Systems Programming in C/C++

Mohammed Bahja
School of Computer Science
University of Birmingham

Outline of the module

- Computer architecture: programmer's perspective

- C programming
  - Structure of a C program
  - Pointers and memory management
  - Applications of memory management

- Programming with threads

- Kernel programming ans OS topics

What this module is Not

- Not a ‘Computer Architecture’ module.
- This module is different from non-CS modules on C
- Not a re-run of Software Workshop
- Not a C++ software development module

Assessment

- 50% exam, 50% coursework

- Description of coursework and assessment criteria will be made available on Canvas.

- There will be 3 challenging assignments.
  (lab sessions will start from the second week).

- Make use of the labs (initially two hours per week) to work on the coursework problems.

- Will use virtual machine for exercises; see Canvas for details.

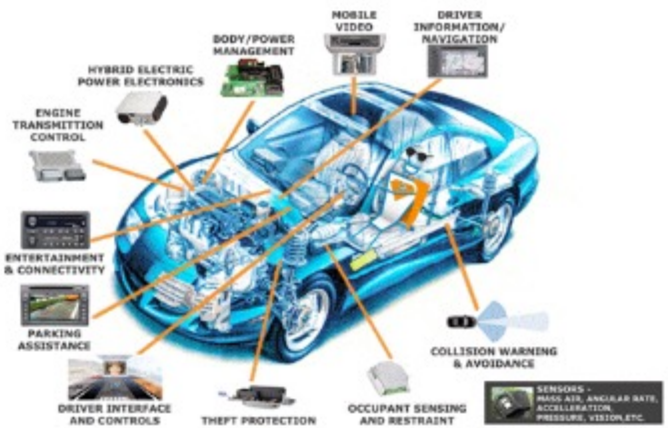
Study materials

- Recommended Course Books
  - The C Programming Language (2nd Edition) Kernighan and Ritchie
  - OS Concepts (10th Edition) Silberschatzet al.

Computers







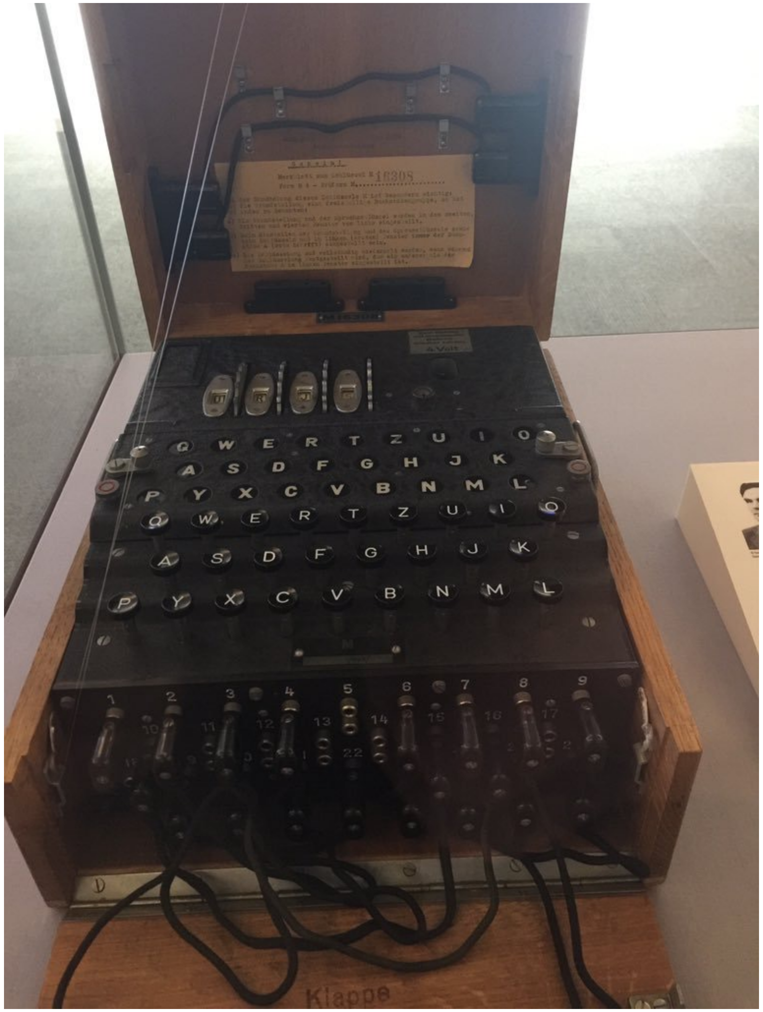
Computers are everywhere

Computers: two types

From application point of view, computers can be classified into two major categories.

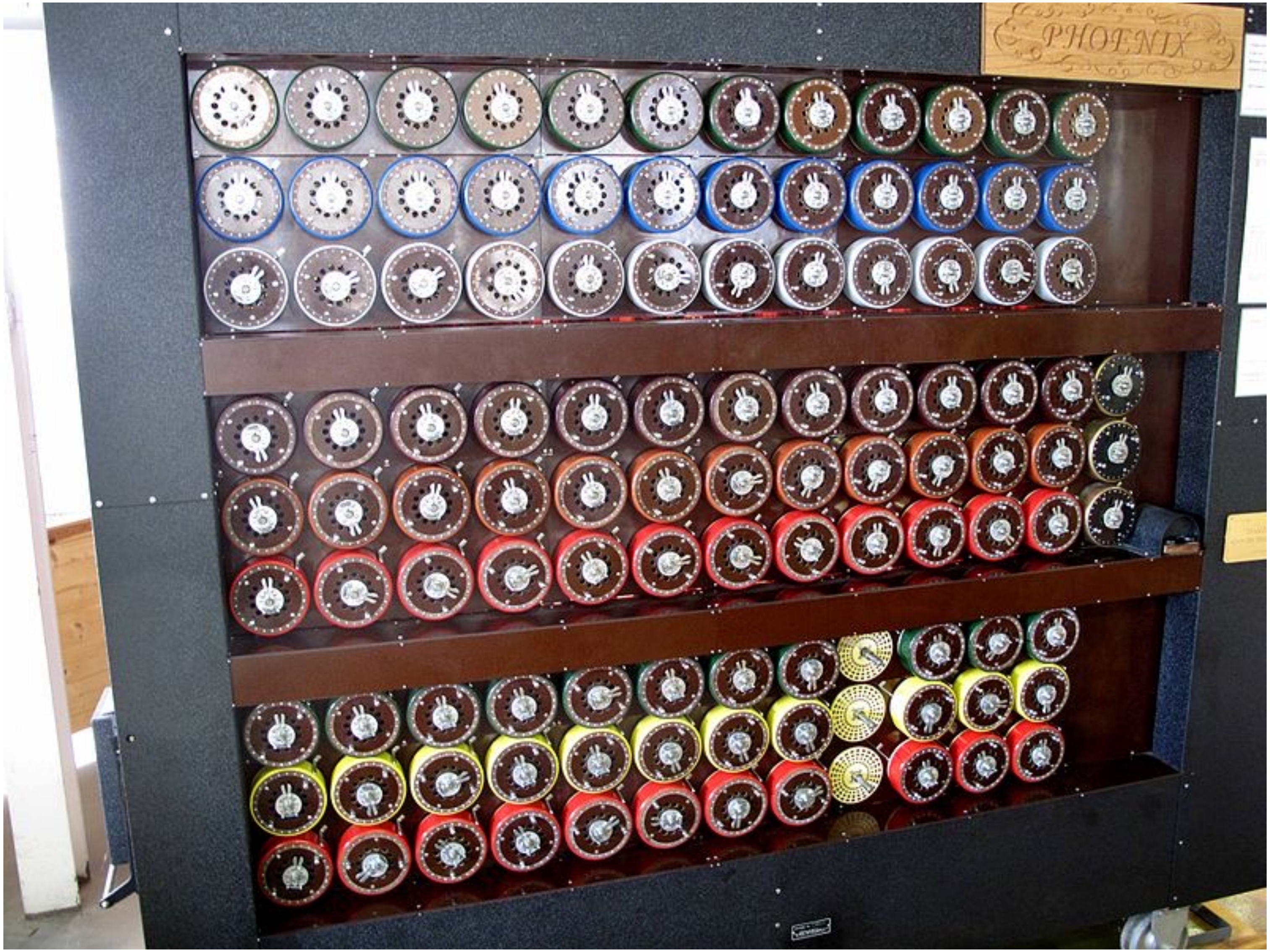
Can perform specific tasks.
E.g. only simple calculations.
Application Specific Computer

Can perform different tasks.
E.g. calculations, watch movies, play games, browse internet etc.
General Purpose Computer



Enigma machine used during World War II
(Photo of the machine at The Alan Turing Institute, London)

Can perform only encryptions → hence application specific



‘British bombe’ an electromechanical computer designed by Alan Turing to break codes produced by the Enigma machine

Major bottleneck: Programming required major re-wiring.

Snap from movie “The Imitation Game”.
Benedict Cumberbatch as Alan Turing.

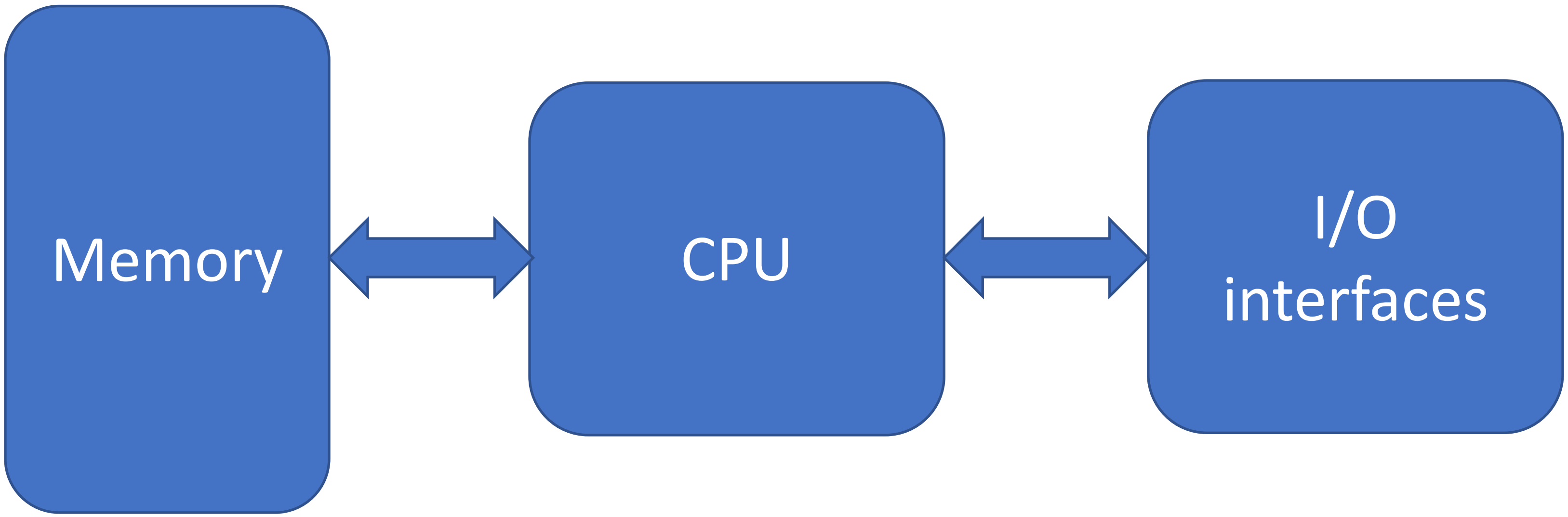
Stored Program Computers

- The invention of stored program computers has been ascribed to John von Neumann.

- Stored-program computers have become known as 'von Neumann Architecture' systems.

A 'stored-program computer' is a computer that stores program instructions in memory.
→ Re-programming does not require any hardware modifications.

The von Neumann Architecture



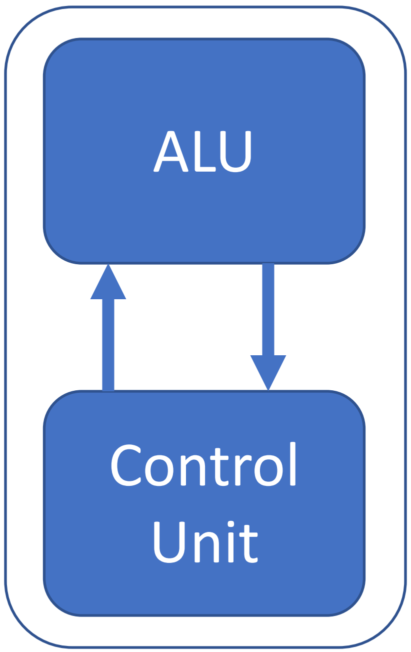
Consists of three main components

1. Central Processing Unit (CPU)
2. Memory
3. Input/Output (I/O) interfaces.

The CPU

The CPU can be considered the heart of the computing system.
It includes two main components:

1. Control Unit (CU),
2. **Arithmetic and Logic Unit (ALU)**



ALU performs the mathematical or logical operations.

