Programming Design

Digital Systems

Ling-Chieh Kung

Department of Information Management
National Taiwan University

Thank you

- Most of the materials in this set of slides are adopted from the teaching materials of Professor Yuh-Jzer Joung's (莊裕澤).
  - Who failed the instructor in the course “Introduction to Computer Science”.

https://www.stpi.narl.org.tw/public/leader.htm

Road map

- **Number systems**
- Complements
- Digital circuits
- Miscellaneous things

Number systems

- decimal numbers: $7397 = 7\times10^3 + 3\times10^2 + 9\times10^1 + 7\times10^0$
- In general,

$$a_4a_3a_2a_1a_0 . a_{-1}a_{-2}$$
$$= a_4\times10^4 + a_3\times10^3 + a_2\times10^2 + a_1\times10^1 + a_0\times10^0 + a_{-1}\times10^{-1} + a_{-2}\times10^{-2}$$

  - a_i: **coefficient**
  - 10: **base** or **radix**
  - a_4: most significant bit (**msb**)
  - a_{-2}: least significant bit (**lsb**)

Base-r system

- In general, number X in a **base-r system** is represented as

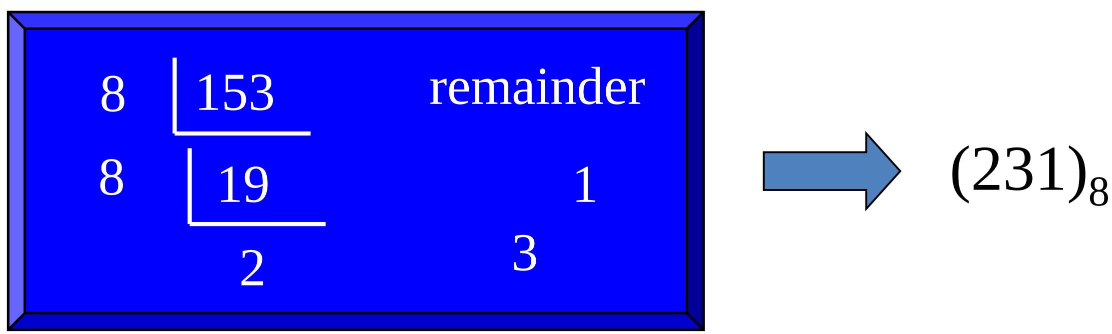
  $X = (a_n a_{n-1} \ldots\ldots a_1 a_0 . a_{-1} a_{-2} \ldots\ldots a_{-m})$

  which has the value

  $X = a_n r^n + a_{n-1} r^{n-1} + \ldots\ldots + a_1 r + a_0 + a_{-1} r^{-1} + a_{-2} r^{-2} + \ldots\ldots + a_{-m} r^{-m}$.
- In a **binary** system, $r = 2$, $a_i \in \{0, 1\}$.
- In an **octal** system, $r = 8$, $a_i \in \{0,1,\ldots,7\}$.
- In a **decimal** system, $r = 10$, $a_i \in \{0,1,\ldots,9\}$.
- In a **hexadecimal** system, $r = 16$, $a_i \in \{0,1,\ldots 9,A,B,C,D,E,F\}$.

Base conversion

- Base-*r* to base-10 conversion:
  Straightforward!
- Base-*r* to base-*s* conversion:
  By **repeated division for integers** and **repeated multiplication for fractions**.
- **Example.** Converting $(153.513)_{10}$ to an octal number.
  Integer part: $(153)_{10} = (231)_8$



Base conversion

- fractional part: 0.513

  $0.513 \times 8 = \mathbf{4}.104$

  $0.104 \times 8 = \mathbf{0}.832$

  $0.832 \times 8 = \mathbf{6}.656$

  $0.656 \times 8 = \mathbf{5}.24$

  …

  $(0.513)_{10} = (0.\mathbf{4065}\ldots)_8$

All together:

$(153.513)_{10} = (231.4065\ldots)_8$

Base conversion: integer part

- $X = a_n r^n + a_{n-1} r^{n-1} + \ldots\ldots + a_1 r + a_0 + a_{-1} r^{-1} + a_{-2} r^{-2} + \ldots\ldots + a_{-m} r^{-m}$.
- Consider the integer part: $XI = a_n r^n + a_{n-1} r^{n-1} + \ldots\ldots + a_1 r + a_0$.
- By dividing XI by r, we obtain

$$\frac{XI}{r} = a_n \cdot r^{n-1} + a_{n-1} \cdot r^{n-2} + a_{n-2} \cdot r^{n-3} + \ldots + a_2 \cdot r + a_1$$

  and the **remainder** is $\boldsymbol{a_0}$

- By dividing XI/r by r, we obtain

$$\frac{XI}{r^2} = a_n \cdot r^{n-2} + a_{n-1} \cdot r^{n-3} + \ldots + a_3 \cdot r + a_2$$

  and the **remainder** is $\boldsymbol{a_1}$

Base conversion: integer part

- By dividing XI/r^2 by r, we obtain

$$\frac{XI}{r^3} = a_n \cdot r^{n-3} + a_{n-1} \cdot r^{n-4} + \ldots + a_4 \cdot r + a_3$$

  and the **remainder** is $\boldsymbol{a_2}$.
- Continually in this fashion, we eventually obtain the coefficients $a_n\ a_{n-1}\ a_{n-2}\ \ldots\ a_1\ a_0$

Base conversion: fraction part

- Consider the faction part: $XF = a_{-1}r^{-1} + a_{-2}r^{-2} + \ldots\ldots + a_{-m}r^{-m}$.
- By multiplying XF by r, we obtain

$$XF \cdot r = \underbrace{a_{-1}}_{\text{integer in between } 0..r-1} + \underbrace{a_{-2} \cdot r^{-1} + a_{-3} \cdot r^{-2} + \ldots + a_{-m+1} \cdot r^{-m+2} + a_{-m} \cdot r^{-m+1}}_{<1}$$

