2장 수체계와 데이터 표현

- 수의 체계
- 진수 변환하기
- 모든 정보를 2진수로 표현하는 컴퓨터
- 컴퓨터에서의 문자 표현
- 컴퓨터에서의 정수 표현
- 컴퓨터에서의 실수 표현

한빛미디어(주)

학습목표

- 10진법, 2진법, 8진법, 16진법에 대해 살펴본다.
- 진수 변환에 대해 살펴본다 .
- 컴퓨터에서 문자를 표현하는 방법에 대해 살펴본다.
- 컴퓨터에서 정수를 표현하는 방법에 대해 살펴본다.
- 컴퓨터에서 실수를 표현하는 방법에 대해 살펴본다.

진법

- 사용할 수 있는 숫자의 개수와 위치 값을 정의해주는 수 체계

"1년은 365일이다" 에서 365란 300 더하기 60 더하기 5라고 알고 있다.

이것이 옳은가? 옳고 그름은 사용하고 있는 수 체계의 진법에 달려있다.

- 진법
  - ✓ 사용할 수 있는 숫자의 개수와 위치 값을 정의해주는 수 체계
  - ✓ 사용할 수 있는 숫자의 개수는 해당 진법과 같음
  - ✓ 사용할 수 있는 숫자는 0에서 시작해서 해당 진법의 수보다 1 적은 수까지
- 사용할 수 있는 숫자
  - ✓ 10진법 : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - ✓ 2진법 : 0과 1
  - ✓ 8진법 : 0, 1, 2, 3, 4, 5, 6, 7
  - ✓ 16진법 : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

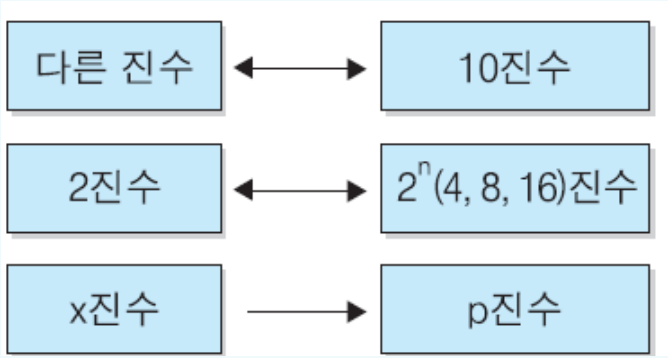
자리 값

- 각 숫자의 자리 값은 그 위치가 의미하는 제곱수를 해당 진법에 적용하면 됨
- 각 위치가 의미하는 제곱수는 가장 오른쪽이 0, 왼쪽으로 가면서 1을 더한 값이다

10진수	2진수	8진수	16진수
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12

[표 2-1] 진수 간의 값 비교

진수 변환하기



[그림 2-1] 진수 변환 형태

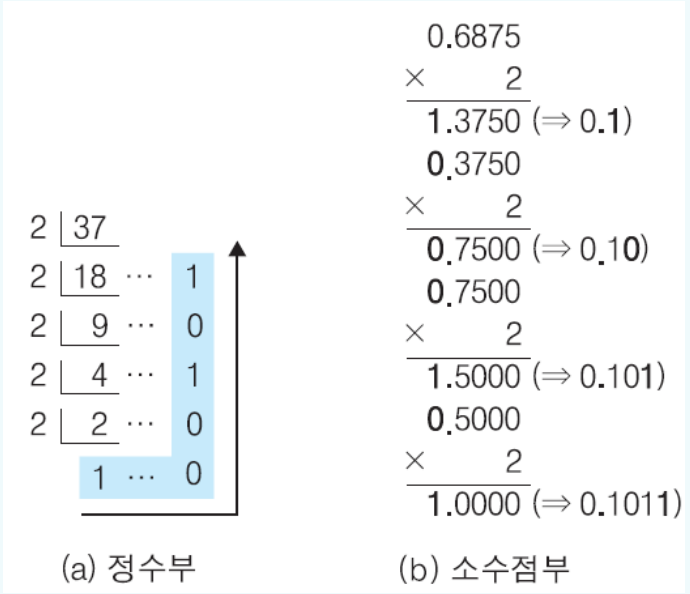
다른 진수에서 10진수로의 변환

$$27.42_8 = 2\times8^1 + 7\times8^0 + 4\times8^{-1} + 2\times8^{-2}$$
$$= 16 + 7 + 4\times\frac{1}{8} + 2\times\frac{1}{8^2}$$
$$= 16 + 7 + 0.5 + 0.03125$$
$$= 23.53125_{10}$$

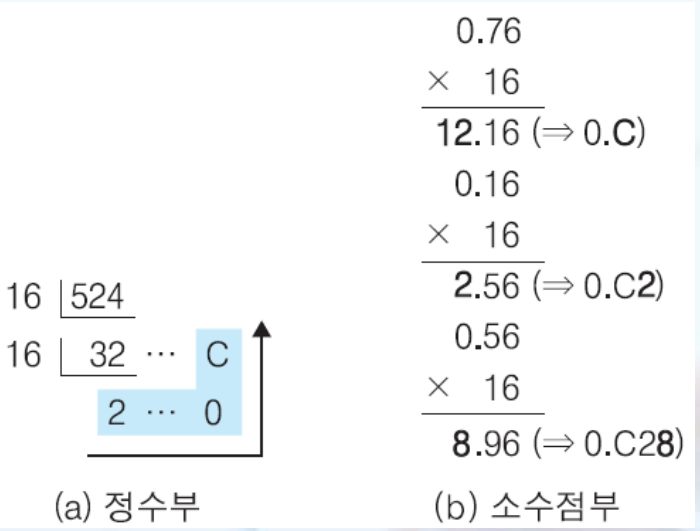
$$AF.8_{16} = A\times16^1 + F\times16^0 + 8\times16^{-1}$$
$$= 10\times16^1 + 15\times16^0 + 8\times\frac{1}{16^1}$$
$$= 160 + 15 + 0.5$$
$$= 175.5_{10}$$

10진수에서 다른 진수로의 변환

- 10진수 37.6857를 2진수로 변화하기



- 10진수 524.76을 16진수로 변환하기



2진수에서 2n진수로의 변환

- 2진수 를 4진수로 변환

[표 2-2] 2진수 2비트는 4진수 한 자리와 대응된다

2진수	00	01	10	11
4진수	0	1	2	3

✓ 2진수 10110.11101을 4진수로 변환하기

```
 1 0 1 1 0 . 1 1 1 0 1    (2진수)
 └─┘ └─┘ └─┘   └─┘ └─┘ └─┘
 1   1   2   .  3   2   2    (4진수)
```

