Computer System Architecture

데 이 터 표 현

- 데이터 종류
- 보완
- 고정 소숫점 표현
- 유동 소숫점 표현
- 다른 **2**진 표현
- 에러 검출 코드

자 료 표 현

❖**Computer**가 다루는 정보

* 데이터
  - 수치 데이터
    숫자 (정수, 실수)
  - 수가 아닌 데이터
    문자, 기호

* 데이터 성분 사이의 구조
  - 데이터 구조
    Linear Lists, Trees, Rings, etc

* **Program(Instruction)**

수치 데이터 표현

❖ 데이타

수치 자료 - 수(정수, 실수)
기호 자료 - 기호, 문자

❖ 진 법

Nonpositional number system

- 로마식 표기

Positional number system (진수)

- 각 자리에 따라서 값을 가지고 있다.
- Decimal(10진), Octal(8진), Hexadecimal(16진), Binary(2진)

❖ **Base (or radix) R number**

- R은 각자리의 의미를 정확하게 부여한다.
- Example $A_R = a_{n-1}\, a_{n-2} \ldots a_1\, a_0 .a_{-1} \ldots a_{-m}$

Radix point(.)
완전수의 자리와 기수의 자리를 나눈다.

- $V(A_R) = \sum_{i=-m}^{n-1} a_i R^i$

R = 10 Decimal number system, R = 2 Binary
R = 8 Octal, R = 16 Hexadecimal

왜 디지털 컴퓨터에서는 진수를 사용할까?

❖ 중요점은 비용과 시간이다.

- 하드웨어 설계에 드는 비용.

  Arithmetic and Logic Unit, CPU, Communications

- 연산에 필요한 시간.

❖ 산술 – 수치 연산 - 테이블 연산

*** Non-positional Number System**

- 테이블 연산을 통해 무한히 연산 가능하다.

  -->설계가 불가능하다.

  (설계시 매우 비싸다.**)**

*** Positional Number System**

- 테이블 연산에 한계가 있다.

  --> 물리적으로 실현할수 있다.

- 비용 때문에 작은 테이블일 수록 비용이 적게 든다.

  --> **2**진수가 **10**진으로 변하기 쉽다.

Binary Addition Table

	0	1
0	0	1
1	1	10

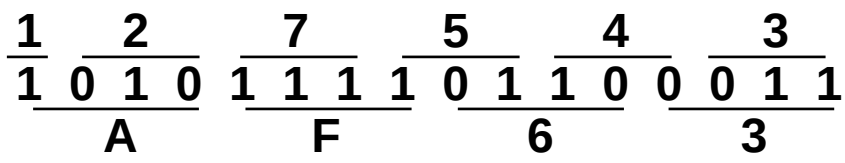
Decimal Addition Table

	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

REPRESENTATION OF NUMBERS - POSITIONAL NUMBERS

Decimal	Binary	Octal	Hexadecimal
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	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

Binary, octal, and hexadecimal conversion



진 수 변 환

❖ 십진수의 표시

$$A = a_{n-1}\, a_{n-2}\, a_{n-3} \ldots a_0 \,.\, a_{-1} \ldots a_{-m}$$

$$V(A) = \sum a_k R^k$$

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

$= 7 \times 64 + 3 \times 8 + 6 \times 1 + 4/8 = (478.5)_{10}$

$(110110)_2 = \ldots = (54)_{10}$

$(110.111)_2 = \ldots = (6.785)_{10}$

$(F3)_{16} = \ldots = (243)_{10}$

$(0.325)_6 = \ldots = (0.578703703 \ldots\ldots\ldots\ldots)_{10}$

❖ **10**진수를 변환 하기

- Separate the number into its *integer* and *fraction* parts and convert each part separately.
- Convert *integer part* into the base R number

  → **R**로 연달아 나눈후 나머지 값을 정리 한다.
- Convert *fraction part* into the base R number

  →정수 자리를 연달아 증가 시키면서 정리(축척) 한다.

EXAMPLE

Convert 41.6875_{10} to base 2.

