Binary Systems

Logic and Digital System Design - CS 303
Erkay Savaş
Sabancı University
Motivation

• Analysis & Design of digital electronic circuits
• Digital circuits are used in
  - digital computers,
  - data communication,
  - digital recording,
  - digital TV,
  - and many other applications require digital hardware
• Fundamental concepts in the design of digital systems
• Basic tools for the design of digital circuits
• Logic gates (AND, OR, NOT)
Digital System

• One characteristic:
• Ability of manipulating discrete elements of information
• A set that has a finite number of elements contains discrete information
• Examples for discrete sets
  - Decimal digits \{0, 1, ..., 9\}
  - Alphabet \{A, B, ..., Y, Z\}
  - Binary digits \{0, 1\}
• One important problem
  - how to represent the elements of discrete sets in physical systems?
How to Represent?

• In electronics circuits, we have electrical signals
  - voltage
  - current

• Different strengths of a physical signal can be used to represent elements of the discrete set.

• Which discrete set?
• Binary set is the easiest
  - two elements \{0, 1\}
  - Just two signal levels: 0 V and 4 V

• This is why we use binary system to represent the information in our digital system.
Binary System

- Binary set \( \{0, 1\} \)
  - The elements of binary set, 0 and 1 are called *binary digits*
  - or shortly *bits*.
- How to represent the elements of other discrete sets
  - Decimal digits \( \{0, 1, \ldots, 9\} \)
  - Alphabet \( \{A, B, \ldots, Y, Z\} \)
- Elements of any discrete sets can be represented using groups of bits.
  - \( 9 \rightarrow 1001 \)
  - \( A \rightarrow 1000001 \)
How Many Bits?

• What is the formulae for number of bits to represent a discrete set of \( n \) elements

• \( \{0, 1, 2, 3\} \)
  - \( 00 \rightarrow 0, 01 \rightarrow 1, 10 \rightarrow 2, \text{ ands } 11 \rightarrow 3 \).

• \( \{0, 1, 2, 4, 5, 6, 7\} \)
  - \( 000 \rightarrow 0, 001 \rightarrow 1, 010 \rightarrow 2, \text{ ands } 011 \rightarrow 3 \)
  - \( 100 \rightarrow 4, 101 \rightarrow 5, 110 \rightarrow 6, \text{ ands } 111 \rightarrow 7 \).

• The formulae, then,
  - \( \lceil \log_2 n \rceil \)
  - If \( n = 9 \), then \( \lceil \log_2 9 \rceil = 4 \).
Nature of Information

• Is information of discrete nature?
  • Sometimes, but not usually.
    – Anything related to money (e.g. financial computations, accounting etc) involves discrete information
  • In nature, information comes in a continuous form
    – temperature, humidity level, air pressure, etc.
  • *Continuous data must be converted (i.e. quantized) into discrete data*
    – lose of some of the information
    – We need ADC
General-Purpose Computers

- Best known example for digital systems
- **Components**
  - CPU,
  - I/O units
  - Memory unit
Why Digital Systems?

• Programmable
  - underlying hardware can be used for many different applications

• Reconfigurable hardware
  - Powerful paradigm
  - (C)PLD, PLA, PAL, FPGA

• Hardware Description Languages (HDL)
  - Facilitate the use of reconfigurable hardware in more efficient way.
  - Simulation and synthesis
  - VHDL, Verilog
Anatomy of Digital Systems

- A digital system is an interconnection of digital modules
- Hierarchical structure
- Each module implements a (logical) function
- This is the essence of this class
  - to understand the logical circuits and their logical function
  - Analyze and synthesize logical circuits that are components in a digital system
Binary Numbers - 1

- Internally, information in digital systems is of binary form
  - groups of bits (i.e. binary numbers)
  - Moreover, while the information is processed, all the processing (arithmetic, logical, etc) are performed on binary numbers.

