CSE-413 Computer Architecture
Lecture 1

Introduction

References

- **Computer Architecture and Organization**
  Hayes J.P., McGraw-Hill.

- **Computer organization and design: The hardware/software interface**
  Patterson D.A., Hennessy J.L., Morgan Kaufmann.

About the Course

- Course Code: CSE-413
- Course Title: Computer Architecture
- Course Teacher: Ayesha Siddika
- Credit: 3
- Total Marks: 100
- Lecture: 22+

Class Schedule

- Two classes per week:
  - Sunday/Tuesday : 12:00 pm – 1:20 pm

What is Computer Architecture?

- Programmer's view: a pleasant environment
- Operating system's view: a set of resources (hardware & software)
- System architecture view: a set of components
- Compiler's view: an instruction set architecture with OS help
- Microprocessor architecture view: a set of functional units
- VLSI designer's view: a set of transistors implementing logic
- Mechanical engineer's view: a heater!

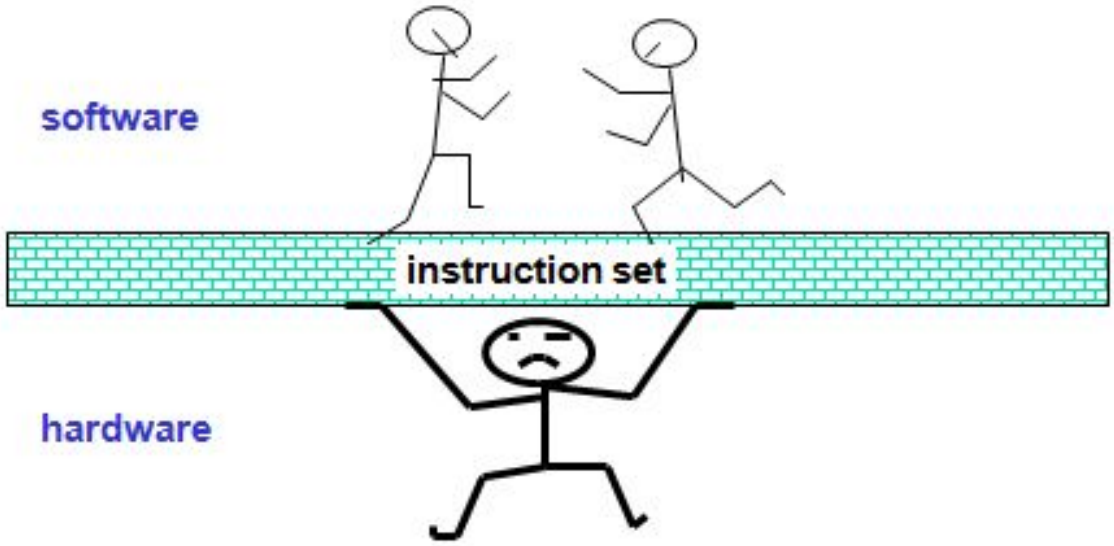
Definition of computer Architecture

In computer engineering, **computer architecture** is a set of rules and methods that describe the functionality, organization, and implementation of computer systems. Some definitions of architecture define it as describing the capabilities and programming model of a computer but not a particular implementation. In other descriptions computer architecture involves instruction set architecture design, microarchitecture design, logic design, and implementation.