Integer = 41

41	
20	1
10	0
5	0
2	1
1	0
0	1

Fraction = 0.6875

0.6875
x 2
1.3750
x 2
0.7500
x 2
1.5000
x 2
1.0000

$(41)_{10} = (101001)_2$

$(0.6875)_{10} = (0.1011)_2$

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

Exercise

Convert $(63)_{10}$ to base 5: $(223)_5$
Convert $(1863)_{10}$ to base 8: $(3507)_8$
Convert $(0.63671875)_{10}$ to hexadecimal: $(0.A3)_{16}$

보 수

❖ **R** 진법에는 두가지 방법이 있다.

- **R** 보수와 **(R-1)**보수

(R-1)보수

각 자리 수를 **(R-1)로뺀것과 같다.**

Example

- **9**보수 835_{10} 는 164_{10} 이다.
- **1**의 보수 1010_2 는 0101_2 이다. **(bit by bit complement operation)**

R의 보수

(R-1)의 보수의 마지막 자리에 **1**을 더한다.

Example

- **10**의 보수 835_{10} 는 $164_{10} + 1 = 165_{10}$
- **2**의 보수 1010_2 는 $0101_2 + 1 = 0110_2$

고정 소숫점

- **Numbers :** 고정 소숫점과 유동 소숫점.

- **2**진 고정 소숫점의 표현

  $X = x_n x_{n-1} x_{n-2} \ldots x_1 x_0 . x_{-1} x_{-2} \ldots x_{-m}$

  Sign Bit(x_n): 0 for positive - 1 for negative

  Remaining Bits($x_{n-1} x_{n-2} \ldots x_1 x_0 . x_{-1} x_{-2} \ldots x_{-m}$)

부호화된 수

❖ 양수 음수 두자기를 모두 표현 한다.

- 3가지 표현에 따라서

- 부호 절대값 표현
- 부호화된 **1**의 보수 표현
- 부호화된 **2**의 보수 표현

Example

+9 와 **-9** 를 **7 bit-binary number**로 표현시

+9의 표현 **==> 0 001001**

3가지 방법에 의한 **-9** 표현

- 부호 절대표현에 의해 : 1 001001
- 부호화된 1의 보수 표현에 의해 : 1 110110
- 부호화된 2의 보수 표현에 의해 : 1 110111

일반적으로 컴퓨터에서 고정된 소숫점은 표현은 오직 **integer part** 로 표현 되거나 **fractional part** 로 표현된다.

CHARACTERISTICS OF 3 DIFFERENT REPRESENTATIONS

❖ 보완

부호 절대값 : 오직 기호비트만 보수를 취한다.
부호화된 **1**의 보수 : 모든 비트를 보수를 취한다.
부호화된 **2**의 보수 : 모든 비트에 보수를 취한후 하위**1**비트를 더한다.

최대값과 최소값는 숫자와 **0**으로 나타내어 진다.

$$X = x_n\ x_{n-1} \ldots x_0 \cdot x_{-1} \ldots x_{-m}$$

Signed Magnitude

Max:	$2^n - 2^{-m}$	011 ... 11.11 ... 1
Min:	$-(2^n - 2^{-m})$	111 ... 11.11 ... 1
Zero:	+0	000 ... 00.00 ... 0
	-0	100 ... 00.00 ... 0

Signed 1's Complement

Max:	$2^n - 2^{-m}$	011 ... 11.11 ... 1
Min:	$-(2^n - 2^{-m})$	100 ... 00.00 ... 0
Zero:	+0	000 ... 00.00 ... 0
	-0	111 ... 11.11 ... 1

Signed 2's Complement

Max:	$2^n - 2^{-m}$	011 ... 11.11 ... 1
Min:	-2^n	100 ... 00.00 ... 0
Zero:	0	000 ... 00.00 ... 0

2의 보수의 중요성

- Signed 2's complement representation follows a "weight" scheme similar to that of unsigned numbers

  - 비트표현은 음수를 가지고 있다.
  - 다른 비트는 기존 값을 가진다.

$$X = x_n\ x_{n-1} \cdots x_0$$

$$\rightarrow\ V(X) = -x_n \times 2^n + \sum_{i=0}^{n-1} x_i \times 2^i$$

덧셈 연산
절대값 표현

[1] 부호를 비교한다.
[2] 만약, 두수의 부호가 같으면 두수를 더한다.

