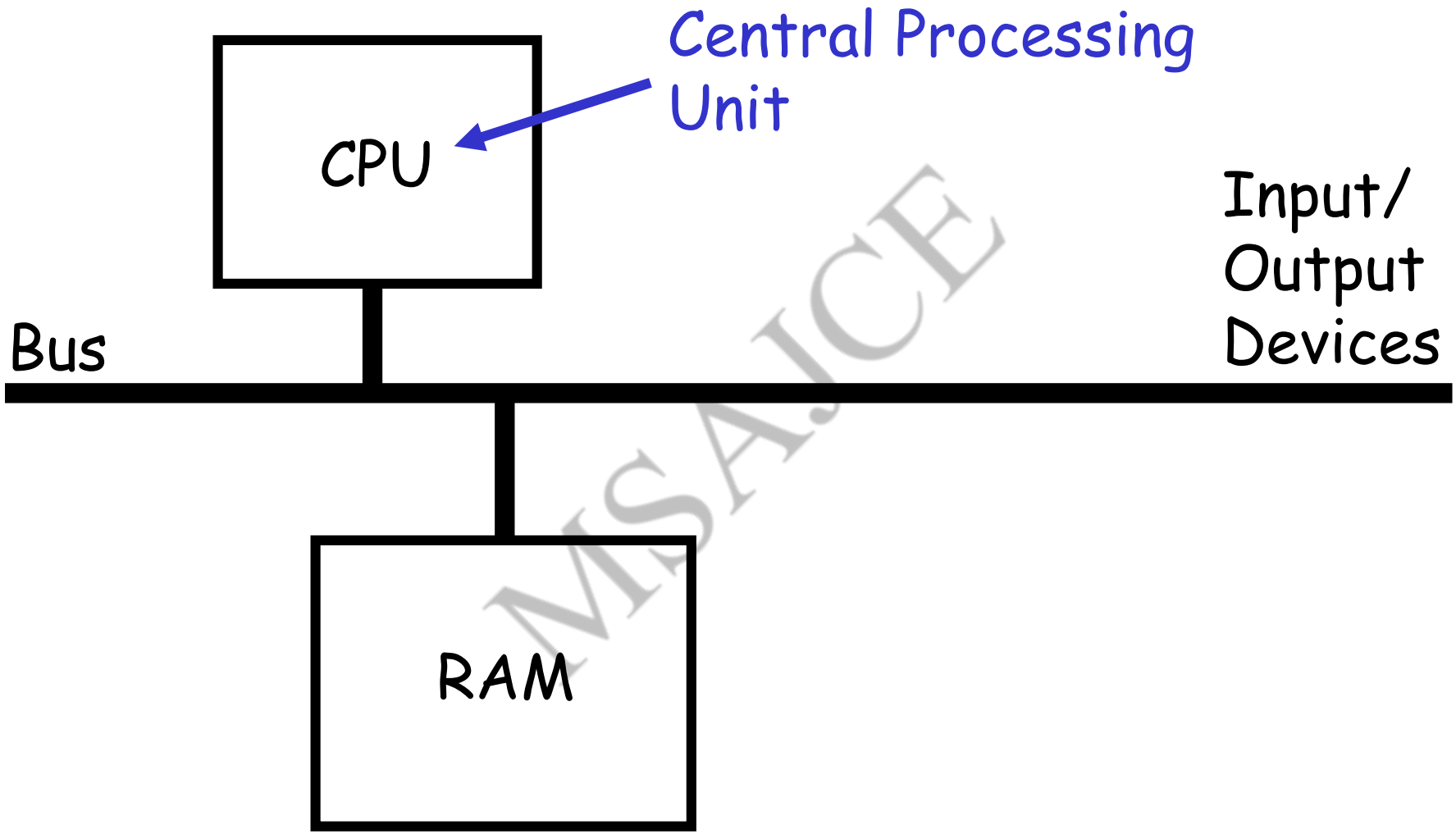
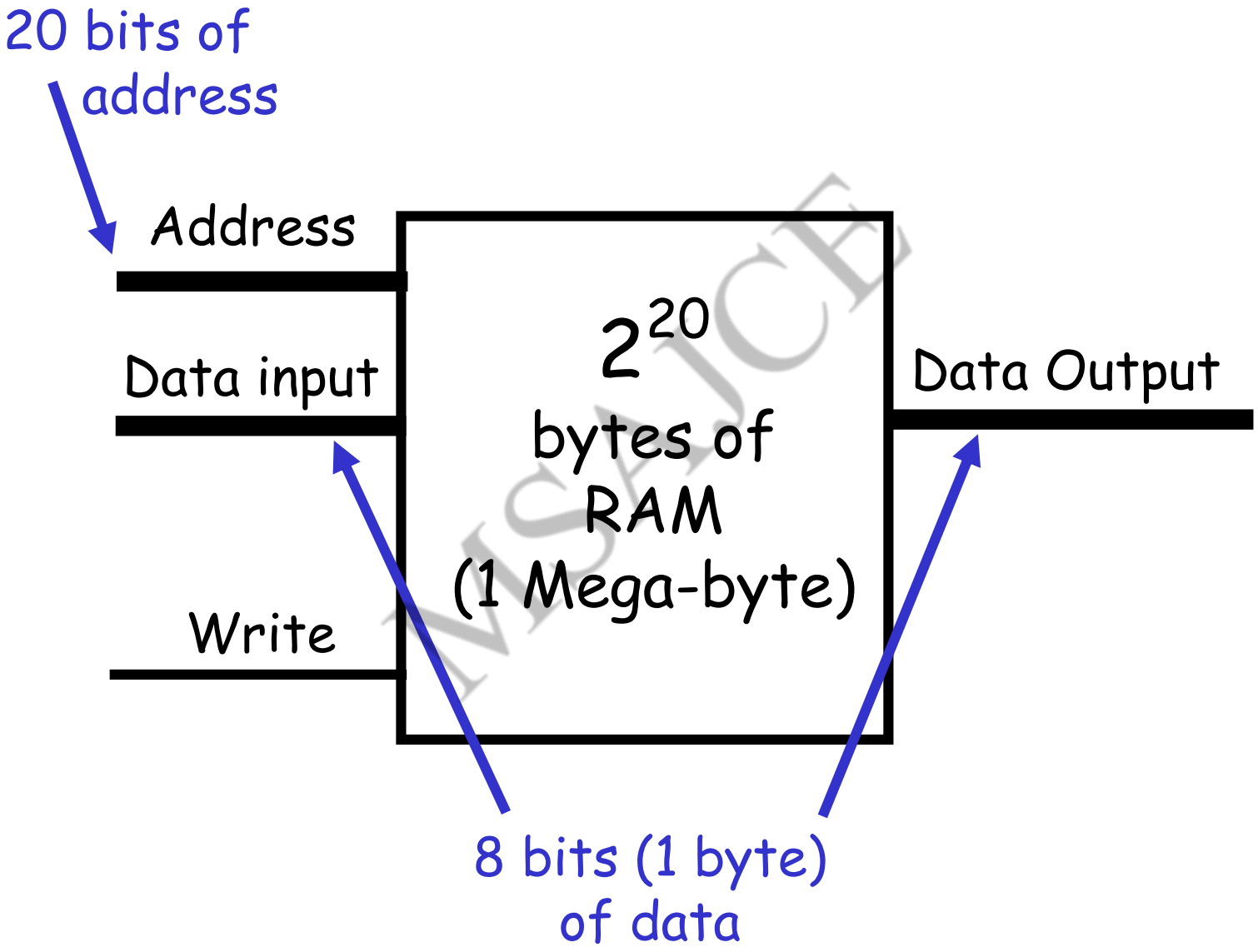
COMPUTER ARCHITECTURE

