Chapter 02
디지털 정보의 표현

목차

학습목표

- 컴퓨터의 저장 용량 단위와 처리 속도 단위를 알아본다.
- 2진법과 16진법을 이해하고, 진법 변환 방법을 살펴본다.
- 컴퓨터 내부의 데이터 표현 방법을 살펴본다.
- 논리 연산자(AND, OR, XOR, NOT)의 연산 방법을 알아본다.

용량 단위

비트(bit)

- 컴퓨터에서 데이터를 표시할 때 사용하는 최소 단위
- 2진법을 사용하기 때문에 1비트로 표현할 수 있는 수는 0과 1

바이트(byte)

- 비트 8개를 묶어 사용하는 단위
- 1바이트 = 8비트

그림 3-1 비트와 바이트

용량 단위

CPU 데이터 처리 속도

- 초기 : 한 번에 8비트를 처리하는 CPU 사용 (예: 개인용 컴퓨터 애플 II)
- 이후 : 32비트 CPU 등장
- 현재 : 64비트 CPU 대중화

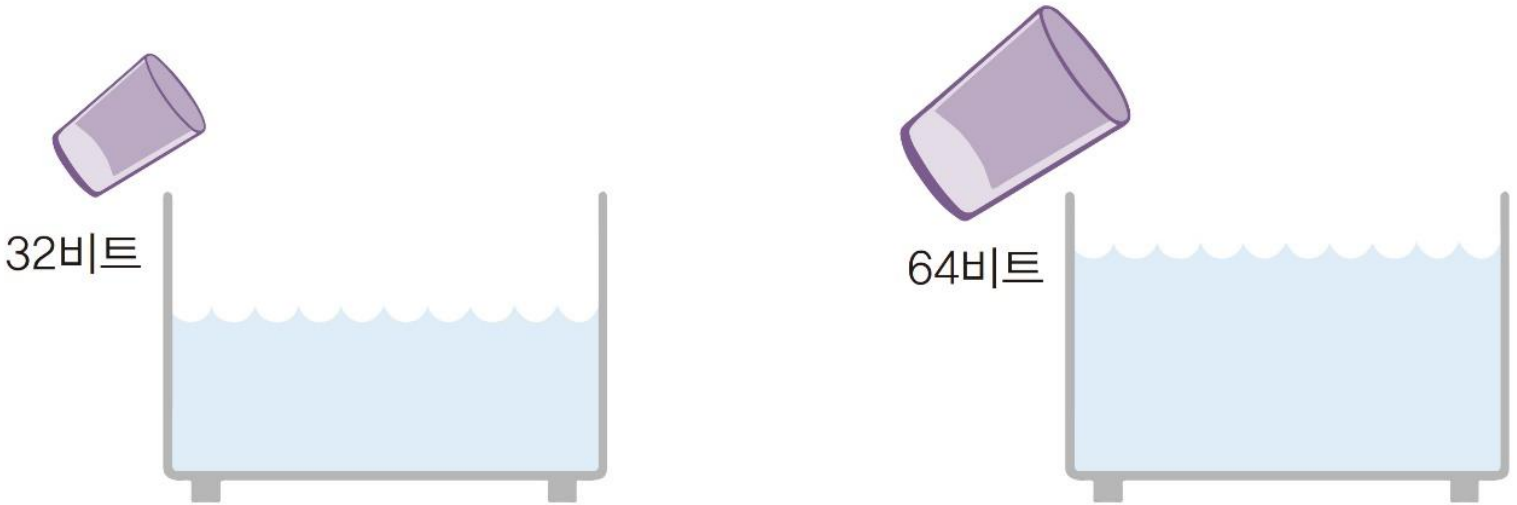


그림 3-2 32비트와 64비트 차이

용량 단위

워드(word)

- 컴퓨터가 한 번에 처리할 수 있는 데이터 크기를 나타내는 단위
- 32비트 CPU : 한 번에 32비트 데이터 처리, 1워드=32비트
- 64비트 CPU : 한 번에 64비트 데이터 처리, 1워드=64비트

용량 단위

■ 요약

표 3-1 컴퓨터 용량 단위

용량 단위	설명
비트(bit)	데이터를 표현하는 최소 단위
바이트(byte)	8비트를 하나로 모은 것
워드(word)	컴퓨터가 한 번에 처리할 수 있는 데이터 단위 예 32비트 CPU의 1워드는 32비트

큰 용량을 표현하는 단위

표 3-2 큰 용량을 표현하는 단위

용량 단위	표기	2진 크기	10진 크기	바이트 대비 크기	10진 단위
바이트(Byte)	B	1	1	1B	일
킬로바이트(Kilo Byte)	KB	2^{10}	10^{3}	1,000B	일천
메가바이트(Mega Byte)	MB	2^{20}	10^{6}	1,000,000B	일백만
기가바이트(Giga Byte)	GB	2^{30}	10^{9}	1,000,000,000B	일십억
테라바이트(Tera Byte)	TB	2^{40}	10^{12}	1,000,000,000,000B	일조
페타바이트(Peta Byte)	PB	2^{50}	10^{15}	1,000,000,000,000,000B	일천조

- 큰 용량 단위는 바로 한 단계 낮은 단위보다 1,024(2^{10})배 큼
- 1킬로바이트(1KB)는 정확히 1,024바이트(2^{10}B)
- 그러나 사람은 2진수보다 10진수에 더 익숙하므로 보통 1KB를 약 1,000B(10^{3}B)로 씀
- **엑사 – 제타 – 요타로 이어짐.**

속도 단위

하드디스크 속도 (rpm)

- 디스크 원반이 1분 동안 회전하는 수
- 7,200rpm → 디스크 원반이 1분에 7,200번 회전
- 숫자가 클수록 데이터를 저장하거나 읽는 속도가 빠름
  (5,400rpm < 7,200rpm)

속도 단위

네트워크 전송량 (bps)

- ‘bit per second’의 약어로 네트워크상에서 1초 동안 보내는 데이터의 양
- 파일 용량 표기 vs 네트워크 전송량 표기
  - 파일 용량 표기 : 단위가 바이트(byte)며 대문자 B로 표기
  - 네트워크 전송량 표기 : 단위가 비트(bit)며 소문자 b로 표기

** 1바이트는 8비트이므로 10MB는 기본적으로 10Mbps보다 8배 큼

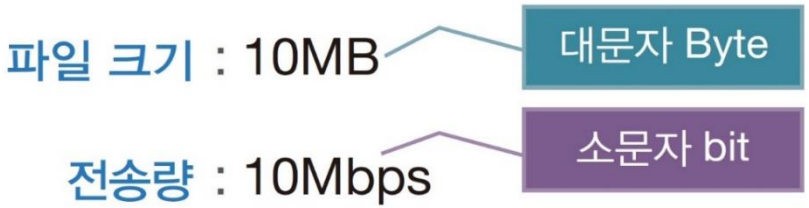


그림 3-5 파일 크기와 네트워크 전송량 표기 차이

2진법과 10진법

■ 컴퓨터가 2진법을 사용하는 이유

- 컴퓨터는 0과 1로 표현하는 2진법 사용
- 인간은 0부터 9까지 숫자 10개로 표현하는 10진법 사용
- 컴퓨터가 2진법을 사용하게 된 것은 최초의 컴퓨터가 켜기와 끄기만 할 수 있는 진공관을 사용했기 때문(꺼지면 0, 켜지면 1로 인식)
- 10진법을 사용하는 컴퓨터를 만들 수도 있으나 빠르게 계산하려면 2진법을 사용하는 것이 유리

