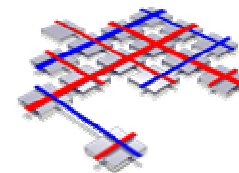


Computer Science II

(Summer Semester 2003)

Prof. Dr. Dieter Hogrefe
Dr. Xiaoming Fu
Kevin Scott, M.A.

Telematics group
University of Göttingen, Germany



Course Administration

- Instructor:
 - Prof. Dr. Dieter Hogrefe
 - Dr. Xiaoming Fu
 - Kevin Scott, M.A.
- Office Hours
 - D. Hogrefe: per E-mail (hogrefe@informatik.uni-goettingen.de)
 - X. Fu: Thursday 16:00-17:00pm
- TA: Dipl.-Inf. Rene Soltwisch
- Prerequisites:

Computer Science I; basic familiarity with Java or C
- Final Exam:

Prerequisite: passing 8 out of 10 homeworks
Date: July 22, 2003 Tuesday (Week 30), 14:00-16:00pm

Tutor Groups

- Group 1: Monday 8:00-10:00, VG419
- Group 2: Wednesday 9:00-11:00, VG319 (German & English)
- Group 3: Wednesday 10:00-12:00, VG419
- Group 4: Wednesday 12:00-14:00, VG419
- Group 5: Wednesday 16:00-18:00, VG419
- Group 6: Wednesday 18:00-20:00, VG316
- Group 7: Thursday 10:00-12:00, VG413
- Group 8: Friday 16:00-18:00, VG419

*Please register in **one** of the application forms (during the class 29.04.03)!*

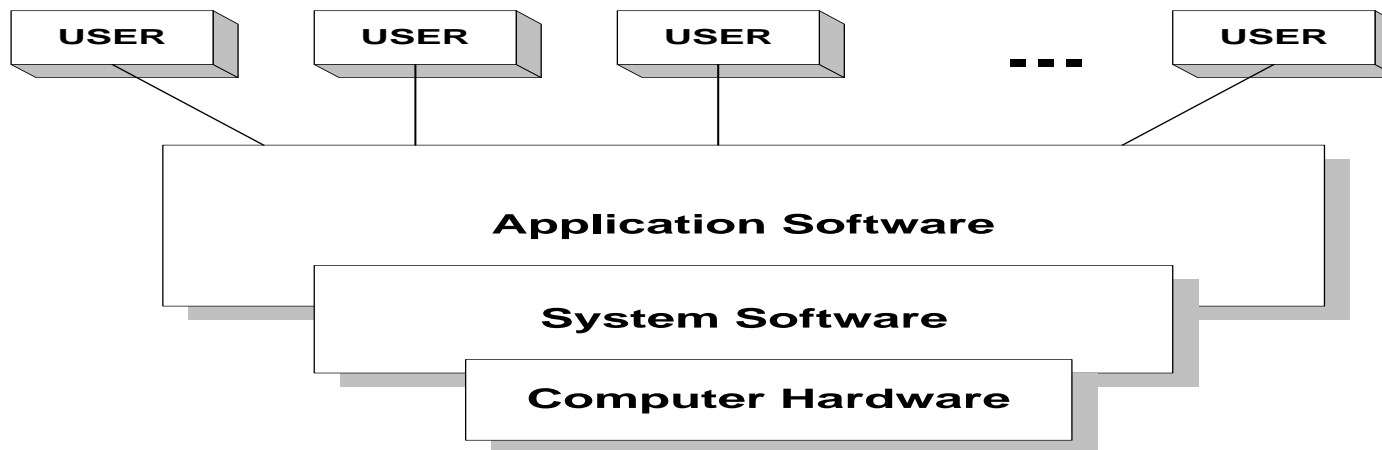
Course Overview

- Schedules:
 - Part I: Digital Logic, Boolean Algebra (Week 18)
 - Part II: Computer Systems Organization, von Neumann Architecture (Week 19)
 - Part III: Assembly Language (Week 20)
 - Part IV: Operating Systems (Week 21-22)
 - Part V: Computer Communications (Week 23,25)
 - Part VI: Compilers (Week 26-27)
 - Part VII: Automata and formal languages (Week 28-29)
- Homework Assignment and Exercise Courses:
 - Each Tuesday announced in the course page for "Computer Science-II"
 - Possibly, including both practical and theoretical exercises
 - The coming week: hand in your homework to corresponding tutors
 - The following week (except week 18, 30): illustrations of homework, answer questions - different tutor's groups
 - Passing 8 out of 10 assignments, you're eligible to take the final exam