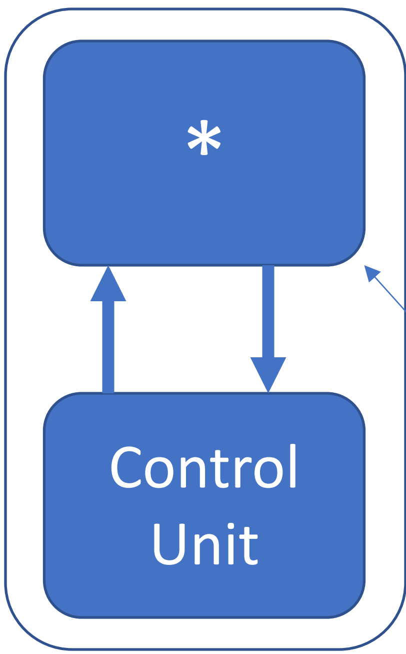
Example:

```
main()
{
        int a=5, b=6, c;
        c = a*b;
        c = c + b
}
```

The CPU

The CPU can be considered the heart of the computing system.
It includes two main components:

1. Control Unit (CU),
2. **Arithmetic and Logic Unit (ALU)**



ALU computes multiplication

ALU performs the mathematical or logical operations.

Example:

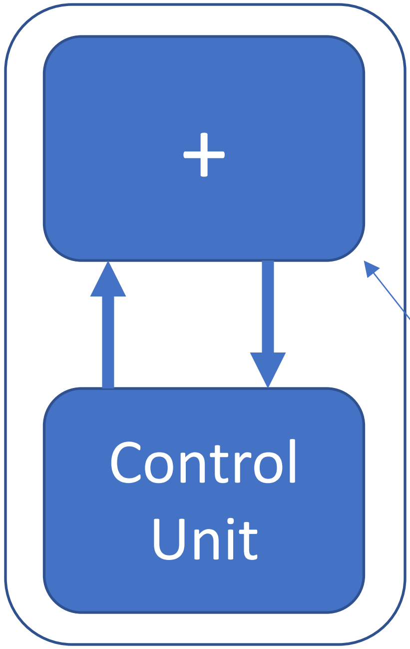
```
main()
{
    int a=5, b=6, c;
    c = a*b;
    c = c + b
}
```

The CPU

The CPU can be considered the heart of the computing system.

It includes two main components:

1. Control Unit (CU),
2. **Arithmetic and Logic Unit (ALU)**



ALU computes addition

ALU performs the mathematical or logical operations.

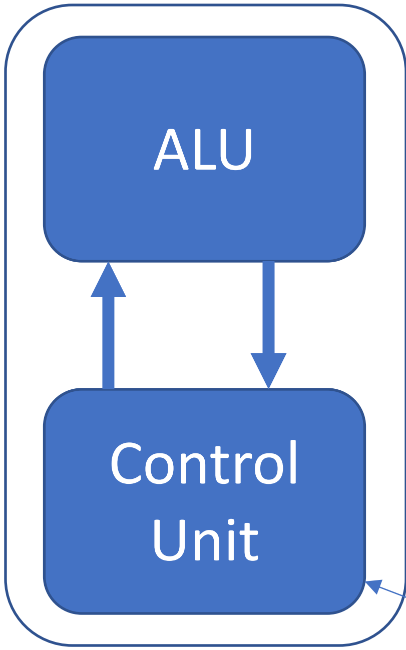
Example:

```
main()
{
    int a=5, b=6, c;
    c = a*b;
    c = c + b
}
```

The CPU

The CPU can be considered the heart of the computing system.
It includes two main components:

1. **Control Unit (CU),**
2. Arithmetic and Logic Unit (ALU)



1. Retrieves the operands
2. Asks ALU to compute *
3. Stores the result
4. Jumps to the next line

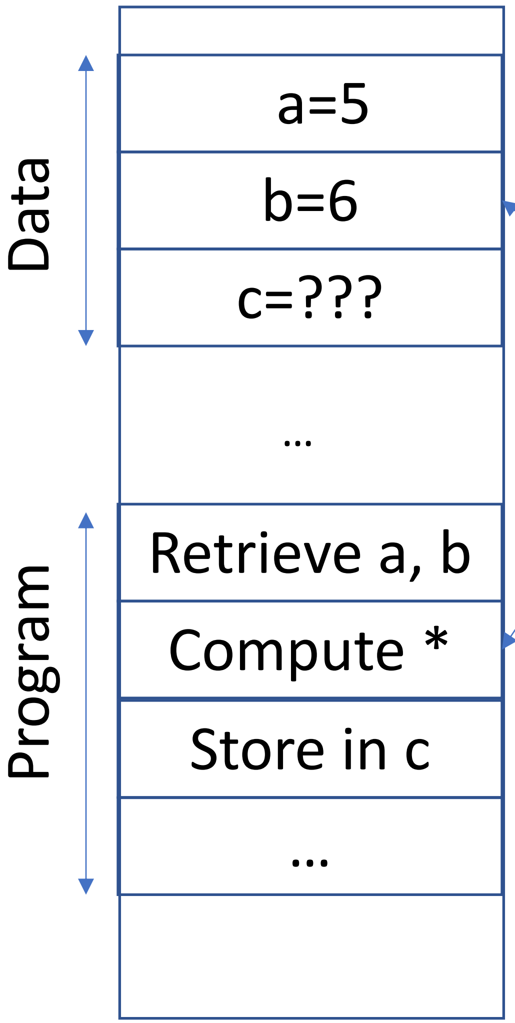
Control Unit determines the order in which instructions should be executed and controls the retrieval of the proper operands.

Example:

```
main()
{
      int a=5, b=6, c;
      c = a*b;
      c = c + b
}
```

The Memory

The computer's memory is used to store both program instructions and data.

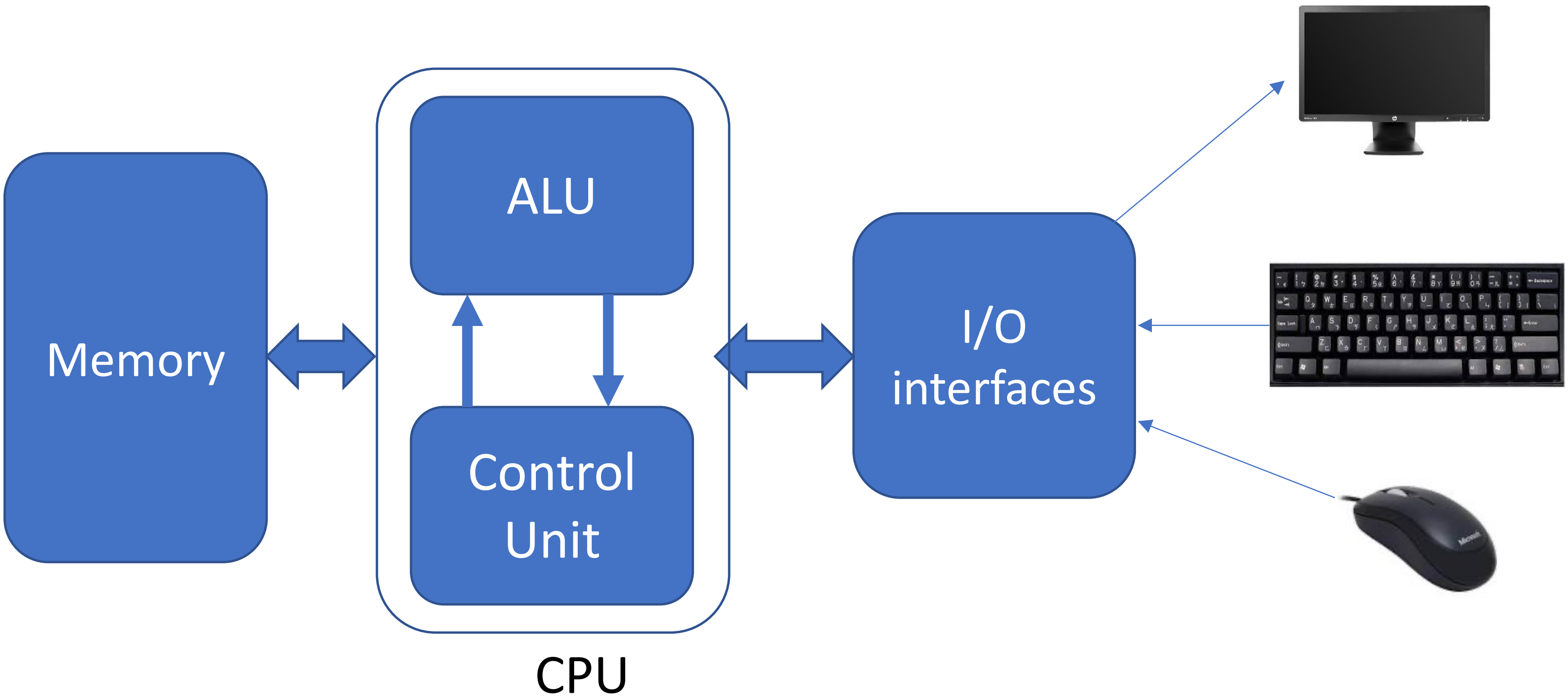



```
main()
{
        int a=5, b=6, c;
        c = a*b;
        c = c + b
}
```

The Input/Output Interfaces

- The I/O interfaces are used to receive or send information from/to connected devices.

- Connected devices are called **peripheral devices**.

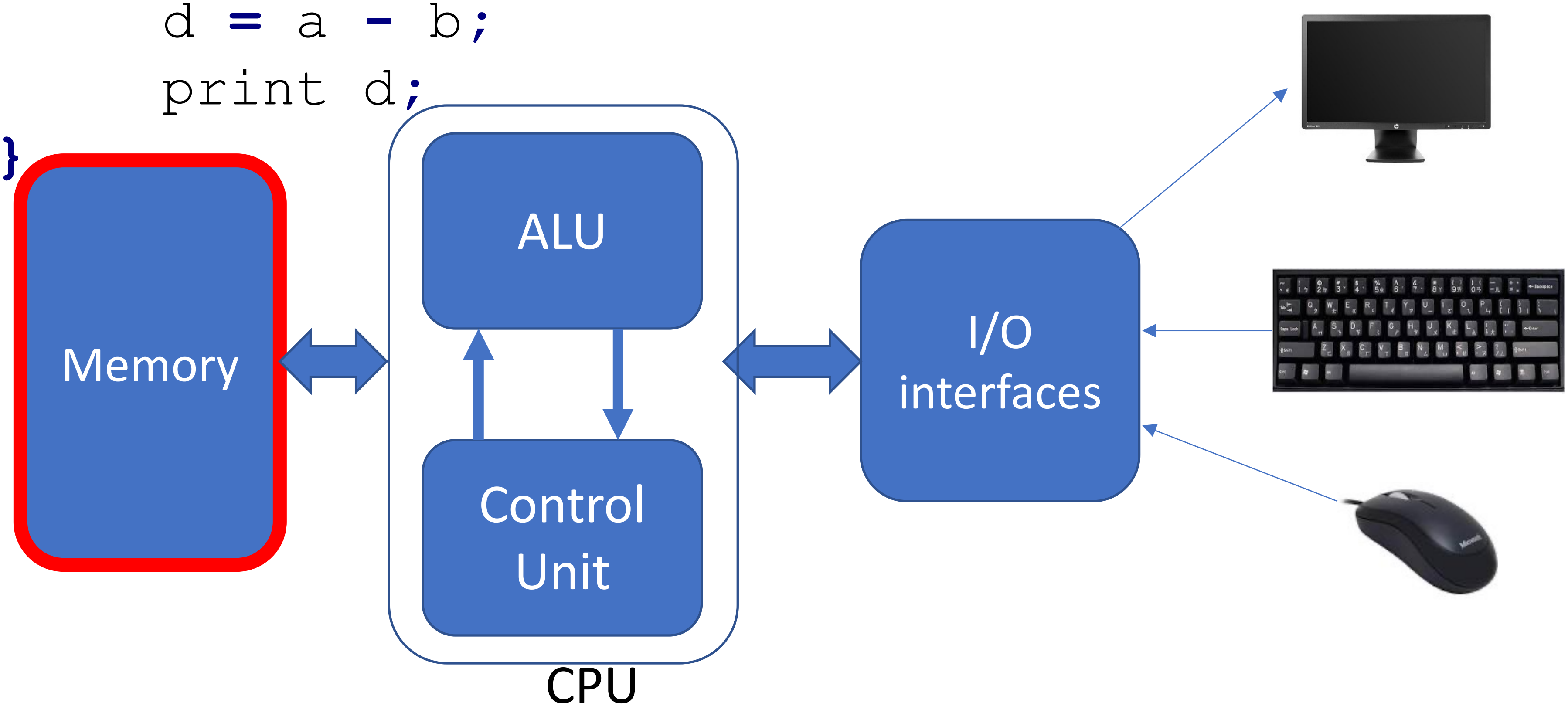


Program execution on von Neumann computer

```
main(){

      readIO a;
      readIO b;
      c = a + b;
      store c;
      d = a - b;
      print d;
}
```

A program is stored in the memory.

