디지털 데이터 표현

김종성
국민대학교

http://dfnc.kookmin.ac.kr/

목차

1. 개요

- **데이터의 표현 방식을 배우는 이유는?**
  - 디지털 포렌식의 근본은 디지털 데이터이며, 이에 대한 이해와 분석을 위해서는 다양한 데이터의 표현 형태를 이해해야 함.
  - 사전에 파악하지 못한 파일 포맷을 접할 수 있으며, 이를 분석하기 위해서는 다양한 데이터 표현 방식에 대한 이해가 필요

2. 이진수 체계 (Binary numeral system)

- **디지털 데이터는 기본적으로 이진수를 사용**
- **데이터는 '0' 또는 '1'로 표현**
  - 메모리, 저장 장치, 네트워크 통신 등 모든 디지털 시스템에서는 이진수를 사용해서 데이터를 저장하고 처리
  - 예를 들어, 전기 신호가 통할 때는 '1', 그렇지 않을 때는 '0'으로 간주
  - 이진수 체계는 디지털 데이터의 최소 단위이며, '0' 또는 '1'을 비트(binary digit = bit) 라고 함
  - 대부분의 경우는 최소 단위를 1 byte (= 8 bit)를 사용 (압축 파일과 같은 특수 경우는 비트가 최소 단위가 됨)

0 1 0 1 1 1 1 0 1 0 1 0 0 0 0 1 ……

비트(bit)로 표현된 데이터

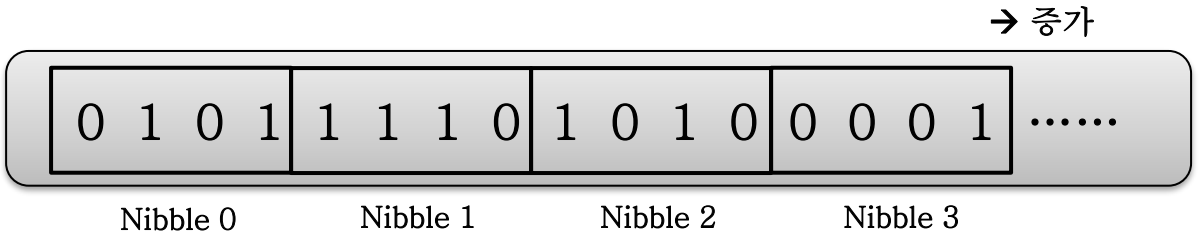
3. 데이터의 구성 단위 (물리적 단위)

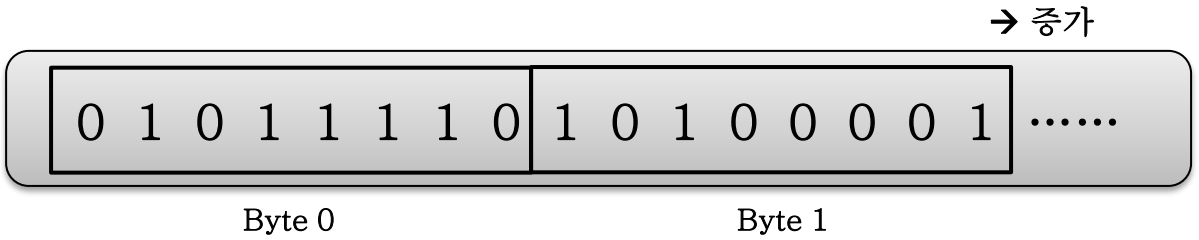
- **물리적 단위**
  - 실제 물리적 장치(메모리, 저장 장치 등)에서 사용되는 단위로 앞에서 설명한 비트(bit)가 최소 단위
  - 모든 물리적 단위는 비트로 표현
  - 최근에 저장 장치의 용량이 증가하면서 바이트(byte) 단위를 주로 사용
  - 4바이트인 워드(word), 8바이트인 더블 워드(double word) 등은 컴퓨터에서 연산의 단위로 사용

비트 (bit)	**- bit는 binary digit의 약자 - 데이터 구성의 최소 단위 - 0과 1로 이루어짐**	
쿼터 (quarter)	**= 1/4 바이트**	**= 2 비트**
니블 (nibble)	**= 1/2 바이트**	**= 4 비트**
바이트 (byte)	**= 1 바이트**	**= 8 비트**
워드 (word)	**= 4 바이트**	**= 32 비트**
더블 워드 (double word)	**= 8 바이트**	**= 64 비트**
Kilobyte (KB)	**= 1,024 bytes**	**= 2^{10} bytes**
Megabyte (MB)	**= 1,024 KB**	**= 2^{20} bytes**
Gigabyte (GB)	**= 1,024 MB**	**= 2^{30} bytes**
Terabyte (TB)	**= 1,024 GB**	**= 2^{40} bytes**
Petabyte (PB)	**= 1,024 TB**	**= 2^{50} bytes**
Exabyte (EB)	**= 1,024 PB**	**= 2^{60} bytes**
Zettabyte (ZB)	**= 1,024 EB**	**= 2^{70} bytes**
Yottabyte (YB)	**= 1,024 ZB**	**= 2^{80} bytes**

물리적 단위의 예

- 니블(Nibble) 과 바이트(Byte)





3. 데이터의 구성 단위 (논리적 단위)

- **논리적 단위**
  - 논리적 단위는 정보를 저장 및 처리하는데 사용
  - 디지털 포렌식 관점에서 분석의 대상이 되는 최소 단위이며, 그 내부는 물리적 단위로 구성
  - 즉, 물리적 단위에 대한 이해가 있어야 포렌식 분석에서 논리적 단위의 내부를 파악할 수 있음
  - 특히, 파일(file)은 레코드(record)의 집합으로 응용프로그램의 처리 단위이며, 포렌식 분석의 주요 대상임