MSAJCE

Computer Architecture

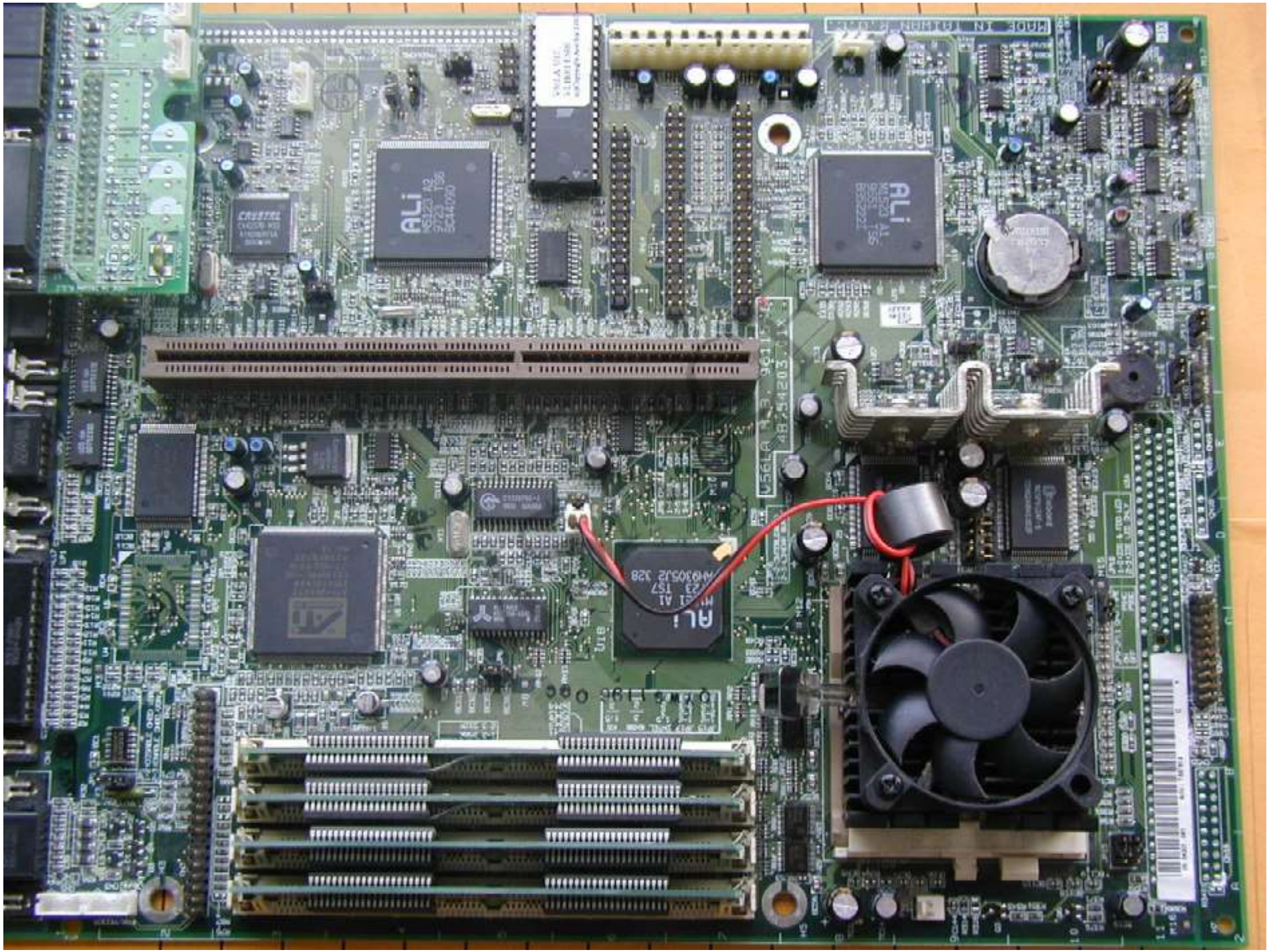


RAM (cont.)

