ESc 101: Fundamentals of Computing

Lecture 1

Dec 31, 2009

Instructor

- Who? Manindra Agrawal
- Where? Professor, Dept of CSE
- Why? It is interesting to teach a large class

Contents

The goals of the course are:

1. Learning to solve problems algorithmically
2. Learning to Convert algorithms to programs
3. Learn a programming language: C
4. And if time permits, another one: Mathematica

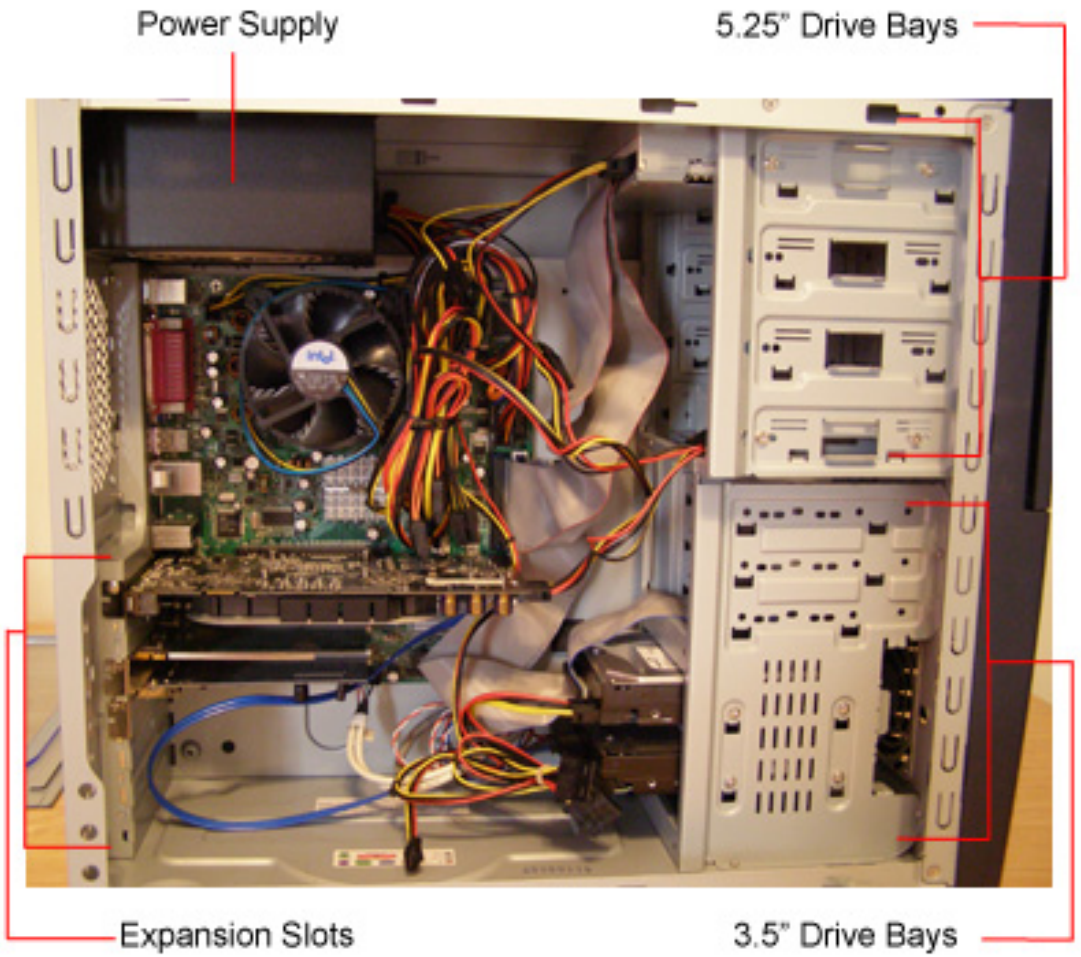
Books

- Reference Book: The C Programming Language, by Kernighan and Ritchie.
- Additional Reading: The UNIX Programming Environment, by Kernighan and Pike.

