Computer System Architecture

(THIRD EDITION)

M. Morris Mano

PRENTICE HALL

3-1. Data Types

Data Types

- Binary information is stored in ***memory*** or ***processor registers***
- Registers contain either ***data*** or ***control information***
  - ***Data*** are numbers and other binary-coded information
  - ***Control information*** is a bit or a group of bits used to specify the sequence of command signals
- Data types found in the registers of digital computers
  - ***Numbers*** used in arithmetic computations
  - ***Letters*** of the alphabet used in data processing
  - ***Other discrete symbols*** used for specific purpose
    - » 위의 Number 와 Letter 이외 모두, 예) gray code, error detection code, …
- The binary number system is the most natural system to use in a digital computer
- Number Systems
  - ***Base*** or ***Radix r system*** : uses distinct symbols for ***r digits***
  - Most common number system :Decimal, Binary, Octal, Hexadecimal
  - Positional-value(weight) System : $\mathbf{r^2\ r^1 r^0 . r^{-1}\ r^{-2}\ r^{-3}}$
    - » Multiply each digit by an integer power of r and then form he sum of all weighted digits

3-1. Data Types

- **Decimal System/Base-10 System**

  Composed of 10 symbols or numerals(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0)

- **Binary System/Base-2 System**

  Composed of 10 symbols or numerals(0, 1)

  Bit = Binary digit

- **Hexadecimal System/Base-16 System : Tab. 3-2**

  Composed of 16 symbols or numerals(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)

- **Binary-to-Decimal Conversions**

$$1011.101_2 = (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^o) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3})$$
$$= 8_{10} + 0 + 2_{10} + 1_{10} + 0.5_{10} + 0 + 0.125_{10}$$
$$= 11.625_{10}$$

- **Octal-to-Decimal Conversions**

$$(736.4)_8 = 7 \times 8^2 + 3 \times 8^1 + 6 \times 8^0 + 4 \times 8^{-1}$$
$$= 7 \times 64 + 3 \times 8 + 6 \times 1 + 4/8 = (478.5)_{10}$$

- **Hexadecimal-to-Decimal Conversions**

$$(F3)_{16} = F \times 16 + 3 = 15 \times 16 + 3 = (243)_{10}$$

3-1. Data Types

◆ Conversion of decimal 41.6875 into binary

Repeated division(See p. 69, ***Fig. 3-1***)

8 4 2 1

Integer = 41

41 / 2 = 20 remainder 1 (binary number will end with 1) : **LSB**

20 / 2 = 10 remainder 0

10 / 2 = 5 remainder 0

5 / 2 = 2 remainder 1

2 / 2 = 1 remainder 0

1 / 2 = 0 remainder 1 (binary number will start with 1) : **MSB**

Read the result upward to give an answer of $(41)_{10} = (101001)_2$

Fraction = 0.6875

0.6875 x 2 = 1.3750 integer 1 : **MSB**

1.3750 x 2 = 0.7500 integer 0 .

0.7500 x 2 = 1.5000 integer 1

1.5000 x 2 = 1.0000 integer 1 : **LSB**

Read the result downward $(0.6875)_{10} = (0.1011)_2$

$(41.6875)_{10} = (101001.1011)_2$

곱한 결과 소수점 윗자리가 2로 나오면 0으로 고치고, 3이 나오면 1로 고치어 계산한다

3-1. Data Types

Table 3-2 Hex	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15
14	0001 0100	20
F8	1111 1000	248

◆ Hex-to-Decimal Conversion

$$2AF_{16} = (2 \times 16^2) + (10 \times 16^1) + (15 \times 16^0)$$
$$= 512_{10} + 160_{10} + 15_{10}$$
$$= 687_{10}$$

◆ Decimal-to-Hex Conversion

423_{10} / 16 = 26 remainder 7 (Hex number will end with 7) : **LSB**

26_{10} / 16 = 1 remainder 10

1_{10} / 16 = 0 remainder 1 (Hex number will start with 1) : **MSB**

Read the result upward to give an answer of $423_{10} = 1A7_{16}$

◆ Hex-to-Binary Conversion

$9F2_{16}$ = 9 F 2

$\downarrow$ $\downarrow$ $\downarrow$

= 1001 1111 0010

$= 100111110010_2$

◆ Binary-to-Hex Conversion

$$1\,1\,1\,0\,1\,0\,0\,1\,1\,0_2 = \underbrace{0\,0\,1\,1}_{3}\ \underbrace{1\,0\,1\,0}_{A}\ \underbrace{0\,1\,1\,0}_{6}$$