- 2진수를 8진수로 변환

2진수	000	001	010	011	100	101	110	111
8진수	0	1	2	3	4	5	6	7

✓ 2진수 10110.11101을 8진수로 변환하기

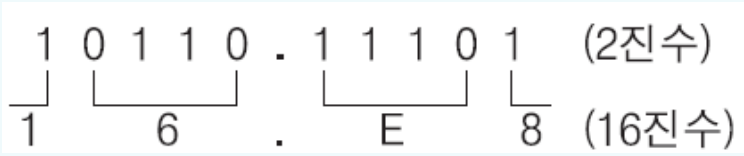
2진수	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
16진수	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

2진수에서 2n진수로의 변환

- 2진수를 16진수로 변환

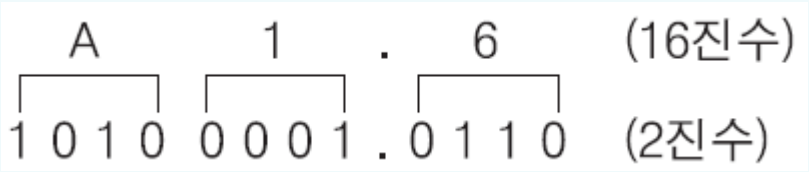
2진수	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
16진수	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

- 2진수 10110.11101을 16진수로 변환하기



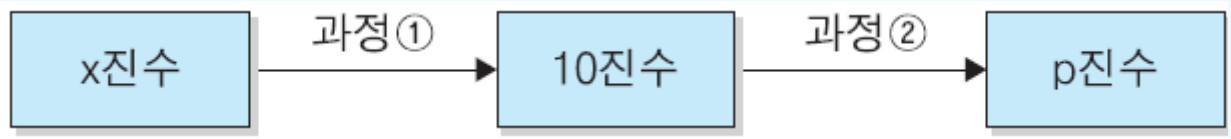
2n진수에서 2진수로의 변환

- 2진수를 진수로 변환하는 원리를 그대로 이용
- 16진수 A1.6을 2진수로 변환하기



기타 변환(x진수의 p진수의 변환)으로 구분

- x진수를 p진수로 변환
  - ✓ x진수를 먼저 10진수로 변환한 후 p진수로 변환

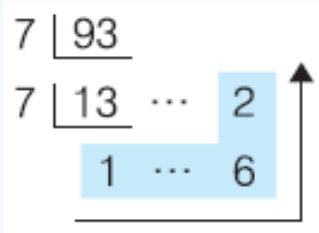


[그림 2-8] x진수를 p진수로 변환하기

- 2진수 1011101을 7진수로 변환하기
  - ✓ 2진수 1011101을 10진수로 변환하면 93이 된다.

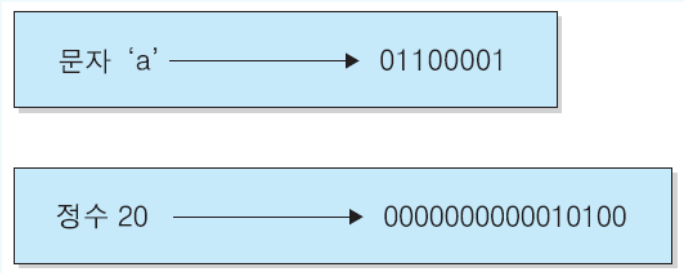
$$1011101_2 = 1\times 2^6 + 1\times 2^4 + 1\times 2^3 + 1\times 2^2 + 1 = 93_{10}$$

  - ✓ 10진수 93을 7진수로 변환하면 162가 된다.



모든 정보를 2진수로 표현하는 컴퓨터

- 오늘날의 컴퓨터는 문자, 정수, 실수, 그림, 소리, 동영상 등의 모든 정보를 안정성이 뛰어난 2진수 형식으로 표현 한다
- 0 또는 1의 2진수 개념은 모든 전기적인 장치의 on/off와 맞는 개념이다.



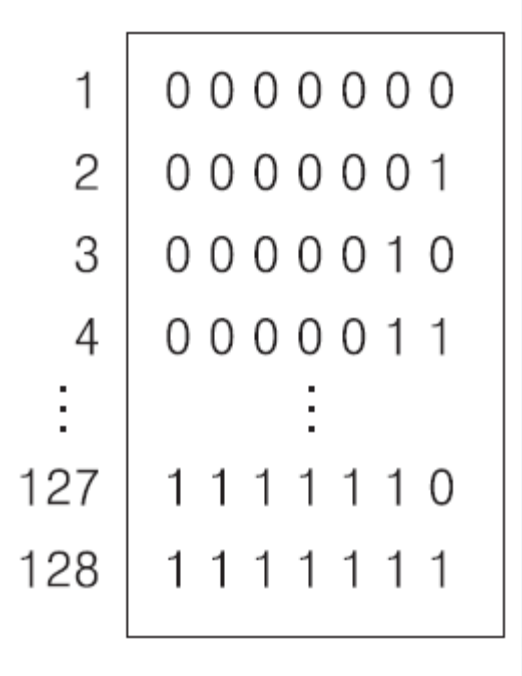
[그림 2-9] 컴퓨터 내부에서의 문자'a'와 정수 20의 표현

1비트	2비트	3비트
0	00	000
1	01	001
	10	010
	11	011
		100
		101
		110
		111

[표 2-5] 비트 표현

ASCII

- 호환 등 여러 가지 문제를 해결하기 위해 ANSI에서 ASCII (American Standard Code for Information Interchange)라는 표준 코드 체계를 제시
- ASCII는 각 문자를 7비트로 표현하므로 총 128(= 27)개의 문자를 표현



1	0 0 0 0 0 0 0
2	0 0 0 0 0 0 1
3	0 0 0 0 0 1 0
4	0 0 0 0 0 1 1
⋮	⋮
127	1 1 1 1 1 1 0
128	1 1 1 1 1 1 1

