

# 데이터 구성

## 1. 물리적 구성

- **비트(Bit)**

- 1 자리의 바이너리 숫자, 0, 1. Binary digit.

- **니블(Nibble)**

- 4 자리의 바이너리 숫자, 0000, 0001, 0010, 0011, 0100, 1000, 1111, ....

- **바이트(Byte)**

- 8 자리의 바이너리 숫자, 01000011, 10011001, 11001100, ....

- **워드(Word)**

- 컴퓨터 종류에 따라 길이가 다름. CPU 연산의 기본 처리 단위와 동일

- 8-비트 컴퓨터 → 워드 길이는 8-비트
    - 16-비트 컴퓨터 → 워드 길이는 16-비트
    - 32-비트 컴퓨터 → 워드 길이는 32-비트

## 2. 논리적 구성

- 논리적 구성

- 플래그(Flag) : 1-비트 데이터, 상태 또는 이벤트 발생 유무 표시
- 필드(Field) : n-비트로 구성된 데이터.
- 레코드(Record) : 서로 관련된 여러 개의 필드로 구성된 데이터. 논리적 레코드
- 블록(Block) : 저장매체에 입출력할 때 기본 단위. 물리적 레코드
- 파일(File) : 레코드의 집합.
- 데이터베이스 : 파일의 집합으로 계층적 구조로 데이터 조직화.

### 3. 데이터 유형

- 수치 (Numeric) 데이터

- 정수(Integer) 표현
- 실수(Floating-point) 표현

- 비수치(Non-numeric) 데이터

- 문자(Character) 표현
- 멀티 미디어(오디오, 이미지, 비디오, 그래픽) 데이터 표현

## 정수 표현

## 1. 정수 표현


- 이진수 '0' 과 '1' 로 표현
- 고정 소수점(Fixed-point) 표현 : 우측끝에 소수점 위치.
- 양수만을 다루는 경우, 정수를 바이너리로 표현하는 것은 간단
- 음수 표현
  - Sign-Magnitude 표현법 : MSB = sign-bit
    - 부호 비트와 크기 값을 분리하여 표현
  - 1의 보수(1's Complement)
    - 크기 값을 negation ( $0 \rightarrow 1, 1 \rightarrow 0$ ) 해서 음수를 표현
  - 2의 보수 (2's Complement)
    - 1의 보수 + '1'

## 2. Sign-Magnitude

- 부호비트(Sign bit)과 크기를 분리 표시
  - 최상위 비트(좌측 끝)가 부호비트(Sign bit)
    - 0 = positive
    - 1 = negative
  - 나머지 비트로 크기표시

Sign	Magnitude
------	-----------

+18 = 00010010  
-18 = 10010010



Sign Bit

- 단점
  - 연산할 때 부호와 크기를 별도로 고려해야 하기 때문에 복잡해진다.
  - '0'을 2가지로 표현하게 된다. (+0 and -0) : 100...000, 000...000

### 3. 1's Complement

- 1의 보수(One's Complement) 표현

- 최상위 비트를 부호비트로 사용, 나머지 비트들은 크기.
- 크기를 양수로 표현한 후, 각 비트를 negation 한 결과 값
  - 예) -5를 4-bit, 1의 보수로 표현하면, 10110
- N을 n-bit 의 이진수로 표현할 경우,
  - N을 1의 보수로 표현한 값 =  $(2^n - 1) - N$

- 1's Complement 특징

- '0' 표현이 2가지 : 000...000, 111...111
- 표현 범위는 :  $-(2^{n-1} - 1) \sim 2^{n-1} - 1$
- 현대 컴퓨터에서 거의 사용하지 않음.

+3 = 00000011  
+2 = 00000010  
+1 = 00000001  
+0 = 00000000  
-0 = 11111111  
-1 = 11111110  
-2 = 11111101  
-3 = 11111100



## 4. 2's Complement

- 2의 보수(Two's Complement) 표현

- 최상위 비트를 부호비트로 사용, 나머지 비트들은 크기표현.
  - 크기 비트들의 해석방법이 다름.
- 예) -5를 4-bit, 2의 보수로 표현하면, 10111
- N을 n-bit 의 이진수로 표현할 경우,
  - N을 2의 보수로 표현한 값 =  $2^n - N$