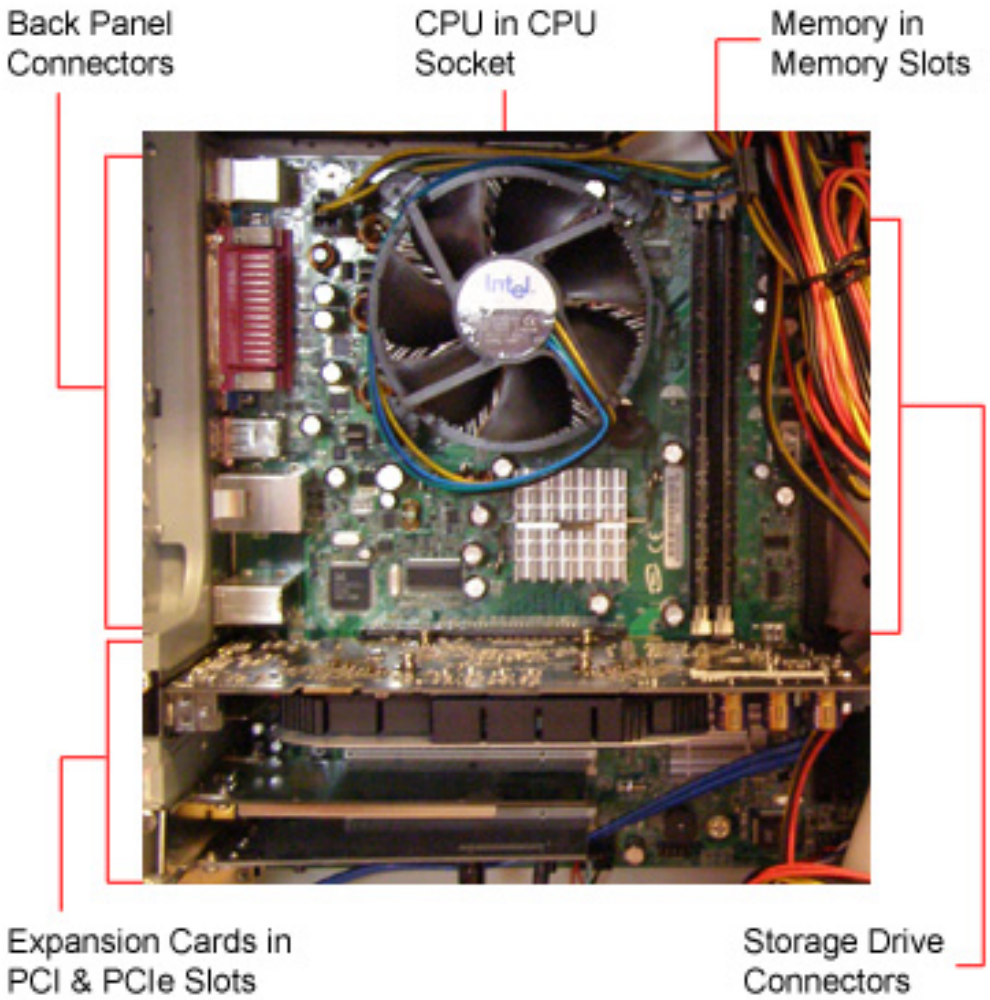
Computers

- What? A machine that can carry out any computational task.
- Really? It is formally proven that a computer, given sufficient memory and time, can carry our any computational task!
- How? For this, we look inside a typical computer.

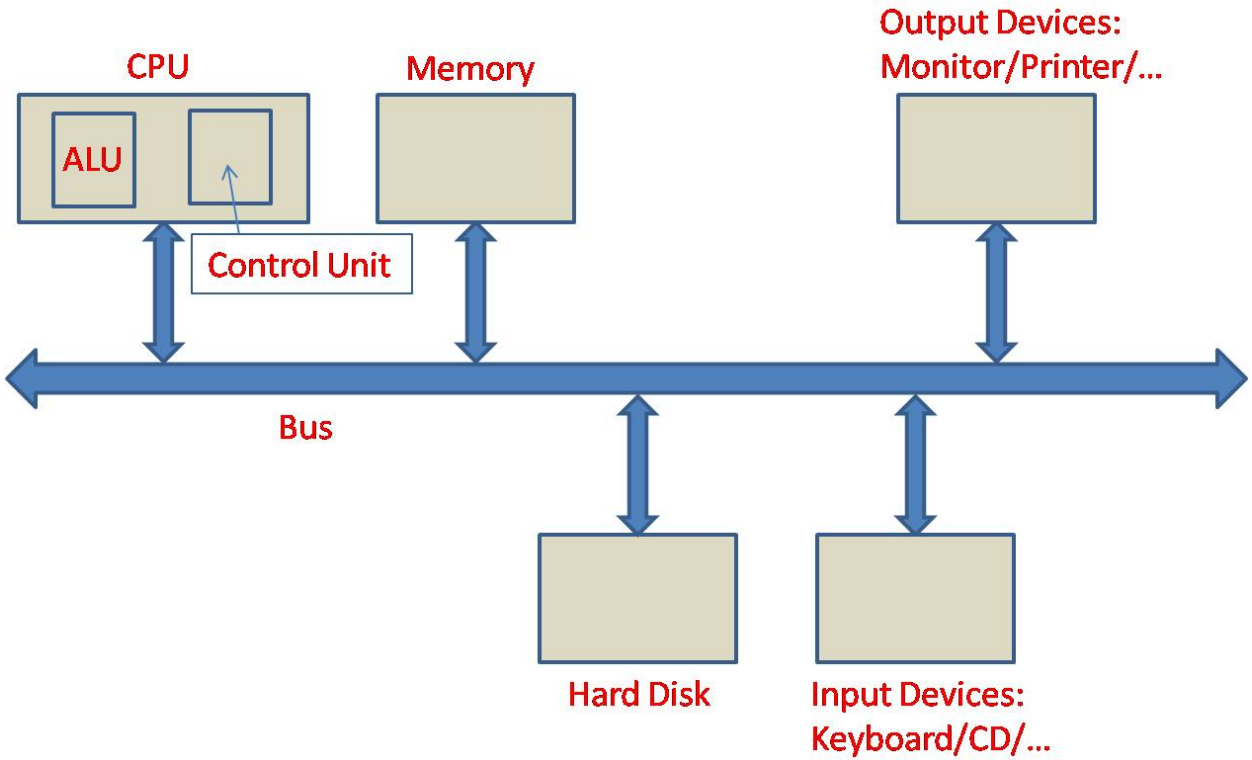
Inside a PC



The Motherboard



Functional Units



CPU: Central Processing Unit (ALU: Arithmetic and Logic Unit; Control Unit: Executes instructions)

Memory: Storage area; quickly accessible from CPU

Hard Disk: Storage area; not so quickly accessible from CPU

Binary Format

- In a computer, everything is stored in binary format: a sequence of 0's and 1's.
- The components of a computer understand only binary format.
- Number 4 is stored as 00000100, 1 is stored as 00000001 etc.

