Grundzüge der Wirtschaftsinformatik
Introduction to Business Information Systems

Unit 5

Prof. Dr. Martin Hepp
http://www.heppnetz.de
mhepp@computer.org

http://www.heppnetz.de/teaching/gwi/

Logistics

- **Lecture**
  - Tuesdays, 13:15 - 14:45, Auditorium Maximum (Building 33)
- **Tutorial and Exercises**
  - Wednesdays, 11:30 – 13:00, Building 33 Room 2401 (in German)
  - Thursdays, 09:45 - 11:15, Building 43 Room 4/126 (in German)
  - Thursdays, 15:00 - 16:30, Building 33 Room 2216 (in German)
  - ~~Thursdays, 16:45 - 18:15, Building 33 Room 2116 (in German)~~
- **Exam**
  - In conjunction with the exam in „Accounting“
  - Date: January 18, 2008
  - Time: 13:00 – 15:00
  - Classroom: Building 35 R, rooms 1210 A und 1210 B

Structure of the Lecture

Unit 1: Introduction
Unit 2: Central Processing Units
Unit 3: Storage and Data Structures
Unit 4: Input and Output Devices
Unit 5: Software
Unit 6: Networks, Data Interchange, and the Internet
Unit 7: Design, Development, Deployment, and Operations of Information Systems
Unit 8: Office Applications
Unit 9: Enterprise Applications
Unit 10: Supply Chain Applications and E-Business
Unit 11: Management Support Systems
Unit 12: Exam Review

Assignment from last week

- WI2, pp. 301-385; IBIS, pp. 20-31
- Review the slides

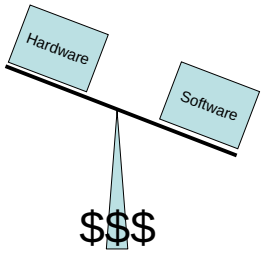
WI1 = Hansen/Neumann: Wirtschaftsinformatik 1; WI2 = Hansen/Neumann: Wirtschaftsinformatik 2; IBIS = Wigand et al: Introduction to Business Information Systems.

Link to the Previous Unit

- Last Unit:
  - How can we collect information from reality and get it into a computer system?
  - How can we display or print out computer data?
  - What types of equipment exist and how do they work?
- Today:
  - How can we tell a computer what to do?
  - What is a program? What is an operating system? How do they interact?
  - What languages and tools exist for developing software?

Hardware vs. Software

- **Hardware:** The tangible components of a computer system
  - CPU, power supply, display, memory, ...
- **Software:** The intangible components of a computer system
  - Applications
  - Operating system
  - Data
  - Documentation and instructions



cf. Stair/Reynolds

Problem, Algorithm, Program

- A **problem** is a *task described by its input and expected output*.
  - Example: Sorting
- An **algorithm** is the *description of a solution* for a given problem by *breaking the problem into a sequence of simple instructions.*
- A **program** is the *implementation of an algorithm for a specific type of computer*.

An Overview of Software

- **Computer program** - sequences of instructions for the computer
- **Documentation** - describes program functions
- **Systems software** - coordinates the activities of hardware & programs
- **Applications software** - helps users solve particular problems

cf. Stair/Reynolds

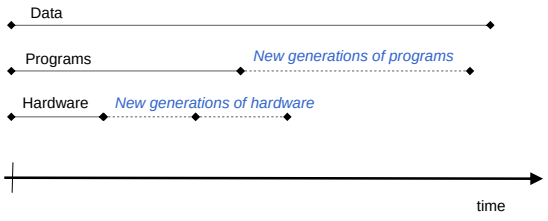
Categories of Software

- Operating Systems
- Development Tools
- Applications
  - Office Applications
  - Business Applications

The Economics of Software Development

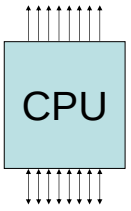
- High development costs, low distribution costs
- Network Externalities in the Software Market

Duration of Use: Hardware, Programs, and Data

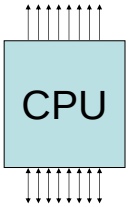


Machine Language

- The CPU can directly understand only a very small set of very simple commands.
- Each command is assigned a single number.
- This allows us to tell a computer what to do by
  - putting a sequence of numbers into its memory and
  - telling the CPU to start executing the commands represented by those numbers at a given address.