+3 = 00000011  
+2 = 00000010  
+1 = 00000001  
+0 = 00000000  
-1 = 11111111  
-2 = 11111110  
-3 = 11111101

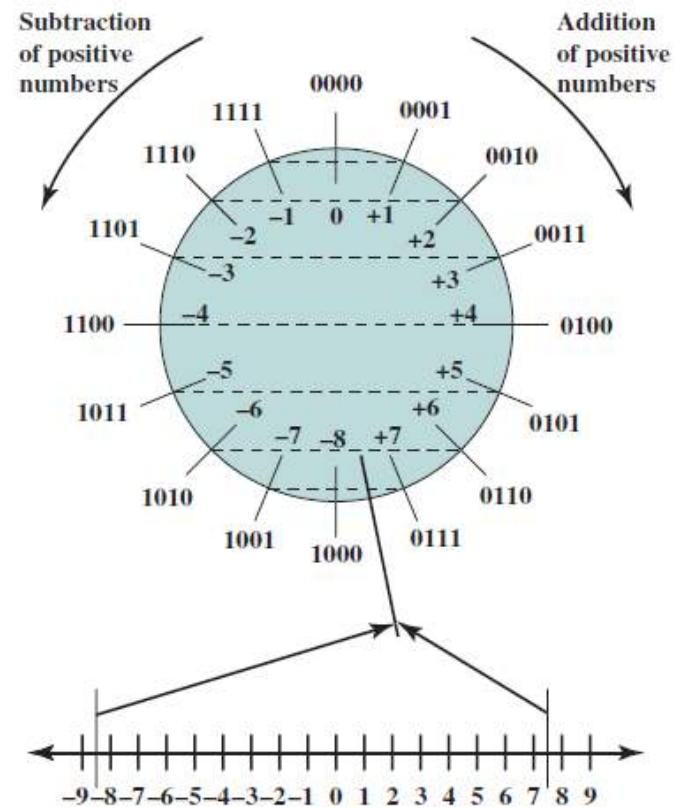
### '0' 의 2's Complement 구하기

0 = 00000000  
Negation of '0' = 11111111  
'1' 더한 결과 = 1 00000000  
Carry 무시 = 00000000  
결과적으로 - 0 표현과 0 이 동일

### '128' 의 2's Complement 구하기

128 = 10000000  
Negation of -128 = 01111111  
'1' 더한 결과 = 10000000  
결과적으로  
128 = -128 (오류!!!!)  
따라서, 128 표현불가!!!  
부호비트 캐리 (0), 그 다음 비트 캐리(1)을  
XOR한 결과 '1' --> Overflow 발생 !!

## 4.1 4-bit 2's Complement 의 표현범위



## 4.2 2's Complement 특징

- 2's Complement 특징

- '0' 표현은 1가지 : 000...000
- 표현 범위는 :  $-2^{n-1} \sim 2^{n-1} - 1$
- 음수 표현 과정이 복잡.
  - 크기를 양수로 표현한 후, 각 비트를 negation ( $0 \rightarrow 1, 1 \rightarrow 0$ ) 한 결과 값에 '1'을 더한다. 이때 발생하는 Carry 비트는 무시.
- 오버플로우(Overflow) 검사가 간단.
  - 부호비트와 그 다음 상위비트에서 발생한 Carry 비트를 XOR 한 결과 '1' 이면 오버플로우 발생.
- 뺄셈이 간단하다.
  - $A - B$  인 경우,  $A + (-B)$  로 처리 : B의 2's Complement를 구하여 더한다.

## 5. 표현 범위 확장

- 정수 표현 범위 확장할 경우, 부호 연장(Sign Extension) 을 하면 된다.
- Sign Extension
  - Sign-Magnitude

+18	=	00010010
+18	=	0000000000010010
-18	=	10010010
-18	=	1000000000010010

- Two's Complement

+18	=	00010010
+18	=	0000000000010010
-18	=	11101110
-32,658	=	1000000001101110
-18	=	111111111101110

Error !!

O.K !!