- **Example: 4392**
  - In decimal, $4392 = (4 \times 10^3 + 3 \times 10^2 + 9 \times 10^1 + 2 \times 10^0)$
  - Convention: write only the coefficients.
  - $A = a_6 a_5 a_4 a_3 a_2 a_1 a_0 . a_{-1} a_{-2} a_{-3}$ where $a_j \in \{0, 1, \ldots, 9\}$
  - How do you calculate the value of $A$?
Binary Numbers - 2

• Decimal system
  - coefficients are from \{0,1, ..., 9\}
  - and coefficients are multiplied by powers of 10
  - base-10 or radix-10 number system

• Using the analogy, binary system \{0,1\}
  - base(radix)-2

• Example: 25.625
  - 25.625 = \(2 \times 10^1 + 5 \times 10^0 + 6 \times 10^{-1} + 2 \times 10^{-2} + 5 \times 10^{-3}\)
  - 25.625 = \(1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}\)
  - 25.625 = \((11001.101)_2\)
Base-r Systems

- **base-r** \((n, m)\)
  \[A = a_{n-1} r^{n-1} + ... + a_1 r^1 + a_0 r^0 + a_{-2} r^{-2} + ... + a_{-m} r^{-m}\]

- **Octal**
  - base-8
  - digits \{0,1, ..., 7\}
  - **Example**: \((31.5)_8 = 3 \times 8^1 + 1 \times 8^0 + 5 \times 8^{-1} = 25.625\)

- **Hexadecimal**
  - base-16
  - digits \{0,1, ..., 9, A, B, C, D, E, F\}
  - **Example**: \((19.A)_{16} = 1 \times 16^1 + 9 \times 16^0 + A \times 16^{-1} = 25.625\)
Powers of 2

- $2^{10} = 1,024$ (K - Kilo)
- $2^{20} = 1,048,576$ (M - Mega)
- $2^{30} \rightarrow G - Giga$
- $2^{40} \rightarrow T - Tera$

**Examples:**
- A byte is 8-bit
- 16 Gigabyte = $2^4 \times 2^{30} = 2^{34}$ bytes = 17,179,869,184
## Arithmetic with Binary Numbers

<table>
<thead>
<tr>
<th>Augend</th>
<th>Addend</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>21</td>
<td>10100</td>
</tr>
<tr>
<td>+ 10011</td>
<td>19</td>
<td>10100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Minuend</th>
<th>Subtrahend</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>21</td>
<td>10100</td>
</tr>
<tr>
<td>- 10011</td>
<td>19</td>
<td>00010</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multiplier (11)</th>
<th>Multiplier (2)</th>
<th>Product (22)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>10100</td>
<td>000110100</td>
</tr>
<tr>
<td>× 10100</td>
<td>000110100</td>
<td>000111000</td>
</tr>
<tr>
<td>+ 00000000</td>
<td></td>
<td>000111000</td>
</tr>
</tbody>
</table>
Multiplication with Octal Numbers

\[
\begin{array}{cccc}
3 & 4 & 5 & 229 \\
\times & 6 & 2 & 1 \\
\hline
3 & 4 & 5 \\
7 & 1 & 2 \\
\hline
\end{array}
\]

\[
\begin{array}{cccc}
2 & 5 & 3 & 6 \\
\hline
2 & 6 & 3 & 2 \\
6 & 5 & 91,829 \\
\end{array}
\]
Base Conversions

• From base-r to decimal is easy
  - expand the number in power series and add all the terms
• Reverse operation is somewhat more difficult
• Simple idea: divide the decimal number successively by r and accumulate the remainders.
• If there is a fraction, then integer part and fraction part are handled separately.
Base Conversion Examples - 1

- **Example 1**: 55 (decimal to binary)
  1. \(55/2 = 2 \times 27 + 1\) \(a_0 = 1\)
  2. \(27/2 = 2 \times 13 + 1\) \(a_1 = 1\)
  3. \(13/2 = 2 \times 6 + 1\) \(a_2 = 1\)
  4. \(6/2 = 2 \times 3 + 0\) \(a_3 = 0\)
  5. \(3/2 = 2 \times 1 + 1\) \(a_4 = 1\)
  6. \(1/2 = 2 \times 0 + 1\) \(a_5 = 1\)