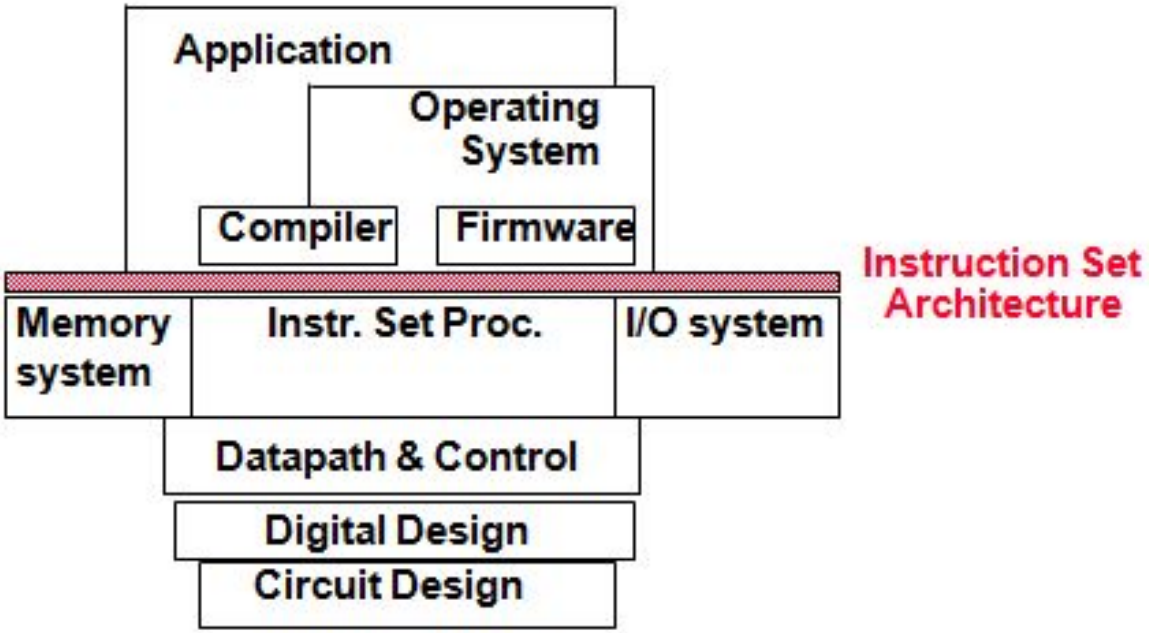
Computer architecture = Instruction set architecture + Machine organization + Hardware

Instruction Set Architecture (ISA)

The *Instruction Set Architecture* (ISA) is the part of the processor that is visible to the programmer or compiler writer.
The ISA serves as the boundary between software and hardware.



Cont.



- Example: MIPS

- Desired properties
  - Convenience (from software side)
  - Efficiency (from hardware side)

Cont.

- Part of computer architecture related to programming

- Define registers
  - R1, R2,...
- Define data transfer modes (instructions) between registers, memory and I/O
  - e.g. MOV, ADD, INC

- There should be sufficient instructions to efficiently translate any program
  - e.g. MOV, ADD, INC

Machine Organization

High-level aspects of a computer's design

- Principal components: memory, CPU, I/O, …
- How components are interconnected
- How information flows between components
- e.g. AMD Opteron 64 and Intel Pentium 4: same ISA but different organizations

Hardware

Detailed logic design and the packaging technology of a computer

- E.g. Pentium 4 and Mobile Pentium 4: nearly identical organizations but different hardware details