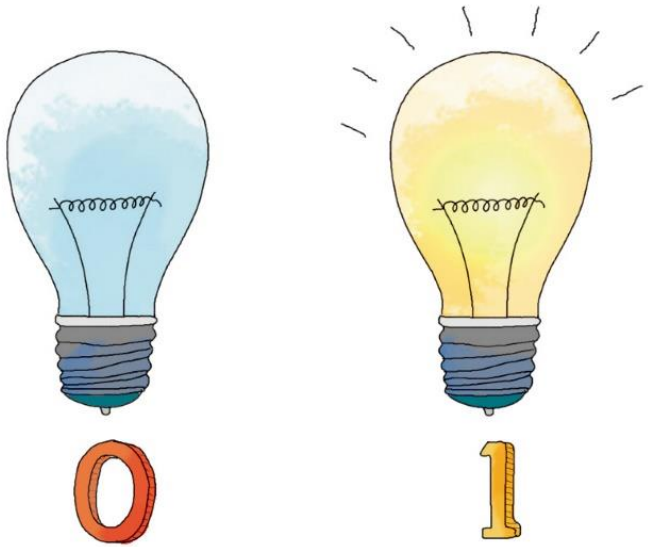


그림 3-6 2진법 개념

진법 변환

10진수 표현의 원리

$$
\begin{aligned}
237 &= 2\times10^2 + 3\times10^1 + 7\times10^0 \\
&= 2\times100 + 3\times10 + 7\times1 \\
&= 237
\end{aligned}
$$

그림 3-8 10진수 표현

2진수 → 10진수 변환

- 2진수의 각 자릿수를 곱한 후 모두 더함

** 숫자 아래의 밑첨자는 진법을 나타냄, 10진수는 표기 생략

$$
\begin{aligned}
11101101_2 &= 1\times2^7 + 1\times2^6 + 1\times2^5 + 1\times2^3 + 1\times2^2 + 1\times2^0 \\
&= 128 + 64 + 32 + 8 + 4 + 1 \\
&= 237
\end{aligned}
$$

그림 3-9 2진수의 10진수 변환

진법 변환

10진수 → 2진수 변환

- 10진수를 계속 2로 나누면서 몫은 아래에, 나머지는 오른쪽에 기록한 후 더 이상 나누어지지 않을 때 나머지를 거꾸로 읽음

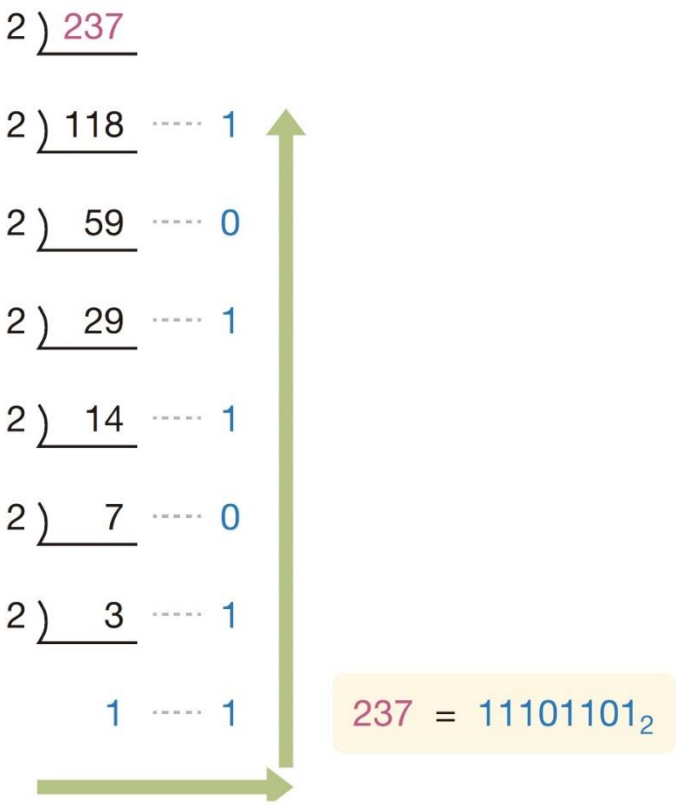
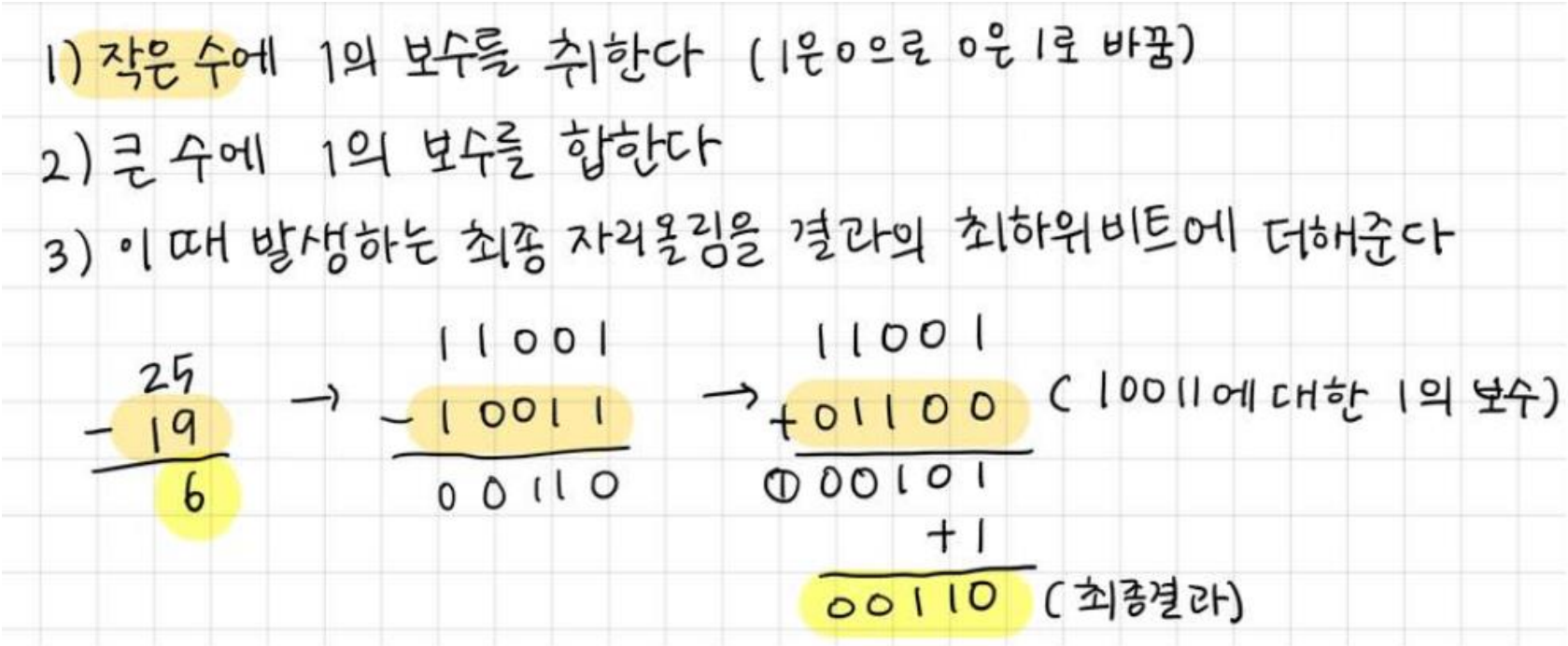


