 정보가 변경되더라도 사용자가 이를 인식하지 못하게 됨
- **투명성의 종류** : 위치 투명성, 이주 투명성, 복제 투명성, 병행 투명성, 접근 투명성, 성능 투명성, 규모 투명성, 고장 투명성

핵심 154 | 위상에 따른 분산 처리 시스템의 분류

완전 연결 (Fully Connection)형	<ul style="list-style-type: none"> · 각 사이트들이 시스템 내의 다른 모든 사이트들과 직접 연결된 구조 · 사이트의 수가 n개이면 링크(연결) 수는 $n(n-1)/2$ 개임 · 기본 비용은 많이 들지만 통신 비용은 적게 들고, 신뢰성이 높음
부분 연결 (Partially Connection)형	<ul style="list-style-type: none"> · 시스템 내의 일부 사이트들 간에만 직접 연결된 형태로, 직접 연결되지 않은 사이트는 연결된 다른 사이트를 통해 통신하는 구조 · 기본 비용은 완전 연결형보다 적게 들고, 통신 비용은 완전 연결형보다 많이 듦 · 완전 연결형보다 신뢰성이 낮음
트리(Tree)/계층(Hierarchy)형	<ul style="list-style-type: none"> · 분산 처리 시스템의 가장 대표적인 형태로, 각 사이트들이 트리 형태로 연결된 구조 · 기본 비용은 부분 연결형보다 적게 들고, 통신 비용은 트리의 깊이에 비례함 · 부모(상위) 사이트의 자식(하위) 사이트들은 그 부모 사이트를 통해 통신이 이루어짐 · 부모 사이트가 고장되면 그 자식 사이트들은 통신이 불가능함
스타(Star)형/성형	<ul style="list-style-type: none"> · 모든 사이트가 하나의 중앙 사이트에 직접 연결되어 있고, 그 외의 다른 사이트와는 연결되어 있지 않은 구조 · 기본 비용은 사이트의 수에 비례하며, 통신 비용은 적게 듦 · 중앙 사이트를 제외한 사이트의 고장이 다른 사이트에 영향을 미치지 않지만, 중앙 사이트가 고장 날 경우 모든 통신이 단절됨
링(Ring)형/환형	<ul style="list-style-type: none"> · 시스템 내의 각 사이트가 인접하는 다른 두 사이트와만 직접 연결된 구조 · 정보는 단방향 또는 양방향으로 전달될 수 있음 · 기본 비용은 사이트 수에 비례하고, 목적 사이트에 데이터를 전달하기 위해 링을 순환할 경우 통신 비용이 증가함

다중 접근 버스 연결(Multi Access Bus Connection)형	<ul style="list-style-type: none"> · 시스템 내의 모든 사이트들이 공유 버스에 연결된 구조 · 기본 비용은 사이트 수에 비례하고, 통신 비용은 일반적으로 저렴함 · 사이트의 고장은 다른 사이트의 통신에 영향을 주지 않지만, 버스의 고장은 전체 시스템에 영향을 줌
---	---

핵심 155 | UNIX의 특징

- 시분할 시스템을 위해 설계된 대화식 운영체제로, 소스가 공개된 개방형 시스템(Open System)
- 대부분 C언어로 작성되어 있어 이식성이 높으며 장치, 프로세스간의 호환성이 높음
- 크기가 작고 이해하기가 쉬우며, Multi-User, Multi-Tasking 지원
- 많은 네트워킹 기능을 제공하므로 통신망(Network) 관리용 운영체제로 적합함
- 트리 구조의 파일 시스템으로, 전문적인 프로그램 개발에 용이함
- 다양한 유틸리티 프로그램들이 존재함

핵심 156 | UNIX 시스템의 구성

커널(Kernel)	<ul style="list-style-type: none"> · UNIX의 가장 핵심적인 부분 · 하드웨어를 보호하고, 프로그램들과 하드웨어 간의 인터페이스 역할 담당 · 프로세스 관리, 기억장치 관리, 파일 관리, 입·출력 관리, 프로세스간 통신, 데이터 전송 및 변환 등 여러 가지 기능 수행
셸(Shell)	<ul style="list-style-type: none"> · 사용자의 명령어를 인식하여 프로그램을 호출하고, 명령을 수행하는 명령어 해석기 · 시스템과 사용자 간의 인터페이스 담당 · DOS의 COMMAND.COM과 같은 기능 수행 · 주기억장치에 상주하지 않고, 명령어가 포함된 파일 형태로 존재하며 보조기억장치에서 교체 처리 가능
유틸리티(Utility)	<ul style="list-style-type: none"> · 일반 사용자가 작성한 응용 프로그램을 처리하는 데 사용 · DOS에서의 외부 명령어에 해당됨

핵심 157 | UNIX 파일 시스템의 구조

- **부트 블록** : 부팅시 필요한 코드를 저장하고 있는 블록
- **슈퍼 블록** : 전체 파일 시스템에 대한 정보를 저장하고 있는 블록
- **Inode 블록** : 각 파일이나 디렉토리에 대한 모든 정보를 저장하고 있는 블록으로, 파일 소유자의 사용자 번호(UID) 및 그룹 번호(GID), 파일 크기, 파일 type, 생성 시기, 최종 변경 시기, 최근 사용 시기, 파일의 보호 권한, 파일 링크 수, 데이터가 저장된 블록의 시작 주소 등의 정보를 가지고 있음
- **데이터 블록** : 디렉토리별로 디렉토리 엔트리와 실제 파일에 대한 데이터가 저장된 블록

핵심 158 | UNIX의 주요 명령어

명령어	의미
fork	새로운 프로세스 생성
exec	새로운 프로세스 수행
&	백그라운드 처리를 위해 명령의 끝에 입력
mv	파일 이동, 이름 변경
rm	파일 삭제
chmod	파일의 사용 허가 지정
mount	파일 시스템을 마운팅 함
mkfs	파일 시스템 생성
chdir	현재 사용할 디렉토리 위치 변경
fsck	파일 시스템 검사·보수
rmdir	디렉토리 삭제
ls	현재 디렉토리 내의 파일 목록 확인

핵심 159 | Windows의 주요 특징

- **GUI(Graphic User Interface, 그래픽 사용자 인터페이스)** : 키보드로 명령어를 직접 입력하지 않고, 아이콘이나 메뉴를 마우스로 선택하여 모든 작업을 수행하는 방식
- **선점형 멀티태스킹(Preemptive Multi-Tasking)** : 동시에 여러 개의 프로그램을 실행하는 멀티태스킹을 하면서 운영체제가 각 작업의 CPU 이용 시간을 제어하여 응용 프로그램 실행 중 문제가 발생하면 해당 프로그램을 강제 종료시키고 모든 시스템 자원을 반환하는 방식
- **FAT 32 파일 시스템 사용**
- **PnP(Plug and Play : 자동 감지 기능) 사용** : 컴퓨터 시스템에 프린터나 사운드 카드 등의 하드웨어를 설치했을 때, 해당 하드웨어를 사용하는 데 필요한 시스템 환경을 운영체제가 자동으로 구성해 주는 기능
- **OLE(Object Linking and embedding) 사용** : 다른 여러 응용 프로그램에서 작성된 문자나 그림 등의 개체(Object)를 현재 작성 중인 문서에 자유롭게 연결(Linking)하거나 삽입(Embedding)하여 편집할 수 있게 하는 기능
- **255자의 긴 파일명 사용** : 파일 이름을 VFAT(Virtual File Allocation Table)를 이용하여 최대 255자까지 지정할 수 있음

핵심 160 | MS-DOS의 특징과 파일

- **CUI(Character User Interface, 문자 중심의 사용자 인터페이스)** : 작업을 위한 실행 명령을 문자(Character)로 직접 입력하여 실행시킴
- **Single-User** : 하나의 컴퓨터를 한 사람만이 사용
- **Single-Tasking** : 한 번에 하나의 프로그램만을 수행
- **MSDOS.SYS** : 프로세스 관리, 메모리 관리, 주변장치 관리, 파일 관리 등의 파일 입·출력 시스템 호출 담당
- **IO.SYS** : MSDOS.SYS의 입·출력 요구에 따라 실제 입·출력 처리 담당
- **COMMAND.COM** : 명령을 해독하여 실행하는 파일
- **AUTOEXEC.BAT** : 부팅시 먼저 수행될 일정하고 반복적인 명령들을 일괄적으로 모아놓은 파일
- **CONFIG.SYS** : 컴퓨터 시스템의 환경 설정을 위한 파일

핵심 161 | MS-DOS의 명령어

- **내부 명령어** : 실행 과정이 간단하고 기본적인 기능을 수행하는 것으로, 메모리에 항상 상주하는 명령어(DIR, COPY, TYPE, REN, DEL, MD, CD, RD 등)
- **외부 명령어** : 실행 과정이 복잡하거나 자주 사용하지 않는 것으로, 디스크에 파일로 저장되어 있는 명령어(UNDELETE, SYS, ATTRIB, MOVE, FIND, CHKDSK 등)

4과목 · 소프트웨어 공학

핵심 162 | 소프트웨어와 시스템

- **소프트웨어** : 하드웨어를 동작시켜 사용자가 작업을 편리하게 수행하도록 하는 프로그램과 자료 구조 등을 총칭하며, 프로그램 자체 뿐만 아니라 프로그램의 개발, 운용 및 유지 보수에 관련된 모든 문서와 정보를 포함함
- **시스템** : 공통의 목적이나 목표를 달성하기 위하여 여러 가지 상호 관련된 요소들을 유기적으로 결합한 것으로, 구성 요소는 입력, 처리, 출력, 제어, 피드백으로 나눌 수 있음

핵심 163 | 소프트웨어 위기(Crisis)

- 여러 가지 원인에 의해 소프트웨어 개발 속도가 하드웨어 개발 속도를 따라가지 못해 소프트웨어에 대한 사용자들의 요구사항을 처리할 수 없는 문제가 발생한 것을 의미함
- **소프트웨어 위기의 원인과 결과**

원인	<ul style="list-style-type: none"> • 소프트웨어의 특성에 대한 이해 부족 • 소프트웨어의 관리 부재 • 프로그래밍에만 치중
결과	<ul style="list-style-type: none"> • 개발 인력의 부족과 그로 인한 인건비 상승 • 성능 및 신뢰성의 부족 • 개발 기간의 지연 및 개발 비용의 증가 • 유지보수가 어렵고, 이에 따른 비용 증가 • 소프트웨어의 생산성 저하, 소프트웨어의 품질 저하

핵심 164 | 소프트웨어 공학의 기본 원칙

- 현대적인 프로그래밍 기술을 계속적으로 적용해야 함
- 개발된 소프트웨어의 품질이 유지되도록 지속적으로 검증해야 함
- 소프트웨어 개발 관련 사항 및 결과에 대한 명확한 기록을 유지해야 함

핵심 165 | 일반적인 소프트웨어 생명 주기

- **정의 단계** : '무엇(What)'을 처리하는 소프트웨어를 개발할 것인지 정의하는 단계로, 관리자와 사용자가 가장 많이 참여하는 단계

타당성 검토 단계	개발할 소프트웨어가 법적, 경제적, 기술적으로 실현 가능성이 있는지 조사하는 단계
개발 계획 단계	소프트웨어 개발에 사용될 자원과 비용을 측정하는 단계
요구 사항 분석 단계	사용자가 요구한 문제를 보다 상세하고 정확히 분석하는 단계

- **개발 단계** : '어떻게(How)'에 초점을 두고 실제로 소프트웨어를 개발하는 단계

설계 단계	소프트웨어의 구조, 알고리즘, 자료 구조 등을 작성하는 단계로, 에러가 가장 많이 발생하는 단계
구현 단계	설계 단계에서 작성된 문서를 기초로 하여 코딩하고 번역하는 단계
테스트 단계	구현된 소프트웨어에 내재되어 있는 오류를 찾아주는 단계

- **유지 보수 단계** : 소프트웨어를 직접 운용하며, '변경(Change)'에 초점을 두고 여러 환경 변화에 따라 소프트웨어를 적응 및 유지시키는 단계로, 시간과 비용이 가장 많이 투입되는 단계

핵심 166 | 폭포수 모형(Waterfall Model)

- 소프트웨어 개발 각 단계를 확실히 매듭짓고 그 결과를 철저하게 검토하여 승인과정을 거친 후에 다음 단계를 진행하며 이전 단계로 넘어갈 수 없는 방식
- 소프트웨어 공학에서 가장 오래되고 가장 폭넓게 사용된 전통적인 소프트웨어 생명 주기 모형(고전적 생명 주기 모형)
- 소프트웨어 개발 과정의 앞 단계가 끝나야만 다음 단계로 넘어갈 수 있는 선형 순차적 모형
- 소프트웨어의 일부가 될 매뉴얼을 작성해야 함
- **개발 순서** : 타당성 검토 → 계획 → 요구 분석 → 설계 → 구현(코딩) → 시험(검사) → 유지보수

장점	<ul style="list-style-type: none"> • 모형의 적용 경험과 성공 사례가 많음 • 단계별 정의가 분명하고, 전체 공조의 이해가 용이함 • 단계별 산출물이 정확하여 개발 공정의 기준점을 잘 제시함
단점	<ul style="list-style-type: none"> • 개발 과정 중에 발생하는 새로운 요구나 경험을 반영하기 어려우므로 처음부터 사용자들이 모든 요구사항들을 명확하게 제시해야 함 • 단계 별로 오류 없이 다음 단계로 진행해야 하는데 현실적으로 오류 없이 다음 단계로 진행하기는 어려움 • 개발된 프로그램을 업무에 운용할 때 검출되지 않은 오류로 인하여 개발된 프로그램을 업무에 운용할 때 사용자들이 큰 인내심을 가져야 함

핵심 167 | 프로토타입 모형(Prototype Model)

- 사용자의 요구 사항을 정확히 파악하기 위해 실제 개발될 소프트웨어에 대한 견본(시제품(Prototype))을 만들어 최종 결과물을 예측하는 모형
- 시스템의 일부 혹은 시스템의 모형을 만드는 과정으로서 요구된 소프트웨어를 구현하는데 이는 추후 구현 단계에서 사용될 골격 코드가 됨
- 프로토타입은 요구 분석 단계에서 사용하게 되며, 프로토타입의 평가가 끝나고 개발이 승인되면 다른 모형을 이용하여 본격적인 개발이 이루어짐
- **개발 순서** : 요구 수집 → 빠른 설계 → 프로토타입 구축 → 고객 평가 → 프로토타입 조정 → 구현

장점	<ul style="list-style-type: none"> • 요구 사항을 충실히 반영하며, 요구 사항의 변경이 용이함 • 최종 결과물이 만들어지기 전에 의뢰자가 최종 결과물의 일부 또는 모형을 볼 수 있음 • 프로토타입은 의뢰자나 개발자 모두에게 공동의 참조 모델을 제공함
단점	<ul style="list-style-type: none"> • 미리 제작된 소프트웨어를 사용할 경우 실제 소프트웨어와의 차이가 발생할 수 있어 사용자에게 혼란을 줄 수 있음 • 단기간에 제작해야 하기 때문에 비효율적인 언어나 알고리즘을 사용할 수 있음

핵심 168 | 나선형 모형(Spiral Model), 점진적 모형

- 폭포수 모형과 프로토타입 모형의 장점에 위험 분석 기능을 추가한 모형
- 나선을 따라 돌 듯이 여러 번의 소프트웨어 개발 과정을 거쳐 점진적으로(프로토타입을 지속적으로 발전시켜) 완벽한 최종 소프트웨어를 개발하는 것
- 소프트웨어를 개발하면서 발생할 수 있는 위험을 관리하고 최소화하는 것을 목적으로 함
- **개발 순서** : 계획 및 정의(Planning) → 위험 분석(Risk Analysis) → 공학적 개발(Engineering) → 고객 평가(Customer Evaluation)

장점	<ul style="list-style-type: none"> • 가장 현실적인 모형으로, 대규모 시스템에 적합함 • 점진적으로 개발 과정이 반복되므로 누락되거나 추가된 요구사항을 첨가할 수 있고, 정밀하며, 유지 보수 과정이 필요 없음
단점	<ul style="list-style-type: none"> • 위험성 평가에 크게 의존하기 때문에 이를 발견하지 않으면 반드시 문제가 발생함 • 비교적 최신기법이므로 폭포수 모형이나 프로토타입 모형보다 널리 사용되지 않음