## 실수 표현

# 1. 실수

- 실수(Real Number)

- 실수 = 정수 + 소수
- 부동 소수점(Floating-point) 방식을 사용하여 표현

$$1001.1010 = 2^4 + 2^0 + 2^{-1} + 2^{-3} = 9.625$$

- 부동소수점 표현

- 가수(Significant, Mantissa) : 유효숫자 표시. 양수.
- 지수(Exponent) : 소수점의 위치 표시. 양수 또는 음수.
- 부호 비트(Sign Bit) : 가수부의 부호. 음수, 양수 표시, 1-비트
- 밑(Base) : 이진법인 경우, '2'

$$\pm S \times B^{\pm E}$$

## 1.1 Biased Exponent

### • Exponent의 Biased 표현

- Biased Exponent : 실제 Exponent 값에 Bias 값을 더한 값.
  - Bias 값 :  $2^{k-1} - 1$ ,  $k$  = Exponent의 비트 수
    - 예를 들어,  $k=8$  인 경우, Bias 값은 127.
    - 실제 Exponent 범위(-127 ~ +128)를 Biased Exponent의 범위 0~255로 표현.
- Exponent 값에는 부호비트가 따로 없기 때문에 음수 지수를 표현하기 위해 Biased Exponent 표현을 사용.
  - Biased Exponent 장점 : 실수 값 비교를 정수처럼 다룰 수 있다.

실제 Exponent 값이 '20' 인 경우

```
8-bit Exponent 표현 = 0001 0100
Biased Exponent      = 0001 0100
                      + 0111 1111
                      -----
                      1001 0011
```

## 1.2 Normalized Significant

- **Normalized Significant 표현**

- Significant를 1.xxxx 형식으로 정규화하여 표현한다.
  - Significant bit 수를 1-bit 줄일 수 있다.

$$\pm S \times B^{\pm E} \longrightarrow \pm 1.bbb \dots b \times 2^{\pm E}$$

Normalized Significant (Binary)

Significant 값이 '1.6328125' 인 경우

Binary 표현 = 1.1010001

0.11010001 x 2<sup>21</sup>

1.1010001 x 2<sup>20</sup> (Normalized 표현)

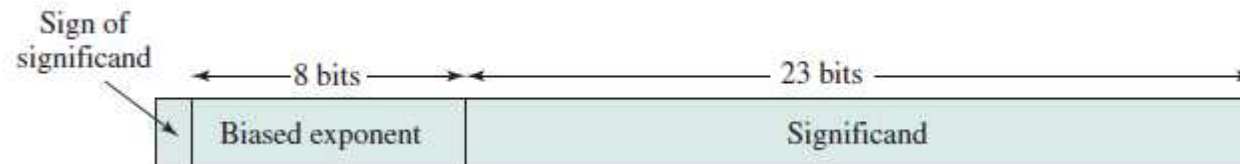
11.010001 x 2<sup>19</sup>



## 1.3 32-bit 실수 표현

- 32-bit 실수 표현

- 23-bit Significant, 8-bit Exponent, 1-bit Sign Bit



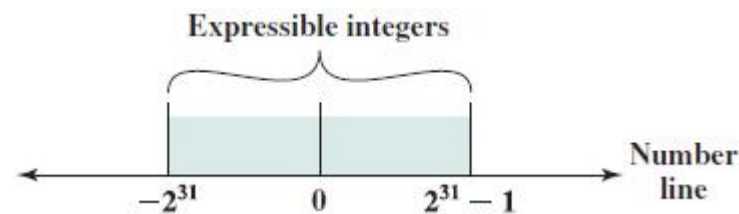
$$\begin{array}{llll} 1.1010001 \times 2^{10100} & = & 0 \ 10010011 \ 101000100000000000000000 & = \ 1.6328125 \times 2^{20} \\ -1.1010001 \times 2^{10100} & = & 1 \ 10010011 \ 101000100000000000000000 & = -1.6328125 \times 2^{20} \\ 1.1010001 \times 2^{-10100} & = & 0 \ 01101011 \ 101000100000000000000000 & = \ 1.6328125 \times 2^{-20} \\ -1.1010001 \times 2^{-10100} & = & 1 \ 01101011 \ 101000100000000000000000 & = -1.6328125 \times 2^{-20} \end{array}$$

- 정확도(Accuracy)

- Significant 의 LSB 값 차이 만큼 오차발생
  - 23-bit Significant 경우 :  $2^{-23} \approx 1.2 \times 10^{-7}$