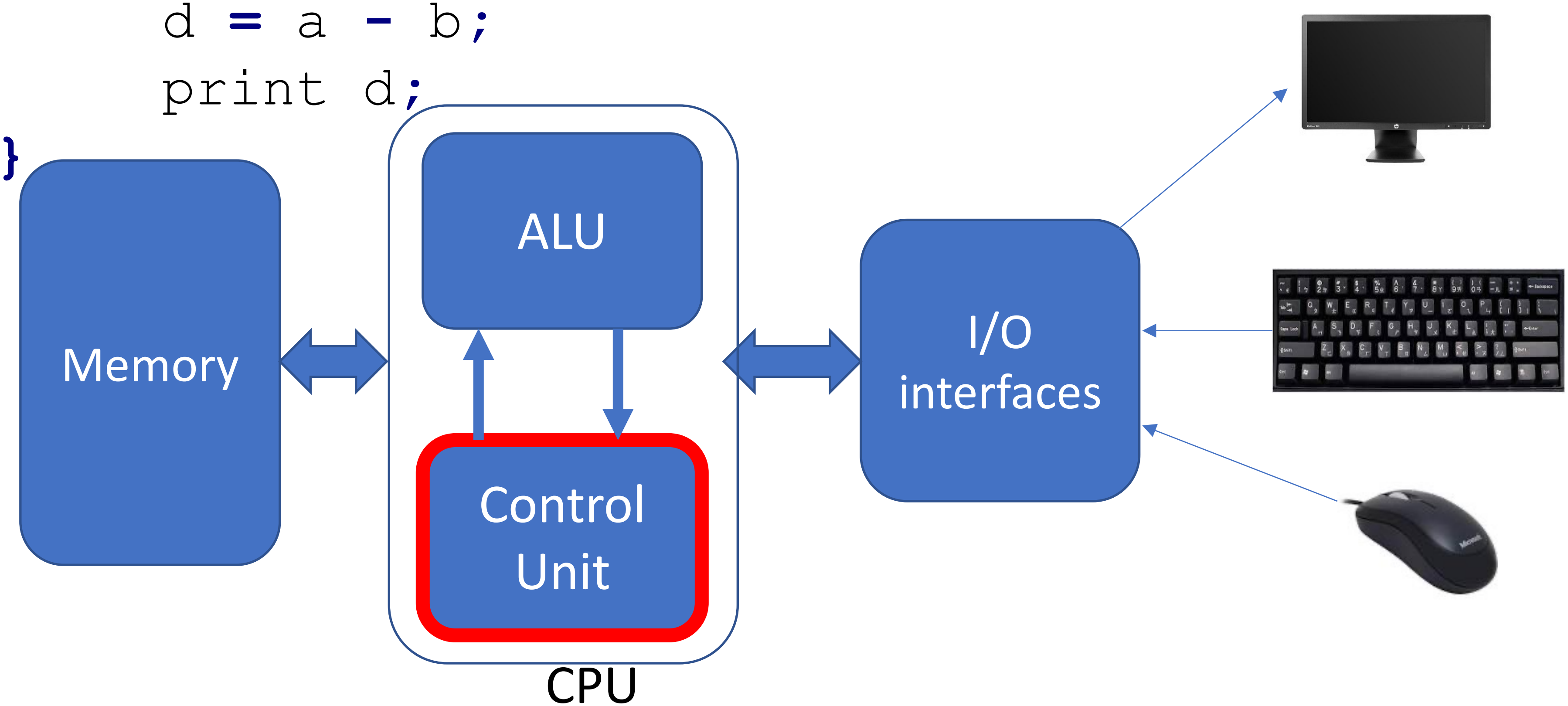


Program execution on von Neumann computer

```
main(){

    readIO a;
    readIO b;
    c = a + b;
    store c;
    d = a - b;
    print d;
}
```

1. Control Unit understands that data needs to be provided by User.

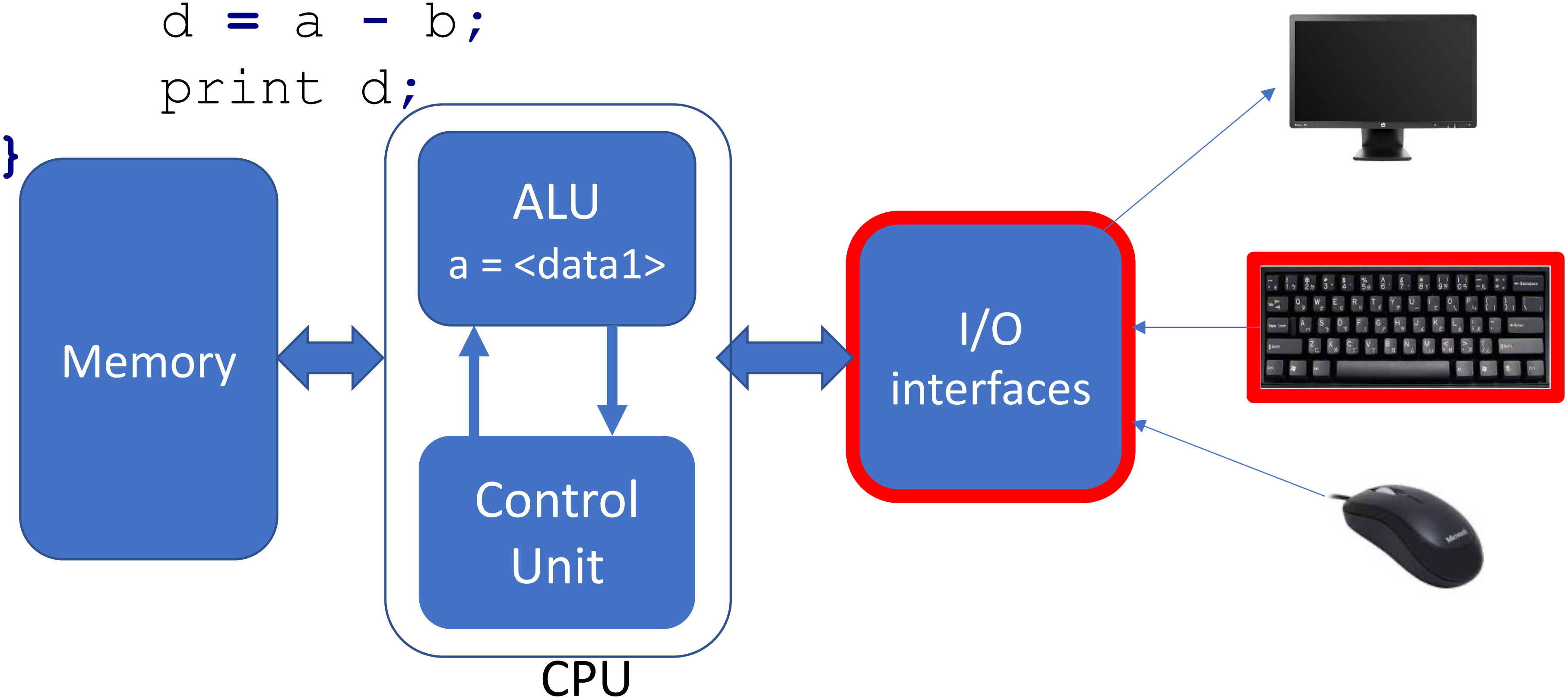


Program execution on von Neumann computer

```
main(){

    readIO a;
    readIO b;
    c = a + b;
    store c;
    d = a - b;
    print d;
}
```

2. Data is read from input device (e.g. keyboard) and brought to the CPU

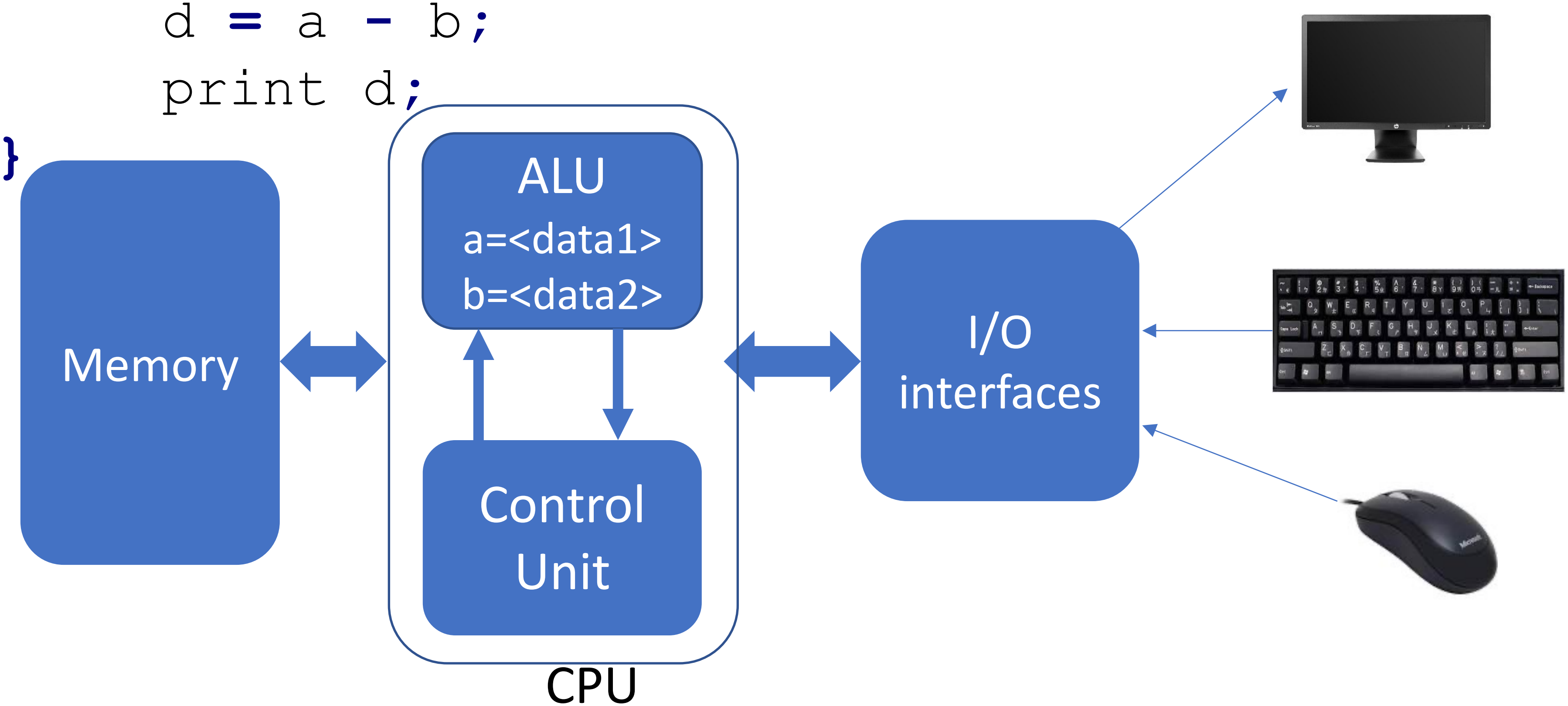


Program execution on von Neumann computer

```
main(){

    readIO a;
    readIO b;
    c = a + b;
    store c;
    d = a - b;
    print d;
}
```

3-4. Similar steps are followed

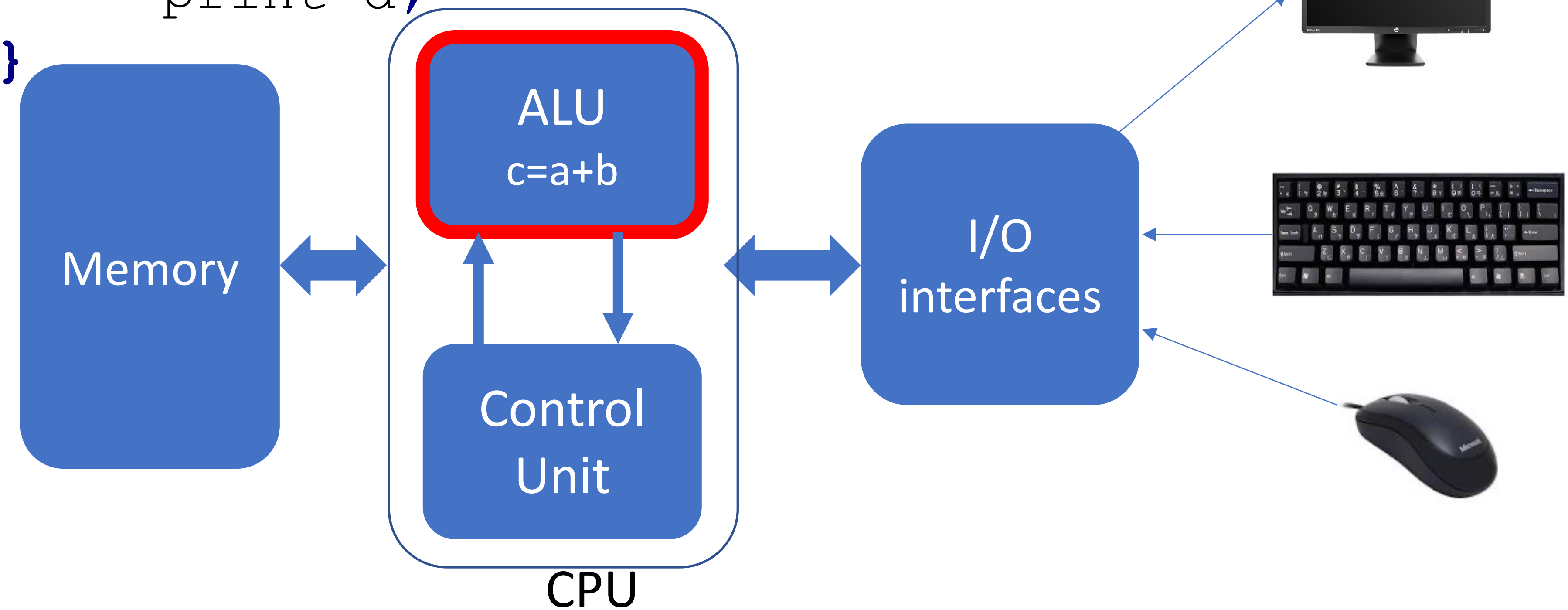


Program execution on von Neumann computer

```
main(){

      readIO a;
      readIO b;
      c = a + b;
      store c;
      d = a - b;
      print d;
}
```

