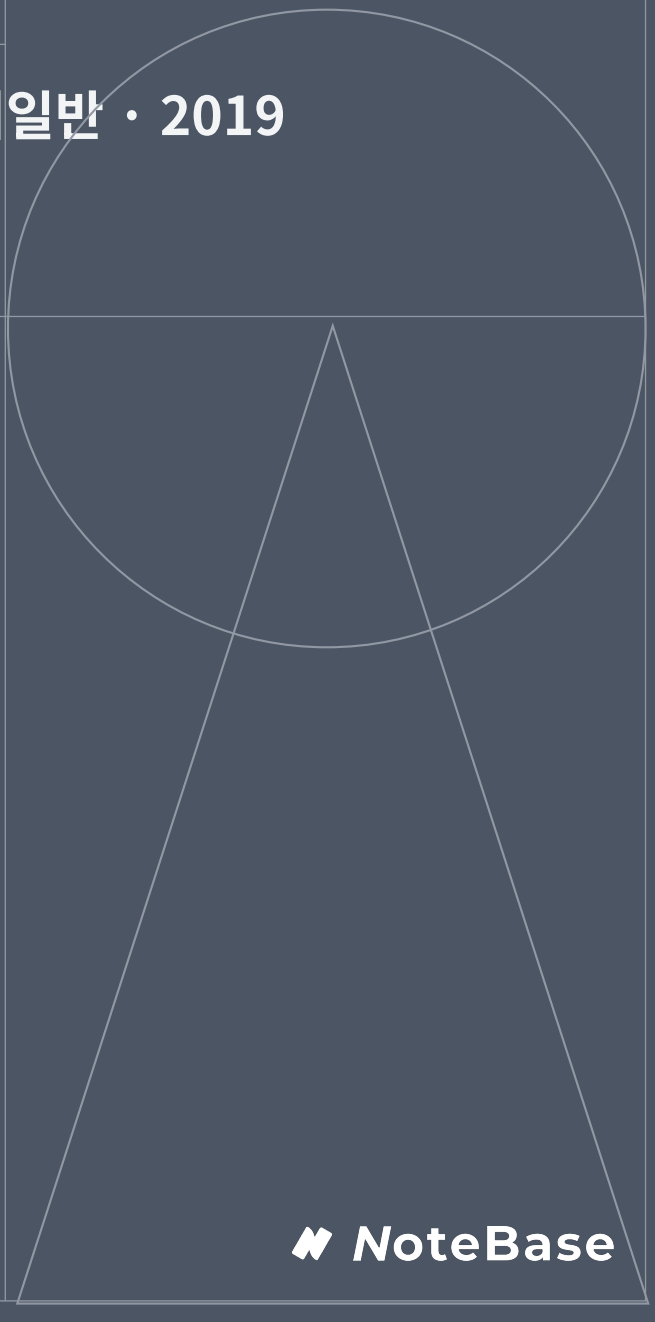
기출OX 해설집

공무원/국가직/9급 · 컴퓨터일반 · 2019



NoteBase

기출OX 해설집

공무원/국가직/9급 · 컴퓨터일반 · 2019

공무원/국가직/9급 · 컴퓨터일반 · 2019

문 1. CPU 내부 레지스터로 옳지 않은 것은?

① 누산기(accumulator)

② 캐시 메모리(cache memory)

③ 프로그램 카운터(program counter)

④ 메모리 버퍼 레지스터(memory buffer register)

정답 2번

간단한 해설

캐시 메모리는 주기억장치의 명령어·데이터 일부를 보관하여 접근 시간을 줄이는 **고속 기억장치**이며, CPU 레지스터로 분류하지 않는다. CPU 내부에 내장되어 있더라도 레지스터와는 구별된다.

공통 해설

CPU 레지스터는 **명령 실행에 직접 필요한 값이나 주소**를 보관하는 작은 고속 저장장소이다. 누산기는 연산의 중간 결과를, 프로그램 카운터는 다음에 가져올 명령어의 주소를, 메모리 버퍼 레지스터는 메모리에서 읽거나 메모리에 쓸 데이터를 보관한다. 이들은 모두 명령 실행 과정의 값이나 주소를 직접 담는 레지스터에 해당한다.

캐시 메모리는 주기억장치의 명령어·데이터 일부를 복사해 두고, CPU가 필요한 내용을 여기서 찾으면 주기억장치까지 접근하지 않도록 하는 기억장치이다. **메모리 접근 시간을 줄이는 역할**을 하며, 명령 실행 상태를 직접 담는 레지스터와는 분류가 다르다. 따라서 CPU 내부 레지스터에 해당하지 않는 것은 캐시 메모리이다. 캐시가 CPU 내부에 구현되어 있다는 사실만으로 레지스터가 되는 것은 아니다.

NoteBase

문 2. 다음 전위(prefix) 표기식의 계산 결과는?

+ − 5 4 × 4 7

① −19

② 7

③ 28

④ 29

정답 4번

간단한 해설

전위 표기식에서는 연산자가 두 피연산자보다 앞에 온다. 주어진 식은 **5 − 4의 결과와 4 × 7의 결과를 더하는 구조**이므로 1 + 28 = 29가 된다.

계산 과정

전위 표기식은 연산자를 먼저 쓰고 그 뒤에 피연산자를 쓰는 방식이다. 피연산자 자리에는 수뿐 아니라 하나의 식도 올 수 있다. 주어진 식의 맨 앞 '+'는 **'− 5 4'와 '× 4 7'의 계산 결과**를 더하라는 뜻이다.