[그림 2-10] 7비트로 128가지를 달리 표현할 수 있다

ASCII

[표 2-6] ASCII 문자 코드

0000000	NUL	0100000	Space	1000000	@	1100000	`
0000001	SOH (Start of Heading)	0100001	!	1000001	A	1100001	a
0000010	STX (Start of Text)	0100010	"	1000010	B	1100010	b
0000011	ETX (End of Text)	0100011	#	1000011	C	1100011	c
0000100	EOT (End of Transmission)	0100100	$	1000100	D	1100100	d
0000101	ENQ (Enquiry)	0100101	%	1000101	E	1100101	e
0000110	ACK (Acknowledge)	0100110	&	1000110	F	1100110	f
0000111	BEL (Bell)	0100111	'	1000111	G	1100111	g
0001000	BS (Backspace)	0101000	(	1001000	H	1101000	h
0001001	HT (Horizontal Tabulation)	0101001	)	1001001	I	1101001	i
0001010	LF (Line Feed)	0101010	*	1001010	J	1101010	j
0001011	VT (Vertical Tabulation)	0101011	+	1001011	K	1101011	k
0001100	FF (Form Feed)	0101100	,	1001100	L	1101100	l
0001101	CR (Carriage Return)	0101101	-	1001101	M	1101101	m
0001110	SO (Shift Out)	0101110	.	1001110	N	1101110	n
0001111	SI (Shift In)	0101111	/	1001111	O	1101111	o

ASCII

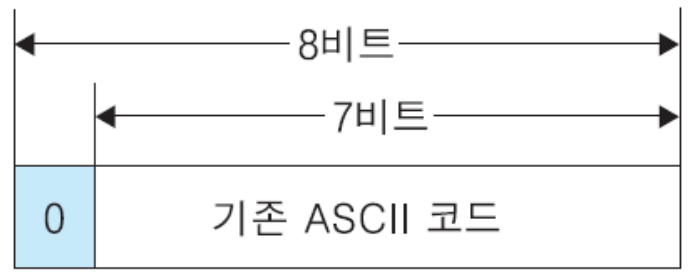
[표 2-6] ASCII 문자 코드(계속)

0010000	DLE (Data Link Escape)	0110000	0	1010000	P	1110000	p
0010001	DC1 (Device Control 1)	0110001	1	1010001	Q	1110001	q
0010010	DC2 (Device Control 2)	0110010	2	1010010	R	1110010	r
0010011	DC3 (Device Control 3)	0110011	3	1010011	S	1110011	s
0010100	DC4 (Device Control 4)	0110100	4	1010100	T	1110100	t
0010101	NAK (Negative Acknowledge)	0110101	5	1010101	U	1110101	u
0010110	SYN (Synchronous Idle)	0110110	6	1010110	V	1110110	v
0010111	ETB (End of Transmission Block)	0110111	7	1010111	W	1110111	w
0011000	CAN (Cancel)	0111000	8	1011000	X	1111000	x
0011001	EM (End of Medium)	0111001	9	1011001	Y	1111001	y
0011010	SUB (Substitute)	0111010	:	1011010	Z	1111010	z
0011011	ESC (Escape)	0111011	;	1011011	[	1111011	{
0011100	FS (File Separator)	0111100	<	1011100	\	1111100	|
0011101	GS (Group Separator)	0111101	=	1011101	]	1111101	}
0011110	RS (Record Separator)	0111110	>	1011110	^	1111110	~
0011111	US (Unit Separator)	0111111	?	1011111	_	1111111	DEL

ASCII

확장(extended) ASCII

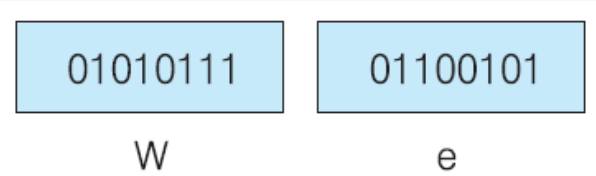
- 256( )개의 문자를 표현
- 기존 7비트 ASCII 코드에는 가장 왼쪽에 0을 추가



[그림 2-11] 기존 ASCII의 확장 ASCII로의 표현

ASCII로 표현한 "We"

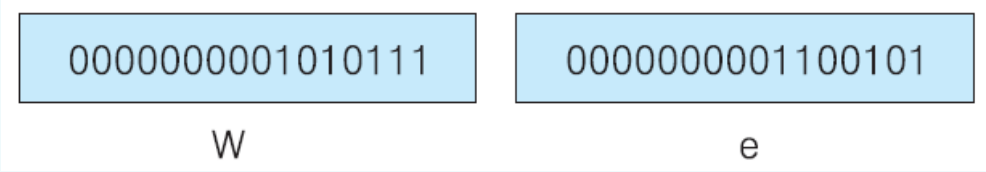
- ASCII는 각 나라별 언어는 표현이 불가능



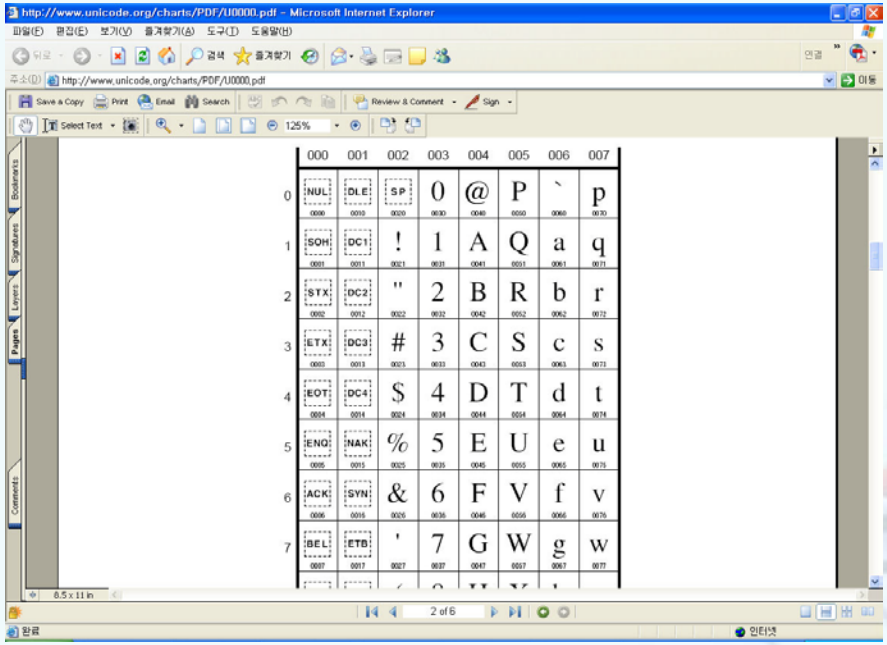
[그림 2-12] ASCII로 표현한"We"