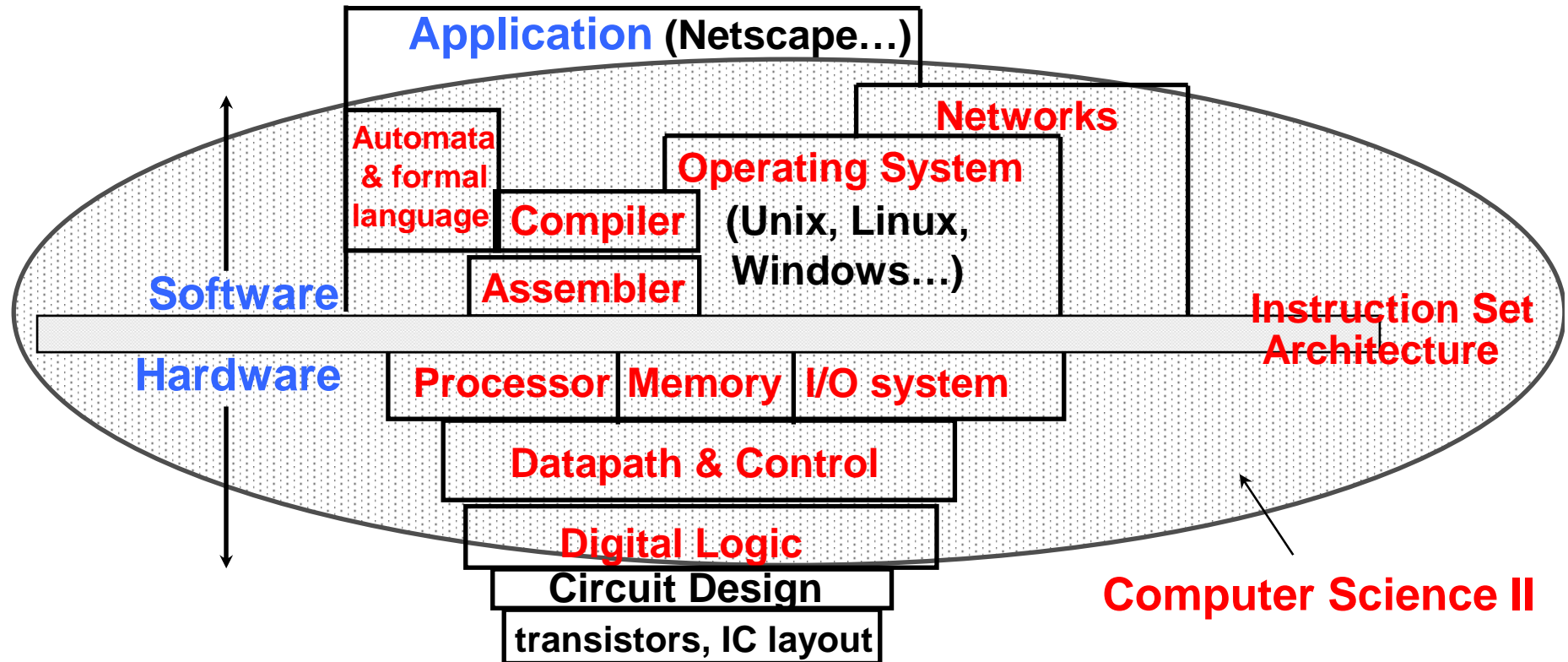
Textbooks

- A. Tanenbaum, "**Structured Computer Organization**" 4th edition (chapter 1-3), Prentice Hall, 1999
 - A. S. Tanenbaum / J. Goodman, "*Computerarchitektur*", 2001. (German)
- D. Patterson / J. Hennessy, "**Computer Organization and Design: The Hardware/Software Interface**" (Appendix A), 2nd edition, Morgan Kaufmann Publishers, 1997
- A. Tanenbaum, "**Modern Operating Systems**" 2nd edition (chapter 1-6, 9-10), Prentice Hall, 2001
 - A. Tanenbaum, *Moderne Betriebssysteme* (German)
- J. Kurose / K. Ross, "**Computer Networking**" 2nd edition, Addison-Wesley, 2002
 - *Computernetze: Ein Top-Down-Ansatz mit Schwerpunkt Internet* (German)
- M. Scott, "**Programming Language Pragmatics**", Morgan Kaufmann Publishers, 2000

Abstract View of Hardware and Software Components



How Does a Computer Work?



- Key Idea: *levels of abstraction*
 - hide unnecessary implementation details
 - helps us cope with enormous complexity of real systems

Virtual Machines

- Programming Language analogy:
 - Each computer has a native machine language (language L0) that runs directly on its hardware
 - A more human-friendly language is usually constructed above machine language, called Language L1
- Programs written in L1 can run two different ways:
 - Interpretation – L0 program interprets and executes L1 instructions one by one
 - Translation – L1 program is completely translated into an L0 program, which then runs on the computer hardware

Translating Languages

English: Display the sum of A times B plus C.

