Datorarkitektur

(Advanced Computer Architecture)

Petru Eles
Institutionen för Datavetenskap (IDA)
Linköpings Universitet

email: petru.eles@liu.se
http://www.ida.liu.se/~petel
phone:013 281396
B building, room 329:220

Course Information

Web page: http://www.ida.liu.se/~TDDI03

Examination: written, January 16th, 2027, kl. 8 - 12.

Lecture notes: available from the web page, latest 24 hours before the lecture.

Text book: William Stallings: *Computer Organization and Architecture*, Pearson/Prentice Hall, 9th edition, 2013 or 10th edition, 2016 or 11th edition, 2021.

In addition to lectures: Two big seminars!

Preliminary Course Plan

Lecture 1

Introduction: **Outline, Basic computer architecture and organization, Basic functions of a computer and its main components, The von Neumann architecture.**

This is to refresh your memory!!!

Lectures 2 and 3

The Memory System: **Memory hierarchy, Cache memories, Virtual memories, Memory management.**

Lectures 4 and 5

Instruction Pipelining: **Organization of pipelined units, Pipeline hazards, Reducing branch penalties, Branch prediction strategies.**

Lectures 6

RISC Architectures: **An analysis of instruction execution for code generated from high-level language programs, Compiling for RISC architectures, Main characteristics of RISC architectures, RISC-CISC trade-offs.**

Preliminary Course Plan

Lectures 7 and 8

Superscalar Architectures: Instruction level parallelism and machine parallelism, Hardware techniques for performance enhancement, Data dependencies, Policies for parallel instruction execution, Limitations of the superscalar approach.

Lectures 9 and 10

VLIW Architectures: The VLIW approach - advantages and limitations. Compiling for VLIW architectures. The Merced (Itanium) architecture.

Lectures 11 and 12

Architectures for Parallel Computation: Parallel programs, Performance of parallel computers, A classification of computer architectures, Array processors, Multiprocessors, Multicomputers, Vector processors.

Multiprocessors on chip.

Multithreaded processors.

General purpose graphic processors.

COMPUTER ARCHITECTURE

(BASIC ISSUES)

1. What is a Computer/Computer System?

2. The von Neumann Architecture

3. Application Specific vs. General-Purpose

4. Representation of Data and Instructions

5. Instruction Execution

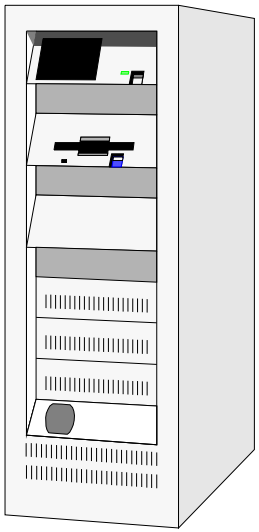
6. The Control Unit

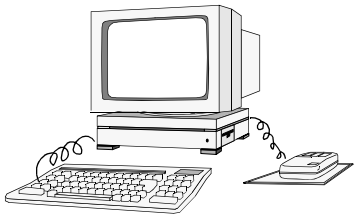
7. The Computer System

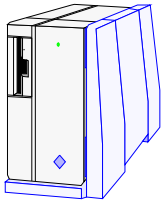
8. Main and Secondary Memory

9. The Intel x86 and ARM Families

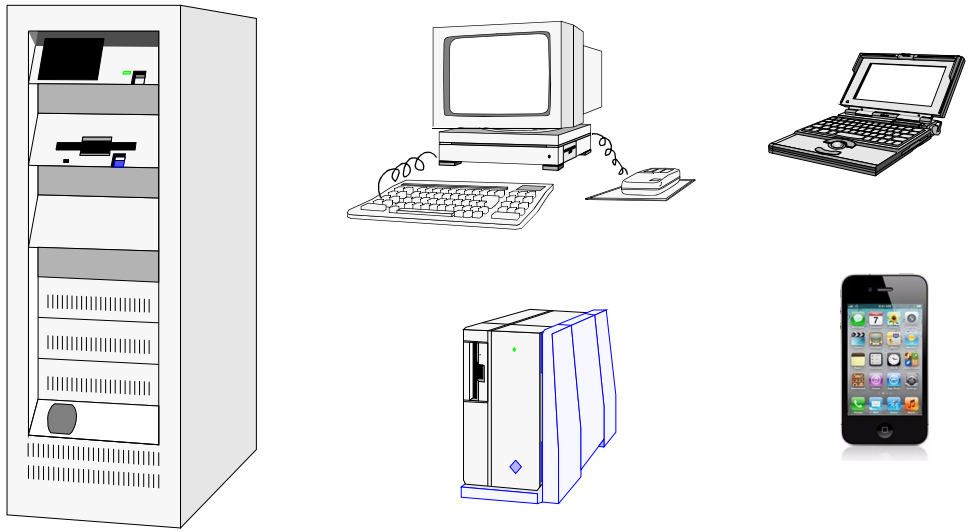
What Is a Computer?





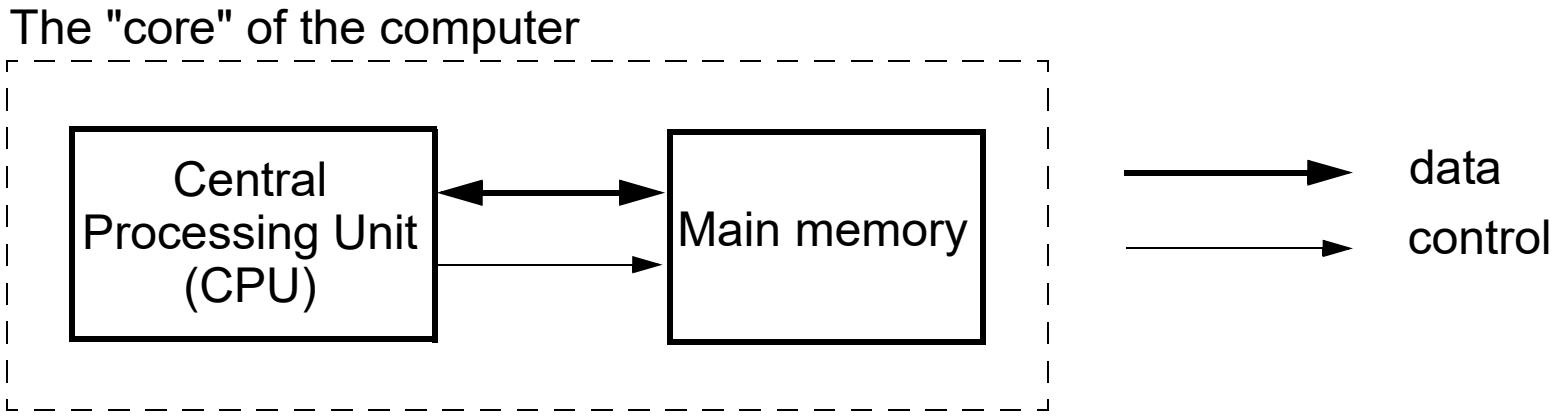


What Is a Computer?

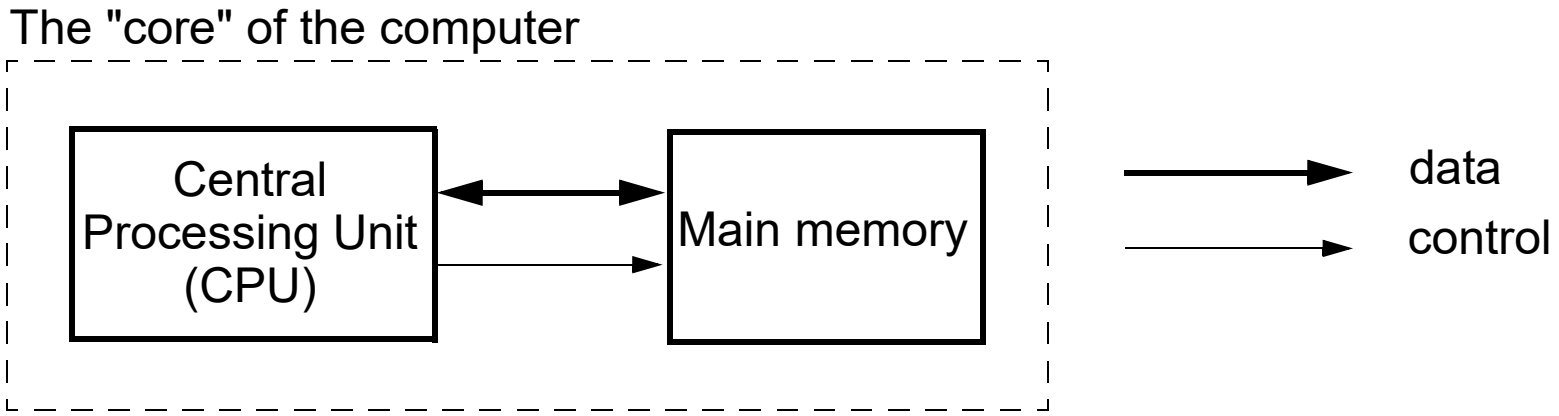


- A computer is a data processing machine which is operated automatically under the control of a list of instructions (called a program) stored in its main memory.

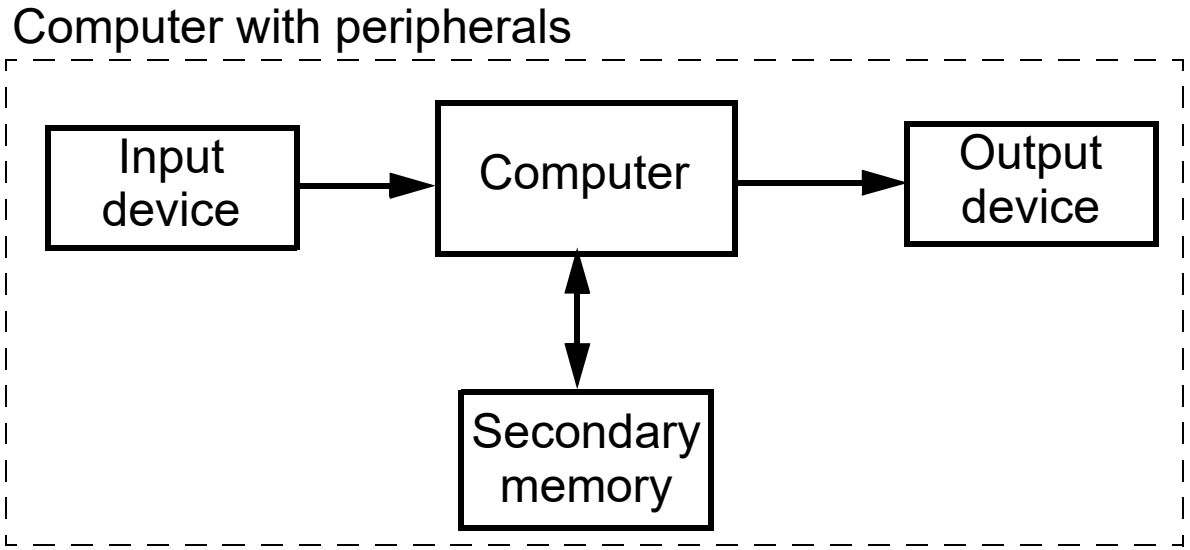
What Is a Computer?



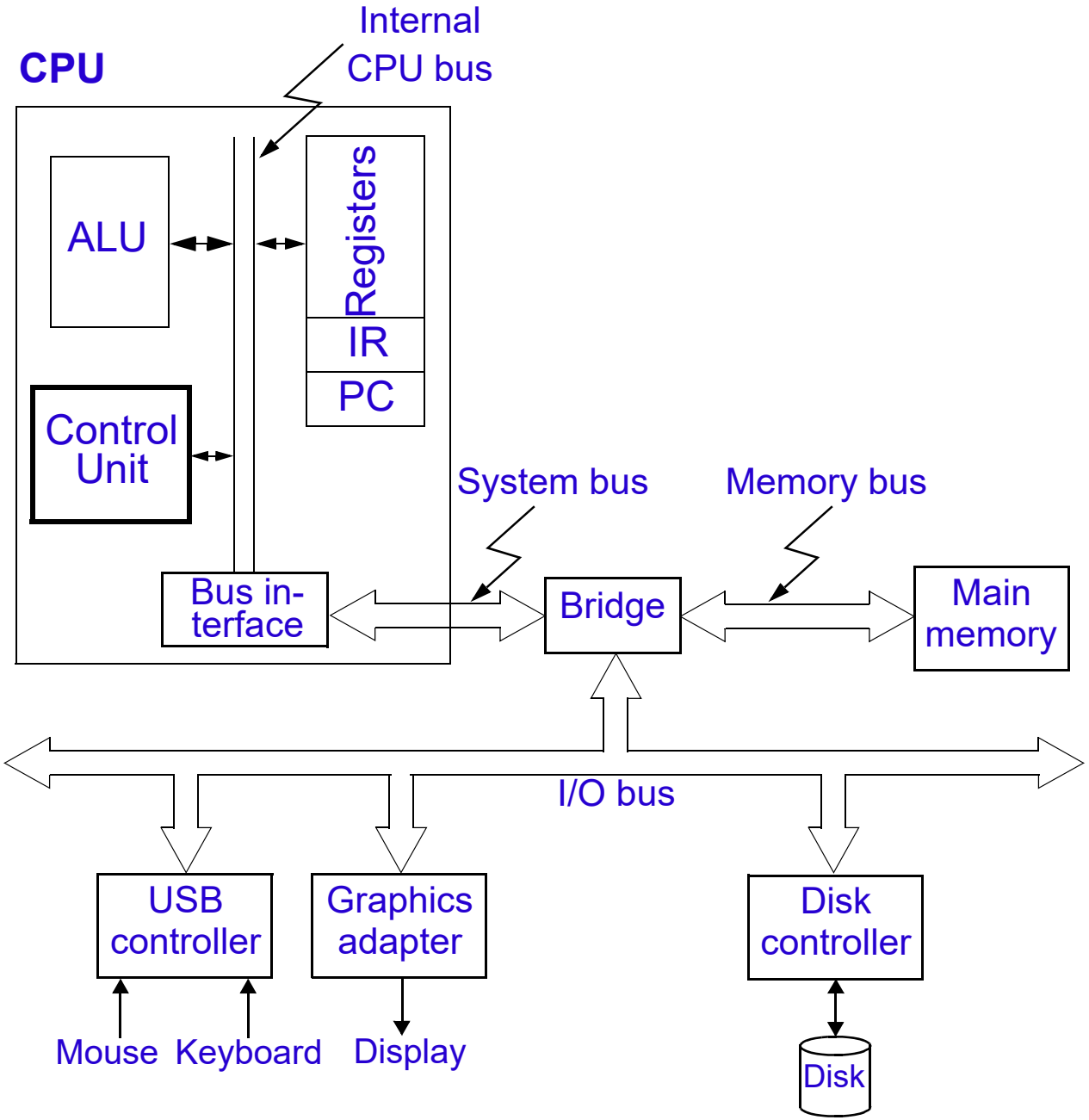
What Is a Computer?