유니코드

- 각 나라별 언어를 모두 표현하기 위해 나온 코드 체계
- 문자를 16비트로 표현하므로 최대 65,536자를 표현



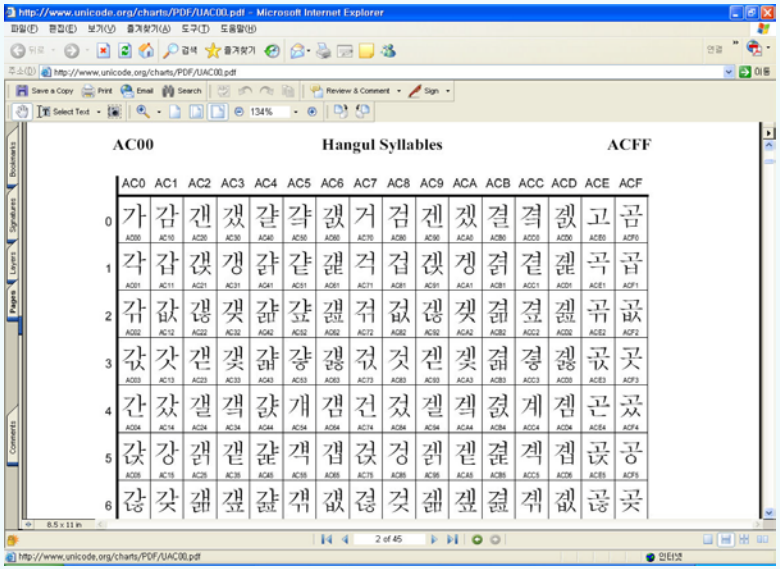
[그림 2-13] 영문자, 숫자에 대한 유니코드



[그림 2-14] 유니코드로 표현한"We"

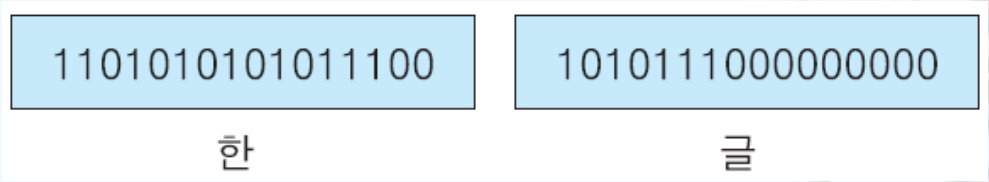
유니코드

- 한글에 대한 유니코드 (http://www.unicode.org/charts/PDF/UAC00.pdf)



[그림 2-15] 한글에 대한 유니코드

- 유니코드로 표현한 "한글"



[그림 2-16] 유니코드로 표현한"한글"

텍스트 압축

런 렝스 코딩

- 파일의 내용을 '반복문자×탈출문자×반복회수'로 나타내어 압축하는 방법

허프만 코딩

- 자주 사용되는 문자는 적은 비트로 된 코드로 변환해서 표현하고, 별로 사용되지 않는 문자는 많은 비트로 된 코드로 변환하여 표현
- 다음 텍스트를 허프만 코딩을 이용해 압축해 보자

AAAAAAABBCCCDEEEEFFFFFFG

- ① 데이터에서 사용되는 각 문자에 대한 출현 빈도수를 구한다.

문자	A	B	C	D	E	F	G
출현빈도	7	2	3	1	4	6	1

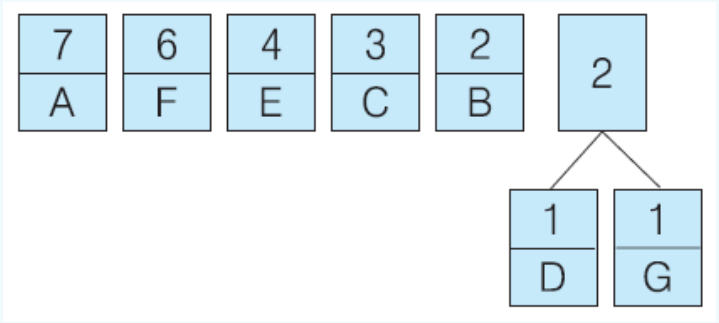
[그림 2-17] 허프만 코드를 사용한 텍스트 압축 과정 ①

- ② 빈도수를 기준으로 내림차순으로 정렬한다.

7	6	4	3	2	1	1
A	F	E	C	B	D	G

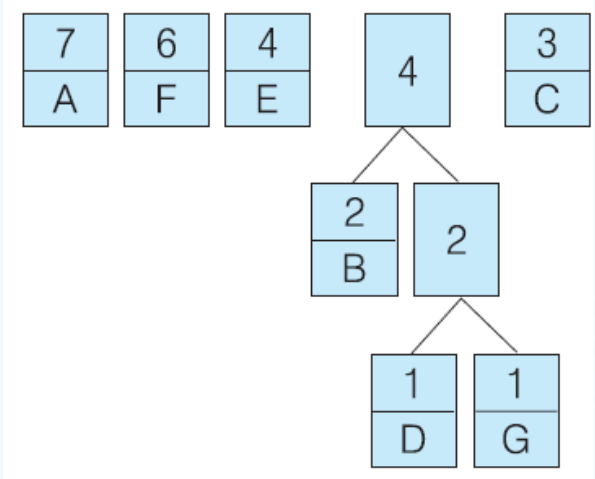
[그림 2-18] 허프만 코드를 사용한 텍스트 압축 과정 ②

- ③ 출현 빈도가 가장 적은 2개의 문자인 D와 G를 가지로 연결하고, 가지 위에 두 문자의 빈도수의 합인 2를 적는다. 빈도수의 합인 2를 기준으로 재배열한다.