필드 (field)	**- 여러 개의 바이트나 워드가 모여 이루어짐 - 파일 구성의 최소 단위로 항목 또는 아이템**
레코드 (record)	**- 프로그램 내의 자료 처리 기본 단위 (논리적 레코드)**
블록 (block)	**- 저장매체에 입출력될 때의 기본 단위 (물리적 레코드)**
파일 (file)	**- 관련된 레코드의 집합으로 하나의 프로그램 처리 단위**
데이터베이스 (database)	**-파일(레코드)의 집합으로 계층적 구조를 갖는 자료 단위**

수 체계 (Number System)

- **수 체계**
  - 디지털 데이터는 기본적으로 이진수를 사용
  - 우리가 실생활에 사용하는 수 체계인 10진수와는 다름
  - 그러므로 디지털 기기에서 저장 및 처리되는 데이터를 이해하기 위해서는 디지털 기기가 사용하는 수 체계에 대한 이해가 필요

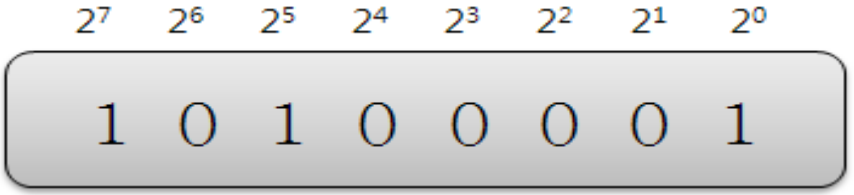
수 체계	수의 범위
2진수 (binary)	0, 1
8진수 (octal)	0 ~ 7
10진수 (decimal)	0 ~ 9
16진수 (hexadecimal)	0 ~ 9, a ~ f

2 진수 (Binary Numbers)

- **1 Byte**
  - 컴퓨터 메모리의 기본 단위인 1바이트(8비트)에 저장된 데이터를 나타내며, 이 값은 아래와 같은 공식을 이용해서 10진수로 변환
  - 계산식: $(1\times2^7)+(1\times2^5)+(1\times2^0)=128+32+1=161$
  - 1바이트는 2진수로 표현하면 최소 0 ~ 최대 11111111의 값을 저장

$$2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0=255$$

  - 즉 1바이트는 0~255의 수를 저장할 수 있으며, 더 큰 수를 저장하기 위해서는 2개 이상의 바이트가 필요



2진수 계산법

2 진수 (Binary Numbers)

- **2 Byte 계산**
  - 2바이트(16비트)는 최대 11111111 11111111의 값을 저장할 수 있으며, 이는 10진수로 65535 을 나타냄
  - 즉, 2바이트는 0~65535의 수를 저장할 수 있으며, 이처럼 n개의 바이트를 이용해서 큰 수를 저장

2Byte 표현

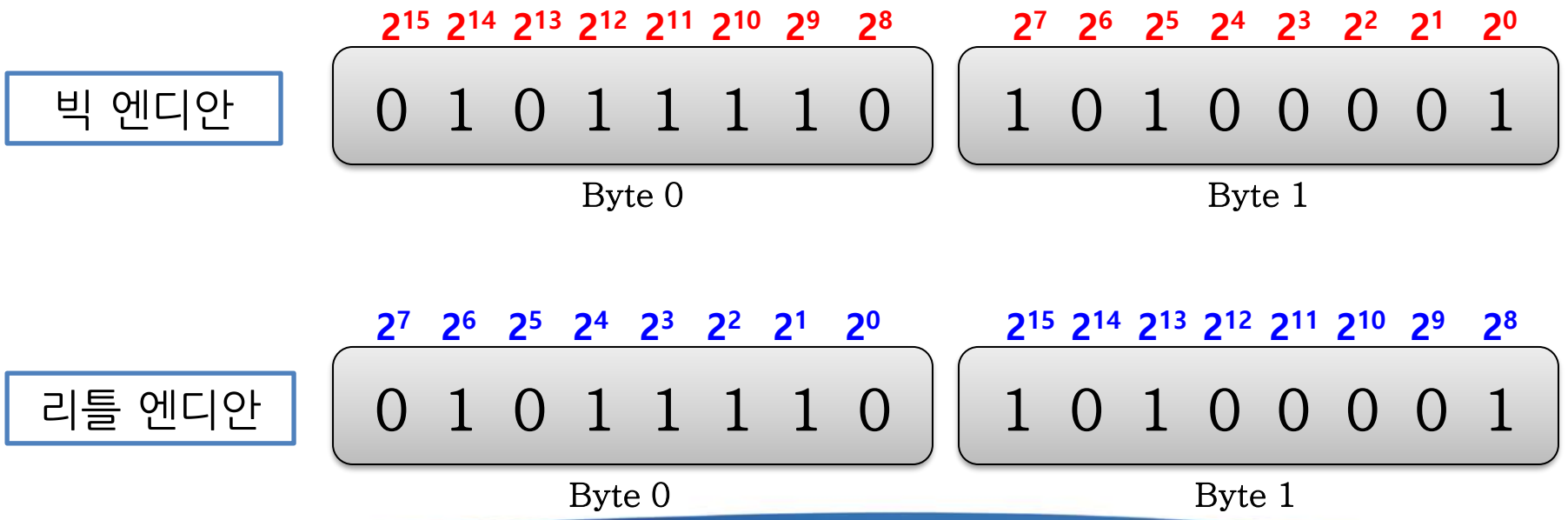
2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
0	1	0	1	1	1	1	0	1	0	1	0	0	0	0	1
Byte 0								Byte 1							

