

컴개실 PPT HW1

컴퓨터 내부에서 데이터의 표현 방법

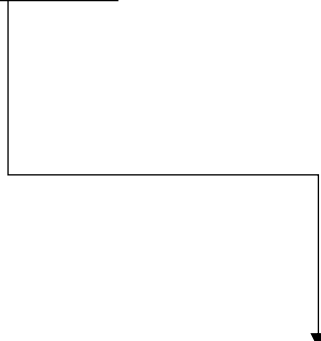
13조 조강현 백승우 박재완2 배수민

목차

- 01 ● 정수, 아주 큰 정수의 표현방법
- 02 ● 실수, 높은 정확도가 요구되는 실수의 표현방법
- 03 ● 알파벳과 한글의 표현방법

01 정수의 이진법 표현

기본적으로 컴퓨터 내에서 숫자는
이진법으로 표현됨.



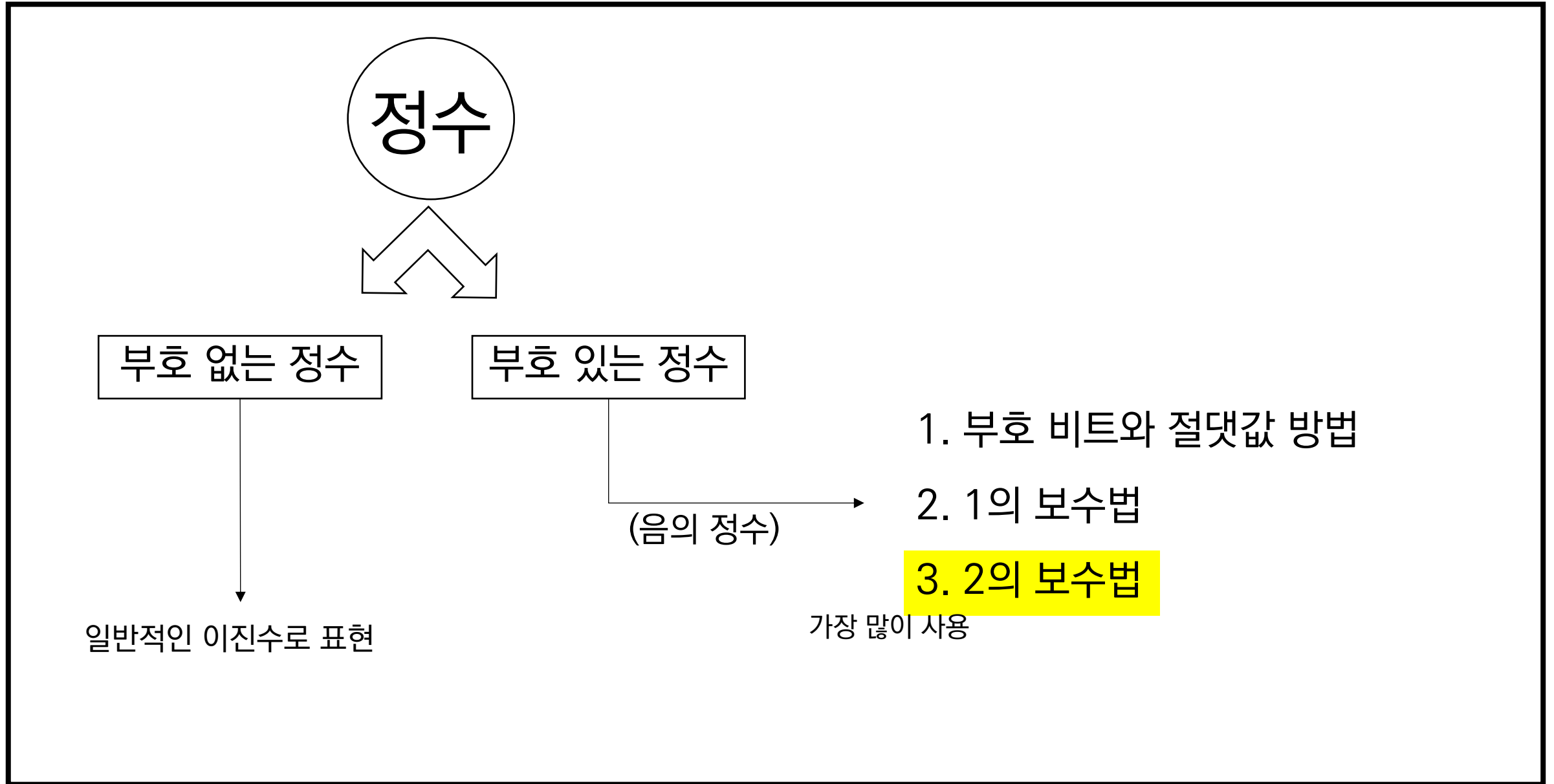
0과 1로 모든 수를 표현하는 방식

예시

$$87 \quad (87=64+16+4+2+1)$$
$$= 1010111_{(2)}$$

- (1) 87보다 작은 수 중 가장 큰 2의 제곱수 찾기 : $64=2^8$
- (2) 87에서 64를 뺀 값보다 작은 수 중 가장 큰 2의 제곱수 찾기
- (3) (2)의 과정을 0이나 1이 나올 때 까지 반복

