Computer Science for Engineers

Lecture 2
Basics

Prof. Dr. Dr.-Ing. Jivka Ovtcharova
Dipl. Wi.-Ing. Dan Gutu
31st of October 2008
1. Introduction

2. Basics
   - 2.1. Information representation and processing
     - 2.1.1. Alphabet - Data – Signals - Information
     - 2.1.2. Coding – number systems
     - 2.1.3. Boolean algebra and propositional logic
     - 2.1.4. Computer architecture
     - 2.1.5. Introduction to data structures
     - 2.1.6. Introduction to algorithms
   - 2.2. Programming paradigms
     - 2.2.1. Term programming
     - 2.2.2. Procedural programming
     - 2.2.3. Object-oriented programming

3. Object Orientation
In order to use a computer to process information, the data must be in a usable form.

A data representation is created through defining an alphabet such as:

**Definition 1.2:** An alphabet is a collection of valid symbols in a language.

**Example**

Alphabet $\alpha = \{a, b, c, d, e, f, g, h, i, m, n, p, r, s, t, u\}$

Valid words in the language using alphabet $\alpha$ would be:

„computer“, „science“, „not“, „is“, „fun“.
Definition 1.3: **A signal** is the representation of a message by the temporal modification of a physical value

**Signal parameter:** modifiable property of a signal  
**Example:** Frequency or amplitude of a wave  
**Signal transmission:** passing a message

**Persistent inscription:** durable representation of a message  
**Storage medium:** physical carrier of an inscription

Computer Science is based on signals, inscriptions and their processing
Definition 1.4: **Data** is a sequence of signals, linked by meaningful context

- We speak about data, when abstraction can be made of its usage
- A signal is the smallest accessible data element when a program is executed [Tami04].
- In a hierarchy, signals are beneath data
Information

Definition 1.5: Informations are interpreted data, when seen in the context of a problem and used to reach a goal

- Data becomes information, when it is seen in a certain context
- Information is strongly linked to its usage

[Info04]
**Definition 1.6:** Knowledge is the information how to interpret data

- **Knowledge by facts:** Knowledge of the information given by data
- **Procedural knowledge:** Knowledge on interpretation rules for the creation of information

**Examples:**
- The multiplication table is knowledge by facts
- Computation of the multiplication table is procedural knowledge

Engineers must deal with both kinds of knowledge
Information representation and processing

- Precondition for data and information processing:
  - Fragment the information into signs or signal chains, ex.
    - Fragment a text into words and letters
    - Fragment a picture into picture elements

- Representation and transmission of information units in different ways:
  - Speech: acoustic signals
  - Storage in a book: graphic symbols (letters)
  - Processing by a computer: electromagnetic signals
The translation of a data representation into another form can be performed through the use of a code.

**Definition 1.4:** A code is a defined method of transforming the characters of one alphabet into the characters of another.

**Example**

```
A → Coding definition
B → „Morsecode“
```

```
. -
- · ·
```
Coding Example: Voice-over-IP (VoIP) Telephony

- Different data representations must be able to be transformed between one another, so that exchange of data between different systems is possible.

Data transfer takes place most using TCP (Transmission Control Protocol) / IP (Internetwork Protocol) – Industry standard network technology.
• The elementary data entity of a digital computer is the **Bit**.

• **A bit can take one of 2 states**, defined by 0 and 1. Thus the alphabet of a digital computer is:

\[ \alpha = \{0, 1\} \]

**Example**

<table>
<thead>
<tr>
<th>Light switch</th>
<th>Punch card</th>
<th>Digital Logic</th>
<th>Morsecode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: current &gt; 0</td>
<td>hole</td>
<td>voltage &gt; 3V</td>
<td>• (&quot;short&quot;)</td>
</tr>
<tr>
<td>0: current = 0</td>
<td>no hole</td>
<td>voltage &lt; 2V</td>
<td>– (&quot;long&quot;)</td>
</tr>
</tbody>
</table>

• The 0 and 1 represent the basics of digital computer language (dual system/ binary number system)
Number System

- The general equation for the number system is:

\[ a = \sum_{i=0}^{n-1} z_i B^i \quad (n \geq 0, \text{n integer}) \]

with \( B \) as the base and \( z_i \) (\( 0 \leq z_i < B \)) as respective numbers.