C++: `cout << (A * B + C);`

Assembly Language:

```
mov eax,A
mul B
add eax,C
call WriteInt
```

Intel Machine Language:

```
A1 00000000
F7 25 00000004
03 05 00000008
E8 00500000
```

Multilevel Machines

| | |
|---------|------------------------------|
| Level 5 | Application Program Language |
| Level 4 | Assembly Language |
| Level 3 | Operating System |
| Level 2 | Instruction Set Architecture |
| Level 1 | Microarchitecture |
| Level 0 | Digital Logic |

High-Level Language

- Level 5
- Application-oriented languages
 - C++ (C), Java, Pascal, Visual Basic . . .
- Programs **compile** (translate) into assembly language (Level 4)

Assembly Language

- Level 4
- Instruction mnemonics that have a one-to-one correspondence to machine language
- Calls functions written at the operating system level (Level 3)
- Programs are translated (**assembled**) into machine language (Level 2)

Operating System

- Level 3
- Provides services to Level 4 programs
- Translated (**partially interpreted**) and run at the instruction set architecture level (Level 2)

Instruction Set Architecture

- Level 2
- Also known as conventional machine language
- **Executed (or interpreted)** by Level 1 (microarchitecture) program

Microarchitecture

- Level 1
- Interprets conventional machine instructions (Level 2)
- Executed by digital **hardware** (Level 0)

Digital Logic

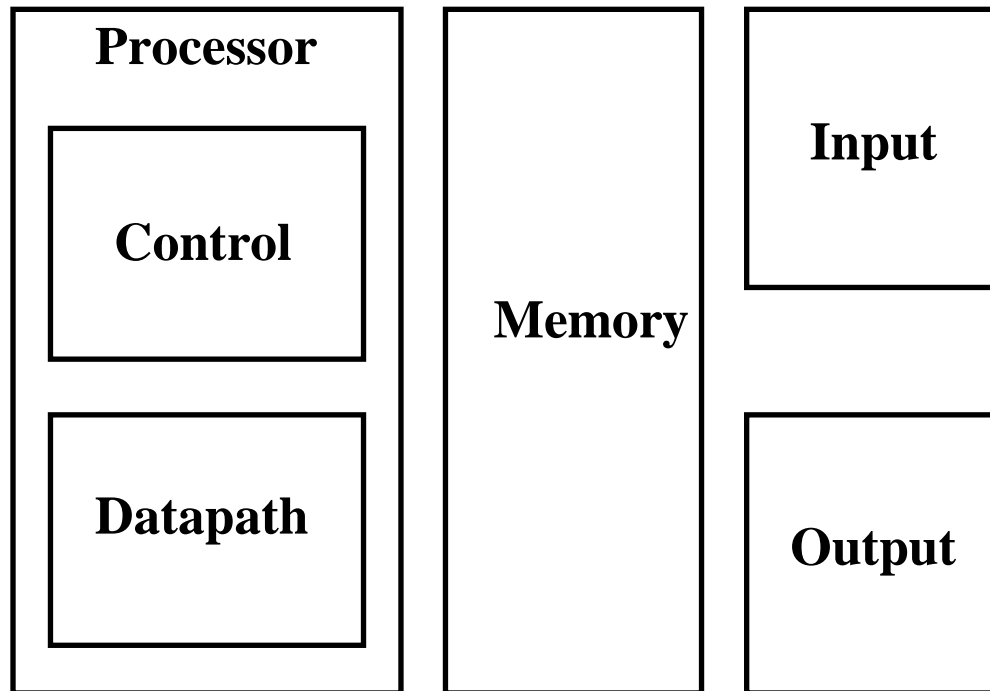
- Level 0
- CPU, constructed from digital logic **gates**
- System bus
- Memory
- Implemented using bipolar transistors

Computer Families

- The **instruction set** chosen for a computer determines the way that machine language programs are constructed
- Either of the following architectures define different family of processors
 - **Reduced Instruction Set Computers (RISC)**
 - Fewer instructions, fewer addressing modes, fixed length
 - E.g., Sun SPARC, **MIPS**, PowerPC
 - **Complex Instruction Set Computers (CISC)**
 - Large number of operations & addressing modes
 - E.g., Pentium, Motorola 68000

