Binary Systems

Mantıksal Tasarım – BBM231

section instructor:

Ufuk Çelikcan

Binary Numbers 1/2

- Internally, information in digital systems is of binary form
  - groups of bits (i.e. binary numbers)
  - all the processing (arithmetic, logical, etc) are performed on binary numbers.
- Example: 4392
  - In decimal, 4392 = …
  - Convention: write only the coefficients.
  - $A = a_1 a_0 . a_{-1} a_{-2} a_{-3}$ where $a_j \in \{0, 1, \ldots, 9\}$
  - How do you calculate the value of A?

Binary Numbers 2/2

- Decimal system
  - coefficients are from {0,1, …, 9}
  - and coefficients are multiplied by powers of 10
  - base-10 (radix-10) number system
- Using the analogy, binary system {0,1}
  - base/radix 2
- Example: 25.625
  - 25.625 = decimal expansion
  - 25.625 = binary expansion
  - 25.625 =

Base-r Systems

- base-r (n, m)
  - $A = a_{n-1}\, r^{n-1} + \ldots + a_1\, r^1 + a_0\, r^0 + a_{-1}\, r^{-1} + a_{-2}\, r^{-2} + \ldots + a_{-m}\, r^{-m}$
- Octal
  - base-8 = base-2^3
  - digits {0,1, …, 7}
  - Example: $(31.5)_8$ = octal expansion =
- Hexadecimal
  - base-16
  - digits {0, 1, …, 9, A, B, C, D, E, F}
  - Example:
  - $(19.A)_{16}$ = hexadecimal expansion =

Powers of 2

- $2^{10} = 1{,}024$ (K) -
- $2^{20} = 1{,}048{,}576$ (M) -
- $2^{30} \rightarrow$ (G) -
- $2^{40} \rightarrow$ (T) -
- $2^{50} \rightarrow$ (P) -
- exa, zetta, yotta, ... -> exbi, zebi, yobi, ...
- <u>Examples</u>:
  - A **byte** is 8-bit, i.e. 1 B
  - 16 GB = ? B = 17,179,869,184 B

Arithmetic with Binary Numbers

	10101	21	augend
+	10011	19	addend
1	01000	40	sum

	10101	21	minuend
-	10011	19	subtrahend
0	00010	2	difference

			0	0	1	0	multiplicand (2)
		×	1	0	1	1	multiplier (11)
			0	0	1	0	
		0	0	1	0		
	0	0	0	0			
+ 0	0	1	0				
0	0	1	0	1	1	0	product (22)

Multiplication with Octal Numbers

				3	4	5	229	multiplicand
			×	6	2	1	401	multiplier
				3	4	5		
			7	1	2			
+	2	5	3	6				
	2	6	3	2	6	5	91829	product

Base Conversions

- From base-r >> decimal is easy
  - expand the number in power series and add all the terms
- Decimal >> base-r requires division
- <u>Simple idea</u>:
  - divide the decimal number successively by r
  - accumulate the remainders
- If there is a fraction, then integer part and fraction part are handled separately.

Base Conversion Examples 1/3

- Example 1:
  - 55
  - (decimal to binary)

55	1	1
27	1	2
13	1	4
6	0	
3	1	16
1=	1	32

– Example 2:
  - 144
  - (decimal to octal)

144	0	$0x8^0$
18	2	$2x8^1$
2=	2	$2x8^2$

Base Conversion Examples 2/3

- Example 1: 0.6875 (decimal to binary)
  - When dealing with fractions, instead of dividing by r multiply by r until we get an integer
  - $0.6875 \times 2 = 1.3750 = 1 + 0.375 \rightarrow a_{-1} = 1$
  - $0.3750 \times 2 = 0.7500 = 0 + 0.750 \rightarrow a_{-2} = 0$
  - $0.7500 \times 2 = 1.5000 = 1 + 0.500 \rightarrow a_{-3} = 1$
  - $0.5000 \times 2 = 1.0000 = 1 + 0.000 \rightarrow a_{-4} = 1$