$= 3A6_{16}$

◆ Binary, octal, and hexadecimal Conversion

$$\overbrace{1}^{1}\ \overbrace{0\,1\,0}^{2}\ \overbrace{1\,1\,1}^{7}\ \overbrace{1\,0\,1}^{5}\ \overbrace{1\,0\,0}^{4}\ \overbrace{0\,1\,1}^{3}$$

$$\underbrace{1\,0\,1\,0}_{A}\ \underbrace{1\,1\,1\,1}_{F}\ \underbrace{0\,1\,1\,0}_{6}\ \underbrace{0\,0\,1\,1}_{3}$$

Octal 8진수

Binary 2진수

Hexadecimal 16진수

3-1. Data Types

- **Binary-Coded-Decimal Code**

  Each digit of a decimal number is represented by its binary equivalent

8	7	4	**(Decimal)**
↓	↓	↓	
1000	0111	0100	**(BCD)**

  Only the four bit binary numbers from 0000 through 1001 are used

  Comparison of BCD and Binary

  $137_{10} = 10001001_2$ **(Binary) -** *require only 8 bits*

  $137_{10} = 0001\ 0011\ 0111_{BCD}$ **(BCD) -** *require 12 bits*

- **Alphanumeric Representation**

  Alphanumeric character set(***Tab. 3-4***)

  » 10 decimal digits, 26 letters, special character($, +, =,….)

  » A complete list of ASCII : p. 384, Tab. 11-1

  ASCII(American Standard Code for Information Interchange)

  » Standard alphanumeric binary code uses seven bits to code 128 characters

  - Unicode
    - 16bits code ($2^{16} = 65536$)

□ 코드는 다양한 정보를 2진수로 표현하기 위한 이진코드의 종류다

3-1. Data Types

Character	Binary code	Character	Binary code	Character	Binary code	Character	Binary code
A	100 0001	U	101 0101	0	011 0000	/	010 1111
B	100 0010	V	101 0110	1	011 0001	,	010 1100
C	100 0011	W	101 0111	2	011 0010	=	011 1101
D	100 0100	X	101 1000	3	011 0011		
E	100 0101	Z	101 1010	4	011 0100		
F	100 0110			5	011 0101		
G	100 0111			6	011 0110		
H	100 1000			7	011 0111		
I	100 1001			8	011 1000		
J	100 1010			9	011 1001		
K	100 1011						
L	100 1100						
M	100 1101			space	010 0000		
N	100 1110			.	010 1110		
O	100 1111			(	010 1000		
P	101 0000			+	010 1011		
Q	101 0001			$	010 0100		
R	101 0010			*	010 1010		
S	101 0011			)	010 1001		
T	101 0100			-	010 1101		

Table 3-4. ASCII

3-2. Complements

보수

6에 대한 9의 보수는 3, (3+6=9)

Complements are used in digital computers for simplifying the ***subtraction operation*** and for logical manipulation

- There are two types of complements for base r system
  1) r's complement 2) (r-1)'s complement
  - » Binary number : 2's or 1's complement
  - » Decimal number : 10's or 9's complement

간단하게 생각하면,

546700

453299

각자리 더하면 9가됨

- (r-1)'s Complement

  (r-1)'s Complement of N = $(r^n-1)-N$

  N : given number
  r : base
  n : digit number

  - » 9's complement of N=***546700***

    9의보수,

    (10^6-1)-546700= (1000000-1)-546700= 999999-546700

    = ***453299***

    546700(N) + 453299(9's com) =999999

  - » 1's complement of N=***101101***

    (2^6-1)-101101= (1000000-1)-101101= 111111-101101

    = ***010010***

    1의보수를 구하려면 비트를 다 바꾼다

    101101(N) + 010010(1's com) =111111

- r's Complement

  r's Complement of N = r^n-N

  * ***r's Complement***
  (r-1)'s Complement +1 =$(r^n-1)-N+1= r^n-N$

  10의 보수 = 9의보수 + 1
  - » 10's complement of ***2389***= 7610+1= ***7611***

  2의보수 = 1의보수 + 1
  - » 2's complement of ***1101100***= 0010011+1= ***0010100***

3-2. Complements

- **Subtraction of Unsigned Numbers** (M-N), N≠0
  1) M + (r^n-N)
  2) M ≥ N : Discard end carry, Result = M-N
  3) M < N : No end carry, Result = - r's complement of (N-M)