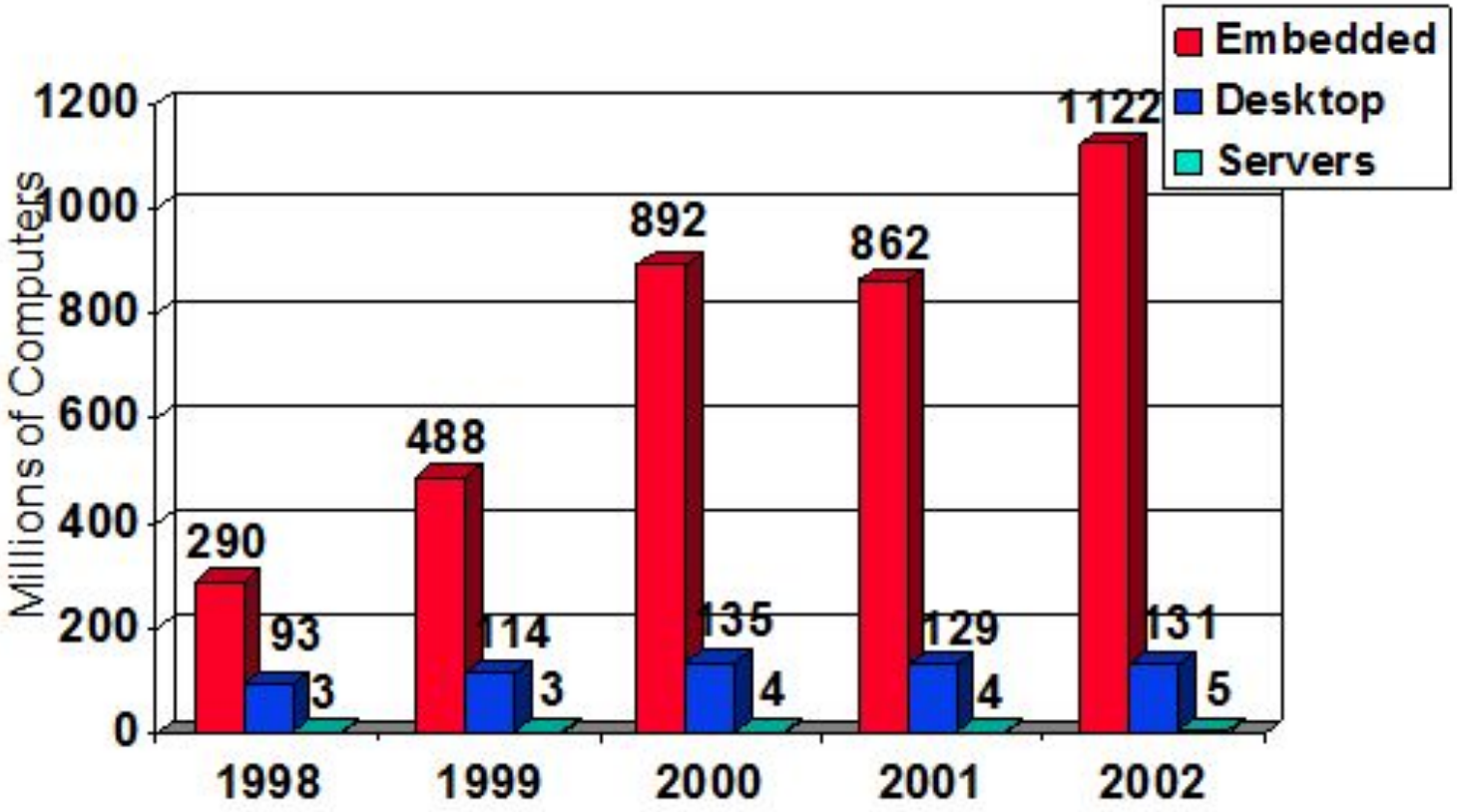
Types of computers and their applications

- Desktop
  - Run third-party software
  - Office to home applications
- Servers
  - Modern version of what used to be called mainframes, minicomputers and supercomputers
  - Large workloads
  - Built using the same technology in desktops but higher capacity
    - Expandable
    - Scalable
    - Reliable
  - Large spectrum: from low-end (file storage, small businesses) to supercomputers (high end scientific and engineering applications)
    - Gigabytes to Terabytes to Petabytes of storage
  - Examples: file servers, web servers, database servers