핵심 169 | 프로젝트 관리 대상

- **계획 관리** : 프로젝트 계획, 비용 산정, 일정 계획, 조직 계획
- **품질 관리** : 품질 통제, 품질 보증
- **위험 관리** : 위험 식별, 위험 분석 및 평가, 위험 관리 계획, 위험 감시 및 조치

핵심 170 | 소프트웨어 개발 영역 결정

- 프로젝트 계획 수립의 첫 번째 업무로, 개발될 소프트웨어의 영역을 결정하는 것
- **소프트웨어 개발 영역을 결정하는 주요 요소** : 처리될 데이터와 소프트웨어에 대한 기능, 성능, 제약 조건, 인터페이스 및 신뢰도 등

핵심 171 | LOC(원시 코드 라인 수) 기법

- 소프트웨어 각 기능의 원시 코드 라인 수의 비관치, 낙관치, 기대치를 측정하여 예측치를 구하고 이를 이용하여 비용을 산정하는 기법
- 측정이 용이하고, 이해가 쉬워 가장 많이 사용됨
- 예측치를 이용하여 생산성, 노력, 개발 기간 등의 비용 산정

▪ 예측치 = $\frac{a+4m+b}{6}$ (단, a : 낙관치, b : 비관치, m : 중간치(기대치))

▪ 산정 공식

노력(인월)	개발기간 × 투입 인원, LOC / 1인당 월평균 생산 코드 라인수
개발 비용	노력(인월) × 단위 비용(1인당 월평균 인건비)
개발 기간	노력(인월) / 투입 인원
생산성	LOC / 노력(인월)

핵심 172 | COCOMO 모형

- Boehm이 제안한 것으로 원시 프로그램의 규모(LOC)에 의한 비용 산정 기법
- 개발할 소프트웨어의 규모를 예측한 후 이를 소프트웨어 종류에 따라 다르게 책정되는 비용 신장 방정식(공식)에 대입하여 비용 산정
- 비용 견적의 강도 분석 및 비용 견적의 유연성이 높아 소프트웨어 개발비 견적에 널리 통용되고 있음
- **소프트웨어 개발 유형**

조직형(Organic Mode)	<ul style="list-style-type: none"> 기관 내부에서 개발된 중·소규모의 소프트웨어로 일괄 자료 처리나 과학 기술 계산용, 비즈니스 자료 처리용으로 5만(50KDSI) 라인 이하의 소프트웨어를 개발하는 유형 사무 처리용, 업무용, 과학용 응용 소프트웨어 개발에 적합함
반분리형(Semi-Detached Mode)	<ul style="list-style-type: none"> 조직형과 내장형의 중간형으로 트랜잭션 처리 시스템이나 운영체제, 데이터베이스 관리 시스템 등의 30만(300KDSI) 라인 이하의 소프트웨어를 개발하는 유형 컴파일러, 인터프리터와 같은 유틸리티 개발에 적합함
내장형(Embedded Mode)	<ul style="list-style-type: none"> 최대형 규모의 트랜잭션 처리 시스템이나 운영체제 등의 30만(300KDSI) 라인 이상의 소프트웨어를 개발하는 유형 신호기 제어 시스템, 미사일 유도 시스템, 실시간 처리 시스템 등의 시스템 프로그램 개발에 적합함

핵심 173 | COCOMO 모형의 종류

- 비용 산정 단계 및 적용 변수의 구체화 정도에 따라 기본(Basic), 중간(Intermediate), 발전(Detailed)형으로 구분할 수 있음
- 기본(Basic)형 COCOMO** : 소프트웨어의 크기와 개발 유형만을 이용하여 비용을 산정하는 모형
- 중간(Intermediate)형 COCOMO** : 기본 COCOMO 공식을 토대로 사용하나, 제품의 특성, 컴퓨터의 특성, 개발 요원의 특성, 프로젝트 특성에 의해 비용을 산정하는 모형
- 발전(Detailed)형 COCOMO** : 중간(Intermediate)형 COCOMO를 보완하여 만들어진 방법으로, 개발 공정별로 보다 자세하고 정확하게 노력을 산출하여 비용을 산정하는 모형

핵심 174 | 사람-노력 관계와 노력 분배

사람-노력 관계	<ul style="list-style-type: none"> 소규모의 개발 프로젝트에서는 한 사람이 요구사항을 분석하고, 설계, 코딩, 테스트까지 수행할 수 있음 프로젝트의 크기가 증가할수록 더 많은 사람들이 참여해야 함 프로젝트 진행 중에 새로운 인력을 투입할 경우 작업 적응 기간과 부작용으로 인해 일정을 더욱 지연시키고, 프로젝트에 혼란을 가져오게 됨(Brooks의 법칙)
노력 분배	<ul style="list-style-type: none"> 예측된 노력을 각 개발 과정에 분배할 때는 40-20-40 규칙을 권장하며, 이는 분석과 설계에 40%, 코딩에 20%, 테스트에 40%를 분배한다는 의미임 일반적으로 노력은 요구 분석이 10~25%, 설계가 20~25%, 코딩이 15~25%, 테스트와 디버깅이 30~40%를 차지함

핵심 175 | WBS(업무 분류 구조)

- 개발 프로젝트를 여러 개의 작은 관리 단위(소작업)로 분할하여 계층적으로 기술한 업무 구조
- 일정 계획의 첫 단계에서 작업을 분해할 때 사용되는 방법
- 계획 관리 단계에 있어서 일정 계획과 인력 계획, 비용 산정의 기준이 됨
- 프로젝트 진행 중에 발생하는 모든 작업을 알 수 있음

핵심 176 | PERT/CPM

- 프로젝트의 지연을 방지하고 계획대로 진행되게 하기 위한 일정을 계획하는 것으로, 대단위 계획의 조직적인 추진을 위해 자원의 제약 하에 비용을 적게 사용하면서 초단시간내 계획 완성을 위한 프로젝트 일정 방법

- 프로젝트 개발 기간을 결정하는 임계 경로(OP ; Critical Path) 제공
- 통계적 모델을 적용해서 개별 작업에 대한 가장 근접한 시간 측정의 기준이 됨
- 각 작업에 대한 시작 시간을 정의하여 작업들간의 경계 시간을 계산할 수 있게 함

PERT	<ul style="list-style-type: none"> 프로젝트에 필요한 전체 작업의 상호 관계를 표시하는 네트워크로 각 작업별로 낙관적인 경우, 가능성이 있는 경우, 비관적인 경우로 나누어 각 단계별 종료시기를 결정하는 방법 노드와 간선으로 구성되었으며 원 노드에는 작업을, 간선에는 낙관치, 기대치, 비관치를 표시함
CPM	<ul style="list-style-type: none"> 프로젝트 완성에 필요한 작업을 나열하고 작업에 필요한 소요 기간을 예측하는데 사용하는 기법 CPM은 노드와 간선으로 구성된 네트워크로 노드는 작업을, 간선은 작업 사이의 전후 의존 관계를 나타냄 원형 노드는 각 작업을 의미하며 각 작업 이름과 소요 기간을 표시하고, 박스 노드는 이정표를 의미하며 박스 노드 위에는 예상 완료 시간을 표시함

핵심 177 | 간트 차트(Gantt Chart)

- 프로젝트의 각 작업들이 언제 시작하고 종료되는지에 대한 작업 일정을 막대 도표를 이용하여 표시하는 프로젝트 일정표
- 중간 목표 미달성시 이유와 기간을 예측할 수 있게 함
- 사용자와의 문제점이나 예산의 초과 지출 등도 관리할 수 있게 함
- 자원 배치와 인원 계획에 유용하게 사용됨
- 다양한 형태로 변경하여 사용할 수 있음
- 작업 경로는 표시할 수 없으며, 계획의 변화에 대한 적응성이 약함
- 이정표, 작업 일정, 작업 기간, 산출물로 구성되어 있음

핵심 178 | 프로젝트 팀 구성

분산형 팀	<ul style="list-style-type: none"> 팀원 모두가 의사 결정에 참여하는 비이기적인 구성 방식(민주주의식 팀) 의사 결정을 민주주의식으로 하며 팀 구성원의 참여도와 작업 만족도를 높이고 이직률을 낮게 함 팀 구성원 각자가 서로의 일을 검토하고 다른 구성원이 일한 결과에 대하여 같은 그룹의 일원으로 책임을 지며, 장기 프로젝트 개발에 적합함
중앙집중형 팀	<ul style="list-style-type: none"> 의사 소통 경로의 수 = $\frac{n(n-1)}{2}$ (n : 팀원 수)
중요한 작업 담당 팀	<ul style="list-style-type: none"> 한 관리자가 의사 결정을 하고, 팀 구성원들은 그 결정을 따르는 구성 방식(책임 프로그래머 팀) 프로젝트 수행에 따른 모든 권한과 책임을 한 관리자에게 위임하고, 기술 및 관리 지원을 위해 인력을 투입하는 형태 의사 결정이 빠르고, 의사 교환 경로를 줄일 수 있음 책임 프로그래머 역할 : 요구 분석 및 설계, 중요한 기술적 판단 프로그래머에게 작업 지시 및 배분 등 프로그래머 역할 : 책임 프로그래머의 지시에 따른 원시 코드 작성, 테스트, 디버깅, 문서 작성 등 프로그래밍 사서 역할 : 프로그램 리스트, 설계 문서, 테스트 계획 등을 관리 보조 프로그래머 역할 : 책임 프로그래머의 업무 지원, 여러 가지 기술적인 문제에 대한 자문, 사용자, 품질 보증 담당자 등의 섭외, 책임 프로그래머 감독 하에 분석, 설계, 구현 담당

계 층 적 팀	<ul style="list-style-type: none"> · 분산형 팀 구성과 중앙 집중형 팀 구성을 혼합한 형태(혼합형 팀) · 5~7명의 초보 프로그래머를 작은 그룹으로 만들어 각 그룹을 고 급 프로그래머가 관리하게 함 · 경험자(고급 프로그래머)와 초보자를 구별함 · 프로젝트 리더와 고급 프로그래머에게 지휘 권한을 부여하고, 의사 교환은 초급 프로그래머와 고급 프로그래머로 분산함
------------------	---

핵심 179 | 품질 표준

- 명확하게 정의된 소프트웨어의 특성을 의미하며, 소프트웨어의 품질을 평가하는 기준 항목

·종류

종류	의미
정확성(Correctness)	사용자의 요구 기능을 충족시키는 정도
신뢰성(Reliability)	정확하고 일관된 결과를 얻기 위해 요구된 기능을 오류없이 수행하는 정도
효율성(Efficiency)	요구되는 기능을 수행하기 위해 필요한 자원의 소요 정도
무결성(Integrity)	허용되지 않는 사용이나 자료의 변경을 제어하는 정도
사용 용이성(Usability)	사용에 필요한 노력을 최소화하고 쉽게 사용할 수 있는 정도
유지보수성(Maintainability)	변경 및 오류 사항의 교정에 대한 노력을 최소화하는 정도
유연성(Flexibility)	소프트웨어를 얼마만큼 쉽게 수정할 수 있는가 하는 정도
시험 역량(Testability)	의도된 기능을 수행하도록 보장하기 위해 프로그램을 시험할 수 있는 정도
이식성(Portability)	다양한 하드웨어 환경에서도 운용 가능하도록 쉽게 수정될 수 있는 정도
재사용성(Reusability)	전체나 일부 소프트웨어를 다른 목적으로 사용할 수 있는가 하는 정도
상호운용성(Interoperability)	다른 소프트웨어와 정보를 교환할 수 있는 정도

핵심 180 | 품질 보증

- 어떠한 소프트웨어가 이미 설정된 요구사항과 일치하는가를 확인하는데 필요한 개발 단계 전체에 걸친 계획적이고 체계적인 작업
- 소프트웨어 개발 초기에 소프트웨어의 특성과 요구사항을 철저히 파악하여 품질 목표를 설정하고, 개발 단계에서는 정형 기술 검토를 통하여 품질 목표의 충족 여부를 점검하며, 개발 후에는 디버깅과 시험 과정을 거침

핵심 181 | 정형 기술 검토/검토회의/검열

정형 기술 검토(FTR)	<ul style="list-style-type: none"> · 가장 일반적인 검토 방법으로 소프트웨어 기술자들에 의해 수행되는 소프트웨어 품질 보증 활동 · 정형 기술 검토 유형에는 검토 회의(Walkthrough), 검열(Inspections) 등이 있으며 이는 모두 회의 형태로 수행됨
검토 회의(Walkthrough)	<ul style="list-style-type: none"> · 소프트웨어 개발의 각 단계에서 개최하는 기술 평가 회의로, 소프트웨어 구성 요소와 같은 작은 단위를 검토하는 것 · 오류의 조기 검출을 목적으로 하며 발견된 오류는 문서화 하고, 검토 회의 후에 해결함
검열(Inspections, 심사)	검토 회의를 발전시킨 형태로, 소프트웨어 개발 단계에서 산출된 결과물의 품질을 평가하며 이를 개선시키는데 사용

핵심 182 | 소프트웨어의 신뢰성과 가용성

- **신뢰성** : 프로그램이 주어진 환경에서 주어진 시간 동안 오류없이 작동할 확률
- **가용성** : 한 프로그램이 주어진 시점에서 요구사항에 따라 운영되는 확률
- 간단한 신뢰성 측정은 MTBF를 이용함
- **MTBF(Mean Time Between Failure)** : 평균 고장 간격으로, 수리가 가능한 시스템이 고장난 후부터 다음 고장이 날 때까지의 평균 시간. $MTBF = MTTF + MTTR$
- **MTTF(Mean Time To Failure)** : 평균 가동 시간으로, 수리 불가능한 시스템의 사용 시점부터 고장이 발생할 때까지의 가동 시간 평균, 고장 평균 시간이라고도 함
- **MTTR(Mean Time To Repair)** : 평균 수리 시간으로, 시스템에 고장이 발생하여 가동하지 못한 시간들의 평균
- **가용성 측정** : 시스템의 총 운용 시간 중 정상적으로 가동된 시간의 비율

$$\text{가용성} = \frac{MTBF}{MTBF + MTTR} \times 100\% = \frac{MTTF}{MTTF + MTTR} \times 100\% = \frac{MTTF}{MTBF} \times 100\%$$

핵심 183 | 위험 관리

- 프로젝트 추진 과정에서 예상되는 각종 돌발 상황(위험)을 미리 예상하고 이에 대한 적절한 대책을 수립하는 일련의 활동
- 위험은 불확실성과 손실을 내재하고 있는데 위험 관리란 이러한 위험의 불확실성을 감소시키고, 손실에 대비하는 작업임

· 위험 관리의 절차

위험 식별	알려지거나 예측 가능한 위험 요소를 파악하는 작업
위험 분석 및 평가	<ul style="list-style-type: none"> · 위험이 현실화될 가능성과 실제 발생하였을 때의 문제들을 분석하고 영향력을 파악하는 위험 추산(Risk Estimation) 작업을 통해 수행됨 · 위험 추산을 위해 위험표(Risk Table)를 작성하여 활용함 · 위험표의 구성 요소 : 위험 내용, 위험 범주, 발생 확률, 영향력, 위험 감시 및 조치
위험 관리 계획	위험을 예방하고 위험 발생시 대책을 준비하며 문서화 하는 작업
위험 감시 및 조치	<ul style="list-style-type: none"> · 위험 회피(Risk Avoidance) : 위험 관리에 대한 최상의 전략으로 위험이 발생할 것을 예상하고 회피하는 것 · 위험 감시(Risk Monitoring) : 위험 요소 징후들에 대하여 계속적으로 인지하는 것 · 위험 관리(Risk Management) 및 비상 계획(Contingency Plan) 수립 : 위험 회피 전략이 실패할 경우 위험에 대해 관리하고, 대비책과 비상 계획 수립

핵심 184 | 형상 관리(SCM)

- 소프트웨어의 개발 과정에서 소프트웨어의 변경 사항을 관리하기 위해 개발된 일련의 활동
- 소프트웨어 변경의 원인을 알아내고 제어하며 적절히 변경되고 있는지 확인하여 해당 담당자에게 통보하는 작업
- 소프트웨어 개발의 전 단계에 적용되는 활동으로, 유지보수 단계에서 수행됨
- 소프트웨어 개발의 전체 비용을 줄이고, 개발 과정의 여러 문제점을 해결하여 방해 요인을 최소화하는 것을 목적으로 함