뺄셈 부분: '− 5 4'에서 첫 번째 피연산자는 5, 두 번째 피연산자는 4이므로 5 − 4 = 1이다. 뺄셈은 **첫 번째 피연산자에서 두 번째 피연산자를 빼므로** 순서를 바꾸면 안 된다.

곱셈 부분: '× 4 7'은 두 피연산자 4와 7을 곱하므로 4 × 7 = 28이다.

전체 계산: 맨 앞 '+'에 두 부분의 계산 결과 1과 28을 적용하면 1 + 28 = 29이다. 이는 ④의 29에 해당한다.

NoteBase

문 3. 사진이나 동영상 등의 디지털 콘텐츠에 저작권자나 판매자 정보를 삽입하여 원본의 출처 정보를 제공하는 기술은?

① 디지털 사이니지
② 디지털 워터마킹
③ 디지털 핑거프린팅
④ 콘텐츠 필터링

정답 2번

간단한 해설

디지털 워터마킹은 **저작권자나 판매자 정보를 콘텐츠에 삽입**하여 원본의 출처를 확인할 수 있게 한다.

공통 해설

디지털 워터마킹은 사진·동영상 등의 데이터에 **출처를 나타내는 식별 정보**를 삽입하는 기술이다. 콘텐츠에서 삽입된 정보를 검출하면 해당 콘텐츠의 저작권자나 판매자를 식별하는 근거로 사용할 수 있다. 문항에서 요구하는 것은 이처럼 콘텐츠 자체에 담긴 정보를 통해 원본의 출처를 알려 주는 기능이다.

NoteBase

문 4. 1K × 4bit RAM 칩을 사용하여 8K × 16bit 기억장치 모듈을 설계할 때 필요한 RAM 칩의 최소 개수는?

① 4개

② 8개

③ 16개

④ 32개

정답 4번

간단한 해설

칩 4개를 병렬로 연결해 **데이터 폭을 4bit에서 16bit로** 늘리고, 이 묶음을 8개 사용해 저장 위치 수를 1K개에서 8K개로 늘린다. 따라서 필요한 칩은 4 × 8 = 32개이다.

계산 과정

1K × 4bit RAM 칩은 저장 위치가 1K개이고 각 위치에 4bit를 저장한다는 뜻이다. 목표 모듈은 저장 위치 8K개와 위치당 16bit를 모두 갖추어야 하므로, 저장 위치 수와 데이터 폭을 각각 확장한다.

데이터 폭 확장: 한 주소에서 16bit를 읽고 쓰려면 4bit 칩을 $\frac{16}{4} = 4$개 병렬로 연결한다. 각 칩이 16bit 중 서로 다른 4bit를 맡으므로, 이 묶음의 구성은 **1K × 16bit**가 된다. 병렬 연결만으로 저장 위치 수가 늘어나는 것은 아니다.

저장 위치 수 확장: 1K × 16bit 묶음으로 8K개의 저장 위치를 만들려면 $\frac{8K}{1K} = 8$묶음이 필요하다. 각 묶음이 **서로 다른 1K개의 주소 영역**을 담당하도록 하고, 접근하는 주소에 따라 해당 묶음을 선택한다.

전체 칩 수: 묶음당 칩 4개가 필요하고 묶음은 8개이므로, 4 × 8 = 32개이다. 이 구성은 목표 데이터 폭과 저장 위치 수를 정확히 충족하므로 최소 칩 수는 32개이며, ④의 32개와 일치한다.

NoteBase

문 5. 프로세스와 스레드(thread)에 대한 설명으로 옳지 않은 것은?

① 하나의 스레드는 여러 프로세스에 포함될 수 있다.

② 스레드는 프로세스에서 제어를 분리한 실행단위이다.

③ 스레드는 같은 프로세스에 속한 다른 스레드와 코드를 공유한다.

④ 스레드는 프로그램 카운터를 독립적으로 가진다.

정답 1번

간단한 해설

하나의 스레드는 **하나의 프로세스에 속하며**, 여러 프로세스에 동시에 포함되지 않는다. 하나의 프로세스가 여러 스레드를 포함할 수 있다.

선지별 해설

① X: 스레드는 프로세스 내부의 실행 흐름이므로 **하나의 프로세스에 속한다**. '하나의 스레드가 여러 프로세스에 포함된다'가 아니라, '하나의 프로세스가 여러 스레드를 포함할 수 있다'로 고쳐야 한다.

② O: 프로세스가 메모리와 같은 자원을 보유하는 단위라면, 스레드는 그 자원을 이용하여 명령을 실행하는 단위이다. 프로세스에서 **명령을 실행하는 제어 흐름을 분리**한 것이 스레드이므로, 한 프로세스 안에 여러 실행 흐름을 둘 수 있다.

③ O: 같은 프로세스의 스레드들은 **프로세스의 코드 영역을 공유**한다. 각 스레드는 공유된 코드의 서로 다른 부분을 실행할 수도 있으므로, 코드 공유가 모든 스레드의 실행 위치까지 같다는 뜻은 아니다.

④ O: 프로그램 카운터는 다음에 실행할 명령어의 주소를 나타낸다. 스레드마다 실행 중인 위치가 다를 수 있으므로 **프로그램 카운터를 각각 독립적으로 유지**한다. 실행이 중단되었다가 재개될 때도 각 스레드는 자신의 실행 위치를 기준으로 이어서 실행한다.

NoteBase

문 6. 보이스 코드 정규형(BCNF: Boyce-Codd Normal Form)을 만족하기 위한 조건에 해당하지 않는 것은?

① 조인(join) 종속성이 없어야 한다.

② 모든 속성 값이 원자 값(atomic value)을 가져야 한다.

③ 이행적 함수 종속성이 없어야 한다.

④ 기본 키가 아닌 속성이 기본 키에 완전 함수 종속적이어야 한다.