그림 3-10 10진수의 2진수 변환

진법 변환

■ 1의 보수를 이용한 뺄셈 연산



진법 변환

■ 2의 보수를 이용한 뺄셈 연산

1) 작은수에 2의 보수를 취해줌
2) 큰 수에 2의 보수를 더함
3) 자리올림을 무시해버린다

$$\begin{array}{r} 25 \\ -19 \\ \hline 6 \end{array} \rightarrow \begin{array}{r} 11001 \\ -10011 \\ \hline 00110 \end{array} \rightarrow \begin{array}{r} 11001 \\ +01101 \\ \hline 00110 \end{array}$$

(10011에 대한 2의 보수)

(자리올림 무시함)

00110 (최종결과)

16진법

- 0~F까지 숫자 16개를 사용
- 1~9는 10진수와 같고 이후 숫자 6개는 알파벳 사용
  (10은 A, 11는 B, 12는 C, 13은 D, 14는 E, 15는 F로 표기)

16진법을 사용하는 이유

- 컴퓨터에서는 16진법은 바이트를 좀 더 적은 숫자로 표현하는 데 사용
- 2진수 1111111을 표현하려면 8자리가 필요하지만,
  16진수로 표현하면 FF의 2자리로 표현 가능

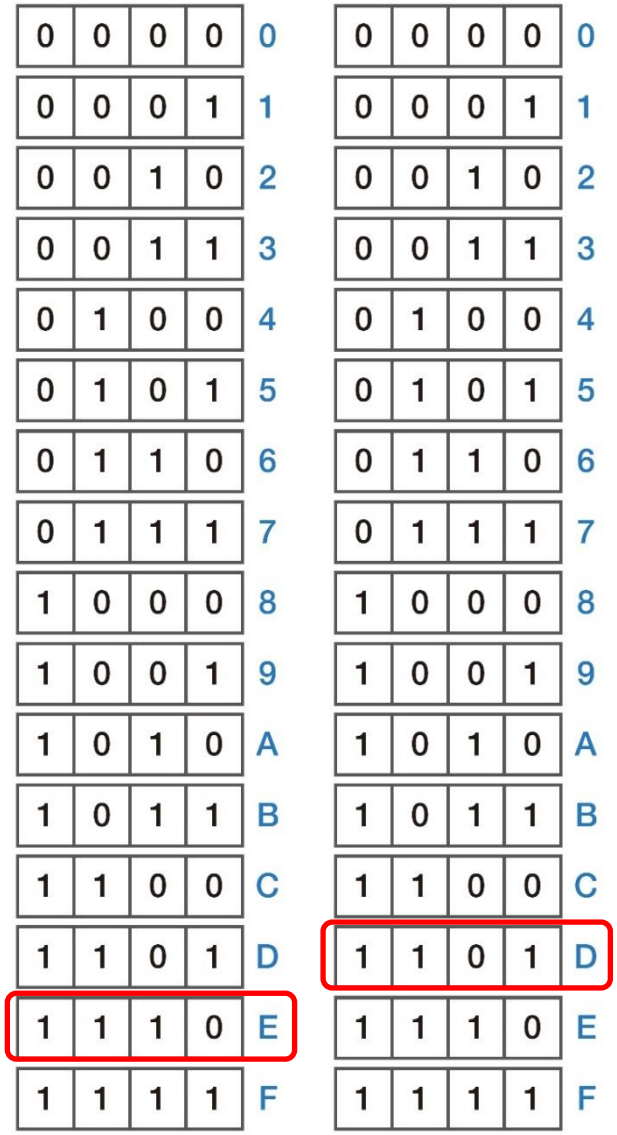
0	0	0	0	0	1	0	0	0	8
0	0	0	1	1	1	0	0	1	9
0	0	1	0	2	1	0	1	0	A
0	0	1	1	3	1	0	1	1	B
0	1	0	0	4	1	1	0	0	C
0	1	0	1	5	1	1	0	1	D
0	1	1	0	6	1	1	1	0	E
0	1	1	1	7	1	1	1	1	F

그림 3-11 4자리 2진수에 대응하는 1자리 16진수

문제

- 2진수 11101101을 16진수로 변환하면?

  정답 : ED



16진수 → 10진수 변환

- 각 자리의 숫자와 해당 위치의 단위 값을 곱한 후 모두 더하면 됨

$$
\begin{aligned}
ED_{16} &= E\times16^{1} + D\times16^{0} \\
&= 14\times16 + 13\times1 \\
&= 224 + 13 \\
&= 237
\end{aligned}
$$

그림 3-12 16진수의 10진수 변환

10진수 → 16진수 변환

- 10진수를 계속 16로 나누면서 몫은 아래에, 나머지는 오른쪽에 기록한 후 더 이상 나누어지지 않을 때 나머지를 거꾸로 읽음

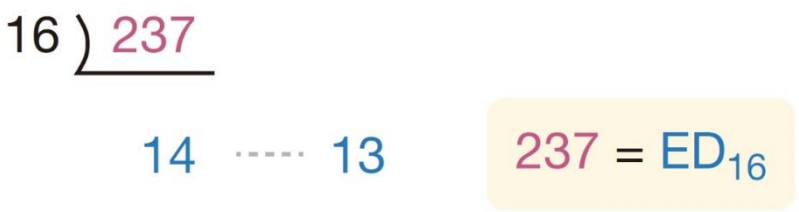


그림 3-13 10진수의 16진수 변환

16진수 사용 예

- RGB는 컴퓨터에서 이미지의 점 색상 하나를 표현할 때 사용

  ** RGB는 빛의 삼원색인 빨간색(Red), 녹색(Green), 파란색(Blue)을 뜻함

- RGB는 각각 1바이트 크기를 가지므로 1바이트가 표현할 수 있는 값은 0에서 255까지 총 256(2^8)단계
- 컴퓨터에서 하나의 색상을 표현할 때는 빨간색(R), 녹색(G), 파란색(B)을 256단계로 섞어 사용
  - RGB(0, 0, 0) : 검은색
  - RGB(255, 255, 255) : 흰색
  - RGB(255, 0, 0) : 빨간색
  - RGB(0, 255, 0) : 녹색
  - RGB(0, 0, 255) : 파란색