- **Example 2**: 144 (decimal to octal)
  1. \(144/8 = 8 \times 18 + 0\) \(a_0 = 0\)
  2. \(18/8 = 8 \times 2 + 2\) \(a_1 = 2\)
  3. \(2/8 = 8 \times 0 + 2\) \(a_2 = 2\)
  - \(144 = (220)_8\)
Base Conversion Examples - 2

- **Example 1**: 0.6875 (decimal to binary)
  - When dealing with fractions, multiply by \( r \) until we get an integer instead of dividing by \( r \)
  1. \( 0.6875 \times 2 = 1.3750 = 1 + 0.3750 \) \( a_{-1} = 1 \)
  2. \( 0.3750 \times 2 = 0.7500 = 0 + 0.7500 \) \( a_{-2} = 0 \)
  3. \( 0.7500 \times 2 = 1.5000 = 1 + 0.5000 \) \( a_{-3} = 1 \)
  4. \( 0.5000 \times 2 = 1.0000 = 1 + 0.0000 \) \( a_{-4} = 1 \)
  - \( 0.6875 = (0.1011)_2 \)

- We are not always this lucky
- *Consider the example (124.478) to octal*
Base Conversion Examples - 3

• 124.478
  - Treat the integer part and fraction part separately
  - 124 = (174)₈
  - Fraction part:
    1. \(0.478 \times 8 = 3.824 = 3 + 0.824\) \(a_{-1} = 3\)
    2. \(0.824 \times 8 = 6.592 = 6 + 0.592\) \(a_{-2} = 6\)
    3. \(0.592 \times 8 = 4.736 = 4 + 0.736\) \(a_{-3} = 4\)
    4. \(0.736 \times 8 = 5.888 = 5 + 0.888\) \(a_{-4} = 5\)
    5. \(0.888 \times 8 = 3.824 = 7 + 0.104\) \(a_{-5} = 7\)
    6. \(0.104 \times 8 = 0.832 = 0 + 0.832\) \(a_{-6} = 0\)
    7. \(0.832 \times 8 = 6.656 = 6 + 0.656\) \(a_{-7} = 6\)
  - 124.478 = (174.3645706 ... )₈
Conversions between Binary, Octal and Hexadecimal

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

  10110001101011.111100000110

  010 110 001 101 011. 111 100 000 110 26153.7406

  0010 1100 0110 1011. 1111 0000 0110 2C6B.F06

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

- 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 (a.k.a. r’s complement)
  2. Diminished complement (a.k.a. (r-1)’s complement)
- When r = 2
  1. 2’s complement
  2. 1’s complement
How to Complement?

- A number N in base-r
  1. $r^n - N$  
     r’s complement
  2. $(r^n-1) - N$  
     (r-1)’s complement
  - where n is the number of bits we use
- Example: $r = 2$, $n = 4$, $N = 7$
  - $r^n = 2^4 = 16$, $r^n - 1 = 15$.
  - 2’s complement of 7 $\rightarrow$ 16 - 7 = 9
  - 1’s complement of 7 $\rightarrow$ 15 - 7 = 8
- Easier way to compute complements
  - $7 = (0111)_2 \rightarrow (1000)_2 + (0001)_2 = 8$  
    (2’s complement)
  - $7 = (0111)_2 \rightarrow (1000)_2 = 8$  
    (1’s complement)
Subtraction with Complements - 1

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

- **With complements**
  - \( M - N \)
  - \( r^n - N \) \( r \)'s complement of \( N \)
  - \( M + (r^n - N) = M - N + r^n \)
  1. if \( M \geq N \), the sum will produce a carry, that can be 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 \( N - M \)
Subtraction with Complements - 2

Example:
- \( X = 1010100 \) (84) and \( Y = 1000011 \) (67)
- \( X-Y = ? \) and \( Y-X = ? \)

<table>
<thead>
<tr>
<th>X</th>
<th>1010100</th>
</tr>
</thead>
<tbody>
<tr>
<td>2’s complement of Y</td>
<td>+ 0111101</td>
</tr>
<tr>
<td>the result X – Y</td>
<td>1 0010001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Y</th>
<th>1000011</th>
</tr>
</thead>
<tbody>
<tr>
<td>2’s complement of X</td>
<td>+ 0101100</td>
</tr>
<tr>
<td>the result Y – X</td>
<td>0 1101111</td>
</tr>
</tbody>
</table>