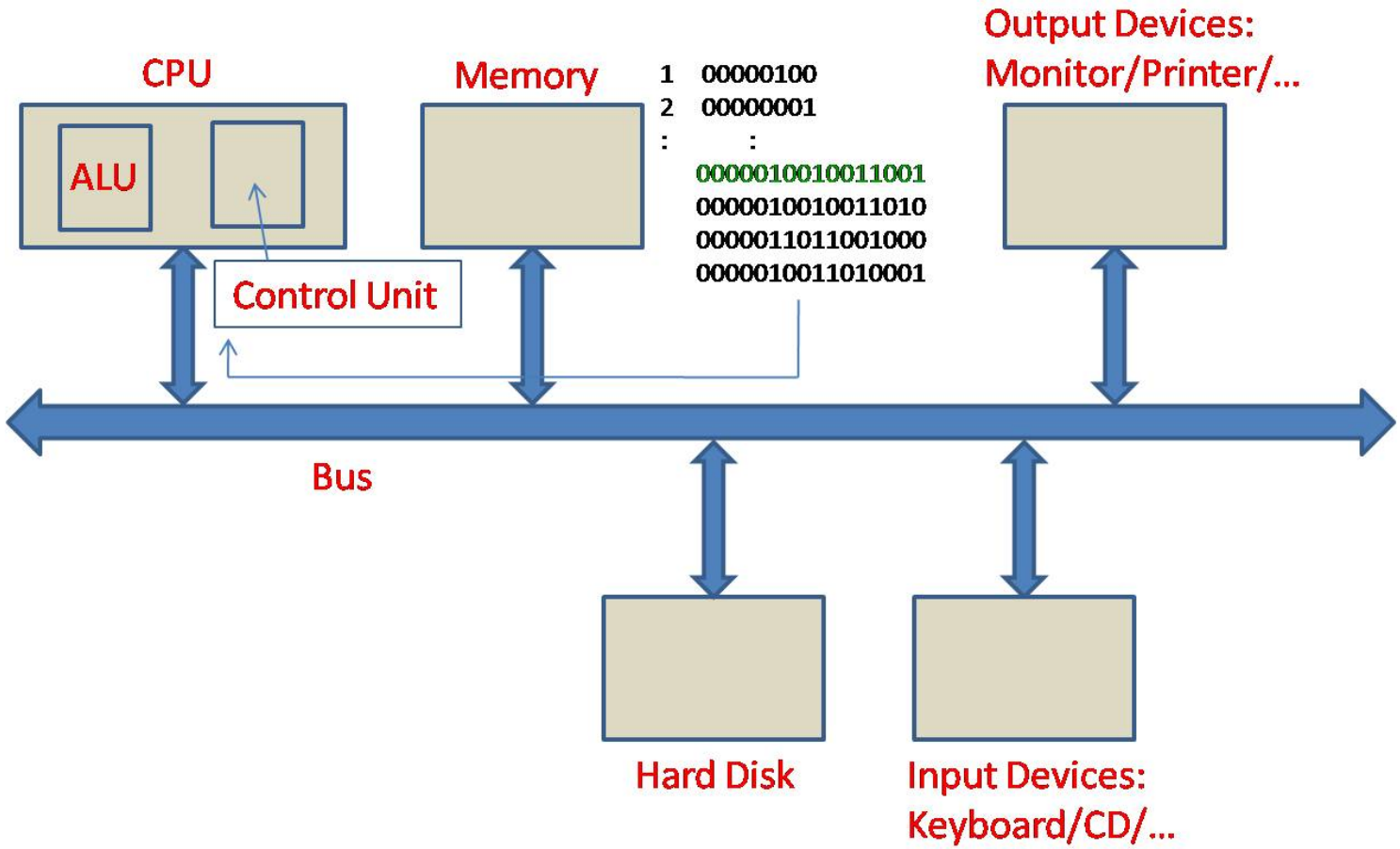
Execution in a Computer

- To begin with, all the data and commands related to a computation is stored in Memory.
- Commands are then brought into the CPU through the Bus, one at a time.
- Each command is executed inside the CPU in the following way:
  - If the command requires data, it is brought to CPU from Memory
  - Command is then executed using the data
  - The command may be for storing data present inside the CPU to Memory
- A program is a collection of commands.

A Small Program

0000010010011001 - read memory location 001
0000010010011010 - read memory location 010
0000011011001000 - add two numbers read
0000010011010001 - store the result in memory location 001