- **소프트웨어 형상 항목** : 정의 단계의 문서, 개발 단계의 문서와 프로그램, 유지보수 단계의 변경 사항 등

핵심 185 | 구조적 분석 기법

- 도형 중심의 분석용 도구와 분석 절차를 이용하여 사용자의 요구사항을 파악하고 문서화함
- 도형 중심의 도구를 사용하므로 분석가와 사용자 간의 대화가 용이함
- 하향식 방법을 사용하여 시스템을 세분화할 수 있고, 분석의 중복을 배제할 수 있음
- 사용자의 요구사항을 논리적으로 표현하여 전체 시스템을 일관성 있게 이해할 수 있음
- 시스템 분석의 질이 향상되고, 시스템 개발의 모든 단계에서 필요한 명세서 작성이 가능함
- 구조적 분석 기법에서는 자료사전(DD), 개체 관계도(ERD), 자료 흐름도(DFD) 등의 도구를 이용함

핵심 186 | 자료 흐름도(DFD)

- 요구사항 분석에서 자료의 흐름 및 변환 과정과 기능을 도형 중심으로 기술하는 방법
- 시스템 안의 프로세스, 자료 저장소 사이에 자료의 흐름을 나타내는 그래프로 자료 흐름과 기능을 모델화하는데 적합함
- 자료 흐름도는 자료 흐름과 기능을 자세히 표현하기 위해 단계적으로 세분화됨

▪ 자료 흐름도 구성 요소의 일반적 표기법

기호	의미	표기법
프로세스 (Process)	자료를 변환시키는 시스템의 한 부분(처리 과정)을 나타내며, 처리, 기능, 변환, 버블이라고도 함	물품 확인
자료 흐름(Flow)	자료의 이동을 나타냄	→
자료 저장소(Data Store)	시스템에서의 자료 저장소(파일, 데이터베이스)를 나타냄	_____
단말(Terminator)	시스템과 교신하는 외부 개체로, 입력 데이터가 만들어지고, 출력 데이터를 받음(정보의 생산자와 소비자)	□

핵심 187 | 자료 사전(DD)

- 자료 흐름도상에 있는 자료를 더 자세히 정의하고 기록한 것이며, 이처럼 데이터를 설명하는 데이터를 데이터의 데이터 또는 메타 데이터(Meta Data)라고 함
- 자료 흐름도에 시각적으로 표시된 자료에 대한 정보를 체계적이고 조직적으로 모아 개발자나 사용자가 편리하게 사용할 수 있음

▪ 자료 사전 표기 기호

기호	의미
=	자료의 정의 : ~로 구성되어 있다(is composed of)
+	자료의 연결 : 그리고(and)
()	자료의 생략 : 생략 가능한 자료(Optional)
[]	자료의 선택 : 또는(or)
{ }	자료의 반복
* *	자료의 설명 : 주석(Comment)

핵심 188 | 소단위 명세서/개체 관계도

- **소단위 명세서(Mini-Specification)** : 세분화된 자료 흐름도에서 최하위 단계 버블(프로세스)의 처리 절차를 기술하는 것으로 프로세스 명세서라고도 함
- **개체 관계도(ERD)** : 시스템에서 처리되는 개체(자료)와 개체의 구성과 속성, 개체간의 관계를 표현하여 자료를 모델화하는데 사용됨

핵심 189 | HIPO

- 시스템의 분석 및 설계나 문서화할 때 사용되는 기법으로 시스템 실행 과정인 입력, 처리, 출력의 기능을 나타냄
- 기본 시스템 모델은 입력, 처리, 출력으로 구성되며, 하향식 소프트웨어 개발을 위한 문서화 도구
- 체계적인 문서 관리가 가능하고, 기호, 도표 등을 사용하므로 보기 쉬우며 이해하기도 쉬움
- 기능과 자료의 의존 관계를 동시에 표현할 수 있음

▪ HIPO의 종류

가시적 도표 (도식 목차)	시스템의 전체적인 기능과 흐름을 보여주는 계층(Tree) 구조도
총체적 도표 (개요 도표, 총괄 도표)	프로그램을 구성하는 기능을 기술한 것으로 입력, 처리, 출력에 대한 전반적인 정보를 제공하는 도표
세부적 도표 (상세 도표)	총체적 도표에 표시된 기능을 구성하는 기본 요소들을 상세히 기술하는 도표

핵심 190 | 구조적 설계 기법/설계

- **구조적 설계 기법** : 구조적 분석 기법의 결과물인 자료 흐름도 등으로부터 소프트웨어의 기능(자료 구조)과 프로그램 구조, 모듈을 설계하기 위한 전략, 평가 지침 및 문서화 도구를 제공하는 체계화된 기법
- **설계** : 요구사항 분석 단계의 산출물인 요구사항 분석 명세서의 기능이 실현되도록 알고리즘과 그 알고리즘에 의해 처리될 자료 구조를 문서화하는 것으로, 설계 모형은 데이터 설계, 구조 설계, 인터페이스 설계, 절차 설계로 구성됨

핵심 191 | 구조적 설계의 주요 기본 원리

모듈화(Modularity)	소프트웨어를 모듈 단위로 나누는 것
추상화 (Abstraction)	<ul style="list-style-type: none"> • 문제의 세부 사항을 먼저 설계하기보다는 전체적이고 포괄적인 개념을 설계한 후 차례로 세분화하여 구체화시켜 나가는 설계 방법 • 기능 추상화 : 입력 자료를 출력 자료로 변환하는 과정을 추상화하는 방법 • 제어 추상화 : 제어의 정확한 매커니즘(절차, 방법)을 정의하지 않고 원하는 효과를 정하는데 이용하는 방법 • 자료 추상화 : 자료와 자료에 적용될 수 있는 기능을 함께 정의함으로써 자료 객체를 구성하는 방법
정보 은닉(Information Hiding)	<ul style="list-style-type: none"> • 한 모듈 내부에 포함된 절차와 자료들의 정보가 감추어져 다른 모듈이 접근하거나 변경하지 못하도록 하는 기법 • 모듈을 독립적으로 수행할 수 있고, 하나의 모듈이 변경되더라도 다른 모듈에 영향을 주지 않으므로 수정, 시험, 유지보수가 용이함

프로그램 구조	<ul style="list-style-type: none"> 소프트웨어의 구성 요소인 모듈의 계층적 구성을 나타내는 것(제어 계층 구조) 프로그램의 순서, 선택, 반복과 같은 소프트웨어의 절차적인 처리 과정을 나타내지 않음
자료 구조	자료 사이의 논리적인 관계를 표현한 것으로 자료의 구성, 결합 정도, 접근 방법 등을 나타냄

공동 결합도(Common Coupling)	<ul style="list-style-type: none"> 공유되는 공통 데이터 영역을 여러 모듈이 사용할 때의 결합도 공통 데이터 영역의 내용을 조금만 변경하여도 이를 사용하는 모든 모듈에 영향을 미치므로 모듈의 독립성을 약하게 만듦
내용 결합도(Content Coupling)	한 모듈이 다른 모듈의 내부 기능 및 그 내부 자료를 직접 참조하거나 수정할 때의 결합도

핵심 192 | 프로그램 구조에서 사용되는 용어

- **공유도(Fan-in, 팬-입력)** : 어떤 모듈을 제어(호출)하는 모듈의 수
- **제어도(Fan-out, 팬-출력)** : 어떤 모듈에 의해 제어(호출)되는 모듈의 수
- **깊이(Depth)** : 제어의 계층 수
- **넓이(Width)** : 제어의 분기된 수
- **주종적 모듈(Superordinate)** : 다른 모듈을 제어(호출)하는 모듈
- **종속적 모듈(Subordinate)** : 어떤 모듈에 의해 제어되는 모듈

핵심 193 | 바람직한 설계의 특징

- 설계는 소프트웨어 구조를 나타내야 함
- 독립적인 기능적 특성을 가진 요소(모듈)로 구성되어야 함
- 모듈 구조, 즉 특정 기능 또는 부기능을 수행하는 논리적 요소들로 분리되는 구조를 가져야 함
- 소프트웨어 요소(모듈) 간의 효과적인 제어를 위해 설계에서 계층적 자료 조직이 제시되어야 함
- 자료와 프로시저에 대한 분명하고 분리된 표현을 포함해야 함
- 모듈 간과 외부 개체 간의 연결 복잡성을 줄이는 인터페이스를 가져야 함
- 요구사항 분석에서 얻어진 정보를 이용하여 반복적인 방법으로 이루어져야 함

핵심 194 | 결합도(Coupling)

- 모듈 간에 상호 의존하는 정도
- 독립적인 모듈이 되기 위해서는 각 모듈 간의 결합도가 약해야 하며 의존하는 모듈이 적어야 함
- **결합도의 종류**(자료 결합도 < 스탬프 결합도 < 제어 결합도 < 외부 결합도 < 공동 결합도 < 내용 결합도)

자료 결합도(Data Coupling)	<ul style="list-style-type: none"> 모듈간의 인터페이스가 자료 요소로만 구성될 때의 결합도 어떤 모듈이 다른 모듈을 호출하면서 매개 변수, 인수로 데이터를 넘겨주고, 호출받은 모듈은 받은 데이터에 대한 처리 결과를 다시 돌려주는 것
스탬프(검인) 결합도(Stamp Coupling)	<ul style="list-style-type: none"> 모듈 간의 인터페이스로 배열이나 레코드 등의 자료 구조가 전달될 때의 결합도 두 모듈이 동일한 자료 구조를 조회하는 경우의 결합도이며 자료 구조의 어떠한 변화, 즉 포맷이나 구조의 변화는 그것을 조회하는 모든 모듈 및 변화되는 필드를 실제로 조회하지 않는 모듈에까지도 영향을 미치게 됨
제어 결합도(Control Coupling)	<ul style="list-style-type: none"> 한 모듈에서 다른 모듈로 논리적인 흐름을 제어하는데 사용하는 제어 요소(Function Code, Switch, Tag, Flag)가 전달될 때의 결합도 상위 모듈이 하위 모듈의 상세한 처리 절차를 알고 있어 이를 통제할 경우나 처리 기능이 두 모듈에 분리되어 설계된 경우에 발생함
외부 결합도(External Coupling)	<ul style="list-style-type: none"> 어떤 모듈에서 외부로 선언한 데이터(변수)를 다른 모듈에서 참조할 때의 결합도 참조되는 데이터의 범위를 각 모듈에서 제한할 수 있음

핵심 195 | 응집도(Cohesion)

- 정보 은닉 개념을 확장한 것으로 모듈 안의 요소들이 서로 관련되어 있는 정도, 즉 모듈이 독립적인 기능으로 정의되어 있는 정도
- 독립적인 모듈이 되기 위해서는 각 모듈의 응집도가 강해야 함
- **응집도 종류**(우연적 응집도 < 논리적 응집도 < 시간적 응집도 < 절차적 응집도 < 교환적 응집도 < 순차적 응집도 < 기능적 응집도)

기능적(Functional) 응집도	모듈 내부의 모든 기능 요소들이 단일 문제와 연관되어 수행될 경우의 응집도
순차적(Sequential) 응집도	모듈 내의 하나의 활동으로부터 나온 출력 데이터를 그 다음 활동의 입력 데이터로 사용할 경우의 응집도
교환(통신)적(Communication) 응집도	동일한 입력과 출력을 사용하여 서로 다른 기능을 수행하는 구성 요소들이 모였을 경우의 응집도
절차적(Procedural) 응집도	모듈이 다수의 관련 기능을 가질 때 모듈 안의 구성 요소들이 그 기능을 순차적으로 수행할 경우의 응집도
시간적(Temporal) 응집도	특정 시간에 처리되는 몇 개의 기능을 모아 하나의 모듈로 작성할 경우의 응집도
논리적(Logical) 응집도	유사한 성격을 갖거나 특정 형태로 분류되는 처리요소들로 하나의 모듈이 형성되는 경우의 응집도
우연적(Coincidental) 응집도	모듈 내부의 각 구성 요소들이 서로 관련 없는 요소로만 구성된 경우의 응집도

핵심 196 | N-S 차트(Nassi-Schneiderman Chart)

- 논리의 기술에 중점을 둔 도형을 이용한 표현 방법(박스 다이어그램, Chapin Chart)
- 순차(연속, Sequence) 구조, 반복(While) 구조, 반복(Do ~ Unit) 구조, 선택(If~Then~Else) 구조, 다중 선택(Case) 구조 등을 표현
- GOTO나 화살표를 사용하지 않으며, 선택과 반복 구조를 시각적으로 표현
- 조건이 복합되어 있는 곳의 처리를 시각적으로 명확히 식별하는데 적합함
- 이해하기 쉽고, 코드 변환이 용이함
- 읽기는 쉽지만 작성하기가 어려우며, 임의로 제어를 전이하는 것이 불가능함

핵심 197 | 구현/프로그래밍 언어 선정 기준

- **구현** : 설계 단계에서 생성된 설계 명세서를 컴퓨터가 알 수 있는 모습으로 변환하는 과정으로 프로그래밍 또는 코딩 단계라고도 하며, 각 모듈을 특정 프로그래밍 언어를 이용하여 원시 코드로 작성하고 문서화하는 작업임
- **프로그래밍 언어 선정 기준** : 친밀감, 프로그램 구조, 언어의 능력, 프로그램의 길이, 처리의 효율성, 이식성, 과거의 개발실적, 대사업무의 성격, 알고리즘과 계산상의 난이도, 소프트웨어의 수행 환경, 자료 구조의 난이도, 개발담당자의 경험과 지식 등

핵심 198 | 구조적 프로그래밍

- Dijkstra에 의해 제안된 것으로, 신뢰성 있는 소프트웨어의 생산과 코딩의 표준화 등을 위해 개발된 방법
- 구조적 프로그래밍의 기본적인 제어 구조

순차(Sequence)	명령을 순서적으로 나열
선택(Selection)	특정 논리에 기초하여 명령 선택
반복(Iteration)	순환 제공

핵심 199 | 화이트 박스 테스트

- 모듈의 원시 코드를 오픈시킨 상태에서 원시 코드의 논리적인 모든 경로를 검사하여 검사 사례를 설계하는 방법
- 설계된 절차에 초점을 둔 구조적 테스트로, 프로시저 설계의 제어 구조를 사용하여 검사 사례를 설계하며, 테스트 과정의 초기에 적용됨
- 모듈 안의 작동을 직접 관찰할 수 있음
- 원시 코드의 모든 문장을 한 번 이상 수행함으로써 수행됨
- 프로그램의 제어 구조에 따라 선택, 반복 등의 부분들을 수행함으로써 논리적 경로를 제어함
- 각 조건에서의 참과 거짓의 모든 논리적 결정이 적어도 한 번 이상 실행됨
- 종류 : 기초 경로 검사(Basic Path Testing), 제어 구조 검사(조건 검사, 루프 검사, 데이터 흐름 검사) 등

핵심 200 | 제어 흐름도/순환 복잡도

제어 흐름도	제어 흐름을 표현하기 위해 사용되는 그래프
순환 복잡도	<ul style="list-style-type: none"> · 한 프로그램의 논리적인 복잡도를 측정하기 위한 소프트웨어의 척도로, 제어 흐름도 이론에 기초를 둠 · 순환 복잡도를 이용하여 계산된 값은 프로그램의 독립적인 경로의 수를 정의하고, 모든 경로가 한 번 이상 수행되었음을 보장하기 위해 행해지는 테스트 횟수의 상한선을 제공함 · 순환 복잡도 계산 : 영역 수를 계산하거나 $V(G) = E - N + 2$ 로 계산(E는 화살표 수, N은 노드 수)

핵심 201 | 블랙 박스 테스트

- 소프트웨어가 수행할 특정 기능을 알기 위해서 각 기능이 완전히 작동되는 것을 입증하는 검사로, 기능 검사라고도 함
- 부정확하거나 누락된 기능, 인터페이스 오류, 자료 구조나 외부 데이터베이스 접근에 따른 오류, 행위나 성능 오류, 초기화와 종료 오류 등을 발견하기 위해 사용되며 테스트 과정을 후반부에 적용됨
- 소프트웨어 산물의 각 기능별로 적절한 정보 영역(입출력)을 정하여 적합한 입력에 대한 출력의 정확성을 점검함
- 종류 : 동치 분할(Equivalence Partitioning) 검사, 경계값 분석(Boundary Value Analysis), 원인-효과 그래프(Cause-Effect Graphing) 검사, 오류 예측 검사(Fault Based), 비교(Comparison) 검사 등