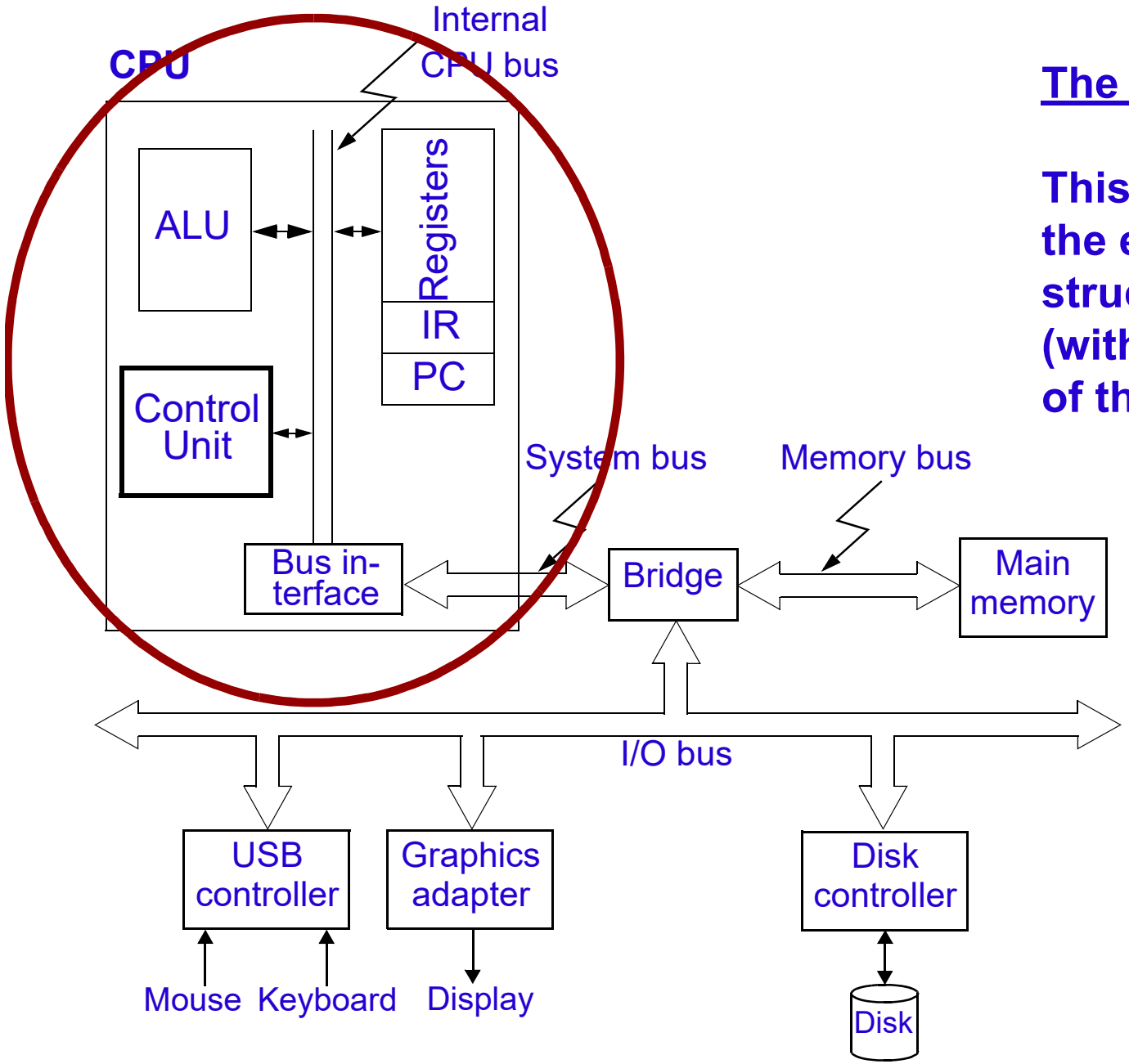
- **Besides the "core" we also have the peripherals. Computer peripherals include input devices, output devices, and secondary memories. That makes the whole to a *Computer System*.**



Computer Systems



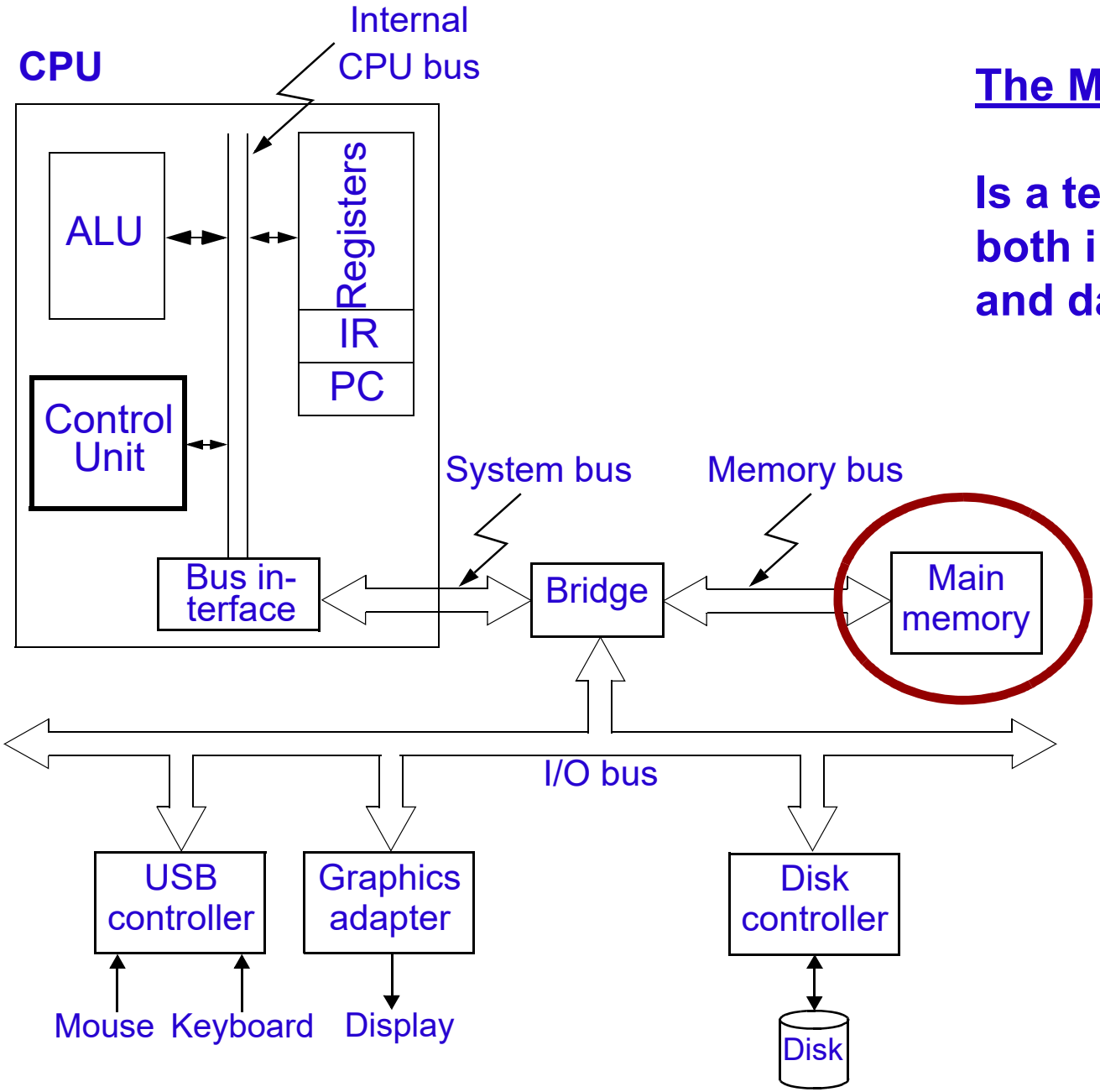
Computer Systems



The CPU (Central Processing Unit)

This is the hart of the system; it is the engine that interprets the instructions and executes them (with the help of other components of the computer system).

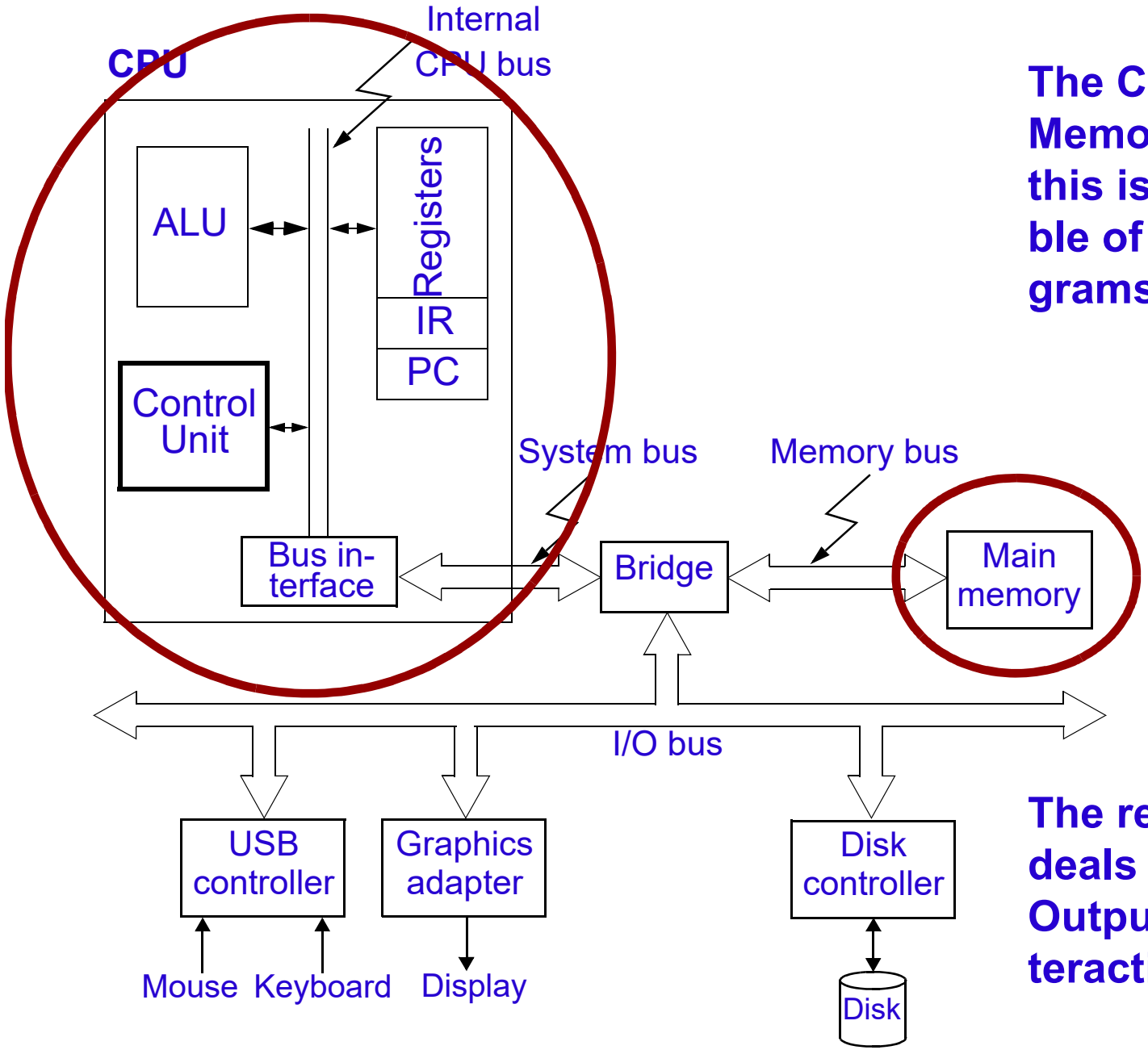
Computer Systems



The Main Memory

Is a temporary storage that stores both instructions (the program) and data.