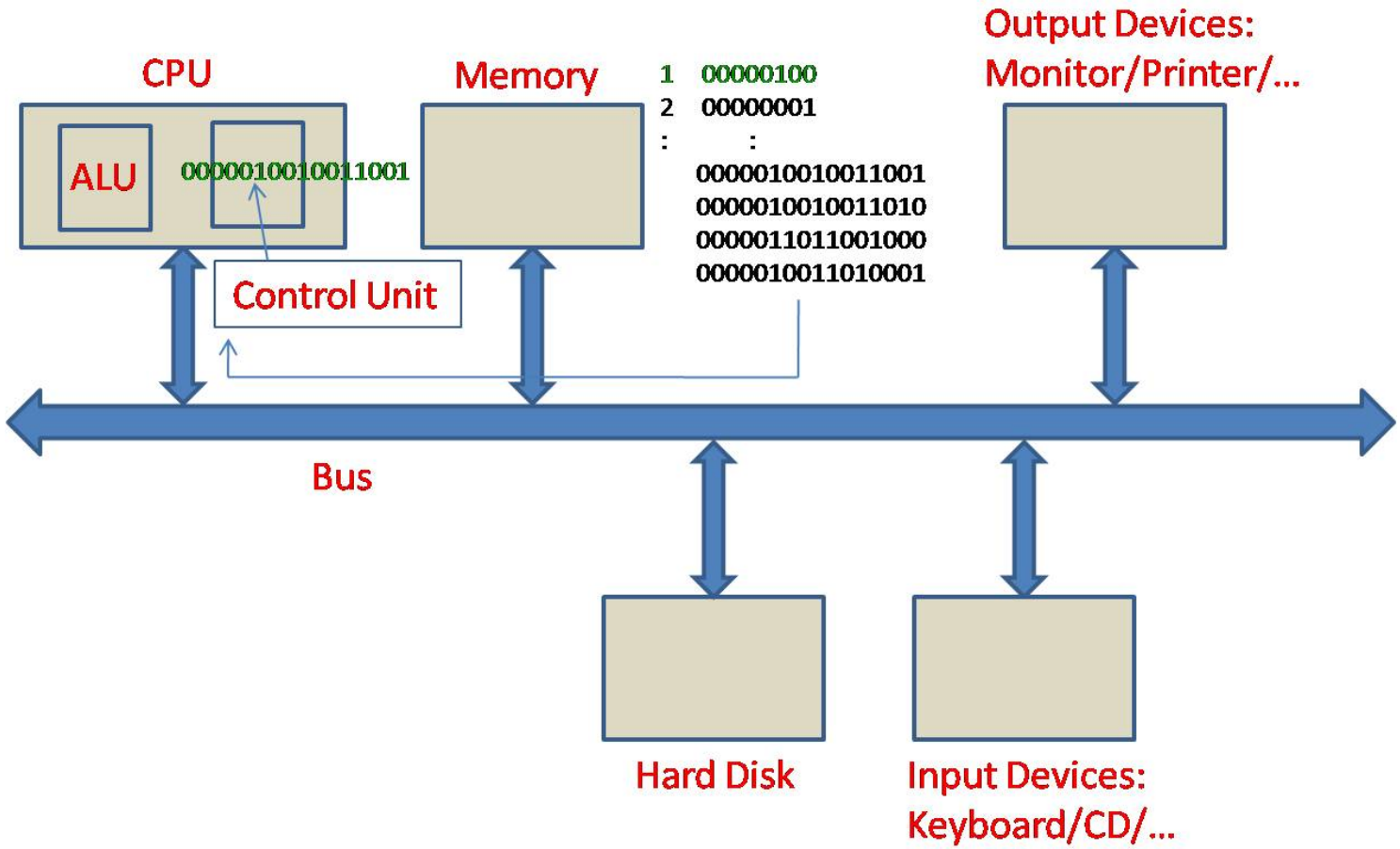
Execution



Step 1: Bring 0000010010011001 to CPU

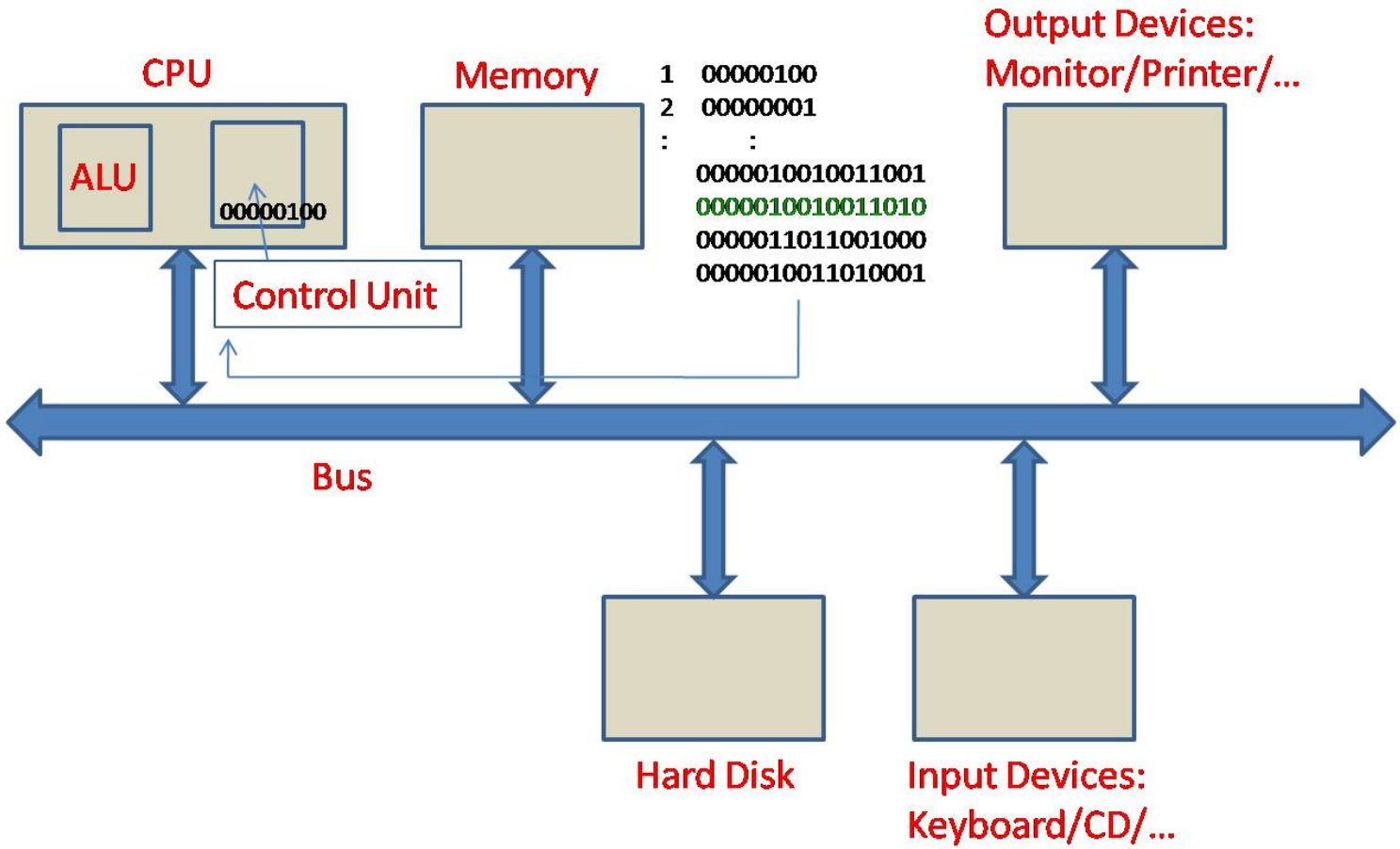
0000010010011001 = Bring data stored in memory location 001 to CPU

Execution



Step 2: Execution of 0000010010011001 in Control Unit
Brings data stored in location 1, number 4, to CPU