핵심 202 | 검사 전략 순서

- ① 단위(코드) 검사 : 검사는 프로그램의 기본 단위인 모듈(코드) 수준에서 시작함

- ② 통합(설계) 검사 : 단위 검사 후 모듈을 결합하여 전체 시스템에 대해 검사함
- ③ 검증(요구사항) 검사 : 사용자의 요구 사항을 충족시키는지를 검사함
- ④ 시스템 검사 : 개발된 소프트웨어가 컴퓨터 시스템에서 완벽하게 수행되는지를 검사함

핵심 203 | 단위 검사

- 코딩이 이루어진 후 소프트웨어 설계의 최소 단위인 모듈에 초점을 맞추어 검사하는 것
- 화이트 박스 검사 기법을 사용하며, 인터페이스, 외부적 I/O, 자료 구조, 경계 조건 등을 검사함

핵심 204 | 하향식 통합 검사

- 프로그램의 상위 모듈에서 하위 모듈 방향으로 통합하면서 검사하는 기법
- 주요 모듈(주요 프로그램)을 기준으로 하여 아래 단계로 이동하면서 통합되는데, 이때 깊이 우선 통합법이나 넓이 우선 통합법을 사용할 수 있음
- 하향식 통합 검사 절차
 - ① 주요 제어 모듈을 드라이버로 사용하고, 주요 제어 모듈의 종속 모듈들을 스텐드(Stub)로 대체
 - ② 깊이 우선 또는 넓이 우선 등의 통합 방식에 따라 종속 스텐드들이 실제 모듈로 교체됨
 - ③ 모듈이 통합될 때마다 검사 실시
 - ④ 회귀 검사 실시

핵심 205 | 상향식 통합 검사

- 프로그램의 하위 모듈에서 상위 모듈 방향으로 통합하면서 검사하는 기법
- 가장 하위 단계의 모듈부터 통합 및 검사가 수행되므로 스텐드(Stub)는 필요하지 않지만 하나의 주요 제어 모듈과 관련된 종속 모듈의 그룹인 클러스터가 필요함
- 상향식 통합 검사 절차
 - ① 하위 모듈들을 클러스터(Cluster)로 결합
 - ② 검사 사례 입출력을 조정하기 위해 드라이버(Driver)를 작성
 - ③ 클러스터 검사
 - ④ 드라이버를 제거하고, 클러스터는 프로그램 구조의 상위로 이동하여 결합

핵심 206 | 검증(확인, 인수) 검사

- 소프트웨어가 사용자의 요구사항을 충족시키는데에 중점을 두고 검사하는 방법
- 통합 검사가 끝난 후 전체가 하나의 소프트웨어 단위로 통합되어 요구사항 명세서를 토대로 진행하며, 블랙박스 테스트 기법을 사용함
- 검증 검사의 종류

형상 검사 (구성 검토 검사)	소프트웨어 구성 요소, 목록, 유지보수를 지원하기 위해 필요한 모든 사항들이 제대로 표현되었는지를 검사하는 기법
알파 검사	<ul style="list-style-type: none"> · 개발자의 장소에서 사용자가 개발자 앞에서 행해지는 검사 기법 · 통제된 환경에서 행해지며, 오류와 사용상의 문제점을 사용자와 개발자가 함께 확인하면서 기록함

베타 검사	<ul style="list-style-type: none"> · 선정된 최종 사용자가 여러 명의 사용자 앞에서 행해지는 검사 기법 · 실업무를 가지고 사용자가 직접 시험하는 것으로, 개발자에 의해 제어되지 않은 상태에서 검사가 행해지며, 발견된 오류와 사용상의 문제점을 기록하고 개발자에게 주기적으로 보고함
-------	--

핵심 207 | 시스템 검사/디버깅

- **시스템 검사** : 개발된 소프트웨어가 해당 컴퓨터 시스템에서 완벽하게 수행하는가를 검사하는 것으로 종류에는 복구 검사, 보안 검사, 강도 검사, 성능 검사 등이 있음
- **디버깅** : 검사 단계에서 검사 사례에 의해 오류를 찾은 후 그 오류를 수정하는 과정으로, 디버깅 접근법에는 맹목적 강요, 역추적, 원인 제거 등이 있음

핵심 208 | 유지보수

- 개발된 소프트웨어의 품질을 항상 최상의 상태로 유지하기 위한 것으로 소프트웨어 개발 단계 중 가장 많은 노력과 비용이 투입되는 단계
- 소프트웨어가 사용자에게 인수되고, 설치된 후 발생하는 모든 공학 적 작업임
- 소프트웨어 유지보수를 용이하게 하려면 시험용이성, 이해성, 수정 용이성, 이식성 등이 고려되어야 함
- **유지보수의 유형**

수정(Corrective) 보수 =수리 · 교정 · 정정 · 하자 보수	시스템을 운영하면서 검사 단계에서 발견하지 못한 오류를 찾아 수정하는 활동
적응(Adaptive) 보수 =환경 적응, 조정 보수	소프트웨어의 수명 기간 중에 발생하는 환경의 변화(하드웨어, 운영체제 등)를 기존의 소프트웨어 산물에 반영하기 위하여 수행하는 활동
완전화(Perfective) 보수 = 기능 개선, 기능 보수	<ul style="list-style-type: none"> · 소프트웨어의 본래 기능에 새로운 기능을 추가하거나 성능을 개선하기 위해 소프트웨어를 확장시키는 활동 · 유지보수 활동 중 가장 큰 업무 및 비용을 차지하는 활동
예방(Preventive) 보수	미래에 유지보수를 용이하게 하거나 기능을 향상시키기 위해 소프트웨어를 변경하는 활동

- 유지보수의 비용 산정 : $M = P + Ke^{(c-d)}$

핵심 209 | 외계인 코드

- 아주 오래전에 개발되어 유지보수 작업이 매우 어려운 프로그램
- 일반적으로 15년전 또는 그 전에 개발된 프로그램을 의미하며, 프로그램 내에 문서화(Documentation)를 철저하게 해 두면 외계인 코드를 방지할 수 있음

핵심 210 | 객체지향 기법

- 현실 세계의 개체(Entity)를 기계의 부품처럼 하나의 객체(Object)로 만들어, 기계적인 부품들을 조합하여 제품을 만들듯이 소프트웨어를 개발할 때도 객체들을 조합해서 작성할 수 있도록 하는 기법
- 구조적 기법의 문제점으로 인한 소프트웨어 위기의 해결책으로 채택되어 사용되고 있음
- 소프트웨어의 재사용 및 확장을 용이하게 함으로써 고품질의 소프트

- 웨어를 빠르게 개발할 수 있으며 유지보수가 쉬움
- 복잡한 구조를 단계적·계층적으로 표현하고, 멀티미디어 데이터 및 병렬 처리를 지원함
- 현실 세계를 모형화하여 사용자와 개발자의 이해가 용이함

핵심 211 | 객체지향 기법의 구성 요소

데이터	<ul style="list-style-type: none"> · 객체가 가지고 있는 정보로 속성이나 상태, 분류 등을 나타냄 · 속성(Attribute), 상태, 변수, 상수, 자료 구조라고도 함
객체 함수	<ul style="list-style-type: none"> · 객체가 수행하는 기능으로 객체가 갖는 데이터(속성, 상태)를 처리하는 알고리즘 · 객체의 상태를 참조하거나 변경하는 수단이 되는 것으로 메소드(Method, 행위), 서비스(Service), 동작(Operation), 연산이라고도 함 · 기존의 구조적 기법에서의 함수, 프로시저에 해당하는 연산 기능
클래스(Class)	<ul style="list-style-type: none"> · 공통된 속성과 연산(행위)을 갖는 객체의 집합으로 객체의 일반적인 타입(Type)을 의미함 · 각각의 객체들이 갖는 속성과 연산을 정의하고 있는 틀 · 클래스에 속한 각각의 객체를 인스턴스(Instance)라 하며, 클래스로부터 새로운 객체를 생성하는 것을 인스턴스화(Instantiation)라 함
메시지(Message)	<ul style="list-style-type: none"> · 메시지는 객체들 간에 상호작용을 하는데 사용되는 수단으로 객체의 메소드(동작, 연산)을 일으키는 외부의 요구 사항임 · 메시지의 구성 요소 : 메시지를 받는 객체(수신자의 이름, 객체가 수행할 메소드 이름, 메소드를 수행할 때 필요한 인자(속성값))

핵심 212 | 객체지향 기법의 주요 기본 원칙

캡슐화(Encapsulation)	<ul style="list-style-type: none"> · 데이터와 데이터를 처리하는 함수를 하나로 묶는 것 · 캡슐화된 객체의 세부 내용이 외부에 은폐(정보 은폐)되어, 변경이 발생할 때 오류의 파급 효과가 적음 · 캡슐화된 객체들은 재사용이 용이함
정보 은닉(Information Hiding)	<ul style="list-style-type: none"> · 캡슐화에서 가장 중요한 개념으로 다른 객체에게 자신의 정보를 숨기고 연산만을 통하여 접근을 허용하는 것 · 각 객체의 수정이 다른 객체에게 주는 영향을 최소화하는 기술
상속성(Inheritance)	<ul style="list-style-type: none"> · 이미 정의된 상위 클래스(슈퍼 클래스나 부모 클래스)의 모든 속성과 연산을 하위 클래스가 물려받는 것 · 상속성을 이용하면 하위 클래스는 상위 클래스의 모든 속성과 연산을 자신의 클래스 내에서 다시 정의하지 않고서도 즉시 자신의 속성으로 사용할 수 있음 · 다중 상속성(Multiple Inheritance) : 한 개의 클래스가 두 개 이상의 상위 클래스로부터 속성과 연산을 상속받는 것
다형성(Polymorphism)	<ul style="list-style-type: none"> · 메시지에 의해 객체(클래스)가 연산을 수행하게 될 때 하나의 메시지에 대해 각 객체(클래스)가 가지고 있는 고유한 방법으로 응답할 수 있는 능력 · 객체(클래스)들은 동일한 메소드명을 이용하여 같은 의미의 응답을 함 · 응용 프로그램상에서 하나의 함수나 연산자가 2개 이상의 서로 다른 클래스의 인스턴스들을 같은 클래스에 속한 인스턴스처럼 수행할 수 있도록 하는 것

핵심 213 | 객체지향 기법의 생명 주기/분석

- **객체지향 기법의 생명 주기** : 각 과정에서 사용되는 객체, 클래스, 메소드, 속성 등이 동일한 개념으로 사용되며, 계획 및 분석, 설계, 구현, 테스트 및 검증 과정으로 이루어짐
- **객체지향 분석** : 사용자의 요구사항을 분석하여 요구된 문제와 관련

된 모든 클래스(객체), 이와 연관된 속성과 연산, 그들 간의 관계 등을 정의하여 모델링하는 작업

핵심 214 | 럼바우(Rumbaugh)의 분석 기법

- 모든 소프트웨어 구성요소를 그래픽 표기법을 이용하여 모델링하는 기법
- 분석 활동은 객체 모델링, 동적 모델링, 기능 모델링을 통해 이루어짐

객체(Object) 모델링	시스템에서 요구되는 객체를 찾아내어 속성과 연산 식별 및 객체들 간의 관계를 규정하여 객체 다이어그램으로 표현한 것
동적(Dynamic) 모델링	상태도를 이용하여 시간의 흐름에 따른 객체들 사이의 제어 흐름, 상호 작용, 동작 순서 등의 동적인 행위를 표현한 것
기능(Functional) 모델링	자료 흐름도(DFD)를 이용하여 다수의 프로세스들 간의 자료 흐름을 중심으로 처리 과정을 표현한 것

핵심 215 | 객체지향 설계(OOD)

객체 지향 분석(OOA)을 사용해서 생성한 여러 가지 분석 모델을 설계 모델로 변환하는 작업으로, 시스템 설계와 객체 설계를 수행함

시스템 설계	전체적인 시스템 구조를 설계하는 것으로, 분석 단계의 분석 모델을 서브 시스템으로 분할하고, 시스템의 계층을 정의하며 분할 과정 중에서 성능의 최적 방안, 문제 해결 전략, 자원 분배 등을 확정하는 것
객체 설계	분석 단계에서 만들어진 클래스, 속성, 관계, 메시지를 이용한 통신들을 설계 모델로 제작하고 상세화하여 구체적인 자료 구조와 알고리즘을 정의함

핵심 216 | 객체지향 프로그래밍

- 새로운 개념의 모듈 단위, 즉 객체라는 단위를 중심으로 하여 프로그램을 개발하는 기법
- 객체라는 단위를 이용하여 현실 세계에 가까운 방식으로 프로그래밍함
- 현실 세계에 가까운 방식이므로 이해하기 쉽고 조작하기 쉬운 프로그램을 개발할 수 있음
- 유지보수가 쉽고 재사용 가능한 프로그램을 만들 수 있음
- 이미 개발된 프로그램을 이용해 빠르게 확장된 프로그램을 개발할 수 있음
- 객체지향 프로그래밍 언어

객체 기반 언어	Ada, Actor와 같이 객체의 개념만을 지원하는 언어
클래스 기반 언어	Clu와 같이 객체와 클래스의 개념을 지원하는 언어
객체 지향성 언어	<ul style="list-style-type: none"> 객체, 클래스, 상속의 개념을 모두 지원하는 가장 좋은 언어 초기에 발표된 Simula로부터, Smalltalk, C++, Objective-C와 같은 언어가 있음

핵심 217 | 소프트웨어의 재사용

- 이미 개발된 인장판은 소프트웨어의 전체 혹은 일부분을 다른 소프트웨어에 개발이나 유지에 사용하는 것
- 소프트웨어 개발의 품질과 생산성을 높이기 위한 방법으로, 기존에 개발한 소프트웨어와 경험, 지식 등을 새로운 소프트웨어에 적용함
- 클래스, 객체 등의 소프트웨어 요소는 소프트웨어 재사용성을 크게 향상시켰음
- 재사용이 가능한 요소 : 전체 프로그램, 부분 코드, 응용 분야에 관

한 지식, 논리적인 데이터 모형, 프로세스 구조, 시험 계획, 설계에 관한 결정, 시스템 구조에 관한 지식, 요구 분석 사항, 문서화, 전문적인 기술과 개발 경험, 품질 보증, 응용 분야 지식 등

- 소프트웨어의 부품(모듈)의 크기가 작고 일반적인 설계일수록 재사용율이 높음
- 소프트웨어 재사용의 이점 : 개발 시간과 비용 단축, 소프트웨어 품질 및 생산성 향상, 프로젝트 실패 위험 감소, 시스템 구축 방법에 대한 지식 공유, 시스템 명세, 설계, 코드 등 문서 공유

핵심 218 | 소프트웨어 재공학

- 새로운 요구에 맞도록 기존 시스템을 이용하여 보다 나은 시스템을 구축하고, 새로운 기능을 추가하여 소프트웨어 성능을 향상시키는 것
- 유지보수 비용이 소프트웨어 개발 비용의 대부분을 차지하는 문제를 고려하여 기존 소프트웨어의 데이터와 기능들의 개조 및 개선을 통해 유지보수성과 품질을 향상시키려는 기술
- 유지보수 생산성 향상을 통해 소프트웨어 위기를 해결하는 방법
- 소프트웨어 재공학도 자동화된 도구를 사용하여 소프트웨어를 분석하고 수정하는 과정을 포함함
- 재공학의 목표 : 복잡한 시스템을 다루는 방법 구현, 다른 뷰의 생성, 잃어 버린 정보의 복구 및 제거, 부작용의 발견