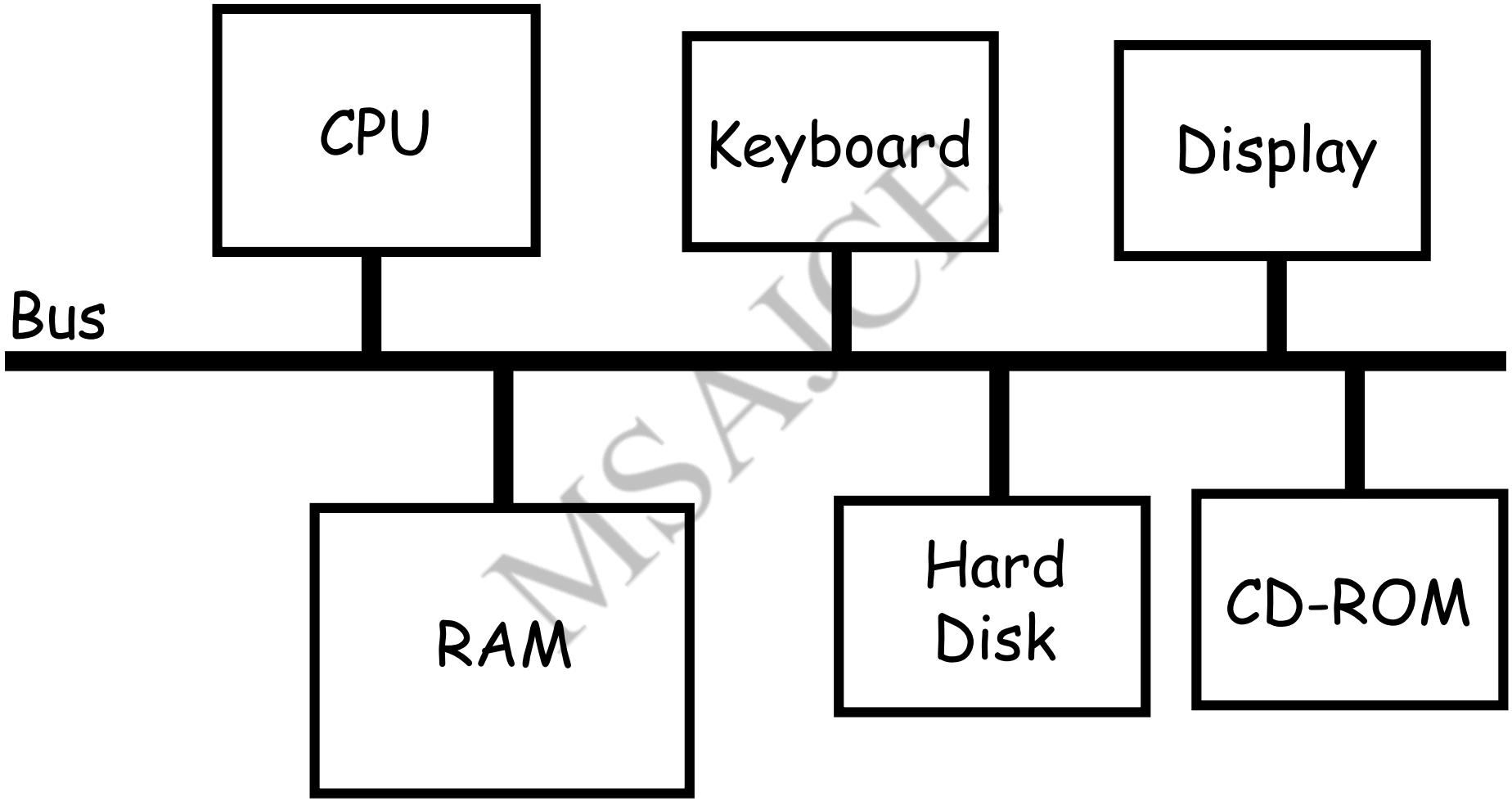


RAM (cont.)

- When you talk about the memory of a computer, most often you're talking about its RAM.
- If a program is stored in RAM, that means that a sequence of instructions are stored in consecutively addressed bytes in the RAM.
- Data values (variables) are stored anywhere in RAM, not necessarily sequentially
- Both instructions and data are accessed from RAM using addresses
- RAM is one (crucial) part of the computer's overall architecture