- The most important number systems are:

<table>
<thead>
<tr>
<th>Number system</th>
<th>Base</th>
<th>Possible numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dual system</td>
<td>2</td>
<td>0,1</td>
</tr>
<tr>
<td>Octal system</td>
<td>8</td>
<td>0,1,2,3,4,5,6,7</td>
</tr>
<tr>
<td>Decimal system</td>
<td>10</td>
<td>0,1,2,3,4,5,6,7,8,9</td>
</tr>
<tr>
<td>Hexadecimal system</td>
<td>16</td>
<td>0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F</td>
</tr>
</tbody>
</table>

(The letters A-F stand for the values 10-15)

Source: Bronstein, Taschenbuch der Mathematik
Converting to other number systems (1)

- Convert „Binary representation → Decimal representation“
  - Calculation:
    \[
    (b_{n-1} \ldots b_1 b_0)_2 = \left( \sum_{i=0}^{n-1} b_i 2^i \right)_{10}
    \]
  - Example:
    \[
    (101)_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 4 + 1 = (5)_{10}
    \]
    \[
    (10110)_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 16 + 4 + 2 = (22)_{10}
    \]
Converting to other number systems (2)

- Convert „Decimal representation → Binary representation“:
  - Divide the decimal number by 2 and round up until you reach 0. At each step note the remainder (0,1). The binary representation is produced from the noted remainders (in reverse order)
  
- Example: \((22)_{10}\)

\[
\begin{align*}
22 : 2 & = 11 \quad \text{remainder} \ 0 \\
11 : 2 & = 5 \quad \text{remainder} \ 1 \\
5 : 2 & = 2 \quad \text{remainder} \ 1 \\
2 : 2 & = 1 \quad \text{remainder} \ 0 \\
1 : 2 & = 0 \quad \text{remainder} \ 1
\end{align*}
\]

\Rightarrow (22)_{10} = (10110)_{2}

Read in reverse order!
• Digital circuits are created „top-down“

• Several mathematic procedures are necessary for chip development:
  - Logic description
  - Minimization

• The basis for the **binary and logic description** of circuits gives a close coherence between boolean functions and propositional logic formulas

• With these description methods, circuits can be developed, minimized and analyzed
Boolean Algebra

- A computer built on the binary numbering system is able to perform logic operations with the help of a logic processing unit.
- Mathematics describes this using Boolean algebra.

**Definition 1.5:** Boolean Algebra consists of a carrier A, for which the following operations are defined:

\[ \land : A \times A \rightarrow A \quad \text{“and”} \]

\[ \lor : A \times A \rightarrow A \quad \text{“or”} \]

\[ \neg : A \rightarrow A \quad \text{Negation} \]

- All other complex operations performed by a computer are built upon these basic operations.
## Rules of Boolean Algebra

### Elementary Rules

<table>
<thead>
<tr>
<th>Rule Type</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Involution Rule</strong></td>
<td>$\neg (\neg f) = f$</td>
</tr>
<tr>
<td><strong>Commutative Rule</strong></td>
<td>$f \land g = g \land f$</td>
</tr>
<tr>
<td></td>
<td>$f \lor g = g \lor f$</td>
</tr>
<tr>
<td><strong>Associative Rule</strong></td>
<td>$(f \land g) \land h = h \land (g \land f)$</td>
</tr>
<tr>
<td></td>
<td>$(f \lor g) \lor h = h \lor (g \lor f)$</td>
</tr>
<tr>
<td><strong>Idempotent Rule</strong></td>
<td>$f \land f = f$</td>
</tr>
<tr>
<td></td>
<td>$f \lor f = f$</td>
</tr>
<tr>
<td><strong>Absorption Rule</strong></td>
<td>$f \land (f \lor g) = f$</td>
</tr>
<tr>
<td></td>
<td>$f \lor (f \land g) = f$</td>
</tr>
<tr>
<td><strong>Distribution Rule</strong></td>
<td>$f \land (g \lor h) = (f \land g) \lor (f \land h)$</td>
</tr>
<tr>
<td></td>
<td>$f \lor (g \land h) = (f \lor g) \land (f \lor h)$</td>
</tr>
<tr>
<td><strong>De Morgan's Law</strong></td>
<td>$\neg (f \land g) = \neg f \lor \neg g$</td>
</tr>
<tr>
<td></td>
<td>$\neg (f \lor g) = \neg f \land \neg g$</td>
</tr>
<tr>
<td><strong>Neutrality Rule</strong></td>
<td>$f \land (g \lor \neg g) = f$</td>
</tr>
<tr>
<td></td>
<td>$f \lor (g \land \neg g) = f$</td>
</tr>
</tbody>
</table>