» *Decimal Example)*

M ≥ N

72532(M) - 13250(N) = 59282

72532

+ 86750 (10's complement of 13250)

Discard End Carry

1 59282

Result = *59282*

M < N

13250(M) - 72532(N) = -59282

13250

+ 27468 (10's complement of 72532)

No End Carry

0 40718

Result = -(10's complement of 40718)

= -(59281+1) = *-59282*

» *Binary Example)*

X ≥ Y

1010100(X) - 1000011(Y) = 0010001

1010100

+ 0111101 (2's complement of 1000011)

1 0010001

Result = *0010001*

X < Y

1000011(X) - 1010100(Y) = -0010001

1000011

+ 0101100 (2's complement of 1010100)

0 1101111

Result = -(2's complement of 1101111)

= -(0010000+1) = *-0010001*

3-2. Complements

2진수의 뺄셈

[1's Complement]

10진법	9'Complement	보통방법	1'Complement	
6	6	0 1 1 0	0 1 1 0	
— 4	+ 5	— 0 1 0 0	+ 1 0 1 1	← 0 1 0 0에 대한 1'Complement
2	1 1	0 0 1 0	1 0 0 0 1	
	+ 1		+ 1	← 자리 올림수를 더한다
	2		0 0 1 0	← 연산결과

9의 보수

1의 보수

carry는 아래로

내린다

4		0 1 0 0	0 1 0 0	
— 6		— 0 1 1 0	+ 1 0 0 1	← 0 1 1 0 에 대한 1'Complement
— 2		— 0 0 1 0	□ 1 1 0 1	
			↓	← 1의 보수로 변환 한다
			— 0 0 1 0	← 연산결과 (자리올림이 없으므로 '—'부호 를 붙인다)

1의보수

2진수의 뺄셈

[2's Complement]

10의 보수 · 2의보수 · 올림수 삭제

10진법	10'Complement	보통방법	2'Complement	
6	6	0 1 1 0	0 1 1 0	
— 4	+ 6	— 0 1 0 0	+ 1 1 0 0	← 0 1 0 0에 대한 2'Complement
2	~~1~~2	0 0 1 0	~~1~~0 0 1 0	← 올림수 무시
	2		0 0 1 0	← 연산결과

2의보수

4	0 1 0 0	0 1 0 0	
— 6	— 0 1 1 0	+ 1 0 1 0	← 0 1 1 0 에 대한 2'Complement
— 2	— 0 0 1 0	□ 1 1 1 0	
		↓	← 자리 올림이 없으므로2의 보수로 변환
		— 0 0 1 0	← 하고 '—'부호를 붙인다)