Because the maximum value of this part is

$$(r-1) \cdot r^{-1} + (r-1) \cdot r^{-2} + \ldots + (r-1) \cdot r^{-m+2} + (r-1) \cdot r^{-m+1}$$
$$\leq (r-1) \cdot (r^{-1} + r^{-2} + r^{-3} + \ldots + r^{-m+1})$$
$$< (r-1) \cdot \frac{1}{(r-1)} = 1$$

Base conversion: fraction part

- By continually multiplying the fraction part of $XF \times r$, we obtain

$$XF_1 = \underbrace{a_{-2}}_{\substack{\text{integer in} \\ \text{between } 0..r-1}} + \underbrace{a_{-3} \cdot r^{-1} + \ldots + a_{-m+1} \cdot r^{-m+3} + a_{-m} \cdot r^{-m+2}}_{<1}$$

  So we obtain the second digit a_{-2}.
- Similarly, by continually multiplying the fraction part of XF_1, we can obtain the third digit a_{-3}, then a_{-4}, then a_{-5}, and so on.

Base 2^i to base 2^j

- Conversion between base 2^i to base 2^j can be done more quickly.
  - **Example.** Convert $(10111010011)_2$ to octal and hexadecimal

$$10111010011 \Rightarrow \underbrace{10}_{2}\ \underbrace{111}_{7}\ \underbrace{010}_{2}\ \underbrace{011}_{3} \Rightarrow (2\ 7\ 2\ 3)_8$$

$$10111010011 \Rightarrow \underbrace{101}_{5}\ \underbrace{1101}_{D}\ \underbrace{0011}_{3} \Rightarrow (5\ D\ 3)_{16}$$

Base 2^i to base 2^j

- **Example.** Convert $(12A7F)_{16}$ to binary and octal.

$$(12A7F)_{16} \Rightarrow (0001\ 0010\ 1010\ 0111\ 1111)_2$$

$$(00\ \underset{2}{\underline{010}}\ \underset{2}{\underline{010}}\ \underset{5}{\underline{101}}\ \underset{1}{\underline{001}}\ \underset{7}{\underline{111}}\ \underset{7}{\underline{111}})_2 \Rightarrow (2\ \ 2\ 5\ 1\ 7\ 7)_8$$

Road map

- Number systems
- **Complements**
- Digital circuits
- Miscellaneous things

Addition/subtraction for binary numbers

- In our mind, subtraction appears to take a different approach from addition.
- The difference will complicate the design of a logical circuit.

Addition

$$\begin{array}{r} 9 \\ +\ 13 \\ \hline 22 \end{array} \qquad \begin{array}{r} 1001 \\ +\ 1101 \\ \hline 10110 \end{array}$$

Subtraction

$$\begin{array}{r} 9 \\ -\ 13 \\ \hline -\ 4 \end{array} \qquad \begin{array}{r} 1001 \\ -\ 1101 \\ \hline ? \end{array}$$

- The problem can be solved if we can represent “negative” numbers (so that subtraction becomes addition to a negative number.)

Complement: for simplifying subtraction

- Two types of complements for each base-r system:
  - **(*r* –1)'s complement**
  - ***r*'s complement**
- The (r-1)'s complement of an n-digital number X: $(\boldsymbol{r^n} - 1) - \boldsymbol{X}$
- **Example.** In decimal system, the 9's complement of 546700 is $(10^6 - 1) - 546700 = 999999 - 546700 = 453299$
- **Example.** The 9's complement of 012398 is $999999 - 012398 = 987601$
- **Example.** The 1's complement of 01011000 is

$$(2^8 - 1) - 01011000$$
$$= 11111111 - 01011000$$
$$= 10100111$$

- **Example.** The 1's complement of 0101101 is 1010010.

8-bit 1's complement numbers

0	0000 0000
1	0000 0001
2	0000 0010
...	
127	0111 1111
−127	1000 0000
...	
−2	1111 1101
−1	1111 1110
−0	1111 1111

all positive numbers begin with 0

all negative numbers begin with 1

r's complement

- The *r*'s complement of an *n*-digital number *X*: $\begin{cases} r^n - X & X \neq 0 \\ 0 & X = 0 \end{cases}$
- **Example.**
  - The 10's complement of 012398 is 987602.
  - The 10's complement of 2467000 is 7533000.
  - The 2's complement of 1101100 is 0010100.
  - The 2's complement of 0110111 is 1001001.
- To compute the complement of a number having radix point, first, remove the radix point, compute the complement of the new number, and restore the radix point.
  - The 1's complement of 01101.101 is 10010.010
  - The 2's complement of 01101.101 is 10010.011

8-bit 2’s complement numbers

0	0000 0000
1	0000 0001
2	0000 0010
…	
127	0111 1111
−128 ?	1000 0000
−127	1000 0001
…	
−2	1111 1110
−1	1111 1111
−0	0000 0000

all positive numbers begin with 0

all negative numbers begin with 1

Complement of the complement

- The complement of the complement of X is X.

$$(r-1)\text{'s complement of } (2^n - 1) - X$$
$$= (2^n - 1) - ((2^n - 1) - X)$$
$$= X$$

$$r\text{'s complement of } 2^n - X$$
$$= 2^n - (2^n - X)$$
$$= X \qquad \text{(if } X \neq 0\text{, and 0's 2's complement is 0)}$$

Representation of signed numbers

- Three possible representations of – 9 with 8 bits:
  - signed magnitude: **1**0001001
  - signed -1's-complement of +9 (00001001): 11110110
  - signed -2's-complement +9 (00001001): 11110111
- Numbers that can be represented in n bits:
  - signed magnitude: $(-2^{n-1}+1) \sim (2^{n-1}-1)$
  - signed-1's-complement: $(-2^{n-1}+1) \sim (2^{n-1}-1)$
  - signed-2's-complement: $(-2^{n-1}) \sim (2^{n-1}-1)$