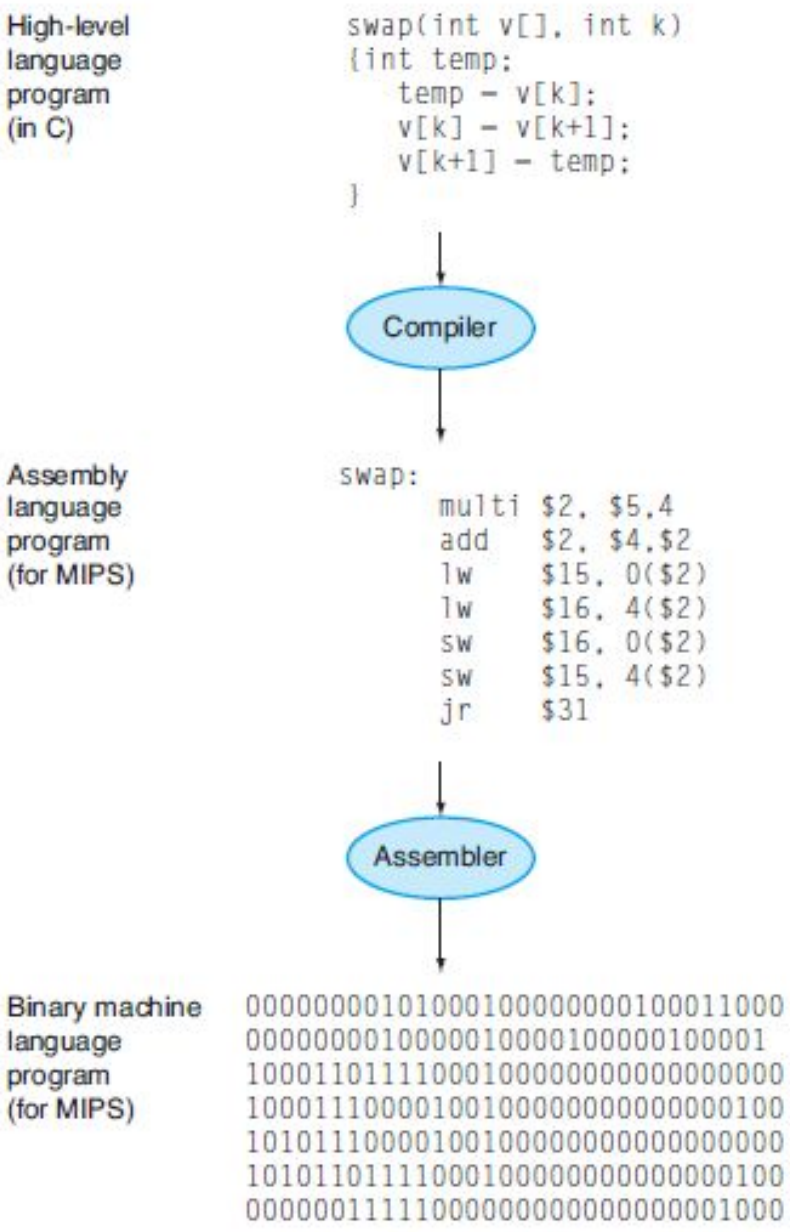
Cont.

- Embedded
  - Microprocessors everywhere! (washing machines, cell phones, automobiles, video games)
  - Run one or a few applications
  - Specialized hardware integrated with the application
  - Usually stringent limitations (battery power)
  - High tolerance for failure
  - Becoming ubiquitous
  - Engineered using *processor cores*
    - The core allows the engineer to integrate other functions into the processor for fabrication on the same chip
    - Using hardware description languages: Verilog, VHDL

What is the Market ?



From a High-Level Language to the Language of Hardware



Evolution…

- In the beginning there were only bits… and people spent countless hours trying to program in machine language

  01100011001 011001110100

- Finally before everybody went insane, the **assembler** was invented: write in mnemonics called assembly language and let the assembler translate (a one to one translation)

  Add A,B

Evolution…Cont.

- Assembler is a program that translates a symbolic version of instructions into the binary version.
- This program translates a symbolic version of an instruction into the binary version. For example, the programmer would write

```
add A, B
```

and the assembler would translate this notation into

```
1000110010100000
```

This instruction tells the computer to add the two numbers A and B.

Evolution…Cont.

- The name coined for this symbolic language, still used today, is assembly language. In contrast, the binary language that the machine understands is the machine language.
- Assembly language requires the programmer to write one line for every instruction that the computer will follow, forcing the programmer to think like the computer.
- This wasn't for everybody, obviously… (imagine how modern applications would have been possible in assembly), so high-level language were born (and with them compilers to translate to assembly, a many-to-one translation)

Evolution…Cont.

- A compiler enables a programmer to write this high-level language expression:

A+B

- The compiler would compile it into this assembly language statement:

add A,B

- The assembler would translate this statement into the binary instructions that tell the computer to add the two numbers A and B.

Benefits of High Level Programming Language

- First, they allow the programmer to think in a more natural language, using English words and algebraic notation, resulting in programs that look much more like text.

- Moreover, they allow languages to be designed according to their intended use. Hence, Fortran was designed for scientific computation, Cobol for business data processing, Lisp for symbol manipulation, and so on.

- There are also domain-specific languages for even narrower groups of users, such as those interested in simulation of fluids, for example.

Cont.