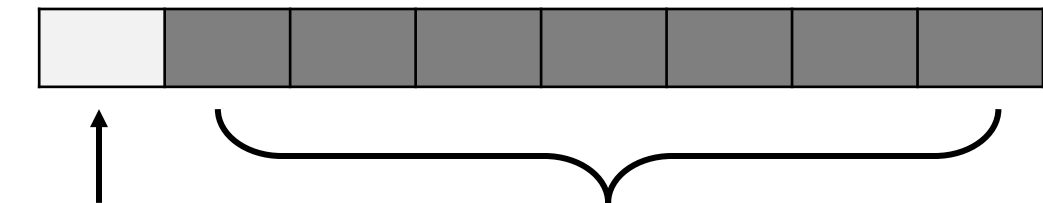
01 정수의 이진법 표현



01 음의 정수의 표현 - 부호 비트와 절댓값 방법

부호 비트와 절댓값 방법 : 최상위 1비트로 부호를 표현, 나머지 비트로 절댓값을 표현

8 bit 체계를 이용하는 경우,



부호 비트

양수: 0
음수: 1

정수부 (7 bit)

특징 : 나타낼 수 있는 수들의 절댓값 범위가 반으로 준다 !

문제 : 간편하지만, +0과 -0이 따로 존재하는 문제가 있음 !

(예시) 87의 부호 비트 표현



양수 2^6 + 2^4 + 2^2 + 2^1 + 2^0 =

87

01 음의 정수의 표현 - 1의 보수법

1의 보수법: 해당 양수의 모든 비트를 반전하여 음수를 표현하는 방법

1. 양의 정수 87을 이진법으로 표현하면, —————→

0	1	0	1	0	1	1	1
---	---	---	---	---	---	---	---



2. 절댓값이 같고 부호가 다른 -87을 1의 보수법으로 표현하면,

1	0	1	0	1	0	0	0
---	---	---	---	---	---	---	---

문제 : 간편하지만, +0과 -0이 따로 존재하는 문제가 있음 !

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

+0

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

-0

01 음의 정수의 표현 - 2의 보수법 (1)

2의 보수법: 1의 보수법으로 표현한 이진수에 1을 더하여 표현하는 방법

1. 양의 정수 32를 이진법으로 표현하면, \longrightarrow

0	0	1	0	0	0	0	0
---	---	---	---	---	---	---	---

 =32

2. 절댓값이 같고 부호가 다른 -32를 1의 보수법으로 표현하면,

1	1	0	1	1	1	1	1
---	---	---	---	---	---	---	---

 =-32

3. 1의 보수법으로 표현한 이진수에 1을 더하여 표현하면,

1	1	1	0	0	0	0	0
---	---	---	---	---	---	---	---

 =-32

01 음의 정수의 표현 - 2의 보수법 (2)

부호 비트와 절댓값 방법

1의 보수법

P: 0의 값이 +0과 -0 두 가지로 존재

S: 2의 보수법

P.S. 이 때문에 다수의 시스템에서는
현재 2의보수법 사용 중



0	0	1	0	0	0	0	0
---	---	---	---	---	---	---	---

(+32)

1	1	0	1	1	1	1	1
---	---	---	---	---	---	---	---

(-32)

1	0	0	0	0	0	0	0
--------------	---	---	---	---	---	---	---

최상위 비트에서 오버플로우 발생
8비트로 표현 시 자연스럽게 0을 나타냄.



문제 해결 !

01 메모리 저장 방식

Big-endian vs Little-endian

컴퓨터 메모리에 바이트(Byte)를 배열하는 두 방법

예시) 메모리에 0x12345678을 대입

Big-endian : 큰 단위의 바이트가 앞에

 메모리 주소 증가

메모리 주소	...	0x100	0x101	0x102	0x103	...
		0x12	0x34	0x56	0x78	

Little-endian : 작은 단위의 바이트가 앞에

 메모리 주소 증가

메모리 주소	...	0x100	0x101	0x102	0x103	...
		0x78	0x56	0x34	0x12	

01 정수형의 종류

정수 자료형은 메모리에서 차지하는 크기와 부호 유무(signed vs unsigned)에 따라 나뉜다.

자료형	크기(Byte)	크기(Bit)	범위
short	2	16	-32,768~32,767
unsigned short	2	16	0~65,535
int	4	32	-2,147,483,648~ 2,147,483,647
unsigned int	4	32	0~4,294,967,295
long	4	32	-2,147,483,648~ 2,147,483,647
unsigned long	4	32	0~4,294,967,295
long long	8	64	-9,223,372,036,854,775,808~ 9,223,372,036,854,775,807
unsigned long long	8	64	0~18,446,744,073,709,551,615

o C++ 기준

02 실수의 이진법 표현

고려사항 1 : 정수 부분과 소수 부분의 표현

11.625

정수 표현 방법과 동일

$$\begin{aligned} 11 &= 8 + 2 + 1 \\ &= 1011_{(2)} \end{aligned}$$

$$\begin{array}{l} 11 \div 2 = 5 \cdots 1 \\ 5 \div 2 = 2 \cdots 1 \\ 2 \div 2 = 1 \cdots 0 \end{array}$$

1011

정수 표현 방법과 반대

$$\begin{aligned} 0.625 &= 0.5 + 0.125 \\ &= 0.101_{(2)} \end{aligned}$$

$$\begin{array}{l} 0.625 \times 2 = 0.25 + 1 \\ 0.25 \times 2 = 0.5 + 0 \\ 0.5 \times 2 = 1 \end{array}$$