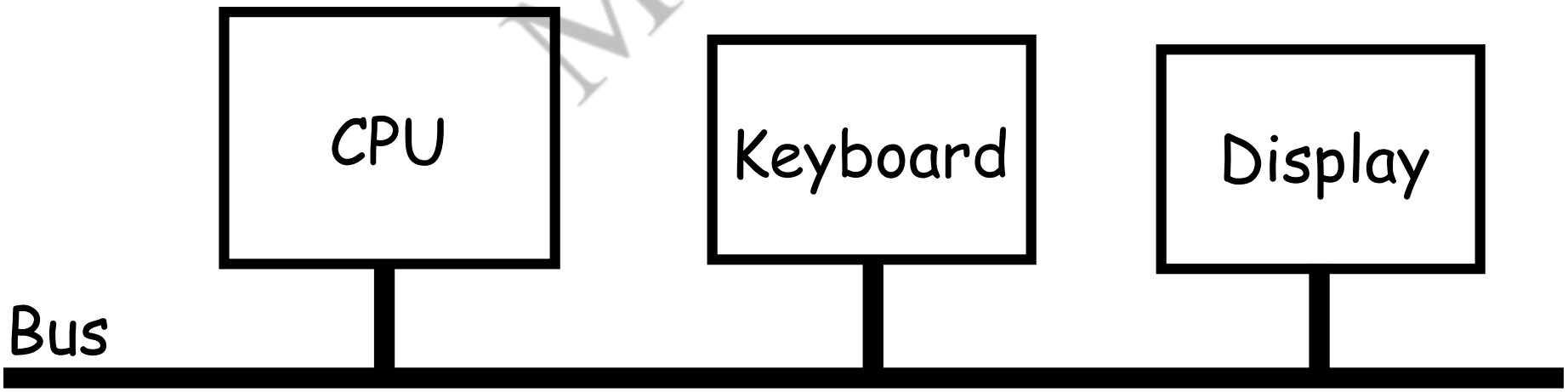

ALi
ALi

Computer Architecture

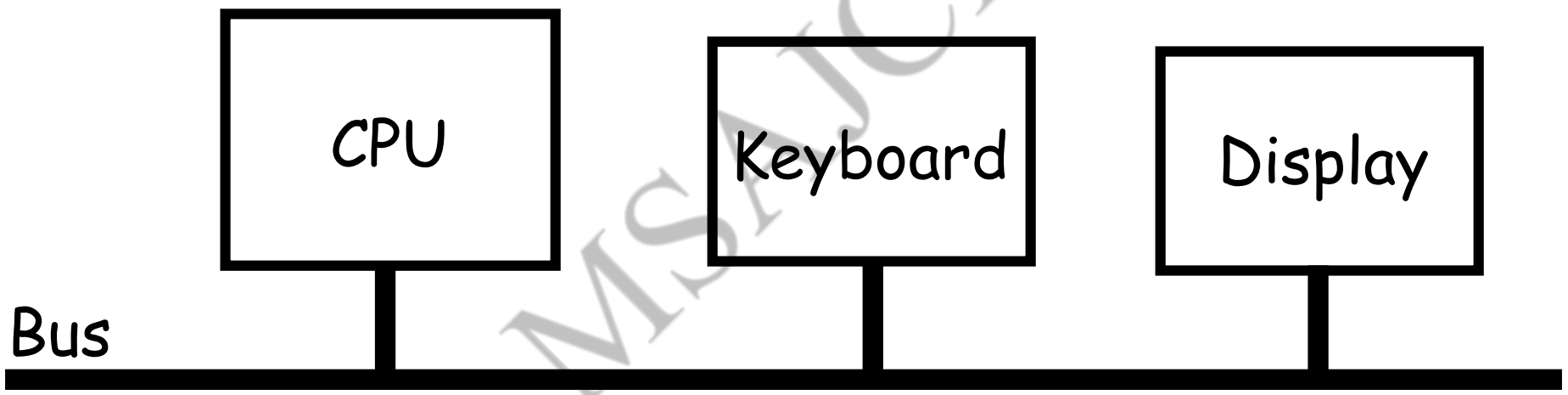


The Bus

- What is a bus?
- It is a simplified way for many devices to communicate to each other.
- Looks like a "highway" for information.
- Actually, more like a "basket" that they all share.