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

Base Conversion Examples 2/3

- We are not always this lucky
- <u>Example 2</u>: (144.478) to octal
  - Treat the integer part and fraction part separately
  - $0.478 \times 8 = 3.824 = 3 + 0.824 \rightarrow a_{-1} = 3$
  - $0.824 \times 8 = 6.592 = 6 + 0.592 \rightarrow a_{-2} = 6$
  - $0.592 \times 8 = 4.736 = 4 + 0.736 \rightarrow a_{-3} = 4$
  - $0.736 \times 8 = 5.888 = 5 + 0.888 \rightarrow a_{-4} = 5$
  - $0.888 \times 8 = 7.104 = 7 + 0.104 \rightarrow a_{-5} = 7$
  - $0.104 \times 8 = 0.832 = 0 + 0.832 \rightarrow a_{-6} = 0$
  - $0.832 \times 8 = 6.656 = 6 + 0.656 \rightarrow a_{-7} = 6$
  - $144.478 = (220.3645706\ldots)_8$

Conversions between Binary, Octal and Hexadecimal

- r = 2 (binary), r = 8 (octal), r = 16 (hexadecimal)

10110001101001.101100010111

10 110 001 101 001.101 100 010 111

10 1100 0110 1001.1011 0001 0111

- Octal and hexadecimal representations are more compact.
- Therefore, we use them in order to communicate with computers directly using their internal representation

What if you can’t make exact groups?

For example:

10110001101001.10110001011101

10 110 001 101 001.101 100 010 111 010

10 1100 0110 1001.1011 0001 0111 0100

Complement

- Complementing is an operation on base-r numbers
- Goal: To simplify subtraction operation
  - Rather turn the subtraction operation into an addition operation
- Two types
  1) Radix complement (r's complement)
  2) Diminished complement ((r-1)'s complement)
- When r = 2
  1) 2's complement
  2) 1's complement

How to Complete?

- A number M in base-r (n-digit)
  - $r^n - M$ r's complement
  - $(r^n\text{-}1) - M$ (r-1)'s complement

  where n is the number of digits we use
- <u>Example</u>: Base r = 2, #Digits n = 4, Given M = 7
  - $r^n = 2^4 = 16$, $r^n - 1 = 15$.
  - 2's complement of 7 → 9
  - 1's complement of 7 → 8
- **Easier way** to compute 1's and 2's complements
  - Use binary expansions
  - 1's complement: negate
  - 2's complement: negate + increment

Better expressed this way

1’s complement: flip

2’s complement: flip + 1

How to Complement?

- 10’s complement of 9 is 0+1=1
- 10’s complement of 09 is 90+1=91
- 10’s complement of 009 is 990+1=991
- 9’s complement of 9 is 0
- 9’s complement of 09 is 90
- 9’s complement of 009 is 990
- 2’s complement of 100 is 011+1=100
- 2’s complement of 111 is 000+1=001
- 2’s complement of 000 is 000
- 1’s complement of 11110001 is 00001110

Subtraction with Complements

- Conventional subtraction
  - Borrow concept
  - If the minuend digit is smaller than the subtrahend digit, you borrow "1" from *a* digit in higher significant position

$$\begin{array}{rl} & \text{minuend} \\ - & \text{subtrahend} \\ \hline & \text{difference} \end{array}$$

- With complements
  - $M - N = ?$
  - $(r^n - N)$: r's complement of N
  - $M + (r^n - N) =$

Subtraction with Complements

$M + (r^n - N) = M - N + r^n$

1. if $M \geq N$,
   - the sum will produce a carry
   - >> that carry is discarded
2. Otherwise,
   - the sum will not produce a carry, and will be equal to $r^n - (N - M)$, which is the r's complement of N-M

Signed Binary Numbers

- When we use pencil-and-paper
  - We use symbols “+” and “-”
- Computers:
  We need to represent these symbols using bits
  - Convention:
    - 0 positive
      1 negative
    - **The leftmost bit position is used as a sign bit**
  - In signed representation, the leftmost bit is the sign bit. Bits to the right of sign bit is the number
  - In unsigned representation, the leftmost bit is a part of the number (i.e. the most significant bit (MSB))

Signed Binary Numbers

- Example: 5-bit numbers
  - 01011 → (unsigned binary)
  - → (signed binary)
  - 11011 → (unsigned binary)
  - → (signed binary)

  >> This method is called **signed-magnitude** and is rarely used in digital systems **(if at all)**
- In computers, *a negative number is represented by the* ***complement*** *of its* ***absolute*** *value*.
- **Signed-complement** system
  - positive numbers have always "0" in the MSB position
  - negative numbers have always "1" in the MSB position