정답 1번

간단한 해설

조인 종속성을 없애야 한다는 것은 **BCNF의 필수 조건이 아니다**. BCNF는 함수 종속성을 기준으로 하며, 조인 종속성에 관한 추가 조건은 **제5정규형**에서 다룬다.

선지별 해설

BCNF에서는 비자명한 함수 종속성의 결정자가 모두 슈퍼키여야 한다. 비자명한 함수 종속성은 결정자에 포함되지 않은 속성을 결정하는 관계이며, 슈퍼키는 각 행을 유일하게 식별할 수 있는 속성 집합이다. BCNF는 제3정규형보다 엄격하므로 제1·제2·제3정규형의 조건도 만족한다.

① X: 조인 종속성은 릴레이션을 여러 부분으로 나눈 뒤 조인하여 원래 릴레이션을 복원할 수 있다는 제약이다. **BCNF는 조인 종속성의 제거를 요구하지 않는다**. 이는 제5정규형과 관련된 설명으로, 엄밀히는 후보키로부터 유도되지 않는 비자명한 조인 종속성이 없어야 한다는 조건이다.

② O: 원자 값은 해당 도메인에서 더 나누어 취급하지 않는 하나의 값이다. 모든 속성 값이 원자 값이어야 한다는 것은 **제1정규형의 조건**이며, BCNF도 이 조건을 만족해야 한다.

③ O: 여기서 이행적 함수 종속성은 기본 키가 비키 속성을 결정하고, 그 비키 속성이 다시 다른 비키 속성을 결정하는 관계를 뜻한다. 예를 들어 사원번호가 부서번호를 결정하고 부서번호가 부서명을 결정하면, 사원번호는 부서번호를 거쳐 부서명을 결정한다. 이때 부서번호가 슈퍼키가 아니라면 BCNF 조건을 위반한다. 이러한 **비키 속성을 거치는 이행적 함수 종속성의 제거**는 제3정규형의 조건이며 BCNF에서도 요구된다.

④ O: 완전 함수 종속이란 기본 키 전체로는 해당 속성을 결정할 수 있지만, **기본 키의 일부만으로는 결정할 수 없는** 관계이다. 예를 들어 기본 키가 학번과 과목번호의 조합일 때, 성적이 그 조합 전체에 의해 결정되고 학번이나 과목번호만으로는 결정되지 않는 경우이다. 부분 함수 종속성을 배제하는 제2정규형의 조건으로, BCNF도 이를 만족해야 한다.

NoteBase

문 7. UDP(User Datagram Protocol)에 대한 설명으로 옳은 것만을 모두 고르면?

ㄱ. 연결 설정이 없다.
ㄴ. 오류검사에 체크섬을 사용한다.
ㄷ. 출발지 포트 번호와 목적지 포트 번호를 포함한다.
ㄹ. 혼잡제어 메커니즘을 이용하여 링크가 과도하게 혼잡해지는 것을 방지한다.

① ㄱ, ㄴ
② ㄱ, ㄷ
③ ㄱ, ㄴ, ㄷ
④ ㄴ, ㄷ, ㄹ

정답 3번

간단한 해설

UDP 자체에는 **혼잡제어 기능이 없다**. 네트워크 혼잡에 맞추어 전송량을 조절하는 것은 TCP의 대표적인 기능이다.

보기별 해설

ㄱ. O: UDP는 **연결을 미리 설정하지 않는** 비연결형 전송 계층 프로토콜이다. TCP와 달리 통신 상대와 연결 설정 절차를 거치지 않고 데이터그램을 전송한다.

ㄴ. O: UDP는 **체크섬으로 오류를 검출**한다. 송신 측에서 계산한 체크섬을 수신 측에서 검사하여 전송 중 데이터가 손상되었는지 확인하는 데 사용한다. 다만 오류를 검출하는 것과 손상된 데이터를 복구하거나 재전송하는 것은 별개이며, UDP 자체는 오류 복구나 재전송을 제공하지 않는다.

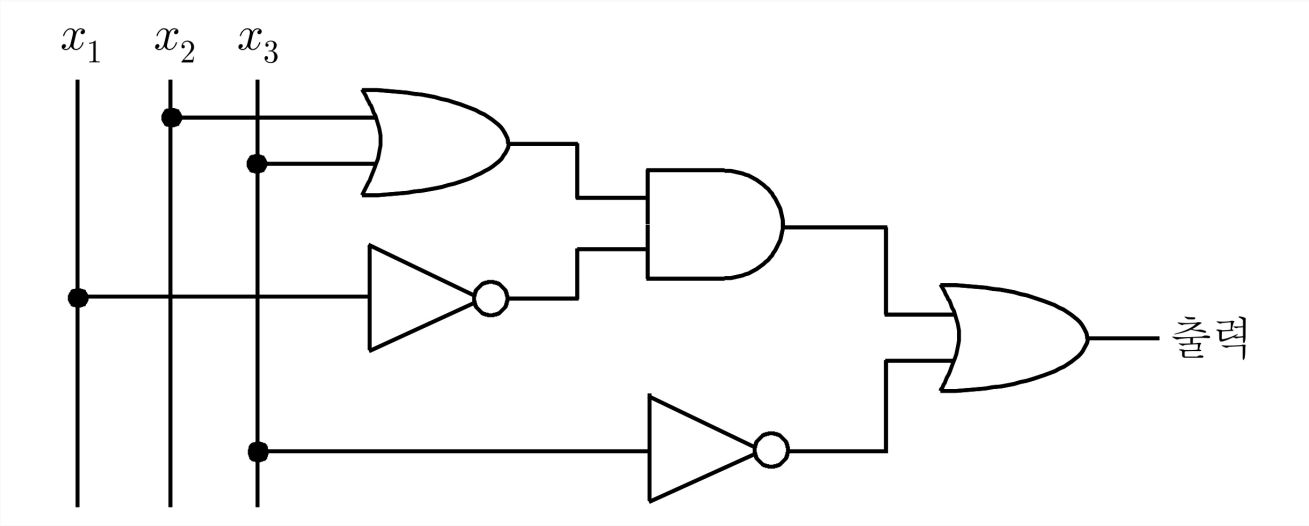
ㄷ. O: UDP 헤더에는 **출발지 포트 번호와 목적지 포트 번호** 필드가 있다. IP 주소가 통신할 호스트를 식별한다면, 포트 번호는 그 호스트 안에서 통신하는 응용 프로그램을 구분하는 데 사용된다.