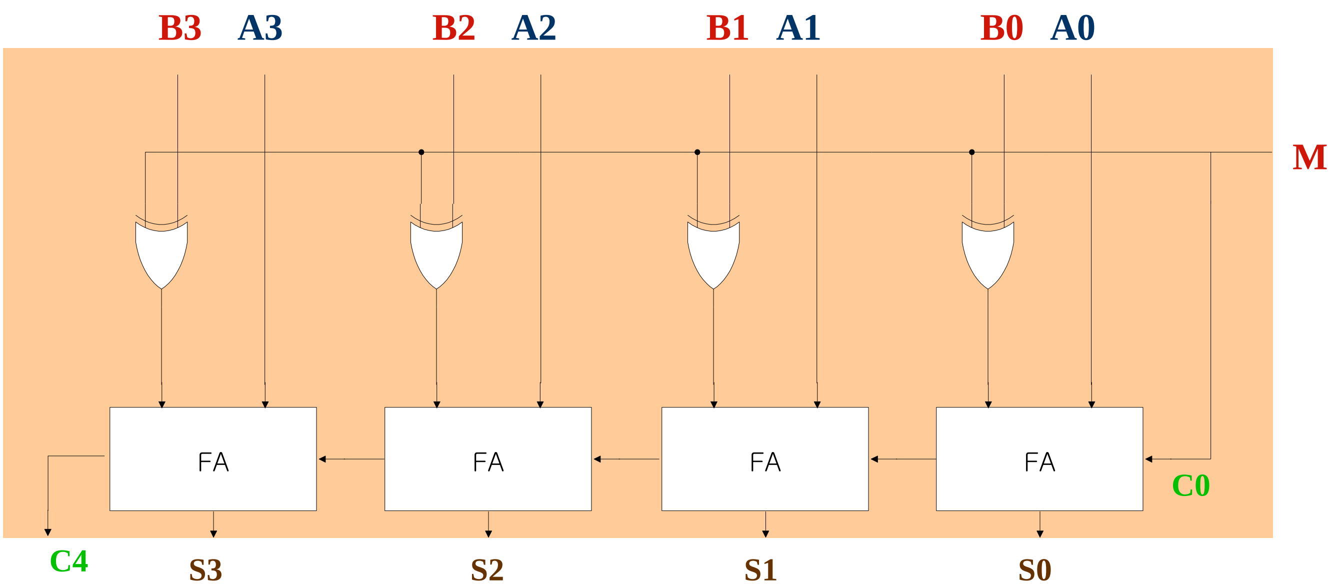
가감산기

4-bit Binary Adder-Subtractor

4-bit Binary Adder-Subtractor : ***Fig. 4-7***

- One common circuit by including an exclusive-OR gate with each full-adder
- M =0 : Adder $B \oplus M + C = B \oplus 0 + 0 = B$, $\therefore$ ***A + B***
- M =1 : Subtractor $B \oplus M + C = B \oplus 1 + 1 = B' + 1 = -B$ (2's comp), $\therefore$ ***A - B***

XOR : 달라야 1



3-3. Fixed-Point Representation

*Numeric Data
1) Fixed Point 고정소수점
2) Floating Point 부동소수점

Fixed-Point Representation

- Computers must represent everything with 1's and 0's, including the sign of a number and fixed/floating point number
- Binary/Decimal Point

  * 32.25
  1) 0.25, 2) 32.0, 3) 32.25

  The position of the binary/decimal point is needed to represent ***fractions***, ***integers***, or ***mixed integer-fraction*** number
- Two ways of specifying the position of the binary point in a register

  1) Fixed Point : the binary point is always fixed in one position
     » A binary point in the ***extreme left*** of the register(***Fraction*** : 0.xxxxx)
     » A binary point in the ***extreme right*** of the register(***Integer*** : xxxxx.0)

  0 0001110
  MSB

  The binary point is not actually present, but the number stored in the register is treated as a fraction or as an integer

  2) Floating Point : the second register is used to designate the position of the binary point in the first register(***refer to 3-4***)

- Integer Representation

  * MSB for Sign
  "0" is plus +
  "1" is minus -

  Most Common

Representation	+14	-14
Signed-magnitude representation	0 0001110	1 0001110
Signed-1's complement representation	0 0001110	1 1110001
Signed-2's complement representation	0 0001110	1 1110010

3-3. Fixed-Point Representation

Arithmetic Addition

Addition Rules of Ordinary Arithmetic

» The signs are ***same*** : sign= ***common*** sign, result= ***add***

» The signs are ***different*** : sign= ***larger*** sign, result= ***larger-smaller***

(-12) + (-13) = -25
(+12) + (+13) = +25

(+25) + (-37)
= 37 - 25 = -12

Addition Rules of the signed 2's complement

» Add the two numbers including their sign bits

» Discard any carry out of the sign bit position

***Addition Exam)**

+ 6	00000110	- 6	11111010
+ 13	00001101	+ 13	00001101
+ 19	00010011	+ 7	00000111
+ 6	00000110	- 6	11111010
- 13	11110011	- 13	11110011
- 7	11111001	- 19	11101101

Arithmetic Subtraction

Subtraction is changed to an Addition

» (± A) - (+ B) = (± A) + (- B)

» (± A) - (- B) = (± A) + (+ B)