핵심 219 | 소프트웨어 재공학의 주요 활동

분석	기존 소프트웨어의 명세서를 확인하여 소프트웨어의 동작을 이해하고, 재공학 대상을 선정하는 것
개조(재구조, 재구성)	<ul style="list-style-type: none"> 상대적으로 같은 추상적 수준에서 하나의 표현을 다른 표현 형태로 바꾸는 것 기존 소프트웨어의 구조를 향상시키기 위하여 코드를 재구성하는 것으로 시스템의 기능과 외적인 동작은 바뀌지 않음
역공학	기존 소프트웨어를 분석하여 소프트웨어 개발 과정과 데이터 처리 과정을 설명하는 분석 및 설계 정보를 재발견하거나 다시 만들어 내는 작업
이식	기존 소프트웨어를 다른 운영체제나 하드웨어 환경에서 사용할 수 있도록 변환하는 작업

핵심 220 | 클라이언트/서버 시스템

- 분산 시스템의 가장 대표적인 모델로, 정보를 제공하는 서버와 정보를 요구하는 클라이언트로 구성되어 있는 방식
- 서버의 종류 : 파일 서버(File Server), 데이터베이스 서버(Database Server), 트랜잭션 서버(Transaction Server), 그룹웨어 서버(Groupware Server)
- 클라이언트/서버 시스템의 소프트웨어 요소 : 애플리케이션(응용 프로그램) 요소, 데이터베이스 요소, 프리젠테이션/상호작용 요소

핵심 221 | CASE

- 소프트웨어 개발 과정에서 사용되는 요구 분석, 설계, 구현, 검사 및 디버깅 과정 전체 또는 일부를 컴퓨터와 전용 소프트웨어 도구를 사용하여 자동화하는 것
- 소프트웨어 생명 주기의 전체 단계를 연결해 주고 자동화해 주는 통합된 도구를 제공해주는 기술
- 소프트웨어 개발 도구와 방법론이 결합된 것으로, 정형화된 구조 및 방법(매커니즘)을 소프트웨어 개발에 적용하여 생산성 향상을 구

현하는 공학 기법

- CASE 사용의 이점 : 소프트웨어 개발 기간 단축 및 비용 절감, 품질 향상, 유지보수 용이, 생산성 향상, 재사용성 향상 등

•CASE 분류

상위 (Upper) CASE	소프트웨어 생명 주기의 전반부에서 사용되는 것으로, 문제를 기술(Description)하고 계획하며 요구 분석과 설계 단계를 지원하는 CASE
하위(Lower) CASE	소프트웨어 생명 주기의 후반부에서 사용되는 것으로 코드의 작성과 테스트, 문서화하는 과정을 지원하는 CASE
통합(Integrate) CASE	소프트웨어 생명 주기에 포함되는 전체 과정을 지원하기 위한 CASE

핵심 222 | 정보 저장소

- 소프트웨어를 개발하는 과정 동안에 모아진 정보를 보관하여 관리하는 곳으로 CASE 정보 저장소, CASE 데이터베이스, 요구사항 사전, 저장소 등이라고도 불림
- 초기의 소프트웨어 개발 환경에서는 사람이 정보 저장소 역할을 하였지만 오늘날에는 데이터베이스가 정보 저장소 역할을 담당함
- 도구들의 통합, 소프트웨어 시스템의 표준화, 소프트웨어 시스템 정보의 공유, 소프트웨어 재사용성의 기본이 됨

단말 장치 (DTE)	<ul style="list-style-type: none"> •데이터 통신 시스템과 외부 사용자와의 접속점에 위치하여 최종적으로 데이터를 입·출력하는 장치 •기능 : 입·출력 기능, 전송 제어 기능, 기억 기능 •지능형(스마트) 단말 장치 : CPU와 저장 장치가 내장된 단말 장치로, 프로그램을 설치하여 단독으로 일정 수준 이상의 작업 처리가 가능 •비지능형(더미) 단말 장치 : 입력 장치와 출력 장치로만 구성되어 단독으로 작업을 처리할 수 있는 능력이 없는 단말 장치
신호 변환 장치(DCE)	<ul style="list-style-type: none"> •컴퓨터나 단말 장치의 데이터를 통신 회선에 적합한 신호로 변경하거나, 통신 회선의 신호를 컴퓨터나 단말 장치에 적합한 데이터로 변경하는 신호 변환 기능을 수행 •전화 : 아날로그 데이터 ↔ 아날로그 신호 •MODEM : 디지털 데이터 ↔ 아날로그 신호 •CODEC : 아날로그 데이터 ↔ 디지털 신호 •DSU : 디지털 데이터 ↔ 디지털 신호
통신 회선	단말 장치에 입력된 데이터 또는 컴퓨터에서 처리된 결과가 실제로 전송되는 전송 선로
통신 제어 장치 (CCU)	전송 회선과 주컴퓨터 사이에 위치하여, 컴퓨터가 데이터 처리에 전념할 수 있도록 컴퓨터를 대신해 데이터 전송에 관한 전반적인 제어 기능을 수행
컴퓨터	단말 장치로부터 보내진 데이터가 처리되는 곳

핵심 225 | 꼬임선(Twisted Pair Wire)

- 전기적 간섭 현상을 줄이기 위해서 균일하게 서로 감겨있는 형태의 케이블
- 가격이 저렴하고, 설치가 간편함
- 거리, 대역폭, 데이터 전송률면에서 제약이 많음
- 다른 전기적 신호의 간섭이나 잡음에 영향을 받기가 쉬움

핵심 226 | 광섬유 케이블(Optical Fiber Cable)

- 유리를 원료로 하여 제작된 가느다란 광섬유를 여러 가닥 묶어서 케이블의 형태로 만든 것으로 광 케이블이라고도 함
- 데이터를 빛으로 바꾸어 빛의 반사 원리를 이용하여 전송
- 유선 매체 중 가장 빠른 속도와 높은 주파수 대역폭을 제공
- 대용량, 장거리 전송이 가능
- 가늘고, 가벼워 설치 용이
- 도청이 어려워 보안성이 뛰어남
- 무유도, 무누화의 성질을 가짐
- 감쇠율이 적어 리피터의 설치 간격이 넓으므로 리피터의 소요가 적음
- 설치 비용은 비싸지만 단위 비용은 저렴
- 광섬유 간의 연결이 어려워 설치시 고도의 기술이 필요

핵심 227 | 위성 마이크로파

- 지상에서 쏘아 올린 마이크로 주파수를 통신 위성을 통해 변환, 증폭한 후 다른 주파수로 지상에 송신하는 방식으로, 위성 통신에 사용됨
- 위성 통신에 사용하고 있는 주파수 대역은 3-30GHz의 극초단파(SHF)임
- 위성 통신 시스템은 통신 위성, 지구 국, 채널로 구성됨
- 대역폭이 넓어 고속대용량 통신이 가능하고, 통신 비용이 저렴
- 오류율이 적어 고품질의 정보 전송이 가능
- 통신 범위가 넓고, 전송 비용이 거리에 무관하게 일정
- 전송 지연 시간이 길고, 보안성이 취약

5과목 · 데이터 통신

핵심 223 | 데이터 통신의 개요

- 데이터 통신 : 컴퓨터의 발달을 배경으로 하여 생겨난 것으로, 컴퓨터와 각종 통신 기기 사이에서 디지털 형태로 표현된 2진 정보(0과 1)를 송수신하는 것(데이터 통신 = 데이터 전송 기술 + 데이터 처리 기술)
- 정보 통신 : 컴퓨터와 통신 기술의 결합에 의해 통신 처리 기능과 정보 처리 기능은 물론 정보의 변환, 저장 과정이 추가된 형태의 통신(정보 통신 = 전기 통신(정보 전송) + 컴퓨터(정보 처리))
- 통신의 3요소 : 정보원, 수신원, 전송 매체(=통신 회선)
- SAGE : 최초의 데이터 통신 시스템
- SABRE : 최초의 상업용 데이터 통신 시스템
- CTSS : 최초의 시분할 시스템
- APPANET : 인터넷의 효시가 된 통신 시스템
- ALPHA : 최초의 무선 패킷 교환 시스템. 회선 제어 방식 중 경쟁 방식의 모체
- SNA : 데이터 통신 시스템의 표준화가 시작

핵심 224 | 데이터 통신 시스템의 기본 구성

- 데이터 전송계 : 단말 장치, 데이터 전송 회선(신호 변환 장치, 통신 회선), 통신 제어 장치
- 데이터 처리계 : 컴퓨터(하드웨어, 소프트웨어)

- 통신 위성은 약 35,800km 정도의 정지 궤도에 위치하여 지구의 자전 속도로 운행
- 한 대의 통신 위성은 지구 표면의 약 42.4% 지역을 커버할 수 있으므로 이론상 최소한 3개의 정지 위성만 있으면 극 지역을 제외한 지구상의 어느 지점과도 통신이 가능

핵심 228 | 통신 제어 장치(CCU)

•통신 제어 장치의 기능

전송 제어	다중 접속 제어, 교환 접속 제어, 통신 방식 제어, 우회 중계 회선 설정(경로 설정)
동기 및 오류 제어	동기 제어, 오류 제어, 흐름 제어, 응답 제어, 정보 전송 단위의 정합, 데이터 신호의 직·병렬 변환, 투과성, 정보 표시 형식의 변환, 우선권 제어
그 밖의 기능	제어 정보 식별, 기밀 보호, 관리 기능

•**전처리기(FEP)** : 호스트 컴퓨터와 단말기 사이에 고속 통신 회선으로 설치되며, 통신 회선 및 단말기 제어, 메시지의 조립과 분해, 전송 메시지 검사 등을 수행하므로, 컴퓨터의 부담이 적어짐

핵심 229 | 통신 제어 프로그램

- 데이터 전송 회선과 통신 제어 장치를 이용하여 컴퓨터와 터미널 간의 데이터를 송수신(통신)하기 위한 프로그램(소프트웨어)을 총칭
- 주요 기능** : 데이터 송수신, 통신 하드웨어 제어, 이용자 인터페이스 제어
- 그 밖의 기능** : CPU의 기능 분담, 하드웨어와의 인터페이스, 데이터 통신 회선과 신호 변환기 등의 회선 제어, 접속의 확인과 종료를 제어하는 전송 제어, 오류 제어, 데이터 처리와 교환, 코드 변환, 데이터 압축력 제어, 단말 제어, 데이터 버퍼링(Buffering), 파일 관리와 회복

핵심 230 | 데이터 전송 기본

•데이터

아날로그 데이터	셀 수 없는 연속적인 값 (예) 음성, 화상, 온도, 유압 등
디지털 데이터	셀 수 있는 이산적인 값 (예) 문자, 숫자

•**신호** : 데이터를 전송 매체를 통해 전송할 수 있는 상태로 변환시켜 놓은 것

아날로그 신호	정현파(Sine Wave)에 주파수, 진폭, 위상 특성을 포함하여 표현되는 전기적 신호가 연속적으로 변하는 파형
디지털 신호	2진수 0과 1에 대한 전압 펄스의 연속적인 구성

•**주파수** : 단위 시간(주로 1초) 내에 신호 파형이 반복되는 횟수를 의미(단위 : Hz)

고주파	파형의 가로 폭이 좁음, 고속 전송에 사용, 전송 거리가 짧음
저주파	파형의 가로 폭이 넓음, 저속 전송에 사용, 전송 거리가 길음
단위 시간과의 관계	$f = \frac{1}{T}$ (f : 주파수, T : 주기)
주요 데이터의 주파수	•음성 : 300Hz ~ 3400Hz •UHF(Ultra High Frequency) : 300MHz ~ 3000MHz

•**대역폭(Bandwidth)** : 주파수의 변화 범위, 즉 상한 주파수와 하한 주파수의 차이를 의미

핵심 231 | 아날로그/디지털 전송

아날로그 전송	<ul style="list-style-type: none"> •전송 매체를 통해 전달되는 신호가 아날로그 형태인 것 •신호의 감쇠 현상이 심하므로 장거리 전송시 증폭기에 의해 신호를 증폭하여 전송하며, 이때 신호에 포함된 잡음까지도 같이 증폭되기 때문에 오류의 확률이 높음
디지털 전송	<ul style="list-style-type: none"> •전송 매체를 통해 전달되는 신호가 디지털 형태인 것 •장거리 전송시 증폭기에 의해 원래의 신호 내용을 다시 복원한 다음 전송하는 방식이기 때문에 잡음에 의한 오류율이 낮음 •대역폭을 효율적으로 이용하여 더 많은 용량을 전송할 수 있음 •아날로그나 디지털 정보의 암호화를 쉽게 구현할 수 있음 •전송 장비의 소형화, 가격의 저렴화

핵심 232 | 직렬/병렬 전송

직렬 전송	<ul style="list-style-type: none"> •정보를 구성하는 각 비트들이 하나의 전송 매체를 통하여 한 비트씩 순서적으로 전송되는 형태 •전송 속도가 느리지만 구성 비용이 적게 듦 •원거리 전송에 적합하며 대부분의 데이터 통신에 사용됨
병렬 전송	<ul style="list-style-type: none"> •정보를 구성하는 각 비트들이 여러 개의 전송 매체를 통하여 동시에 전송되는 형태 •전송 속도는 빠르지만 구성 비용이 많이 듦 •근거리 전송에 적합하며 주로 컴퓨터와 주변기기 사이의 데이터 전송에 사용됨

핵심 233 | 통신 방식

- 단방향(Simplex) 통신** : 한쪽 방향으로만 전송이 가능한 방식 (예) 라디오, TV
- 반이중(Half-Duplex) 통신** : 양방향 전송이 가능하지만 동시에 양쪽 방향에서 전송할 수 없는 방식 (예) 무전기, 모뎀을 이용한 데이터 통신
- 전이중(Full-Duplex) 통신** : 동시에 양방향 전송이 가능한 방식으로, 전송량이 많고, 전송 매체의 용량이 클 때 사용 (예) 전화, 전용선을 이용한 데이터 통신

핵심 234 | 비동기식 전송

- 한 문자를 나타내는 부호(문자 코드) 앞뒤에 Start Bit와 Stop Bit를 붙여서 Byte와 Byte를 구별하여 전송하는 방식
- 시작 비트, 전송 문자(정보 비트), 정지 비트로 구성된 한 문자를 단위로 하여 전송하며, 오류 검출을 위한 패리티 비트(Parity Bit)를 추가하기도 함
- 문자와 문자 사이의 휴지 시간(Idle Time)이 불규칙함
- 2000bps(약 2Kbps) 이하의 단거리 전송에 사용
- 문자마다 시작, 정지를 알리기 위한 비트가 2~3bit씩 추가되므로, 전송 효율이 떨어짐

핵심 235 | 동기식 전송

- 미리 정해진 수 만큼의 문자열을 한 블록(프레임)으로 만들어 일시에 전송하는 방식
- 프레임 단위로 전송하므로 전송 속도가 빠름
- 시작/종료 비트로 인한 오버헤드가 없고, 휴지 시간이 없으므로, 전송 효율이 좋음
- 주로 원거리 전송에 사용

- 단말기는 반드시 버퍼 기억 장치를 내장하여야 함
- 비트 동기 방식과 블록 동기 방식이 있음
- 블록 동기 방식은 문자 동기 방식과 비트 동기 방식으로 나뉨

문자 동기 방식	<ul style="list-style-type: none"> • SYN 등의 동기 문자(전송 제어 문자)에 의해 동기를 맞추는 방식 • BSC 프로토콜에서 사용됨
비트 동기 방식	<ul style="list-style-type: none"> • 데이터 블록의 처음과 끝에 8비트의 플래그 비트(01111110)를 표시하여 동기를 맞추는 방식 • HDLC, SDLC 프로토콜에서 사용됨

핵심 236 | 모뎀(MODEM)

- 컴퓨터나 단말 장치로부터 전송되는 디지털 데이터를 아날로그 회선에 적합한 아날로그 신호로 변환하는 변조(Modulation) 과정과 그 반대의 복조(DEModulation) 과정을 수행
- 디지털 데이터를 공중 전화 교환망(PSTN)과 같은 아날로그 통신망을 이용하여 전송할 때 사용
- 기능 : 변복조 기능, 자동 응답 기능, 자동 호출 기능, 자동 속도 조절 기능, 모뎀 시험 기능

핵심 237 | DSU(Digital Service Unit)