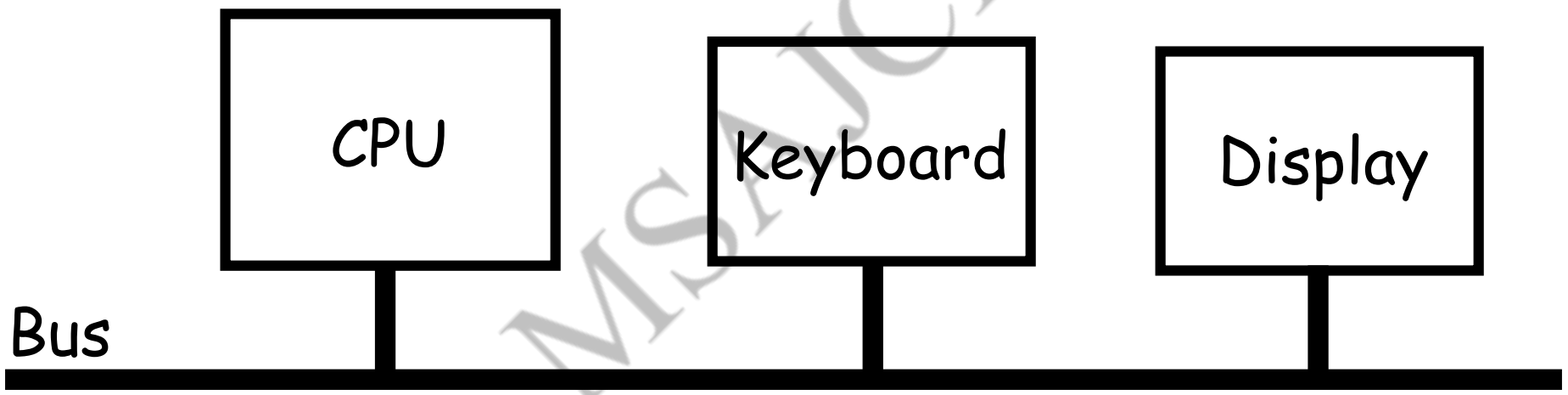


The Bus



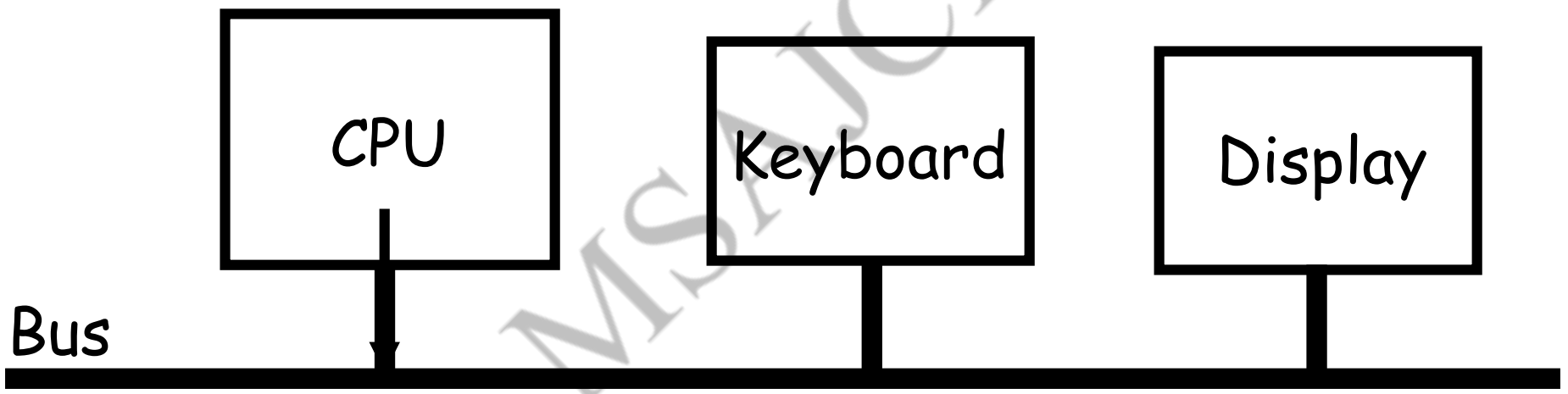
The Bus

- Suppose CPU needs to check to see if the user typed anything.



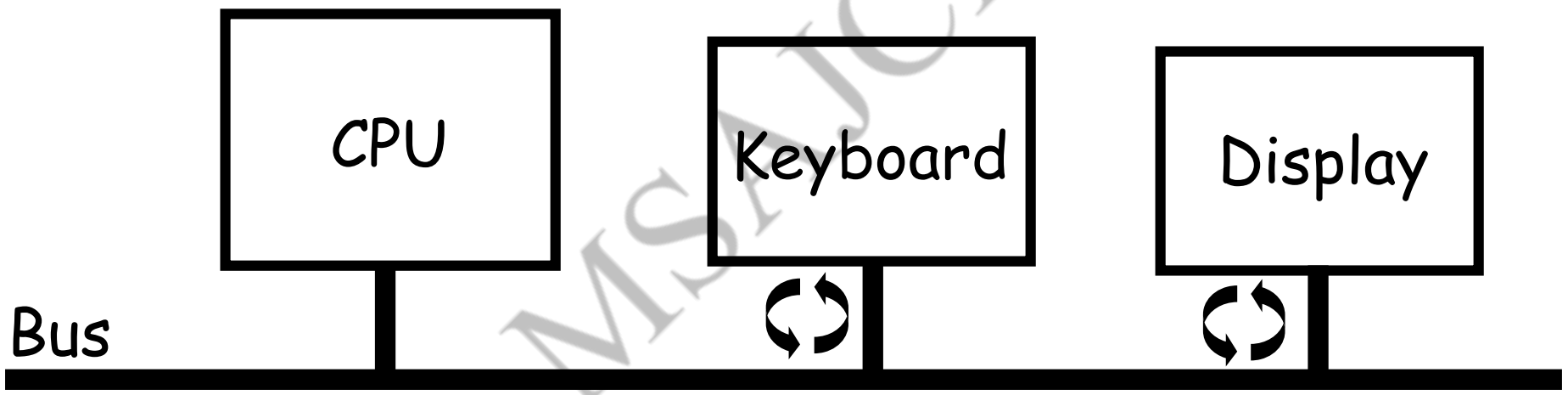
The Bus

- CPU puts "Keyboard, did the user type anything?" (represented in some way) on the Bus.



The Bus

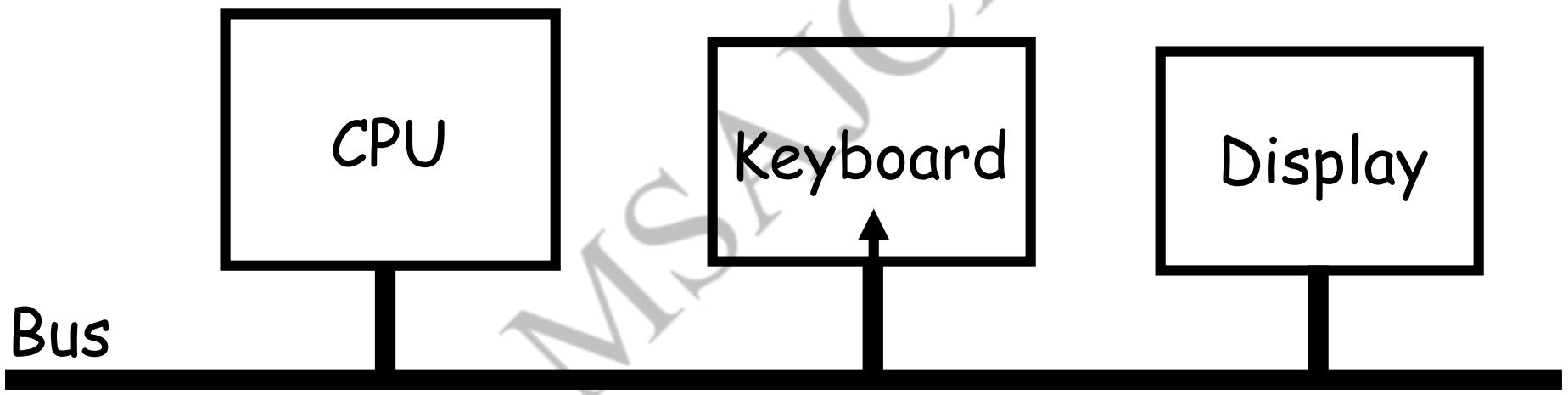
- Each device (except CPU) is a State Machine that constantly checks to see what's on the Bus.