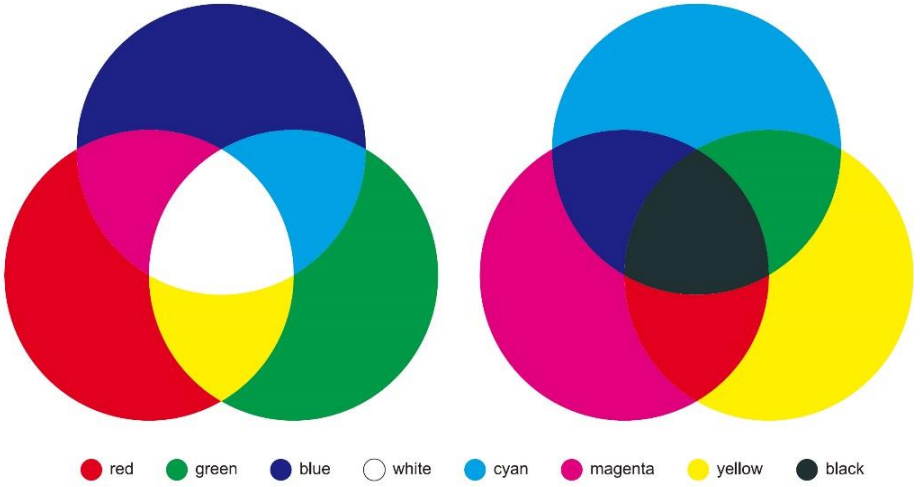


그림 3-14 RGB와 CMYK

포토샵의 컬러 피커

- 빨간색(R), 녹색(G), 파란색(B) 입력란에 0부터 255까지 숫자를 입력
- 아래쪽 입력란에 16진수를 넣어도 됨(녹색은 #00ff00이라고 작성)

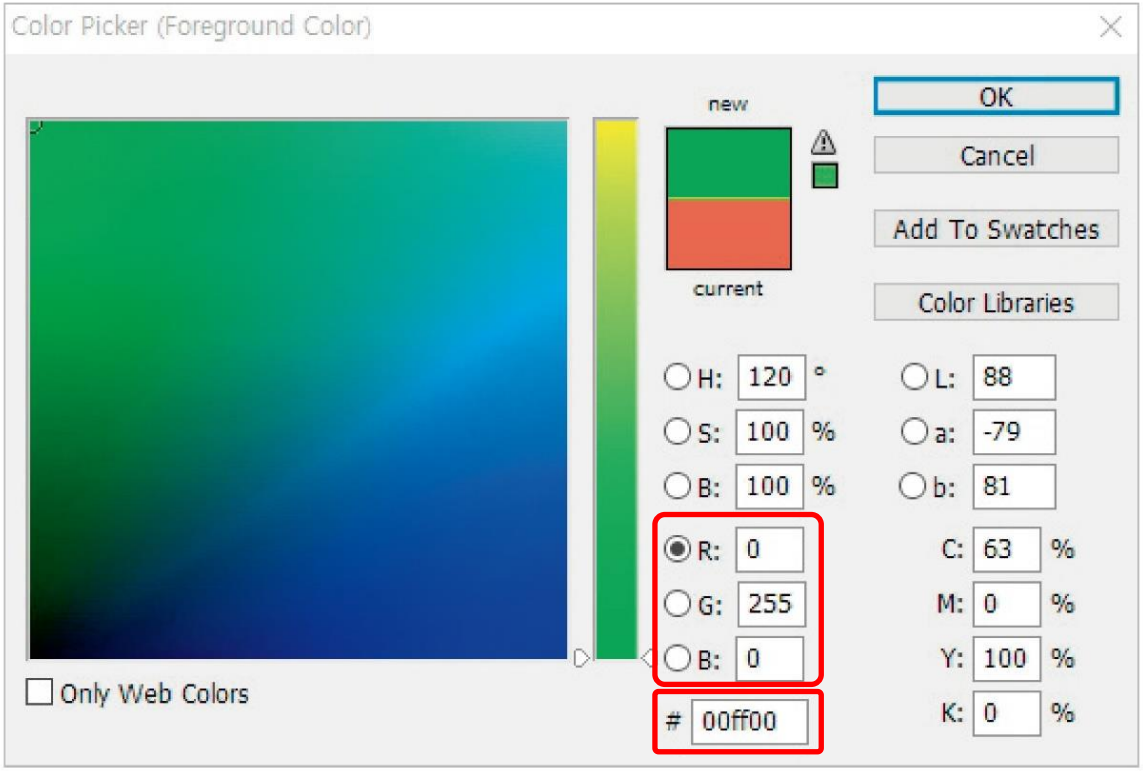


그림 3-15 포토샵에서 녹색 RGB 값을 10진수와 16진수로 설정하는 모습

CMYK

- 이미지를 종이에 출력하거나 인쇄할 때는 RGB 대신 CMYK를 사용
- CMYK : 시안(Cyan), 마젠타(Magenta), 옐로(Yellow), 블랙(Black=key)

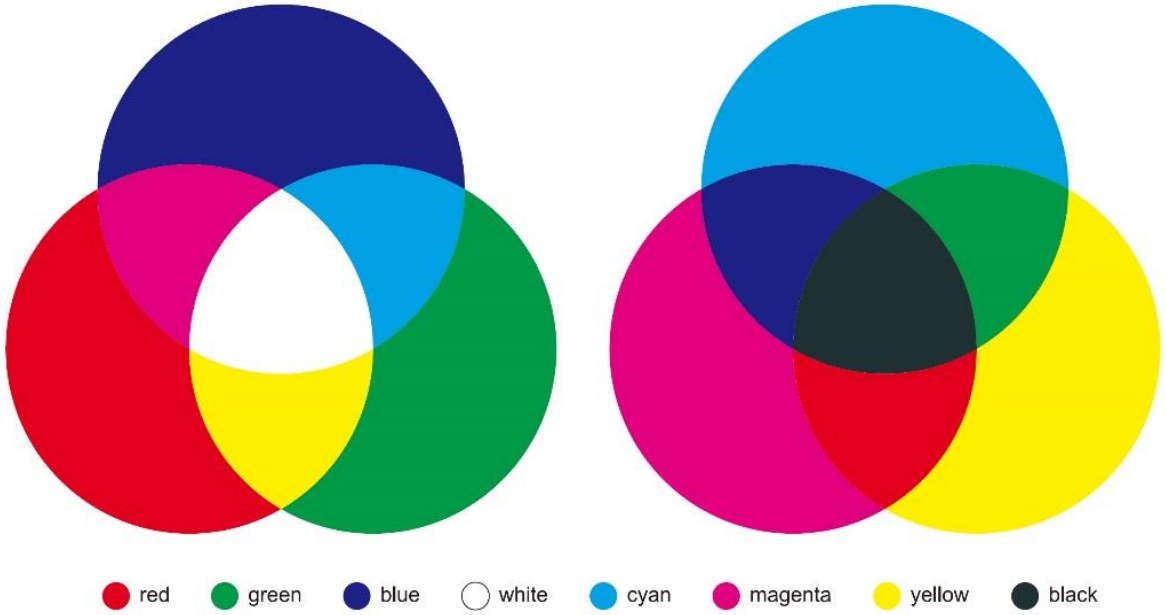


그림 3-14 RGB와 CMYK

문자 표현

아스키코드

- 컴퓨터는 문자를 처리하려고 숫자와 문자를 하나씩 대응하는 코드를 사용, 아스키코드(ASCII code)가 대표적
- 7비트로 구성되기 때문에 아스키코드로 표현할 수 있는 문자는 0~127, 총 128(2^7)개
- 52개의 영문 알파벳(a~Z), 10개의 숫자(0~9), 32개의 특수문자(@,# 등) 그리고 하나의 공백문자로 구성

0x30 = 0 (숫자 0)

0x40 = A (대문자 알파벳)

0x61= a (소문자 알파벳)