-*overflow 를 주의 한다.*

[3] 같지 않다면, 두수의 크기를 비교하고 큰수로 부터 작은수를 뺀다.

--> 덧셈연산을 위해 필요 하다.

[4] Determine the sign of the result

6 + 9

```
  6   0110
+) 9   1001
 15   1111 -> 01111
```

-6 + 9

```
  9   1001
- ) 6   0110
  3   0011 -> 00011
```

6 + (- 9)

```
  9   1001
-) 6   0110
- 3   0011 -> 10011
```

-6 + (-9)

```
  6   0110
+) 9   1001
-15   1111 -> 11111
```

Overflow **9 + 9 or (-9) + (-9)**

```
    9   1001
+)  9   1001
overflow (1)0010
```

덧셈 연산
2의 보수

두수를 더할때 부호비트까지 연산하여 케리 비트를 가지고 부호를 결정한다.

– ***overflow*를 주의 한다.**

Example

```
   6   0   0110
+) 9   0   1001
  15   0   1111
```

```
   -6   1   1010
+) 9    0   1001
   3    0   0011
```

```
   6   0   0110
+) -9  1   0111
   -3  1   1101
```

```
   -9    1  0111
+) -9    1  0111
  -18  (1)0  1110
```

$x'_{n-1}y'_{n-1}s_{n-1}$
$(c_{n-1} \oplus c_n)$

overflow

```
   9   0  1001
+) 9   0  1001
  18   1  0010
```

$x_{n-1}y_n s'_{n-1}$
$(c_{n-1} \oplus c_n)$

2 operands have the same sign and the result sign changes

$x_{n-1}y_{n-1}s'_{n-1} + x'_{n-1}y'_{n-1}s_{n-1} = c_{n-1} \oplus c_n$

덧셈 연산
1의 보수 사용

두수의 덧셈에 부호비트도 같이 연산한다.

- 최상의 비트를 넘는 오버플로우 비트 발생시, 그들의 결과 값에 케리된 수를 더한다.

Example

```
      6   0  0110
+)   -9   1  0110
-----------------
     -3   1  1100
```

end-around carry

```
     -6   1  1001
+)    9   0  1001
-----------------
  (1) 0(1)0010
          ↘
+)            1
-----------------
      3   0  0011
```

not overflow $(c_{n-1} \oplus c_n) = 0$

```
     -9   1  0110
+)   -9   1  0110
-----------------
   (1)0   1100
          ↘
+)            1
-----------------
      0   1101
```

```
      9   0  1001
+)    9   0  1001
-----------------
      1  (1)0010
```

overflow

$(c_{n-1} \oplus c_n)$

COMPARISON OF REPRESENTATIONS

❖ 음수 변환 수월함

S + M > 1's Complement > 2's Complement

❖ **Hardware**

- S+M: Needs an adder and a subtractor for Addition
- 1's and 2's Complement: Need only an adder

❖ 연산 속도

2's Complement > 1's Complement(end-around C)

❖ **Zero**의 인식

2's Complement is fast

뺄 셈 연 산

❖ **2**의 보수를 이용한 뺄셈 연산

뺄셈을 보완한다**. (Sign bit**포함**)**
피감수 **sign bits**를 포함하여 덧셈 연산을 한다**.**

$$(\pm A) - (-B) = (\pm A) + B$$
$$(\pm A) - B = (\pm A) + (-B)$$

유동소수점 표현

- The location of the fractional point is not fixed to a certain location
- The range of the representable numbers is wide

$$F = EM$$

m_n	$e_k e_{k-1} \ldots e_0$	$m_{n-1} m_{n-2} \ \ldots \ m_0 . m_{-1} \ \ldots \ m_{-m}$
부호	지수	가수

가수

점에 의해 표시된다. (정수 혹은 분수에 의해)

지수

기수에 의해서 위치를 가리키다.