Machine Language (2)

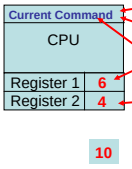


Bit Pattern	Decimal Value	Command
0000 0000	0	Wait a little moment
0000 0001	1	Load the value ***from the next memory cell*** into register 1
0000 0010	2	Load the value ***from the next memory cell*** into register 2
0000 0011	3	Add the two values from register 1 and register 2 and store the result in register 1
0000 0100	4	Subtract the value from register 2 from register 1 and store the result in register 1
0000 0101	5	Multiply the two values from register 1 and register 2 and store the result in register 1
0000 0110	6	Divide the two values in register 1 by the value in register 2 and store the result in register 1

Register:
The few memory locations included in the CPU.

Machine Language: Example

Execute the program that starts at address 0



Address	Bit Pattern	Decimal Value	Command
0	0000 0001	1	Load the value ***from the next memory cell*** into register 1
1	0000 0110	6	Value 6
2	0000 0010	2	Load the value ***from the next memory cell*** into register 2
3	0000 0100	4	Value 4
4	0000 0011	3	Add the two values from register 1 and register 2 and store the result in register 1

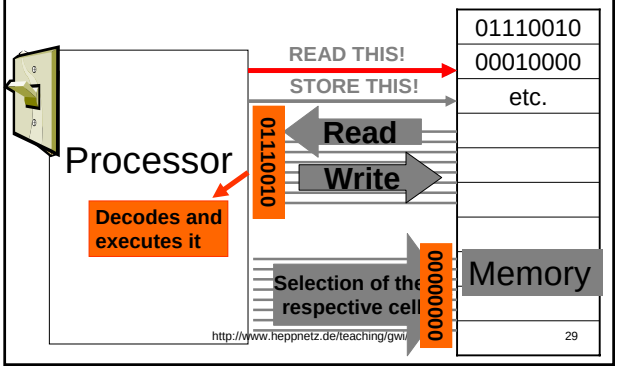
In this example, the byte sequence (1,6,2,4,3) is a machine-language program that computes „6+4“.

Machine Language: Data and Instructions

Address	Bit Pattern	Decimal Value	Command
0	0000 0001	1	**Load the value *from the next memory cell* into register 1**
1	0000 0110	6	**Value 6**
2	0000 0010	2	**Load the value *from the next memory cell* into register 2**
3	0000 0100	4	**Value 4**
4	0000 0011	3	**Add the two values from register 1 and register 2 and store the result in register 1**

The CPU can only distinguish
(a) numbers that represent instructions from
(b) numbers that represent data
by whether the previous command comes with an additional value.

When you turn on your Computer…



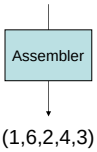
Assembler Language

Bit Pattern	Decimal Value	Mnemonic	Command
0000 0000	0	**WAIT**	Wait a little moment
0000 0001	1	**LOAD_R1, X**	Load the value ***from the next memory cell*** into register 1
0000 0010	2	**LOAD_R2, X**	Load the value ***from the next memory cell*** into register 2
0000 0011	3	**ADD**	Add the two values from register 1 and register 2 and store the result in register 1
0000 0100	4	**SUB**	Subtract the value from register 2 from register 1 and store the result in register 1
0000 0101	5	**MULT**	Multiply the two values from register 1 and register 2 and store the result in register 1
0000 0110	6	**DIV**	Divide the two values in register 1 by the value in register 2 and store the result in register 1

Assembler: Development Tool

- For almost any processor make and model there exists at least one program that can translate a program written in assembler language into the respective machine code.
- This type of program is also called „assembler“.