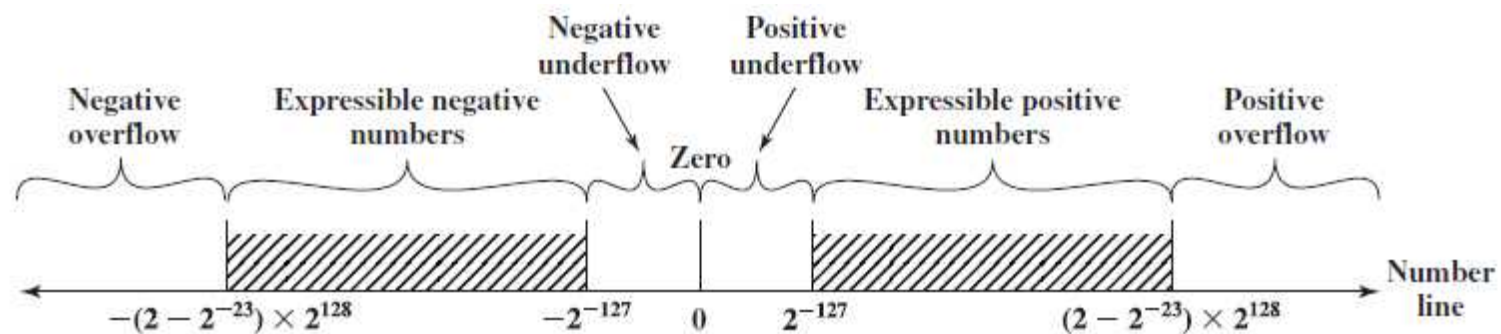
## 1.4 32-bit 정수와 실수 표현범위 비교

- 32-bit 정수



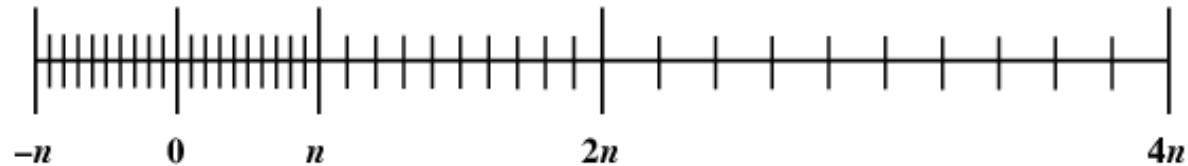
- 32-bit 실수

- 23-bit Significant, 8-bit Exponent, 1-bit Sign Bit 인 경우



## 2. 실수 표현의 한계성

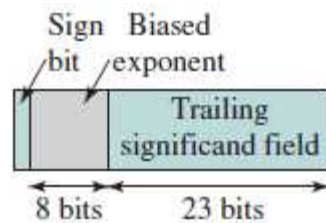
- 모든 실수 값을 표현할 수 없다.
  - 32-bit 실수 인 경우, 232 가지 실수만을 표현가능.
    - 표현하고자 하는 실수 값을 근사값으로 표현.
- 실수로 표현하는 값들은 동일한 간격으로 분포된 것이 아니라, '0'에 가까울 수록 조밀하고, '0'에서 멀어질수록 성기다.



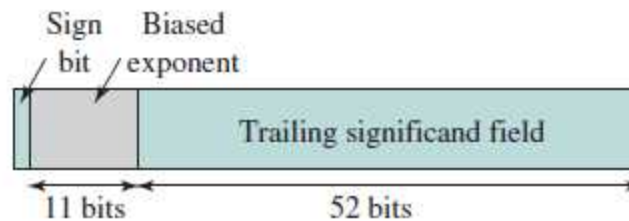
- 고정된 길이의 실수 표현에서는 표현범위(Range)와 정밀도(Precision)는 서로 상충관계(Trade-off) 관계
  - 표현범위를 늘리면(Exponent의 비트 수를 증가시키면), 정밀도는 떨어진다(Significant 비트수가 감소).