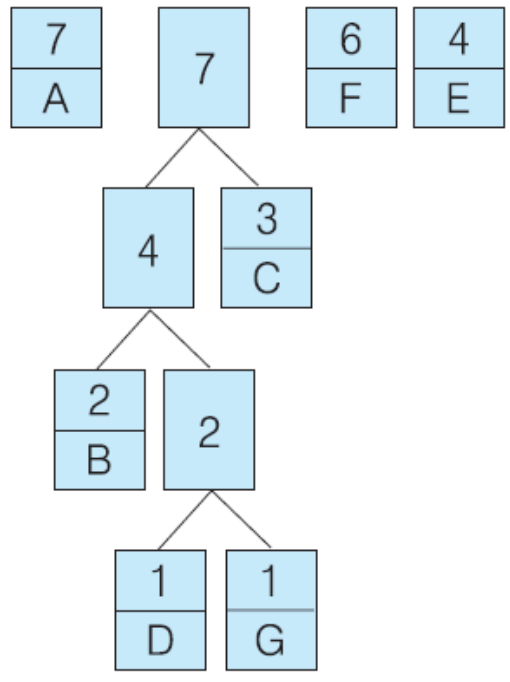
[그림 2-19] 허프만 코드를 사용한 텍스트 압축 과정 ③

- ④ 마찬가지로 값이 가장 작은 두 개의 노드를 가지로 연결하고, 두 값의 합인 4를 적는다. 새롭게 생성된 노드를 재배열한다.



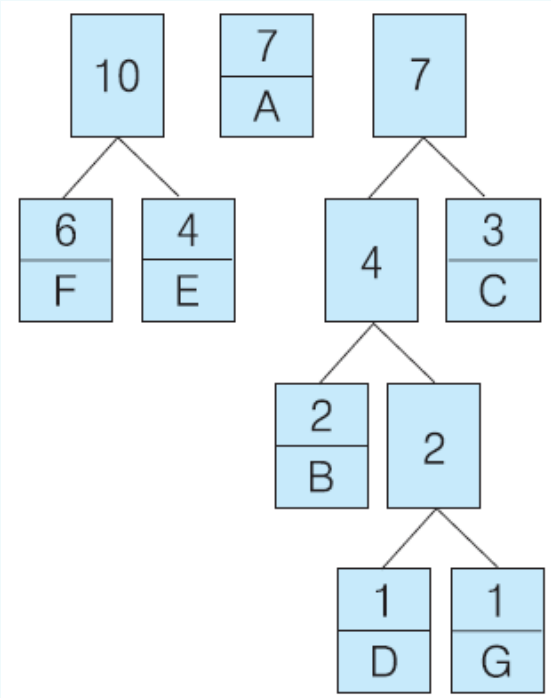
[그림 2-20] 허프만 코드를 사용한 텍스트 압축 과정 ④

- ⑤ 노드 값이 가장 작은 두 개의 노드를 연결하고, 재배열한다



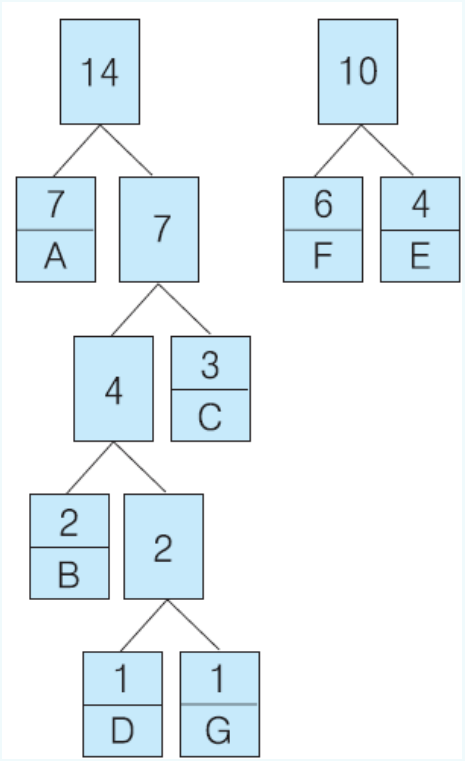
[그림 2-21] 허프만 코드를 사용한 텍스트 압축 과정 ⑤

- ⑥ 노드 값이 가장 작은 두 개의 노드를 연결하고, 재배열한다.



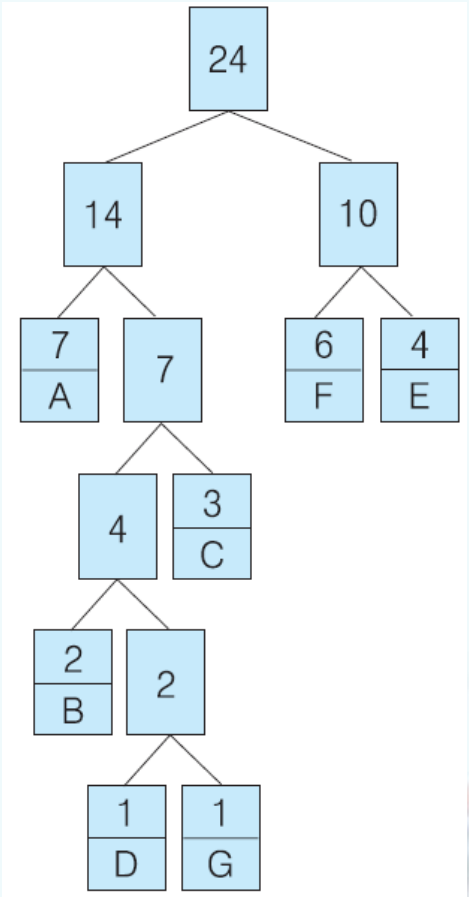
[그림 2-22] 허프만 코드를 사용한 텍스트 압축 과정 ⑥

- ⑦ 마찬가지 작업을 한다.