Signed-magnitude numbers

$$\begin{array}{r} +\ 9 \\ +\ 13 \\ \hline +\ 22 \end{array} \rightarrow \begin{array}{r} 00001001 \\ +\ 00001101 \\ \hline 00010110 \end{array}$$

$$\begin{array}{r} -\ 9 \\ +\ 13 \\ \hline +\ 4 \end{array} \rightarrow \begin{array}{r} 10001001 \\ +\ 00001101 \\ \hline \end{array} \rightarrow \begin{array}{r} 00001101 \\ -\ 00001001 \\ \hline 00000100 \end{array}$$

$$\begin{array}{r} +\ 9 \\ -\ 13 \\ \hline -\ 4 \end{array} \rightarrow \begin{array}{r} 00001001 \\ +\ 10001101 \\ \hline \end{array} \rightarrow \begin{array}{r} 10001101 \\ -\ 00001001 \\ \hline 10000100 \end{array}$$

Signed-magnitude numbers

$$\begin{array}{r} -\ \ 9 \\ -\ 13 \\ \hline -\ 22 \end{array} \quad \Rightarrow \quad \begin{array}{r} {\color{red}1}0001101 \\ +\ {\color{red}1}0001001 \\ \hline {\color{red}1}0010110 \end{array}$$

Summary:
to perform arithmetic operations on signed magnitude numbers, we need to compare the signs and the magnitudes of the two numbers, and then perform either addition or subtraction, much like what we were taught to do in primary school. So this do not simplify the problem.

Signed-1's-complement numbers

- Assume that $X, Y \geq 0$, and they have n bits (including sign)
  - **Case *X* + *Y*:** normal binary addition.

$$\begin{array}{r} +\ \ 9 \\ +\ 13 \\ \hline +\ 22 \end{array} \Rightarrow \begin{array}{r} 00001001 \\ +\ 00001101 \\ \hline 00010110 \end{array}$$

Signed-1's-complement numbers

- **Case $-X + Y$:**

(1's complement of X) + Y

$$= [(2^n - 1) - X] + Y$$

$$= (2^n - 1) - (X - Y)$$

Sub-case: $X - Y \geq 0$: there will be no carry.

Sub-case: $X - Y < 0$:

$$(2^n - 1) - (X - Y) = \underset{\text{carry bit}}{2^n} + \underbrace{(Y - X) - 1}_{\geq 0}$$

So there will be a carry. To obtain ($Y - X$), we discard the carry and add 1 to the result.

Signed-1's-complement numbers

$$
\begin{array}{r}
-\ \ 9 \\
+\ 13 \\
\hline
+\ \ 4
\end{array}
\qquad\Rightarrow\qquad
\begin{array}{r}
11110110 \\
+\ 00001101 \\
\hline
{\color{red}1}00000011 \\
+\qquad\quad 1 \\
\hline
00000100
\end{array}
$$

end-around carry

The 1's complement of 9 (00001001) is 11110110

Signed-1's-complement numbers

- **Case $(-X) + (-Y)$:**

(1's complement of X) + (1's complement of Y)

$$= [(2^n - 1) - X] + [(2^n - 1) - Y]$$

$$= (2^n - 1) + \underbrace{(2^n - 1) - (X + Y)}_{\text{1's complement of } (X + Y)}$$

carry bit

extra -1

So there will be a carry. If we discard the carry, then the result is the 1's complement of $(X + Y)$ minus 1. To obtain the correct result, we need to add 1 to the result of $[-1 + (2^n - 1) - (X + Y)]$.

Signed-1's-complement numbers

$$\begin{array}{r} -\ \ 9 \\ -13 \\ \hline -22 \end{array}$$

$$\begin{array}{r} 11110110 \\ +\ 11110010 \\ \hline 111101000 \\ +\qquad 1 \\ \hline 11101001 \end{array}$$

The 1's complement of 9 (00001001) is 11110110

The 1's complement of 13 (00001101) is 11110010

The 1's complement of 22 (00010110)

Signed-2's-complement numbers

- **Addition**: add the corresponding digits (including the sign digits) and ignore any carry out.

$$\begin{array}{r} +\ \ 9 \\ +\ 13 \\ \hline +\ 22 \end{array} \Rightarrow \begin{array}{r} 00001001 \\ +\ 00001101 \\ \hline 00010110 \end{array}$$

$$\begin{array}{r} -\ \ 9 \\ +\ 13 \\ \hline +\ \ 4 \end{array} \Rightarrow \begin{array}{r} 11110111 \\ +\ 00001101 \\ \hline 100000100 \end{array} \Rightarrow \begin{array}{r} 11110111 \\ +\ 00001101 \\ \hline 00000100 \end{array}$$

The 2's complement of 9 (00001001) is 11110111

discarded carryout

Signed-2's-complement numbers

$$\begin{array}{r} +\ 9 \\ -\ 13 \\ \hline -\ 4 \end{array} \Rightarrow \begin{array}{r} 00001001 \\ +\ 11110011 \\ \hline 11111100 \end{array}$$

the 2's complement of 00000100 (4)

The 2's complement of 13 (00001101) is 11110011

$$\begin{array}{r} -\ 9 \\ -\ 13 \\ \hline -\ 22 \end{array} \Rightarrow \begin{array}{r} 11110111 \\ +\ 11110011 \\ \hline 111101010 \end{array}$$

the 2's complement of 00010110 (22)

discarded carryout

The 2's complement of 9 (00001001) is 11110111

Signed-2's-complement numbers

- Discussion

  $(-X) + (Y) \quad \Rightarrow \quad (2^n - X) + Y$

  $\qquad\qquad\qquad = \ 2^n - (X - Y)$

  Case $X > Y$:

  there will be no carry out, and the result is the 2's complement of $(X - Y)$

  Case $X \leq Y$: $= 2^n + (Y - X)$

  there will be a carry out, and the result after discarding the carry out is $(Y - X)$