- 컴퓨터나 단말 장치로부터 전송되는 디지털 데이터를 디지털 회선에 적합한 디지털 신호로 변환하는 과정과 그 반대의 과정을 수행
- 신호의 변조 과정이 없이 단순히 유니폴라(단극성) 신호를 바이폴라(양극성) 신호로 변환하여 주는 기능만 제공하기 때문에 모뎀에 비하여 단순
- 디지털 데이터를 공중 데이터 교환망(PDSN)과 같은 디지털 통신망을 이용하여 전송할 때 사용됨
- 송수신 기능과 타이밍 회복 기능을 DSU 자체에서 수행함
- 속도가 빠르고, 오류율이 낮음

핵심 238 | 신호 변환 방식 - 디지털 변조

- 모뎀(MODEM)을 이용하여 디지털 데이터를 아날로그 신호로 변조하는 방식
- 변조 방식

진폭 편이 변조(ASK)	<ul style="list-style-type: none"> • 2진수 0과 1을 서로 다른 진폭의 신호로 변조 • 이 방식을 사용하는 모뎀은 구조가 간단하고, 가격이 저렴함 • 신호 변동과 잡음에 약하여 데이터 전송용으로 거의 사용되지 않음
주파수 편이 변조(FSK)	<ul style="list-style-type: none"> • 2진수 0과 1을 서로 다른 주파수로 변조 • 1200bps 이하의 저속도 비동기식 모뎀에서 사용됨 • 이 방식을 사용하는 모뎀은 구조가 간단하고, 신호 변동과 잡음에도 강함
위상 편이 변조(PSK)	<ul style="list-style-type: none"> • 2진수 0과 1을 서로 다른 위상을 갖는 신호로 변조 • 위상의 시작 위치를 다르게 하여 신호를 전송 • 한 위상에 1비트(2위상), 2비트(4위상), 또는 3비트(8위상)를 대응시켜 전송하므로, 속도를 증가시킬 수 있음 • 중·고속의 동기식 모뎀에 많이 사용됨
직교 진폭 변조(QAM) = 진폭 위상 변조, 직교 위상 변조	<ul style="list-style-type: none"> • 반송파의 진폭과 위상을 상호 변환하여 신호를 얻는 변조 방식 • 제한된 전송 대역 내에서 고속 전송이 가능(9600bps)함 • 9600bps 모뎀의 표준 방식 • 신호의 진폭과 위상을 표시하는 신호의 구분점이 통신 회선의 잡음과 위상 변화에 대하여 우수한 특성을 지님

핵심 239 | 신호 변환 방식 - 펄스 코드 변조(PCM)

- 화상, 음성, 동영상 비디오, 가상 현실 등과 같이 연속적인 시간과 진폭을 가진 아날로그 데이터를 디지털 신호로 변조하는 방식으로, CODEC을 이용함
- 펄스 변조 : 펄스파의 진폭, 폭, 위상 등을 변화시키는 변조 방식

연속 레벨 변조	펄스 진폭 변조(PAM), 펄스 폭 변조(PWM), 펄스 위상 변조(PPM), 펄스 주파수 변조(PFM)
불연속 레벨 변조	펄스 수 변조(PNM), 펄스 부호 변조(PCM), 델타 변조(ΔM)

- 펄스 코드 변조(PCM) 순서 : 표준화 → 양자화 → 부호화

표본화 (Sampling)	<ul style="list-style-type: none"> • 음성, 영상 등의 연속적인 신호 파형을 일정 시간 간격으로 검출하는 단계 • 사명의 표본화 이론 : 어떤 신호 $f(t)$가 의미를 지니는 최고 주파수보다 2배 이상의 주파수로 균일한 시간 간격동안 채집된다면 이 채집된 데이터는 원래의 신호가 가진 모든 정보를 포함함 • 표본화에 의해 검출된 신호를 PAM 신호라고 하며, 아날로그 형태임 • 표본화 횟수 = 2배 × 최고 주파수 • 표본화 간격 = 1/표본화 횟수
양자화 (Quantizing)	<ul style="list-style-type: none"> • 표본화된 PAM 신호를 유한 개의 부호에 대한 대표값으로 조정하는 과정 • 실수 형태의 PAM 신호를 반올림하여 정수형으로 만듦 • 양자화 잡음 : 표본 측정값과 양자화 파형과의 오차를 말하며, 주로 PCM 단국 장치에서 발생 • 양자화 잡음은 양자화 레벨을 세밀하게 함으로써 줄일 수 있으나, 이 경우 데이터의 양이 많아지고 전송 효율이 낮아짐 • 양자화 레벨 : PAM 신호를 부호화할 때 2진수로 표현할 수 있는 레벨(양자화 레벨 = $2^{\text{표본당 전송 비트 수}}$)
부호화 (Encoding)	양자화된 PCM 펄스의 진폭 크기를 2진수 (1과 0)로 표시하는 과정
복호화 (Decoding)	수신된 디지털 신호(PCM 신호)를 PAM 신호로 되돌리는 단계
여파화 (Filtering)	PAM 신호를 원래의 입력 신호인 아날로그 신호로 복원하는 과정

핵심 240 | 베이스밴드(Base Band) 전송

- 컴퓨터나 단말 장치등에서 처리된 디지털 데이터를 다른 주파수 대역으로 변조하지 않고 직류 펄스의 형태 그대로 전송하는 것으로, 기저대역 전송이라고도 함
- 신호만 전송되기 때문에 전송 신호의 품질이 좋음
- 직류를 사용하므로 감쇠 등의 문제가 있어 장거리 전송에 적합하지 않음
- 컴퓨터와 주변 장치 간의 통신이나 LAN 등 비교적 가까운 거리에서 사용됨

핵심 241 | 다중화기(Multiplexer)

- 하나의 통신 회선을 여러 대의 단말기가 동시에 접속하여 사용할 수 있도록 하는 장치
- 다중화(Multiplexing) : 하나의 통신 회선을 다수의 단말기가 공유할 수 있도록 하는 것으로, 다중화를 위한 장치에는 다중화기, 집중화기, 공동 이용기가 있음
- 고속 통신 회선의 주파수나 시간을 일정한 간격으로 나누어 각 단말기에 할당함

- 통신 회선을 공유함으로써 전송 효율을 높이고, 통신 회선의 수와 설치 비용을 줄일 수 있음
- 다중화기는 주파수 분할 다중화기와 시분할 다중화기로 구분됨
- 입력 회선의 수와 출력 회선의 수가 같음
- 여러 대의 단말기의 속도의 합이 하나의 통신 회선의 속도와 같음($A + B + C = D$)

핵심 242 | 주파수 분할 다중화기(FDM)

- 통신 회선의 주파수를 여러 개로 분할하여 여러 대의 단말 장치가 동시에 사용할 수 있도록 한 것
- 전송 신호에 필요한 대역폭보다 전송 매체의 유효 대역폭이 큰 경우에 사용
- 다중화기 자체에 변복조 기능이 내장되어 있어 모뎀을 설치할 필요 없음
- 다른 다중화기에 비해 구조가 간단하고 가격이 저렴함
- 대역폭을 나누어 사용하는 각 채널들 간의 상호 간섭을 방지하기 위한 보호 대역(Guard Band)이 필요함
- 보호 대역(Guard Band) 사용으로 인한 대역폭의 낭비가 초래됨
- 저속(1200baud 이하)의 비동기식 전송, 멀티 포인트 방식, 아날로그 신호 전송에 적합함

핵심 243 | 시분할 다중화기(TDM)

- 통신 회선의 대역폭을 일정한 시간 폭(Time Slot)으로 나누어 여러 대의 단말 장치가 동시에 사용할 수 있도록 한 것
- 디지털 회선에서 주로 이용하며, 대부분의 데이터 통신에 사용됨
- 다중화기의 내부 속도와 단말 장치의 속도 차이를 보완해 주는 버퍼가 필요함

동기식 시분할 다중화기 (STDM)	<ul style="list-style-type: none"> • 모든 단말 장치에 균등한(고정된) 시간 폭을 제공 • 전송되는 데이터의 시간 폭을 정확히 맞추기 위한 동기 비트가 필요 • 전송 매체의 데이터 전송률이 전송 디지털 신호의 데이터 전송률을 능가할 때 사용 • 전송할 데이터가 없는 경우에도 시간 폭이 제공되므로 효율성이 떨어짐
비동기식 시분할 다중화기 (ATDM)	<ul style="list-style-type: none"> • 전송할 데이터가 있는 단말 장치에만 시간 폭을 제공 하므로, 전송 효율이 높음 • 동기식 시분할 다중화기보다 많은 수의 단말기들을 전송 매체에 접속할 수 있음 • 데이터 전송률이 많아질수록 전송 지연이 길어짐 • 동기식 시분할 다중화기에 비해 접속에 소요되는 시간이 김 • 주소 회로, 흐름 제어, 오류 제어 등의 기능이 필요하므로 장비가 복잡하고, 가격이 비쌈 • 지능 다중화기, 확률적 다중화기, 통계 시분할 다중화기라고도 함

핵심 244 | 역 다중화기(Inverse Multiplexer)

- 광대역 회선 대신에 두 개의 음성 대역 회선을 이용하여 데이터를 전송할 수 있도록 하는 장치
- 광대역 통신 회선을 사용하지 않고도 9600bps 이상의 광대역 속도를 얻을 수 있으므로, 비용을 절감할 수 있음
- 하나의 통신 회선이 고장 나더라도 나머지 하나의 회선을 통해 1/2 속도로 전송을 계속 유지할 수 있음

- 여러 가지 변화에 대응해 여러 가지의 전송 속도를 얻을 수 있음
- 음성 회선의 특성상 두 회선의 상대적 전송 지연이 발생할 수 있음

핵심 245 | 집중화기(Concentrator)

- 하나 또는 소수의 회선에 여러 대의 단말기를 접속하여 사용할 수 있도록 하는 장치
- 실제 전송할 데이터가 있는 단말기에만 통신 회선을 할당하여 동적으로 통신 회선을 이용할 수 있도록 함
- 한 개의 단말 장치가 통신 회선을 점유하게 되면 다른 단말 장치는 회선을 사용할 수가 없으므로, 다른 단말기의 자료를 임시로 보관할 버퍼가 필요함
- 입력 회선의 수가 출력 회선의 수보다 같거나 많음
- 여러 대의 단말 장치의 속도의 합이 통신 회선의 속도보다 크거나 같음($A + B + C \geq D$)
- 회선의 이용률이 낮고, 불규칙적인 전송에 적합함

핵심 246 | 통신 속도와 통신 용량

• 통신 속도

변조 속도	1초 동안 몇 개의 신호 변화가 있었는가를 나타내는 것 단위 : baud)
신호 속도	1초 동안 전송 가능한 비트의 수(단위 : bps(bit/sec)) · 데이터 신호 속도(bps) = 변조 속도(baud) × 변조시 상태 변화 수 · 변조 속도(baud) = 데이터 신호 속도(bps) / 변조시 상태 변화 수
전송 속도	단위 시간에 전송되는 데이터의 양(문자, 블록, 비트, 단어 수 등)
베어리 속도	데이터 신호에 동기 문자, 상태 신호 등을 합한 속도 단위 : bps(bit/sec))

• 변조시 상태 변화 수 : 모노비트(Monobit) = 1비트, 디비트(Dibit) = 2비트, 트리비트(Tribit) = 3비트, 쿼드비트(Quadbit) = 4비트

• 통신 용량 : 단위 시간 동안 전송 회선이 최대 전송할 수 있는 통신 정보량

샤논(Shannon)의 정의	$C = W \cdot \log_2(1+S/N)$ [bps] C : 통신 용량 W : 대역폭 S : 신호 전력, N : 잡음 전력
-----------------	--

전송로의 통신 용량을 늘리기 위한 방법	<ul style="list-style-type: none"> • 주파수 대역폭을 늘림 • 신호 세력을 높임 • 잡음 세력을 줄임
-----------------------	---

핵심 247 | 전송 제어의 기본

- 전송 제어 : 데이터의 원활한 흐름을 위하여 입출력 제어, 회선 제어, 동기 제어, 오류 제어, 흐름 제어 등을 수행하는 것
- OSI 7 계층의 데이터 링크 계층(2계층)에서 수행하는 기능
- 전송 제어 절차 : 데이터 통신 회선의 접속 → 데이터 링크 설정(확립) → 정보 메시지 전송 → 데이터 링크 종결 → 데이터 통신 회선의 절단

데이터 통신 회선 접속	<ul style="list-style-type: none"> • 통신 회선과 단말기를 물리적으로 접속 • 교환 회선을 이용한 포인트 투 포인트 방식이나 멀티 포인트 방식으로 연결된 경우에 필요한 단계
데이터 링크 설정(확립)	접속된 통신 회선상에서 송 · 수신측 간의 확실한 데이터 전송을 수행하기 위해서 논리적 경로를 구성하는 단계
정보 메시지 전송	설정된 데이터 링크를 통해 데이터를 수신측에 전송하며, 오류 제어와 순서 제어를 수행

데이터 링크 종결	송·수신측 간의 논리적 경로를 해제
데이터 통신 회선의 절단	통신 회선과 단말기 간의 물리적 접속을 절단

핵심 248 | BSC

- 문자(Character) 위주의 프로토콜로, 각 프레임에 전송 제어 문자를 삽입하여 전송을 제어
- 문자 코드에 의존적이며, 사용할 수 있는 코드가 제한적임
- 통신하는 컴퓨터들이 사용하는 문자 코드 체계가 통일되어 있어야 함
- 반이중(Half Duplex) 전송만 지원
- 주로 동기 전송을 사용하나 비동기 전송 방식을 사용하기도 함
- 포인트 투 포인트, 멀티 포인트 방식에서 주로 사용
- 에러 및 흐름 제어를 위해 Stop-and-Wait ARQ를 사용
- 전파 지연 시간이 긴 선로에서는 비효율적
- 오류 검출이 어렵고, 전송 효율이 나쁨

• 프레임 구조

SYN	SYN	SOH	헤더	STX	본문	ETX/ETB	BCC
-----	-----	-----	----	-----	----	---------	-----

- 전송 제어 문자 : 링크 관리, 프레임의 시작 및 끝의 구별과 에러 제어 등의 기능을 함

기호	기능
SYN	문자 동기
SOH	헤드의 시작
STX	본문의 시작 및 헤드의 종료
ETX	본문의 종료
ETB	블록의 종류
EOT	전송 종료 및 데이터 링크의 해제
ENQ	상대편에 데이터 링크 설정 및 응답 요구
DLE	전송 제어 문자 앞에 삽입하여 전송 제어 문자임을 알림 데이터 투과성을 위해 삽입)
ACK	수신된 메시지에 대한 긍정 응답
NAK	수신된 메시지에 대한 부정 응답

핵심 249 | HDLC

- 비트(Bit) 위주의 프로토콜로, 각 프레임에 데이터 흐름을 제어하고 오류를 보정할 수 있는 비트 열을 삽입하여 전송
- 포인트 투 포인트 및 멀티 포인트, 루프 방식에서 모두 사용 가능
- 단방향, 반이중, 전이중 통신을 모두 지원하며 동기식 전송 방식을 사용
- 에러 제어를 위해 Go-Back-N과 선택적 재전송(Selective Repeat) ARQ를 사용
- 흐름 제어를 위해 슬라이딩 윈도우 방식을 사용
- 전송 제어상의 제한을 받지 않고 자유로이 비트 정보를 전송할 수 있음(비트 투과성)
- 전송 효율과 신뢰성이 높음

• HDLC 프레임 구조

8bit	8bit(확장가능)	8bit	임의(n)bit	16/32bit	8bit
플래그	주소부	제어부	정보부	FCS	플래그

- 플래그(Flag) : 프레임의 시작과 끝을 나타내는 고유한 비트 패턴(01111110). 프레임의 시작과 끝을 구분, 동기 유지(통화로의 혼선을

방지하기 위해), 비트 투과성을 이용한 기본적인 오류 검출 등의 역할을 함

- FCS(프레임 검사 순서 필드) : 프레임 내용에 대한 오류 검출을 위해 사용되는 부분으로, 일반적으로 CRC 코드가 사용됨

• HDLC의 프레임 종류

정보(I) 프레임	• 제어부가 '0'으로 시작하는 프레임 • 사용자 데이터를 전달하는 역할
감독(S) 프레임	• 제어부가 '10'으로 시작하는 프레임 • 오류 제어와 흐름 제어를 위해 사용
비번호(U) 프레임	• 제어부가 '11'으로 시작하는 프레임 • 링크의 동작 모드 설정과 관리를 함