*** Subtraction Exam)** (- 6) - (- 13) = +7

11111010 - 11110011 = 11111010 + 2's comp of 11110011

= 11111010 + 00001101

= 1 00000111 = +7

Discard End Carry

Overflow

Two numbers of n digits each are added and the sum occupies n+1 digits

n + 1 bit cannot be accommodated in a register with a standard length of n bits(*many computer detect the occurrence of an overflow, and a corresponding F/F is set*)

3-3. Fixed-Point Representation

Overflow 문제점

An overflow may occur if the two numbers added are both positive or both negative

» When two unsigned numbers are added

an overflow is detected from the end carry out of the MSB position

» When two signed numbers are added

the MSB always represents the sign

- the sign bit is treated as part of the number

- the end carry does not indicate an overflow

BCD코드를 연산
하려면 특별한
연산기가 필요하다

*** Overflow Exam)**

	out in		*out in*
carries	0 1	carries	1 0
+ 70	0 1000110	- 70	1 0111010
+ 80	0 1010000	- 80	1 0110000
+ 150	1 0010110	- 150	0 1101010

Overflow Detection

Detected by observing the ***carry into*** the sign bit position and the ***carry out*** of the sign bit position

If these two carries are not equal, an overflow condition is produced(***Exclusive-OR gate = 1***)

Decimal Fixed-Point Representation

A 4 bit decimal code requires four F/Fs for each decimal digit

The representation of 4385 in BCD requires 16 F/Fs (0100 0011 1000 0101)

The representation in decimal is *wasting a considerable amount of storage* space and the circuits required to perform decimal arithmetic are *more complex*

***Decimal Exam) (+375) + (-240)**

375 + (10's comp of 240)= 375 + 760

0 375	(0000 0011 0111 0101)
+9 760	(1001 0111 0110 0000)
0 135	(0000 0001 0011 0101)

* Advantage *
Computer I/O data are generated by people who use the decimal system

3-4 Floating-Point Representation

- The floating-point representation of a number has two parts

  가수 -> 1) Mantissa : signed, fixed-point number

  지수 -> 2) Exponent : position of binary(decimal) point

* Decimal + 6132.789	
Fraction	***Exponent***
+0.6132789	+4

  정규화

  $\rightarrow$ 0.6132789 x 10^4

- Scientific notation : $m \times r^e$ ($+0.6132789 \times 10^{+4}$)

  m : mantissa, ***r*** : radix, ***e*** : exponent

- Example : $m \times 2^e = +(.1001110)_2 \times 2^{+4}$

Fraction	***Exponent***
01001110	000100

  가수 x 지수

- 정규화 Normalization - 가장 높은 정밀도 제공

  Most significant digit of mantissa is ***nonzero***

  유효자리를 최대로 하기 위해 가수부분의 값이 **0.1 ~ 1** *사이에 있도록 조작 하는 것*

  $(0.0000145786)_{10}$를 정규화 하면 0.145786×10^{-4} 소수점 아래 1번째에 수가 올 것

0	1 ~ 7	8 ~ 31
Sign	Exponent	Mantissa
MSB	지수	가수

32bit

Virtual point

0 → +

1 → -

3-5. Other Binary Codes

Gray Code

한 비트만 바뀜

Gray code ***changes by only one bit*** (***Tab. 3-5*** 4-bit Gray Code)

용도 :

» The data must be converted into digital form before they can be used by a digital computer(***Analog to Digital Converter***)

» The analog data are represented by the continuous change of a shaft position(***Rotary Encoder of Motor***)

BCD→Gray

처음 비트는 그대로 내려오고 두번째부터는 2개 더한 값

Binary number → Gray code

이웃해 있는 2진수의 합만을 다음의 그레이코드 비트로...

1 + 0 1 0 1 1 0 0 1 1 1 (binary number)

1 1 1 1 1 0 1 0 1 0 0 (gray code)

Gray→BCD

처음 비트는 그대로 내려오고

Gray code → Binary number

대각선으로 합해서 올림은 버리고 합만을 2진수로...