Signed-2's-complement numbers

- Case $-X - Y$:

  $(-X) + (-Y) \quad \Rightarrow \quad (2^n - X) + (2^n - Y)$

  $= 2^n + [2^n - (X + Y)]$

  – there will be a carry out, and the result after discarding the carry out is the 2's complement of $(X + Y)$ (that is, $-(X + Y)$ in our representation)

Subtraction is replaced by addition

- $X - Y = X + (-Y) = X + (2\text{'s complement of } Y)$

$$\begin{array}{r} 9 \\ -\ 13 \\ \hline -\ 4 \end{array} \Rightarrow \begin{array}{r} 00001001 \\ -\ 00001101 \\ \hline \end{array} \Rightarrow \begin{array}{r} 00001001 \\ +\ 11110011 \\ \hline 11111100 \end{array}$$

2's complement of −13

Comparison of the three systems

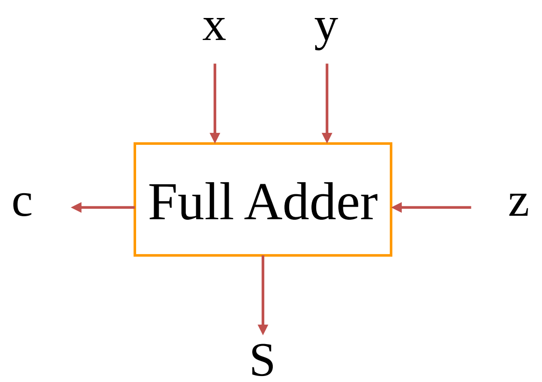
- signed-magnitude:
  - useful in ordinary arithmetic, awkward in computer arithmetic
- signed-1's-complement:
  - used in old computers, but is now seldom used.
- signed-2's-complement:
  - used in most computers.

Road map

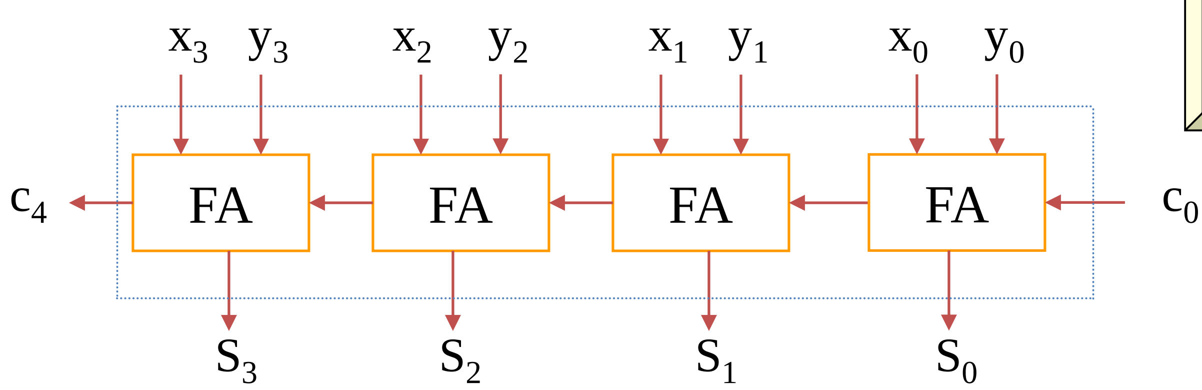
- Number systems
- Complements
- **Digital circuits**
- Miscellaneous things

Digital circuits

Full Adder



4-bits Adder

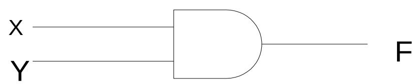


$$
\begin{array}{r}
c_0 \\
x_3x_2x_1x_0 \\
+\ \ y_3y_2y_1y_0 \\
\hline
c_4S_3S_2S_1S_0
\end{array}
$$

Basic circuit gates

Name	Gate	Algebraic Function	Truth Table

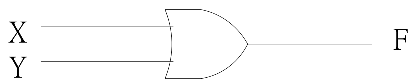
AND



$F = xy$

x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

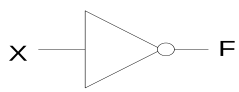
OR



$F = x+y$

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

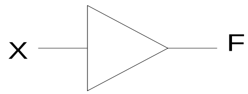
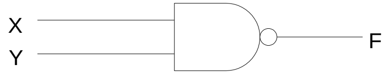
Inverter



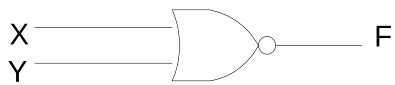
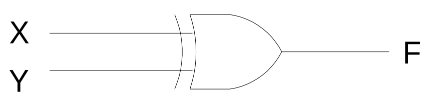
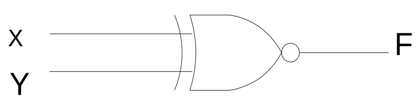
$F = x'$