십진 표현

$$V(F) = V(M) * R^{V(E)}$$

M: Mantissa
E: Exponent
R: Radix

유 동 소 수 점

Example

sign		sign	
0	.1234567	0	04
가수		지수	

==> +.1234567 x 10^{+04}

Note:

유동 소숫점 표현에서는, 오직 가수**(Mantissa(M))**와 지수**Exponent(E)** 중심으로 표현된다**.. (** 기수와 소숫점은 생략 된다**.)**

Example

2진수 **+1001.11**는 **16-bit** 유동 소숫점으로 밑의 표현으로 나타내어 진다.
(6-bit exponent and 10-bit fractional mantissa)

Sign	Exponent	Mantissa
0	0 00100	100111000
or		
0	0 00101	010011100

유동 소수점의 특성 표현

❖ 표준

- 같은 수일지라도 유종 소숫점의 표현 은 배우 다르다.
  → **Need for a unified representation in a given computer**

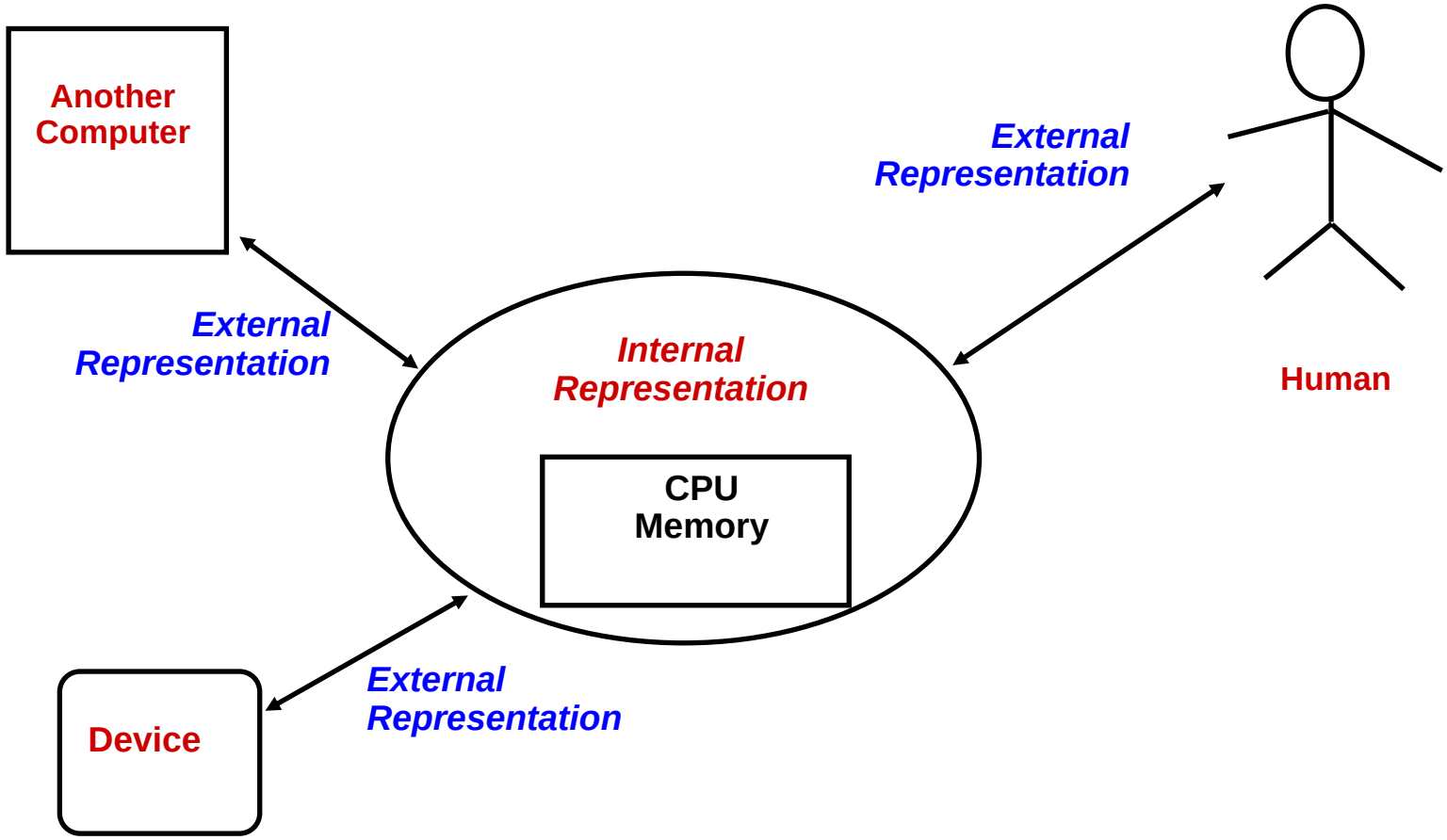
- 가수의 **non-zero digit** 점의 자리 표현은 매우 중요하다.