문자 표현

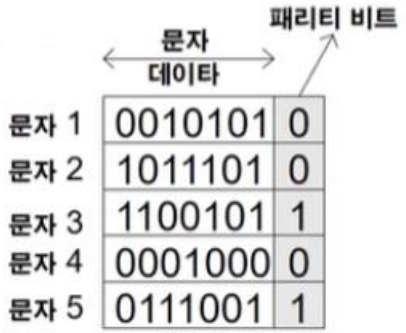
■ 아스키코드와 Parity Check

- ■ 짝수 패리티(Even Parity) : 패리티 비트 값이 자신을 포함하여 1의 개수가 짝수
- ■ 홀수 패리티(Odd Parity) : 패리티 비트 값이 자신을 포함하여 1의 개수가 홀수

- 예 : 송신측이 ASCII 코드 q (1110001)를 홀수 패리티를 사용하여 전송한다면, 11100011을 전송
  -> 짝수 패리티 사용하여 전송한다면 11100010
- 한계점 : 두 개 혹은 임의의 짝수 개의 비트가 바뀌면 오류를 검출해 내지 못함

문 자	패리티 비트
0 0 0 0 0 0 0	0
0 0 0 0 0 0 1	1
0 0 0 0 0 1 0	1
0 0 0 0 0 1 1	0

그림 2.28 짝수 패리티 문자의 전송 예



홀수 패리티의 예

문자 표현

표 3-3 아스키코드 일부

10 진수	16 진수	부호	10 진수	16 진수	부호	10 진수	16 진수	부호	10 진수	16 진수	부호
032	20	SP	056	38	8	080	50	P	104	68	h
033	21	!	057	39	9	081	51	Q	105	69	i
034	22	"	058	3A	:	082	52	R	106	6A	j
035	23	#	059	3B	;	083	53	S	107	6B	k
036	24	$	060	3C	〈	084	54	T	108	6C	l
037	25	%	061	3D	=	085	55	U	109	6D	m
038	26	&	062	3E	〉	086	56	V	110	6E	n
039	27	'	063	3F	?	087	57	W	111	6F	o
040	28	(	064	40	@	088	58	X	112	70	p

문자 표현

표 3-3 아스키코드 일부(계속)

10 진수	16 진수	부호	10 진수	16 진수	부호	10 진수	16 진수	부호	10 진수	16 진수	부호
041	29	)	065	41	A	089	59	Y	113	71	q
042	2A	*	066	42	B	090	5A	Z	114	72	r
043	2B	+	067	43	C	091	5B	[	115	73	s
044	2C	,	068	44	D	092	5C	₩	116	74	t
045	2D	-	069	45	E	093	5D	]	117	75	u
046	2E	.	070	46	F	094	5E	^	118	76	v
047	2F	/	071	47	G	095	5F	_	119	77	w
048	30	0	072	48	H	096	60	`	120	78	x
049	31	1	073	49	I	097	61	a	121	79	y
050	32	2	074	4A	J	098	62	b	122	7A	z
051	33	3	075	4B	K	099	63	c	123	7B	{
052	34	4	076	4C	L	100	64	d	124	7C	|
053	35	5	077	4D	M	101	65	e	125	7D	}
054	36	6	078	4E	N	102	66	f	126	7E	~
055	37	7	079	4F	O	103	67	g	127	7F	DEL

문자 표현

■ 아스키코드의 저장과 변환

- 사람이 'YOU'라고 입력하면 컴퓨터는 '89, 79, 85'의 2진수 값 저장
- 저장된 내용을 사용자에게 보여 줄 때는 숫자를 다시 문자로 바꾸어 보여줌

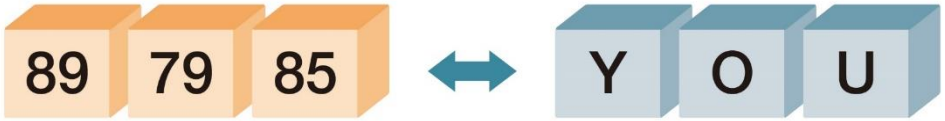


그림 3-16 아스키코드의 저장과 변환

문자 표현

■ 유니코드

- 전 세계 모든 문자를 컴퓨터에서 일관되게 표현하고 다룰 수 있도록 설계된 산업 표준 문자코드
- 10만 개가 넘는 문자를 표현할 수 있기 때문에 대부분의 언어 표현이 가능
- 한글 '가'의 유니코드 값은 AC00,
  컴퓨터 내부에는 2진수 1010110000000000으로 저장

	AC0	AC1	AC2	AC3	AC4	AC5	AC6	AC7	AC8	AC9	ACA	ACB	ACC	ACD	ACE	ACF
0	가 AC00	감 AC10	갠 AC20	갰 AC30	걀 AC40	걬 AC50	걠 AC60	거 AC70	검 AC80	겐 AC90	겠 ACA0	결 ACB0	겹 ACC0	곀 ACD0	고 ACE0	곰 ACF0
1	각 AC01	갑 AC11	갡 AC21	갱 AC31	걁 AC41	걑 AC51	걡 AC61	건 AC71	겁 AC81	겑 AC91	겡 ACA1	겱 ACB1	곁 ACC1	곡 ACD1	곡 ACE1	곱 ACF1
2	갂 AC02	값 AC12	갢 AC22	갲 AC32	걂 AC42	걒 AC52	걢 AC62	걲 AC72	겂 AC82	겒 AC92	겢 ACA2	겹 ACB2	경 ACC2	곲 ACD2	곢 ACE2	곲 ACF2

그림 3-17 한글 유니코드

문자 표현

UTF-8

- 아스키 코드와 호환 가능
- 유니코드를 인코딩(Encoding)하는 방법
  - **인코딩(Encoding)** 이란 컴퓨터가 이해할 수 있는 형태로 바꾸어주는 것
- 가변형 인코딩 방식(1byte ~ 4byte)
- 가변을 구분하기 위해 첫 바이트에 표식을 넣었는데 2 byte는 110으로 시작하고, 3바이트는 1110으로 시작한다.
- 나머지는 10으로 시작

아스키(ASCII) 문자들은 변형없이 1바이트에 그대로 인코딩하고, 중동과 유럽 지역의 언어는 2바이트, 한글을 포함한 아시아권은 3byte 이상으로 인코딩합니다.