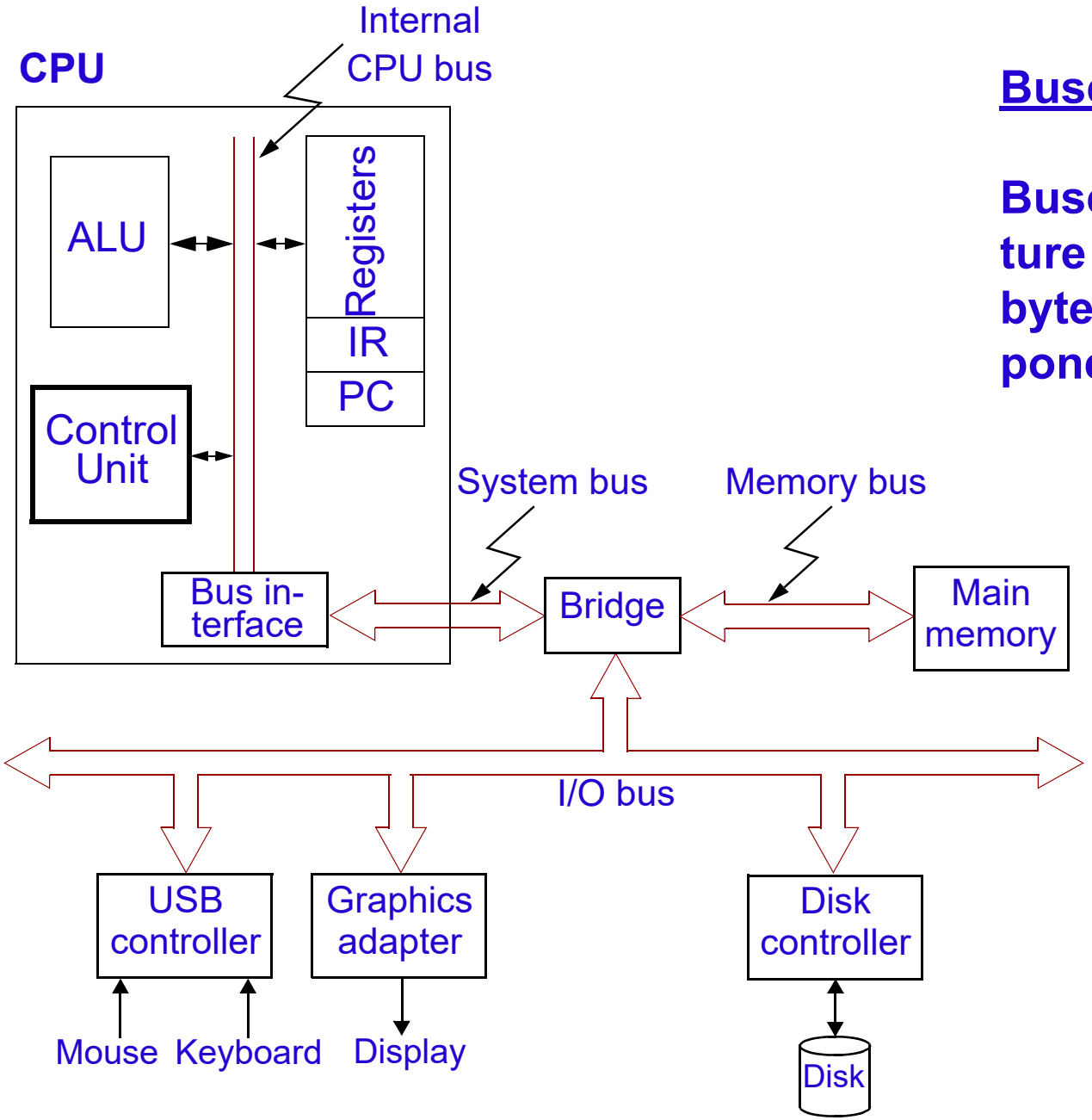
Computer Systems



The CPU together with the Main Memory build the core computer; this is the minimal structure capable of storing and executing programs.

The rest of the computer system deals with communication, Input/ Output, long term storage, and interaction with the environment.

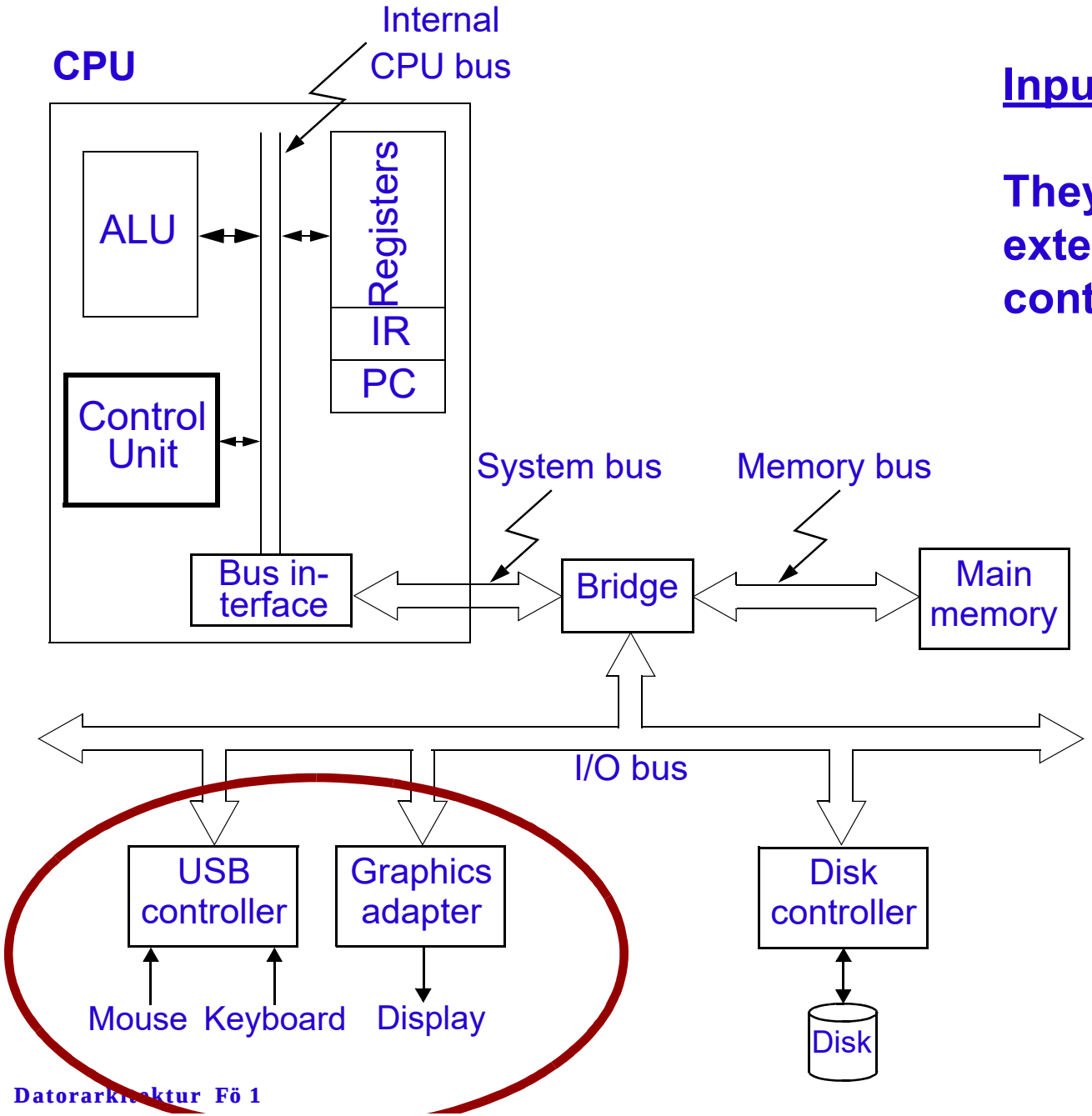
Computer Systems



Buses

Buses are the physical infrastructure (electrical wiring) over which bytes are travelling between components of the computer system.

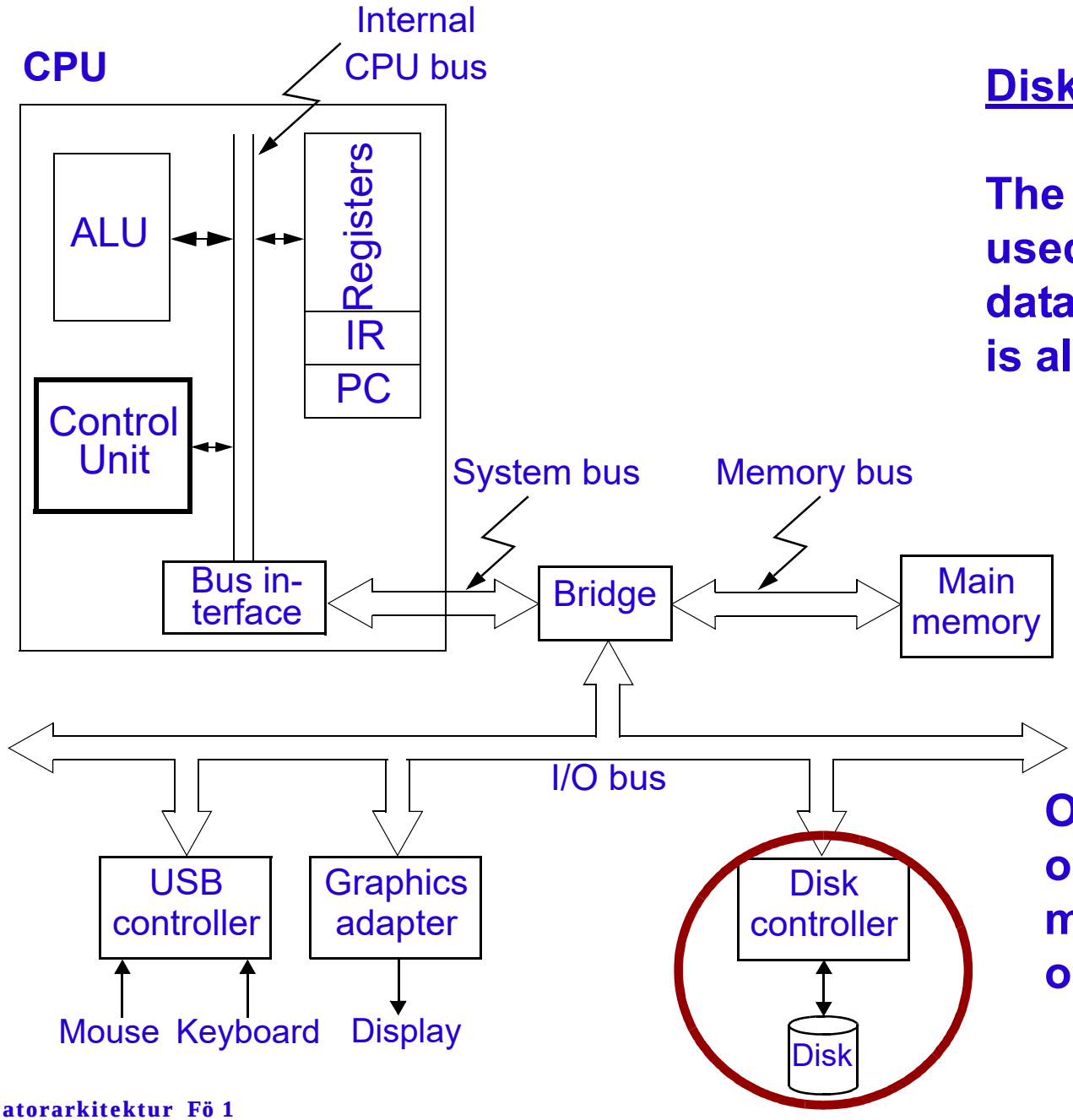
Computer Systems



Input/Output Devices

They connect the computer to the external world. Connection is via controllers/adaptors.

Computer Systems



Disk drive

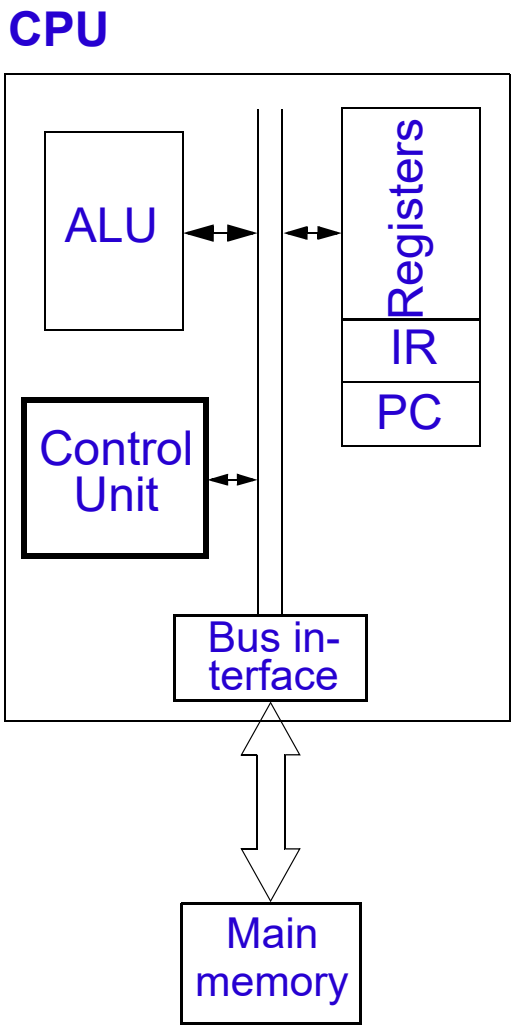
The disk drive is a special device used as a long term storage for data and programs. Such a storage is also called *Secondary Memory*.