5. Controller commands ALU to compute addition

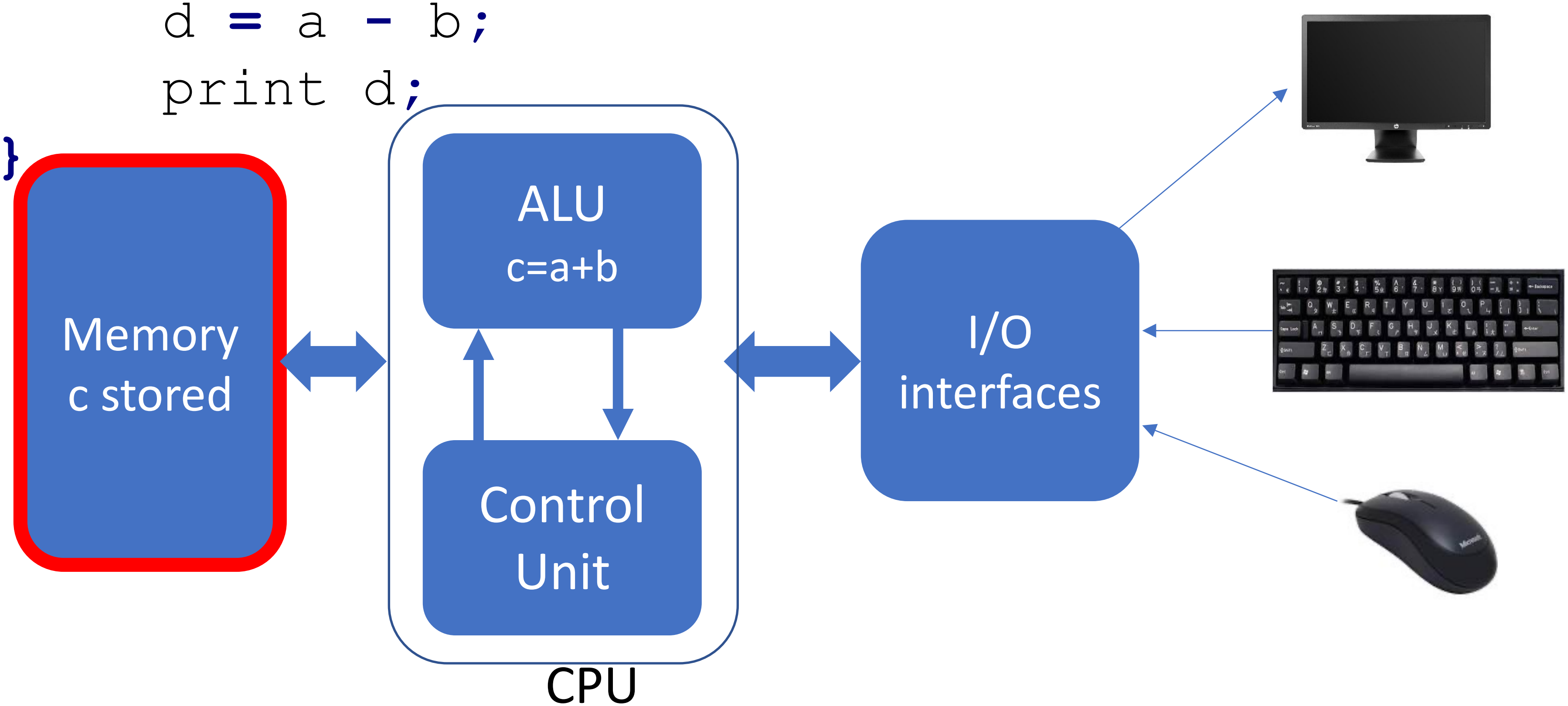


Program execution on von Neumann computer

```
main() {

      readIO a;
      readIO b;
      c = a + b;
      store c;
      d = a - b;
      print d;
}
```

6. Controller copies data from ALU to Memory

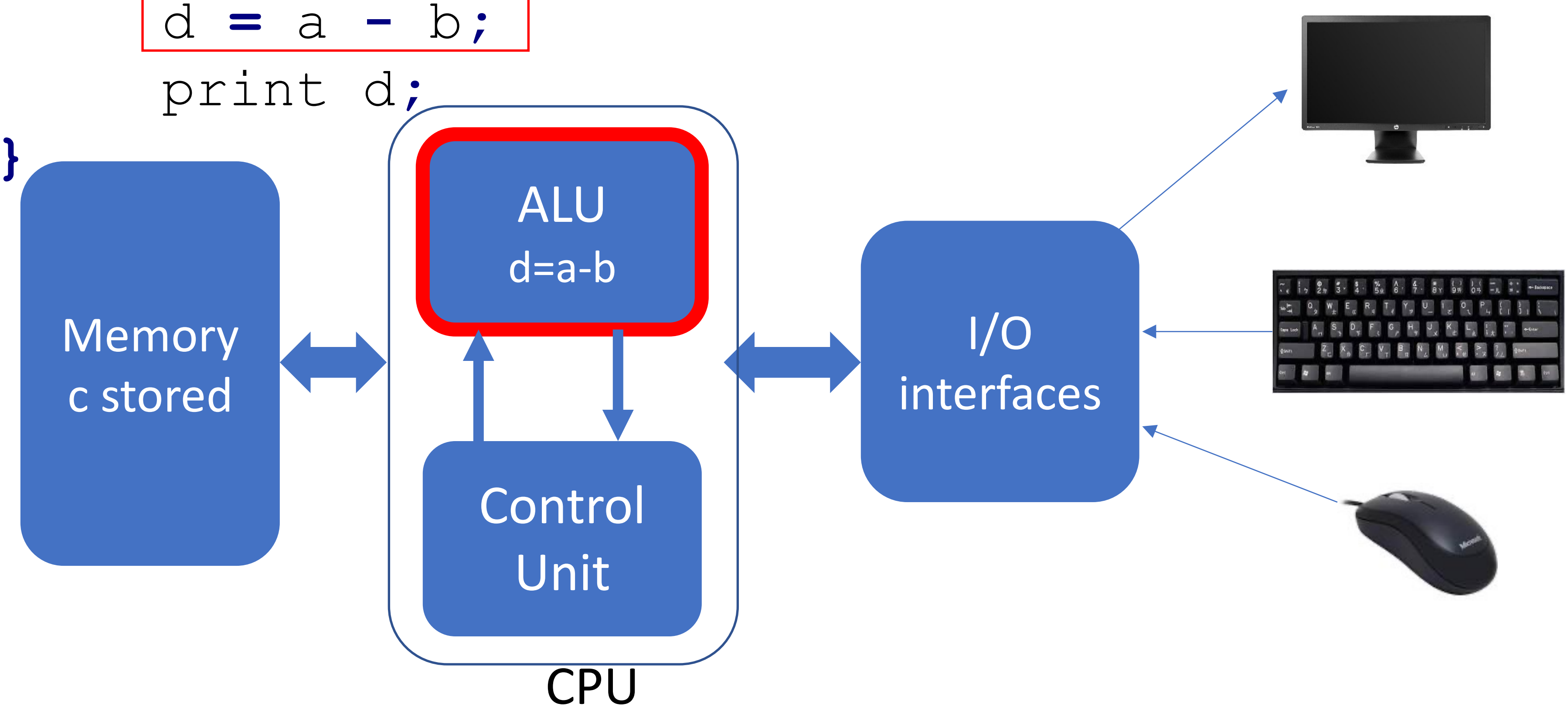


Program execution on von Neumann computer

```
main(){

    readIO a;
    readIO b;
    c = a + b;
    store c;
    d = a - b;
    print d;
}
```

7. Controller commands ALU to compute subtraction

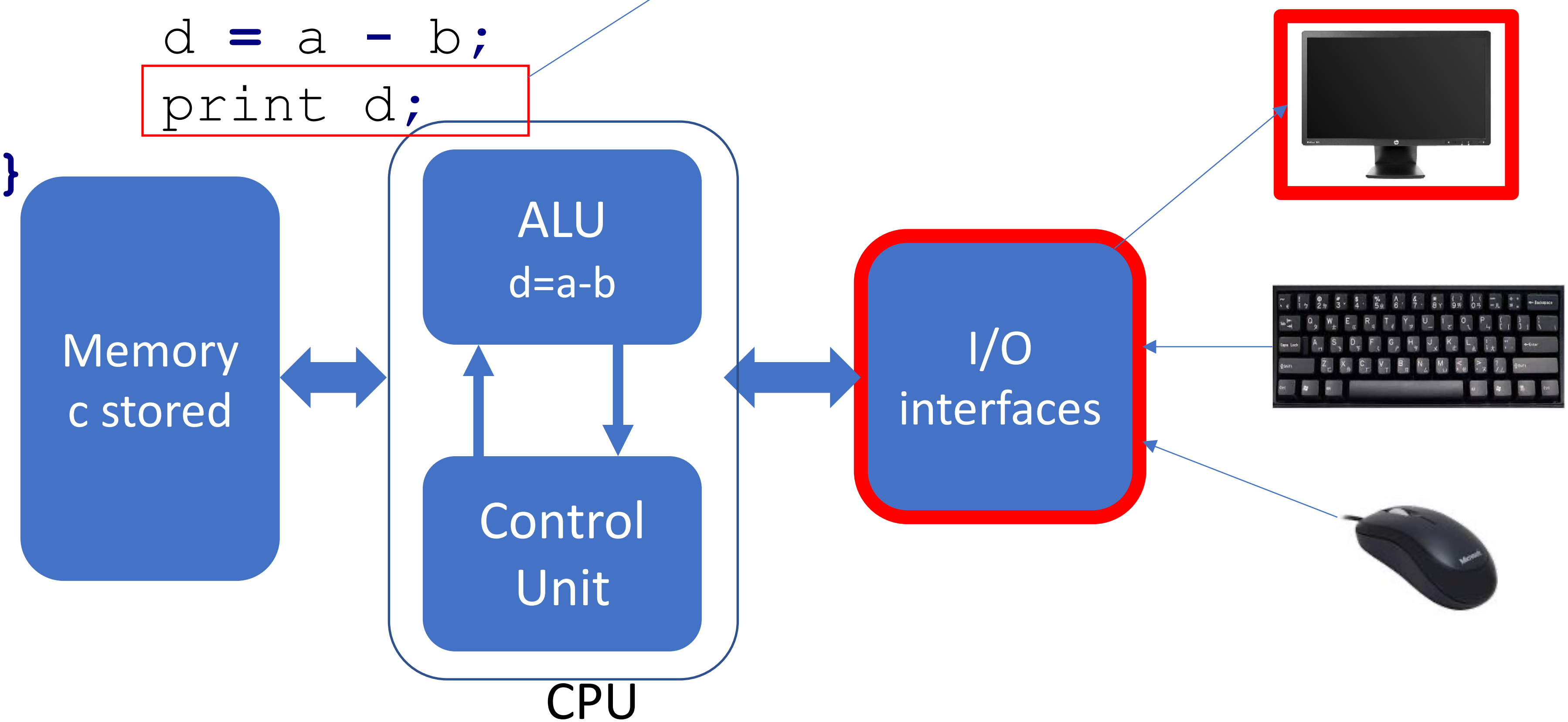


Program execution on von Neumann computer

```
main(){

      readIO a;
      readIO b;
      c = a + b;
      store c;
      d = a - b;
      print d;
}
```

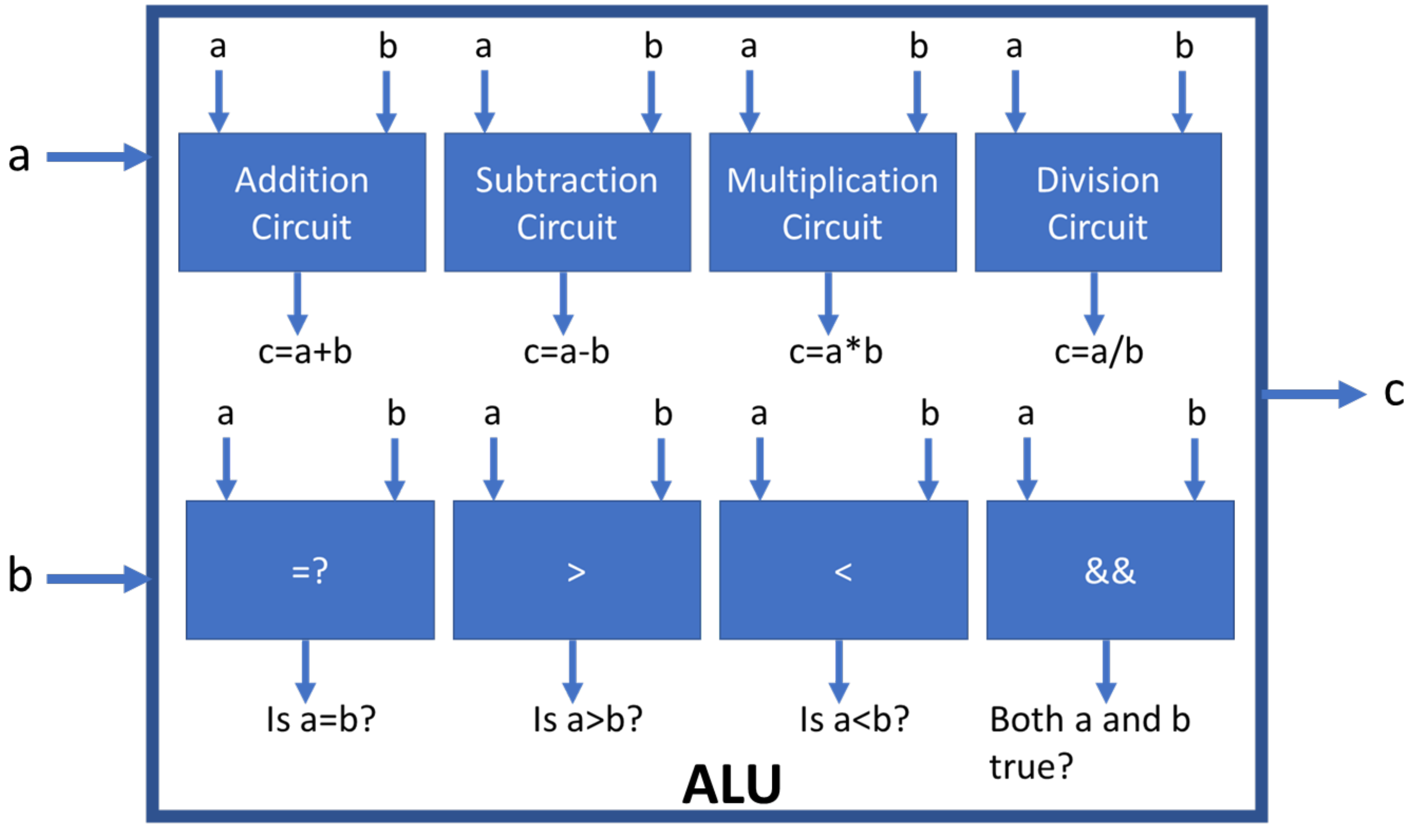
8. Controller sends data from ALU to display