x	F
0	1
1	0

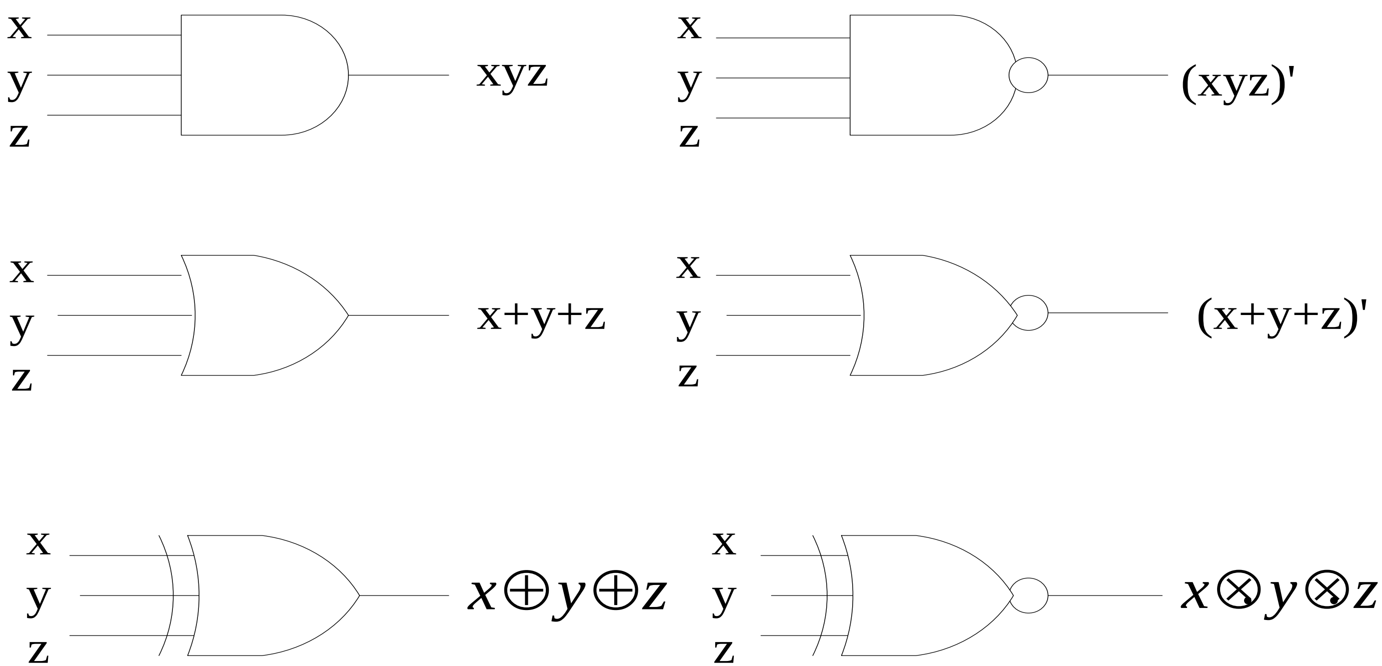
Basic circuit gates

<table>
<tr><th>Name</th><th>Gate</th><th>Algebraic Function</th><th>Truth Table</th></tr>
<tr><td>Buffer</td><td>
</td><td>$F = x$</td><td><table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table></td></tr>
<tr><td>NAND</td><td>
</td><td>$F = (xy)'$</td><td><table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td></tr>
</table>

Basic circuit gates

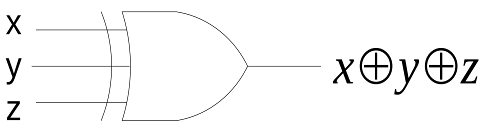
<table>
<tr><th>Name</th><th>Gate</th><th>Algebraic Function</th><th>Truth Table</th></tr>
<tr><td>NOR</td><td>
</td><td>$F = (x+y)'$</td><td><table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td></tr>
<tr><td>Exclusive-OR (XOR)</td><td>
</td><td>$F = xy'+x'y$
$= x \oplus y$</td><td><table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td></tr>
<tr><td>Exclusive-NOR (equivalence)</td><td>
</td><td>$F = xy+x'y'$
$= x \odot y$
or $x \otimes y$</td><td><table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td></tr>
</table>

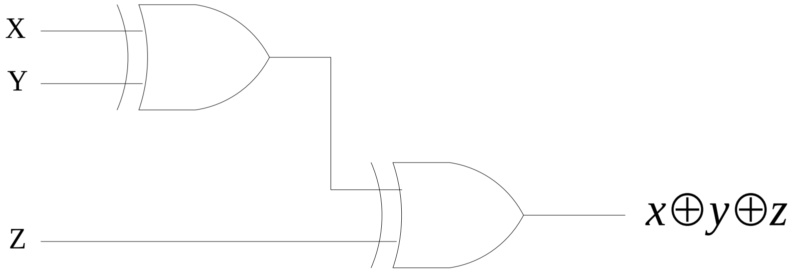
Extension to multiple inputs



Composition of gates

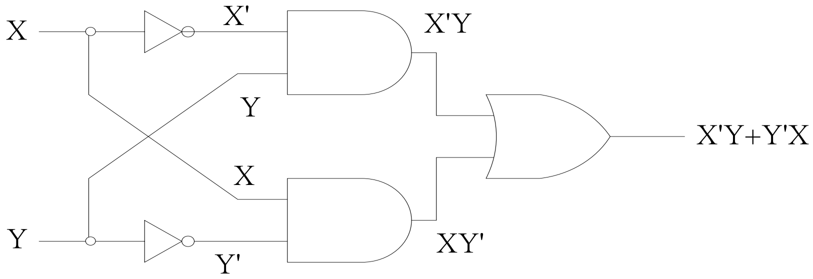
- In real implementation, the three-input Exclusive-OR is usually implemented by 2-input Exclusive-OR:





- Even a two input Exclusive-OR is usually constructed with other types of gates.