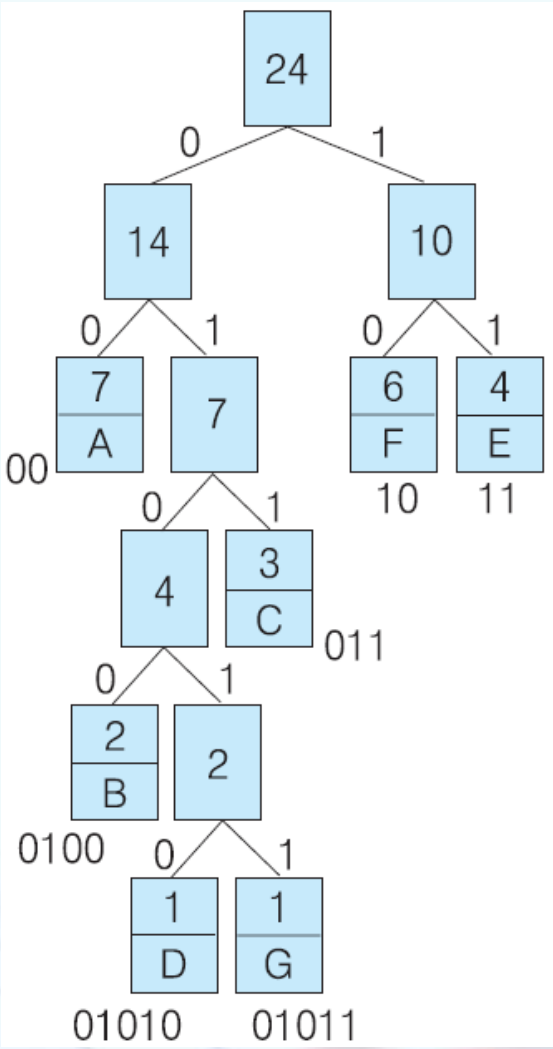
[그림 2-23] 허프만 코드를 사용한 텍스트 압축 과정 ⑦

- ⑧ 마찬가지 작업을 한다.



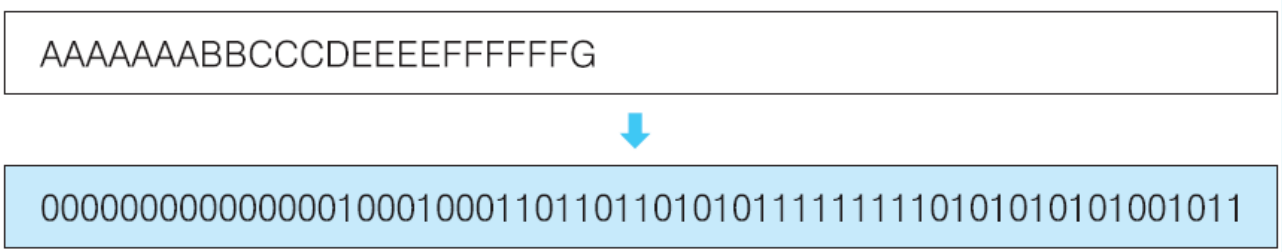
[그림 2-24] 허프만 코드를 사용한 텍스트 압축 과정 ⑧

- ⑨ 각 가지의 왼쪽에는 0, 오른쪽에는 1을 쓴다. 그리고 뿌리로 시작해서 가지로 숫자를 읽어 내려가 알파벳에 적는다. 이 숫자가 허프만 코드가 된다.



[그림 2-25] 허프만 코드를 사용한 텍스트 압축 과정 ⑨

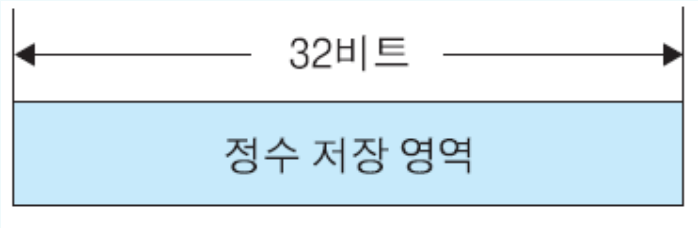
- ⑩ 텍스트를 허프만 코드로 나타내보자



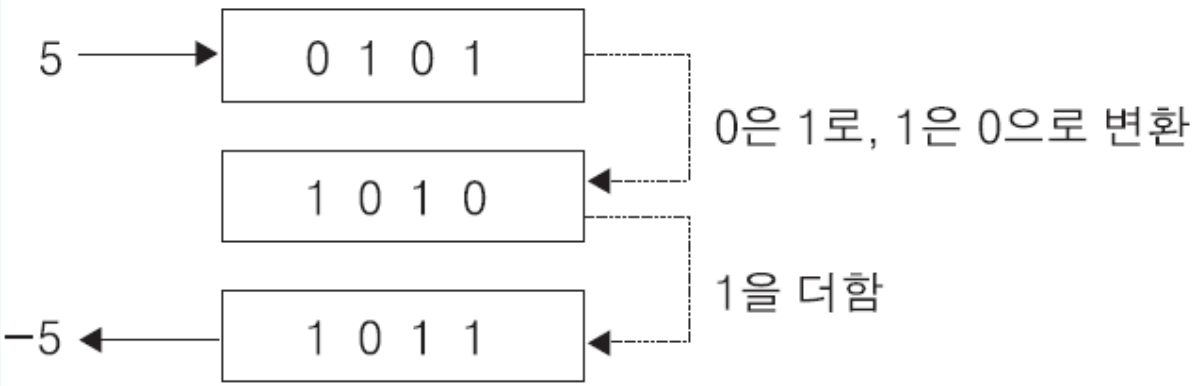
[그림 2-26] 허프만 코드를 사용한 텍스트 압축 결과

정수 표현하기

- 컴퓨터의 기억 공간은 제한적이므로 한 정수를 나타낼 기억 영역도 제한적이다. 시스템에 따라 약간의 차이가 있지만, 대부분 32비트로 정수를 표현
- 컴퓨터에서 정수를 표현하는 방법은 2의 보수(2's complement) 표기법



[그림 2-27] 32비트 정수



[그림 2-28] -5의 2의 보수 표현 과정

음수에 대한 2의 보수 표현

[표 2-9] 음수에 대한 2의 보수 표현

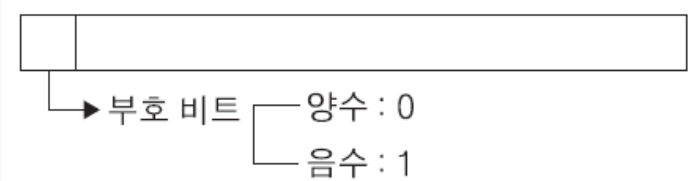
표현	값
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

8비트의 2의 보수 표현

[표 2-10] 8비트의 2의 보수표현

표현	값
01111111	127
01111110	126
01111101	125
⋮	⋮
00000001	1
00000000	0
11111111	-1
11111110	-2
⋮	⋮
10000001	-127
10000000	-128