LOAD_R1, 6
LOAD_R2, 4
ADD

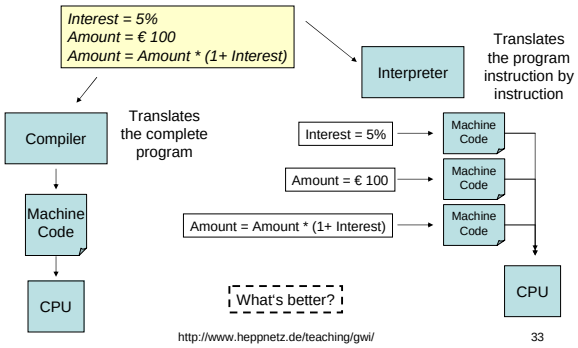


(1,6,2,4,3)

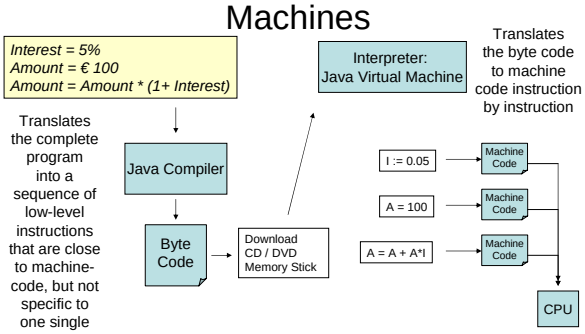
Higher-level Programming Languages

- Programming in machine language or assembler is difficult, tedious, and error-prone.
- After all, machine code is suited best for machines, not for humans.
- Also, machine code runs only on a very specific type of computer (→ portability)
- Modern programming languages provide more human-friendly ways of writing software.
- However, the resulting program cannot be directly executed by a computer.

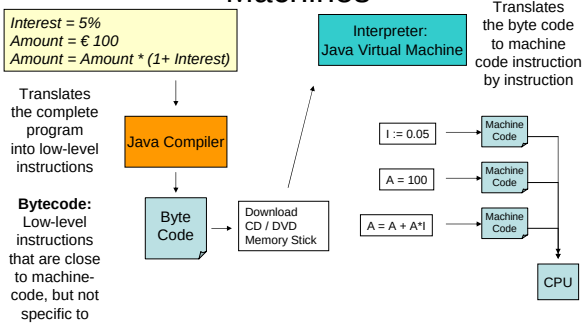
Compilers and Interpreters: Two Approaches of Translating to Machine Code



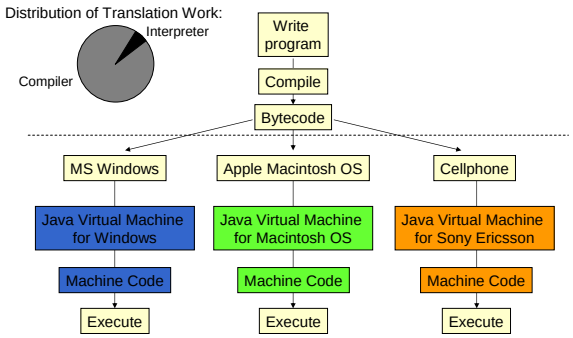
Java: Bytecode and Virtual Machines



Java: Bytecode and Virtual Machines



Advantages of the Java Approach



Control Flow and Control Flow Patterns

Control flow: The order of execution of the instructions in a program

Sequence:
Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5

Loop:
Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Repeat

Branch:
Instruction 1
Instruction 2a Instruction 2b
Instruction 3
Instruction 4
Instruction 5

BASIC

- Early high-level programming language

```
10 PRINT „PERSONAL GREETING“
20 PRINT „WHAT‘S YOUR NAME?“
30 INPUT NAME$
40 PRINT „HELLO, „+NAME$
50 FOR i=1 TO 5
60 PRINT „BYE“
70 NEXT i

Output:
HELLO, PETER
BYE
BYE
BYE
BYE
BYE
```