$x \oplus y = x'y + xy'$

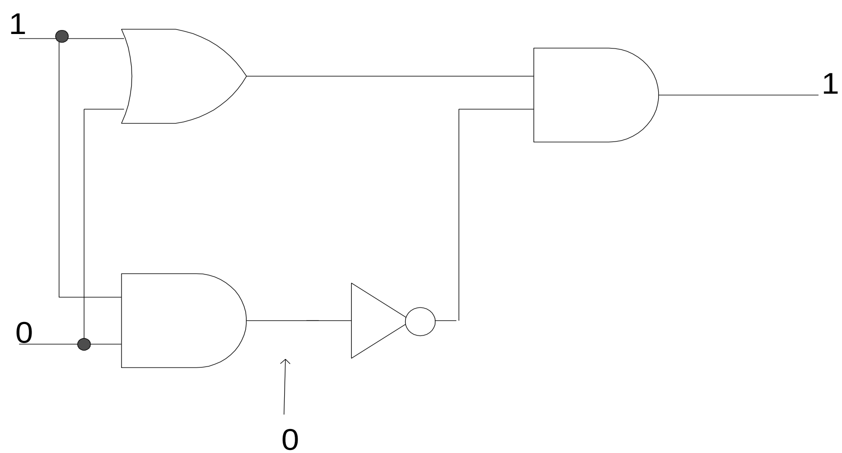


Gates can be used to compute

- This is how a gate adds a binary 1 and 0 in the "ones" place. If you feed a 1 and a 0 to the gate, it puts out a 1, which is the correct result of adding a binary 1 and 0.

Place 1

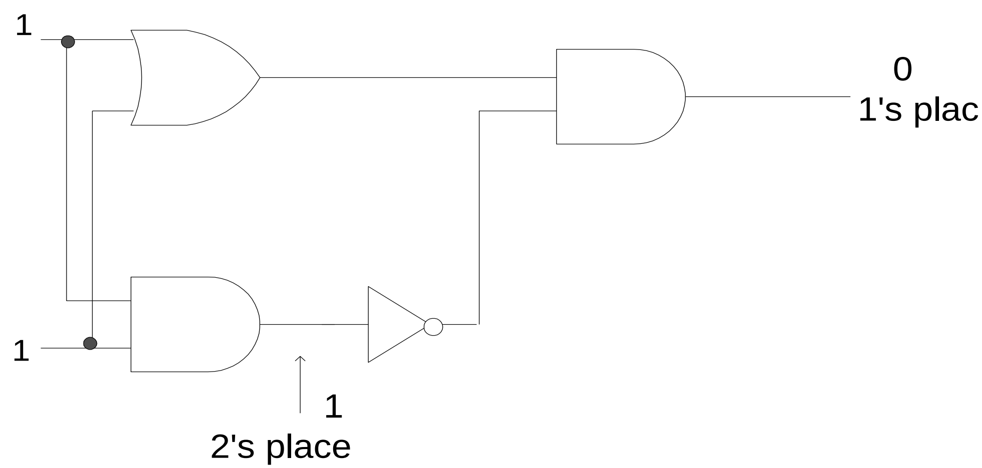
$$\begin{array}{r} 1 \\ +0 \\ \hline 1 \end{array}$$



Make it a bit like an adder

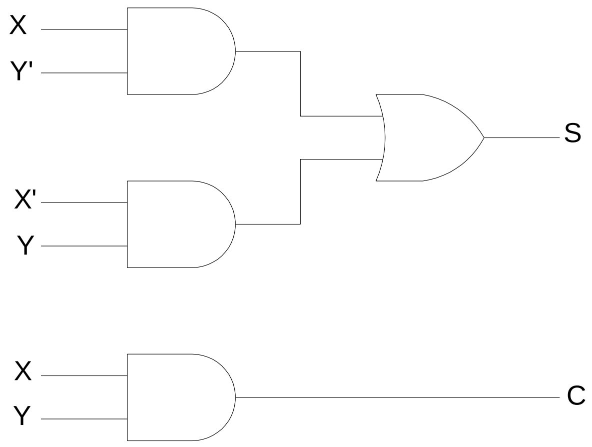
- Addition with a carryover is a little more difficult, for instance, adding 1 plus 1. If you feed the gate a 1 and 1, it will put a 0 in the “ones” place and put a carryover of 1 in the “twos” place. This produces the correct result for adding 1 and 1 in binary.

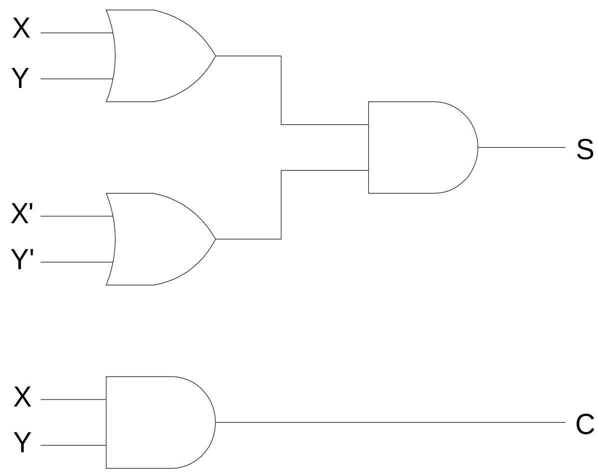
	Place 2	Place 1
	1	1
+		1
	1	0



- All together, it is only a “half” adder.