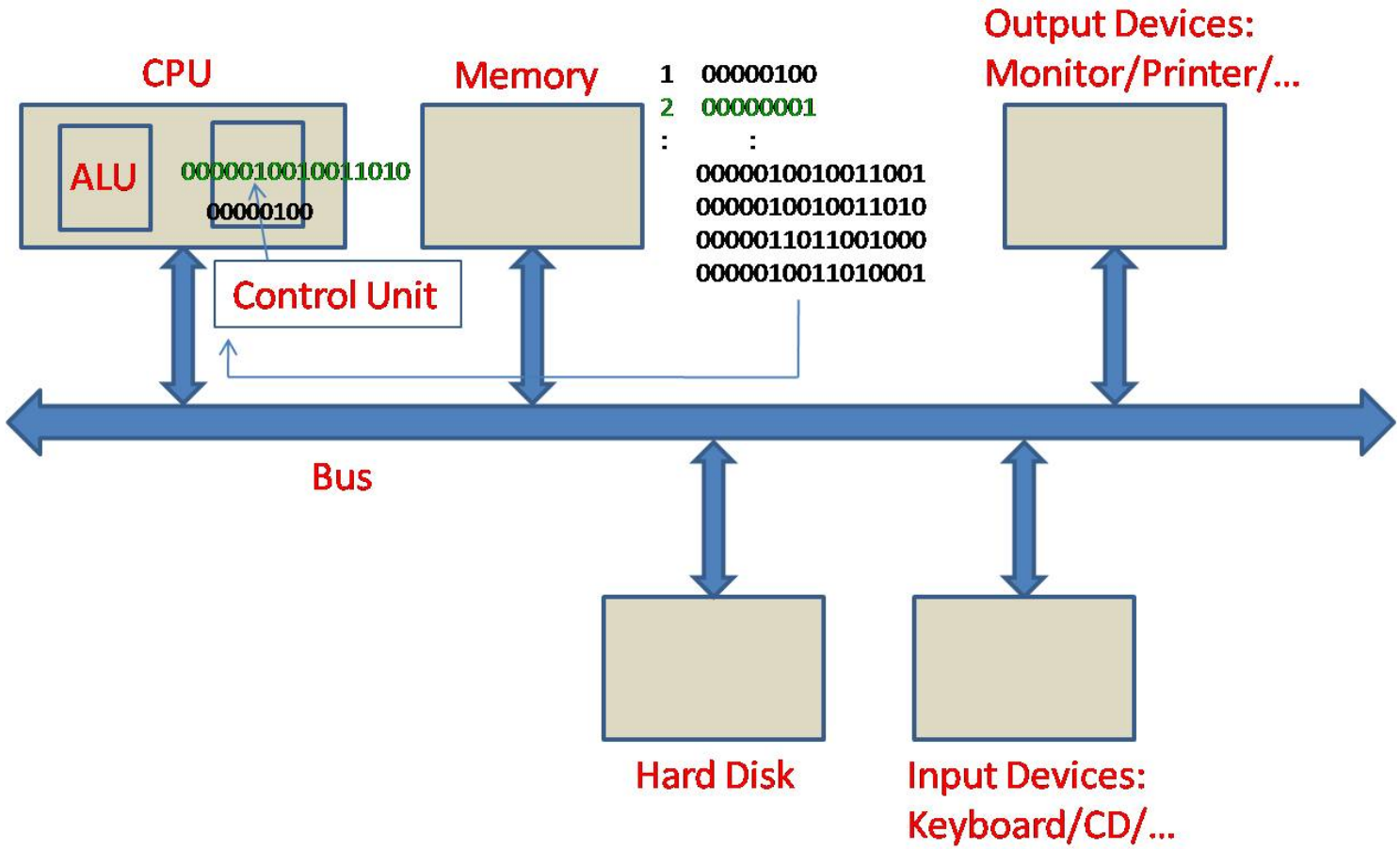
Execution



Step 3: Bring 0000010010011010 to CPU

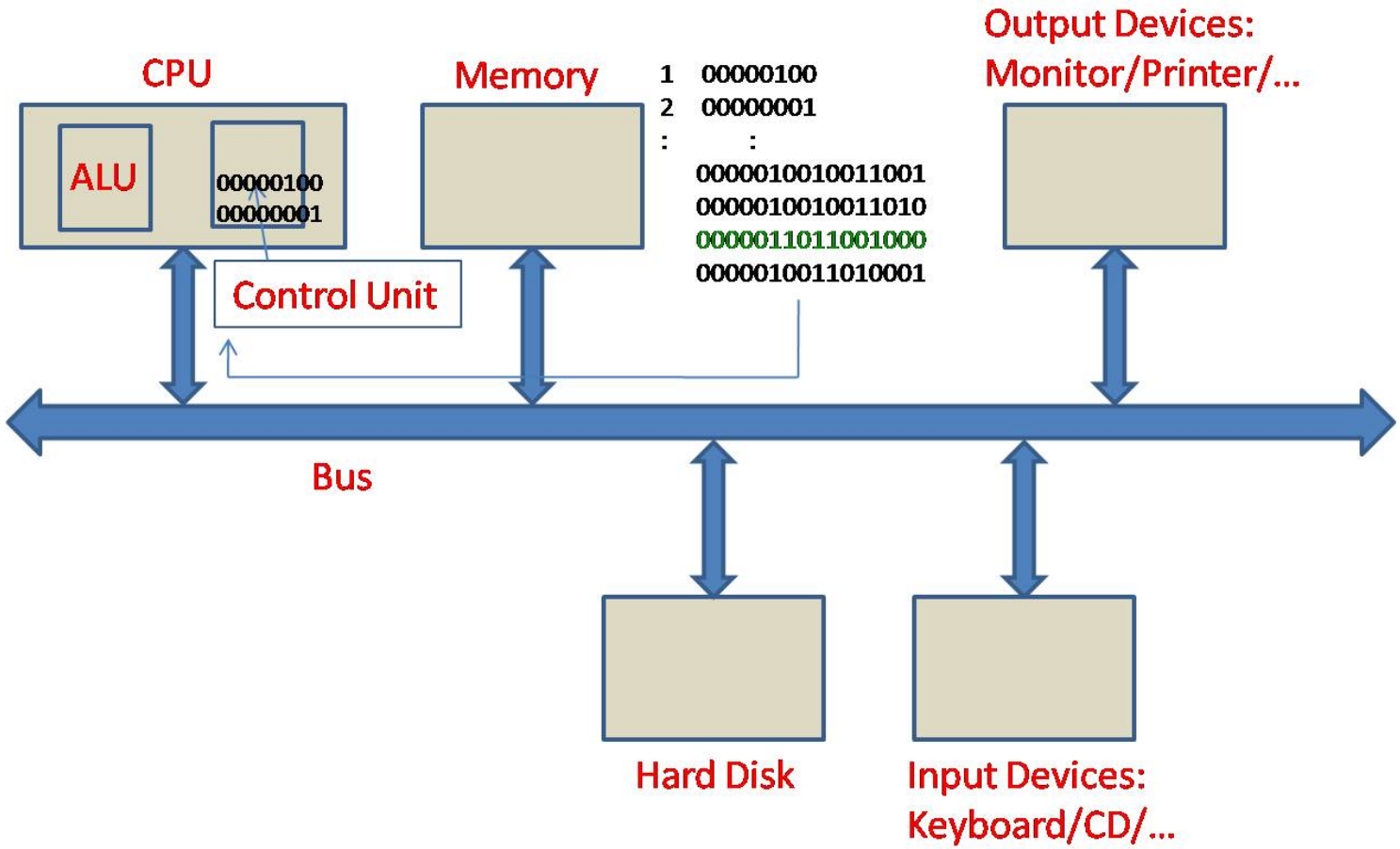
0000010010011010 = Bring data stored in memory location 010 to CPU