ㄹ. X: UDP 자체에는 **혼잡에 따라 전송량을 조절하는 기능이 없다**. 따라서 혼잡제어 메커니즘으로 링크가 과도하게 혼잡해지는 것을 방지한다는 설명은 UDP의 기능에 해당하지 않는다. TCP는 네트워크 혼잡 상태에 맞추어 전송량을 조절하는 혼잡제어 기능을 제공한다.

옳은 보기는 ㄱ, ㄴ, ㄷ이며, 이를 모두 포함한 조합은 ③이다.

NoteBase

문 8. 다음 논리 회로의 출력과 동일한 것은?



① $x_1 + x_3'$

② $x_1' + x_3$

③ $x_1' + x_3'$

④ $x_2' + x_3'$

정답 3번

간단한 해설

마지막 OR 게이트에는 x_3'가 직접 들어가므로, $x_3 = 0$이면 출력은 항상 1이다. $x_3 = 1$이면 위쪽 OR 게이트의 출력이 1이 되어 최종 출력은 x_1'이다. 따라서 전체 출력은 $x_1' + x_3'$이며 x_2**의 값에 영향을 받지 않는다**.

NoteBase

계산 과정

논리식에서 +는 OR, ·는 AND, 프라임(')은 NOT을 뜻한다. OR는 입력 중 하나라도 1이면 1을, AND는 입력이 모두 1일 때만 1을 출력하며, NOT은 0과 1을 뒤집는다. 회로의 최종 출력을 F로 둔다.

윗부분의 출력: 왼쪽 위 OR 게이트에는 x_2와 x_3가 입력되므로 출력은 $x_2 + x_3$이다. 가운데 NOT 게이트는 x_1을 반전하여 x_1'을 출력한다. 이 두 신호가 AND 게이트로 들어가므로 위쪽 경로의 출력은 $x_1' \cdot (x_2 + x_3)$이다.

마지막 OR 게이트: 아래쪽 NOT 게이트는 x_3를 반전하여 x_3'를 출력한다. 마지막 게이트는 위쪽 경로와 아래쪽 경로를 OR로 결합하므로 $F = x_1' \cdot (x_2 + x_3) + x_3'$이다. **아래쪽 경로는 AND 게이트를 거치지 않고** 마지막 OR 게이트에 직접 연결된다.

분배법칙과 보수 관계: 괄호를 전개하면 $F = x_1' \cdot x_2 + x_1' \cdot x_3 + x_3'$이다. 이때 $x_1' \cdot x_3 + x_3' = (x_1' + x_3') \cdot (x_3 + x_3')$이고, $x_3 + x_3' = 1$이므로 이 부분은 $x_1' + x_3'$로 줄어든다. 따라서 $F = x_1' \cdot x_2 + x_1' + x_3'$이다.

흡수법칙과 선택지 연결: **흡수법칙** $A + A \cdot B = A$를 적용하면 $x_1' + x_1' \cdot x_2 = x_1'$이다. $x_1' = 1$이면 OR의 출력이 이미 1이고, $x_1' = 0$이면 곱항도 0이므로 x_2는 결과를 바꾸지 못한다. 최종 출력은 $F = x_1' + x_3'$로, ③의 식과 같다.

NoteBase

문 9. 다음 Java 프로그램의 출력 결과는?

```
class ClassP {
  int func1(int a, int b) {
      return (a+b);
  }
  int func2(int a, int b) {
      return (a-b);
  }
  int func3(int a, int b) {
      return (a*b);
  }
}
public class ClassA extends ClassP {
  int func1(int a, int b) {
      return (a%b);
  }
  double func2(double a, double b) {
      return (a*b);
  }
  int func3(int a, int b) {
      return (a/b);
  }
  public static void main(String[] args) {
      ClassP P = new ClassA();
      System.out.print(P.func1(5, 2) + ", "
          + P.func2(5, 2) + ", " + P.func3(5, 2));
  }
}
```

① 1, 3, 2

② 1, 3, 2.5

③ 1, 10.0, 2.5

④ 7, 3, 10

정답 1번

간단한 해설

func1과 func3은 **실제 객체인 ClassA에서 재정의한 메서드**가 실행된다. func2(double, double)는 오버라이딩이 아니므로 이 호출에서는 **ClassP의 func2(int, int)**가 실행된다. func3의 나눗셈은 두 피연산자가 int이므로 소수 부분을 버린다.

NoteBase

계산 과정

'ClassP P = new ClassA();'에서 P의 선언형은 ClassP이고, 실제로 생성한 객체의 클래스는 ClassA이다. 호출할 메서드의 이름과 매개변수형은 컴파일할 때 결정하며, 그 메서드가 자식 클래스에서 재정의되었다면 실행할 때 **실제 객체의 재정의된 메서드**를 사용한다.