On modern computers the secondary memory is often implemented as *solid state disk (SSD)* on *flash memory*.

How Does It Work? The von Neumann Architecture

- All computers in use, simple or complicated, big or small, cheap or expensive work according to the same basic concept, known as the von Neumann architecture:
  - Data and instructions are both stored in the main memory (stored program concept);
  - The content of the memory is addressable by location (without regard to what is stored in that location);
  - Instructions are executed sequentially (from one instruction to the next, in order of their location in memory) unless the order is explicitly modified.

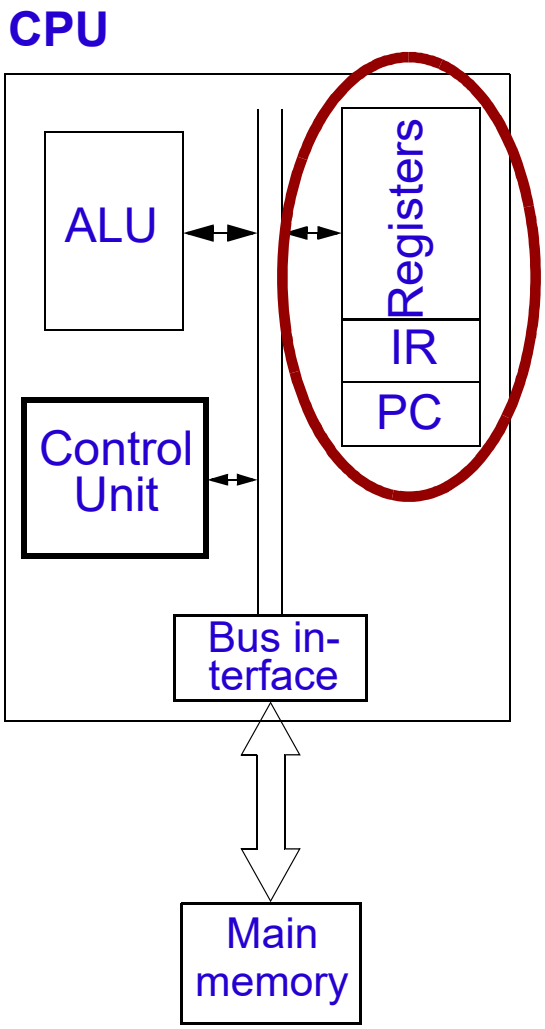
A Simple Computer Architecture



The basic von Neumann organization (architecture):

- ***Central processing unit*** **(CPU) contains:**
  - ***Control unit*** **(CU) that coordinates the execution of instructions;**
  - ***Arithmetic/logic unit*** **(ALU) that performs arithmetic and logic operations;**
  - **A set of *registers.***
- ***Main memory*****.**

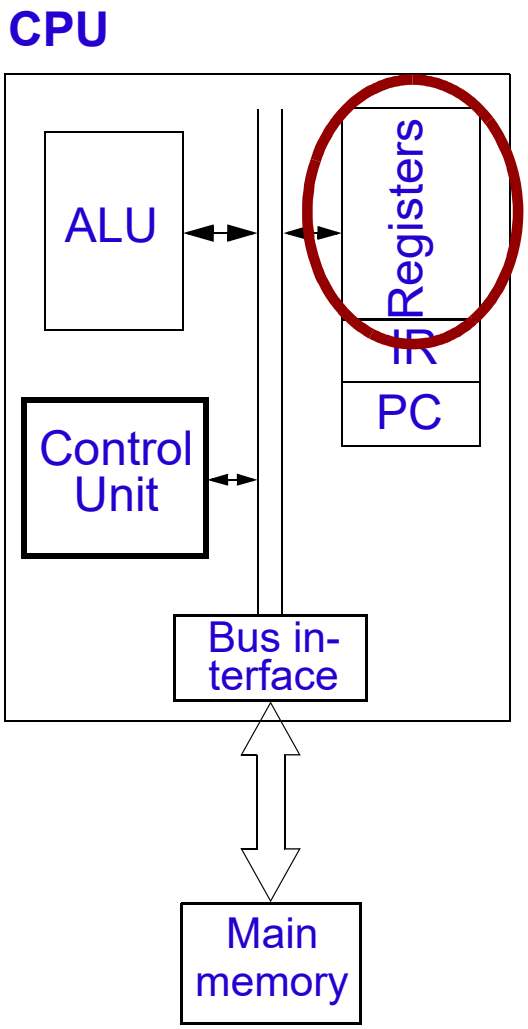
A Simple Computer Architecture



Register Organization

- The set of registers within the CPU represents the top level of the memory hierarchy inside the computer system:
  - User visible registers: can be accessed by programs, for data storing.
  - Control and Status registers: used by the Control Unit to control the operation of the CPU; not directly accessible by the programmer.

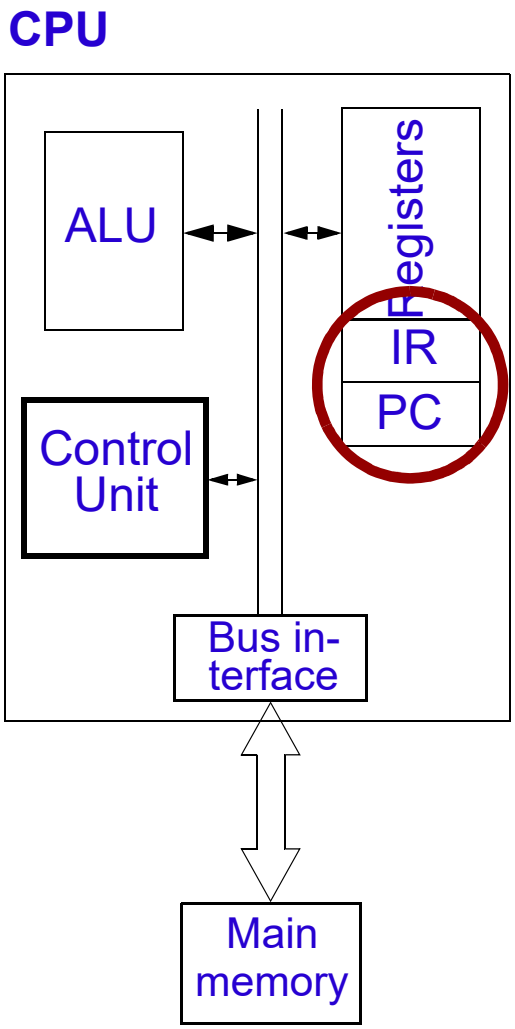
A Simple Computer Architecture



User Visible Registers

- **A set of registers which can be used without restrictions as operands for any operation and as address registers; these are so called *general-purpose registers*.**

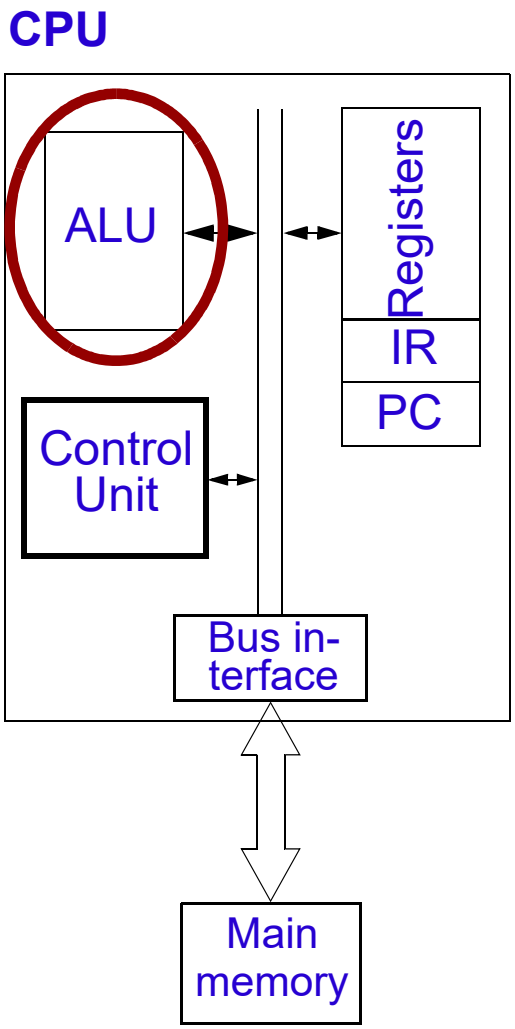
A Simple Computer Architecture



Control and Status Registers

- Program Counter (PC): holds the address of the instruction to be fetched and executed.
- Instruction Register (IR): holds the last instruction fetched.
- Program Status Word (PSW): Condition Code Flags + other bits defining the status of the CPU.
- . . . . . .

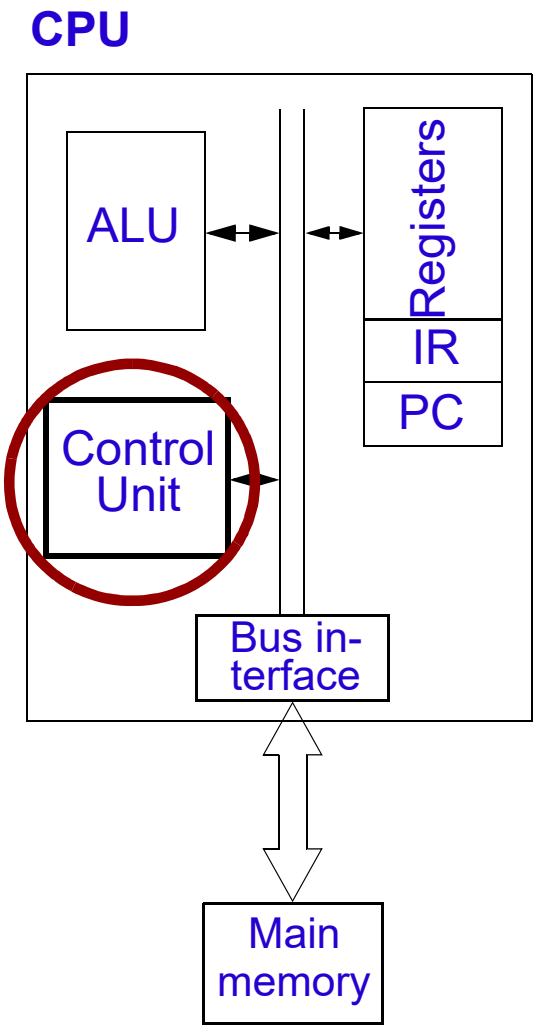
A Simple Computer Architecture



Arithmetic Logic Unit (ALU)

- Performs arithmetic and logic operations. There might be several of them in a CPU. ALUs are different, depending on the data type they operate on: integer ALU, floating point ALU, etc.

A Simple Computer Architecture



Arithmetic Logic Unit (ALU)

- Performs arithmetic and logic operations. There might be several of them in a CPU. ALUs are different, depending on the data type they operate on: integer ALU, floating point ALU, etc.

Control Unit

- The control unit generates the appropriate signals such that all other components of the CPU and the computer system, together, execute the current instruction.
- The current instruction to execute is stored in the instruction register (IR); it is the instruction whose memory address is stored in the program counter (PC)

Representation of Data

- Inside a computer, data and control information (instructions) are all represented in binary format which uses only two basic symbols: "0" and "1".

- The two basic symbols are represented by electronics signals.

- Numeric data are represented using the binary system, in which the positional values are powers of 2:

  $100101 = 1*2^0 + 0*2^1 + 1*2^2 + 0*2^3 + 0*2^4 + 1*2^5$

  $10110 = 0*2^0 + 1*2^1 + 1*2^2 + 0*2^3 + 1*2^4$

- Binary numbers are added, subtracted, multiplied and divided (by the ALU) directly; it is not needed to convert them to decimal numbers first.

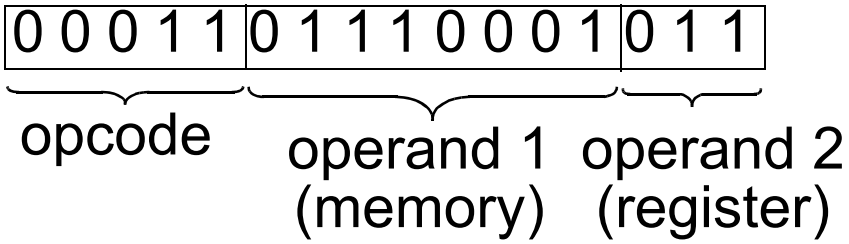
  100101 + 10110 = 111011

Machine Instructions

- A CPU can only execute *machine instructions*,
- Each computer has a set of specific machine instructions which its CPU is able to recognize and execute.