Signed Binary Numbers

- Example: 5-bit numbers
  - 01011 → (unsigned binary) Number is 11
  - → (signed binary) Number is +11
  - 11011 → (unsigned binary) Number is 27
  - → (signed binary) Number is -11

  >> This method is called **signed-magnitude** and is rarely used in digital systems **(if at all)**
- In computers, *a negative number is represented by the* ***complement*** *of its* ***absolute*** *value*.
- **Signed-complement** system
  - positive numbers have always “0” in the MSB position
  - negative numbers have always “1” in the MSB position

Signed-Complement System

- <u>Example</u>:
  - Decimal 11 = $(01011)_2$
  - How to represent –11 in 1's and 2's complements

1) 1's complement -11 =
2) 2's complement -11 =

  - If we use eight-bit precision:

11 =

  - 1's complement **-11 =**
  - 2's complement **-11 =**

Signed-Complement System

- Example:
  - Decimal 11 = $(01011)_2$
  - How to represent –11 in 1's and 2's complements

1) 1's complement -11 =
2) 2's complement -11 =

  - If we use eight-bit precision:

11 = 00001011

  - 1's complement **-11 = 11110100**
  - 2's complement **-11 = 11110101**

Signed Number Representation

Signed magnitude		One's complement		Two's complement	
000	+0	000	+0	000	0
001	+1	001	+1	001	+1
010	+2	010	+2	010	+2
011	+3	011	+3	011	+3
100	-0	111	-0	111	-1
101	-1	110	-1	110	-2
110	-2	101	-2	101	-3
111	-3	100	-3	100	-4

A common Question from Students

A question I get asked by students all the time is :

Given a binary number, how do I know if it is in 2’s complement or 1’s complement; is it already in 2’s complement or do I have put it in 2’s complement,   etc.?

A binary number BY ITSELF can represent ANYTHING (unsigned number, signed number, character code, etc.). You MUST HAVE additional information that tells you what the encoding of the bits mean.

Simpler Arithmetic Rules

- using 2's complement:
  - convert negatives to complement
  - just add all the bits
  - **throw away any carry** out of the MSB

- using 1's complement
  - convert negatives to complement
  - just add all the bits
  - **end-around carry:** if there is a carry out (of 1) from the MSB, then the result will be off by 1, so throw away the carry out and add 1 to the rest of the result.

Subtraction with 2's Complements

- Example:
  - X = 101 0100 (84) and Y = 100 0011 (67)
  - X-Y = ? and Y-X = ? in 8 bits

	X	1010100	84
2's complement of	Y	+	2's comp of 67

	Y	1000011	67
2's complement of	X	+	2's comp of 84

Subtraction with 2's Complements

- Example:
  - X = 101 0100 (84) and Y = 100 0011 (67)
  - X-Y = ? and Y-X = ? in 8 bits

	X	01010100	84
2's complement of	Y	+10111101	2's comp of 67
		100010001	17
	Y	01000011	67
2's complement of	X	+10101100	2's comp of 84
		11101111	-17

throw out the carry out

2's complement of X-Y

Subtraction with **1's Complements**

- Example: Previous example using 1's complement

X = 101 0100 (84) and Y = 100 0011 (67), X-Y = ? and Y-X = ? in 8 bits

	X	**1010100**	**84**
1's complement of	Y	+	**1s comp of 67**

	Y	**1000011**	**67**
1's complement of	X	+	**1s comp of 84**

Subtraction with **1's Complements**

- Example: Previous example using 1's complement

X = 101 0100 (84) and Y = 100 0011 (67), X-Y = ? and Y-X = ? in 8 bits

	X	01010100	84
1's complement of	Y	+ 10111100	1s comp of 67
		100010000	16
		+ 1	
		00010001	17

end-around carry: 1) throw the carry out from MSB 2) increase the rest by 1 to get correct X-Y

	Y	01000011	67
1's complement of	X	+ 10101011	1s comp of 84
		11101110	-17

1's complement of X-Y

Arithmetic with 2's complement

- Examples:

```
+11        00001011        -11        11110101
 +9     +  00001001         +9     +  00001001
        -----------               -----------


+11        00001011        -11        11110101
 -9     +  11110111         -9     +  11110111
        -----------               -----------
```

- No special treatment for sign bits

Arithmetic with 2's complement

- <u>Examples</u>:

```
+11       00001011
 +9     + 00001001
        ----------
          00010100
```

```
-11       11110101
 +9     + 00001001
        ----------
          11111110
```