### 3. IEEE 754

- 1985년에 채택되고, 2008년에 수정된 IEEE의 부동 소수점 표준.
  - 컴퓨터 사이의 프로그램 이식성(Portability) 향상을 목적으로 개발.
- 표준 포맷
  - Binary32, Binary64, Binary128



Binary32 Format



Binary64 Format



Binary128 Format

### 3.1 IEEE 754 Format Parameter

Parameter	Format		
	Binary32	Binary64	Binary128
Storage width (bits)	32	64	128
Exponent width (bits)	8	11	15
Exponent bias	127	1023	16383
Maximum exponent	127	1023	16383
Minimum exponent	-126	-1022	-16382
Approx normal number range (base 10)	$10^{-38}, 10^{+38}$	$10^{-308}, 10^{+308}$	$10^{-4932}, 10^{+4932}$
Trailing significand width (bits)*	23	52	112
Number of exponents	254	2046	32766
Number of fractions	$2^{23}$	$2^{52}$	$2^{112}$
Number of values	$1.98 \times 2^{31}$	$1.99 \times 2^{63}$	$1.99 \times 2^{128}$
Smallest positive normal number	$2^{-126}$	$2^{-1022}$	$2^{-16382}$
Largest positive normal number	$2^{128} - 2^{104}$	$2^{1024} - 2^{971}$	$2^{16384} - 2^{16271}$
Smallest subnormal magnitude	$2^{-149}$	$2^{-1074}$	$2^{-16494}$

Note: \*not including implied bit and not including sign bit

## 비수치 데이터 표현

## 1. 비수치 데이터

- **BCD(Binary Coded Decimal)**

- 2진화10진코드. 4-bit 2진수를 사용하여 10진 숫자 표시.

- **ASCII Code**

- ANSI에서 제정한 7-bit 코드 체계
- 출력가능한 문자 : 95개(알파벳 대/소문자 52, 숫자 10, 특수문자 32, 공백문자 1)
- 출력할 수 없는 제어 문자 : 33개

- **유니코드(Unicode)**

- 다국적 문자표기를 위해 정의된 문자체계

- **기타 제어 및 통신용 코드 등.**

## 2. BCD

- **BCD(Binary Coded Decimal)**

- 10진수의 각 자리 수를 4비트의 2진수로 직접 변환.
  - 10진법 연산에 편리.

- **BCD 코드 유형**

- 8421 BCD
  - 각 자리에 가중치를(8, 4, 2, 1) 적용.
  - 일반적인 BCD 코드를 말함.
- Excess-3 BCD
  - 8421 BCD 코드에 '3'(0011)을 더하여 만든 코드.
  - 임의의 수에 대해 1's Compliment를 취하면 9의 보수가 됨.

- **6-/7-bit BCD 사용**

- 6-bit BCD : Zone Bit (2) + Digit Bits (4)
- 7-bit BCD : Parity bit (1) + Zone Bit (2) + Digit Bits (4)

Decimal Symbol	BCD Digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

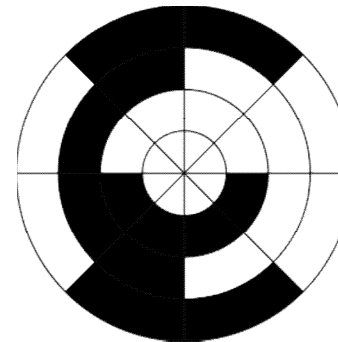


### 3. 그레이 코드

- 그레이 코드(Gray Code)

- 바이너리 코드 값을 증가시키면, 값이 증가할 때마다, 비트 값의 변화가 발생하는데, 이 비트 발생을 최소화한 코드
- 용도
  - 로터리 인코더에 사용 : 인코딩시 glitch 에러 감소.
  - CMOS 회로의 소비전력 감소 : CMOS 회로에서는 비트 변화시에만 전력소모.

Gray Code			
Binary Code	Bit Changes	Gray Code	Bit Changes
000		000	
001	1	001	1
010	2	011	1
011	1	010	1
100	3	110	1
101	1	111	1
110	2	101	1
111	1	100	1
000	3	000	1



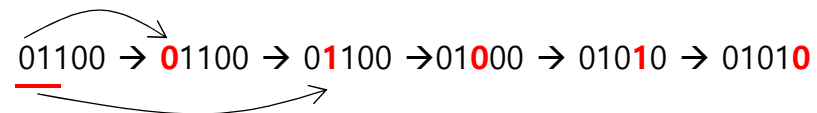
<https://youtu.be/cdeNxFkTwR0>참고

### 3. 그레이 코드

- 2진수를 그레이 코드로 변환하는 방법

- 2진수의 MSB는 그대로 사용하고, 그 다음부터는 차례로 비트 단위로 XOR 연산한 결과 값을 취한다.