UTF-16

- 16비트 기반으로 문자 인코딩, 한글 또한 2바이트로 저장

문자 표현

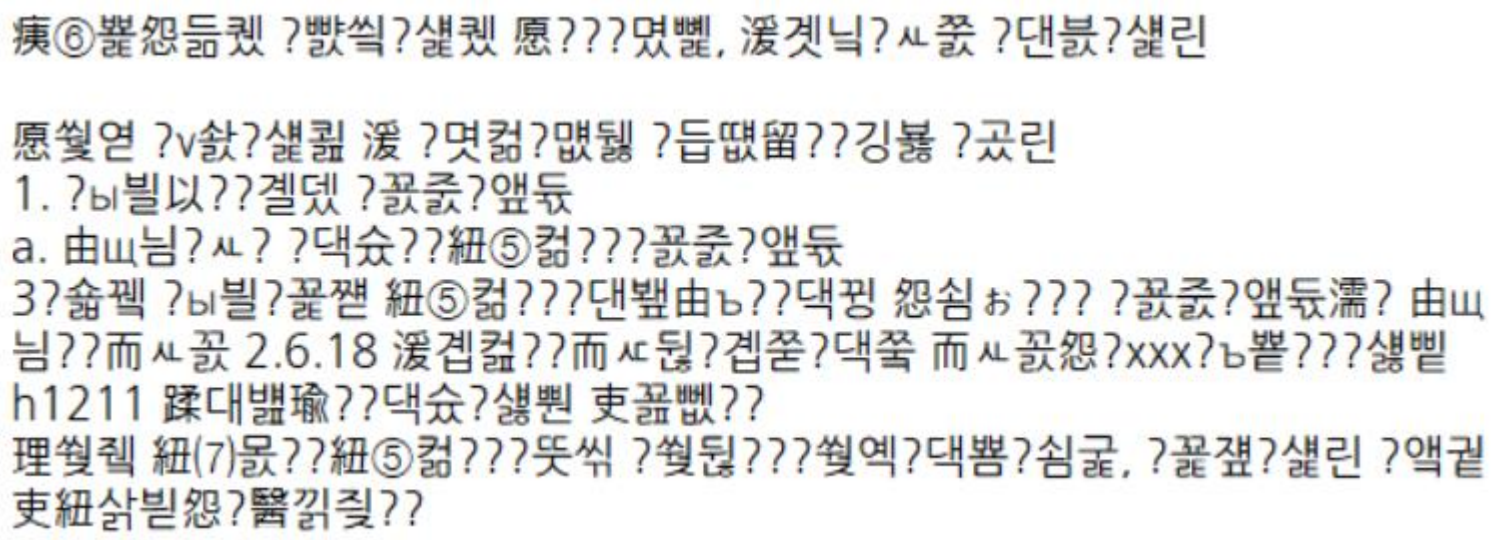
EUC-KR(MS949, CP949)

- 한국에서 독자적으로 사용하는 인코딩 방식
- 완성된 상태의 문자가 2바이트 값으로 정해져서 표현하는 방식
- 영어 같은 1바이트 문자를 사용하는 곳이 아닌 한글, 일본어, 중국어 같은 2바이트 문자를 사용하는 곳에서 많이 사용된다.

문자 표현

?

- 대체 이런 상황은 왜 발생하는 것일까요? 그 이유는 컴퓨터에서 한글을 표현하는 다양한 문자열 셋과 인코딩 방식들이 맞지 않기 때문입니다.



정수 표현

정수란?

- 컴퓨터가 사용하는 숫자 중 가장 기본 값으로 1, 2, 3, 4, …처럼 셀 수 있는 수(int)

컴퓨터의 정수 표현

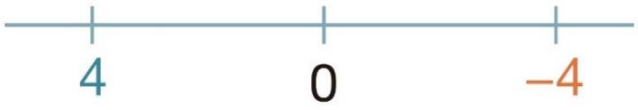


그림 3-18 양수 4와 음수 -4

- 양수는 2진법을 그대로 사용

  [예] 4 표기 : 0100

정수 표현

■ 컴퓨터의 정수 표현

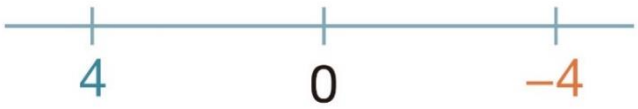


그림 3-18 양수 4와 음수 -4

- 음수는 1의 보수 또는 2의 보수로 표현
  - 1의 보수 : 0은 1로, 1은 0으로 바꿈
  - 2의 보수 : 1의 보수+1

  [예] -4 표기 : 1의 보수 1011, 2의 보수 1100

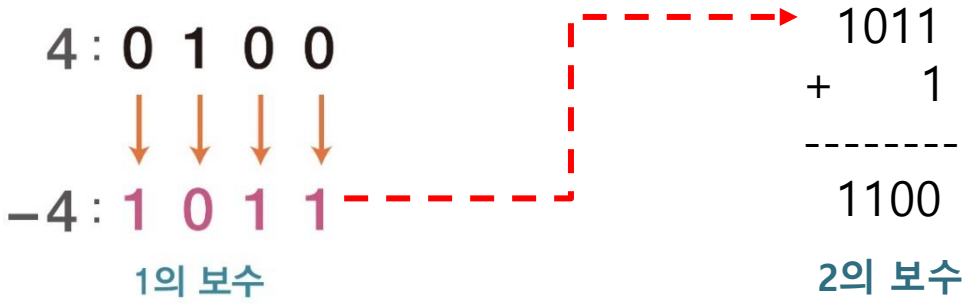


그림 3-19 2진수 0100을 음수로 만드는 과정

정수 표현

■ 컴퓨터의 정수 표현

- 1의 보수
  - 숫자 0이 +0과 −0으로 2개가 됨
  - 정수 값의 범위 : 7 ~ −7
- 2의 보수
  - −0을 없애고 그 밑의 음수가 한 칸씩 올라옴
  - 정수 값의 범위 : 7 ~ −8