No carry, leftmost bit is 0, result is what you want

```
+11       00001011
 -9     + 11110111
        ----------
         100000010
```

```
                 1
 -9     + 11110111
        ----------
         111101100
```

- No special treatment for sign bits

Arithmetic with 2's complement

- Examples:

No carry, leftmost bit is 1, result is negative, take 2s complement, get -2

```
+                              -11        11110101
                                +9     +  00001001
   ________________                   ________________
       00010100                           11111110


+11        00001011            -11        11110101
 -9     +  11110111             -9     +  11110111
   ________________                   ________________
          100000010                      111101100
```

- No special treatment for sign bits

Arithmetic with 2's complement

- Examples:

```
+11      00001011        -11      11110101
 +9    + 00001001         +9    + 00001001
                                  --------
                                  11111110

+11      00001011        -11      11110101
 -9    + 11110111         -9    + 11110111
         --------                 --------
        100000010                111101100
```

Carry=1, leftmost bit is 0, result is what you want

- No special treatment for sign bits

36

Arithmetic with 2's complement

- Examples:

```
+11        00001011          -11        11110101
 +9     +  00001001           +9     +  00001001
        -----------                  -----------
           00010100                     11111110
```

Carry=1, leftmost bit is 1, result is negative, take 2s complement, get -20

```
+11        0000[illegible]   -11        11110101
 -9     +  11110111           -9     +  11110111
        -----------                  -----------
          100000010                    111101100
```

- No special treatment for sign bits

Arithmetic Overflow

- In hardware, we have limited resources to accommodate numbers
  - Computers use 8-bit, 16-bit, 32-bit, and 64-bit registers for the operands in arithmetic operations.
  - Sometimes the result of an arithmetic operation get too large to fit in a register.

Detecting Overflow

- The rules for detecting overflow in a 2's complement sum are simple:
  - If the sum of two positive numbers yields a negative result, the sum has overflowed.
  - If the sum of two negative numbers yields a positive result, the sum has overflowed.
  - Otherwise, the sum has not overflowed.

Arithmetic Overflow (2's complement)

- Example:

```
+2      0010
+4  +   0100
    --------
        0110

+7      0111
+6  +   0110
    --------
        1101
```

```
-3      1101
-5  +   1011
    --------
       10000

-3      1101
-6  +   1010
    --------
       10111
```

Subtraction with Signed Numbers

- Rule: is the same
- We take the 2's complement of the subtrahend
  - It does not matter if the subtrahend is a negative number.
  - $(\pm A) - (-B) = \pm A + B$

```
 -6     11111010
-13 -   11110011
      ----------
```

- Signed-complement numbers are added and subtracted in the same way as unsigned numbers
- With the same circuit, we can do both signed and unsigned arithmetic

Subtraction with Signed Numbers

- Rule: is the same
- We take the 2's complement of the subtrahend
  - It does not matter if the subtrahend is a negative number.
  - $(\pm A) - (-B) = \pm A + B$

```
 -6     11111010            -6     11111010
-13 -   11110011    ==>     13 +   00001101
    ------------               ------------
                                 1 00000111
```

- Signed-complement numbers are added and subtracted in the same way as unsigned numbers
- **With the same circuit, we can do both signed and unsigned arithmetic**

Examples with 2’s complement

-19 + (-7) = -26:

$$
\begin{array}{ccccccccc}
\boxed{1} & 1 & 1 & 1 & 1 & & & 1 & \\
 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
+ & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\
\hline
 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0
\end{array}
$$

Carryout without overflow. Sum is correct.

Examples with 2’s complement

-75 + 59 = -16:

		1	1	1	1	1	1	
	1	0	1	1	0	1	0	1
+	0	0	1	1	1	0	1	1
	1	1	1	1	0	0	0	0

No overflow nor carryout.

Examples with 2's complement

-103 + -69 = -172:

$$
\begin{array}{ccccccccc}
\boxed{1} & & 1 & 1 & 1 & & 1 & 1 & \\
 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
+ & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
\hline
 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0
\end{array}
$$

Overflow, with incidental carryout. Sum is not correct.

Examples

104 + 45 = 149:

	1	1		1				
	0	1	1	0	1	0	0	0
+	0	0	1	0	1	1	0	1
	1	0	0	1	0	1	0	1

Overflow, no carryout. Sum is not correct.