Machine Instructions

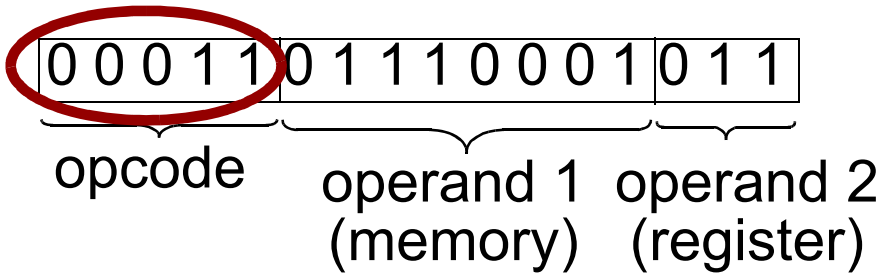
- A CPU can only execute *machine instructions*,
- Each computer has a set of specific machine instructions which its CPU is able to recognize and execute.



- A machine instruction is represented as a sequence of bits (binary digits). These bits are organized into *fields* that define:

Machine Instructions

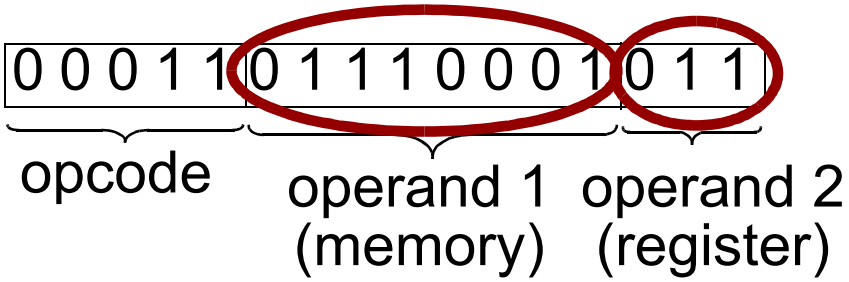
- A CPU can only execute *machine instructions*,
- Each computer has a set of specific machine instructions which its CPU is able to recognize and execute.



- A machine instruction is represented as a sequence of bits (binary digits). These bits are organized into *fields* that define:
  - What has to be done (the operation code).

Machine Instructions

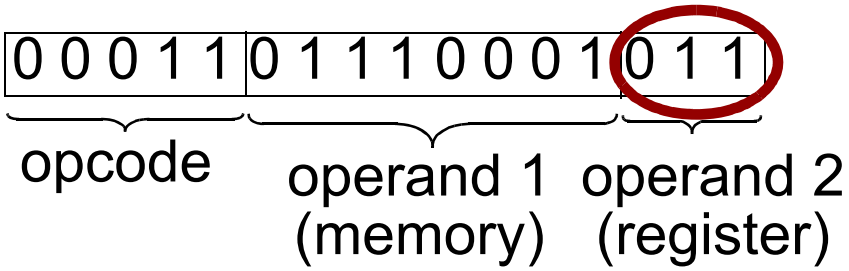
- A CPU can only execute *machine instructions*,
- Each computer has a set of specific machine instructions which its CPU is able to recognize and execute.



- A machine instruction is represented as a sequence of bits (binary digits). These bits are organized into *fields* that define:
  - What has to be done (the operation code).
  - To whom the operation applies (source operands).

Machine Instructions

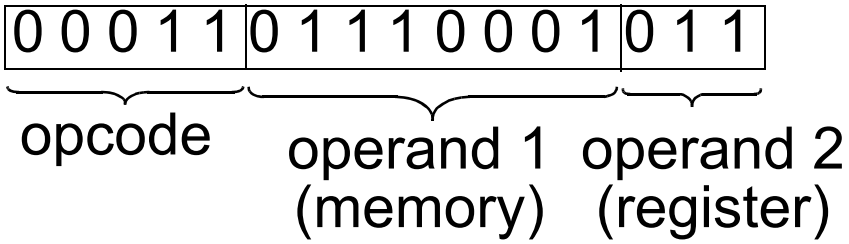
- A CPU can only execute *machine instructions*,
- Each computer has a set of specific machine instructions which its CPU is able to recognize and execute.



- A machine instruction is represented as a sequence of bits (binary digits). These bits are organized into *fields* that define:
  - What has to be done (the operation code).
  - To whom the operation applies (source operands).
  - Where does the result go (destination operand); in this example CPU it is assumed that the result of the operation is stored in the same place where the second operand was stored; no additional field is needed.

Machine Instructions

- A CPU can only execute *machine instructions*,
- Each computer has a set of specific machine instructions which its CPU is able to recognize and execute.



- Number of bits, number and length of the fields and their order is particular to each computer; this defines the *instruction format* of that computer.

Types of Machine Instructions

- Machine instructions are of four types:
  - Data transfer between memory and CPU registers
  - Arithmetic and logic operations
  - Program control (test and branch); these are those instructions that change the flow of instruction execution by *jumping* to an instruction *different* from the instruction following the current one in memory.
  - I/O transfer

You see, there are very simple things a machine instruction does!
But many machine instructions, together, perform the big thing!

Instruction Execution

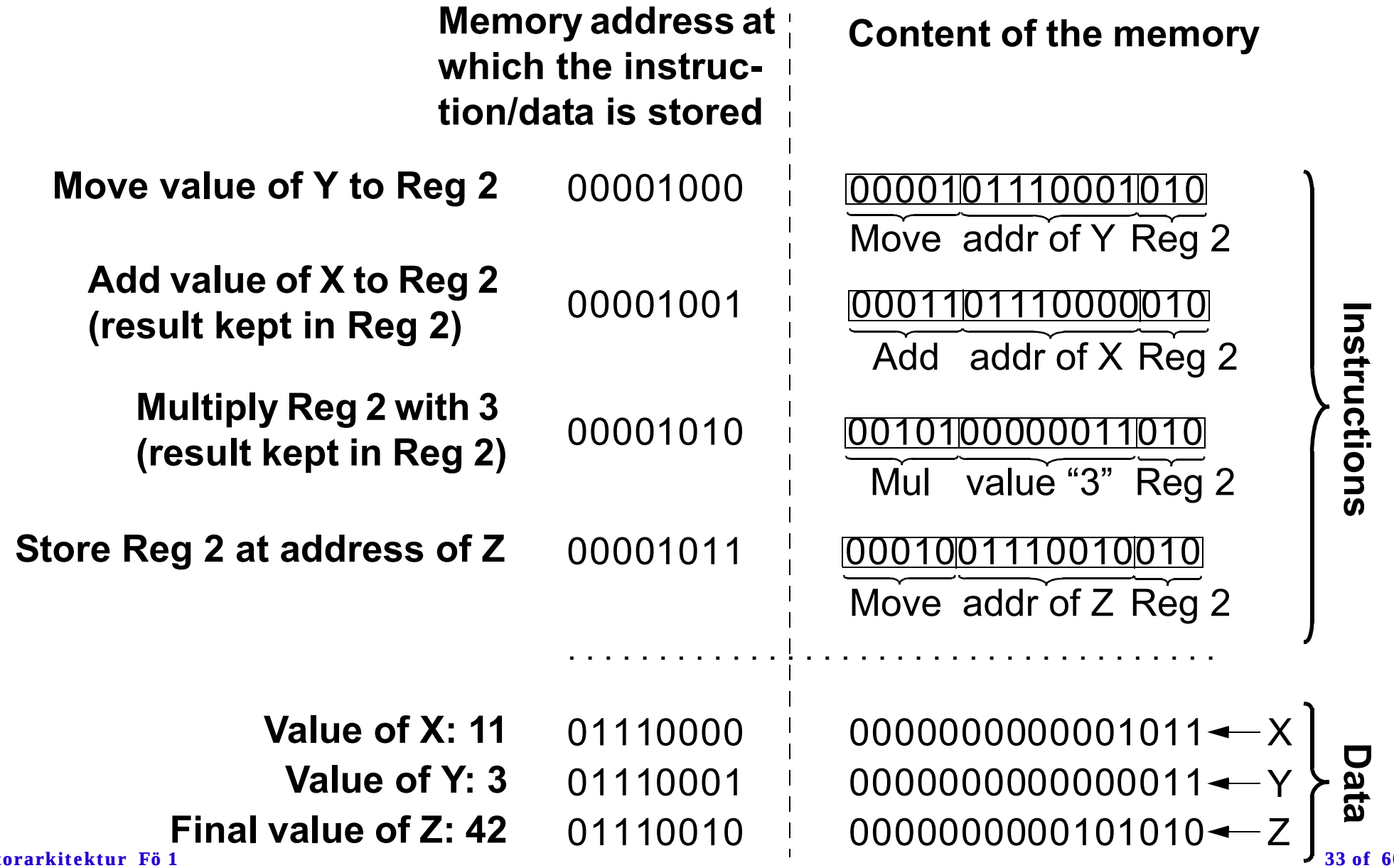
Let's imagine you write in a program the following instruction:

Z := (Y + X) * 3;

The instruction will be executed by the CPU as a sequence of four machine instructions!

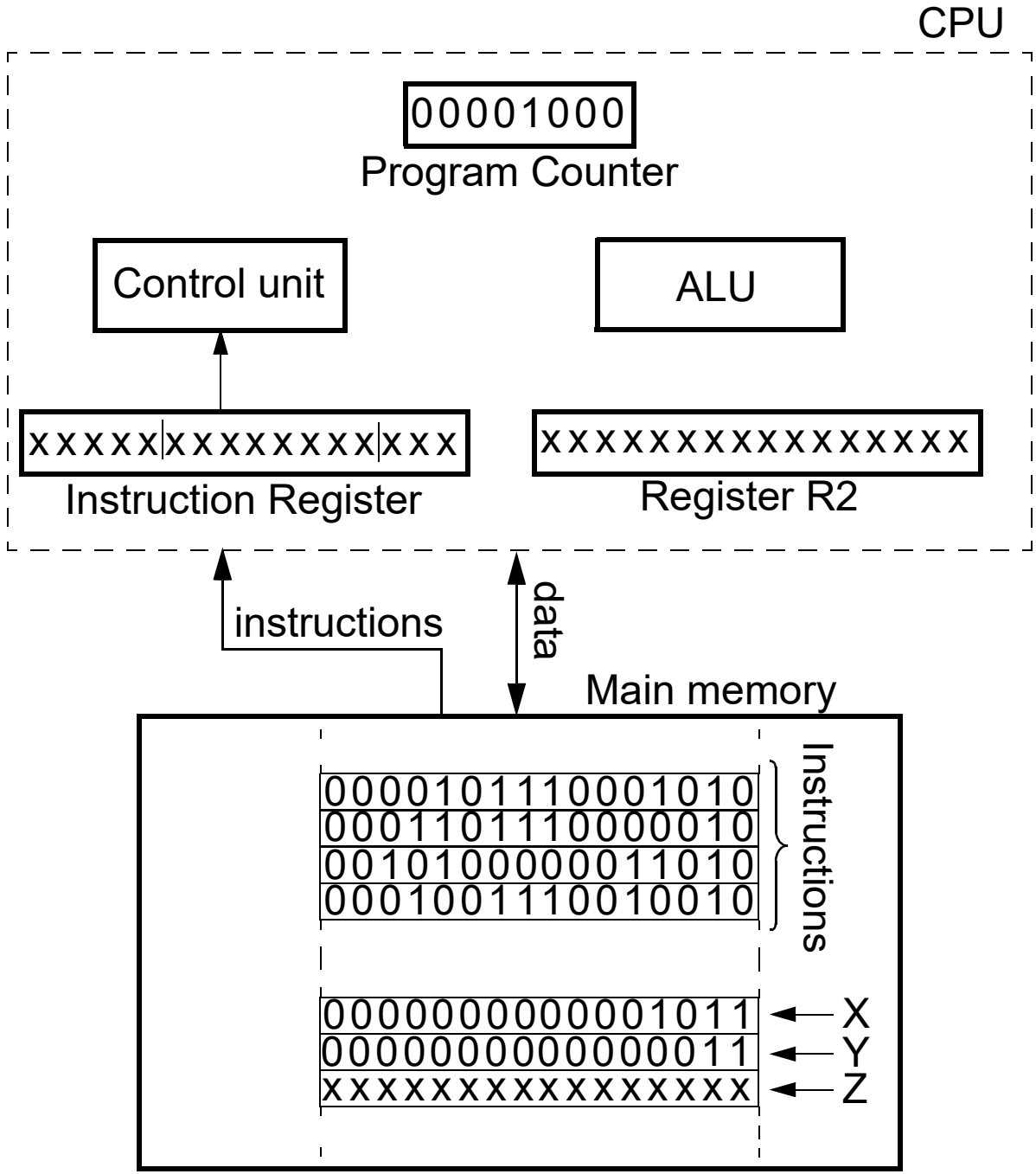
Instruction Execution

Let's imagine you write in a program the following instruction: Z := (Y + X) * 3;