Procedural Programming

- **Idea:** Support better readability and simplify maintenance and reuse by a *modular programming style.*
- Small units of functionality form a procedure (subroutine) that can be invoked whenever needed.

```
Procedure computeInterest (amount, interestRate)
{... instructions on how to accomplish that...)


result = computeInterest (100, 0.05)
```

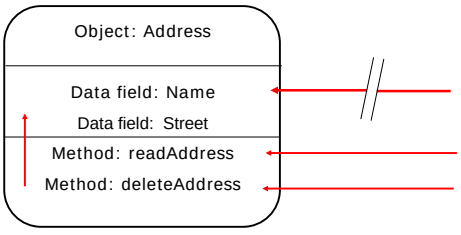

Object-oriented Programming: Motivation

- Developing large applications using procedural programming is error-prone, because
  - Other programmers may access the internal variables of other routines
  - The input and output of procedures is only defined by the data type (integer, character, float,...)
- Reuse of existing parts of programs is difficult, since they may depend on the rest of the program

Object-oriented Programming: Idea

- Develop software on the basis of *small, strictly encapsulated units*, which are called objects
  - Example: A specific customer, an invoice, a peripheral, the screen
- An object has an internal state
  - Example: A customer has a name, an address, a total sales volume
- The state of an object can *only* be changed by other objects *by calling well-defined procedures*, which are called methods
- *The internal realization of functionality in the object is not exposed to other objects* and will not affect other objects.

Core Principle of Object-oriented Software Development



Object-Oriented Programming Languages

- **Objects** - data and actions that can be performed on the data
- **Encapsulation** - group items into an object
- **Polymorphism** - one procedure can work with multiple objects
- **Inheritance** - an object in a particular class gets attributes of that class

cf. Stair/Reynolds

The Role of the Operating System

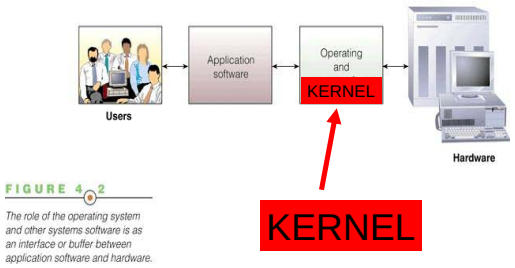


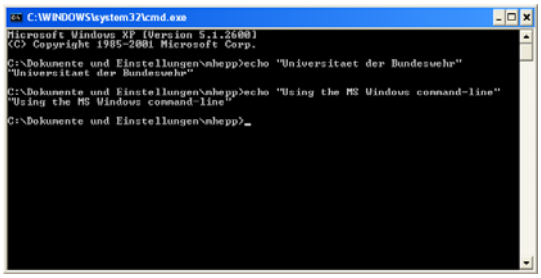
FIGURE 4.2

The role of the operating system and other systems software is as an interface or buffer between application software and hardware.

KERNEL

cf. Stair/Reynolds

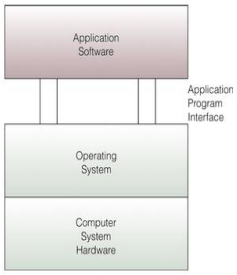
Command Line



Economic Benefit of a Modular Software Architecture: Hardware Independence

FIGURE 4.3

Application Program Interface Links Application Software to the Operating System



Examples:

•Accessing Printers (Windows-API) or

•Scanners and Digital Cameras (TWAIN)

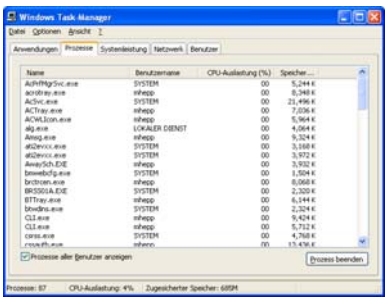
cf. Stair/Reynolds

Processing Tasks

- **Multitasking -** more than one program (task) can run at a time using a single processor
- **Multi-User OS -** multiple users can simultaneously use the resources of a single processor
- **Scalability -** easy adaptation to more users or tasks