2Byte 계산법

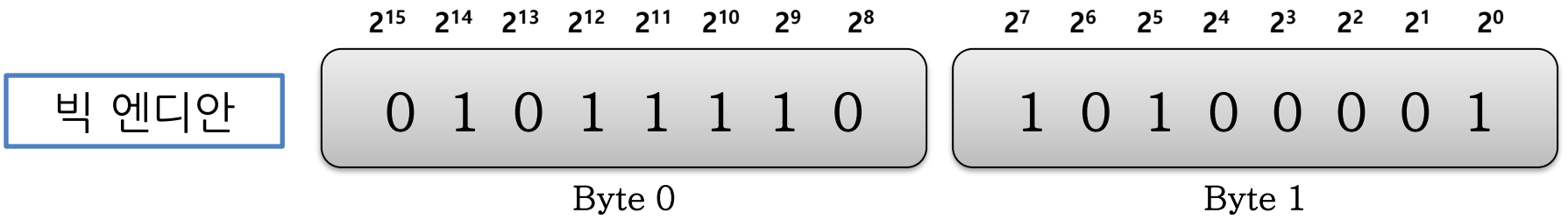
$$(1\times2^{14})+(1\times2^{12})+(1\times2^{11})+(1\times2^{10})+(1\times2^{9})+(1\times2^{7})+(1\times2^{5})+(1\times2^{0})$$
$$= 16384+4096+2048+1024+512+128+32+1$$
$$= 24226$$

Big-Endian과 Little-Endian

- **big-endian과 little-endian은 바이트의 순서를 설명하는 용어**
  - 빅 엔디안은 큰 쪽 (바이트 열에서 가장 큰 값)이 먼저 저장되는 순서
  - 리틀 엔디안은 작은 쪽 (바이트 열에서 가장 작은 값)이 먼저 저장
  - 바이트 내의 비트 배열은 빅 엔디안

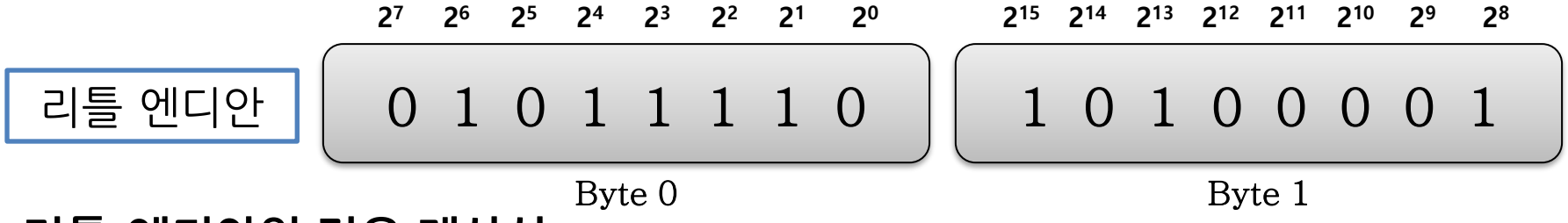


Big-Endian과 Little-Endian



- **빅-엔디안일 경우 계산식**

$$(1\times2^{14})+(1\times2^{12})+(1\times2^{11})+(1\times2^{10})+(1\times2^{9})+(1\times2^{7})+(1\times2^{5})+(1\times2^{0})$$
$$= 16384+4096+2048+1024+512+128+32+1$$
$$= 24225$$



- **리틀-엔디안일 경우 계산식**

$$(1\times2^{6})+(1\times2^{4})+(1\times2^{3})+(1\times2^{2})+(1\times2^{1})+(1\times2^{15})+(1\times2^{13})+(1\times2^{8})$$
$$= 64+16+8+4+2+32768+8192+256$$
$$= 41310$$

Big-Endian과 Little-Endian

- **빅 엔디안을 사용하는 시스템**
  - IBM 370 컴퓨터와 대부분의 RISC(reduced instruction set computer) 기반 컴퓨터들, 그리고 모토로라 마이크로프로세서는 빅 엔디안 방식을 사용
  - 왼쪽에서 오른쪽으로 읽는 언어를 사용하는 사람들에게, 이것은 일련의 문자나 숫자를 저장하는 데 있어 자연스러운 방식
  - 빅 엔디안으로 정렬되어 저장되어 있는 숫자는 두 숫자를 더한 결과를 저장하기 위해 모든 자릿수를 오른쪽으로 옮겨야 하는 일이 종종 발생

- **리틀 엔디안을 사용하는 시스템**
  - 인텔 프로세서나 DEC의 알파 프로세서에서는 리틀 엔디안을 사용
  - 리틀 엔디안을 사용하는 이유는 수의 값을 증가시킬 때 수의 왼편에 자릿수를 추가할 필요가 없기 때문
  - 리틀 엔디안 방식으로 저장된 숫자에서는, 최소 바이트가 원래 있던 자리에 그대로 머물 수 있으며, 새로운 자리 수는 최대 수가 있는 주소의 오른쪽에 추가될 수 있어, 컴퓨터 연산들이 매우 단순해지고 빠르게 수행될 수 있음

16진수 (Hexadecimal Numbers)

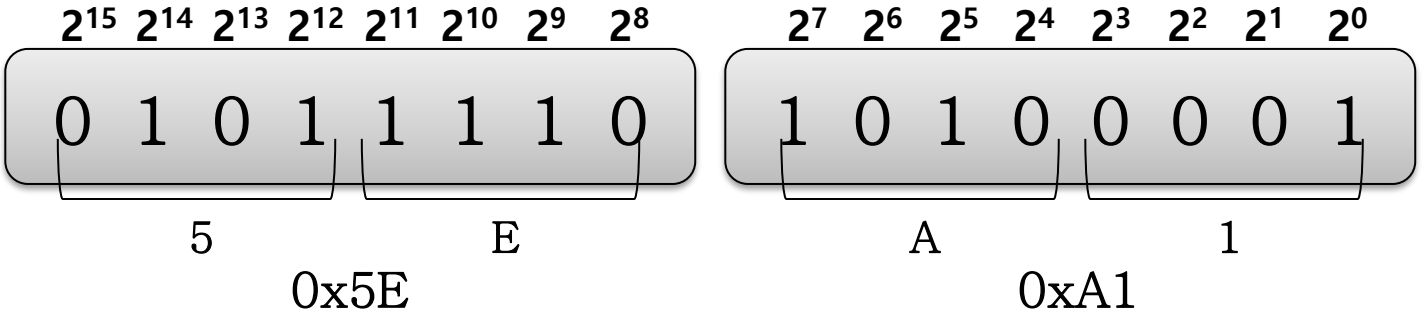
- **16진수의 필요성**
  - 디지털 데이터는 2진수를 사용하지만, 모든 데이터를 비트로 나타내면 데이터를 화면에 출력하거나 처리할 때 비효율적
  - 간단하고 편리하게 사용하기 위해 니블(4비트) 단위로 묶어서 표현
- **16진수 사용의 의미**
  - 컴퓨터 시스템의 모든 데이터는 2진수로 저장되어 있지만, 사람이 분석할 때에는 16진수로 출력된 값을 이용하는 것이 편리
  - 대부분의 분석 프로그램은 16진수를 사용하여 데이터를 표현하므로, 디지털 포렌식 분석에 있어서 매우 중요