• HDLC의 데이터 전송 모드

표준(정규) 응답 모드 (NRM)	• 반이중 통신을 하는 포인트 투 포인트 또는 멀티 포인트 불균형 링크 구성에 사용 • 종국은 주국의 허가(Poll)가 있을 때에만 송신
비동기 응답 모드 (ARM)	• 전이중 통신을 하는 포인트 투 포인트 불균형 링크 구성에 사용 • 종국은 주국의 허가(Poll) 없이도 송신이 가능하지만, 링크 설정이나 오류 복구 등의 제어 기능은 주국만 함
비동기 균형 (평형) 모드 (ABM)	• 포인트 투 포인트 균형 링크에서 사용 • 혼합국끼리 허가없이 언제나 전송할 수 있도록 설정

핵심 250 | 회선 제어 방식

경쟁(Contention) 방식	• 회선 접속을 위해서 서로 경쟁하는 방식 • 송신 요구를 먼저 한 쪽이 송신권을 가짐 • 포인트 투 포인트 방식에서 주로 사용 • 데이터 링크가 설정되면 정보 전송이 종료되기 전까지는 독점적으로 정보를 전송할 수 있음 • 대표적인 시스템으로는 ALOHA가 있음
폴링/셀렉션(Polling/Selection) 방식	• 컴퓨터에서 송수신 제어권을 가지고 있는 방식 • 폴링(Polling) : 컴퓨터에서 단말기에게 전송할 데이터가 있는지를 물어 전송할 데이터가 있다면 전송을 허가하는 방식으로, 단말기에서 컴퓨터로 보낼 데이터가 있는 경우에 사용 • 셀렉션(Selection) : 컴퓨터가 단말기로 전송할 데이터가 있는 경우 그 단말기가 받을 준비가 되었는가를 묻고, 준비가 되어 있다면 컴퓨터에서 단말기로 데이터를 전송하는 방식

핵심 251 | 오류의 발생 원인

- 감쇠 : 전송 신호 세력이 전송 매체를 통과하는 과정에서 거리에 따라 약해지는 현상으로, 감쇠 현상을 해결하기 위해 중계기(기지국)를 이용함
- 지연 왜곡 : 하나의 전송 매체를 통해 여러 신호를 전달했을 때 주파수에 따라 그 속도가 달라지므로 생기는 오류
- 백색 잡음 : 전송 매체 내부에서 온도에 따라 전자의 운동량이 변화함으로써 생기는 잡음으로, 가우스 잡음, 열 잡음이라고도 함
- 상호 변조(간섭) 잡음 : 서로 다른 주파수들이 하나의 전송 매체를 공유할 때 주파수 간의 합(和)이나 차(差)로 인해 새로운 주파수가 생성되는 잡음
- 누화 잡음 = 혼선 : 인접한 전송 매체의 전자기적 상호 유도 작용에 의해 생기는 잡음으로, 전화 통화 중 다른 전화의 내용이 함께 들리는 현상
- 충격성 잡음 : 번개와 같은 외부적인 충격 또는 통신 시스템의 결함이나 파손 등의 기계적인 충격에 의해 생기는 잡음으로, 디지털 데이터를 전송하는 경우 중요한 오류 발생 요인이 됨
- 우연적 왜곡과 시스템적 왜곡

우연적 왜곡	· 예측할 수 없이 무작위로 발생하는 왜곡
시스템적 왜곡	· 백색 잡음, 충격 잡음, 누화 잡음, 위상 히트 잡음 등
시스템적 왜곡	· 전송 매체에서 언제든지 일어날 수 있는 왜곡
왜곡	· 손실, 감쇠, 하모닉 왜곡(신호의 감쇠가 진폭에 의해 달라지는 것) 등

핵심 252 | 전송 오류 제어 방식

전진(순방향 오류 수정 (FEC))	· 재전송 요구 없이 오류 검출과 수정을 스스로 하는 방식 · 역채널이 필요 없고, 연속적인 데이터 흐름이 가능 · 데이터 비트 이외에 오류 검출 및 수정을 위한 비트(잉여 비트)들이 추가로 전송되어야 하기 때문에 전송 효율이 떨어짐 · 해밍 코드, 상승 코드 방식이 있음
후진(역방향 오류 수정 (BEC))	· 데이터 전송 과정에서 오류가 발생하면 송신측에 재전송을 요구하는 방식 · 패리티 검사, CRC, 블록화 방식 등을 사용하여 오류를 검출하고, 오류 제어는 자동 반복 요청(ARQ)에 의해 이루어짐

핵심 253 | 자동 반복 요청(ARQ)

오류 발생시 수신측은 오류 발생을 송신측에 통보하고, 송신측은 오류 발생 블록을 재전송하는 모든 절차를 의미

정지-대기 (Stop-Wait) ARQ	· 송신측에서 한 개의 블록을 전송한 후 수신측으로부터 응답을 기다리는 방식 · 구현 방법이 가장 단순하지만, 전송 효율이 떨어짐
연속(Continuous) ARQ	· 연속적으로 데이터 블록을 보내는 방식 · Go-Back-N ARQ : 오류가 발생한 블록 이후의 모든 블록을 재전송 · 선택적 재전송(Selective Repeat) ARQ : 오류가 발생한 블록만을 재전송하는 방식으로, 수신측에서 데이터를 처리하기 전에 원래 순서대로 조립해야 하므로, 더 복잡한 논리 회로와 큰 용량의 버퍼가 필요
적응적(Adaptive) ARQ	· 데이터 블록의 길이를 채널의 상태에 따라 그때그때 동적으로 변경하는 방식으로, 전송 효율이 제일 좋음 · 제어 회로가 복잡하고, 비용이 많이들어 현재 거의 사용되지 않음

핵심 254 | 오류 검출 방식

패리티 검사	· 전송 비트에 1비트의 검사 비트인 패리티 비트(Parity Bit)를 추가하여 오류를 검출 · 가장 간단한 방식이지만, 2개의 비트 동시에 오류가 발생하면 검출이 불가능 · 오류를 검출만 할 수 있고, 수정은 하지 못함 · 홀수/짝수 수직 패리티 체크와 홀수/짝수 수평 패리티 체크가 있음
순환 중복 검사 (CRC)	· 다항식 코드를 사용하여 오류를 검출하는 방식 · 동기식 전송에서 주로 사용 · HDLC 프레임의 FCS(프레임 검사 순서 필드)에 사용되는 방식 · 집단 오류를 검출할 수 있고, 검출율이 높으므로 가장 많이 사용함
해밍 코드	· 수신측에서 오류가 발생한 비트를 검출한 후 직접 수정하는 방식 · 1비트의 오류만 수정이 가능하며, 정보 비트 외에 잉여 비트가 많이 필요함 · 전송 비트 중 1, 2, 4, 8, 16, 32, 64, ..., 2^n 번째를 오류 검출을 위한 패리티 비트로 사용함
상승 코드 방식	· 순차적 디코딩과 한계값 디코딩을 사용하여 오류를 수정 · 여러 비트의 오류를 수정할 수 있음

핵심 255 | 전용 회선/교환 회선

전용 회선	· 송·수신 상호간에 통신 회선이 항상 고정되어 있는 방식 · 전송 속도가 빠르고, 전송 오류가 적음 · 사용 방법이 간편하며 업무 적용이 쉬움 · 전송할 데이터의 양이 많고, 회선의 사용 시간이 많을 때 효율적 · 고장 발생시 유지·보수 유리 · 연결 방식에는 포인트 투 포인트 방식과 멀티 드롭 방식이 있음
교환 회선	· 교환기에 의해서 송·수신 상호간 통신 회선이 연결되는 방식 · 전용 회선에 비해 전송 속도(4800bps 이하)가 느림 · 정보 보안을 위해 정보 누설과 파괴를 방지하는 조치가 필요 · 회선을 공유하므로 효율도가 높고, 통신 장치와 회선 비용을 줄일 수 있음 · 전송할 데이터의 양이 많지 않고, 회선 사용 시간이 적을 때 효율적 · 교환 기술의 성능 비교 요소 : 전파 지연, 전송 시간, 노드 지연, 데이터 처리율

핵심 256 | 회선 구성 방식

포인트 투 포인트(Point -to-Point)	· 중앙 컴퓨터와 단말기를 일대일 독립적으로 연결한 방식 · 통신망을 성형(Star)으로 구성할 때 사용
멀티 드롭 (Multi-drop) =멀티 포인 트	· 여러 대의 단말기들을 한 개의 통신 회선에 연결하는 방식 · 단말기는 주소 판단 기능과 버퍼를 가지고 있어야 함 · 회선 공유로 효율도가 높고, 가격도 저렴 · 선로의 속도, 단말기에 의해 생기는 교통량, 하드웨어와 소프트웨어의 처리 능력에 따라 연결할 수 있는 단말기의 수가 달라짐 · 통신망을 버스형(Bus)으로 구성할 때 사용
회선 다중 방식 (Line Multiplexing)	· 여러 대의 단말기들을 다중화 장치를 이용하여 중앙 컴퓨터와 연결하는 방식 · 중앙 컴퓨터와 다중화 장치 사이를 대용량 회선으로 연결 · 전송 속도 및 효율 높음

핵심 257 | 회선 교환 방식

- 통신을 원하는 두 지점을 교환기를 이용하여 물리적으로 접속시키는 방식
- 데이터 전송 전에 먼저 통신망을 통한 연결이 필요
- 접속이 되고 나면 그 통신 회선은 전용 회선에 의한 통신처럼 전달됨(고정 대역 전송)
- 접속에는 긴 시간이 소요되나, 일단 접속되면 전송 지연이 거의 없어 실시간 전송이 가능
- 데이터 전송에 필요한 전체 시간이 축적 교환 방식에 비해 김
- 일정한 데이터 전송률을 제공하므로 동일한 전송 속도가 유지됨
- 전송된 데이터의 있어서의 오류 제어나 흐름 제어는 사용자에게 의해 수행되어야 함
- **통신 과정** : 호(링크) 설정 → 데이터 전송 → 호(링크) 해제
- 공간 분할 교환 방식과 시간 분할 교환 방식으로 나뉘고, 시분할 교환 방식에는 TDM 버스 교환 방식, 타임 슬롯 교환 방식, 시간 다중화 교환 방식이 있음
- **제어 신호 방식**

감시(관리) 제어 신호	· 상대방과 통화하는데 필요한 자원을 이용할 수 있는지를 결정하고 알리는데 사용 · 서비스 요청, 응답, 경보 및 휴지 상태 복귀 신호 등의 기능
주소 제어 신호	상대방을 식별하고 경로를 배정하여 전화를 올리게 하는 데 사용
호 정보 제어 신호	신호음, 연결음, 통화중 신호음 등 호의 상태 정보를 송신자에게 제공
망 관리 제어 신호	통신망의 전체적인 운영, 유지, 고장 수리 등을 위해 사용

핵심 258 | 축적 교환 방식

- 송신측에서 전송한 데이터를 수신측 교환기에 저장시켰다가 이를 다시 적절한 통신 경로를 선택하여 수신측 터미널에 전송하는 방식

메시지 교환 방식	· 교환기가 일단 송신측의 메시지를 받아서 저장한 후 전송 순서가 되면 수신측으로 전송 · 각 메시마다 전송 경로를 결정하고, 수신 주소를 붙여서 전송 · 전송 메시지는 추후 검색이 가능 · 전송 지연 시간이 매우 김 · 응답 시간이 느려 대화형 데이터 전송에 부적절
패킷 교환 방식	· 메시지를 일정한 길이의 패킷으로 잘라서 전송 · 패킷 교환망은 OSI 7 계층의 네트워크 계층에 해당함 · 회선 이용률이 높음 · 수신측에서 분할된 패킷을 재조립해야 함 · 응답 시간이 빠르므로, 대화형 응용이 가능 · 음성 전송보다 데이터 전송에 더 적합 · 패킷(Packet) : 전송 혹은 다중화를 목적으로, 메시지를 일정한 비트 수로 분할하여 송·수신측 주소와 제어 정보 등을 부가하여 만든 데이터 블록

핵심 259 | 정보 저장소

패킷 교환 방식의 종류

가상 회선 방식	· 단말기 상호간에 논리적인 가상 통신 회선을 미리 설정하여 송신지와 수신지 사이의 연결을 확립한 후에 설정된 경로를 따라 패킷들을 순서적으로 운반하는 방식 · 통신이 이루어지는 컴퓨터 사이의 데이터 전송의 안정, 신뢰성이 보장됨 · 패킷의 송·수신 순서가 같음 · 통신 과정 : 호 설정 → 데이터 전송 → 호 해제
데이터그램 방식	· 연결 경로를 설정하지 않고 인접한 노드들의 트래픽(전송량) 상황을 감안하여 각각의 패킷들을 순서에 상관없이 독립적으로 운반하는 방식 · 패킷마다 전송 경로가 다르며, 송·수신 순서가 다를 수 있음

핵심 260 | 패킷 교환망의 기능

- **패킷 다중화** : 동시에 다수의 상대 터미널과 통신을 수행하도록 하는 기능
- **경로 제어(Routing)** : 가장 효율적인 전송로를 선택하는 기능
- **논리 채널** : 송수신측 단말기 사이에서 논리 채널(가상 회선)을 설정하는 기
- **순서 제어** : 패킷의 송수신 순서를 제어하는 기능
- **트래픽 제어(Traffic Control)** : 전송되는 패킷의 흐름 또는 그 양을 조절하기 위해 교착 상태(Dead Lock)의 방지, 흐름 제어 등을 수행
- **오류 제어** : 오류를 검출하고 정정하는 기능

핵심 261 | 경로 제어(Routing)

- 송수신측 간의 송신 경로 중에서 최적 패킷 교환 경로를 설정하는 기능
- **경로 설정 요소** : 성능 기준, 경로의 결정 시간과 장소, 정보 발생지, 경로 정보의 갱신 시간
- **경로 설정 프로토콜**

IGP	· 하나의 자율 시스템(AS) 내의 라우팅에 사용되는 프로토콜 · RIP : 소규모 동종의 네트워크 내에서 효율적인 방법으로, 최대 Hop 수를 16으로 제한 · OSPF : 대규모 네트워크에서 많이 사용되는 프로토콜로, 라우팅 정보에 변화가 있을 때에, 변화된 정보만 네트워크 내의 모든 호스트에 알림
EGP	자율 시스템(AS) 간의 라우팅, 즉 게이트웨이 간의 라우팅에 사용되는 프로토콜
BGP	· 자율 시스템(AS) 간의 라우팅 프로토콜로, EGP의 단점을 보완하기 위해 만들어짐 · 임의의 호스트의 라우팅 정보에 변화가 있을 때에, 변화된 정보를 교환함

· 경로 설정 방식

고정 경로 제어(Static Routing) = 착각 부호 방식	네트워크 내의 모든 쌍에 대해서 경로를 미리 정해 놓은 방식
적응 경로 제어(Adaptive Routing)	전송 경로를 동적으로 결정하는 방식
범람 경로 제어(Flooding)	네트워크 정보를 요구하지 않고, 송·수신측 사이에 존재하는 모든 경로로 패킷을 전송
임의의 경로 제어(Random Routing)	인접하는 교환기 중 하나를 임의로 선택하여 전송하는 방식

핵심 262 | 흐름 제어(Flow Control)

- 통신망 내의 원활한 흐름을 위해 송수신측 사이에 전송되는 패킷의 양이나 속도를 규제하는 기능

정지-대기(Stop-and-Wait)	· 수신측의 확인 신호(ACK)를 받은 후에 다음 패킷을 전송하는 방식 · 한 번에 하나의 패킷만을 전송할 수 있음
슬라이딩 윈도우(Sliding Window)	· 수신측의 확인 신호를 받지 않더라도 미리 정해진 패킷의 수만큼 연속적으로 전송하는 방식 · 한 번에 여러 개의 패킷을 전송할 수 있어 전송 효율이 좋음 · 윈도우 크기(Window Size) : 수신측의 확인 신호(Ack) 없이도 전송할 수 있는 패킷의 개수로, 상황에 따라 변함

핵심 263 | 망(Network)의 구성 형태

성형(Star, =중앙 집중형)

- 중앙에 중앙 컴퓨터가 있고, 이를 중심으로 단말기들이 연결되는 중앙 집중식의 네트워크 구성 형태
- 단말기의 추가와 제거가 쉬움
- 교환 노드의 수가 가장 적음