Organization of a CPU

Inside a CPU: Arithmetic and Logic Unit

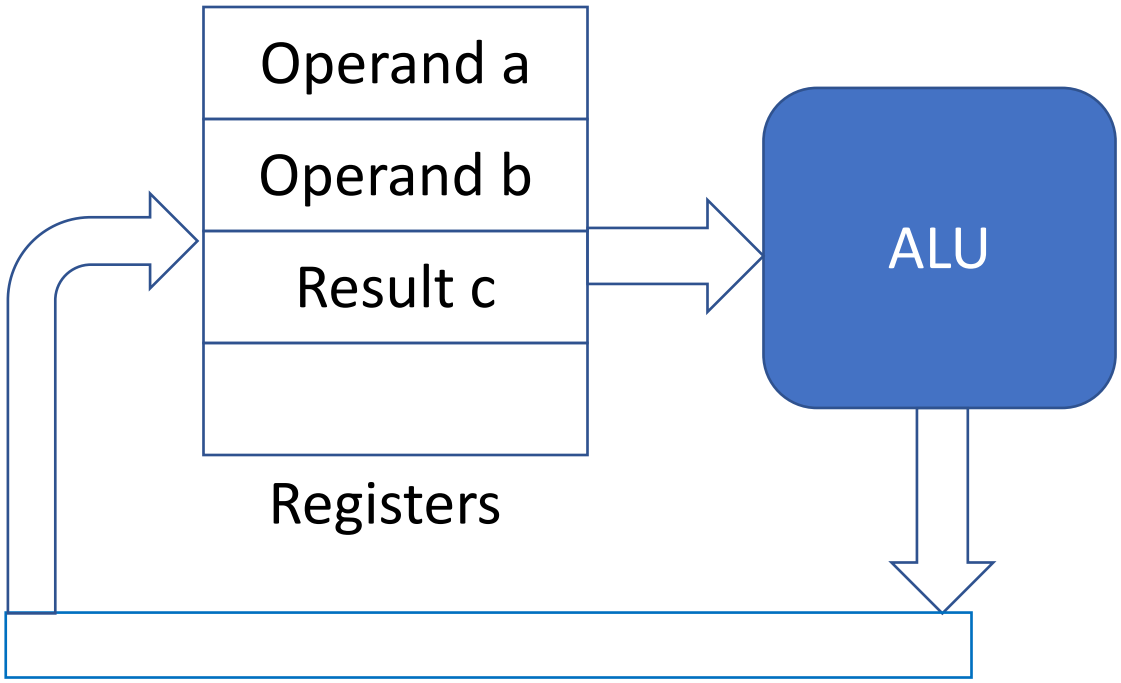
- **Arithmetic and Logic Unit** (ALU) is the union of the circuits for performing arithmetic and logical operations.



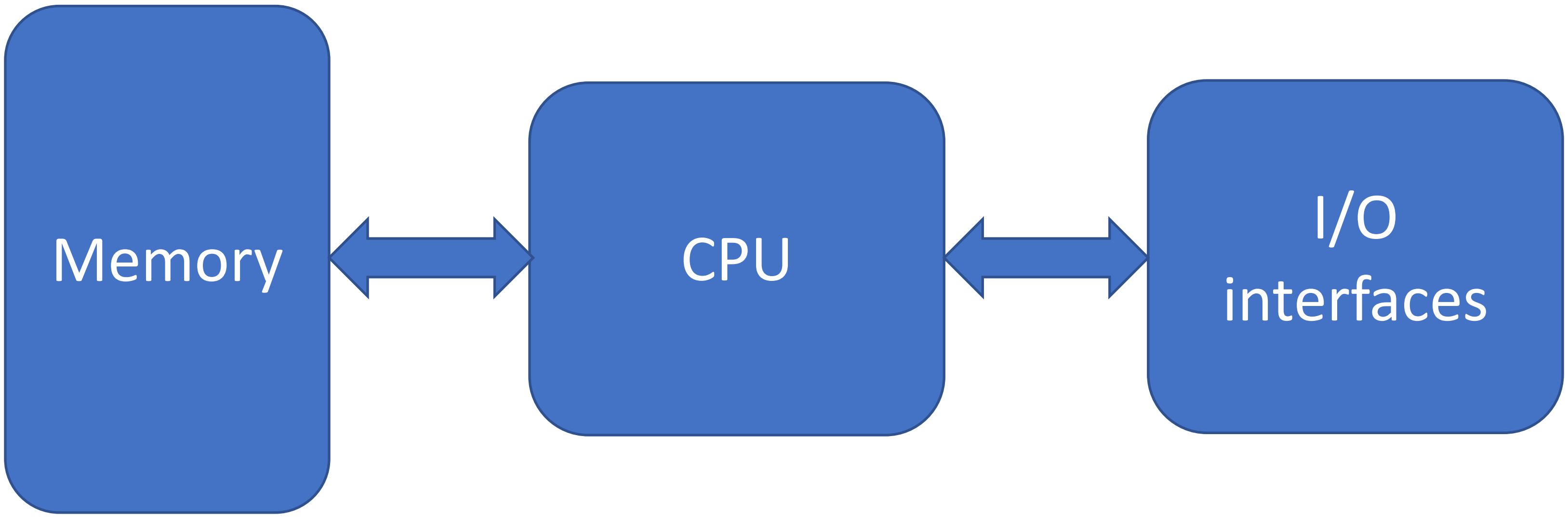
- **Control Unit** is responsible for step-by-step execution of instructions during a program execution.

Inside a CPU: Registers

- Registers are small storage elements that are located very close to the ALU.
- Registers are used as temporary storage during computation.
- ALU can read from and write to registers very fast.



What is the advantage of having registers inside CPU?



von Neumann Architecture has three main components

1. Central Processing Unit (CPU)
2. Memory
3. Input/Output interfaces.

CPU need not have Registers. We can use Memory to store all data.

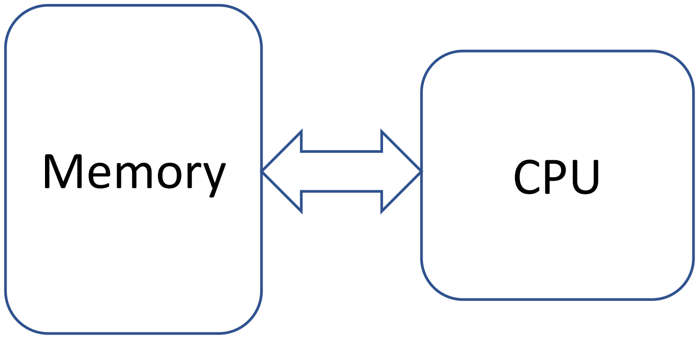
What is the advantage of having registers inside CPU?

Consider this computation: $c = a*b + a + b + (a-b)*a$

What is the advantage of having registers inside CPU?

Consider this computation: c = a*b + a + b + (a-b)*a

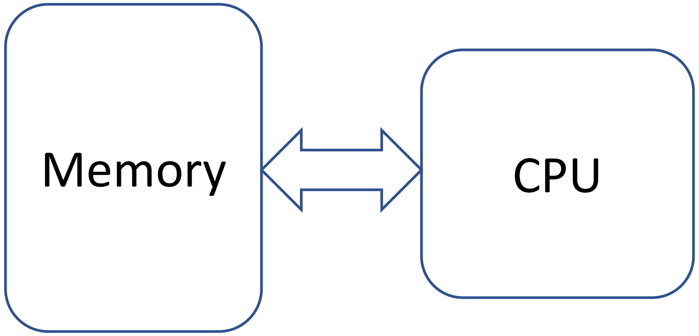
Case 1: without registers



What is the advantage of having registers inside CPU?

Consider this computation: c = a*b + a + b + (a-b)*a

Case 1: without registers



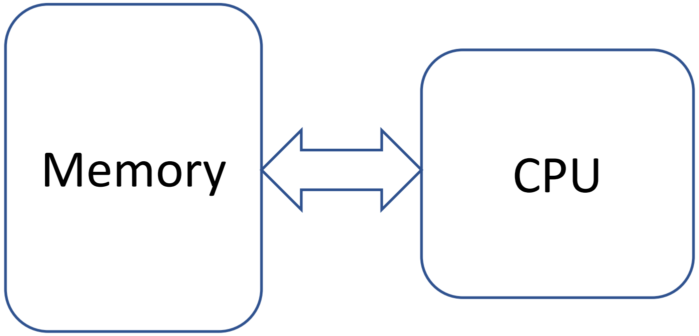
Computation steps

1. read {a,b} from memory

What is the advantage of having registers inside CPU?

Consider this computation: c = a*b + a + b + (a-b)*a

Case 1: without registers



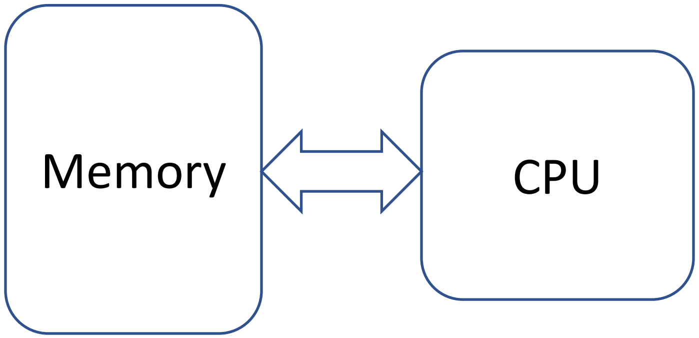
Computation steps

1. read {a,b} from memory
2. compute c = a*b

What is the advantage of having registers inside CPU?

Consider this computation: c = a*b + a + b + (a-b)*a

Case 1: without registers

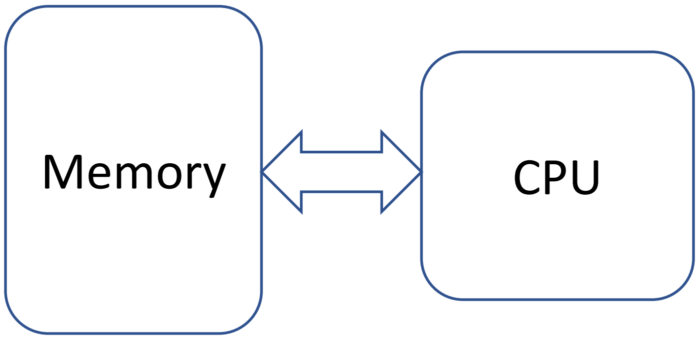


Computation steps
1. read {a,b} from memory
2. compute c = a*b
3. store c in memory

What is the advantage of having registers inside CPU?

Consider this computation: c = a*b + a + b + (a-b)*a

Case 1: without registers



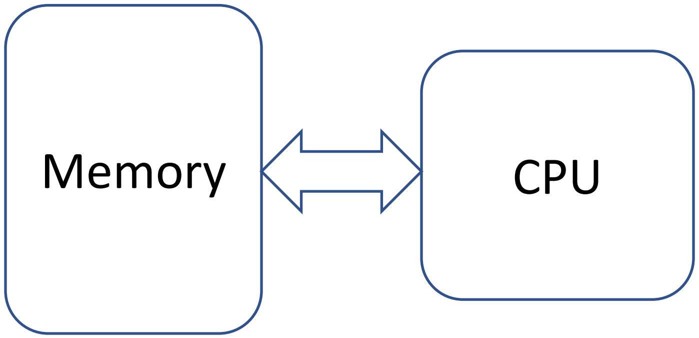
Computation steps

1. read {a,b} from memory
2. compute c = a*b
3. store c in memory
4. read {a,c} from memory
5. compute c = c + a
6. store c

What is the advantage of having registers inside CPU?

Consider this computation: c = a*b + a + b + (a-b)*a

Case 1: without registers



Performance of a computer is measured in 'time requirement' for a task. Assume that

- Each memory read/write takes 4 milliseconds
- Each arithmetic takes 1 millisecond.