❖ **Zero**의 표현

- **Zero**
  가수**(Mantissa) = 0**

- **Real Zero**
  Mantissa = 0
  표현
  = 최소화 된 수 표현은
  00 ... 0
  ← **hardware**에 의해 쉽게 확인 할수 있다.

INTERNAL REPRESENTATION AND EXTERNAL REPRESENTATION



EXTERNAL REPRESENTATION

- **Numbers**
  Most of numbers stored in the computer are eventually changed by some kinds of calculations
  → *Internal Representation* for calculation efficiency
  → Final results need to be converted to as *External Representation* for presentability

- **Alphabets, Symbols, and some Numbers**
  Elements of these information do not change in the course of processing
  → No needs for Internal Representation since they are not used for calculations
  → External Representation for processing and presentability

- **Example**
  Decimal Number: 4-bit Binary Code
  BCD(Binary Coded Decimal)

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

OTHER DECIMAL CODES

Decimal	BCD(8421)	2421	84-2-1	Excess-3
0	0000	0000	0000	0011
1	0001	0001	0111	0100
2	0010	0010	0110	0101
3	0011	0011	0101	0110
4	0100	0100	0100	0111
5	0101	1011	1011	1000
6	0110	1100	1010	1001
7	0111	1101	1001	1010
8	1000	1110	1000	1011
9	1001	1111	1111	1100

Note: 8,4,2,-2,1,-1 in this table is the weight associated with each bit position.

$d_3\ d_2\ d_1\ d_0$: symbol in the codes

BCD: $d_3 \times 8 + d_2 \times 4 + d_1 \times 2 + d_0 \times 1$
$\Rightarrow$ 8421 code.

2421: $d_3 \times 2 + d_2 \times 4 + d_1 \times 2 + d_0 \times 1$

84-2-1: $d_3 \times 8 + d_2 \times 4 + d_1 \times (-2) + d_0 \times (-1)$

Excess-3: BCD + 3

BCD: It is difficult to obtain the 9's complement.
However, it is easily obtained with the other codes listed above.
$\rightarrow$ Self-complementing codes

GRAY CODE

- **Characterized by having their representations of the binary integers differ in only one digit between consecutive integers**
- **Useful in some applications**

4-bit Gray codes

Decimal number	Gray g_3	g_2	g_1	g_0	Binary b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	1	0	0	1	0
3	0	0	1	0	0	0	1	1
4	0	1	1	0	0	1	0	0
5	0	1	1	1	0	1	0	1
6	0	1	0	1	0	1	1	0
7	0	1	0	0	0	1	1	1
8	1	1	0	0	1	0	0	0
9	1	1	0	1	1	0	0	1
10	1	1	1	1	1	0	1	0
11	1	1	1	0	1	0	1	1
12	1	0	1	0	1	1	0	0
13	1	0	1	1	1	1	0	1
14	1	0	0	1	1	1	1	0
15	1	0	0	0	1	1	1	1

그레이 코드 - 분석

Letting $g_n g_{n-1} \ldots g_1 g_0$ be the (n+1)-bit Gray code
for the binary number $b_n b_{n-1} \ldots b_1 b_0$

$$g_i = b_i \oplus b_{i+1} \quad , \quad 0 \le i \le n-1$$
$$g_n = b_n$$

and

$$b_{n-i} = g_n \oplus g_{n-1} \oplus \ldots \oplus g_{n-i}$$
$$b_n = g_n$$

Reflection of Gray codes

ε	0	0 0	0 00
	1	0 1	0 01
		1 1	0 11
		1 0	0 10
			1 10
			1 11
			1 01
			1 00
			0 100
			1 100
			1 101
			1 111
			1 010
			1 011
			1 001
			1 101
			1 000