16진수 변환 표

2진수	16진수	2진수	16진수
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

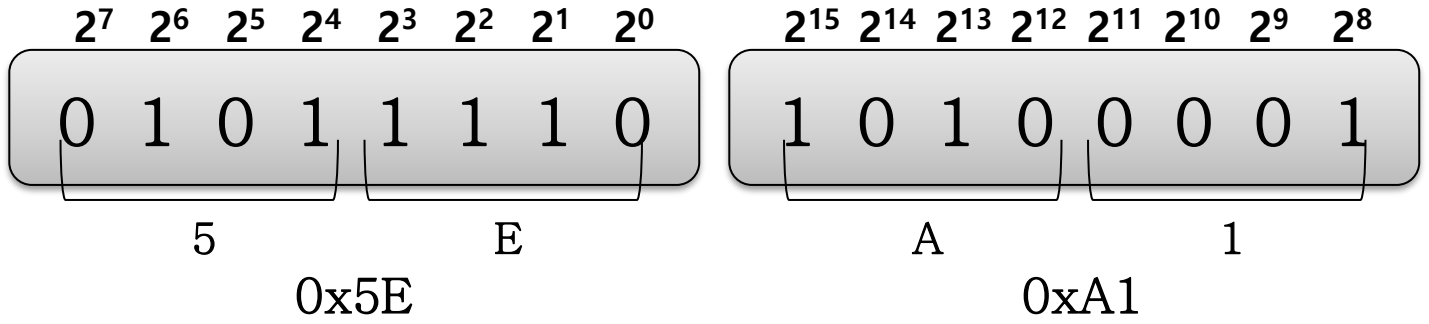
16진수 (Hexadecimal Numbers)

빅 엔디안



= 0x5EA1

리틀 엔디안



= 0xA15E

문자 (Characters)

- **문자 코드**
  - 디지털 포렌식 분석에서 문자 데이터를 분석할 때, 이에 대한 이해가 있어야 데이터의 의미를 정확하게 파악
  - 디지털 기기에서 문자는 화면에 출력되는 형태를 결정하는 중요한 요소
  - 한글, 영어(A~Z, a~z), 숫자(0~9), 특수문자 등의 모양을 특정 2진 값으로 정해 놓은 문자 코드(character codes)를 사용
  - 이러한 문자 코드는 세계 표준으로 지정하여 서로 다른 시스템에서도 동일하게 해석될 수 있도록 함

- **대표적인 문자 코드**
  - 아스키(ASCII), 엡시딕(EBCDIC), 유니코드(Unicode) 등

ASCII 문자 코드

- **아스키 코드 (ASCII: American Standard Code for Information Interchange)**
  - 미국 표준 협회(ANSI, American National Standards Institute)가 제정한 자료 처리 및 통신 시스템 상호간의 정보 교환용 표준 코드
  - 7비트로 구성된 128 종의 기호를 정한 것으로, 1바이트로 하나의 문자를 표현

- **아스키 코드의 구성**
  - 128 종의 기호는 제어 부호 33자, 그래픽 기호 33자, 숫자 10자 (0~9), 알파벳 대소문자 52자 (A~Z, a~z)로 구성
  - 아스키 코드는 영문을 표기하는 대부분의 시스템에서 사용
  - 아스키 코드의 제어 부호는 통신의 시작과 종료, 라인 피드(line feed) 등을 표시할 수 있기 때문에 데이터 통신에도 이용

ASCII 문자 코드 - 예

아스키 코드	
16진수	10진수
00	000
01	001
02	002
03	003
04	004
05	005
06	006
07	007
08	008
09	009
0A	010
0B	011
0C	012
0D	013
0E	014
0F	015

아스키 코드	
16진수	10진수
21	033
22	034
23	035
24	036
25	037
26	038
27	039
28	040
29	041
2A	042
2B	043
2C	044
2D	045

아스키코드		문 자
16진수	10진수	
40	064	@
41	065	A
42	066	B
43	067	C
44	068	D
45	069	E
46	070	F
47	071	G
48	072	H
49	073	I
4A	074	J
4B	075	K
4C	076	L
4D	077	M
4E	078	N
4F	079	O

한글 문자 코드

- **한글 코드의 종류**
  - 조합형, 완성형, 확장 완성형, 유니코드 (Unicode)
- **조합형 코드**
  - 2바이트 완성형 코드가 발표되기 전까지 사용되던 코드로, 한글을 초성, 중성, 종성에 따라 조합하여 표현
  - 이론 상 한글 11,172자를 모두 표현할 수 있으며, 한글 입력에 대한 처리가 쉬웠으나 Microsoft Windows 95에서 완성형 코드를 선택함에 따라 1990년도 중반까지만 사용
- **완성형 코드**
  - 2바이트 완성형 코드를 의미하며, EUC-KR로 표준화되어 사용
  - 완성형에서 한글은 연속된 두 개의 바이트를 이용해서 표현할 수 있으며, 첫 번째 바이트와 두 번째 바이트 모두 0xA1~0xFE 사이의 값을 가짐
  - 완성형 코드는 한글을 2,350자 밖에 지원하지 않지만 Microsoft Windows에서 선택함에 따라 최근까지 널리 사용
- **확장 완성형 코드**
  - Microsoft에서 완성형(EUC-KR) 코드에 글자를 추가한 것으로 코드페이지 949(CP949)라고 불림
  - ECU-KR과 마찬가지로 한글을 표현하는데 2바이트를 사용