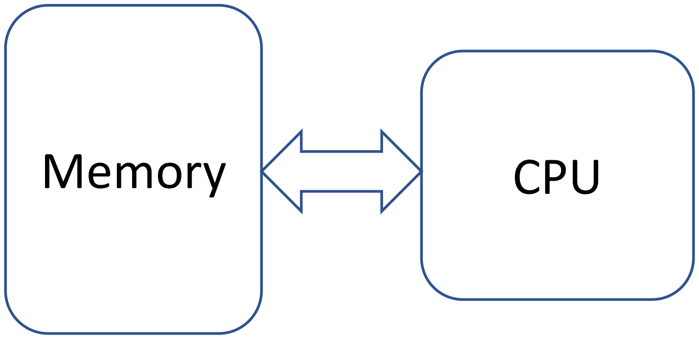
Computation steps (all)

1. read {a,b} from memory
2. compute c = a*b
3. store c in memory
4. read {a,c} from memory
5. compute c = c + a
6. store c
7. read {b,c} from memory
8. compute c = c + b
9. store c
10. read {a,b} from memory
11. compute c' = a-b
12. store c' in memory
13. read {c',a} from memory
14. compute c' = c'*a
15. store c' in memory
16. read {c,c'} from memory
17. compute c = c+c'
18. store c in memory

What is the advantage of having registers inside CPU?

Consider this computation: c = a*b + a + b + (a-b)*a

Case 1: without registers



Computation steps (all)

1. read {a,b} from memory
2. compute c = a*b
3. store c in memory
4. read {a,c} from memory
5. compute c = c + a
6. store c
7. read {b,c} from memory
8. compute c = c + b
9. store c
10. read {a,b} from memory
11. compute c' = a-b
12. store c' in memory
13. read {c',a} from memory
14. compute c' = c'*a
15. store c' in memory
16. read {c,c'} from memory
17. compute c = c+c'
18. store c in memory

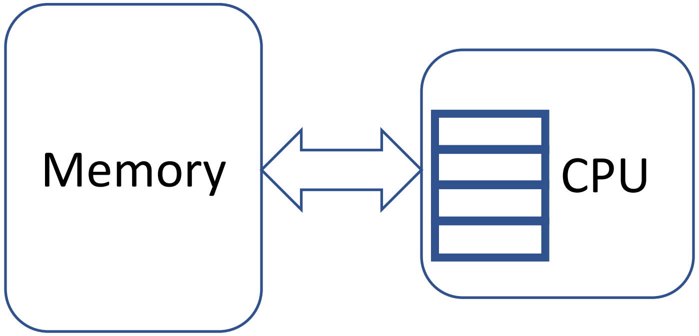
In this particular computation,
No. memory read/write = 12
No. arithmetic = 6

Hence, total time = 12*4 + 6*1
= 54 milliseconds

What is the advantage of having registers inside CPU?

Consider this computation: c = a*b + a + b + (a-b)*a

Case 2: with registers



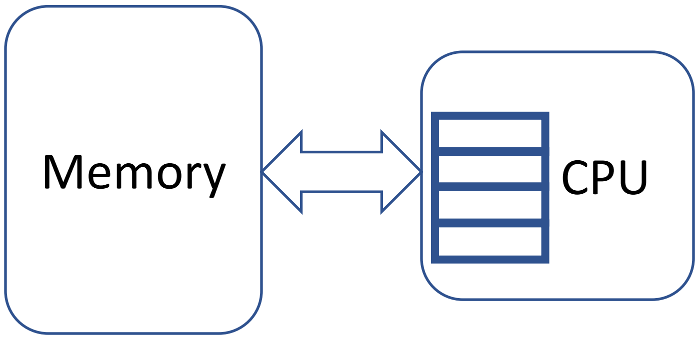
Two assumptions:

1. Initially, a and b are in Memory
2. ALU can read/write registers in 0 time overhead

What is the advantage of having registers inside CPU?

Consider this computation: c = a*b + a + b + (a-b)*a

Case 2: with registers



Two assumptions:

1. Initially, a and b are in Memory
2. ALU can read/write registers in 0 time overhead

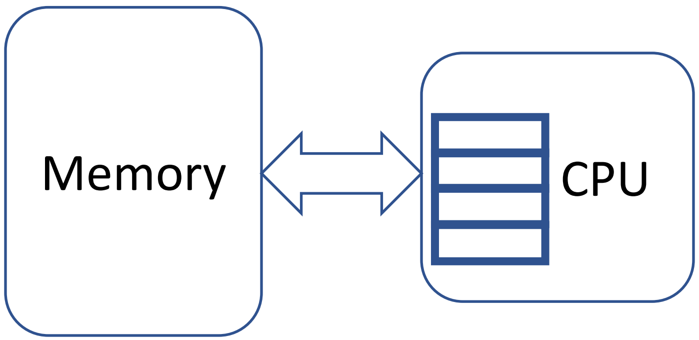
Idea:

We read {a, b} from memory **only once** and then use the Registers to sore all intermediate data.

What is the advantage of having registers inside CPU?

Consider this computation: c = a*b + a + b + (a-b)*a

Case 2: with registers

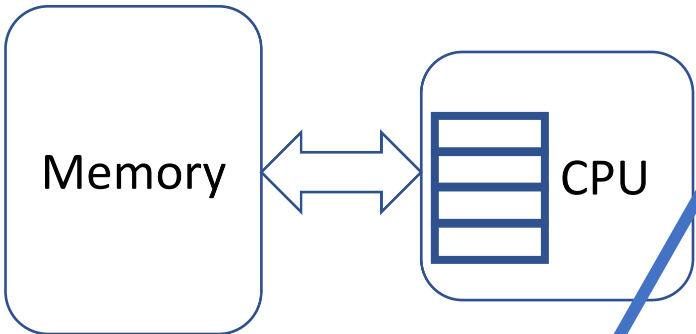


1. Read {a, b} from memory
and load them in {Register1, Register2}
2. Read {Register1, Register2},
compute c=a*b and store c in Register3
3. Read {Register3, Register1},
compute c=c+a and store c in Register3
4. Read {Register3, Register2},
compute c=c+b and store c in Register3
5. Read {Register1, Register2},
compute c'=a-b and store c' in Register4
6. Read {Register1, Register4},
compute c'=a*c' and store c' in Register4
7. Read {Register3, Register4},
compute c=c+c' and store c in Register3
8. Finally, copy c from Register3 to
Memory

What is the advantage of having registers inside CPU?

Consider this computation: c = a*b + a + b + (a-b)*a

Case 2: with registers



Pure arithmetic entirely **within** CPU.

Register reads or writes have zero-time overhead.

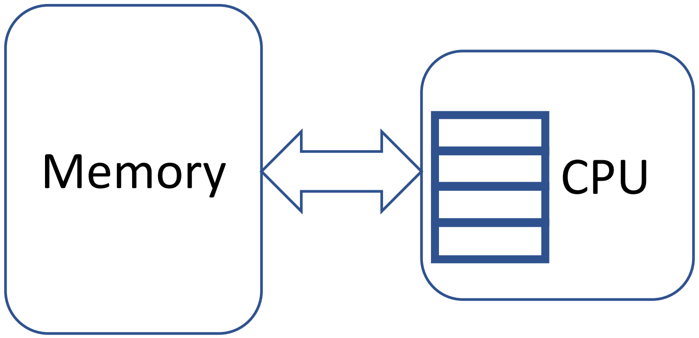
Only two memory read/write

1. Read {a, b} from memory and load them in {Register1, Register2}
2. Read {Register1, Register2}, compute c=a*b and store c in Register3
3. Read {Register3, Register1}, compute c=c+a and store c in Register3
4. Read {Register3, Register2}, compute c=c+b and store c in Register3
5. Read {Register1, Register2}, compute c'=a-b and store c' in Register4
6. Read {Register1, Register4}, compute c'=a*c' and store c' in Register4
7. Read {Register3, Register4}, compute c=c+c' and store c in Register3
8. Finally, copy c from Register3 to Memory

What is the advantage of having registers inside CPU?

Consider this computation: c = a*b + a + b + (a-b)*a

Case 2: with registers



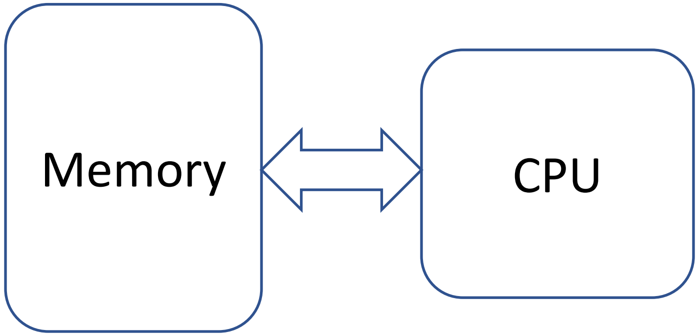
Total time requirement =
2*4 + 6*1 = 14 milliseconds

1. Read {a, b} from memory
and load them in {Register1, Register2}
2. Read {Register1, Register2},
compute c=a*b and store c in Register3
3. Read {Register3, Register1},
compute c=c+a and store c in Register3
4. Read {Register3, Register2},
compute c=c+b and store c in Register3
5. Read {Register1, Register2},
compute c'=a-b and store c' in Register4
6. Read {Register1, Register4},
compute c'=a*c' and store c' in Register4
7. Read {Register3, Register4},
compute c=c+c' and store c in Register3
8. Finally, copy c from Register3 to
Memory

What is the advantage of having registers inside CPU?

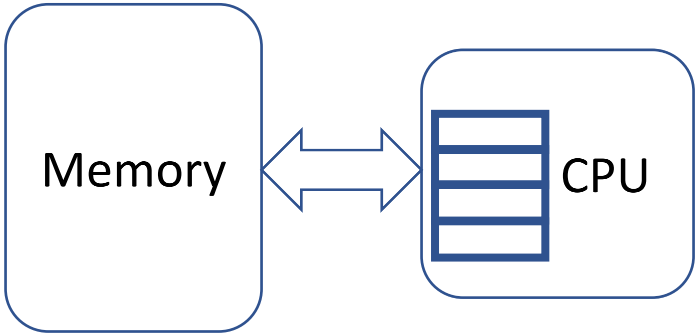
Consider this computation: c = a*b + a + b + (a-b)*a

Case 1: without registers



Total time requirement =
12*4 + 6*1 = 54 milliseconds

Case 2: with registers



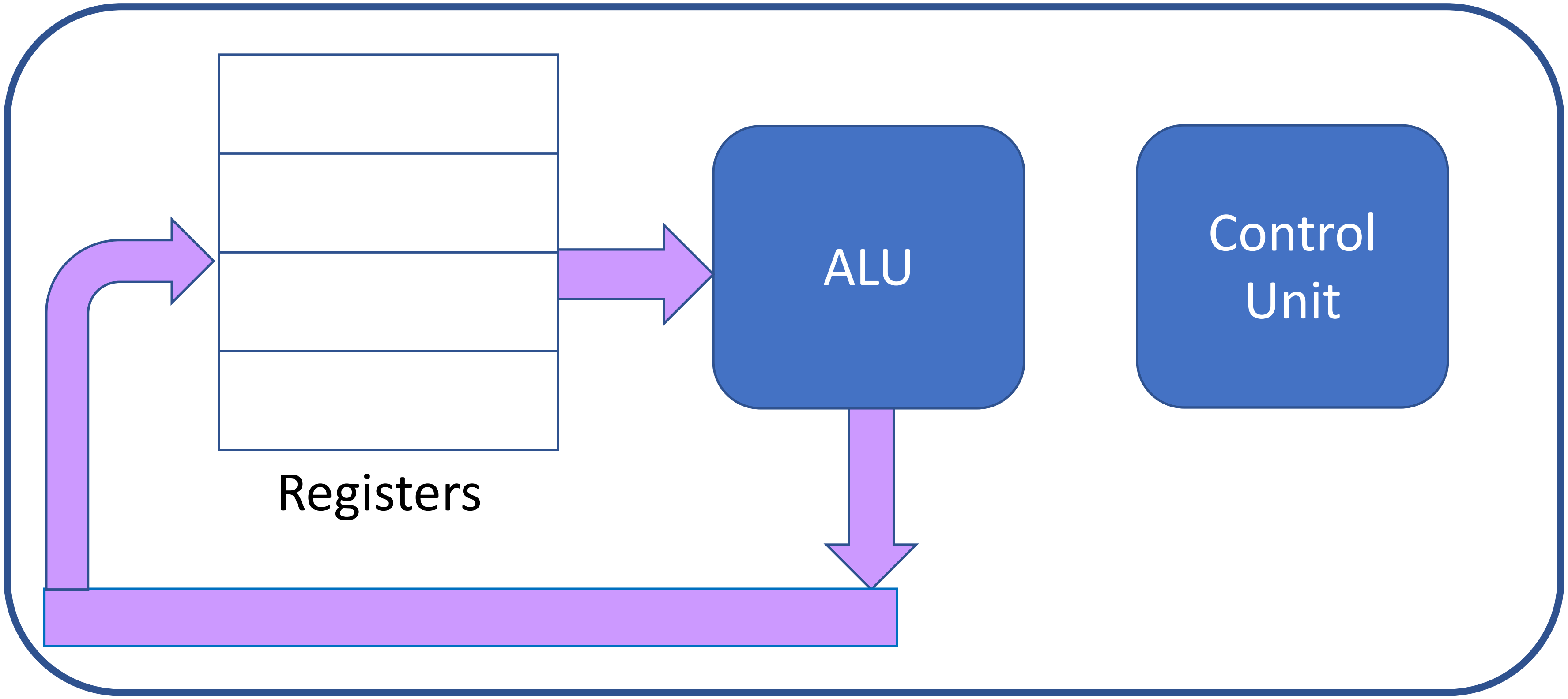
Total time requirement =
2*4 + 6*1 = 14 milliseconds