2의 보수 표현의 부호 비트



[그림 2-29] 2의 보수 표현의 부호 비트

표현할 수 있는 수의 범위

$$-2^{n-1} \sim 2^{n-1}-1$$

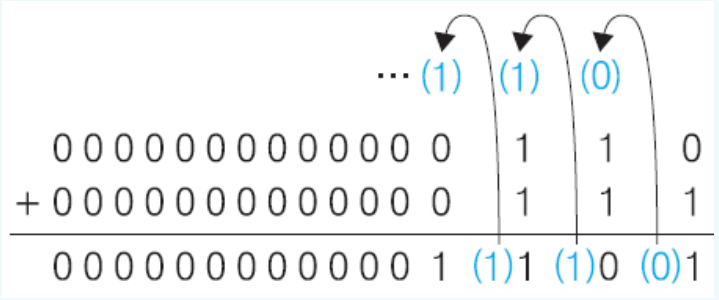
정수의 덧셈과 뺄셈

- 정수의 덧셈

16비트로 정수를 표현한다고 가정

- 6 + 7

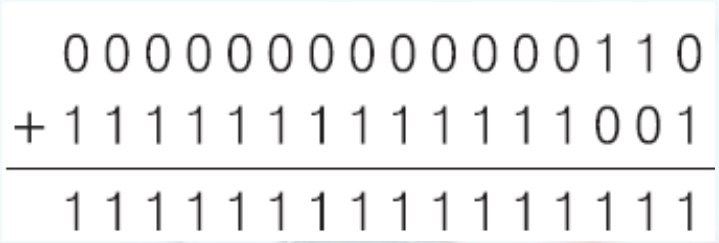
```
6 => 0000000000000110
7 => 0000000000000111
```



[그림 2-30] 6과 7의 덧셈

- 6 + (-7)

```
 6 => 0000000000000110
-7 => 1111111111111001
```



[그림 2-31] 6과 -7의 덧셈

정수의 덧셈

- (-6) + 7

```
-6 => 1111111111111010
 7 => 0000000000000111
```

```
  1111111111111010
+ 0000000000000111
------------------
 10000000000000001
 ↳ 무시
```

[그림 2-32] -6과 7의 덧셈

- (-6) + (-7)

```
-6 => 1111111111111010
-7 => 1111111111111001
```

```
  1111111111111010
+ 1111111111111001
------------------
 11111111111110011
 ↳ 무시
```

[그림 2-33] -6과 -7의 덧셈

- 30000 + 30000

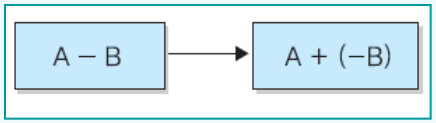
```
30000 => 0111010100110000
```

```
  0111010100110000
+ 0111010100110000
------------------
  1110101001100000
```

[그림 2-34] 30000과 30000의 덧셈

정수의 뺄셈

- 컴퓨터 내부에서의 뺄셈은 덧셈을 하는 하드웨어인 가산기를 이용



- 6 - 7

```
  0000000000000110
+ 1111111111111001
------------------
  1111111111111111
```

[그림 2-35] 6과 7의 뺄셈

- -6 - 7

```
  1111111111111010
+ 1111111111111001
------------------
1 1111111111110011
  ↳ 무시
```

[그림 2-36] -6과 7의 뺄셈

- -30000 – 30000
  - ✓ 최상위 bit를 넘어가는 자리 올림수 1을 무시하면 5536이 되어 오버플로우 발생

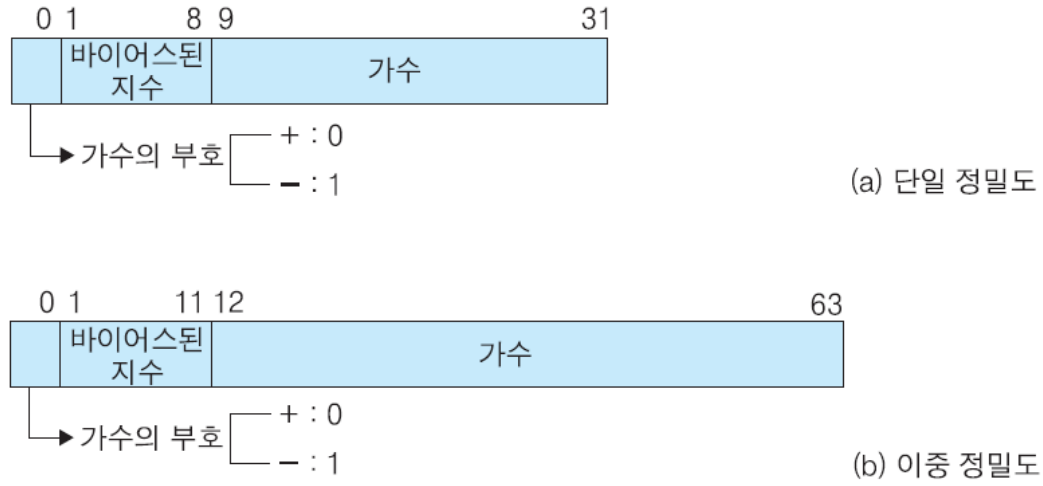
```
  1000101011010000
+ 1000101011010000
------------------
1 0001010110100000
  ↳ 무시
```

[그림 2-37] -30000에서 30000의 뺄셈

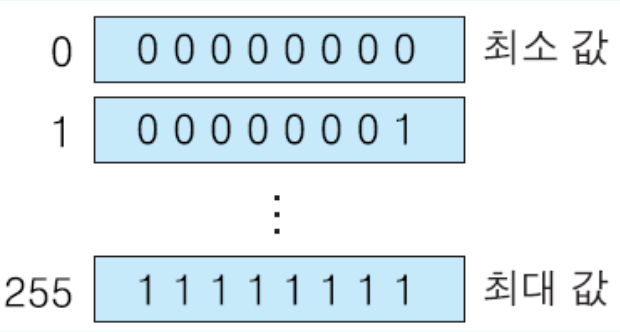
0001010110100000

실수 표현하기

- 소수점 부를 가지고 있는 수를 실수라 하는데. 실수로 컴퓨터 내부에서는 제한된 공간에 2진수 형태로 표현된다.