01100 → 0**1**100 → 0**1**100 → 01**0**00 → 010**1**0 → 0101**0**



## 4. 문자표현

- 문자를 이진수로 표현
- 표준 문자 표현 방법
  - 문자체계를 이진수로 표현하는 방법을 표준화
    - ASCII : ANSI에서 제안한 7-비트 문자표현 체계
    - EBCDIC : IBM에서 제안한 8-비트 문자표현 체계
    - Unicode : 유니코드 컨소시엄에서 제안한 다국어 지원 코드체계

## 4.1 ASCII

- **ASCII (American Standard Code for Information Interchange)**
  - 7-비트를 사용하여 영문자와 특수문자를 표현한 표준 코드 체계
  - 94개의 문자를 표현가능
    - 26개의 영문 대문자
    - 26개의 영문 소문자
    - 10개의 숫자
    - 32개의 특수문자
  - 34개의 제어용 문자

## 4.2 ASCII 제어문자

### • 34개의 제어용 문자

- 단말기 제어 및 통신용
- 화면표시/인쇄 불가
- 데이터 스트림의 메타 데이터 표현용

제어문자

B <sub>4</sub> B <sub>3</sub> B <sub>2</sub> B <sub>1</sub>	B <sub>7</sub> B <sub>6</sub> B <sub>5</sub>							
	000	001	010	011	100	101	110	111
0000	NULL	DLE	SP	0	@	P	^	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Control Characters

NULL	NULL	DLE	Data link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End of transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

## 4.3 EBCDIC

- **EBCDIC(Extended BCD Interchange Code)**

- IBM에서 개발한 BCD 확장형 코드
  - 많은 수의 영문자, 특수문자, 그래픽 문자 표현 가능
- 8-비트로 2표현.
  - Zone Bits(4) : 각 문자의 4그룹을 식별
  - Digit Bits(4) : 8421 코드를 사용하여 문자를 표현
- 패리티 비트 추가하여 9-bit 로 사용가능
- IBM 대형 컴퓨터에서 사용

## 4.4 유니코드

- 유니코드(Unicode)

- 다국적 언어 문자를 채용하기 위한 산업계 표준
  - ISO-10646으로 정의된 UCS(Universal Character Set)을 말함.
  - 한글 11,172 자 포함
- 코드 포인트(Code Point) : 코드체계 식별자
- 코드 체계
  - UTF-8 : 8-비트 가변길이(1~4바이트) 코드 체계, ASCII 포함.
  - UTF-16 : 16-비트 가변길이(2 또는 4 바이트) 코드 체계
  - UTF-32 : 32-비트 고정길이 코드 체계



## 4.5 한글 코드

- **한글코드 유형**

- 조합형, 완성형, 확장 완성형, 유니코드

- **조합형 코드**

- 한글 초성, 중성, 종성을 조합하여 표현
- 이론적으로 한글 11,172 자를 표현 가능.

- **완성형 코드**

- 연속된 2-바이트를 사용하여 한글 표현. EUC-KR로 표준화.
- 각 바이트는 0xA1~0xFE 사이 값을 사용
- 2,350자 지원

- **확장 완성형 코드**

- 2-바이트 코드체계
- 완성형 코드(EUC-KR)에 코드를 추가한 코드 체계
- 코드 페이지 949(CP949)로 알려짐.



## 5. 에러 감지용 코드

- 패리티(Parity) 비트
  - 에러 감지
- 체크섬(Checksum) 바이트
  - 에러 감지
- 해밍(Hamming) 코드
  - 에러 감지 및 정정이 가능한 코드

## 5.1 패리티 비트

### • 패리티(Parity) 비트

- 데이터 처리 또는 전송과정에서 발생할 수 있는 에러를 감지하기 위한 코드 체계
- 바이너리 데이터에 1-비트(parity bit)를 추가하여, 데이터에 포함된 '1'의 개수가 짝수(Even) 또는(Odd)가 되게 만든다.
  - Even parity : 데이터에 포함된 '1'의 개수가 짝수가 되도록 설정
  - Odd parity : 데이터에 포함된 '1'의 개수가 홀수가 되도록 설정
- 패리티 비트의 위치는 사용목적에 따라 달라진다.

	With Even Parity	With Odd Parity
1000001	01000001	11000001
1010100	11010100	01010100

## 5.2 Checksum 코드

- **체크섬(Checksum)**

- 블록 데이터 전송 또는 저장시 에러를 감지하기 위한 목적으로 계산된 에러 감지 방법.