1 1 1 1 1 0 1 0 1 0 0 (gray code)

\+

1 0 1 0 1 1 0 0 1 1 1 (binary number)

3-5. Other Binary Codes

1비트만 변함

BCD code	
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Binary code	Decimal equivalent	Binary code	Decimal equivalent
0000	0	1100	8
0001	1	1101	9
0011	2	1111	10
0010	3	1110	11
0110	4	1010	12
0111	5	1011	13
0101	6	1001	14
0100	7	1000	15

Table 3-5. 4-Bit Gray Code

3-5. Other Binary Codes

그 외 이진 코드

◆ Other Decimal Codes

Gray Code 외에도 더 있다

Binary codes for decimal digits require four bits. A few possibilities are shown in ***Tab. 3-6***

Excess-3 Gray Code

» Gray code로 BCD 표현 시, 9 에서 0 으로 변하면 1101에서 0000으로 되어 3 비트가 동시에 변경되어 Tab. 3-5 에서 3 부터 12까지 사용하면 0010 에서 1010되어 1비트가 바뀜.

Self-Complementing : excess-3 code

» 9's complement of a decimal number is easily obtained by 1's complement(=changing 1's to 0's and 0's to 1's)

Weighted Code : 2421 code 보수를 구하기 쉽다

» The bits are multiplied by the weights, and the sum of the weighted bits gives the decimal digit

*** Self-Complement Exam)**

4_{10} = 0111 (3-excess)
= 1000 (1's comp)
= 5_{10} (3-excess in Tab. 3-6)
= 5_{10}(***9's comp of 4***)

◆ Other Alphanumeric Codes

7bit ASCII Code에서 Tab. 3-4 이외 : p. 384, Tab. 11-1

» Format effector : Functional characters for controlling the layout of printing or display devices(carriage return-CR, line feed-LF, horizontal tab-HT,…)

» Data communication flow control(acknowledge-ACK, escape-ESC, synchronous-SYN,…)

8bit EBCDIC(Extended BCD Interchange Code)

» Used in IBM equipment(제어 문자만 약간 다름)

3-5. Other Binary Codes

Decimal digit	BCD 8421	2421	Excess-3	Excess-3 gray
0	0000	0000	0011	0010
1	0001	0001	0100	0110
2	0010	0010	0101	0111
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1100
6	0110	1100	1001	1101
7	0111	1101	1010	1111
8	1000	1110	1011	1110
9	1001	1111	1100	1010
	1010	0101	0000	0000
Unused	1011	0110	0001	0001
bit	1100	0111	0010	0011
combi-	1101	1000	1101	1000
nations	1110	1001	1110	1001
	1111	1010	1111	1011

\+

보수

보수

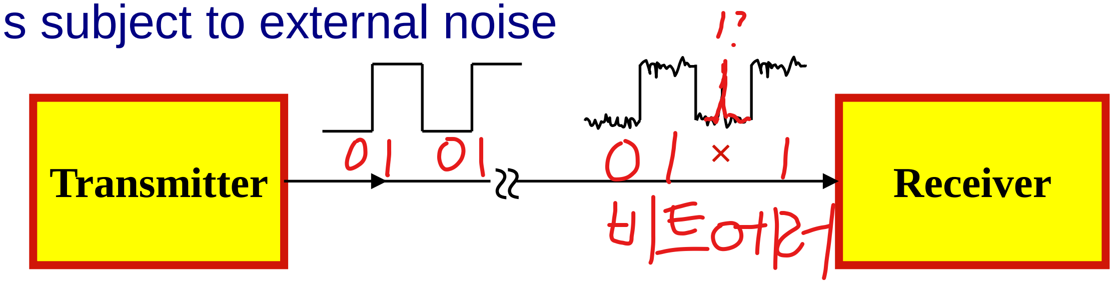
필요에 따라 여러코드 만들 수 있다

Table 3-6. Four Different Binary Codes for the Decimal Digit

비트 에러 방지 코드

3-6 Error Detection Codes

- Binary information transmitted through some form of communication medium is subject to external noise



- Parity Bit

  An extra bit included with a binary message to make the total number of 1's either odd or even(***Tab. 3-7***) 1의 개수 세고 짝수/홀수인지 검사

- Even-parity method

  짝수

  The value of the parity bit is chosen so that the total number of 1s *(including the parity bit)* is an ***even*** number

  1 1 0 0 0 0 1 1

  Added parity bit

  짝수로 맞춰주니, 나중에 다시 뗌

- Odd-parity method

  홀수

  Exactly the same way except that the total number of 1s is an ***odd*** number

  1 1 0 0 0 0 0 1

  Added parity bit

  홀수로 보내기로 약속했는데 1이 짝수면 재전송 요청.