### Created Rules

## Elementary Rules

<table>
<thead>
<tr>
<th>Rule Type</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Involution Rule</strong></td>
<td>$\neg (\neg f) = f$</td>
</tr>
<tr>
<td><strong>Commutative Rule</strong></td>
<td>$f \land g = g \land f$</td>
</tr>
<tr>
<td></td>
<td>$f \lor g = g \lor f$</td>
</tr>
<tr>
<td><strong>Associative Rule</strong></td>
<td>$(f \land g) \land h = h \land (g \land f)$</td>
</tr>
<tr>
<td></td>
<td>$(f \lor g) \lor h = h \lor (g \lor f)$</td>
</tr>
<tr>
<td><strong>Idempotent Rule</strong></td>
<td>$f \land f = f$</td>
</tr>
<tr>
<td></td>
<td>$f \lor f = f$</td>
</tr>
<tr>
<td><strong>Absorption Rule</strong></td>
<td>$f \land (f \lor g) = f$</td>
</tr>
<tr>
<td></td>
<td>$f \lor (f \land g) = f$</td>
</tr>
<tr>
<td><strong>Distribution Rule</strong></td>
<td>$f \land (g \lor h) = (f \land g) \lor (f \land h)$</td>
</tr>
<tr>
<td></td>
<td>$f \lor (g \land h) = (f \lor g) \land (f \lor h)$</td>
</tr>
<tr>
<td><strong>De Morgan's Law</strong></td>
<td>$\neg (f \land g) = \neg f \lor \neg g$</td>
</tr>
<tr>
<td></td>
<td>$\neg (f \lor g) = \neg f \land \neg g$</td>
</tr>
<tr>
<td><strong>Neutrality Rule</strong></td>
<td>$f \land (g \lor \neg g) = f$</td>
</tr>
<tr>
<td></td>
<td>$f \lor (g \land \neg g) = f$</td>
</tr>
</tbody>
</table>
Propositional Logic

- The **propositional logic** works with **elementary terms** that can have the values „true“ (T=1) or „false“ (F=0)

- Formulas of propositional logic:
  - Elementary terms are formulas
  - If \( M \) and \( N \) are formulas, than \( (M \land N) \) and \( (M \lor N) \) are formulas, too.
  - If \( M \) is a formula, than \( \neg M \) is a formula, too.
Propositional logic operations

- **And-operation**: \( a \land b \)
  - The term is **true**, if \( a \) and \( b \) are **true**

- **Or-operation**: \( a \lor b \)
  - The term is **true**, if **at least** \( a \) or \( b \) are **true**

- **XOR-operation** (exclusiv or):
  - The term is **true**, if **either** \( a \) or \( b \) are **true**.

<table>
<thead>
<tr>
<th>( a )</th>
<th>( b )</th>
<th>( a \land b )</th>
<th>( a \lor b )</th>
<th>( a \oplus b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
The correlation is realized by the **equalization** of the binary character set \{0,1\} with the set \{F,T\}.

Each Boolean function can be transformed into an equivalent logic expression.

- **Circuit synthesis**: building a circuit from a boolean function.

The contrariwise transformation is possible, too.

- **Circuit analysis**: an existing circuit shall be examined.

Example: Mechatronics
Computer architecture means:

- The **internal structure** of the computer, its components
- The **organisation of the activities** in the computer
Von Neumann (1903-1957) created the first electronic computer, with the running program stored in its memory. Today, all computers are based on the von Neumann principle.

Components:
- **Processor** (Arithmetic-Logic Unit (ALU), control unit, that „executes programs“)
- **Main Memory** (stores instructions and data)
- **Peripheral** (Input / Output devices (ex. printer, mouse, monitor, harddisk))
- **System bus** for interconnecting the components
Processor (Central Processing Unit): executes machine-coded programs

- The program is a sequence of machine instructions
- Machine instructions define operations

The main memory stores machine instructions and the required data

System bus (bidirectional): interconnects existing components, transmits data, instructions, control signals; bus allocation depending on the devices priorities
Von-Neumann-Architecture: working principle

1. Instruction are fetched from the main memory to the processor (read access)
2. Decode instruction, possibly read operand from the main memory or the computers peripherials
3. Execute command, possibly write to the main memory or the peripherials (write access)
4. Restart with step 1

- Computer executes commands permanently
- Computer is a reactive system
Von-Neumann bottle neck

- Processor executes the instructions 10 to 100 times faster than memory access is

von-Neumann – bottle neck:

- **Work-around:**
  - Registers: Memory on the processor
  - Buffer storage: (engl. cache) fast storage
  - separate data and bus instructions
Organisation of the main memory (today)

- Organized as a sequence of storage cells of size of 8 bits (1 byte), counting starting with 0

**Address:**
- Number of the storage cell

**Direct memory access:**
- Peripherals access the memory directly (no detour over the processor)
- Several peripherals (printer, network card) have their own processor

**Observation:** 98% of the processor sold today are employed in embedded systems (special tasks to be performed in cars, TVs, telecommunication devices, …)

**But:** Intel delivers 2% of the rest and makes 90% of the profit