한글 완성형 코드(EUC-KR)의 예

• 한글 완성형의 부분 코드 표

0x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
B0A0		가	각	간	갇	갈	갉	갊	감	갑	값	갓	갔	강	갖	갗
B0B0	같	갚	갛	개	객	갠	갤	갬	갭	갯	갰	갱	갸	갹	갼	걀
B0C0	걋	걍	걔	걘	걜	거	걱	건	걷	걸	걺	검	겁	것	겄	겅
B0D0	겆	겉	겊	겋	게	겐	겔	겜	겝	겟	겠	겡	겨	격	겪	견
B0E0	겯	결	겸	겹	겻	겼	경	곁	계	곈	곌					
B0F0	곧	골	곪	곬	곯	곰	곱	곳	공	곶	과					

'강'의 경우
0xB0AD

• 한글 완성형의 부분 코드 표

0x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C0A0		웩	웬	웰	웸	웹	웽	위	윅	윈	윌	윔	윕	윗	윙	유
C0B0	육	윤	율	윰	윱	윳	융	윷	으	윽	은	을	읊	음	읍	읏
C0C0	응	읒	읓	읔	읕	읖	읗	의	읜	읠	읨	읫	이	익	인	일
C0D0	읽	읾	잃	임	입	잇	있	잉	잊	잎	자	작	잔	잖	잗	잘
C0E0	잚	잠	잡	잣	잤	장	잦	재	잭	잰	잴	잼	잽	잿	쟀	쟁
C0F0	자	쟉	쟌	쟎	쟐	쟘	쟝	쟤	쟨	쟬	저	적	전	절	젊	

'의'의 경우
0xC0C7

유니 코드

- **유니코드(unicode)란?**
  - 전 세계의 모든 문자를 컴퓨터에서 일관되게 표현하고 다룰 수 있도록 설계된 산업 표준
  - ISO 10646으로 정의된 UCS (Universal Character Set)를 말함.
  - 한글은 1996년 유니코드 2.0에서부터 11,172자가 모두 포함

- **유니코드의 구성**
  - 유니코드는 31비트 문자 셋 이지만, 특수한 문자를 제외한 전 세계 모든 문자들은 하위 16비트(2 Byte)의 영역 안에 정의
  - 이러한 유니코드 첫 65,536개의 코드(0000~FFFF)를 기본 다국어 평면 (BMP, Basic Multilingual Plane)라고 부름

- **유니코드의 종류**
  - UCS-2, UCS-4
  - UTF-8, UTF-16, UTF-32

시간 정보의 표현 방식

- **디지털 포렌식에서 시간 정보**
  - 시간 정보는 범죄가 발생한 시점에 대한 행위를 가늠하는 잣대이므로 디지털 포렌식 관점에서 매우 중요한 정보
  - 2진 값으로 저장된 시간 데이터를 가시적인 형태로 변환하는 방법을 숙지할 필요가 있음

- **시간 정보의 표현 방식**
  - 컴퓨터의 시간 저장 형식
    - MS-DOS Date/Time
    - time_t, time64_t
    - FILETIME
  - 휴대폰의 시간 저장 형식
    - 10진수, 16진수, ASCII
    - Qualcomm TimeStamp, Anycall TimeStamp 등

컴퓨터의 시간- MS-DOS Date/Time

- MS-DOS에서 사용된 시간 저장 형식으로 컴퓨터의 현재 날짜와 시간(local time)을 저장한다. 날짜와 시간을 각각 2바이트로 저장

•MS-DOS 시간 정보의 범위

시작	1980년 01월 01일 00:00:00 (00 : 21 : 00 : 00)
끝	2107년 12월 31일 23:59:58 (FF : 9F : BF : 7F)

날짜 (H)								날짜 (L)								시간 (H)								시간 (L)							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Y	Y	Y	Y	Y	Y	Y	M	M	M	M	D	D	D	D	D	h	h	h	h	h	m	m	m	m	m	m	s	s	s	s	s

•MS-DOS 시간 정보의 구조

Y	: 1980년부터 지난 년도
M	: 1월 = 1, 2월 = 2, … , 12월 = 12
D	: Day. 1~31일
h	: Hour. 0~23시
m	: Minute. 0~59분
s	: Second. 2초 단위로 0~29의 값이 저장됨

컴퓨터의 시간- time_t와 time64_t

- **time_t**
  - 유닉스 운영 체제에서 시간을 저장하는 표준 형식
  - 시간 형식을 4바이트(32비트)로 저장하며, 정수형을 사용하는 1970년 1월 1일 자정 (UTC, Universal Time Coordinated) 이후 경과된 초(second)를 저장

- **time64_t**
  - time_t에 부호 있는 4바이트(32비트) 정수형을 사용하는 경우, 2038년 1월 19일 03:14:07 UTC가 지나면 시간이 1901년 또는 1970년이 되는 문제
  - 현재 이를 해결하기 위해 64비트 아키텍처를 지원하는 운영 체제는 8바이트(64비트)를 이용하도록 time_t의 정의를 변경

① time_t로 부호 있는 4바이트를 사용하는 경우	
시작	: 1970년 01월 01일 00:00:00 (0)
끝	: 2038년 01월 19일 03:14:07 (4,294,967,295)