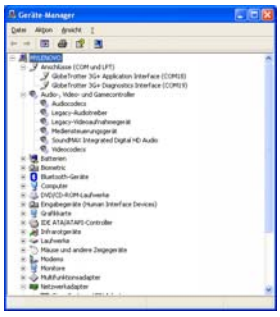
cf. Stair/Reynolds

Task Manager in MS Windows

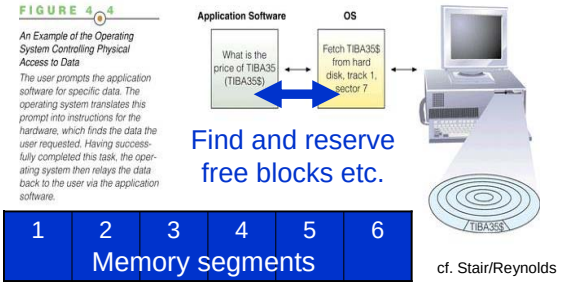


Driver Software: Support for Peripherals

- Small pieces of software that tell the computer how to access a particular peripheral
- Examples
  - Printers
  - Keyboards

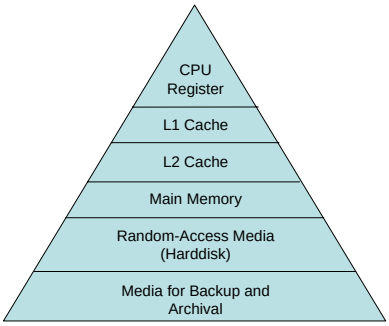


Memory Management



cf. Stair/Reynolds

Memory Hierarchy



System Speed and RAM

- Why does a bigger primary storage capacity (more RAM) increase your PCs speed?
- Does more memory always lead to a higher system performance?

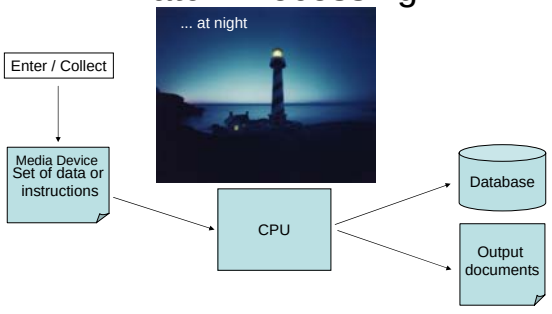
Off-the-Shelf Software

- Two approaches of developing software:
  - Custom development: Software for one particular usage
  - Common-of-the-Shelf (COTS): Software for a large number of usages
    - MS Office
    - SAP
    - Netscape

What is **Integrated** Software?

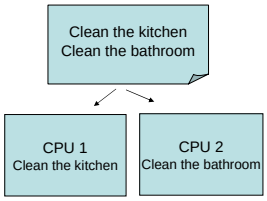
- Data Integration: "One fact at one place"
- Functionality Integration: one function can interact with others (copy and paste inside one program)
- Application Integration: Two or more software applications can interact.
- Process Integration: Two or more business processes are connected.

Batch Processing



Parallel Computing

- Program execution can be accelerated by distributing the task across multiple CPUs
- However, this works well only for tasks that can be broken down into independent sub-tasks.



Assignment for Next Week

- WI2, pp. 517-749; IBIS, pp. 34-51
- Review the slides

WI1 = Hansen/Neumann: Wirtschaftsinformatik 1; WI2 = Hansen/Neumann: Wirtschaftsinformatik 2; IBIS = Wigand et al: Introduction to Business Information Systems.

Thank you!

The slides and additional materials will be available at
http://www.heppnetz.de/teaching/gwi/