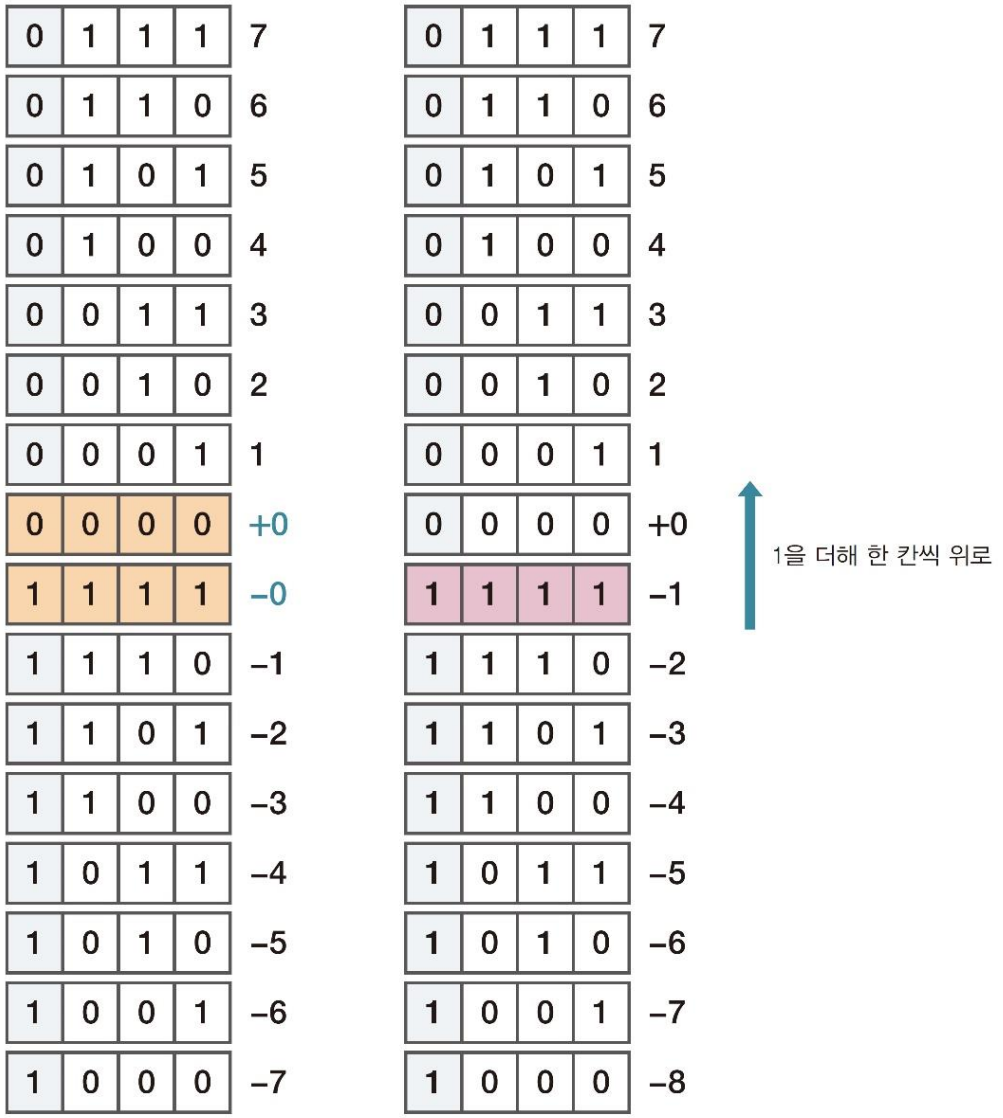
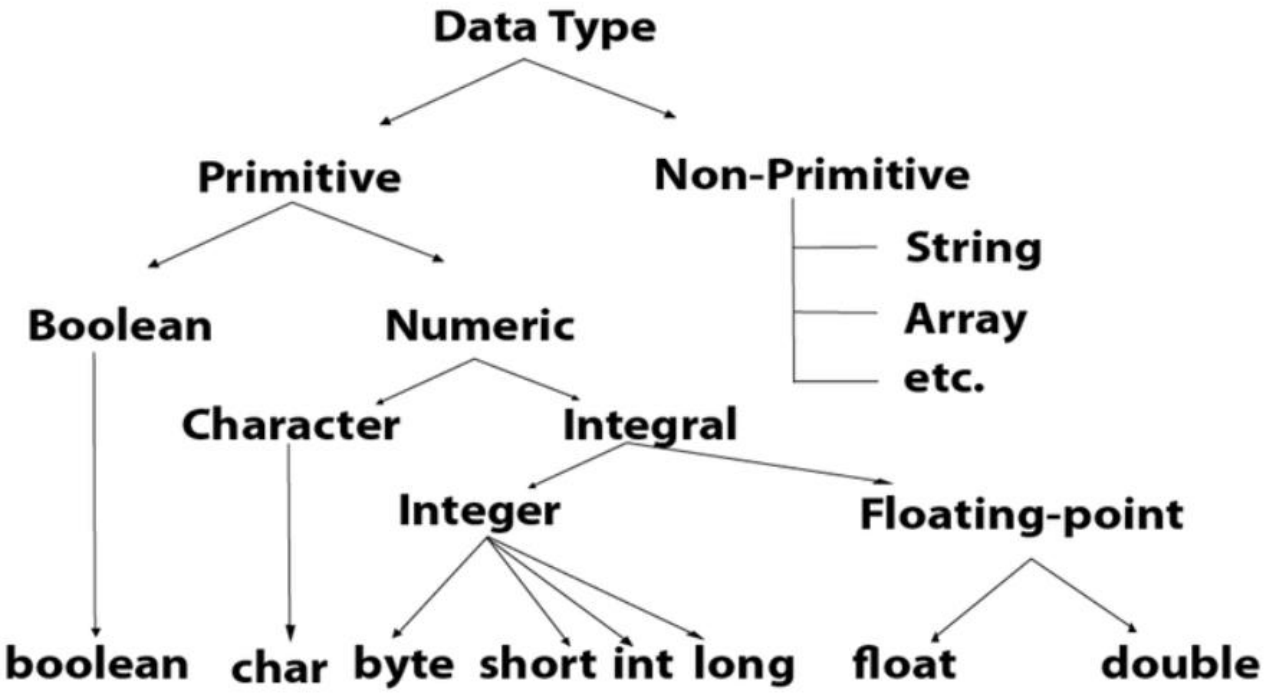


그림 3-20 4비트로 표현할 수 있는 수의 범위

정수 표현

■ 자바의 정수 표현

- byte(1byte) : -128 ~ 127
- short(2byte) : -32,768 ~ 32,767
- int(4byte) : -2,147,483,648 ~ 2,147,483,647(약 21억)
- double(8byte) : -9,223,372,036,854,770,808 ~ 9,223,372,036,854,770,807(약9백경)



실수 표현

실수란?

- 소수점 이하의 자리가 존재하는 숫자(float)

컴퓨터의 실수 표현

- 아주 작은 수부터 매우 큰 수까지 표현할 수 있기 때문에 다루기가 까다로워 정규화하여 표현
- 정규화(normalization)
  - 숫자를 일정한 단위로 맞추는 작업

그림 3-21 정수의 정규화

실수 표현

■ 컴퓨터의 실수 표현

- 실수의 정규화
  - 모든 수를 0.XXXX로 표현(예: 72300000 → 0.723×10^8)
  - 정규화된 수에서 0 이하 숫자와 지수만 저장하면 아무리 큰 수라도 컴퓨터에 간단히 저장할 수 있음
  - 정규화된 표현에서 소수점 아래 숫자를 가수 혹은 멘티사(mentisa)라 함