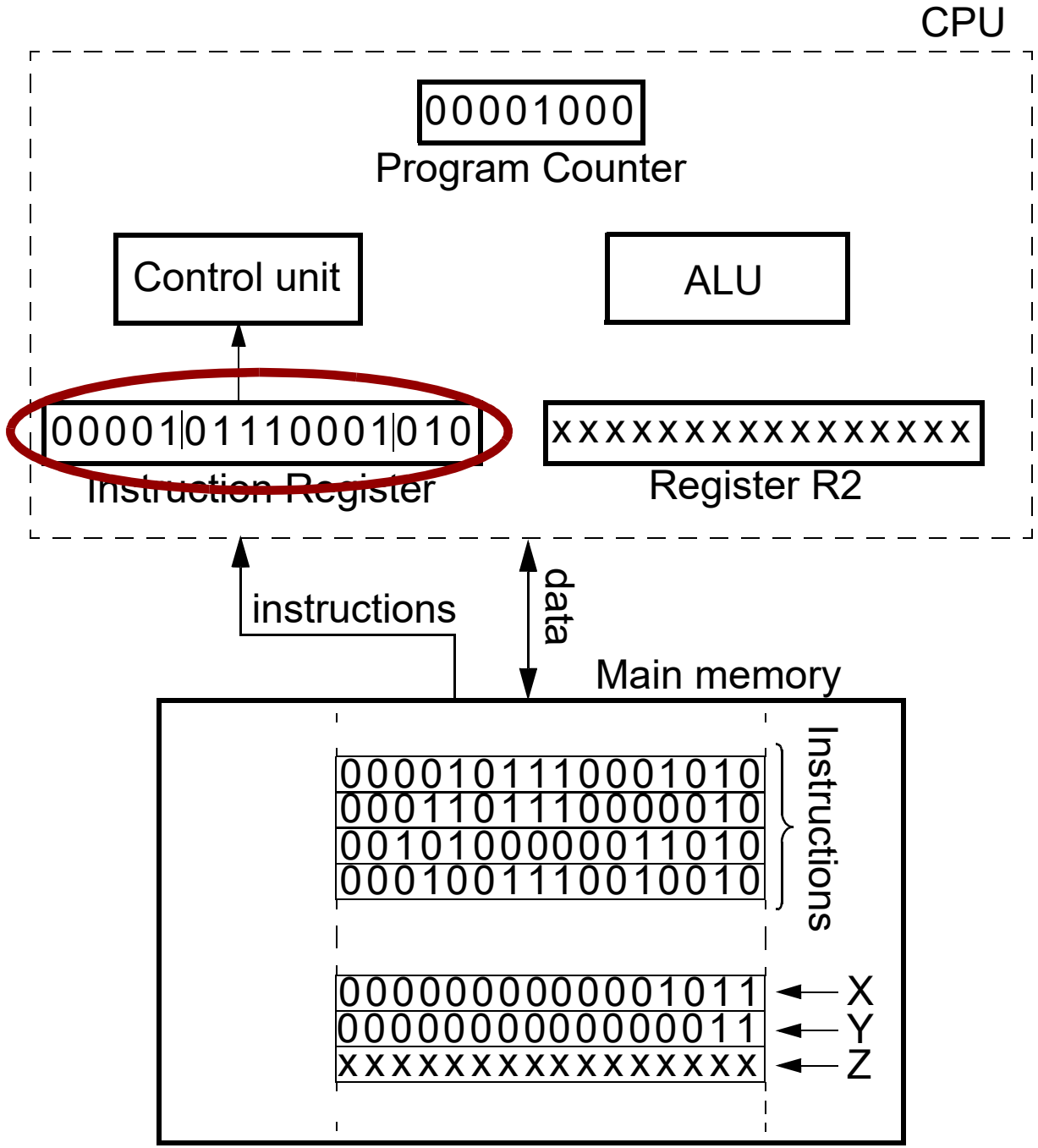
Let's Follow the Instruction Execution

Before the first instruction



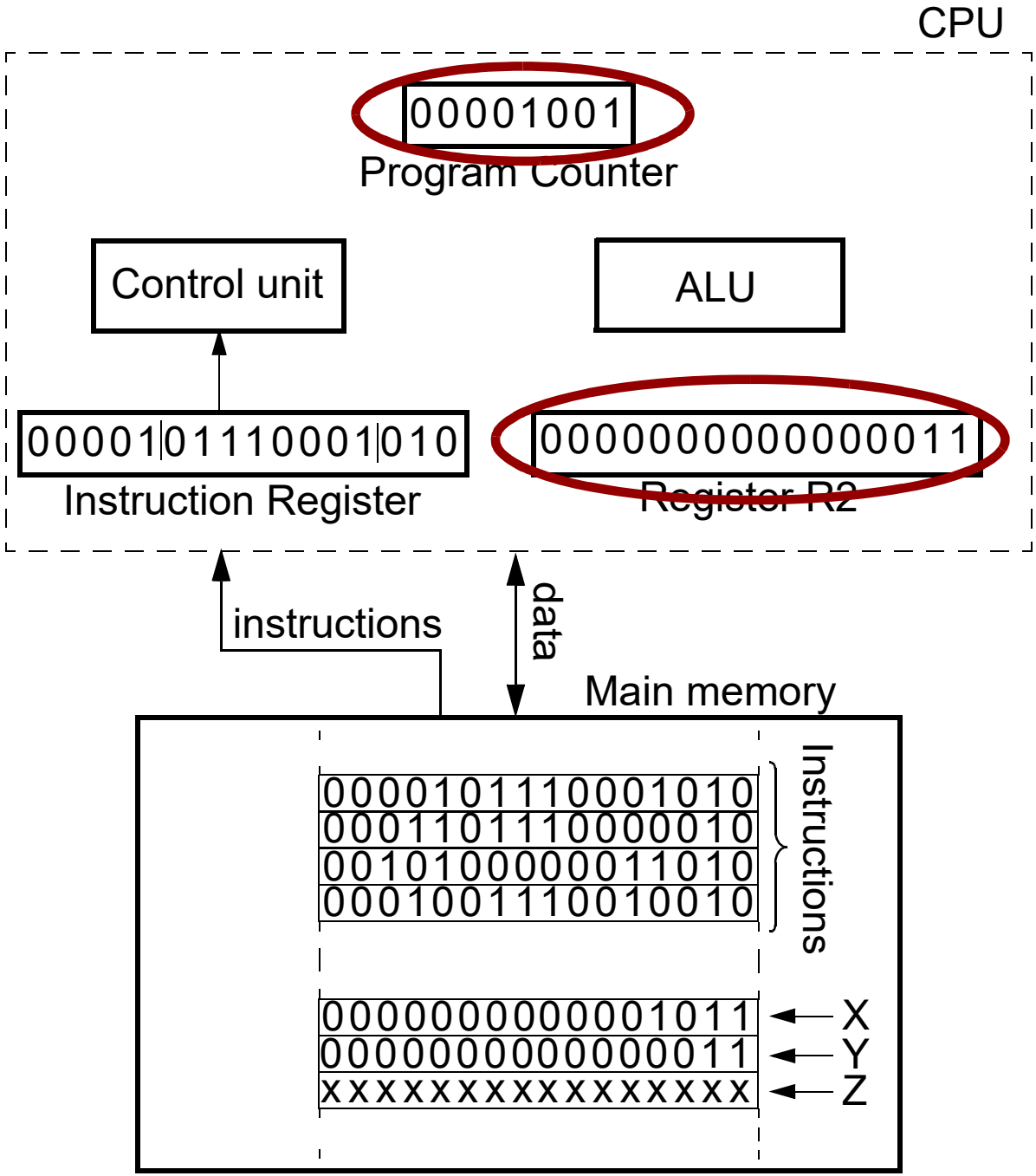
Let's Follow the Instruction Execution

Now the first instruction is fetched



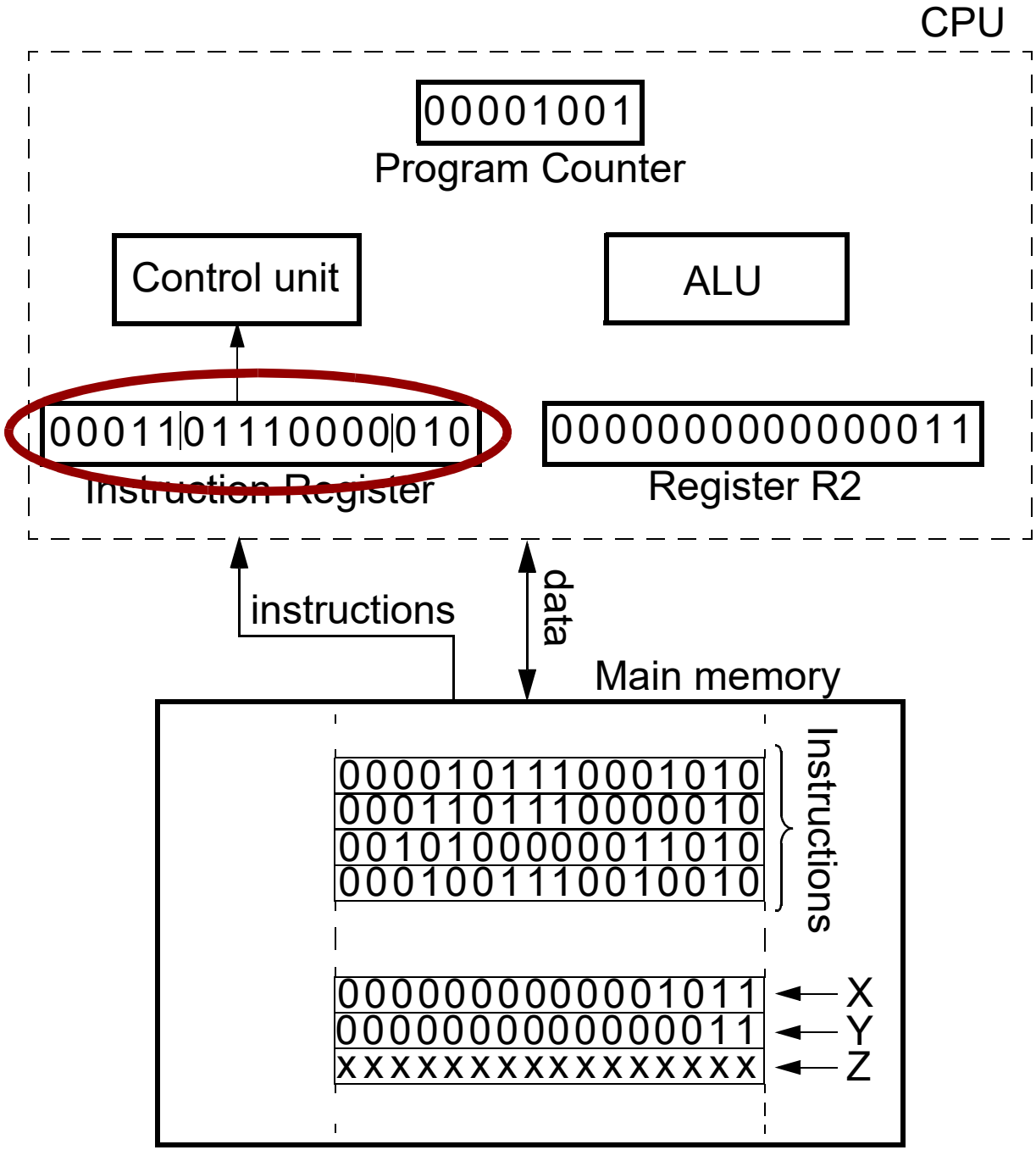
Let's Follow the Instruction Execution

After the first instruction



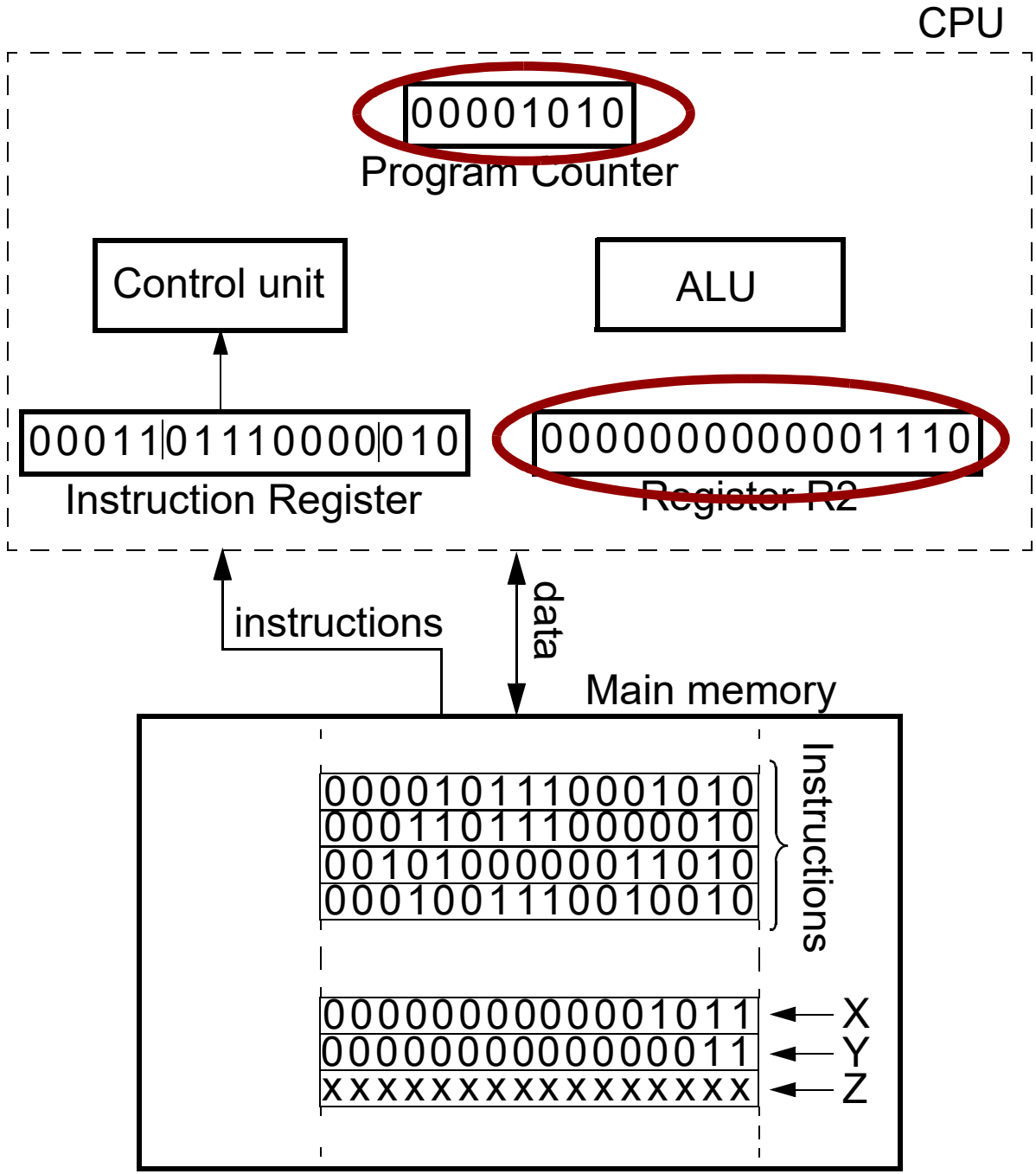
Let's Follow the Instruction Execution

Now the second instruction is fetched



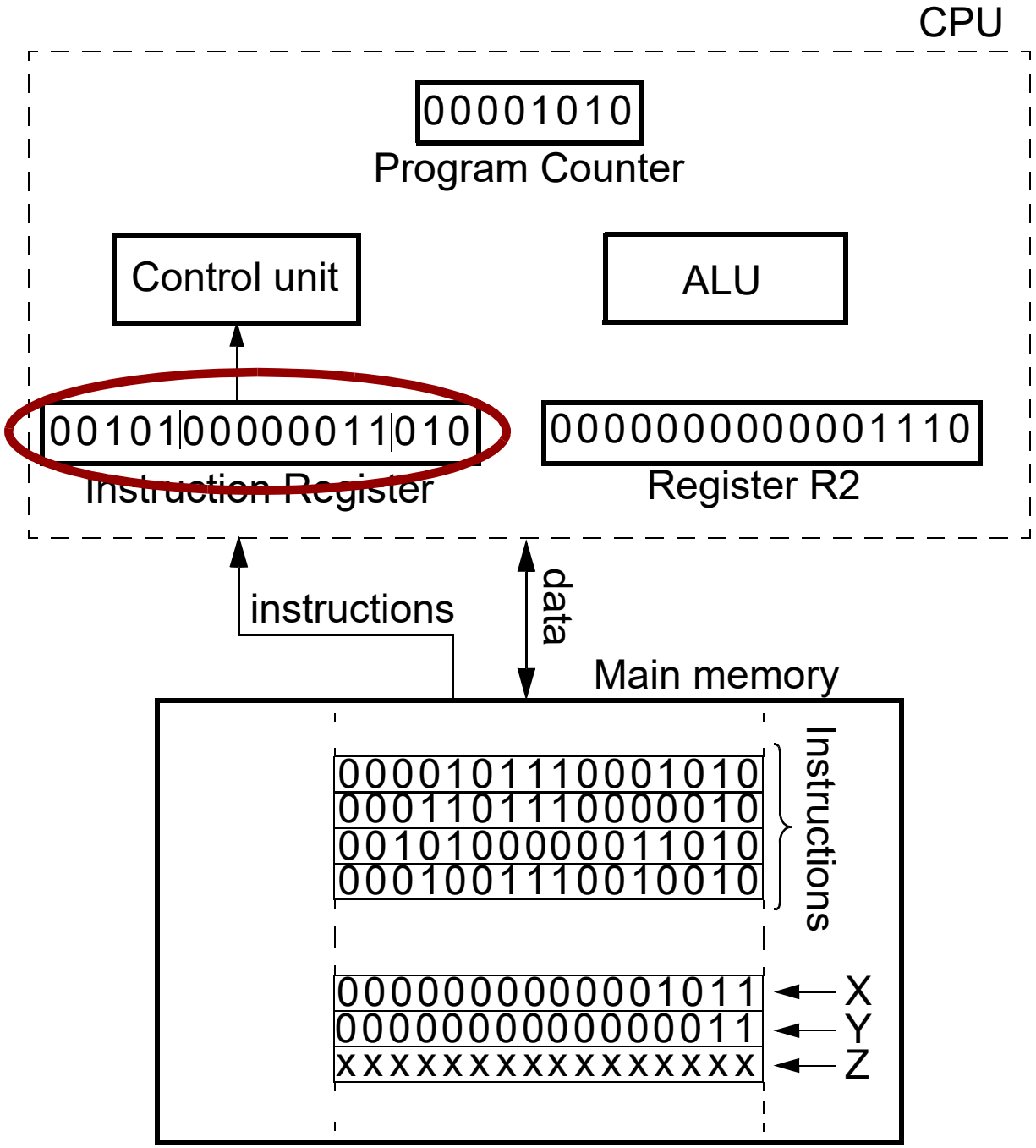
Let's Follow the Instruction Execution

After the sec-ond instruction



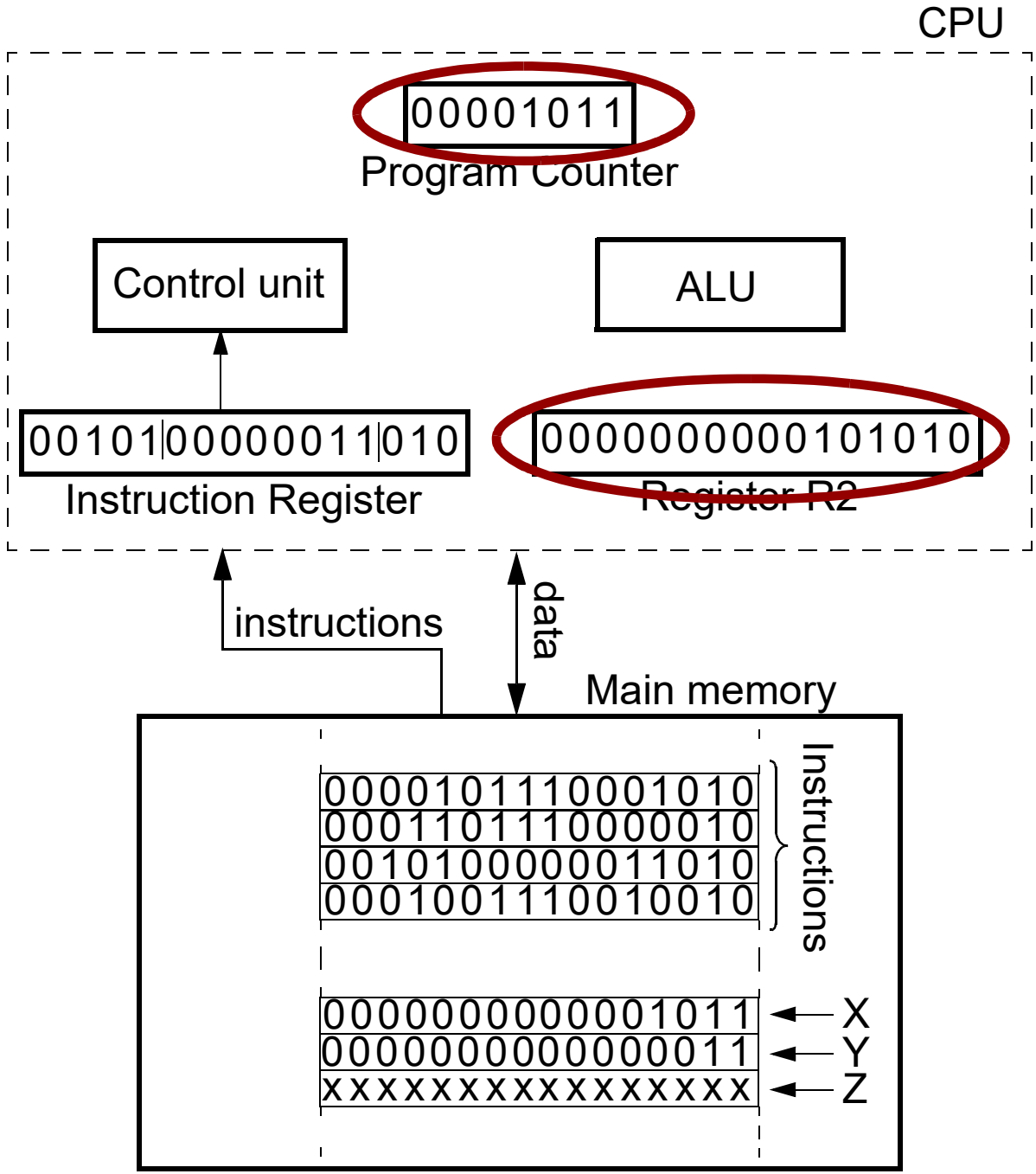
Let's Follow the Instruction Execution

Now the third instruction is fetched



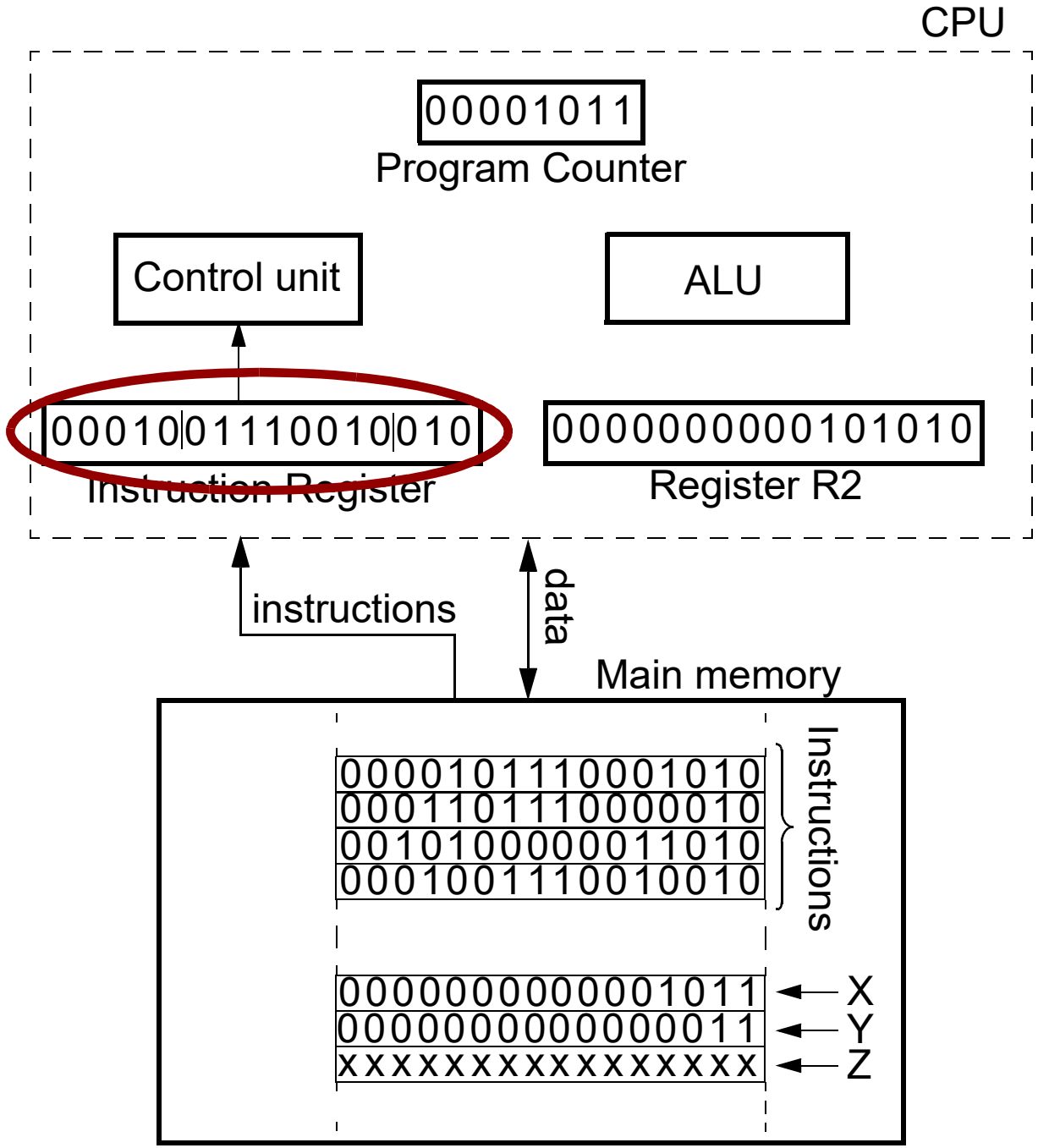
Let's Follow the Instruction Execution

After the third instruction



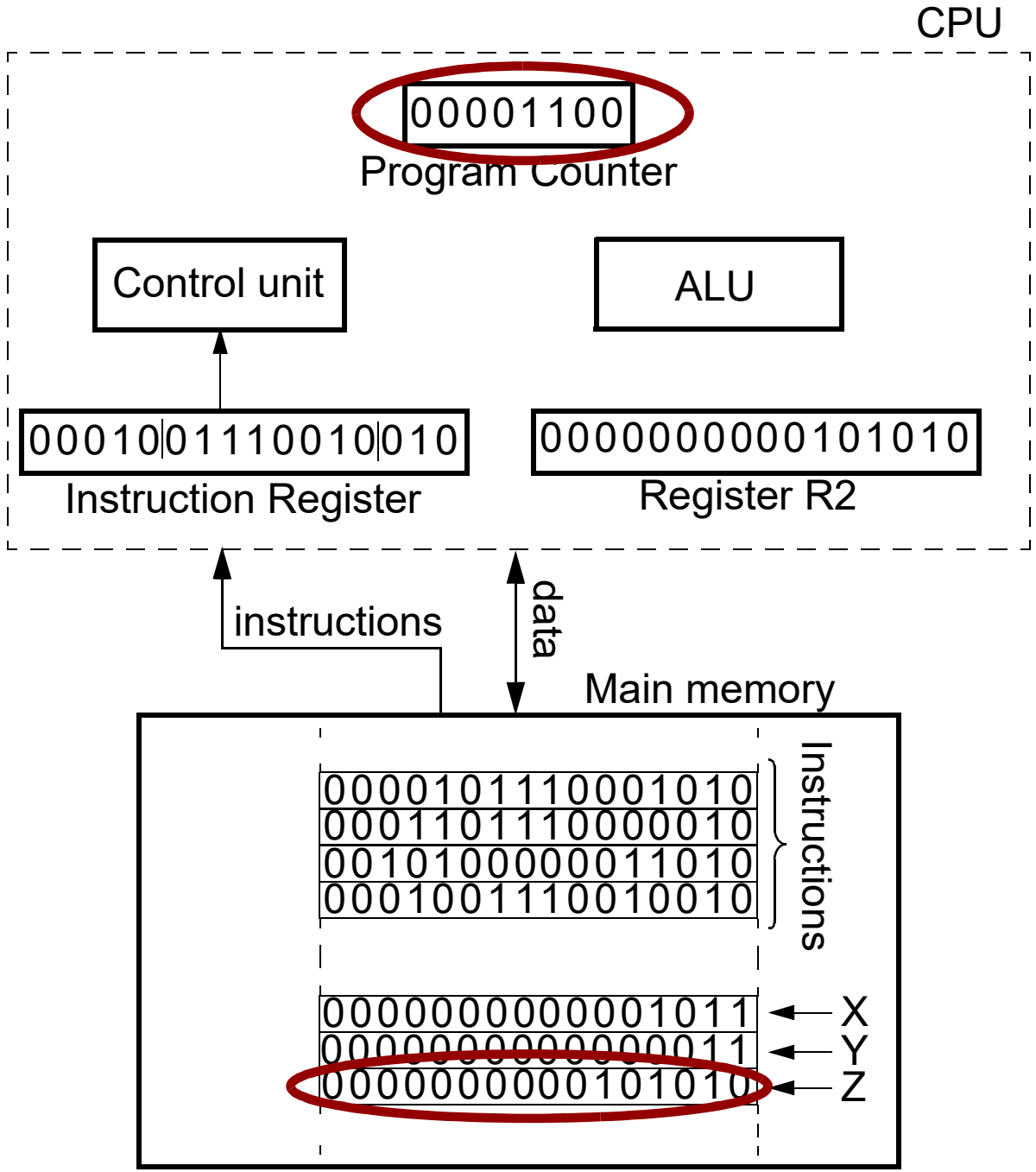
Let's Follow the Instruction Execution

Now the fourth instruction is fetched



Let's Follow the Instruction Execution

After the fourth and last instruction



Compilers

We have written in our program: Z := (Y + X) * 3;

High Level Language (e.g. C, C++, Java)

What the computer executes is:

```
0000101110001010
0001101110000010
0010100000011010
0001001110010010
```

Machine instructions for the particular processor that runs the program.

Compilers

We have written in our program: `Z := (Y + X) * 3;`

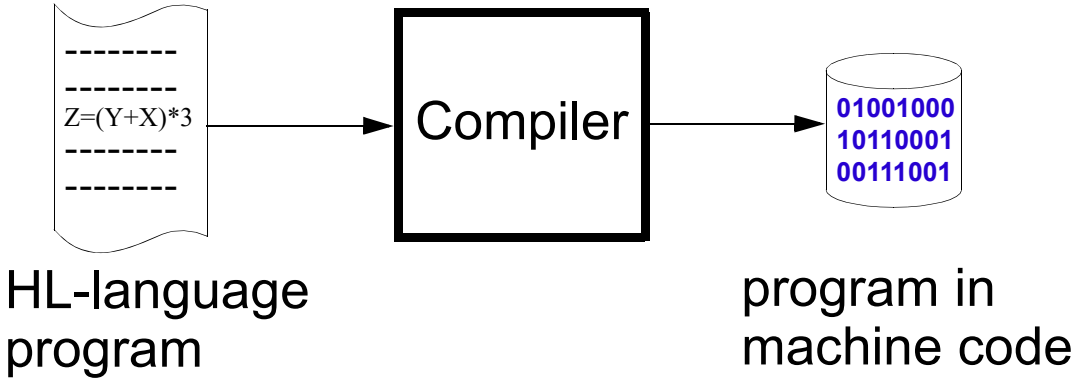
High Level Language (e.g. C, C++, Java)

What the computer executes is:

```
0000101110001010
0001101110000010
0010100000011010
0001001110010010
```

Machine instructions for the particular processor that runs the program.

Who brings us from our program to the machine instructions?



- **A *compiler* is a program that translates programs written in a high level language into machine code to be executed on a certain processor.**

The Machine Cycle

Many things have to be done to execute a simple machine instruction:

- **Fetch instruction**
- **Decode instruction**
- **Execute instruction**

The Machine Cycle

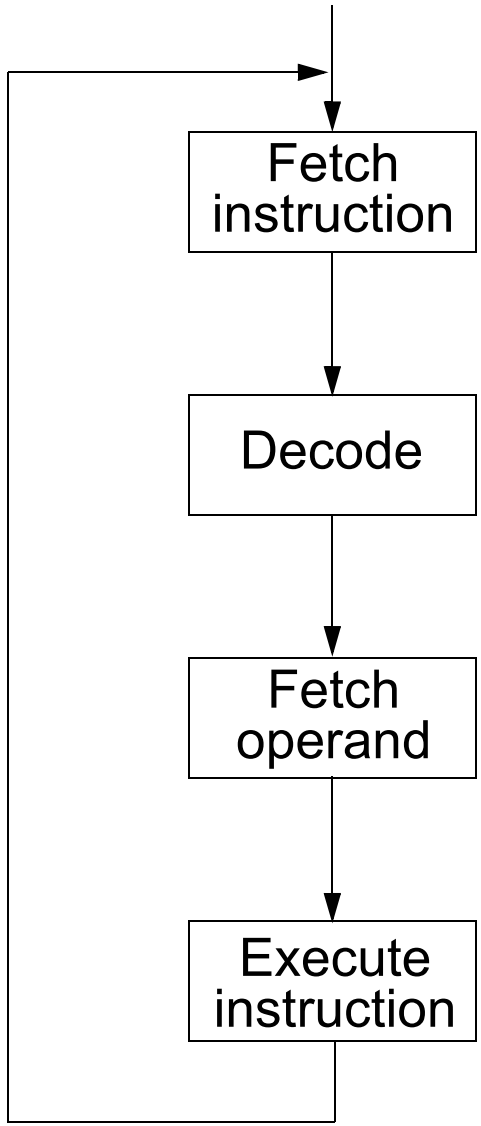
Many things have to be done to execute a simple machine instruction:

- **Fetch instruction**
- **Decode instruction**
- **Execute instruction**
  - **Fetch operand(s)**
  - **Execute instruction**

The Machine Cycle

Many things have to be done to execute a simple machine instruction:

- **Fetch instruction**
- **Decode instruction**
- **Execute instruction**
  - **Fetch operand(s)**
  - **Execute instruction**



Machine Cycle

Each instruction is performed as a sequence of steps; the steps corresponding to the execution of one instruction are referred together as a *machine cycle*.