.101

02 실수의 이진법 표현

고려사항 2 : 부호 및 소수점의 표현

16비트 체계를 이용하는 경우,



부호 비트
0(양수) 또는 1(음수)

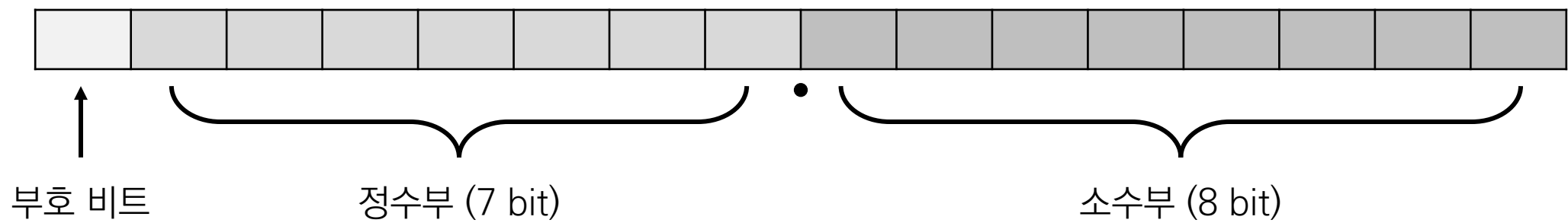
숫자만을 비트 속에 저장하기 때문에
어느 위치에 소수점이 있을 것인가에 관한 약속이 필요함

소수점을 어떻게 표현하는가에 따라
고정 소수점 (Fixed Point) 방식과
부동 소수점 (Floating Point) 방식으로 나뉨

02 소수점의 표현 – 고정 소수점 방식

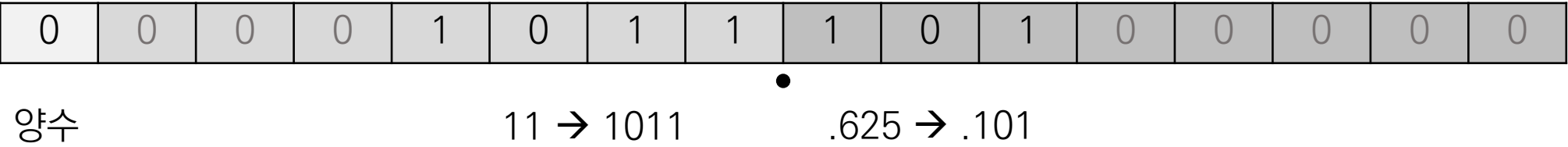
고정 소수점 방식 : 정수부와 소수부의 자릿수를 정하여 실수를 표현하는 방법

16 bit 체계를 이용하는 경우,



위와 같이 소수점의 위치를 고정하여 사용할 수 있다.

(예시) 11.625의 고정 소수점 표현



02 소수점의 표현 – 고정 소수점 방식

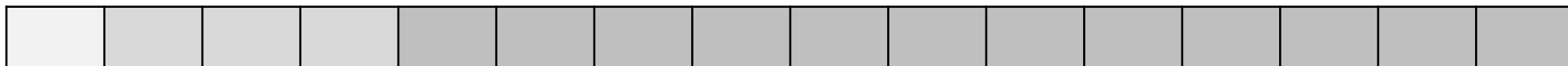
그러나, 고정 소수점 방식을 이용하는 경우 실수의 표현 범위가 작아지는 문제 발생

정수부의 비트 수를 증가시키면



큰 수를 표현하기에는 좋지만, 정밀한 소수는 표현하기가 어려움

소수부의 비트 수를 증가시키면



정밀한 소수를 표현하기에는 좋지만, 큰 수를 표현하기는 어려움

따라서 실수를 많이 다뤄야 하는 시스템에서는 고정 소수점 방식을 주로 이용하지 않음
→ 부동 소수점 방식 이용

02 소수점의 표현 - 부동 소수점 방식 (정규 값)

부동 소수점 방식 : 소수점 위치가 고정되지 않은 채로 실수를 표현하는 방법, 주로 IEEE 754의 표준안을 따름

IEEE 754의 정규화 인코딩은 실수를 부호(sign), 가수(mantissa), 지수(exponent)로 나누어 표현한다.



(예시) 11.625의 표현

$$11.625 = 1011.101_{(2)} = (-1)^0 \times 1.011101 \times 2^3$$

(예시) -0.1875의 표현

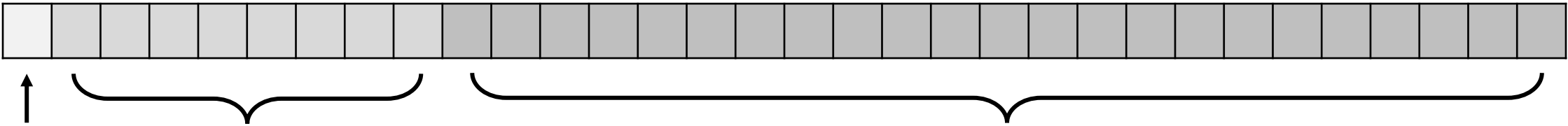
$$-0.1875 = -0.0011_{(2)} = (-1)^1 \times 1.1 \times 2^{-3}$$

컴퓨터에 저장 시에는 부호, 지수, 가수를 각각 지정된 비트에 저장한다.