Execution



Step 4: Execution of 0000010010011010 in Control Unit
Brings data stored in location 2, number 1, to CPU

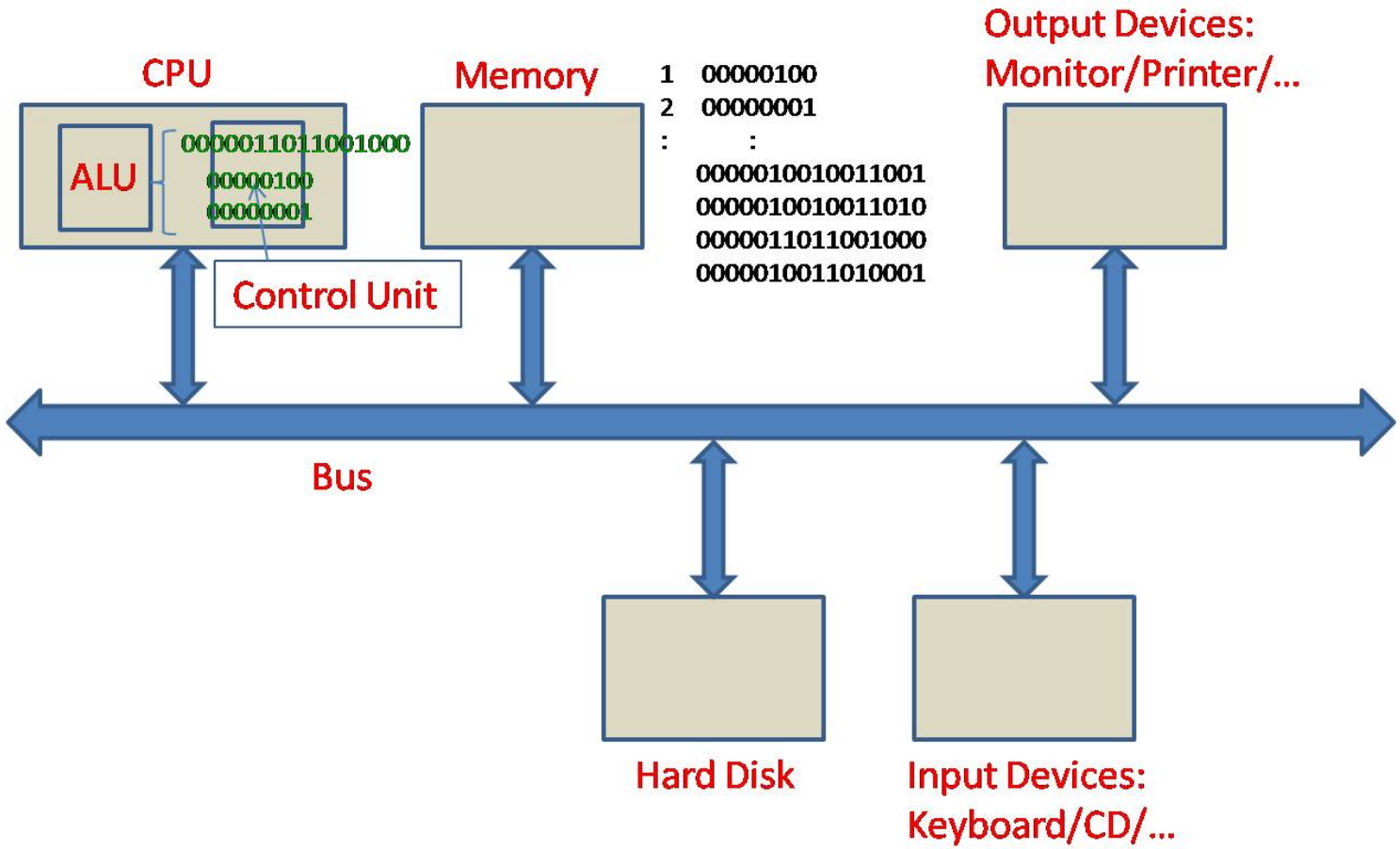
Execution



Step 5: Bring 0000011011001000 to CPU

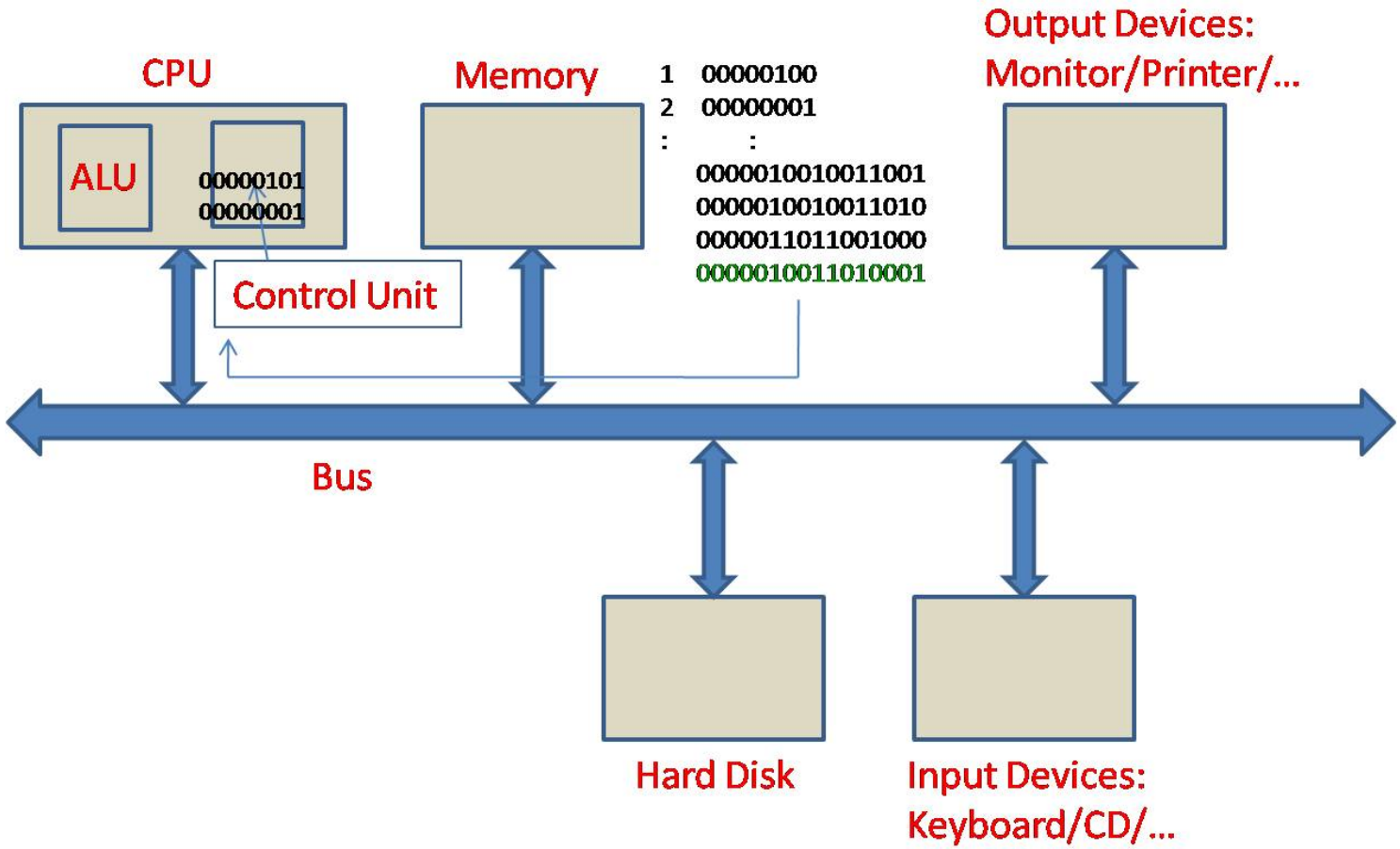
0000011011001000 = Add two stored numbers