Conclusion: Registers improve processing speed

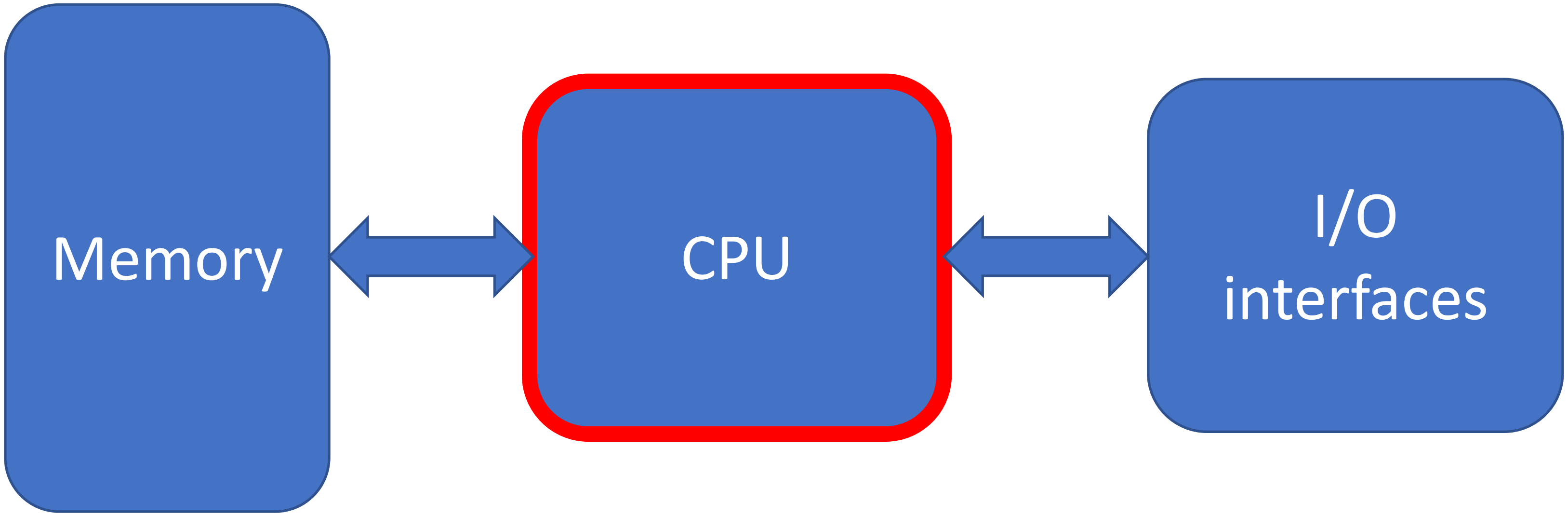
All present-day computers have Registers inside CPUs

Updated: Inside a CPU

So far, we have the following components inside a CPU
- Arithmetic and Logic Unit (ALU)
- Registers (new component for improving performance)
- Control Unit



Recap: The von Neumann Architecture

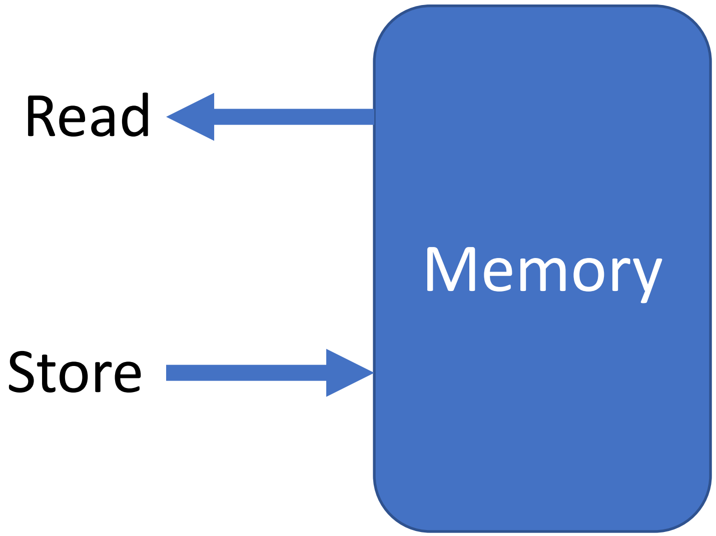


Consists of three main components

1. **Central Processing Unit (CPU)** (we have covered the CPU)
2. Memory (our next topic)
3. Input/Output interfaces.

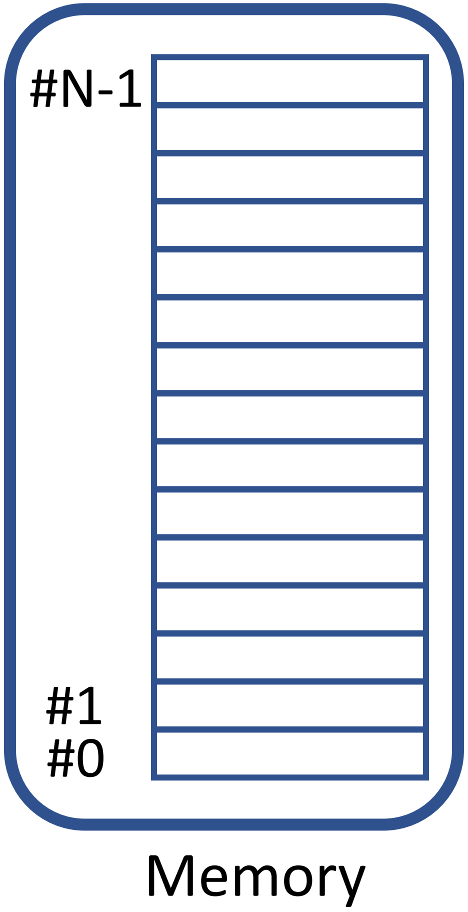
Organization of Memory

Memory



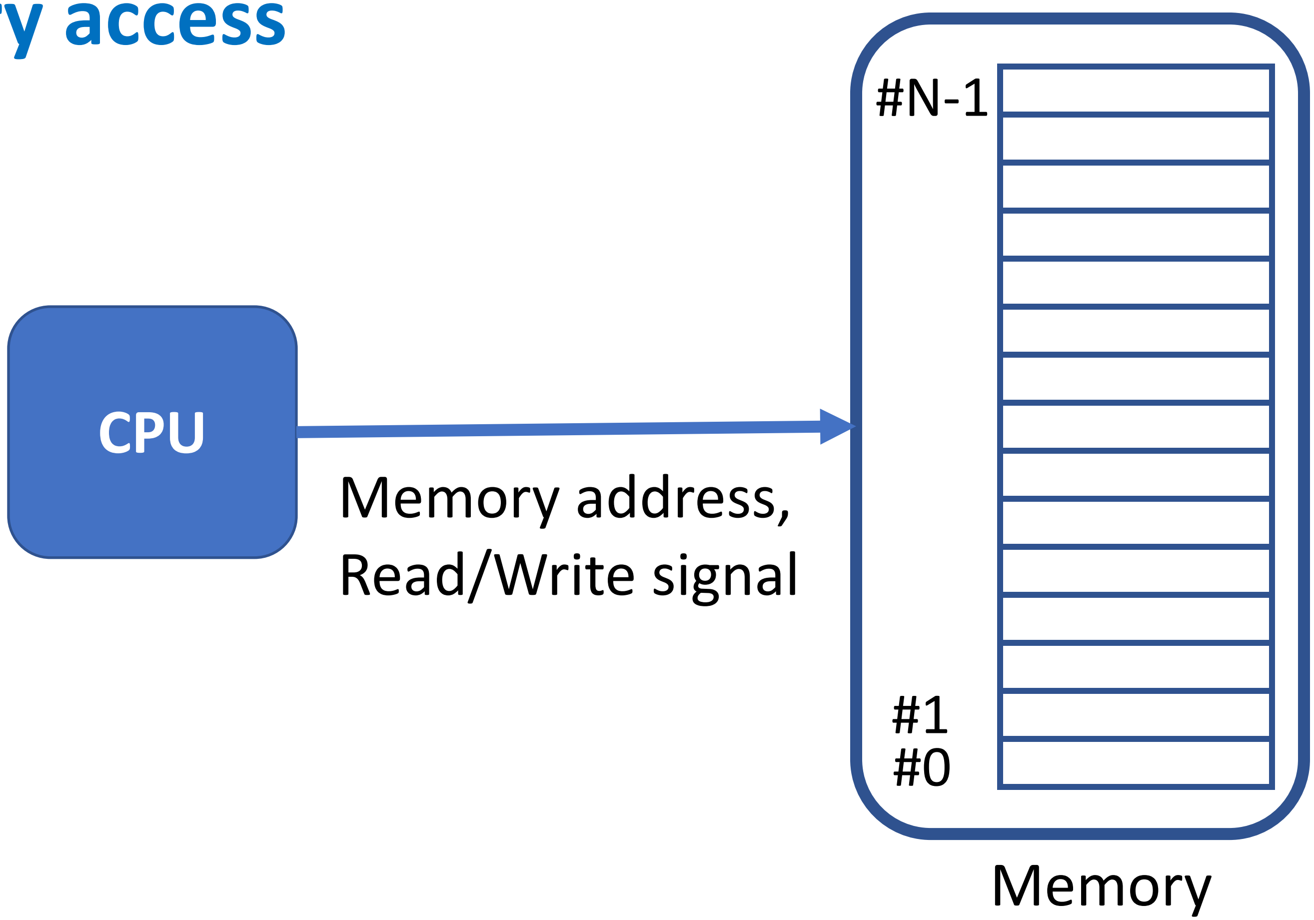
Programmer sees memory as a storage element.

Programmer's View: Memory as an addressable storage



- Memory consists of small 'cells'
- Each cell can store a small piece of data
- The cells have addresses. E.g. 0 to N-1

Memory access



- During Read and Store operations, CPU generates memory addresses.
- Memory Management Unit (MMU) reads or writes from/to the requested memory-location.

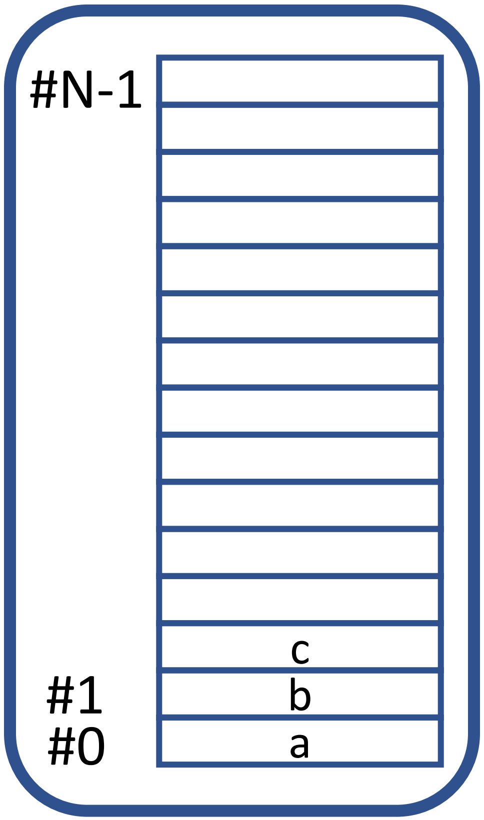
Example: Memory access during program exe.

```
main(){

    read a;
    read b;
    c = a + b;
    store c;
}
```

High-level code

Assume that programmer or compiler has allocated variables ‘a’, ‘b’ and ‘c’ in the memory locations with address #0, #1 and #2.



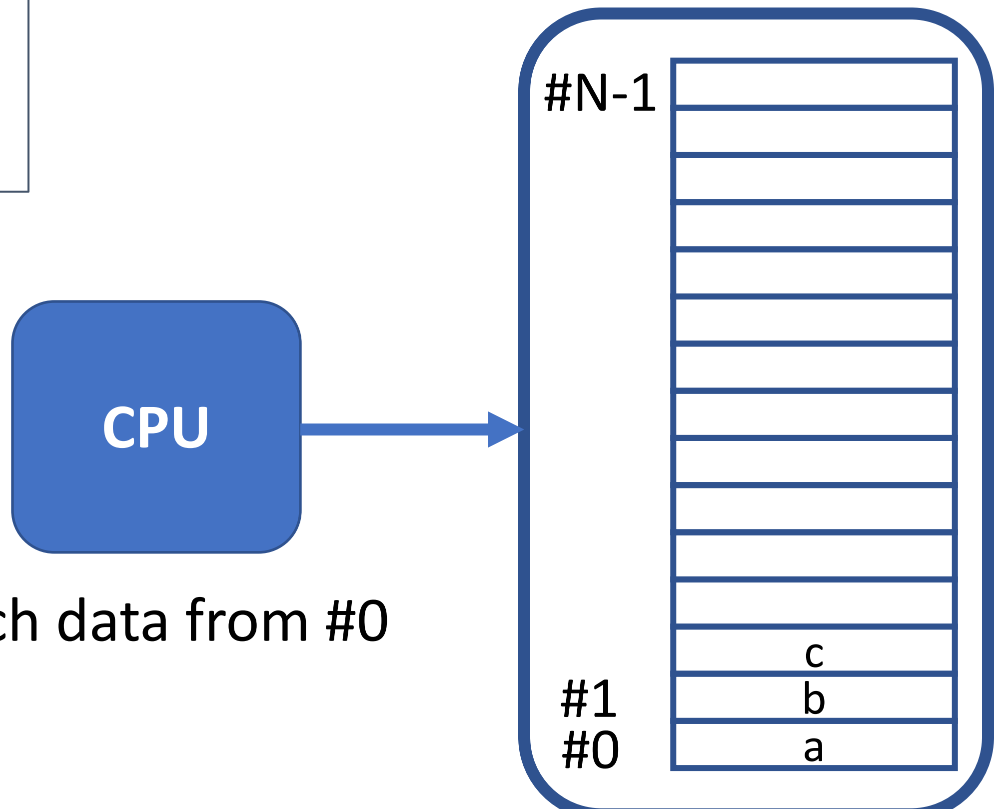
Example: Memory access during program exe.

```
main(){

    read a;
    read b;
    c = a + b;
    store c;
}
```

High-level code

Assume that programmer or compiler has allocated variables ‘a’, ‘b’ and ‘c’ in the memory locations with address #0, #1 and #2.