링형(Ring, =루프형)

- 컴퓨터와 단말기들을 서로 이웃하는 것끼리 포인트 투 포인트 방식으로 연결시킨 형태
- 데이터는 단방향 또는 양방향으로 전송할 수 있으며, 단방향 링의 경우 컴퓨터, 단말기, 통신 회선 중 어느 하나라도 고장나면 전체 통신망에 영향을 미침

버스형(Bus)

- 한 개의 통신 회선에 여러 대의 단말기가 연결되어 있는 형태
- 물리적 구조가 간단하고, 단말기의 추가와 제거가 용이

계층형(Tree, =분산형)

- 중앙 컴퓨터와 일정 지역의 단말기까지는 하나의 통신 회선으로 연결시키고, 이웃하는 단말기는 일정 지역 내에 설치된 중간 단말기로 부터 다시 연결시키는 형태
- 분산 처리 시스템을 구성하는 방식임

망형(Mesh)

- 모든 지점의 컴퓨터와 단말기를 서로 연결한 형태로, 노드의 연결성이 높음
- 많은 단말기로부터 많은 양의 통신을 필요로 하는 경우에 유리
- 공중 데이터 통신 네트워크에서 사용되며, 통신 회선의 총 경로가 가장 김
- 통신 회선 장애시 다른 경로를 통하여 데이터를 전송할 수 있음
- 모든 노드를 망형으로 연결할 때 필요한 회선 수(노드 = n)

$$: n(n-1)/2$$

핵심 264 | LAN(근거리 통신망)

- 광대역 통신망과는 달리 학교, 회사, 연구소 등 한 건물이나 일정 지역 내에서 컴퓨터나 단말기들을 고속 전송 회선으로 연결하여 프로그램 파일 또는 주변 장치를 공유할 수 있도록 한 네트워크
- 단일 기관의 소유, 제한된 지역 내의 통신임
- 광대역 전송 매체의 사용으로 고속 통신이 가능
- 경로 선택이 필요 없고, 오류 발생률이 낮음
- 전송 매체로 꼬임선, 동축 케이블, 광섬유 케이블 등을 사용
- 망의 구성 형태에 따라서 스타형, 버스형, 링형, 트리형으로 분류
- 물리 계층과 데이터 링크 계층으로 나뉨

물리 계층	OSI 7 계층의 물리 계층과 동일한 기능을 제공
데이터 링크 계층	<ul style="list-style-type: none"> • 매체 접근 제어(MAC) 계층과 논리 링크 제어(LLC) 계층으로 나뉨 • 매체 접근 제어(MAC) 방식의 종류 : CSMA, CSMA/CD, 토큰 버스, 토큰 링

IEEE 802의 주요 표준 규격

802.3	CSMA/CD 방식
802.4	토큰 버스 방식
802.5	토큰 링 방식
802.11	무선 LAN

핵심 265 | CSMA/CD 방식

- 통신 회선이 사용 중이면 일정 시간동안 대기하고, 통신 회선 상에 데이터가 없을 때에만 데이터를 송신하며, 송신 중에도 전송로의 상태를 계속 감시함
- 버스형 또는 성형 LAN에 가장 일반적으로 이용됨
- 일정 길이 이하의 데이터를 송신할 경우 충돌을 검출할 수 없음
- 전송량이 적을 때 매우 효율적이고 신뢰성이 높음
- 전송량이 많아지면 채널의 이용률이 떨어지고 전송 지연 시간이 급격히 증가함
- 충돌 발생시 다른 노드에서는 데이터를 전송할 수 없으며, 지연 시

간을 예측하기 어려움

- CSMA/CD 방식을 사용하는 LAN을 이더넷(Ethernet)이라고 함
- **10 BASE T의 의미** : 10은 전송 속도가 10Mbps, BASE는 베이스 밴드 방식, T는 전송 매체로 꼬임선(Twisted Pair)을 사용

핵심 266 | VAN(부가 가치 통신망)

- 공중 통신 사업자로부터 통신 회선을 임대하여 하나의 사설망을 구축하고 이를 통해 정보의 축적, 가공, 변환 처리 등 첨가한 후 불특정 다수를 대상으로 서비스를 제공하는 통신망
- **계층 구조** : 정보 처리 계층, 통신 처리 계층, 네트워크 계층, 기본 통신 계층
- **기능** : 전송 기능, 교환 기능, 통신 처리 기능, 정보 처리 기능
- 통신 처리 기능은 축적 교환 기능과 변환 기능으로 나뉨

축적 교환 기능	전자 사서함, 데이터 교환, 동보 통신, 정시 수집, 정시 배달
변환 기능	속도 변환, 프로토콜 변환, 코드 변환, 데이터 형식(Format) 변환, 미디어 변환

핵심 267 | ISDN(종합 정보 통신망)

- 음성, 문자, 화상 등의 다양한 통신 서비스를 하나의 디지털 통신망을 근간으로 종합적으로 제공할 수 있도록 통합한 것
- 통신 방식 및 전송로가 모두 디지털 방식임
- 단일 통신망으로 음성, 데이터, 영상 등의 다양한 서비스를 제공
- 고속 통신이 가능하며, 확장성과 재배치성이 좋음
- 64Kbps 1회선 교환 서비스를 기본으로 함
- **ISDN의 통신 서비스**

베어러 서비스	단말기가 전송하는 정보를 변형 없이 그대로 전달만하는 서비스
텔레 서비스	<ul style="list-style-type: none"> • 베어러 서비스를 기본으로 고도의 기능을 부가하여 제공하는 서비스 • 실제로 단말을 조작하고 통신하는 이용자측에서 본 서비스
부가 서비스	발신 번호 표시, 수신자 부담, 통화 대기 등

핵심 268 | ISDN의 구조

주요 채널의 종류

채널	채널 속도	용도
B	64Kbps	<ul style="list-style-type: none"> • 디지털 정보용 채널 • 사용자의 정보를 전송하기 위한 채널 • PCM화된 디지털 음성이나 회선 교환에 의한 제어 신호, 패킷 교환에 의한 정보의 전송에 이용
D	16Kbps 64Kbps	<ul style="list-style-type: none"> • 디지털 신호용 채널 • 16kbps 이하 저속의 패킷 교환에 의한 정보 전송을 위해 이용
H	H0	384Kbps
	H11	1,536Kbps
	H12	1,920Kbps

- **기본 속도 인터페이스(BRI)** : $2B + D + \text{오버헤드} = 2 \times 64 + 16 + 48 = 192\text{Kbps}$

핵심 269 | 기능 그룹과 참조점

- **망 중단 장치(NT) :** 망 관련 장비
- **터미널 장비(TE) :** 사용자 관련 장비

NT1	· 가입자 접속망 중단 장치, 선로의 중단점 · 1 계층 선로 유지 보수 및 성능 감시, 다중화 · 비트 동기(Timing)
NT2	· 가입자 구내 교환 장치 · 2~3계층의 프로토콜 처리 및 다중화 · 교환, 집중화, 유지 보수 기능
TE1	ISDN 접속 기능이 없는 단말기
TA	· TE2 단말기를 ISDN에 접속하는 역할 · 프로토콜 및 속도 변환 기능
LE	상대방 ISDN 교환기와 연결을 제공하는 ISDN용 교환기

- **참조점 :** ISDN을 구성하는 각 요소 간의 인터페이스를 구분하는 기능을 하는 것으로, 기준점, 접속점, 분계점이라고도 함. 정보 통신 망 상호간을 연결할 때 시설, 운영 및 유지, 보수의 책임 한계를 구분하기 위한 접속점이 됨

핵심 270 | 인터넷(Internet)

- **TCP/IP** 프로토콜을 기반으로 하여 전세계 수많은 컴퓨터와 네트워크들이 연결된 광범위한 컴퓨터 통신망
- **ARPANET**에서 시작되었으며, 유닉스 운영체제를 기반으로 함
- 인터넷에 연결된 모든 컴퓨터는 고유한 IP 주소를 가짐
- **인터넷 서비스 :** TCP/IP의 응용 계층에서 제공

WWW	· 텍스트, 그림, 동영상, 음성 등 인터넷에 존재하는 다양한 정보를 거미줄처럼 연결해 놓은 종합 정보 서비스 · HTTP 프로토콜을 사용하는 하이퍼텍스트 기반으로 되어 있음
전자우편	· 인터넷을 통해 다른 사람과 편지뿐만 아니라 그림, 동영상 등 다양한 형식의 데이터를 주고받을 수 있도록 해주는 서비스 · 전자 우편에 사용되는 프로토콜 : SMTP, POP3, MIME
FTP	컴퓨터와 컴퓨터 또는 컴퓨터와 인터넷 사이에서 파일을 주고받을 수 있도록 하는 원격 파일 전송 프로토콜
Telnet	· 멀리 떨어져 있는 컴퓨터에 접속하여 자신의 컴퓨터처럼 사용할 수 있도록 해주는 서비스 · 프로그램을 실행하는 등 시스템 관리 작업을 할 수 있는 가상 터미널(Virtual Terminal) 기능을 수행
Usenet	분야별로 공동의 관심을 가진 인터넷 사용자들이 서로의 의견을 주고받을 수 있게 하는 서비스

핵심 271 | 인터넷의 주소 체계

- **IP 주소 :** 인터넷에 연결된 모든 컴퓨터의 자원을 구분하기 위한 고유한 주소로, 숫자로 8비트씩 4부분, 총 32비트로 구성되며, A ~ E 클래스까지 총 5단계로 나뉨
- **서브넷 마스크 :** 4바이트의 IP 주소 중에 네트워크 주소와 호스트 주소를 구분하기 위한 비트
- **IPv6 :** IPv4의 주소 부족 문제를 해결하기 위해 개발된 것으로, 16비트씩 8부분, 총 128비트로 구성되며, 각 부분은 16진수로 표현하고, 콜론으로 구분함
- **도메인 네임 :** 숫자로 된 IP 주소를 사람이 이해하기 쉬운 문자 형태로 표현한 것으로, 호스트 컴퓨터명, 소속 기관 이름, 소속 기관의 종류, 소속 국가명 순으로 구성
- **DNS :** 문자로 된 도메인 네임을 컴퓨터가 이해할 수 있는 IP 주소로 변환

핵심 272 | 네트워크 관련 장비

- **허브(Hub) :** 한 사무실이나 가까운 거리의 컴퓨터들을 연결하는 장치로, 각 회선을 통합적으로 관리하며, 신호 증폭 기능을 하는 리피터의 역할도 포함함
- **리피터(Repeater) :** 물리 계층의 장비로, 전송되는 신호를 재생해줌
- **브리지(Bridge) :** 데이터 링크 계층의 장비로, LAN과 LAN을 연결하거나 LAN 안에서의 컴퓨터 그룹을 연결
- **라우터(Router) :** 네트워크 계층의 장비로, LAN과 LAN의 연결 및 경로 선택, 서로 다른 LAN이나 LAN과 WAN의 연결
- **게이트웨이(Gateway) :** 프로토콜 구조가 전혀 다른 네트워크의 연결을 수행하는 장비로, 세션 계층, 표현 계층, 응용 계층간을 연결하여 데이터 형식 변환, 주소 변환, 프로토콜 변환 등을 수행

핵심 273 | 통신 프로토콜

- **정의 :** 서로 다른 기기들 간의 데이터 교환을 원활하게 수행할 수 있도록 표준화시켜 놓은 통신 규약
- **기본 요소 :** 구문(Syntax), 의미(Semantics), 시간(Timing)
- **기능 :** 단편화, 재결합, 캡슐화, 흐름 제어, 오류 제어, 동기화, 순서 제어, 주소 지정, 다중화, 경로 제어, 전송 서비스(우선순위, 서비스 등급, 보안성)
- **캡슐화 할 때 제어 정보에 포함되는 것 :** 송수신지 주소, 오류 검출 코드, 프로토콜 제어 정보

핵심 274 | OSI 참조 모델

- 다른 시스템 간의 원활한 통신을 위해 ISO(국제 표준화 기구)에서 제안한 통신 규약(Protocol)
- **OSI 7 계층 :** 하위 계층(물리 계층 → 데이터 링크 계층 → 네트워크 계층) → 상위 계층(전송 계층 → 세션 계층 → 표현 계층 → 응용 계층)

물리 계층 (Physical Layer)	전송에 필요한 두 장치 간의 실제 접속과 절단 등 기계적, 전기적, 기능적, 절차적 특성을 정의
데이터 링크 계층 (Data Link Layer)	· 두 개의 인접한 개방 시스템들 간에 신뢰성 있고 효율적인 정보 전송을 할 수 있도록 함 · 흐름 제어, 프레임 동기, 오류 제어, 순서 제어
네트워크 계층 (Network Layer, 망 계층)	· 개방 시스템들 간의 네트워크 연결 관리(네트워크 연결을 설정, 유지, 해제), 데이터의 교환 및 중계 · 경로 설정(Routing), 트래픽 제어, 패킷 정보 전송
전송 계층 (Transport Layer)	· 종단 시스템(End-to-End) 간에 투명한 데이터 전송을 가능하게 함 · 전송 연결 설정, 데이터 전송, 전송 연결 해제 기능 · 주소 설정, 다중화, 에러 제어, 흐름 제어
세션 계층 (Session Layer)	· 송·수신측 간의 관련성을 유지하고 대화 제어를 담당 · 대화(회화) 구성 및 동기 제어, 데이터 교환 관리 기능 · 체크점(=동기점) : 오류가 있는 데이터의 회복을 위해 사용하는 것으로 소동기점과 대동기점이 있음
표현 계층 (Presentation Layer)	· 응용 계층으로부터 받은 데이터를 세션 계층에 맞게, 세션 계층에서 받은 데이터는 응용 계층에 맞게 변환하는 기능 · 코드 변환, 데이터 암호화, 데이터 압축, 구문 검색, 정보 형식(포맷) 변환, 문맥 관리 기능
응용 계층 (Application Layer)	사용자(응용 프로그램)가 OSI 환경에 접근할 수 있도록 서비스를 제공

핵심 275 | X.25

- 패킷 교환망을 통한 DCE와 DTE 간의 인터페이스를 제공
- ITU-T에서 제정한 국제 표준 프로토콜로, 우수한 호환성을 가짐
- 신뢰성과 효율성이 높고, 전송 품질이 우수
- X.25의 계층 구조 : 물리 계층, 프레임 계층, 패킷 계층
- LAPB : HDLC의 원리를 이용한 비트 중심의 프로토콜로, X.25의 2계층에서 사용
- 프레임 릴레이 : 기존의 X.25가 갖는 오버 헤드를 제거하여 고속 데이터 통신에 적합하도록 개선한 프로토콜
- 패킷 교환을 위한 수행 절차 : 호 설정 → 데이터 전송 → 호 해제

핵심 276 | TCP/IP

- UNIX의 기본 프로토콜로 사용되었고, 현재 인터넷 표준 프로토콜임
- TCP/IP의 계층

응용 계층	· 응용 프로그램 간의 데이터 송·수신 제공 · TELNET, FTP, SMTP, HTTP 등
전송 계층	· 호스트들 간의 신뢰성 있는 통신 제공 · TCP, UDP
인터넷 계층	· 데이터 전송을 위한 주소 지정, 경로 설정 · IP, ICMP, IGMP, ARP, RARP 등
링크 계층	· 실제 데이터(프레임)를 송·수신하는 역할 · Ethernet, IEEE 802, HDLC, X.25, RS-232C 등

▪주요 프로토콜

TCP	· OSI 7 계층의 트랜스포트 계층에 해당 · 신뢰성 있는 연결형 서비스 제공 · 패킷의 다중화, 재순서화, 오류 제어, 흐름 제어 기능
IP	· OSI 7 계층의 네트워크 계층에 해당 · 데이터그램을 기반으로 하는 비연결형 서비스 제공 · 패킷의 분해/조립, 주소 지정, 경로 선택 기능
ICMP	IP와 조합하여 통신 중에 발생하는 오류의 처리와 전송 경로 변경 등을 위한 제어 메시지를 관리하는 역할을 함
ARP	호스트의 IP 주소를 호스트와 연결된 네트워크 접속 장치의 물리적 주소(MAC Address)로 변환
RARP	물리적 주소를 IP 주소로 변환