② time_t로 부호 없는 4바이트를 사용하는 경우	
시작	: 1970년 01월 01일 00:00:00 (0)
끝	: 2106년 02월 05일 21:28:15 (2,147,483,647)

③ time64_t를 사용하는 경우 (부호 없는 8바이트)	
시작	: 1970년 01월 01일 00:00:00 (0)
끝	: 292277026596년 12월 4일 (18,446,744,073,709,551,615)

컴퓨터의 시간- FILETIME

- **FILETIME 시간 포맷**
  - Window 32bit에서 주로 사용하는 시간 정보 저장 형식
  - 1601년 1월 1일 자정 (UTC, Universal Time Coordinated) 이후 100나노초 간격의 숫자를 저장
  - FILETIME 구조체는 8바이트(64비트)이며, 날짜와 시간을 각각 4바이트에 저장

- **FILETIME을 time_t로 변환하는 공식**
  - time_t = (FILETIME - 0x19DB1DED53E8000) / 10000000

•FILETIME의 기록 범위

시작	**: 01601년 01월 01일 00:00:00 (00000000 : 00000000)**
끝	**: 60056년 03월 28일 14:36:10 (FFFFFFFF : FFFFFFFF)**

휴대폰의 시간 저장 형식

- 휴대폰에서의 시간 정보는 제조사 혹은 모델 별로 다름.
- 동일한 모델일지라도 받은 문자, 보낸 문자, 메모 등의 항목에 따라 다른 저장 형식을 사용하는 경우도 존재

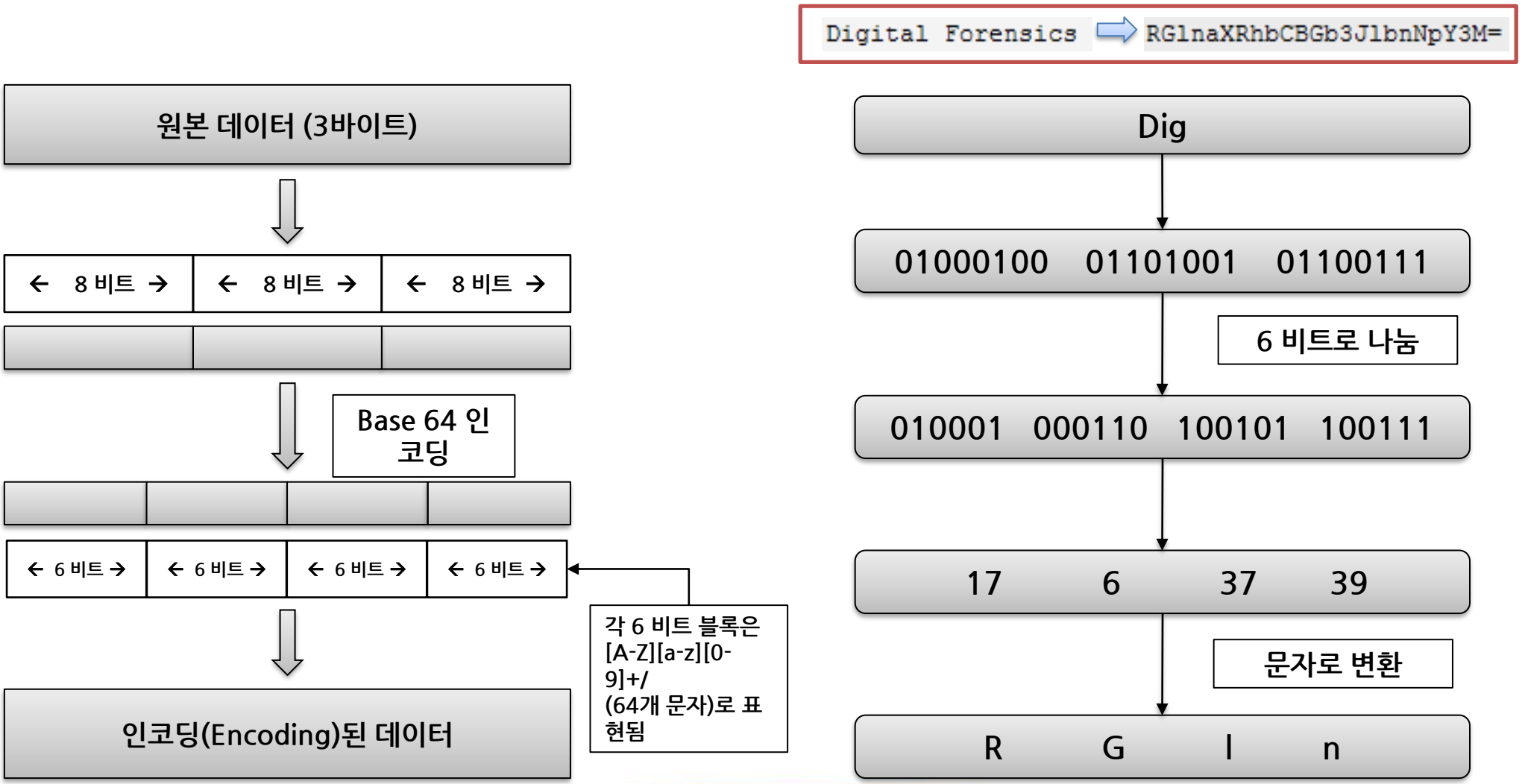
형식	저장 값 (16진수)	시간 표현
10진수	08 02 24 15 46 02	2008년 2월 24일 15시 46분 2초
16진수 (리틀 엔디안)	D807 01 13 13 15 33(0xD807 = 2008)	2008년 1월 19일 19시 21분 51초
16진수	07 03 02 0C 33 36(0x07 = 2007)	2007년 3월 2일 12시 51분 54초
16진수 (빅 엔디안)	07D9 08 05 09 15 06(0x07D9 = 2009)	2009년 8월 5일 9시 21분 6초
ASCII	32303036 2E 3034 2E 3035 20 3135 3A 3031	2006년 4월 5일 15시 1분 (출력형태 : 2006.04.05 15:01)
	3039 2E 3034 2E 3031 20 3132 3A 3033	2009년 4월 1일 12시 3분 (출력형태 : 09.04.01 12:03)
Qualcomm TimeStamp	0x054BEC34 (4바이트)	2008년 2월 24일 15시 8분 53초
	0x383936343630313139 (896460119초, ASCII)	2008년 6월 2일 16시 41분 59초
Anycall TimeStamp	0x93E18003 (4바이트)	2007년 10월 7일 22시 35분 0초