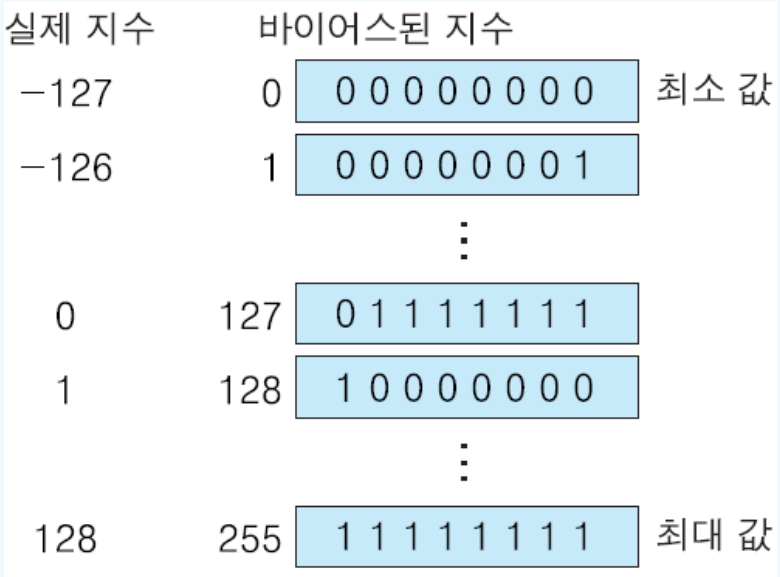


[그림 2-38] IEEE 754 실수 표현 형식



[그림 2-39] 바이어스 개념이 없을 경우의 지수 표현 범위

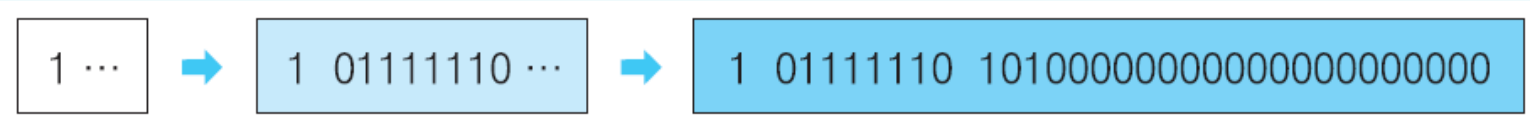
실수 표현하기



[그림 2-40] 바이어스가 127인 경우의 지수 표현 범위

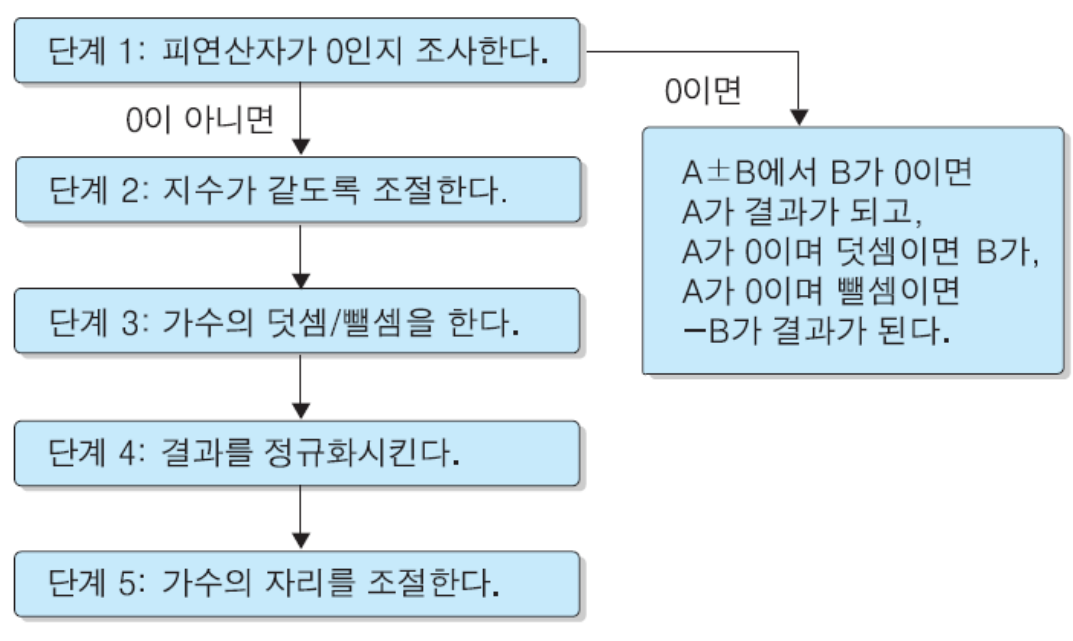
[그림 2-41] 간단하게 표현 한 실수 변환 예제

실수 표현하기



[그림 2-42] 실수 변환 결과

실수의 덧셈과 뺄셈



[그림 2-43] 실수의 덧셈과 뺄셈 절차

실수의 덧셈과 뺄셈

$1.101\times2^2+1.11\times2^4$

$1.101\times2^2 \Rightarrow 0\ 10000001\ 10100000000000000000000$

$1.101\times2^4 \Rightarrow 0\ 10000011\ 11000000000000000000000$

- 피연산자가 0인지를 확인한다. 0이 아니므로 단계 2로 넘어감
- 작은 지수를 갖는 피연산자인 1.101×2^2 을 지수가 4가 되도록 조절

$0.01101+1.11=10.00101$

- 지수가 같으므로 가수를 더함

$1.101\times2^2 \Rightarrow 0.01101\times2^4$

- 연산 결과가 다음과 같은데, 정규화되어 있지 않은 상태므로 정규화시킴

$10.00101\times2^4 \Rightarrow 1.000101\times2^5$

- 이를 IEEE 754 형식으로 나타내면 다음과 같다.

0 10000100 00010100000000000000000

실수의 덧셈과 뺄셈

$$1.101011\times 2^{5}+1.01\times 2^{10}$$

- 가수는 8비트로 표현된다고 가정
- 지수를 같게 조절하고 가수를 더하면

$$0.00001101011\times 2^{10}+1.01\times 2^{10}$$
$$0.00001101011+1.01=1.01001101011$$

- 다음과 같은 정규화된 결과를 얻음

$$1.01001101011\times 2^{10}$$

- 가정에서 가수가 8비트로 표현된다고 했으므로 소수점 8 자리를 초과하는 부분은 표현할 수 없다.
- 그러므로 소수점 9자리에서 반올림하여 다음과 같이 조절해야 하는데, 이런 일을 단계 5에서 수행한다.

$$1.01001101\times 2^{10}$$

- 라운드 오프 오류(round off error)

Thank you