"Keyboard, did the user type anything?"

The Bus

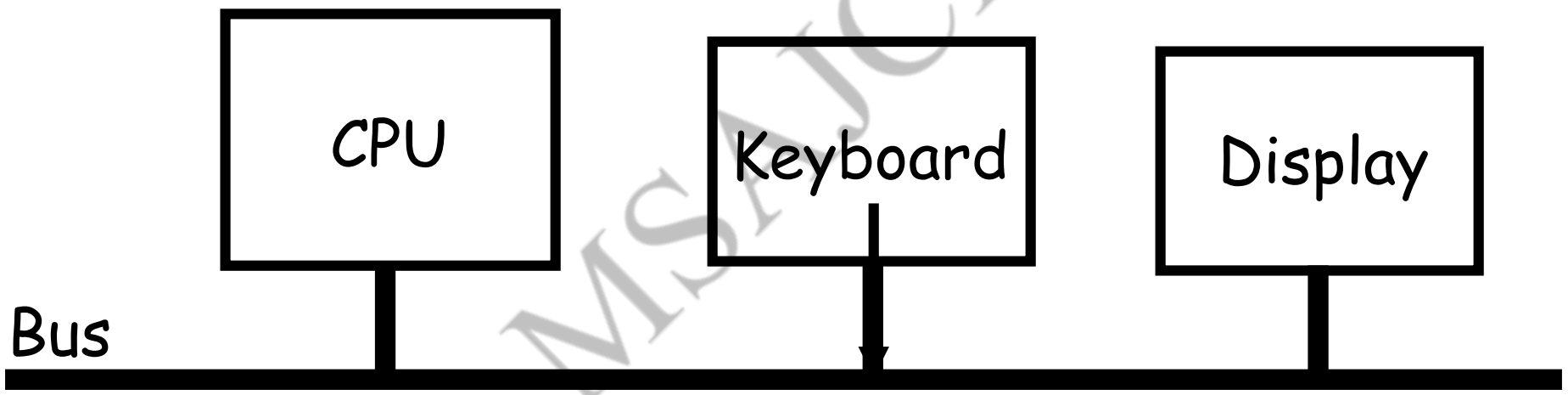
- Keyboard notices that its name is on the Bus, and reads info. Other devices ignore the info.



"Keyboard, did the user type anything?"

The Bus

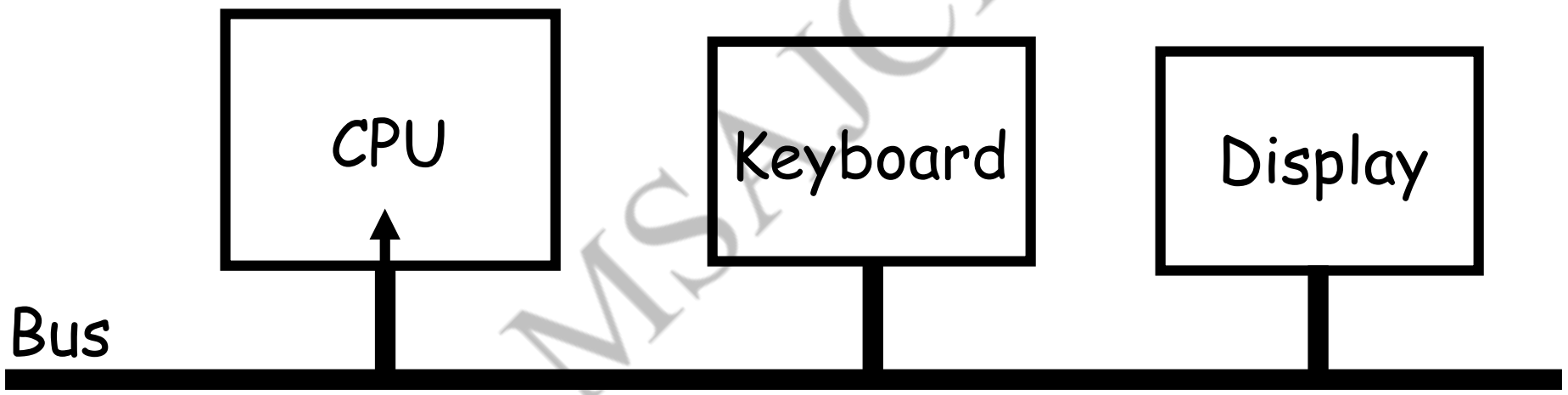
- Keyboard then writes "CPU: Yes, user typed 'a'." to the Bus.



"CPU: Yes, user typed 'a'."