P.func1(5, 2): ClassA의 func1(int, int)은 부모와 이름 및 매개변수형이 같아 오버라이딩한 메서드이다. 따라서 부모의 덧셈 대신 자식의 **나머지 연산 a % b**를 실행한다. 5 % 2 = 1이므로 첫 번째 출력값은 1이다.

P.func2(5, 2): ClassA의 func2(double, double)는 부모의 func2(int, int)과 매개변수형이 다른 **오버로딩**이며, 부모 메서드를 재정의한 것이 아니다. P의 선언형인 ClassP와 int 인수 5, 2를 기준으로 func2(int, int)이 선택된다. 이 메서드는 ClassA에서 재정의되지 않았으므로 부모의 뺄셈을 실행하여 5 - 2 = 3을 반환한다. 두 번째 출력값은 3이다.

P.func3(5, 2): ClassA의 func3(int, int)은 부모 메서드를 오버라이딩했으므로 부모의 곱셈 대신 자식의 나눗셈 a / b를 실행한다. a와 b가 모두 int이므로 **정수 나눗셈**이 적용되어 소수 부분을 버린다. 따라서 5 / 2의 결과는 2이며, 세 번째 출력값은 2이다.

각 반환값을 문자열 ', '와 연결하므로 출력은 '1, 3, 2'이며, ①의 값과 일치한다.

NoteBase

문 10. IPv4에서 데이터 크기가 6,000 바이트인 데이터그램이 3개로 단편화(fragmentation)될 때, 단편화 오프셋(offset) 값으로 가능한 것만을 모두 고르면?

ㄱ. 0 ㄴ. 500 ㄷ. 800 ㄹ. 2,000

① ㄱ, ㄴ

② ㄷ, ㄹ

③ ㄱ, ㄴ, ㄷ

④ ㄴ, ㄷ, ㄹ

정답 1번

간단한 해설

IPv4 단편화 오프셋은 원본 데이터에서 단편이 시작하는 위치를 **8바이트 단위**로 나타낸다. 첫 단편은 0이고, 오프셋 500은 앞에 4,000바이트가 있다는 뜻이므로 6,000바이트 데이터 안에서 가능하다.

보기별 해설

단편화 오프셋은 원본 데이터의 시작점부터 해당 단편 앞까지의 데이터 바이트 수를 8로 나눈 값이다. IP 헤더 길이는 이 계산에 포함하지 않는다. 마지막이 아닌 단편의 데이터 길이는 8바이트의 배수이며, 각 단편의 시작 위치는 원본 데이터 6,000바이트의 범위 안에 있어야 한다.

ㄱ. O: 첫 단편 앞에는 데이터가 없으므로 **첫 단편의 오프셋은 0**이다. 세 단편으로 나누더라도 첫 단편의 값은 변하지 않는다.

ㄴ. O: 오프셋 500은 앞에 500 × 8 = 4,000바이트의 데이터가 있다는 뜻이다. 예를 들어 데이터를 2,000바이트씩 세 단편으로 나누면 세 번째 단편 앞에 4,000바이트가 놓이므로 $\frac{4,000}{8} = 500$이다. 따라서 **세 단편으로 나누는 조건에서도 가능하다**.

ㄷ. X: 오프셋 800은 앞에 800 × 8 = 6,400바이트의 데이터가 있다는 뜻이다. 이는 원본 데이터 6,000바이트를 넘으므로 불가능하다. **800은 바이트 위치가 아니라 8바이트 단위의 값**으로 해석해야 한다.

ㄹ. X: 오프셋 2,000은 앞에 2,000 × 8 = 16,000바이트의 데이터가 있다는 뜻이므로 원본 데이터 범위를 벗어난다. 앞의 예에서 한 단편의 데이터 길이가 2,000바이트여도 세 번째 단편의 오프셋은 **누적 데이터 길이를 8로 나눈 500**이다.

가능한 것은 ㄱ, ㄴ이며, 이 조합은 ①과 일치한다.

NoteBase

문 11. Go-Back-N 프로토콜에서 6번째 프레임까지 전송한 후 4번째 프레임에서 오류가 있음을 알았을 때, 재전송 대상이 되는 프레임의 개수는?

① 1개

② 2개

③ 3개

④ 6개

정답 3번

간단한 해설

Go-Back-N은 **오류가 발생한 프레임부터 그 뒤에 이미 전송한 프레임까지** 재전송한다. 따라서 4번째·5번째·6번째 프레임, 총 3개가 재전송 대상이다.

계산 과정

Go-Back-N에서는 수신 측이 프레임을 순서대로 받아들인다. 4번째 프레임에 오류가 있으면 이후의 5번째·6번째 프레임이 정상적으로 도착해도 받아들이지 않으므로, 송신 측은 **4번째 프레임부터 다시 전송**해야 한다.

이미 전송한 마지막 프레임은 6번째이고 재전송을 시작할 프레임은 4번째이다. 양 끝 프레임을 모두 포함하므로 재전송 개수는 6 − 4 + 1 = 3개이다. 여기서 +1은 오류가 발생한 4번째 프레임 자체도 세기 위한 것이다.

재전송 대상은 4번째·5번째·6번째 프레임이며, 계산한 3개는 ③의 값과 일치한다.

NoteBase

문 12. $0 \sim (64^{10} - 1)$**에 해당하는 정수를 이진코드로 표현하기 위해 필요한 최소 비트 수는?**

① 16비트

② 60비트

③ 63비트

④ 64비트

정답 2번

간단한 해설

$64^{10} = 2^{60}$이므로 주어진 범위의 정수는 모두 2^{60}개이다. 60비트로 이 개수만큼 서로 다른 값을 표현할 수 있으므로 필요한 최소 크기는 **60비트**이다.