The Big Picture

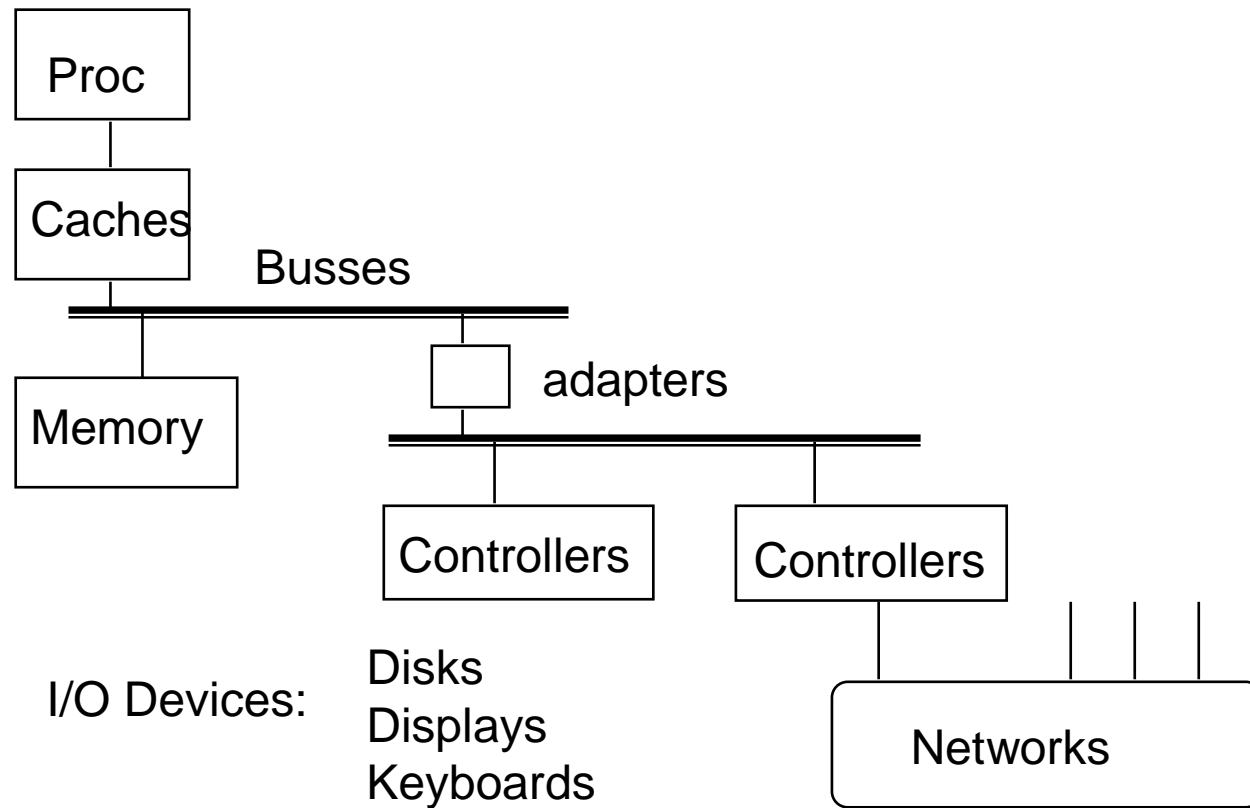
- Since 1946 all computers have had 5 main components



Components of a Computer

- The functions of the different computer components are
 - datapath - performs arithmetic and logic operations
 - e.g., adders, multipliers, shifters
 - memory - holds data and instructions
 - e.g., cache, main memory, disk
 - input - sends data to the computer
 - e.g., keyboard, mouse
 - output - gets data from the computer
 - e.g., screen, sound card
 - control - gives directions to the other components
 - e.g., bus controller, memory interface unit

Computer System Components



- All have interfaces & organizations

Architecture and Programming

- Knowing about architecture helps to explain why programming languages are designed the way they are.
 - What happens when we compile our source code?
 - Why is computer arithmetic sometimes wrong?
 - What is a bus error or segmentation fault?
- You can also learn how to make your code run faster.
 - Where and how you store your data makes a big difference.
 - Just rearranging the order of statements can sometimes help!
- A lot of software development requires knowledge of computer systems.
 - 5 classic components of any computer
 - Compilers generate optimized code for specific processors.
 - Compilation vs. interpretation to move down the layers of the computer system
 - Operating systems manage hardware resources for applications.
 - Good I/O systems are important for databases and networking.
 - Stored program concept: instructions and data stored in memory

Summary

- Computer Science II studies how a computer works:
 - Basic computer hardware components: datapath, memory, input devices, output devices, and control; also underlying digital logic.
 - Principles of system software: operating systems, communication systems, compilers; automata and formal languages theory.
- Principle of abstraction is useful to study and build systems as layers
 - Good data representation is important to increase system performance, lower resource utilization and improve accuracy.
 - Abstraction and hierarchical designs are critical to control complexity.
 - Raw data (binary bit patterns) can mean anything (integers, floating point numbers, chars, etc): a program determines what it is.
- Knowledge of computer systems helps programming and software engineering