- **체크섬 계산방법**

- 데이터 블록을 구성하는 바이트를 순차적으로 더한다.
- 캐리를 버린 나머지를 취한다. (Modular 연산)
- 결과 값의 2의 보수 값을 체크섬 바이트로 정한다.
- 데이터 블록에 체크섬 바이트를 붙여서, 저장하거나 전송한다.

- **에러 감지 방법**

- 수신한 데이터 블록이나 읽어온 데이터 블록을 구성하는 바이트들과 체크섬 바이트를 모두 더한다.
- 결과 값이 '0'이면 오류발생이 없음을 의미한다.

## 5.2.1 Checksum 코드 예시

- 송신측 데이터 블록 : 0x23, 0x41, 0x46, 0xAC, 0x1F
- 체크섬 바이트 계산
  - $0x23 + 0x41 + 0x46 + 0xAC + 0x1F = 0x175$
  - Modulo 결과 값 : 0x75
  - 체크섬 바이트 : 0x8B
- 수신 측 데이터 블록 : 0x23, 0x41, 0x46, 0xAC, 0x1F, 0x8B
- 에러 감지를 위한 계산
  - $0x23 + 0x41 + 0x46 + 0xAC + 0x1F + 0x8B = 0x200$
  - Modulo 결과 값 : 0x00
  - 판정 : 0x00 (정상)

## 6. 해밍 코드

- 에러 검출 및 교정이 가능한 코드

- 송신측에서는 특정 비트 위치에 짝수 패리티 비트를 추가.
- 수신측에서는 패리티 비트를 검사한 후, 에러 발생시 수정.

- 해밍 비트 수

- 데이터 비트 수 (n), 해밍 비트 수 (m), 해밍 코드의 길이 (p) 라 할 때 아래 식을 만족하는 최소의 해밍 비트 수를 구한다.

$$2^m \geq p+1 = m+n+1$$
$$m = p-n$$

- 예)  $n=4, p=7 \rightarrow m = 3$

## 6. 해밍 코드

- 해밍 비트의 위치

- $2(m-1)$  ( $m=1\sim3$ ) 번째 비트
- 예)  $m=3$  인 경우, 해밍 비트의 위치는 1-,2-,4-번째 비트

- 해밍 비트의 패리티 비트를 계산하는 위치의 계산법

n	P4	P2	P1
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

$$P1 = 1, 3, 5, 7$$

$$P2 = 2, 3, 6, 7$$

$$P4 = 4, 5, 6, 7$$

## 6.1 해밍 코드 (4-비트 데이터)

### • 데이터 비트가 4-비트(0101) 에 대한 해밍 코드 계산

- 해밍 비트 수 : 3
- 해밍 비트 위치 : 1-,2-,4-번째 비트
- 해밍 코드의 길이 : 7-비트

7						1
D3	D2	D1	P4	D0	P2	P1
0	1	0	P4	1	P2	P1
7		5		3		1
0	1	0	P4	1	P2	1
7	6			3	2	
0	1	0	P4	1	0	1
7	6	5	4			
0	1	0	1	1	0	1

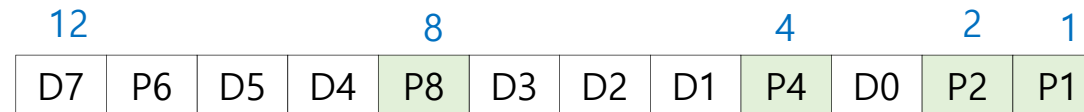
P1 = 1, 3, 5, 7

P2 = 2, 3, 6, 7

P4 = 4, 5, 6, 7

## 6.2 해밍 코드 (8-비트 데이터)

- 데이터 비트가 8-비트인 경우 해밍 코드 위치
  - 해밍 비트 수 : 4
  - 해밍 비트 위치 : 1-,2-,4-,8-번째 비트
  - 해밍 코드의 길이 : 12-비트





## 6.3 해밍 코드 비용

Data Bits	Single-Error Correction		Single-Error Correction/ Double-Error Detection	
	Check Bits	% Increase	Check Bits	% Increase
8	4	50	5	62.5
16	5	31.25	6	37.5
32	6	18.75	7	21.875
64	7	10.94	8	12.5
128	8	6.25	9	7.03
256	9	3.52	10	3.91

**end**