계산 과정

비트 수를 n이라 하면 각 비트는 0 또는 1을 가지므로 서로 다른 이진코드는 2^n개이다. 이를 음이 아닌 정수에 대응시키면 **0을 포함하여** $2^n - 1$까지 표현할 수 있다.

문제의 범위는 0부터 $64^{10} - 1$까지이므로 정수의 개수는 64^{10}개이다. $64 = 2^6$이므로 $64^{10} = (2^6)^{10} = 2^{60}$이다. 따라서 필요한 이진코드의 개수는 2^{60}개이다.

60비트로 표현할 수 있는 최댓값은 $2^{60} - 1$이므로 주어진 범위 전체를 표현할 수 있다. 59비트로는 2^{59}개의 값만 표현할 수 있어 부족하며, 그보다 적은 비트 수도 부족하다. 따라서 필요한 **최소 비트 수는 60비트**로, ②의 값과 일치한다.

NoteBase

문 13. 의료용 심장 모니터링 시스템과 같이 정해진 짧은 시간 내에 응답해야 하는 시스템은?

① 다중프로그래밍 시스템

② 시분할 시스템

③ 실시간 시스템

④ 일괄 처리 시스템

정답 3번

간단한 해설

정해진 시간 안에 응답해야 한다는 **시간 제약**이 실시간 시스템의 핵심이다. 심장 모니터링처럼 응답 시점이 중요한 시스템에서는 처리 결과뿐 아니라 응답 기한을 지키는 것도 중요하다.

공통 해설

실시간 시스템은 입력이나 사건에 대해 **정해진 기한 안에 처리를 완료**해야 하는 시스템이다. 처리 결과가 정확하더라도 필요한 시점보다 늦게 나오면 목적을 달성하지 못할 수 있으므로, 결과의 정확성과 함께 시간 제약의 충족을 요구한다. 심장 모니터링에서 필요한 응답이 늦어지면 환자의 상태에 제때 대응하기 어려우므로, 문제의 '정해진 짧은 시간 내에 응답'은 실시간 시스템의 특성을 나타낸다.

NoteBase

문 14. FIFO 페이지 교체 알고리즘을 사용하는 가상메모리에서 프로세스 P가 다음과 같은 페이지 번호 순서대로 페이지에 접근할 때, 페이지 부재(page-fault) 발생 횟수는? (단, 프로세스 P가 사용하는 페이지 프레임은 총 4개이고, 빈 상태에서 시작한다)

1 2 3 4 5 2 1 1 6 7 5

① 6회

② 7회

③ 8회

④ 9회

정답 3번

간단한 해설

FIFO는 **먼저 적재된 페이지**를 교체하며, 적중해도 교체 순서는 바뀌지 않는다. 처음 네 페이지와 이후 5·1·6·7을 적재할 때 부재가 발생하므로 총 **8회**이다.

NoteBase

계산 과정

페이지 부재는 접근한 페이지가 메모리에 없을 때 발생한다. 빈 프레임이 있으면 그곳에 적재하고, 4개 프레임이 모두 차면 FIFO에 따라 가장 먼저 적재된 페이지를 내보낸다. 아래 대괄호는 프레임의 물리적 위치가 아니라, 메모리에 있는 페이지를 적재된 순서대로 나타낸 것으로 왼쪽이 다음 교체 대상이다.

1~4번째 접근(1, 2, 3, 4): 빈 상태에서 시작하므로 모두 부재이다. 네 페이지를 차례로 적재하면 [1, 2, 3, 4]가 되고, 누적 부재는 4회이다.

5번째 접근(5): 메모리에 없고 프레임이 가득 찼으므로 가장 먼저 적재된 1을 교체한다. [2, 3, 4, 5]가 되며, 누적 부재는 5회이다.

6번째 접근(2): 메모리에 있으므로 적중이다. FIFO는 **적중한 페이지의 교체 순서를 바꾸지 않으므로** [2, 3, 4, 5]를 유지하고, 누적 부재는 5회이다.

7번째 접근(1): 1은 앞서 교체되어 메모리에 없으므로 부재이다. 직전에 사용한 2도 적재 시점은 가장 오래되었으므로 2를 교체한다. [3, 4, 5, 1]이 되며, 누적 부재는 6회이다.

8번째 접근(1): 메모리에 있으므로 적중이다. [3, 4, 5, 1]을 유지하고, 누적 부재는 6회이다.

9번째 접근(6): 메모리에 없으므로 가장 먼저 적재된 3을 교체한다. [4, 5, 1, 6]이 되며, 누적 부재는 7회이다.

10번째 접근(7): 메모리에 없으므로 가장 먼저 적재된 4를 교체한다. [5, 1, 6, 7]이 되며, 누적 부재는 8회이다.

11번째 접근(5): 5는 메모리에 남아 있으므로 적중이다. [5, 1, 6, 7]을 유지하고, 누적 부재는 8회이다.

처음 네 페이지를 적재할 때 4회, 이후 페이지 5·1·6·7을 적재할 때 4회의 부재가 발생한다. 총 4 + 4 = 8회로, ③의 8회와 일치한다.

NoteBase

문 15. 재배치 가능한 형태의 기계어로 된 오브젝트 코드나 라이브러리 등을 입력받아 이를 묶어 실행 가능한 로드 모듈로 만드는 번역기는?

① 링커(linker)
② 어셈블러(assembler)
③ 컴파일러(compiler)
④ 프리프로세서(preprocessor)

정답 1번

간단한 해설

이미 기계어로 번역된 오브젝트 코드와 라이브러리의 필요한 부분을 **결합하여 실행 가능한 로드 모듈을 만드는** 프로그램이 링커이다.