실수 표현의 정밀도에 따라
단정도(single precision), 배정도(double precision)의 두 가지 방식 이용

02 소수점의 표현 – 부동 소수점 방식 (정규 값)

단정도 방식은 32 bit 체계에서 실수를 비교적 낮은 정밀도로 표현한다. (배정도 방식은 64 bit 체계에서 더 높은 정밀도로 표현한다.)



부호 (1 bit)

지수 (8 bit, 배정도는 11 bit)

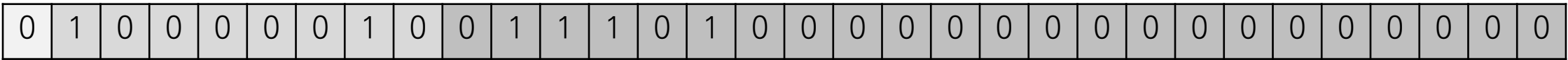
가수 (23 bit, 배정도는 52 bit)

- 실제 지수에 127(01111111)을 더한 값을 저장
- 127은 바이어스(bias)로, 양의 지수와 음의 지수를 모두 표현하기 위해 도입됨 (배정도는 1023)

- 1.을 생략한 가수의 소수부분을 저장

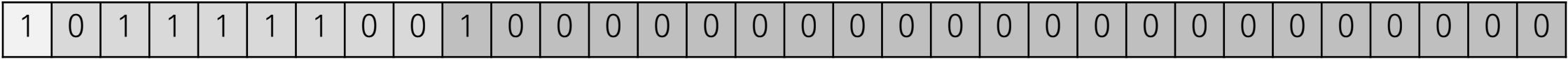
(예시) 11.625의 표현 (단정도)

$(-1)^0 \times 1.011101 \times 2^3 \rightarrow$ 부호 : 0, 지수 : $3 + 127 = 130$ (10000010), 가수 : 011101



(예시) -0.1875의 표현 (단정도)

$(-1)^1 \times 1.1 \times 2^{-3} \rightarrow$ 부호 : 1, 지수 : $-3 + 127 = 124$ (01111100), 가수 : 1



02 소수점의 표현 - 부동 소수점 방식 (비정규 값, 특수 값)

정규화 인코딩은 일반적인 실수의 표현에 사용되지만, 0 근처의 매우 작은 수나 매우 큰 수 등은 다른 방법으로 인코딩한다.

비정규 값 : 0 근처의 절댓값이 매우 작은 수를 표현하기 위한 인코딩 방법

지수 : 000...0으로 고정

가수 : 수를 0.xxx... 형태로 나타낸 뒤, 0.을 생략한 부분을 저장

(예시) 32bit에서 표현 가능한 가장 작은 양수

[illegible]

[illegible]

특수 값 : 무한, NaN(Not a Number) 등의 특별한 수들을 표현하기 위한 인코딩 방법

지수 : $111\cdots 1$ 로 고정

가수 : 각 수에 따라 약속된 값이 있음

$$(|\mathcal{Y}|) + \infty, -\infty$$
[illegible]

02 소수점의 표현 – 부동 소수점 방식

부동 소수점 > 고정 소수점

↑
수의 범위, 정밀도

하지만 여전히 한계는 존재한다.