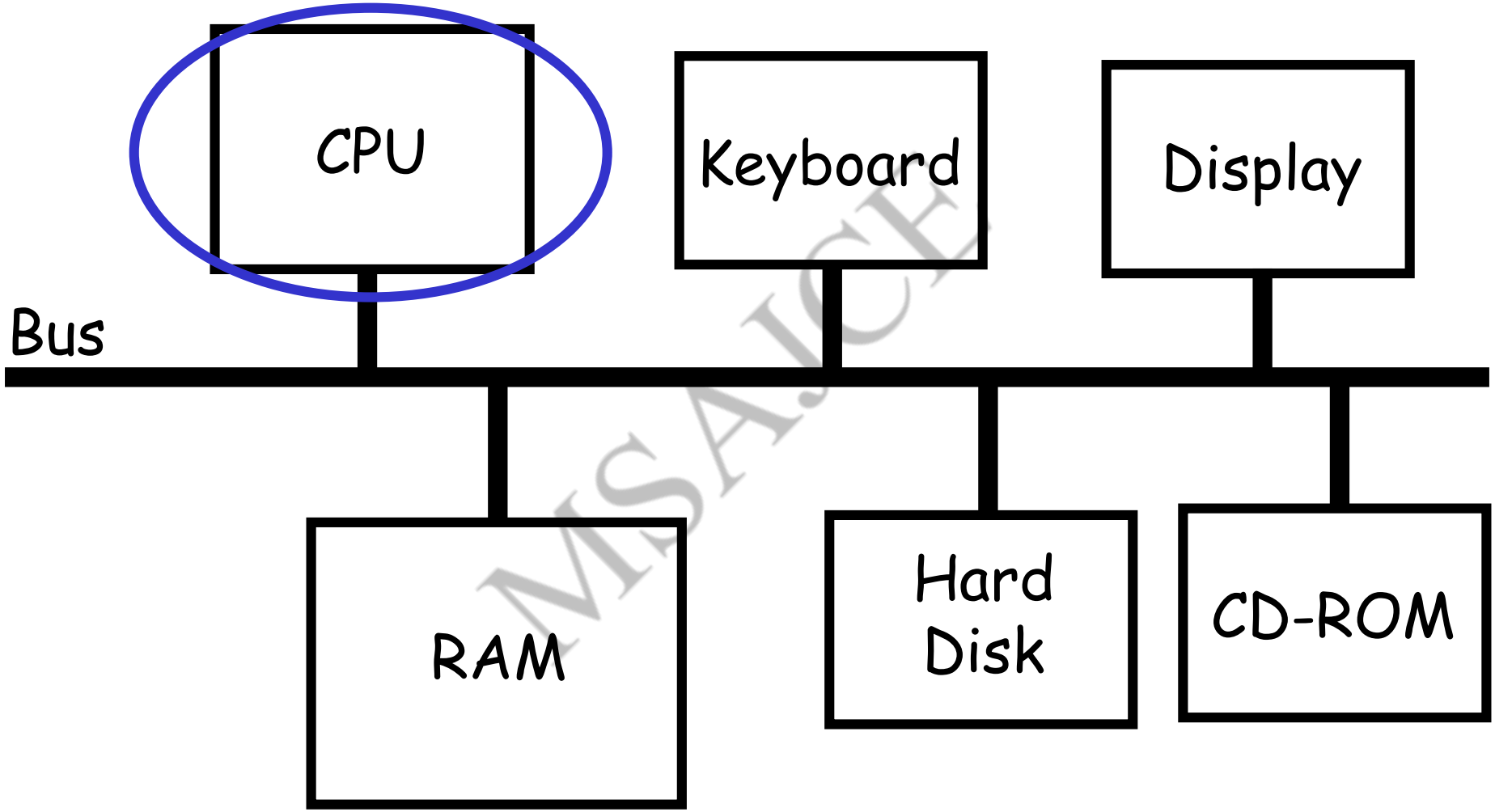
The Bus

- At some point, CPU reads the Bus, and gets the Keyboard's response.



"CPU: Yes, user typed 'a'."

Computer Architecture



Inside the CPU

- The CPU is the brain of the computer.
- It is the part that actually executes the instructions.

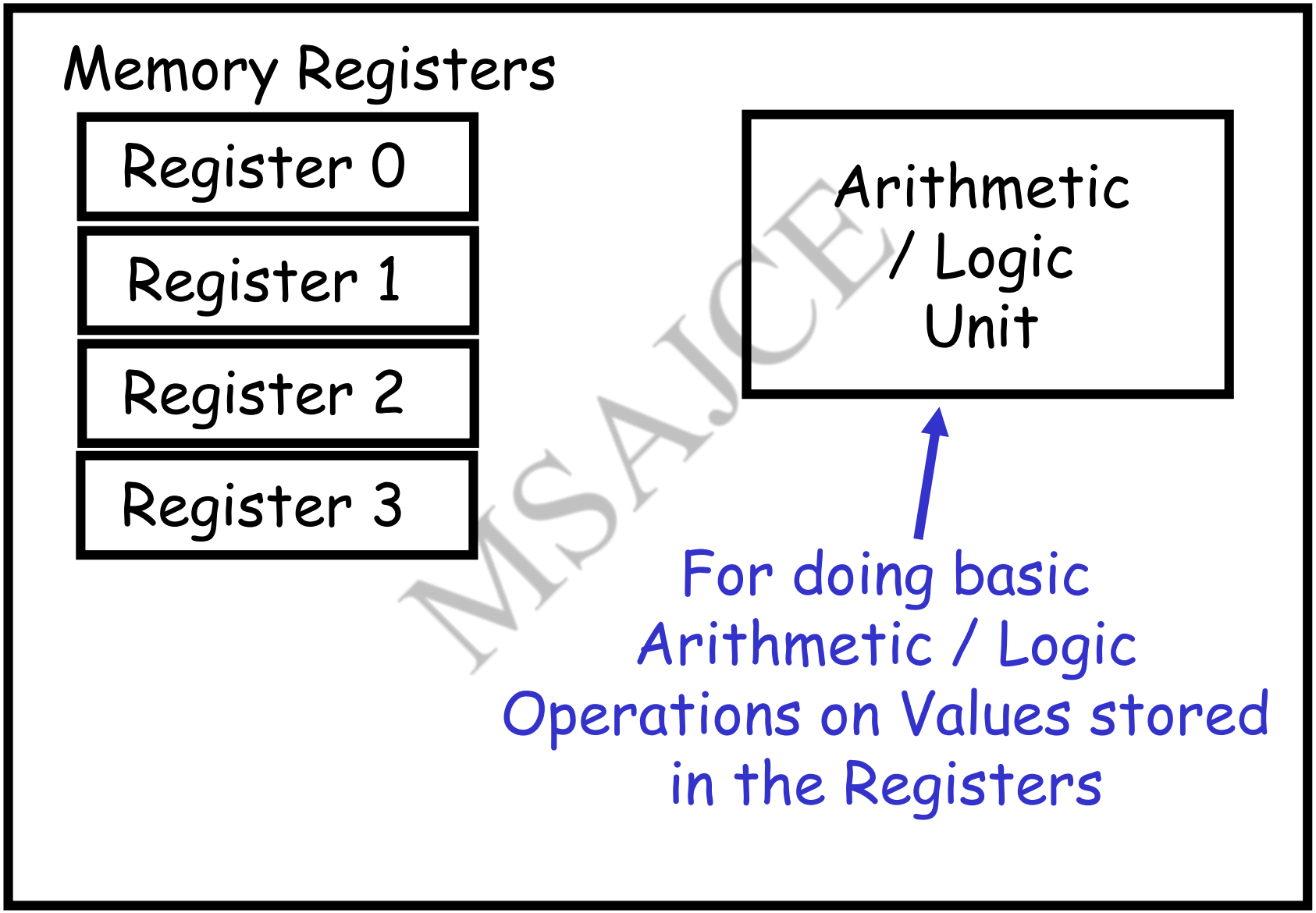
- Let's take a look inside.