공통 해설

링커(linker)는 재배치 가능한 오브젝트 코드들을 결합하고 필요한 라이브러리 루틴을 연결하여 실행 가능한 로드 모듈을 만든다. 이때 각 코드에 있는 이름과 주소 정보를 이용하여 **외부 참조를 해결하고 주소를 재배치**한다. 외부 참조 해결은 다른 모듈에 정의된 함수나 변수를 찾아 연결하는 것이고, 주소 재배치는 결합된 모듈의 배치에 맞게 주소를 조정하는 것이다.

예를 들어 한 오브젝트 코드에서 다른 오브젝트 코드에 정의된 함수를 호출한다면, 링커가 그 함수의 정의를 찾아 호출 위치와 연결한다. 따라서 문제의 '재배치 가능한 기계어 코드나 라이브러리를 입력받아 실행 가능한 로드 모듈로 묶는다'는 설명은 링커의 기능에 해당한다. 후보를 구분하는 기준은 소스 코드를 번역하는 것이 아니라 **이미 번역된 코드들을 연결한다**는 점이다.

NoteBase

문 16. 이메일, ERP, CRM 등 다양한 응용 프로그램을 서비스 형태로 제공하는 클라우드 서비스는?

① IaaS(Infrastructure as a Service)

② NaaS(Network as a Service)

③ PaaS(Platform as a Service)

④ SaaS(Software as a Service)

정답 4번

간단한 해설

이메일·ERP·CRM처럼 사용자가 바로 이용하는 **응용 프로그램을 서비스로 제공**하는 방식은 **SaaS**이다.

공통 해설

클라우드 서비스는 **사용자에게 무엇을 제공하는지**를 기준으로 구분한다. SaaS는 제공자가 운영하는 응용 프로그램의 기능을 사용자가 네트워크를 통해 이용하는 방식이다. 예를 들어 클라우드 이메일 서비스에서는 사용자가 이메일 서버와 프로그램을 직접 구축하지 않고 메일을 보내고 받는 기능을 이용한다. 문제의 ERP와 CRM도 업무에 필요한 응용 프로그램을 서비스로 이용한다는 점에서 같은 유형에 해당한다.

문제에서 제공하는 대상은 인프라(IaaS), 네트워크(NaaS), 개발·실행 플랫폼(PaaS)이 아니라 **응용 프로그램 자체**이므로 SaaS에 해당한다.

NoteBase

문 17. 다음 C 프로그램의 출력 결과는?

```
#include <stdio.h>
int main() {
    char msg[50] = "Hello World!! Good Luck!";
    int i = 2, number = 0;
    while (msg[i] != '!') {
        if (msg[i] == 'a' || msg[i] == 'e' || msg[i] == 'i'
            || msg[i] == 'o' || msg[i] == 'u')
            number++;
        i++;
    }
    printf("%d", number);
    return 0;
}
```

① 2

② 3

③ 5

④ 6

정답 1번

간단한 해설

i가 2에서 시작하므로 'Hello'의 e는 검사하지 않으며, **첫 번째 '!' 직전에 반복을 끝낸다**. 이 범위에서 소문자 모음은 o 두 개뿐이므로 number의 최종 값은 2이다.

NoteBase

계산 과정

초기화: 배열의 첨자는 0부터 시작하므로 msg[0]은 'H', msg[1]은 'e', msg[2]는 'l'이다. i = 2, number = 0이므로 세 번째 문자 'l'부터 검사하고, 모음을 센 횟수는 0에서 시작한다.

반복 범위: while의 조건 msg[i] != '!'는 현재 문자가 느낌표가 아닐 때만 반복문 본문을 실행한다는 뜻이다. i가 2부터 10까지 변하는 동안 검사하는 문자는 차례로 'l', 'l', 'o', 공백, 'W', 'o', 'r', 'l', 'd'이다. i가 11이 되면 msg[11]이 첫 번째 '!'이므로 반복이 끝난다. 따라서 **뒤의 '!! Good Luck!' 부분은 검사하지 않는다**.

모음 개수 계산: if의 ||는 나열된 조건 중 하나라도 참이면 전체 조건이 참이 되는 논리 OR 연산자이다. 현재 문자가 소문자 'a', 'e', 'i', 'o', 'u' 중 하나일 때만 number++로 number를 1 증가시킨다. msg[4]의 'o'에서 number가 0에서 1로, msg[7]의 'o'에서 1에서 2로 바뀐다. 다른 검사 문자는 모음이 아니므로 number를 바꾸지 않는다. **i++는 if문 밖에 있으므로** 모음 여부와 관계없이 매 반복마다 다음 문자로 이동한다.

출력: 반복이 끝난 뒤 number는 2이다. printf의 %d는 number를 십진 정수로 출력하므로 출력 결과는 2이며, ①의 값과 일치한다.

NoteBase

문 18. 마이크로프로세서에 관한 설명으로 옳은 것만을 모두 고르면?

ㄱ. 모든 명령어의 실행시간은 클럭 주기(clock period)보다 작다.
ㄴ. 클럭 속도는 에너지 절약이나 성능상의 이유로 일시적으로 변경할 수 있다.
ㄷ. 일반적으로 RISC는 CISC에 비해 명령어 수가 적고, 명령어 형식이 단순하다.

① ㄷ
② ㄱ, ㄴ
③ ㄱ, ㄷ
④ ㄴ, ㄷ

정답 4번

간단한 해설

클럭 속도는 고정된 값만 쓰는 것이 아니라, **전력 소비나 성능 요구에 따라 조절**할 수 있다. RISC는 일반적으로 CISC보다 **명령어 종류가 적고 형식이 단순**하도록 설계된다.