부동 소수점 방식이 범용적

- 실수 자료형의 종류

float	32bit 단정도 방식
double	64bit 배정도 방식

○ C++ 기준

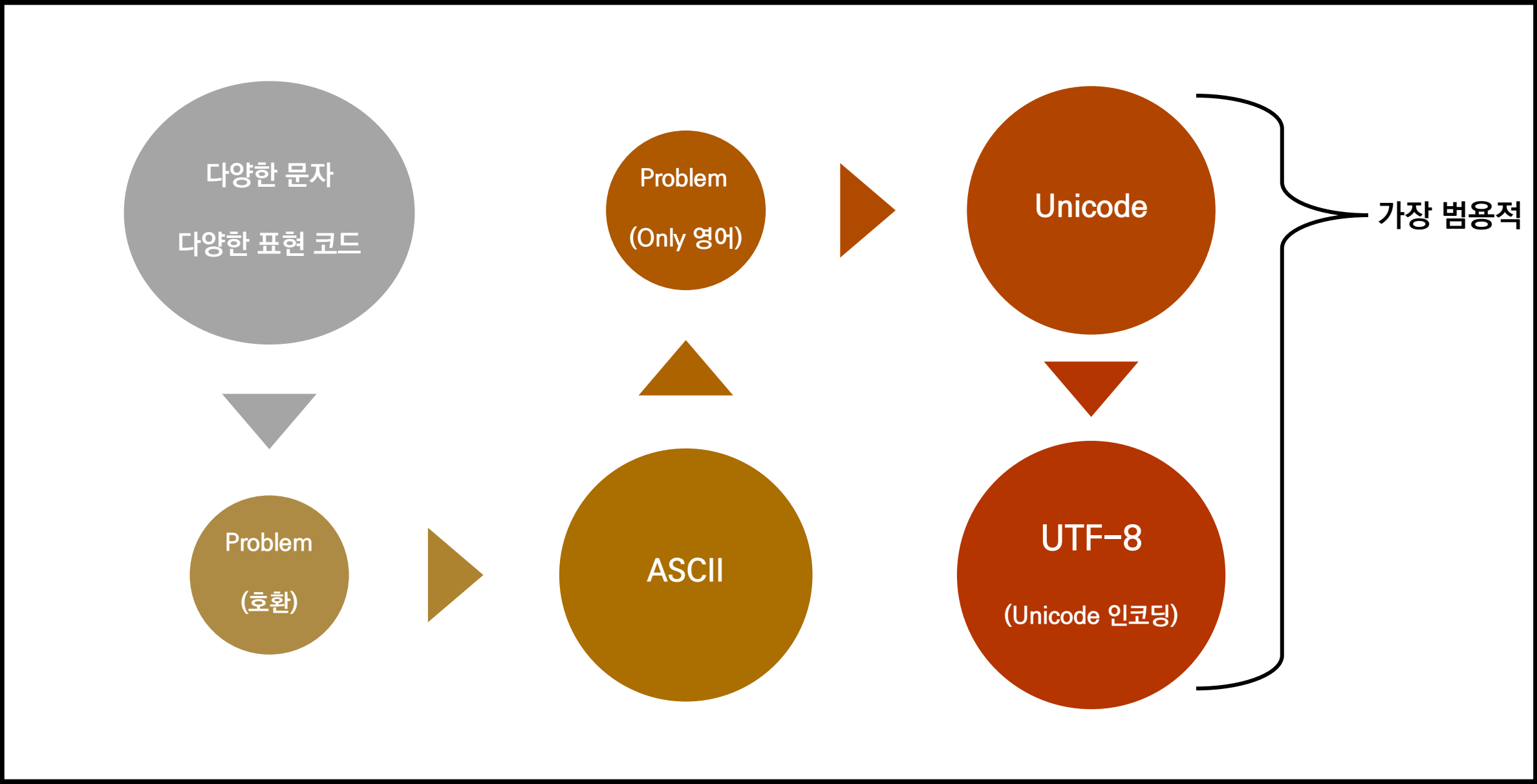
부동 소수점 오차

0.625 = 0.101₍₂₎ : 정확하게 표현 가능
0.3 = 0.0100110011001100...₍₂₎ : 무한히 반복, 근사치만 저장 가능

(예시) 자명한 식에서도 오차가 생길 수 있다.