- The second advantage of programming languages is improved programmer productivity.

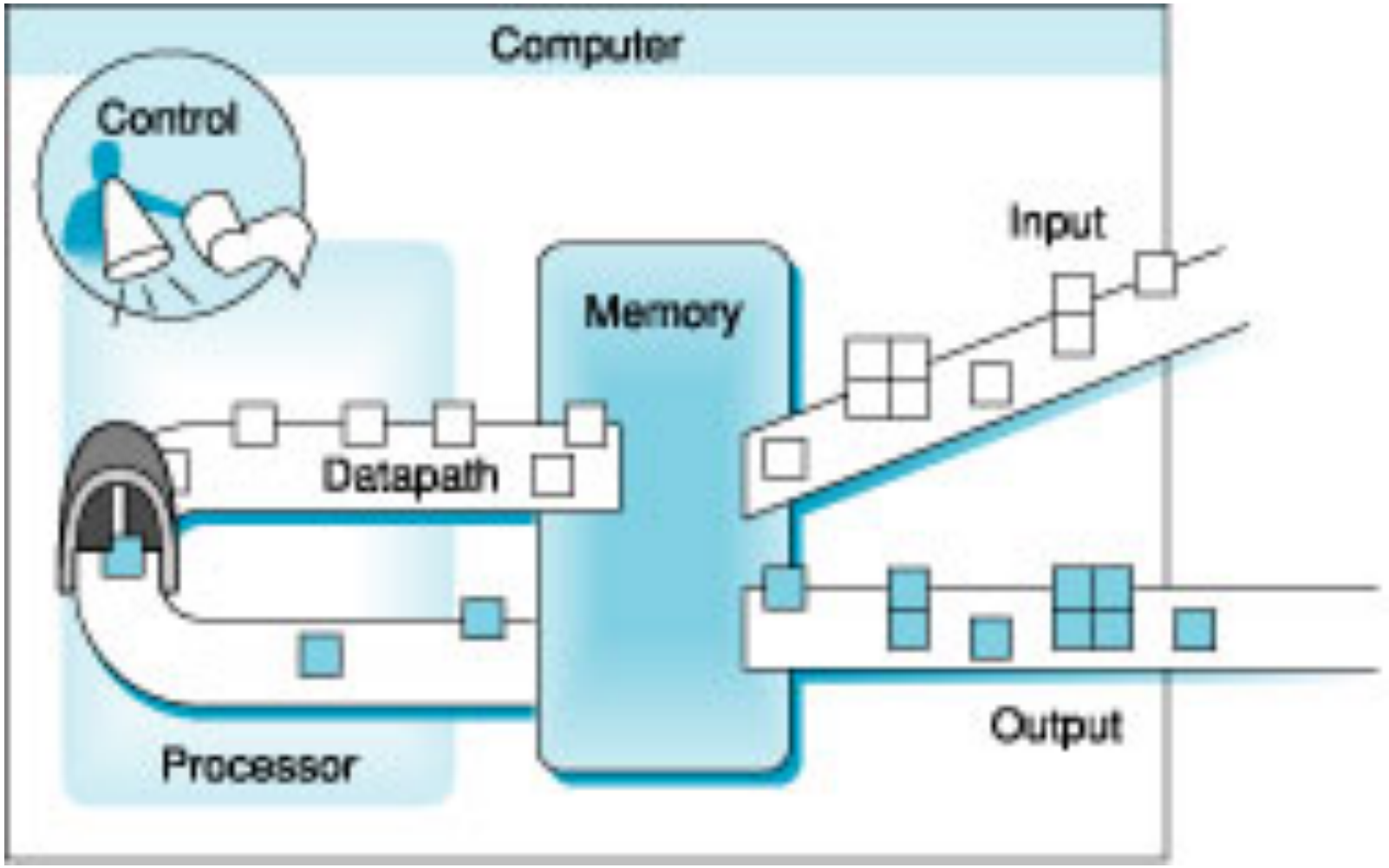
- One of the few areas of widespread agreement in software development is that it takes less time to develop programs when they are written in languages that require fewer lines to express an idea.

Cont.