<p>| 0 1101111 |
| 0010000 |</p>
<table>
<thead>
<tr>
<th>+ 0000001</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010001</td>
</tr>
</tbody>
</table>
Subtraction with Complements - 3

- Example: Previous example using 1’s complement

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>X</strong></td>
<td>1010100</td>
<td><strong>Y</strong></td>
</tr>
<tr>
<td>1’s complement of</td>
<td>+ 0111100</td>
<td><strong>result</strong></td>
</tr>
<tr>
<td><strong>Y</strong></td>
<td></td>
<td><strong>X</strong></td>
</tr>
<tr>
<td>the result</td>
<td></td>
<td>discard carry</td>
</tr>
<tr>
<td><strong>X</strong></td>
<td>0010001</td>
<td><strong>Y</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000011</td>
</tr>
<tr>
<td>1’s complement of</td>
<td>+ 0101011</td>
<td><strong>result</strong></td>
</tr>
<tr>
<td><strong>Y</strong></td>
<td></td>
<td>01101110</td>
</tr>
<tr>
<td>the result</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Signed Binary Numbers

- Pencil-and-paper
  - Use symbols “+” and “-”

- We need to represent these symbols using bits
  - Convention:
    1. 0 positive
    1. 1 negative
  - The leftmost bit position is used as a sign bit
  - In **signed representation**, 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 → 11 (unsigned binary)
  - →+11 (signed binary)
  - 11011 → 27 (unsigned binary)
  - →-11 (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-Complement System

• Example:
  - 11 = (01011)
  - How to represent -11 in 1’s and 2’s complements:
    1. 1’s complement: -11 = 10100
    2. 2’s complement: -11 = 10100 + 00001 = 10101
  - If we use eight bit precision:
    - 11 = 00001011
    - 1’s complement: -11 = 11110100
    - 2’s complement: -11 = 11110101
## Signed Number Representation

<table>
<thead>
<tr>
<th>Signed magnitude</th>
<th>One’s complement</th>
<th>Two’s complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>000   +0</td>
<td>000  +0</td>
<td>000    0</td>
</tr>
<tr>
<td>001   +1</td>
<td>001  +1</td>
<td>001    +1</td>
</tr>
<tr>
<td>010   +2</td>
<td>010  +2</td>
<td>010    +2</td>
</tr>
<tr>
<td>011   +3</td>
<td>011  +3</td>
<td>011    +3</td>
</tr>
<tr>
<td>100   -0</td>
<td>111  -0</td>
<td>111    -1</td>
</tr>
<tr>
<td>101   -1</td>
<td>110  -1</td>
<td>110    -2</td>
</tr>
<tr>
<td>110   -2</td>
<td>101  -2</td>
<td>101    -3</td>
</tr>
<tr>
<td>111   -3</td>
<td>100  -3</td>
<td>100    -4</td>
</tr>
</tbody>
</table>

- **Issues**: balance, number of zeros, ease of operations
- **Which one is best? Why?**
Which One?

• Signed magnitude:
  - Where to put the sign bit?
  - Adders may need an additional step to set the sign
  - There are two representations for 0.

• Try to subtract a large number from a smaller one.
  \[ \begin{align*}
  2 & = 0010 \\
  5 & = 0101 \\
  5 - 2 &= 1101
  \end{align*} \]

• Two’s complement provide a natural way to represent signed numbers (every computer today uses two’s complement)

• Think that there is an infinite number of 1’s in a signed number
  \[-3 = 1101 \equiv 11...11101 \]

• What is 11111100?
Arithmetic Addition

• Examples:

\[
\begin{array}{ccc}
+11 & 00001011 & -11 & 11110101 \\
+9 & + & 00001001 & +9 & + & 00001001 \\
+20 & 00010100 & -2 & 11111110
\end{array}
\]

\[
\begin{array}{ccc}
+11 & 00001011 & -11 & 11110101 \\
-9 & + & 11110111 & -9 & + & 11110111 \\
+2 & 00000010 & -20 & 11101100
\end{array}
\]

• No special treatment for sign bits
Arithmetic Overflow - 1

- 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.