보기별 해설

ㄱ. X: 클럭 주기는 클럭 신호가 한 번 반복되는 데 걸리는 시간이다. 명령어 하나를 실행하는 데는 여러 클럭 주기가 필요할 수 있으므로, 모든 명령어의 실행시간이 클럭 주기보다 작다는 설명은 성립하지 않는다. 명령어 실행시간은 **필요한 클럭 사이클 수와 클럭 주기**에 따라 달라진다.

ㄴ. O: 마이크로프로세서는 동작 중 클럭 속도를 조절할 수 있다. 처리할 일이 적을 때는 속도를 낮춰 전력 소비를 줄이고, 높은 처리 성능이 필요할 때는 허용 범위에서 속도를 높인다. 따라서 클럭 속도를 **에너지 절약이나 성능상의 이유로 일시적으로 변경**할 수 있다.

ㄷ. O: RISC는 자주 쓰는 단순한 명령어를 중심으로 명령어 집합을 구성하고, 명령어 형식을 단순화하는 설계 방식이다. CISC는 상대적으로 다양한 명령어와 복잡한 형식을 지원하므로, 일반적인 비교에서 보기의 설명이 성립한다. 여기서 '명령어 수가 적다'는 **지원하는 명령어의 종류가 적다**는 뜻이며, 같은 프로그램을 실행할 때 수행하는 명령어 개수가 반드시 적다는 뜻은 아니다.

옳은 보기는 ㄴ, ㄷ이며, 이 조합은 ④에 해당한다.

NoteBase

문 19. 소프트웨어 규모를 예측하기 위한 기능점수(function point)를 산정할 때 고려하지 않는 것은?

① 내부논리파일(Internal Logical File)

② 외부입력(External Input)

③ 외부조회(External inQuiry)

④ 원시 코드 라인 수(Line Of Code)

정답 4번

간단한 해설

기능점수는 사용자에게 제공하는 **기능의 규모**를 측정한다. 원시 코드 라인 수는 프로그램을 구현한 **코드의 분량**이므로 기능점수 산정 항목에 포함되지 않는다.

공통 해설

기능점수는 사용자가 요구하는 데이터와 처리 기능을 기준으로 소프트웨어의 규모를 측정하는 방법이다. 기능을 내부논리파일(ILF), 외부인터페이스파일(EIF), 외부입력(EI), 외부출력(EO), 외부조회(EQ)의 다섯 유형으로 나누고, 각 유형의 수와 복잡도를 반영한다. 따라서 선지의 내부논리파일·외부입력·외부조회는 모두 **기능점수를 산정하는 기능 유형**에 해당한다.

원시 코드 라인 수(LOC)는 소스 코드가 몇 줄인지 나타내는 구현 규모의 척도이다. 같은 기능을 제공하더라도 프로그래밍 언어나 구현 방식에 따라 코드 줄 수는 달라질 수 있다. 기능점수는 이러한 코드 분량과 구별하여 **사용자 관점의 기능 규모**를 측정하므로, 원시 코드 라인 수는 기능점수 산정 항목에 포함되지 않는다.

NoteBase

문 20. LTE(Long-Term Evolution) 표준에 대한 설명으로 옳은 것만을 모두 고르면?

ㄱ. 다중입력 다중출력(MIMO) 안테나 기술을 사용한다.
ㄴ. 4G 무선기술로서 IEEE 802.16 표준으로도 불린다.
ㄷ. 음성 및 데이터 네트워크를 통합한 All-IP 네트워크 구조이다.
ㄹ. 다운스트림에 주파수 분할 멀티플렉싱과 시간 분할 멀티플렉싱을 결합한 방식을 사용한다.

① ㄱ, ㄷ
② ㄴ, ㄹ
③ ㄱ, ㄴ, ㄷ
④ ㄱ, ㄷ, ㄹ

정답 4번

간단한 해설

LTE는 **3GPP**가 표준화한 기술이며, IEEE 802.16은 WiMAX 계열의 표준이다. LTE 하향링크는 **OFDMA**를 사용하여 주파수 자원과 시간 자원을 함께 나누어 할당한다.

보기별 해설

ㄱ. O: MIMO는 송신 측과 수신 측에 여러 안테나를 사용하는 기술이다. LTE는 이를 이용하여 **같은 시간·주파수 자원에서 여러 데이터 흐름을 병렬 전송**하거나 수신 신뢰도를 높인다.

ㄴ. X: LTE는 **3GPP가 표준화한 이동통신 기술**이다. IEEE 802.16은 광대역 무선 접속 표준으로, WiMAX 계열 기술의 기반이 된다. 따라서 LTE를 IEEE 802.16 표준이라고 한 부분이 잘못되었다.

ㄷ. O: LTE의 핵심망인 EPC는 IP 패킷 기반의 All-IP 구조이다. 음성과 데이터를 각각 별도의 회선교환망과 패킷교환망으로 처리하는 방식에서 벗어나, **음성도 IP 패킷으로 전달**하여 데이터와 같은 IP 기반 망에서 처리한다. LTE에서 이러한 음성 서비스를 제공하는 방식이 VoLTE이다.

ㄹ. O: 기지국에서 단말로 전송하는 하향링크에는 **직교 주파수 분할 다중 접속(OFDMA)**을 사용한다. 주파수 대역을 여러 직교 부반송파로 나누고, 시간 구간별로 각 사용자에게 자원을 할당하므로 주파수 분할과 시간 분할을 함께 활용한다. 이는 상향·하향 전송을 구분하는 FDD·TDD와는 구별되는 자원 할당 방식이다.

옳은 보기는 ㄱ, ㄷ, ㄹ이며, 이 조합은 ④에 해당한다.

NoteBase