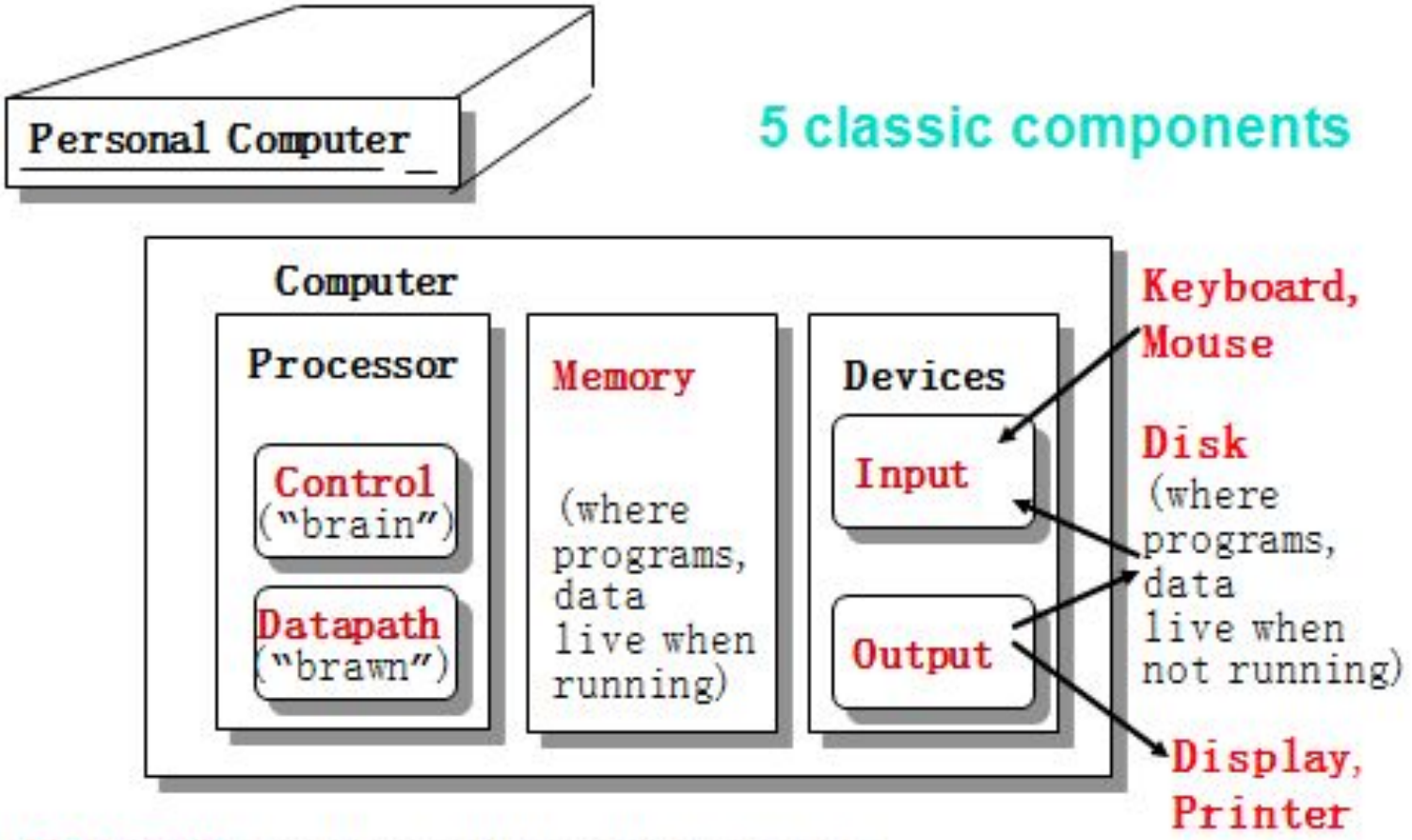
- The final advantage is that programming languages allow programs to be independent of the computer on which they were developed, since compilers and assemblers can translate high-level language programs to the binary instructions of any computer.

- These three advantages are so strong that today little programming is done in assembly language.

Organization of a Computer



Anatomy of a Computer



- Datapath: performs arithmetic operation
- Control: guides the operation of other components based on the user instructions