- **Examples:**

\[
\begin{array}{ccccccc}
+2 & 0010 & -3 & 1101 & +2 & 0010 \\
+4 & + & 0100 & -5 & + & 1011 & +6 & + & 0110 \\
+6 & 0110 & -8 & 1000 & +8 & 1 & 0000
\end{array}
\]
Arithmetic Overflow - 2

-3 1101
-6 + 1010
-9 1 0111

- Rule: If the MSB and the bits to the left of it differ, then there is an overflow
Subtraction with Negative 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.
  - \((\pm A) - (-B) = \pm A + B\)

\[
\begin{array}{c}
-6 & 11111010 \\
-13 & 11110011 \\
\hline
\end{array}
\quad
\begin{array}{c}
-6 & 11111010 \\
+13 & 00001101 \\
+7 & 00000111 \\
\hline
\end{array}
\]
- **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**
BCD Code - 1

- **Binary Coded Decimal - BCD**
  - Decimal number system is natural to human beings

<table>
<thead>
<tr>
<th>Decimal Symbol</th>
<th>BCD Digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
</tbody>
</table>
BCD Code - 2

• **Example:**
  - \(429 = (110101101)_2\)  \(\rightarrow 9\) bits
  - \(429 = (0100 0010 1001)_{BCD}\)  \(\rightarrow 12\) bits

• **Binary numbers from 1010 to 1111 have no meaning**
  - \(10 = (0001 0000)_{BCD} = (1010)_2\)
  - \(14 = (0001 0100)_{BCD} = (1110)_2\)

• **BCD Addition**

  \[
  \begin{array}{c}
  +4 & 0100 \\
  +5 & 0101 \\
  +9 & 1001 \\
  \end{array}
  \]

  \[
  \begin{array}{c}
  +4 & 0100 \\
  +8 & 1000 \\
  \end{array}
  \]

  wrong

  \[
  \begin{array}{c}
  +4 & 0100 \\
  +12 & 1100 \\
  +6 & 0110 \\
  +12 & 1 0010 \\
  \end{array}
  \]

  correction step
BCD Arithmetic

1. Why we add 6 to correct in BCD arithmetic?
   - Any digit in BCD larger than (1001) must produce a carry.
   - 4-bit binary numbers produce a carry when the result is larger than (1111)

\[
\begin{align*}
+9 & \quad 1001 \\
+9 + 1001 & \quad 1000 \\
+18 + 0010 & \quad \text{a natural carry} \\
+6 + 0110 & \quad \text{correction step} \\
+18 + 1000 & \quad 1000 \\
\end{align*}
\]
# More BCD Arithmetic

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0010</td>
<td>1001</td>
<td>0111</td>
</tr>
<tr>
<td>+</td>
<td>0001</td>
<td>1000</td>
<td>0011</td>
</tr>
<tr>
<td></td>
<td>0100</td>
<td>0010</td>
<td>1010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0110</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>0110</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0100</td>
<td>1000</td>
<td>0000</td>
</tr>
</tbody>
</table>


Signed-10’s Complement

• Same approach
  - MSD → 0 indicates positive numbers
  - MSD → 9 indicates negative numbers

• Example: How to represent -345 in BCD?
  - Subtract each digit from 9
  - Add 1 to the resulting number to get 10’s complement
  - 0345 → 9654 + 1 = 9655 = -345

• Why bother with signed-10’s complement arithmetic?
  - Some computers have special hardware to perform arithmetic in BCD directly
  - The reason being is to avoid conversion
**Signed-10’s Complement Arithmetic**

- **Example: 774-345**

  \[
  \begin{array}{c|c|c|c|c}
  & \text{1} & \text{1} & \text{1} & \text{1} \\
  \hline
  0000 & 0111 & 0111 & 0100 \\
  + 1001 & + 0110 & 0101 & 0101 \\
  \hline
  1010 & 1110 & 1100 & 1001 \\
  + 0110 & 0110 & + 0110 & \\
  \hline
  1 & 0000 & 0100 & 0010 & 1001 \\
  \end{array}
  \]
Other Decimal Codes