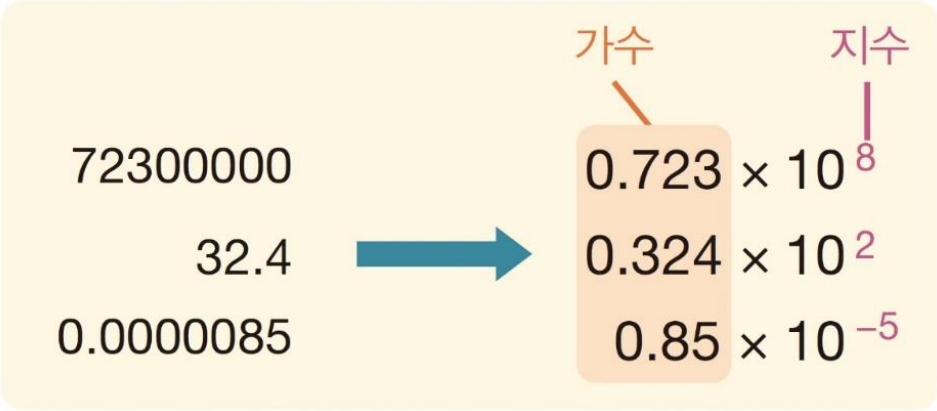


그림 3-22 실수의 정규화

실수 표현

■ 정규화된 실수가 컴퓨터에 저장되는 과정 – 사례 1

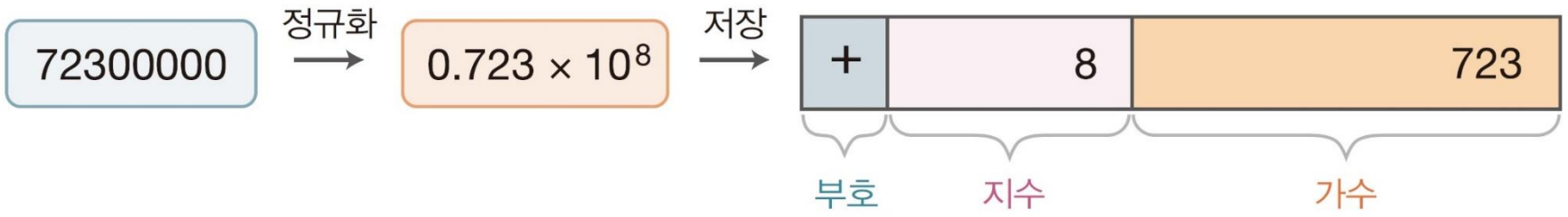


그림 3-23 정규화된 실수 0.723×10^8 저장

실수 표현

■ 정규화된 실수가 컴퓨터에 저장되는 과정 – 사례 2, 사례 3

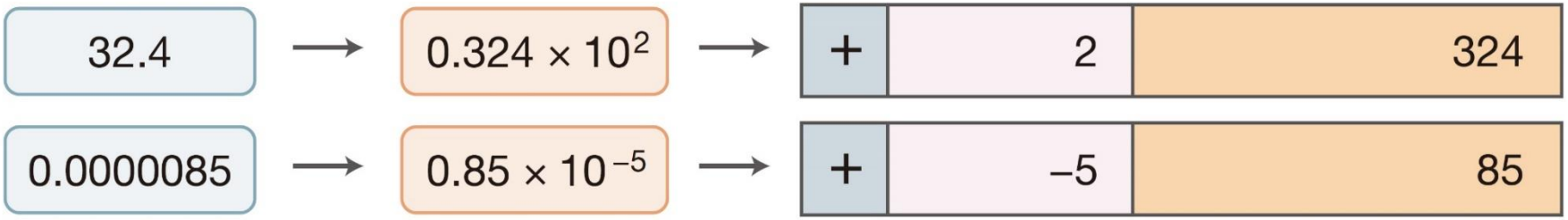


그림 3-24 정규화된 실수 0.324×10^2과 0.85×10^{-5} 저장

자바 실수 표현

■ 정규화된 실수가 컴퓨터에 저장되는 과정 – 사례 2, 사례 3

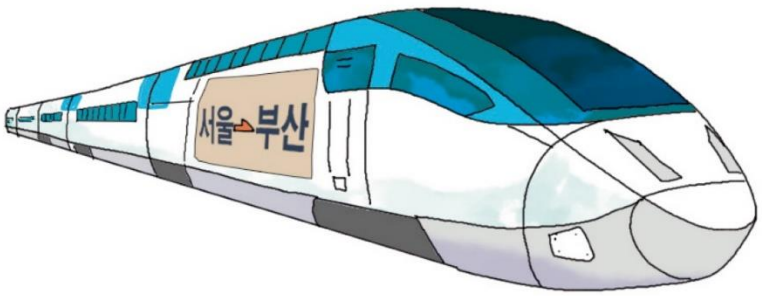
타입	저장 가능한 값의 범위(양수)	정밀도	크기	
			bit	byte
float	$1.4 \times 10^{-45} \sim 3.4 \times 10^{38}$	7자리	32	4
double	$4.9 \times 10^{-324} \sim 1.8 \times 10^{308}$	15자리	64	8

논리 연산자

AND 연산자

- 두 조건이 모두 참(true)일 때만 결과가 참

영자 그리고(AND) 미숙이 가면, 부산 간다.



영자	미숙	부산
OK	OK	간다
OK	NO	못 간다
NO	OK	못 간다
NO	NO	못 간다

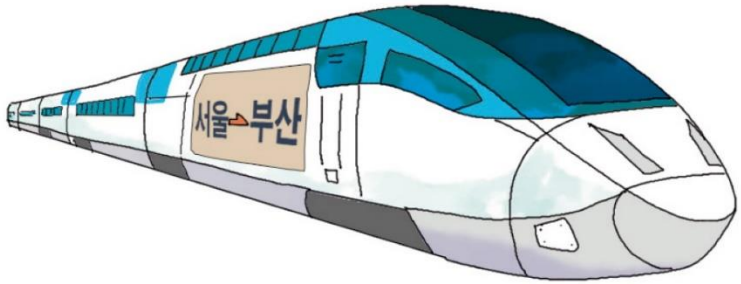
그림 3-25 AND 논리 연산

논리 연산자

OR 연산자

- 두 조건이 모두 거짓(false)일 때만 결과가 거짓

영자 혹은(OR) 미숙이 가면, 부산 간다.



영자	미숙	부산
OK	OK	간다
OK	NO	간다
NO	OK	간다
NO	NO	못 간다

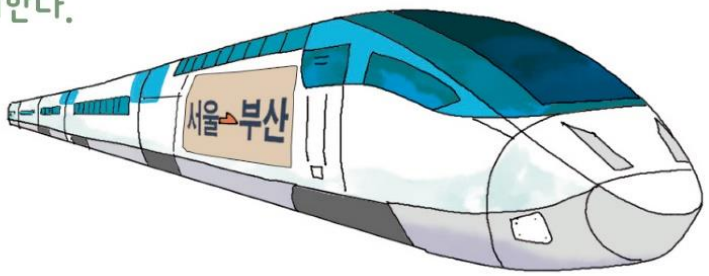
그림 3-26 OR 논리 연산

논리 연산자

XOR 연산자

- 두 조건이 서로 다를 때만 결과가 참

영자나 미숙이 둘 중 하나 하고만(XOR) 부산 가든가 아니면 포기한다.



영자	미숙	부산
OK	OK	못 간다
OK	NO	간다
NO	OK	간다
NO	NO	못 간다

그림 3-27 XOR 논리 연산 결과

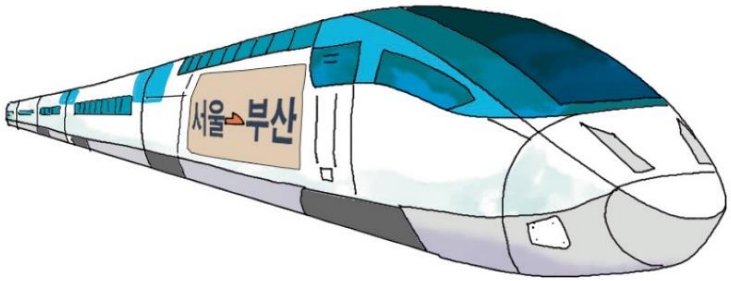
논리 연산자

NOT 연산자

- 참과 거짓을 바꾸어 주는 연산

미숙이와 반대로(NOT) 한다.

그림 3-28 NOT 논리 연산 결과



미숙	부산
OK	못 간다
NO	간다

논리 연산자

■ XOR을 이용한 교체 알고리즘

a = 1(0001), b =2(0010) 일 때

① a = a xor b

a = 0011

② b = a xor b

b = 0001

③ a = a xor b

a = 0010

	0 0 0 1
XOR	0 0 1 0
=	0 0 1 1
	0 0 1 1
XOR	0 0 1 0
=	0 0 0 1
	0 0 1 1
XOR	0 0 0 1
=	0 0 1 0

논리 연산자

■ 정리

- AND 연산 : 값 2개가 모두 참일 때만 참이 되는 연산
- OR 연산 : 값 2개 중 하나라도 참이면 참이 되는 연산
- XOR 연산 : 값 2개 중 하나라도 다르면 참이 되며, 값 2개가 같으면 거짓이 되는 연산
- NOT 연산 : 반대 값을 만드는 연산

표 3-4 논리 연산의 진리표

AND 연산			OR 연산			XOR 연산			NOT 연산	
입력		출력	입력		출력	입력		출력	입력	출력
T	T	T	T	T	T	T	T	F	T	F
T	F	F	T	F	T	T	F	T	F	T
F	T	F	F	T	T	F	T	T		
F	F	F	F	F	F	F	F	F		

Thank you!