Examples with 2’s complement

-39 + 92 = 53:

$$
\begin{array}{ccccccccc}
\boxed{1} & 1 & & & 1 & 1 & & & \\
 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\
+ & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
\hline
 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1
\end{array}
$$

Carryout without overflow. Sum is correct.

Examples

10 + -3 = 7:

$$
\begin{array}{ccccccccc}
\boxed{1} & 1 & 1 & 1 & 1 & & & & \\
 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
+ & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
\hline
 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{array}
$$

Carryout without overflow. Sum is correct.

Examples

127 + 1 = 128:

$$
\begin{array}{ccccccccc}
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \\
 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
+ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

Overflow, no carryout. Sum is not correct.

Examples

-1 + 1 = 0:

$\boxed{1}$	1	1	1	1	1	1	1	
	1	1	1	1	1	1	1	1
+	0	0	0	0	0	0	0	1
	0	0	0	0	0	0	0	0

Carryout without overflow. Sum is correct.

Alphanumeric Codes

- Besides numbers, we have to represent other types of information
  - letters of alphabet, mathematical symbols.
- For English, alphanumeric character set includes
  - 10 decimal digits
  - 26 letters of the English alphabet (both lowercase and uppercase)
  - several special characters
- We need an alphanumeric code
  - ASCII
  - American Standard Code for Information Interchange
  - Uses 7 bits to encode 128 characters

ASCII Code

- 7 bits of ASCII Code
  - $(b_6\, b_5\, b_4\, b_3\, b_2\, b_1\, b_0)_2$
- <u>Examples</u>:
  - A → **65 = (1000001),…,** Z → **90 = (1011010)**
  - a → **97 = (1100001),** …, z → **122 = (1111010)**
  - **0 →48 = (0110000), …,9 → 57 = (0111001)**
- 128 different characters
  - 26 + 26 + 10 = 62 (letters and decimal digits)
  - 32 special printable characters %, *, $
  - 34 special control characters (non-printable): BS, CR, etc.

Representing ASCII Code

- 7-bit
- Most computers manipulate 8-bit quantity as a single unit (byte)
  - One ASCII character is stored using one byte
  - One unused bit can be used for other purposes such as representing Greek alphabet, italic type font, etc.
- The eighth bit can be used for error-detection
  - parity of seven bits of ASCII code is prefixed as a bit to the ASCII code.
  - A → (**0 1000001**) even parity
  - A → (**1 1000001**) odd parity
  - Detects one, three, and any odd number of bit errors

Binary Logic

- Binary logic is equivalent to what it is called "two-valued Boolean algebra"
  - Or we can say: it is an implementation of Boolean algebra
- Deals with variables that take on "two discrete values" and operations that assume logical meaning
- Two discrete values:
  - {true, false}
  - {yes, no}
  - {1, 0}

Binary Variables and Operations

- We use A, B, C, x, y, z, etc. to denote binary variables
  - each can take on {0, 1}

- Logical operations
  1. AND → $x \cdot y = z$ or $xy = z$
  2. OR → $x + y = z$
  3. NOT → $\overline{x} = z$ or $x' = z$

  - For each combination of the values of x and y, there is a value specified by the definition of the logical operation.
  - This definition may be listed in a compact form called <u>truth table</u>.

Truth Table

x	y	AND $x \cdot y$	OR $x + y$	NOT x'
0	0			
0	1			
1	0			
1	1			

Logic Gates

- Binary values are represented as electrical signals
  - Voltage, current
- They take on either of two recognizable values
  - For instance, voltage-operated circuits
  - 0V → 0
  - 4V → 1
- Electronic circuits that operate on one or more input signals to produce output signals
  - AND gate, OR gate, NOT gate