>>> 0.1+0.1+0.1
0.30000000000000004

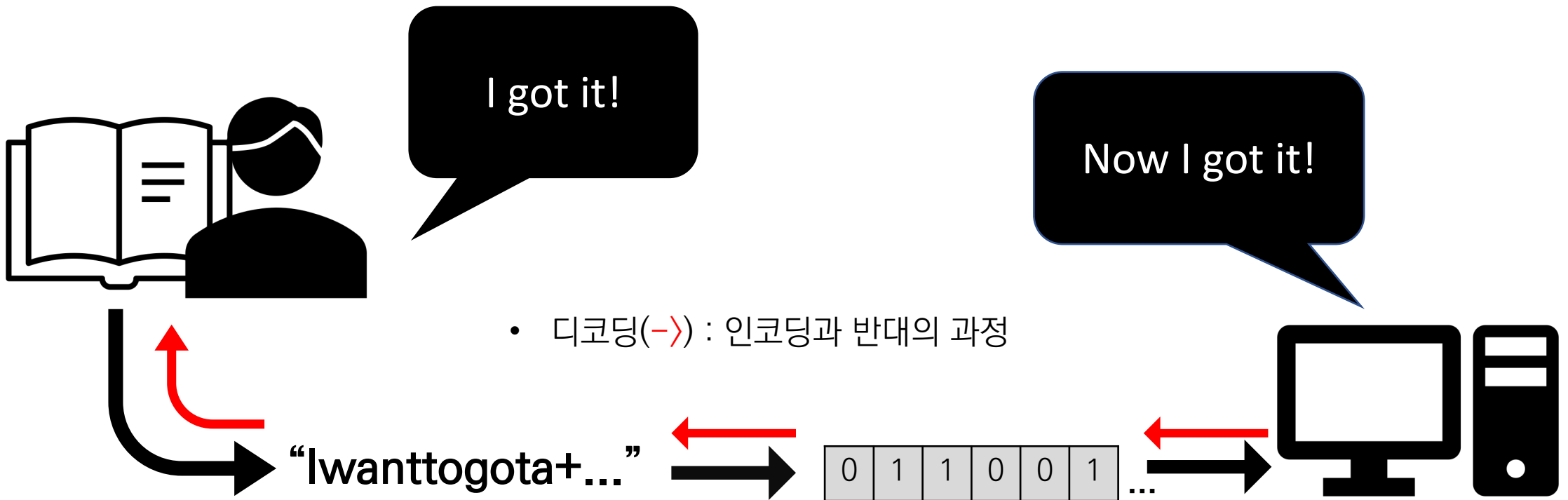
03 문자의 표현 - 문자 표현 코드의 역사



03 인코딩이란

정보의 형태나 형식을 변환하는 것

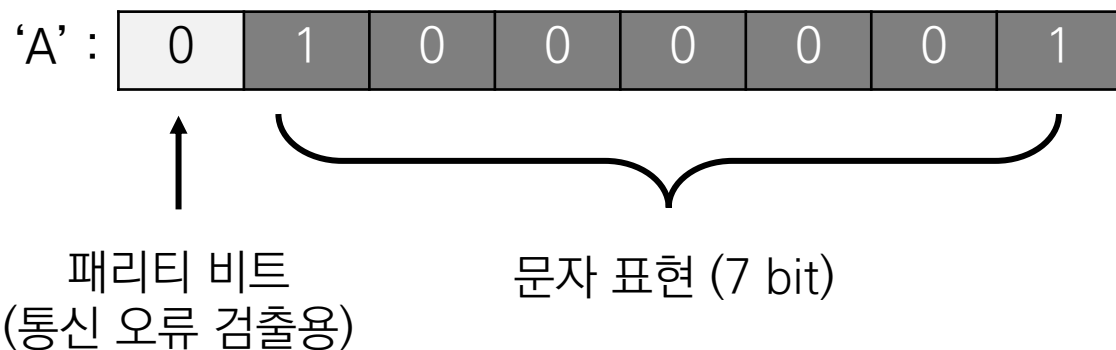
in Computer Science, 사람이 인지할 수 있는 데이터의 형태를 컴퓨터가 인지할 수 있도록 바꾸어주는 과정



03 ASCII 코드

- 미국 정보 교환 표준 부호
- 알파벳, 숫자, 특수문자, 제어문자 표현 가능
- 7bit로 128개의 글자 표현 가능

아스키코드의 구조



✓영어 이외의 글자 표현이 어렵다는 단점 존재

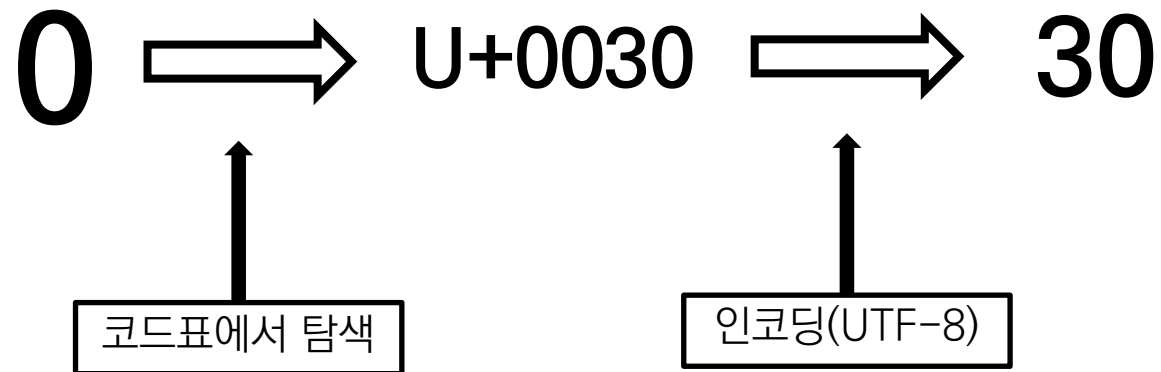
ASCII Table

Hex	Dec	ASCII	Hex	Dec	ASCII	Hex	Dec	ASCII	Hex	Dec	ASCII
00	0	NULL	20	32	SP	40	64	@	60	96	,
01	1	SOH	21	33	!	41	65	A	61	97	a
02	2	STX	22	34	"	42	66	B	62	98	b
03	3	ETX	23	35	#	43	67	C	63	99	c
04	4	END	24	36	\$	44	68	D	64	100	d
05	5	ENQ	25	37	%	45	69	E	65	101	e
06	6	ACK	26	38	&	46	70	F	66	102	f
07	7	BEL	27	39	'	47	71	G	67	103	g
08	8	BS	28	40	(48	72	H	68	104	h
09	9	HT	29	41)	49	73	I	69	105	i
0A	10	LF	2A	42	+	4A	74	J	6A	106	j
0B	11	VT	2B	43	+	4B	75	K	6B	107	k
0C	12	FF	2C	44	,	4C	76	L	6C	108	l
0D	13	CR	2D	45	-	4D	77	M	6D	109	m
0E	14	SO	2E	46	.	4E	78	N	6E	110	n
0F	15	SI	2F	47	/	4F	79	O	6F	111	o
10	16	DLE	30	48	0	50	80	P	70	112	p
11	17	DC1	31	49	1	51	81	Q	71	113	q
12	18	SC2	32	50	2	52	82	R	72	114	r
13	19	SC3	33	51	3	53	83	S	73	115	s
14	20	SC4	34	52	4	54	84	T	74	116	t
15	21	NAK	35	53	5	55	85	U	75	117	u
16	22	SYN	36	54	6	56	86	V	76	118	v
17	23	ETB	37	55	7	57	87	W	77	119	w
18	24	CAN	38	56	8	58	88	X	78	120	x
19	25	EM	39	57	9	59	89	Y	79	121	y
1A	26	SUB	3A	58	:	5A	90	Z	7A	122	z
1B	27	ESC	3B	59	:	5B	91	[7B	123	{
1C	28	FS	3C	60	<	5C	92	¥	7C	124	
1D	29	GS	3D	61	=	5D	93]	7D	125	}
1E	30	RS	3E	62	>	5E	94	^	7E	126	~
1F	31	US	3F	63	?	5F	95	_	7F	127	DE