데이터 인코딩 (Encoding)

- **데이터 인코딩**
  - 숫자, 문자, 시간 등의 데이터는 원본 그대로의 형태로 저장될 수도 있지만, 다양한 인코딩 알고리즘(Encoding) 에 따라 특수한 형태의 데이터로 변환
  - Base 64 인코딩은 웹, 이메일 S/W에서 바이너리 데이터 전송 등에 널리 사용
- **Base 64**
  - 8비트 바이너리 데이터를 아스키(ASCII) 영역의 문자들로만 이루어진 일련의 문자열로 변환
  - 즉, 임의의 바이너리 데이터를 64개 아스키 문자의 조합으로 표현
  - 인코딩된 문자열은 아래와 같이 알파벳 대소문자와 숫자, 그리고 "+", "/" 기호 64개로 이루어진다.

Base 64 인코딩의 64개 문자	**ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 0123456789+/**

Base 64 인코딩 알고리즘



파일

- **파일**
  - 응용프로그램의 처리 단위이며, 디지털 포렌식 분석의 주요 대상
  - 파일은 파일 시스템(file system)에 저장되는 기본 단위로 파일 시스템 내에 존재하는 파일은 일반적으로 이름과 확장자(extensions)를 부여
  - 파일의 확장자는 응용프로그램에서 처리 가능한 파일을 분류하는 역할을 하지만, 파일 시스템에 저장되고 관리되기 때문에 윈도우 탐색기를 이용해서 간단하게 다른 확장자로 변경 가능

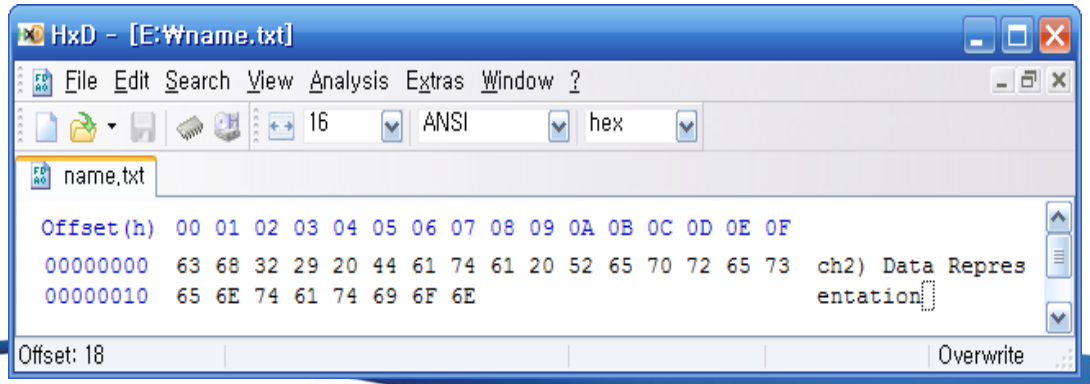
- **파일 분석**
  - 파일의 종류를 판단하기 위한 확장자(extensions)와 시그니처(signatures)를 이용하며, 파일 내부에 기록되는 데이터의 유형을 잘 파악해야 함
  - 파일을 분석하기 위해서는 헥스 뷰어(hex viewer)를 이용하여 그 구조와 데이터 유형을 파악

파일 분석 (File Analysis)

- **헥스 에디터(hex-editor) 또는 헥스 뷰어(hex viewer)**
  - 헥스 에디터는 파일 내부의 실제 데이터를 16진수 형태로 보여주는 도구
  - 일반 텍스트 타입의 파일은 텍스트 편집기를 이용해서 쉽게 확인이 가능하지만, 대부분 파일 내부에는 텍스트가 아닌 데이터가 다수 존재하여 헥스 에디터(hex-editor)를 사용

- **파일 포맷 분석**
  - 포렌식 분석의 대상이 되는 대부분의 파일은 내부에 데이터를 저장하기 위한 자신들만의 형식(format)을 정의
  - 파일의 내부에는 숫자, 문자를 포함한 다양한 형태의 데이터가 저장되며, 헥스 에디터를 이용해서 분석할 수 있음
  - 대표적인 무료 헥스 에디터로는 HxD, Fhred 등이 있으며, 상용으로는 Hex Workshop, WinHex 등이 있음