Various implementations of a half-adder

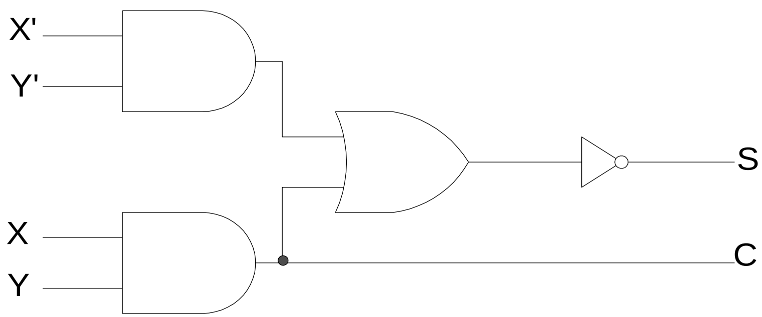




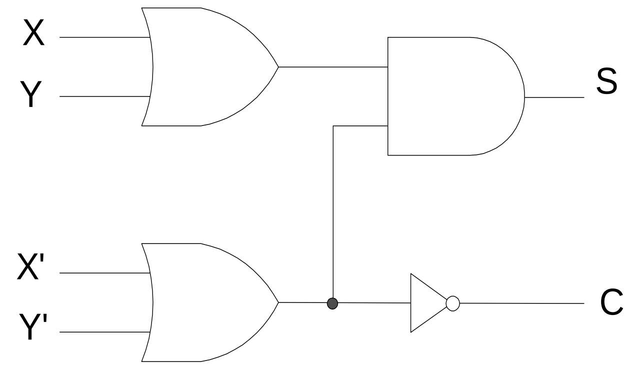
(a) S = xy’+x’y
C = xy

(b) S = (x+y)(x’+y’)
C = xy

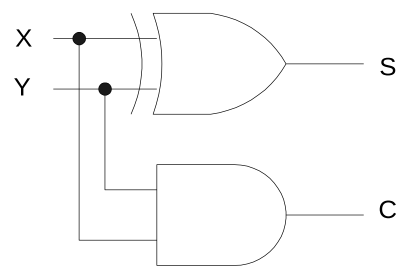
Various implementations of a half-adder



(c) S = (C + x’y’)’
C = xy

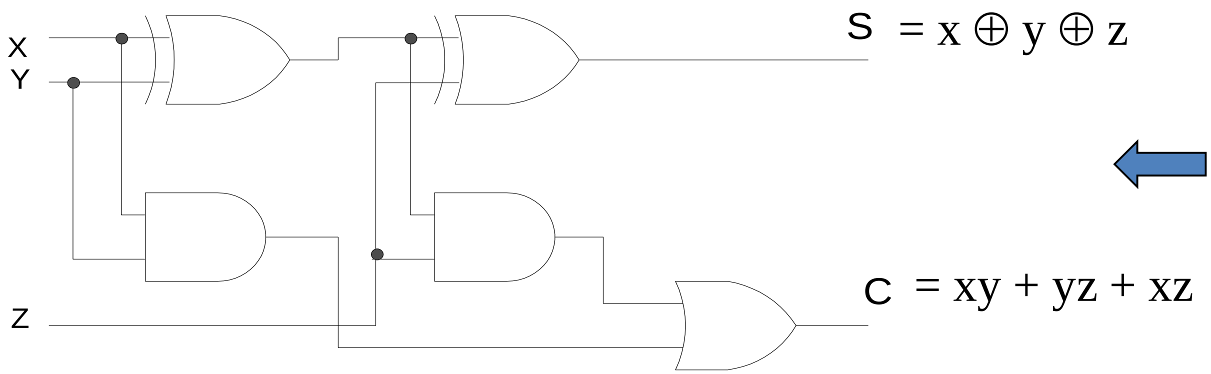


(d) S = (x+y)(x’+y’)
C = xy

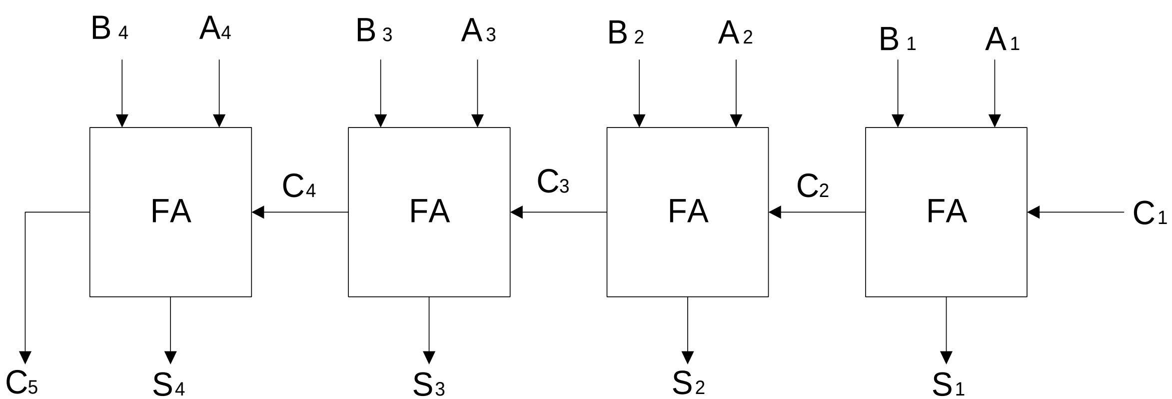


(e) S = x⊕y
C = xy

Binary adder and subtractor

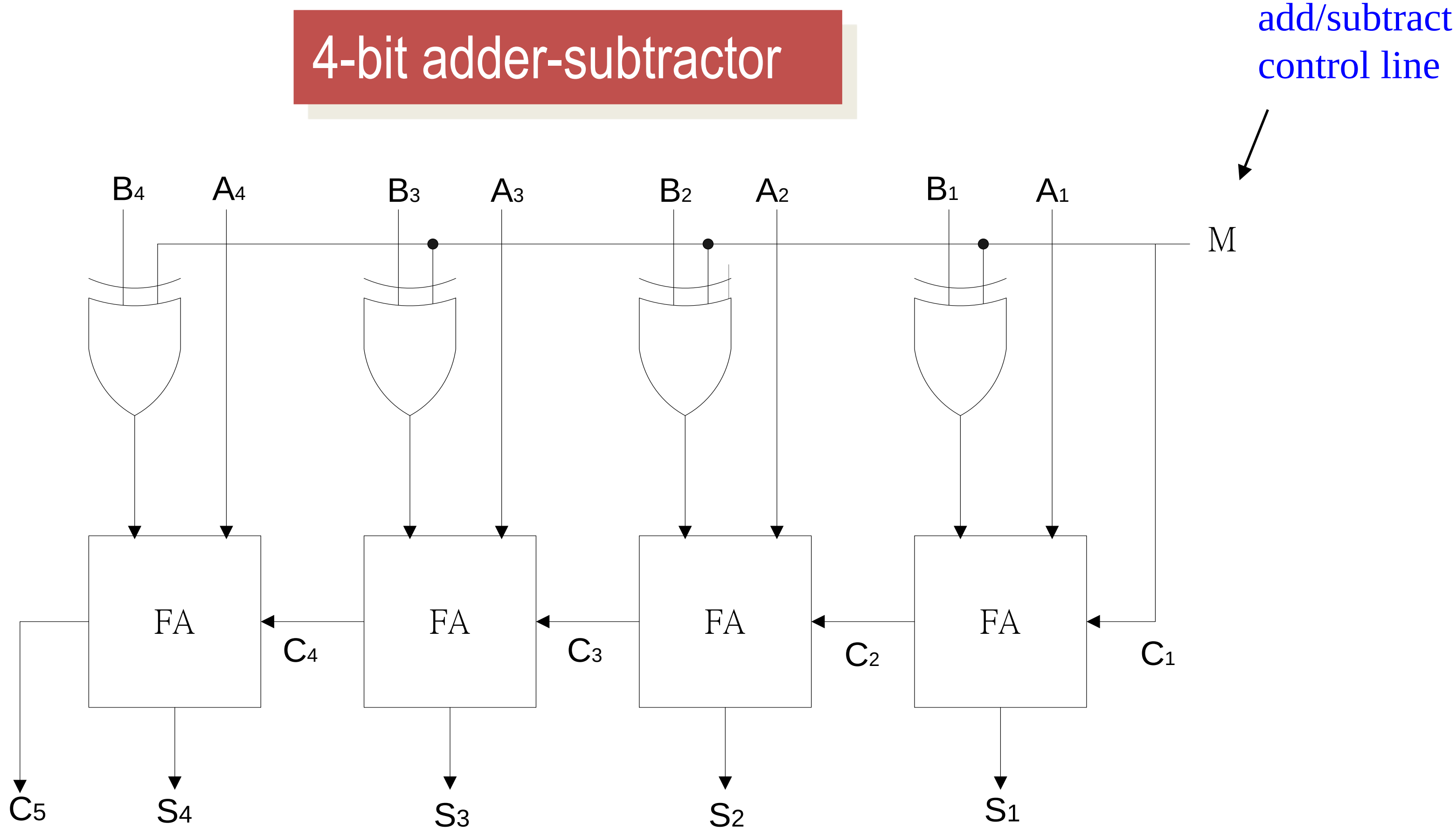


a full adder



4-bit parallel adder

4-bit adder-subtractor



Road map

- Number systems
- Complements
- Digital circuits
- **Miscellaneous things**

Overflow

$$\begin{array}{r} +\ 64 \\ +\ 96 \\ \hline +\ 160 \end{array} \Rightarrow \begin{array}{r} 01000000 \\ +\ 01100000 \\ \hline {\color{red}1}0100000 \end{array}$$

negative number, representing the 2's complement of 01100000 (96); that is, 64+96 = −96! Wrong!

$$\begin{array}{r} -\ 64 \\ -\ 96 \\ \hline -\ 160 \end{array} \Rightarrow \begin{array}{r} 11000000 \\ +\ 10100000 \\ \hline {\color{red}1}01100000 \end{array}$$

if we discard the carryout, then the result becomes a positive number 0110000 (48); that is, −64−96 = +48! Wrong!

Overflow

- The above problems (called **overflow**) are due to that using 8 bits, we can represent only –128 ~ +127! So the results of 64+96 or −64−96 cannot be represented in the 8-bit system.
- In general, when add two *n*-bits (including the sign bit) numbers X and Y, overflow occurs when:
  - $X, Y \geq 0$, $X+Y > 2^{n-1} - 1$
  - $X, Y < 0$, $X+Y < -2^{n-1}$