03 유니코드

- 전세계의 모든 문자를 다루는 표준 방식
- ASCII의 단점을 해결
- 전세계의 모든 문자를 담는 ISO/IEC 10646 코드표 사용
- 문자에 해당하는 코드를 인코딩

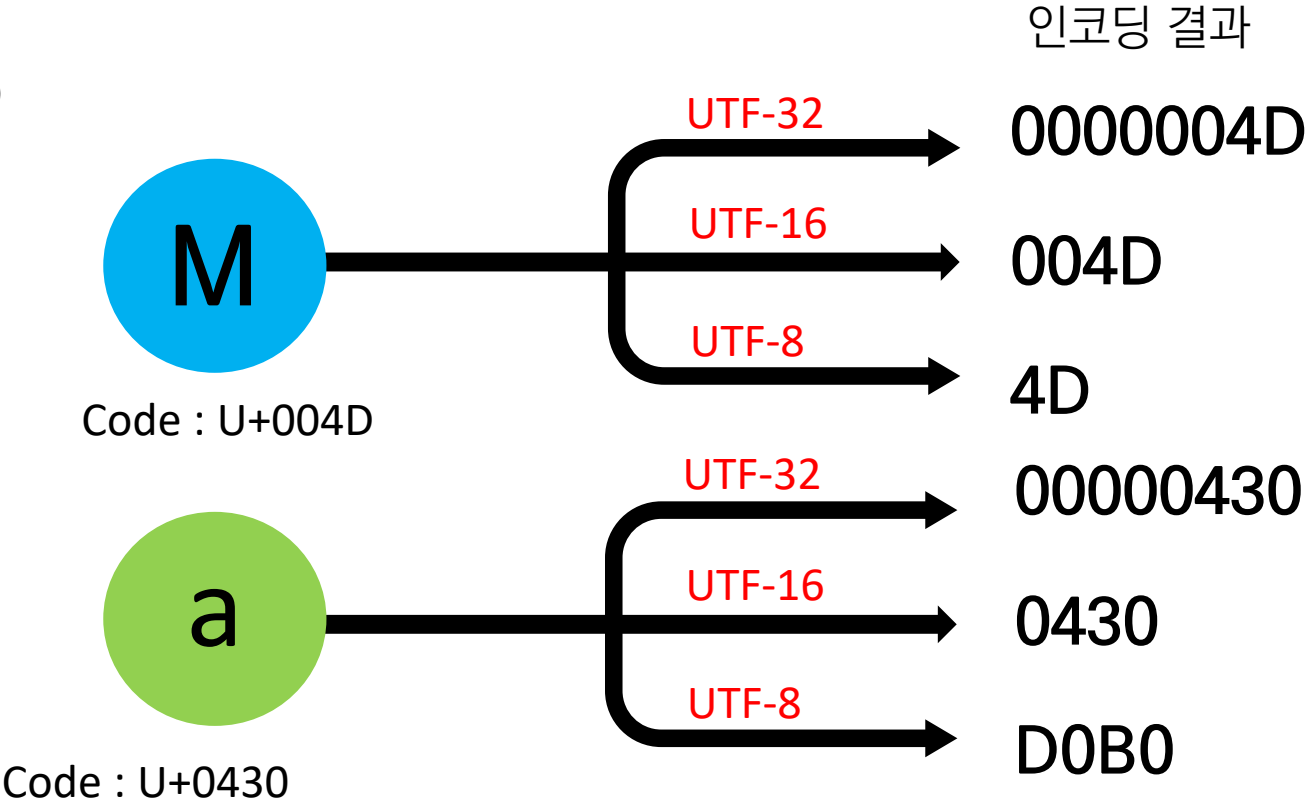
예시)



03 유니코드 – 다양한 인코딩

- 동일한 코드를 가지더라도, 인코딩 방식에 따라 표현되는 형식이 다르다.

예시)



03 UTF-8

가장 범용적으로 사용되는 유니코드 인코딩

- 길이가 일정하지 않은 가변 길이 인코딩 (1~4 바이트)
- 한 Byte마다 처음 몇 개의 Bit에 길이와 관련된 값을 넣는다.

코드값의 자릿수	범위	1 st Byte	2 nd Byte	3 rd Byte	4 th Byte
7비트	0~0x7F	0xxxxxxx			
11비트	0x80~0x7FF	110xxxxx	10xxxxxx		
16비트	0x800~0xFFFF	1110xxxx	10xxxxxx	10xxxxxx	
21비트	0x10000~0x1FFFFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

✓ 0~127번까지의 아스키코드는 7비트로 표현 가능하므로, 이전과 똑같이 사용해주면 된다.(인코딩 결과에 변동 없음)

03 한글의 표현

인코딩 방식은 크게 조합형과 완성형으로 나뉜다.

조합형 : 자모의 조합으로 한글을 표현

예시)

공 = ㄱ + ㅛ + ㅇ
0x020D0D 0x02 0x0D 0x0D

단점 : 1글자가 많은 메모리 차지

VS

완성형 : 자주 쓰이는 글자만 추려 코드와 1대1 대응

예시)

속 ↔ 0xBCD3

단점 : 쓰는 빈도가 낮은 글자는 아예 사용 불가하다

차지하는 크기에 따라

- n바이트 조합형
- 3바이트 조합형
- 2바이트 조합형으로 나뉜다.