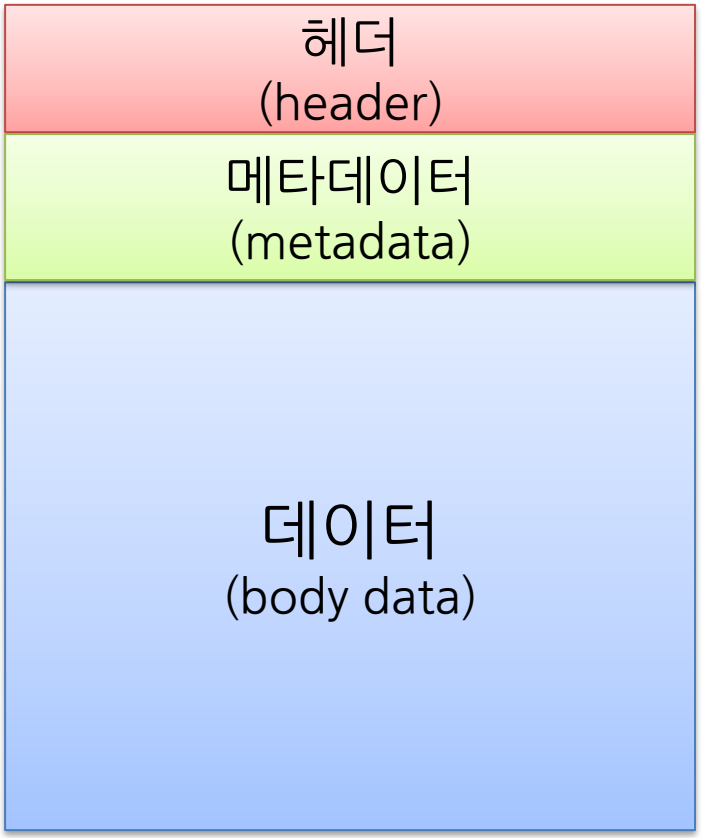
•헥스 에디터 HxD



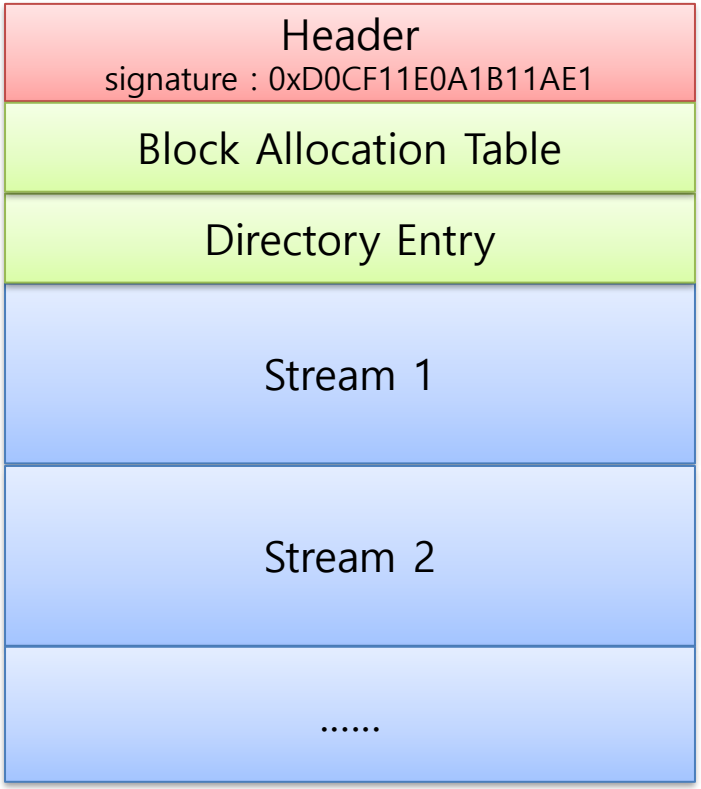
파일의 내부 구조

- **파일 포맷**
  - 각 응용프로그램은 데이터를 저장하기 위해 고유한 저장 형식(format)을 사용
  - 포렌식 조사 과정에서 특정 파일을 분석하기 위해서는 해당 파일의 내부 구조에 대한 철저한 이해가 필수적

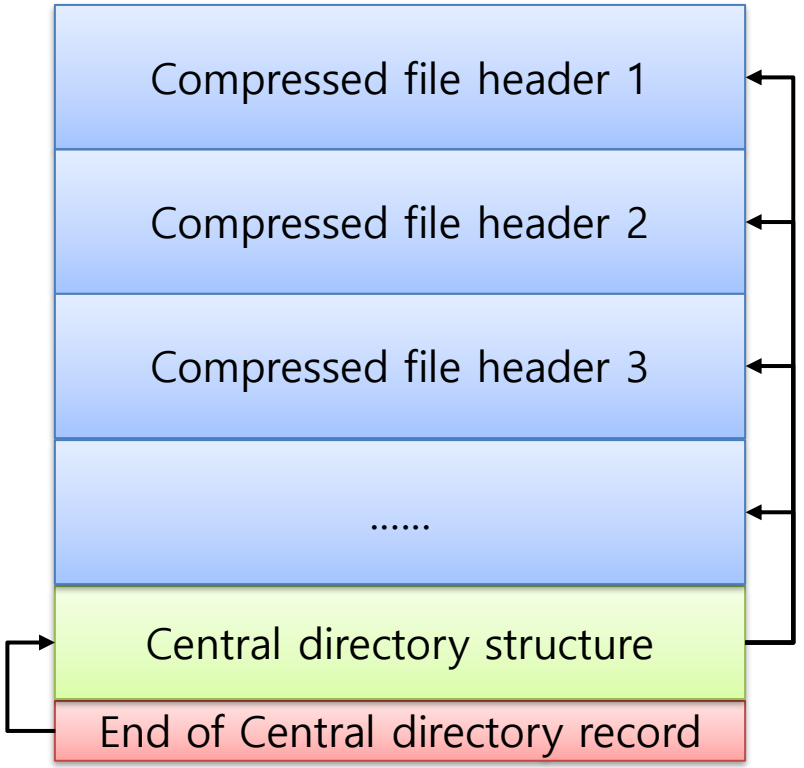
- **파일 포맷의 일반적인 내부 구조**
  - 헤더(header)
    - 헤더에는 해당 파일의 타입을 나타내는 시그너처(signature)가 포함되며, 그 이외에 전체 파일의 크기, 메타데이터의 시작 위치 등의 부가 정보가 저장
  - 메타데이터(metadata)
    - 사용자가 입력한 데이터를 설명하거나 관리하기 위한 데이터로 응용프로그램에서 자동적으로 생성
  - 데이터(body data)
    - 파일의 본문(body)을 구성하는 실제 데이터가 저장되는 영역



파일의 내부 구조 – 예



Microsoft Office 97-2003 파일 내부 구구조



압축 파일(ZIP 아카이브) 내부 구조