Range of Electrical Signals

- What really matters is the range of the signal value

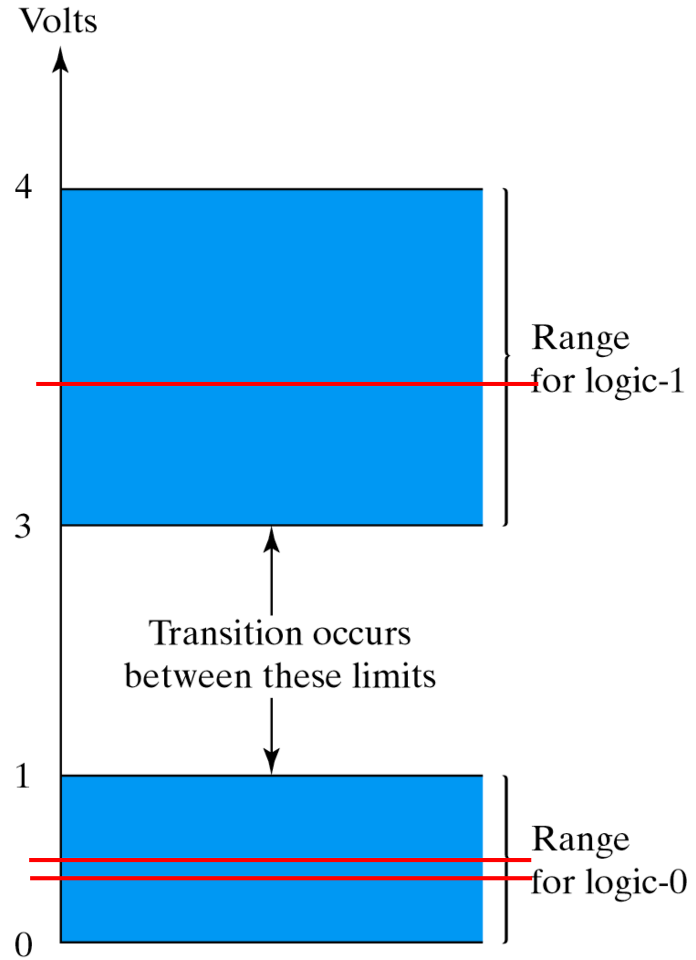
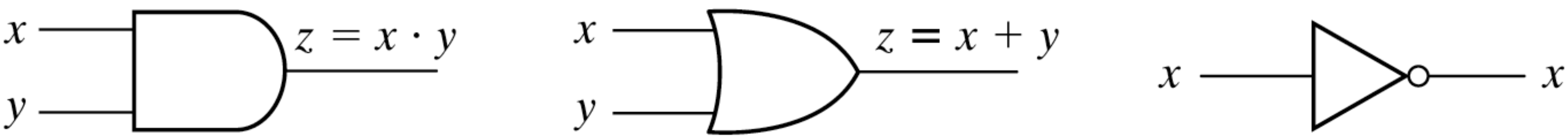


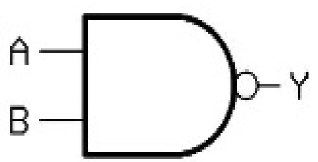
Fig. 1-3 Example of binary signals

Logic Gate Symbols

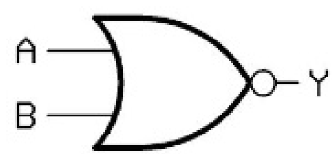


(a) Two-input AND gate (b) Two-input OR gate (c) NOT gate or inverter

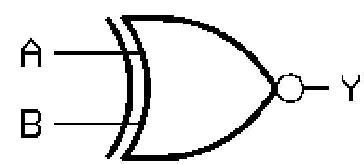
Fig. 1-4 Symbols for digital logic circuits