The number and nature of steps in the machine cycle differ from processor to processor.

Case Studies

- INTEL x86 Family
  - The most successful example of a modern CISC (complex instruction set computer) microprocessor
- ARM Family
  - The most successful RISC (reduced instruction set computer) microprocessor ever.

The INTEL x86

The number one microprocessor used in non-embedded systems.

First member:

- **INTEL 4004 in 1971**

First general purpose microprocessor:

- **INTEL 8080 in 1974**

Milestones: INTEL x86

- 8080 (1974)
  - First general purpose microprocessor;
  - 8 bits;
  - Used in first personal computer: *Altair*.
- 8086 (1978)
  - 16 bits
  - Something like an instruction cache
  - First one used in IBM PC
- 80286 (1982)
  - Huge increase in addressable memory (16 MByte instead of 1 MByte)
- 80386 (1985)
  - 32 bits
- 80486 (1989)
  - Complex cache structure and pipelining
  - Math coprocessor

Milestones: INTEL x86

- Pentium (1993)
  - Introduces superscalar technology
- Pentium Pro (1995)
  - Advanced superscalar techniques
  - Branch prediction and speculative execution
- Pentium II (1997)
  - Intel MMX technology (instruction set extension for multimedia)
- Pentium III (1999)
  - Additional floating point instructions
  - Support for 3D graphics software
- Pentium 4 (2000)
  - Further improvements on the line of Pentium III
- Core (2006)
  - Solo: single core
  - Duo: Dual core - two processors on a chip

Milestones: INTEL x86

- Core 2 (2006)
  - 64 bits
  - Dual: two processors on a chip
- Core 2 Quad (2007)
  - Four processors on a chip
- Core i7 (2009/2010), mobile version 2011
  - Six processors, Hyperthreading
- Core i9 (2017)
  - 10-18 processors, Hyperthreading

Backward compatibility: newer versions can run the programs running on older versions.

The ARM Family

ARM processors are widely used in embedded systems (games, phones, multimedia applications, various hand-held devices, consumer products, automotive, medical equipment, wireless etc.).

- **High performance, low size/cost, low power consumption**

ARM Cambridge, UK, are designing single and multiprocessor architectures and *licence* them to manufacturers.

Milestones: The ARM Family

- ARM1 and ARM2 (1985)
  - 32 bit RISC processor
  - 3 stage pipeline
- ARM3 (1989)
  - cache memory
- ARM6 (1992)
  - Floating point unit
- ARM7 (1994)
  - Most successful family.
  - First used as part of complete Systems on Chip (SoC)
- ARM8 (1996)
  - 5 stage pipeline
- ARM9 (1997)
  - separate data/instruction cache

Milestones: The ARM Family

- StrongArm (1996)