Note:

그레이 코드는 연산 되어진 결과를 반영한다.
- 연산없이 결과 테이블을 예상하기 쉽지다.
- **for any n: reflect case n-1 about a mirror at its bottom and prefix 0 and 1 to top and bottom halves, respectively**

CHARACTER REPRESENTATION
ASCII

ASCII (American Standard Code for Information Interchange) Code

MSB (3 bits)

LSB (4 bits)	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	‘	P
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	“	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	‘	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	I	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	m	n	~
F	SI	US	/	?	O	n	o	DEL

CONTROL CHARACTER REPRESENTAION (ACSII)

NUL	Null	DC1	Device Control 1
SOH	Start of Heading (CC)	DC2	Device Control 2
STX	Start of Text (CC)	DC3	Device Control 3
ETX	End of Text (CC)	DC4	Device Control 4
EOT	End of Transmission (CC)	NAK	Negative Acknowledge (CC)
ENQ	Enquiry (CC)	SYN	Synchronous Idle (CC)
ACK	Acknowledge (CC)	ETB	End of Transmission Block (CC)
BEL	Bell	CAN	Cancel
BS	Backspace (FE)	EM	End of Medium
HT	Horizontal Tab. (FE)	SUB	Substitute
LF	Line Feed (FE)	ESC	Escape
VT	Vertical Tab. (FE)	FS	File Separator (IS)
FF	Form Feed (FE)	GS	Group Separator (IS)
CR	Carriage Return (FE)	RS	Record Separator (IS)
SO	Shift Out	US	Unit Separator (IS)
SI	Shift In	DEL	Delete
DLE	Data Link Escape (CC)		

(CC) Communication Control
(FE) Format Effector
(IS) Information Separator

에러 검출 코드

❖ Parity System

- 에러 검색을 위해 간소화 한다.
- **One parity** 비트를 이용하여 검사 한다.
- 짝수 검사기와 홀수 검사기가 있다.

- 짝수 검사기
  - **One bit is attached to the information so that the total number of 1 bits is an even number**

1011001 0
1010010 1

- 홀수 검사기
  - **One bit is attached to the information so that the total number of 1 bits is an odd number**

1011001 1
1010010 0

PARITY BIT GENERATION

- **Parity Bit Generation**

  For $b_6b_5 \ldots b_0$(7-bit information); even parity bit b_{even}

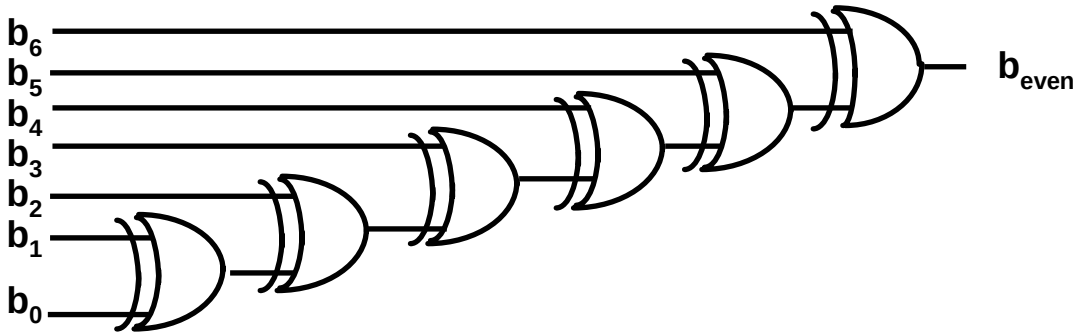
  $$b_{even} = b_6 \oplus b_5 \oplus \ldots \oplus b_0$$

  For odd parity bit

  $$b_{odd} = b_{even} \oplus 1 = \overline{b_{even}}$$

PARITY GENERATOR AND PARITY CHECKER

❖ **Parity Generator Circuit (even parity)**



❖ **Parity Checker**