3-6. Error Detection Codes

- Parity Generator/Checker
  - At the sending end, the message is applied to a parity generator
  - At the receiving end, all the incoming bits are applied to a parity checker

1 1 0 0 0 0 1 1 "C"	**1 1 0 0 0 0 1 0 "B"**
(Even-parity Generator)	**(Even-parity Checker)**
보낼때 1 붙임	받을때 검사

  - Can not tell which bit in error
  - Can detect only single bit error(odd number of errors)
  - 3 bit data line example : ***Fig. 3-3***
- 4 bit data line example :

Next Page

Odd Parity Generator/Checker

◆ Truth Table

even/odd

A	B	C	D	E	O
0	0	0	0	0	1
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	1

◆ K-Map*(Odd Parity)*

C

	0	1	3	2
B	4	5	7	6
A, B	12	13	15	14
A	8	9	11	10

D

XOR는 1의 개수가
홀수면 1, 짝수면 0

◆ Expression

$$\bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}CD + \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + AB\bar{C}\bar{D} + ABCD + A\bar{B}\bar{C}D + A\bar{B}C\bar{D}$$
$$= \bar{A}\bar{B}(\bar{C}\bar{D} + CD) + \bar{A}B(\bar{C}D + C\bar{D}) + AB(\bar{C}\bar{D} + CD) + A\bar{B}(\bar{C}D + C\bar{D})$$
$$= \bar{A}\bar{B}(C \otimes D) + \bar{A}B(C \oplus D) + AB(C \otimes D) + A\bar{B}(C \oplus D)$$
$$= (C \otimes D)(\bar{A}\bar{B} + AB) + (C \oplus D)(\bar{A}B + A\bar{B})$$
$$= (C \otimes D)(A \otimes B) + (C \oplus D)(A \oplus B)$$
$$= \overline{(C \oplus D)}\,\overline{(A \oplus B)} + (C \oplus D)(A \oplus B)$$
$$= \bar{x}\bar{y} + xy$$
$$= x \otimes y$$
$$= \overline{x \oplus y}$$
$$= \overline{(C \oplus D) \oplus (A \oplus B)}$$
$$= \overline{C \oplus D \oplus A \oplus B}$$

$\oplus$: XOR
$\otimes$: XNOR

$x = C \oplus D$
$y = A \oplus B$

☞ Parity check

Error 검출용 비트를 하나 더 추가 시켜서 언제나 전체 부호 속에 포함 되어 있는 1의 수가 홀수 또는 짝수개가 되도록 하는 것

10진수	2^3	2^2	2^1	2^0	패리티비트	1의 합계
0	0	0	0	0	1	1
1	0	0	0	1	0	1
2	0	0	1	0	0	1
3	0	0	1	1	1	3
4	0	1	0	0	0	1
5	0	1	0	1	1	3
6	0	1	1	0	1	3
7	0	1	1	1	0	3
8	1	0	0	0	0	1
9	1	0	0	1	1	3

패리티비트는 단순하지만 기능은 강하지 않아서, 한 비트 에러난건 알아낼 수 있지만 어느 자리에서 에러났는지는 X, 같은 홀,짝일때도 X,

3-6. Error Detection Codes

☞ Word parity 패리티비트를 업그레이드, 상황에 따라 좋은 게 다름,

코드를 한 묶음 단위로 하는 Block단위, 에러 검출은 물론 정정 까지도 가능 하다

Odd parity bit

0	1	1	1	1	0	0	1
0	0	0	0	0	1	1	1
1	0	0	0	1	1	1	1
0	1	1	0	1	1	1	0
0	0	1	1	0	0	0	1
0	1	0	1	1	0	0	0
1	1	1	1	0	0	0	1
1	1	1	1	1	0	0	0

Odd parity word

0	1	1	1	1	0	0	1	
0	0	0	0	0	1	1	1	
1	0	**1**	0	1	1	1	1	Parity Error검출 (짝수)
0	1	1	0	1	1	1	0	
0	0	1	1	0	0	0	1	
0	1	0	1	1	0	0	0	
1	1	1	1	0	0	0	1	
1	1	1	1	1	0	0	0	

Parity Error검출 (짝수)

가로 세로 모두 홀짝 일치