Steps:

1. CPU asks MMU to fetch data from #0

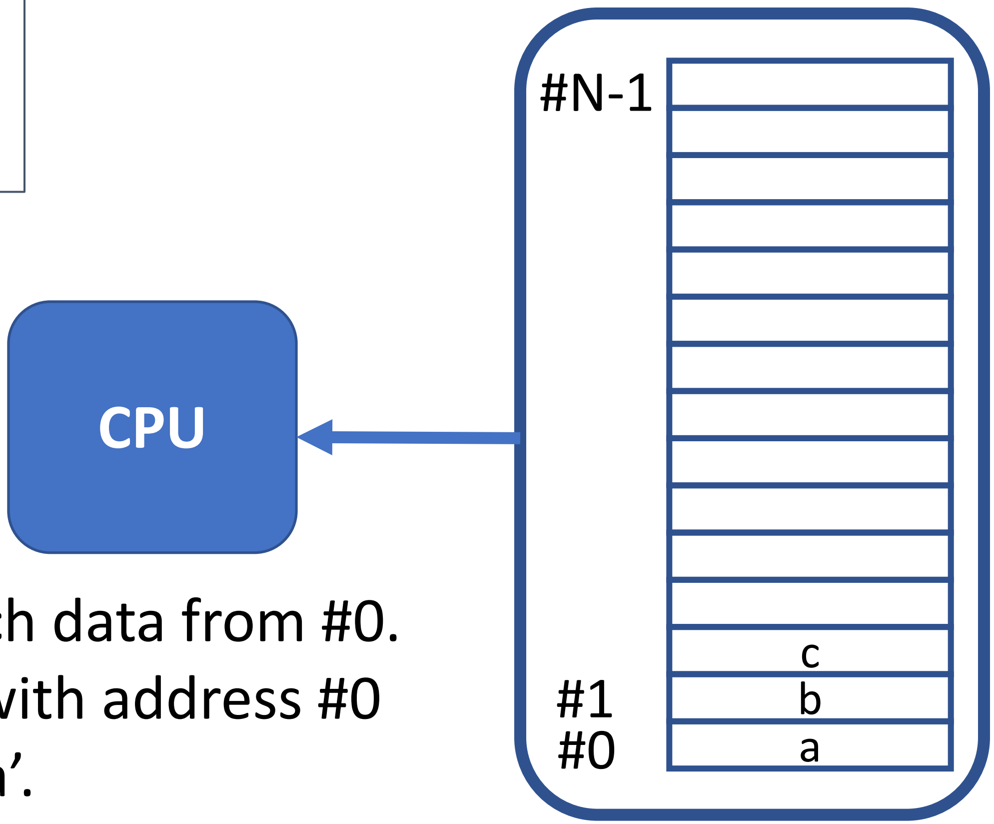
Example: Memory access during program exe.

```
main(){

    read a;
    read b;
    c = a + b;
    store c;
}
```

High-level code

Assume that programmer or compiler has allocated variables ‘a’, ‘b’ and ‘c’ in the memory locations with address #0, #1 and #2.



Steps:

1. CPU asks MMU to fetch data from #0.
2. MMU reads location with address #0 and returns value of ‘a’.

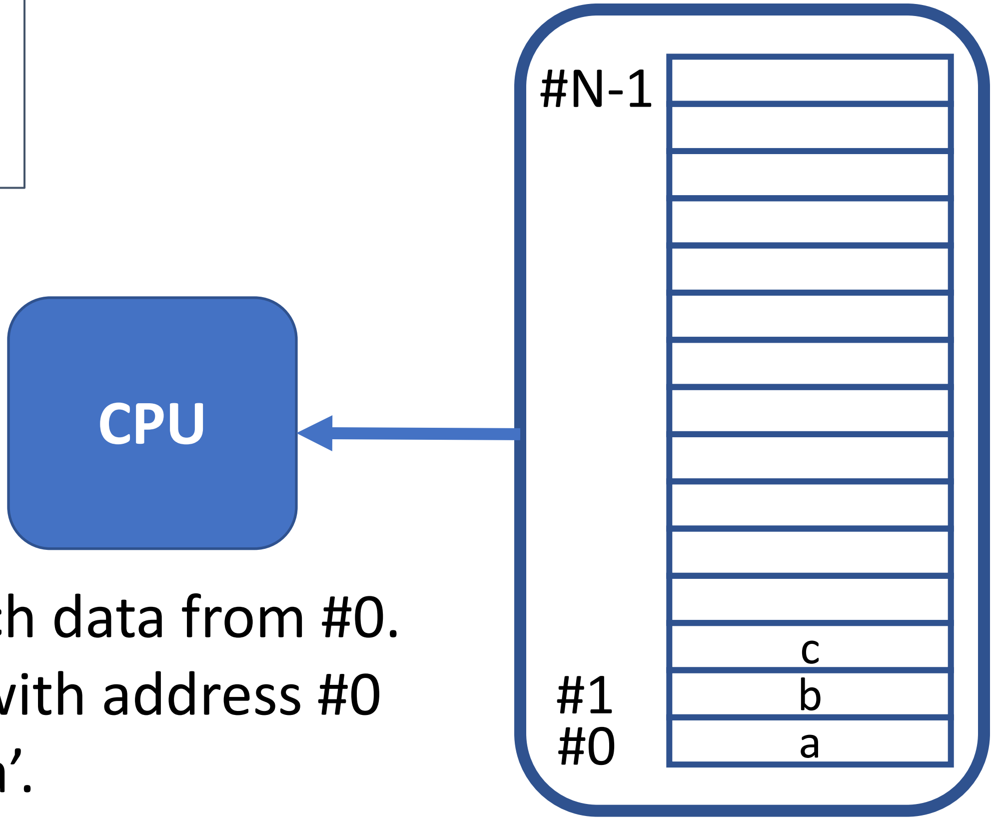
Example: Memory access during program exe.

```
main(){

        read a;
        read b;
        c = a + b;
        store c;
}
```

High-level code

Assume that programmer or compiler has allocated variables ‘a’, ‘b’ and ‘c’ in the memory locations with address #0, #1 and #2.



Steps:

1. CPU asks MMU to fetch data from #0.
2. MMU reads location with address #0 and returns value of ‘a’.
3. Similar steps for ‘read b’

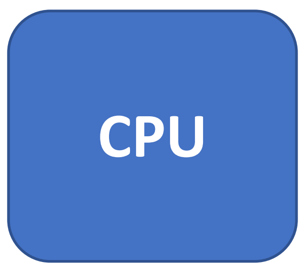
Example: Memory access during program exe.

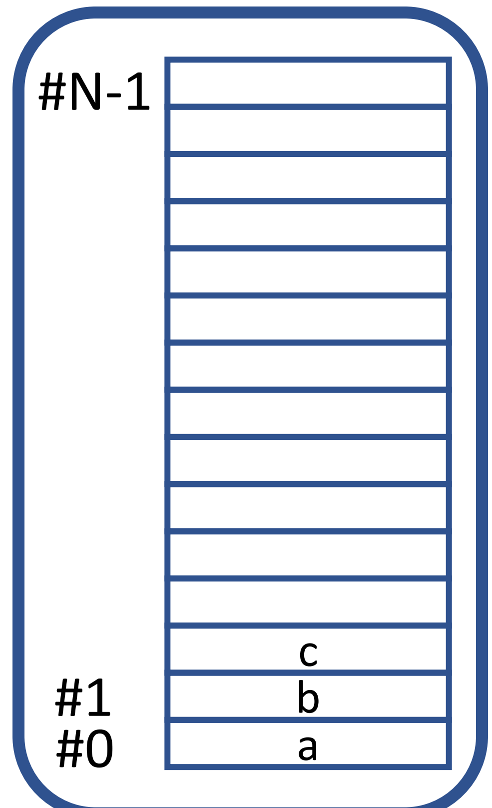
```
main(){

      read a;
      read b;
      c = a + b;
      store c;
}
```

High-level code

Assume that programmer or compiler has allocated variables 'a', 'b' and 'c' in the memory locations with address #0, #1 and #2.





Steps:

4. CPU computes sum 'c'

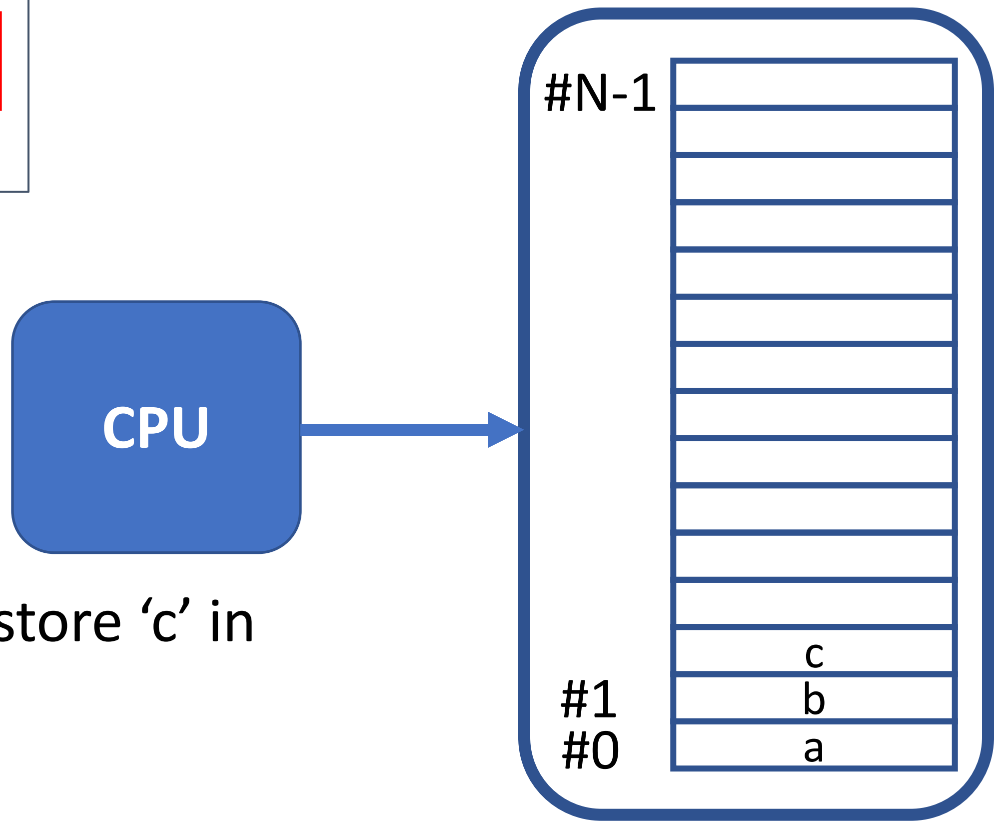
Example: Memory access during program exe.

```
main(){

      read a;
      read b;
      c = a + b;
      store c;
}
```

High-level code

Assume that programmer or compiler has allocated variables ‘a’, ‘b’ and ‘c’ in the memory locations with address #0, #1 and #2.



Steps:

5. CPU instructs MMU to store ‘c’ in location with address #2.
6. MMU writes ‘c’ at #2

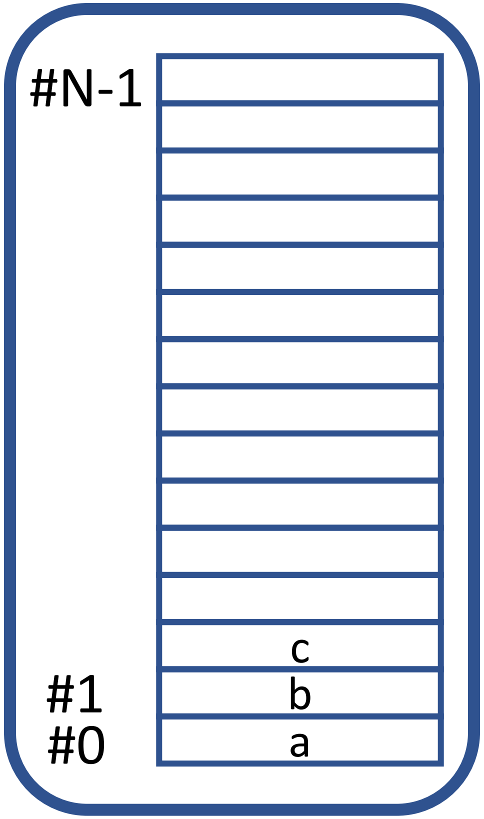
Example: Memory access during program exe.

```
main(){

     read a;
     read b;
     c = a + b;
     store c;
}
```

High-level code

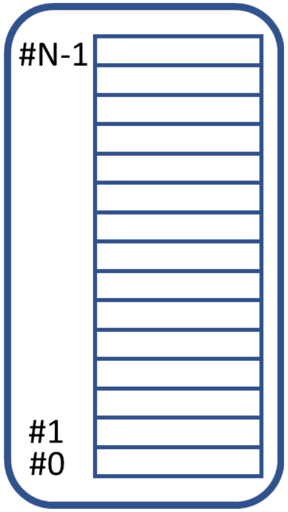
In **C programming**, you will learn how to work with memory addresses using **Pointers**.



Conclusions

- We have studied von Neumann architecture.

- Programmer sees memory as a storage element.
    - Memory consists of small ‘cells’
    - Each cell can store a small piece of data
    - The cells have addresses. E.g. 0 to N-1



Memory