- Note that overflow cannot occur if one of X and Y is positive and the other is negative.

Overflow

- Overflow can be detected by examining the most significant bit of the result.

+ 64	01000000	(positive)
+ 96	+ 01100000	(positive)
+ 160	10100000	(negative)

indicate a negative number

− 64	11000000	(negative)
− 96	+ 10100000	(negative)
− 160	101100000	(positive)

indicate a positive number

Binary codes

- Binary codes can be established for any set of discrete elements.

MAN(1)
WOMAN (0)

WHITE(00)
RED(01) BLUE(10)
BLACK(11)

- Using n bits, we can represent at most 2^n distinct elements.
- So, to represent m distinct objects, we need at least $\lceil \log_2 m \rceil$ bits.
  – For example, we need $\lceil \log_2 10 \rceil = 4$ bits to represent $\{ 0,1,\ldots 9\}$.

Alphanumeric code

- ASCII (American Standard Code for Information Interchange)
  - originally use 7 bits to code 128 characters (32 are non-printing)
  - since most digital systems handle 8-bit (byte) more efficiently, an 8 bit version ASCII has also been developed.

```
 !"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmno
pqrstuvwxyz{|}~
```

95 printable ASCII characters, numbered 32 to 126.

Error-detecting code

- Binary code that can detect errors during data transmission.
- The most common way to achieve error-detection is by means of a **parity bit**.
- A parity bit is an extra bit included in a binary code to make the total number of 1's transmitted either odd **(odd parity)** or **(even parity)**.

Odd parity	
message	Parity bit
0010	0
0110	1
1110	0
1010	1

Even parity	
message	Parity bit
0010	1
0110	0
1110	1
1010	0

Application of parity bit