NAND gate

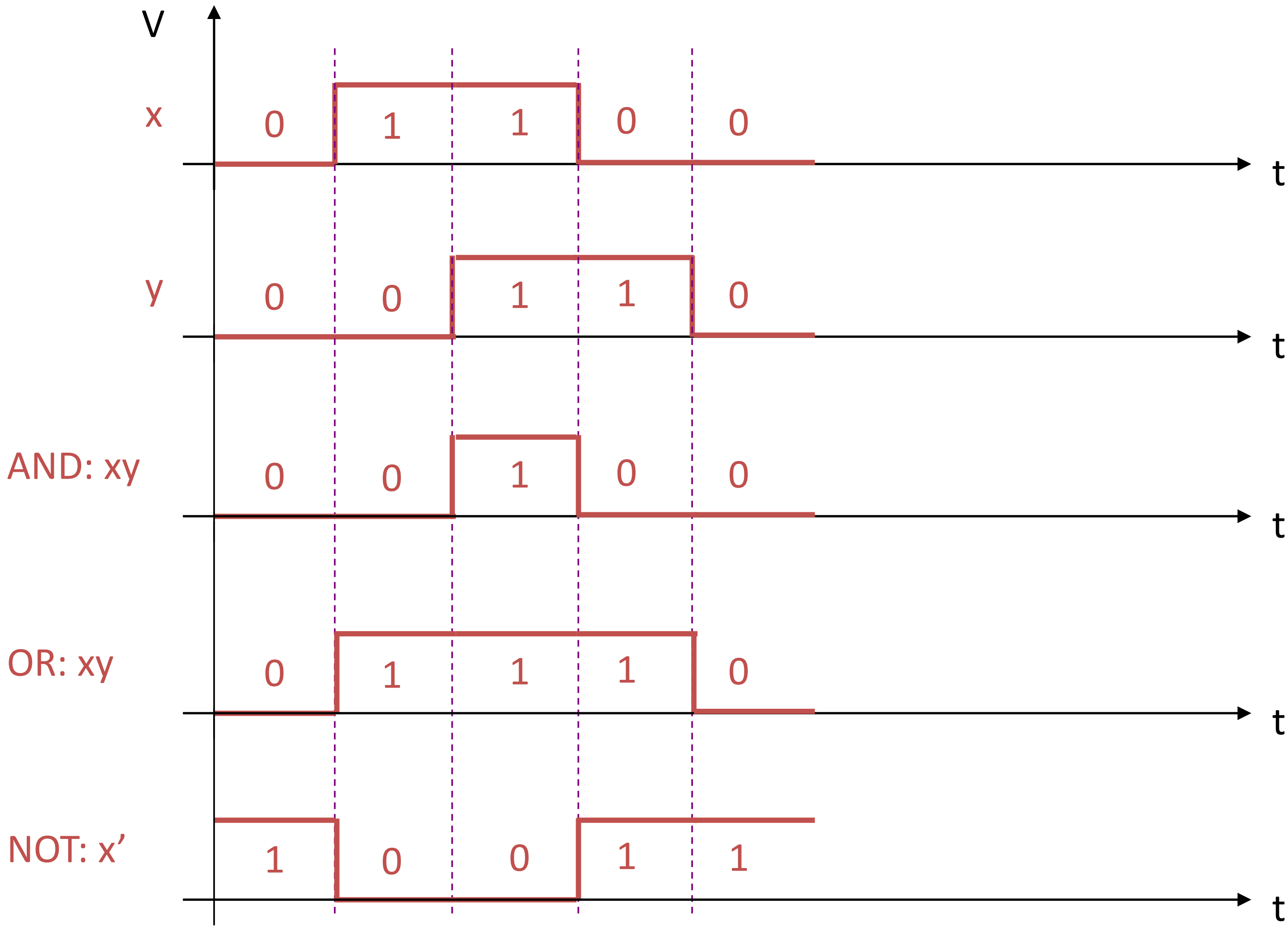


NOR gate



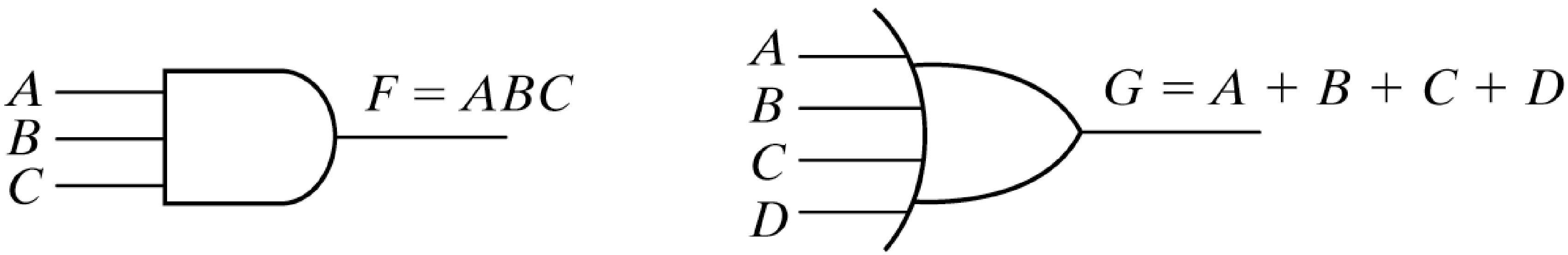
XNOR gate

Gates Operating on Signals



Input-Output Signals for gates

Gates with More Than Two Inputs



(a) Three-input AND gate

(b) Four-input OR gate

Fig. 1-6 Gates with multiple inputs