Execution



Step 6: Execution of 0000011011001000 in Control Unit
Adds two numbers inside CPU using ALU

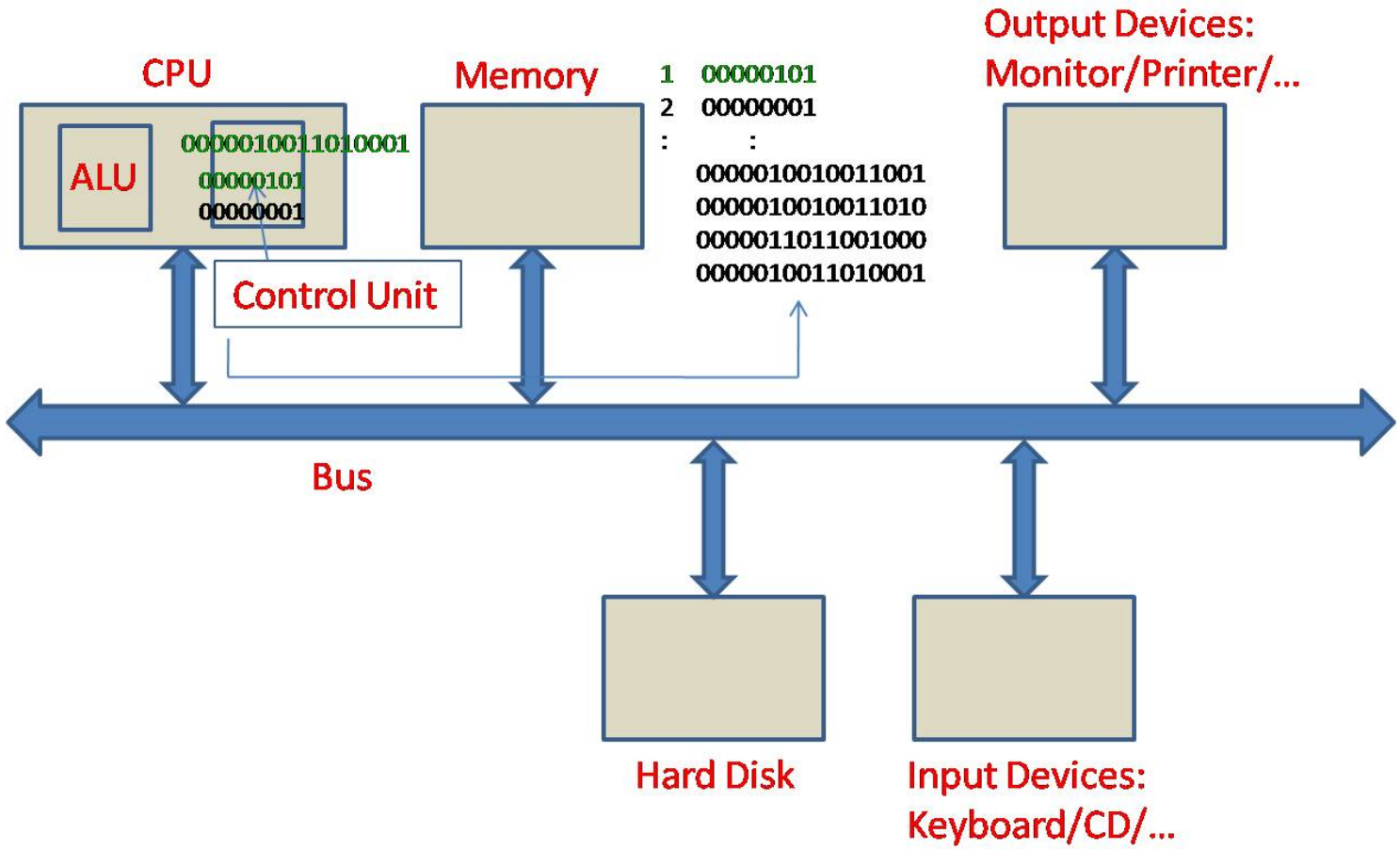
Execution



Step 7: Bring 0000010011010001 to CPU

0000010011010001 = Store number in ALU to memory location 001

Execution

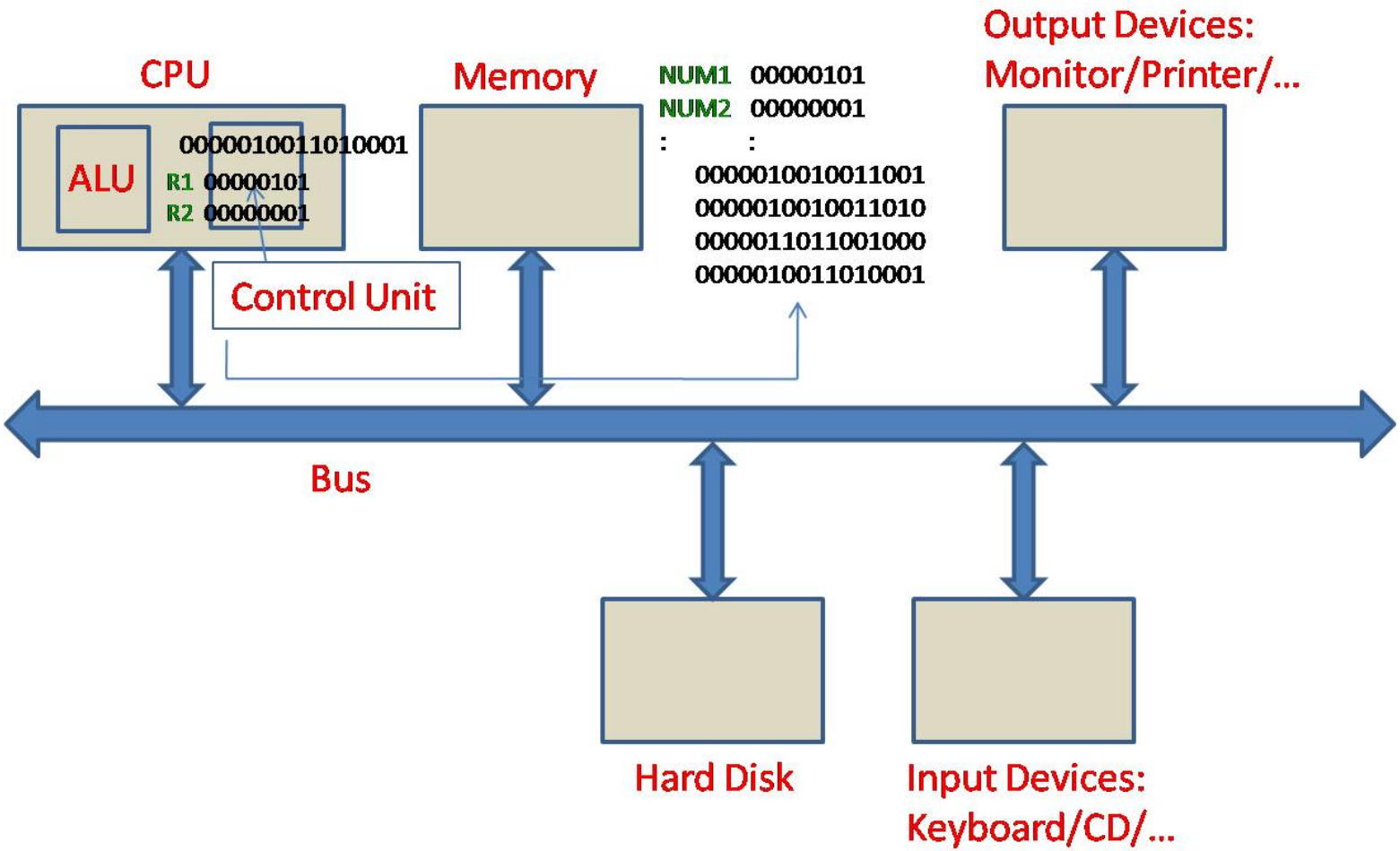


Step 8: Execution of 0000010011010001 in Control Unit
Stores data stored in ALU , number 5, in location 1

Assembly Language

- It is very difficult for us to understand this binary language, called machine language!
- And this is the only language that computers understand!!
- To make it more readable, assembly language was introduced.
- In assembly language, operations and memory locations are addressed by names.

Naming Locations in Example Program



Named Memory and CPU Locations

Example Program in Assembly Language

```
0000010010011001      MOVE  NUM1, R1
0000010010011010      MOVE  NUM2, R2
0000011011001000      ADD  R1, R2
0000010011010001      MOVE  R1, NUM1
```

Move contents of memory location NUM1 to CPU register R1
Move contents of memory location NUM2 to CPU register R2
Add contents of R1 and R2 and store the result in R1
Move the contents of R1 to memory location NUM1

Assemblers

- An assembly language program eventually must be translated to machine language.
- An assembler does this job.
- It maps the names to the corresponding binary values.
- It is also a program!

I/O

- The example only shows how to add numbers already present in the Memory.
- How does one add numbers provided by the user through the keyboard?