  - Special version of ARM9; developed with DEC and, later, bought by Intel.
- ARM10 (2000)
  - 6 stage pipeline
- ARM11 (2002)
  - 9 stage pipeline
  - Media processing extensions
- XScale (2002)
  - Successor to StrongArm, by Intel
  - 7/8 stage pipeline
  - Dynamic voltage & frequency management
- Cortex (2005)
  - 13 stage pipeline
  - superscalar

Milestones: The ARM Family

- ARM11 MPCore (2005)
  - Multicore based on ARM11; up to 4 cores
- Cortex-A9 MPCore (2008)
  - Multicore based on out-of-order superscalar Cortex-A9 core; up to 4 cores
- Cortex- A15 MPCore (2011, commercial 2012)
  - Multicore based on an out-of-order superscalar Cortex-A15 core; 2 clusters, 4 cores each.
- ARM Cortex-A57 (2014, commercial 2016)
  - Out-of-order superscalar, 64-bit instruction set. Used in chips by *Qualcomm*, *Samsung*, *Nvidia*.

What is our Topic in this Course?

***We are interested in some advanced issues, typical to modern microprocessors and computer systems*. These advances are at the origin of *high performance achieved with today's computers*.**

What is our Topic in this Course?

***We are interested in some advanced issues, typical to modern microprocessors and computer systems*. These advances are at the origin of *high performance achieved with today's computers*.**

- **Memory hierarchy:**
  - **cache memory**
  - **virtual memory**
  - **memory management;**

What is our Topic in this Course?

***We are interested in some advanced issues, typical to modern microprocessors and computer systems*. These advances are at the origin of *high performance achieved with today's computers*.**

- **Memory hierarchy:**
  - **cache memory**
  - **virtual memory**
  - **memory management;**
- **Advanced CPU structures and instruction execution strategies:**
  - **pipelining**
  - **RISC architectures**
  - **superscalar architectures**
  - **VLIW architectures**

What is our Topic in this Course?

***We are interested in some advanced issues, typical to modern microprocessors and computer systems*. These advances are at the origin of *high performance achieved with today's computers*.**

- **Memory hierarchy:**
  - **cache memory**
  - **virtual memory**
  - **memory management;**
- **Advanced CPU structures and instruction execution strategies:**
  - **pipelining**
  - **RISC architectures**
  - **superscalar architectures**
  - **VLIW architectures**
- **System Architectures for parallel computing**
  - **performance of parallel computers**
  - **parallel computer architectures**
  - **multicore and multithreaded processors**
  - **general purpose graphic processors**