<table>
<thead>
<tr>
<th>Decimal digit</th>
<th>BCD 8421</th>
<th>2421</th>
<th>Excess-3</th>
<th>8 4 -2 -1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0000</td>
<td>0011</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>0001</td>
<td>0100</td>
<td>0111</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>0010</td>
<td>0101</td>
<td>0110</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>0011</td>
<td>0110</td>
<td>0101</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>0100</td>
<td>0111</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>1011</td>
<td>1000</td>
<td>1011</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>1100</td>
<td>1001</td>
<td>1010</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>1101</td>
<td>1010</td>
<td>1001</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>1110</td>
<td>1011</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>1111</td>
<td>1100</td>
<td>1111</td>
</tr>
<tr>
<td>Unused bit combinations</td>
<td>1010, 1011, 1100, 1101, 1110, 1111</td>
<td>0101, 0110, 0111, 1000, 1001, 1010</td>
<td>0000, 0001, 0010, 1101, 1110, 1111</td>
<td>0001, 0010, 0011, 1100, 1101, 1110</td>
</tr>
</tbody>
</table>
Other Decimal Codes

- Weighted Codes:
  - BCD 8421, 2 4 2 1, 8 4 -2 -1
  - 2 4 2 1: Weights are (2, 4, 2, 1)
    - 9 = 1111 = 1×2 + 1×4 + 1×2 + 1×1 = 9
    - 5 = (1011) = 1×2 + 1×2 + 1×1 = 5
    - how about (0101) ?
    - The advantage is self-complementing
      - 3 = 0011 → 1100 = 6 (9-3 = 6)
      - 5 = 1011 → 0100 = 4
  
- Excess-3 is not weighted
  - also self-complementing
Alphanumeric Codes

• Besides numbers, we have to represent other type of information such as 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 Exchange
  - Uses 7 bits to encode 128 characters
ASCII Code

- 7 bits of ASCII Code
  - \((b_6 \ b_5 \ b_4 \ b_3 \ b_2 \ b_1 \ b_0)_2\)

- **Examples:**
  - A \(\rightarrow\) 65 = (1000001), ...  Z \(\rightarrow\) 90 = (1011010)
  - a \(\rightarrow\) 97 = (1100001), ... z \(\rightarrow\) 122 = (1111010)
  - 0 \(\rightarrow\) 48 = (0110000), ... 9 \(\rightarrow\) 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 a 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.
  - \( A \rightarrow (01000001) \) even parity
  - \( A \rightarrow (11000001) \) odd parity
  - Detects one, three, and any odd number of bit errors
Binary Logic

- Deals with variables that takes on “two discrete values” and operations that assume logical meaning.
- Two discrete values:
  - {true, false}
  - {yes, no}
  - {1,0}
- Binary logic is actually equivalent to what it is called “Boolean algebra”:
  - Or we can say it is an implementation of Boolean algebra.
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 \( \rightarrow x \cdot y = z \) or \( xy = z \)
  2. OR \( \rightarrow x + y = z \)
  3. NOT \( \rightarrow \overline{x} = z \) or \( x' = z \)
  - For each combination of the values of x and y, there is a value of specified by the definition of the logical operation.
  - This definition may be listed in a compact form called truth table.
### Truth Table

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$x \cdot y$</th>
<th>$x + y$</th>
<th>$x'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Truth Table of logical operations:

- **AND**: $x \cdot y$
- **OR**: $x + y$
- **NOT**: $x'$
Logic Gates

- Electronic circuits that operate on one or more input signals to produce an output signals
  - AND gate, OR gate, NOT gate
- These signals are electrical signals
  - voltage
  - current
- They take on either of two recognizable values
- For instance, voltage-operated circuits
  - $0V \rightarrow 0$
  - $4V \rightarrow 1$
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
Gates Operating on Signals

\[ x \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \]

\[ y \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \]

**AND:** \( x \cdot y \)
\[ 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \]

**OR:** \( x + y \)
\[ 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \]

**NOT:** \( x' \)
\[ 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \]

Fig. 1-5 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