- This is the job of another program, called the Operating System (OS in short).

OS

- OS picks the input given by the user and stores it in appropriate locations of Memory.
- It also picks result of computations from Memory and displays to the user.
- It does many other housekeeping jobs that make the interaction of user with the computer easy.
- Examples of OS: Linux, Windows, Mac OS.

Need for a Better Language

- It is very difficult to write large programs in assembly language.
- Several language were created during 1960-80 to simplify the task of the programmer.
- The prominent ones are: COBOL, Fortran, Pascal, C.
- These are called High-level programming languages.
- Assembly and machine language are called low level programming languages.

Adding Two Numbers in C

```
main()
{
    int num1;
    int num2;

    scanf("%d", &num1);
    scanf("%d", &num2);

    num1 = num1 + num2;

    printf("%d", num1);
}
```

Adding Two Numbers in C

```
main()
{
    int num1;
    int num2;

    scanf("%d", &num1);
    scanf("%d", &num2);

    num1 = num1 + num2;

    printf("%d", num1);
}
```

Denotes the beginning and the end of the program.
Names a memory location.
Reads a number from the input and stores it in the specified memory location. Invokes the OS to transfer the number.
Adds two numbers and stores the result in location `num1`. It corresponds to the whole machine language program earlier!
Writes the number on the monitor. Invokes the OS to do it.

Compilers

- A C compiler translates a C program to machine language.
- There are many C compliers. We will use one called gcc.