03 한글의 표현

유니코드에서 한글의 다양한 표현 방법을 모두 지원한다.
또한 주로 UTF-8로 인코딩한다.

다양한 한글 표현방식에 배정된 유니코드 범위

표현방식	(코드의) 처음	(코드의) 끝	개수
한글 자모	0x1100	0x11FF	256
호환용 한글 자모	0x3130	0x318F	96
한글 자모 확장	0xA960	0xA97F	32
한글 소리 마디	0xAC00	0xD7AF	11184
한글 자모 확장 B	0xD7B0	0xD7FF	80

오른쪽 표는 ‘한글 자모’의 코드 영역 예시이며,
‘한글 자모’는 조합형 표현방식이다.

	110	111	112	113	114	115	116	117	118	119	11A	11B	11C	11D	11E	11F
0	ㄱ	ㄲ	ㄴ	ㄷ	ㄸ	ㄹ	ㅁ	ㅂ	ㅃ	ㅄ	ㅅ	ㅆ	ㅇ	ㅈ	ㅊ	ㅋ
1	ㆁ	ㆂ	ㆃ	ㆄ	ㆅ	ㆆ	ㆇ	ㆈ	ㆉ	ㆊ	ㆋ	ㆌ	ㆍ	ㆎ	㆏	㆐
2	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅃ	ㅄ	ㅅ	ㅆ	ㅇ	ㅈ	ㅊ	ㅋ	ㆁ	ㆂ	ㆃ
3	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅃ	ㅄ	ㅅ	ㅆ	ㅇ	ㅈ	ㅊ	ㅋ	ㆁ	ㆂ	ㆃ
4	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅃ	ㅄ	ㅅ	ㅆ	ㅇ	ㅈ	ㅊ	ㅋ	ㆁ	ㆂ	ㆃ
5	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅃ	ㅄ	ㅅ	ㅆ	ㅇ	ㅈ	ㅊ	ㅋ	ㆁ	ㆂ	ㆃ
6	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅃ	ㅄ	ㅅ	ㅆ	ㅇ	ㅈ	ㅊ	ㅋ	ㆁ	ㆂ	ㆃ
7	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅃ	ㅄ	ㅅ	ㅆ	ㅇ	ㅈ	ㅊ	ㅋ	ㆁ	ㆂ	ㆃ
8	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅃ	ㅄ	ㅅ	ㅆ	ㅇ	ㅈ	ㅊ	ㅋ	ㆁ	ㆂ	ㆃ
9	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅃ	ㅄ	ㅅ	ㅆ	ㅇ	ㅈ	ㅊ	ㅋ	ㆁ	ㆂ	ㆃ
A	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅃ	ㅄ	ㅅ	ㅆ	ㅇ	ㅈ	ㅊ	ㅋ	ㆁ	ㆂ	ㆃ
B	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅃ	ㅄ	ㅅ	ㅆ	ㅇ	ㅈ	ㅊ	ㅋ	ㆁ	ㆂ	ㆃ
C	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅃ	ㅄ	ㅅ	ㅆ	ㅇ	ㅈ	ㅊ	ㅋ	ㆁ	ㆂ	ㆃ
D	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅃ	ㅄ	ㅅ	ㅆ	ㅇ	ㅈ	ㅊ	ㅋ	ㆁ	ㆂ	ㆃ
E	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅃ	ㅄ	ㅅ	ㅆ	ㅇ	ㅈ	ㅊ	ㅋ	ㆁ	ㆂ	ㆃ
F	ㄴ	ㄷ	ㄹ	ㅁ	ㅂ	ㅃ	ㅄ	ㅅ	ㅆ	ㅇ	ㅈ	ㅊ	ㅋ	ㆁ	ㆂ	ㆃ

출처

파트 1

- https://ko.wikipedia.org/wiki/1%EC%9D%98_%EB%B3%B4%EC%88%98
- https://ko.wikipedia.org/wiki/2%EC%9D%98_%EB%B3%B4%EC%88%98
- <https://ko.wikipedia.org/wiki/%EC%97%94%EB%94%94%EC%96%B8>
- <https://dojang.io/mod/page/view.php?id=30>

파트 2

- <https://ko.wikipedia.org/wiki/%EA%B3%A0%EC%A0%95%EC%86%8C%EC%88%98%EC%A0%90>
- <https://ko.wikipedia.org/wiki/%EB%B6%80%EB%8F%99%EC%86%8C%EC%88%98%EC%A0%90>
- https://ko.wikipedia.org/wiki/IEEE_754
- https://ko.wikipedia.org/wiki/%EB%B9%84%EC%A0%95%EA%B7%9C_%EA%B0%92

파트 3

- <https://ko.wikipedia.org/wiki/%EB%B6%80%ED%98%B8%ED%99%94>
- <https://ko.wikipedia.org/wiki/ASCII>
- <https://ko.wikipedia.org/wiki/%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C>
- <https://ko.wikipedia.org/wiki/UTF-8>
- https://ko.wikipedia.org/wiki/%ED%95%9C%EA%B8%80_%EC%A1%B0%ED%95%A9%ED%98%95_%EC%9D%B8%EC%BD%94%EB%94%A9
- https://ko.wikipedia.org/wiki/%ED%95%9C%EA%B8%80_%EC%99%84%EC%84%B1%ED%98%95_%EC%9D%B8%EC%BD%94%EB%94%A9
- <https://d2.naver.com/helloworld/76650>