Inside the CPU (cont.)

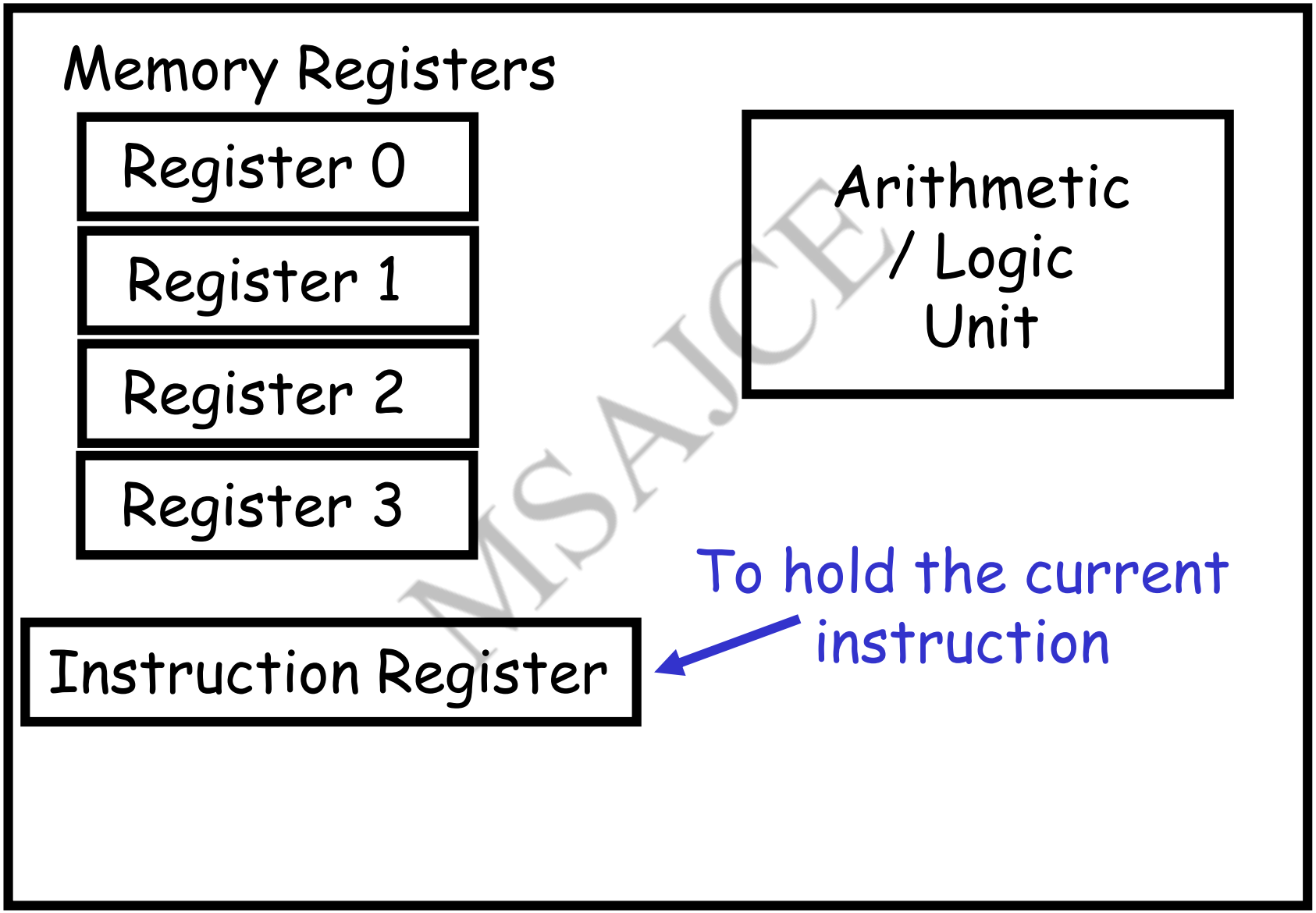
Temporary Memory. Computer "Loads" data from RAM to registers, performs operations on data in registers, and "stores" results from registers back to RAM

Remember our initial example: "read value of A from memory; read value of B from memory; add values of A and B; put result in memory in variable C." The reads are done to registers, the addition is done in registers, and the result is written to memory from a register.

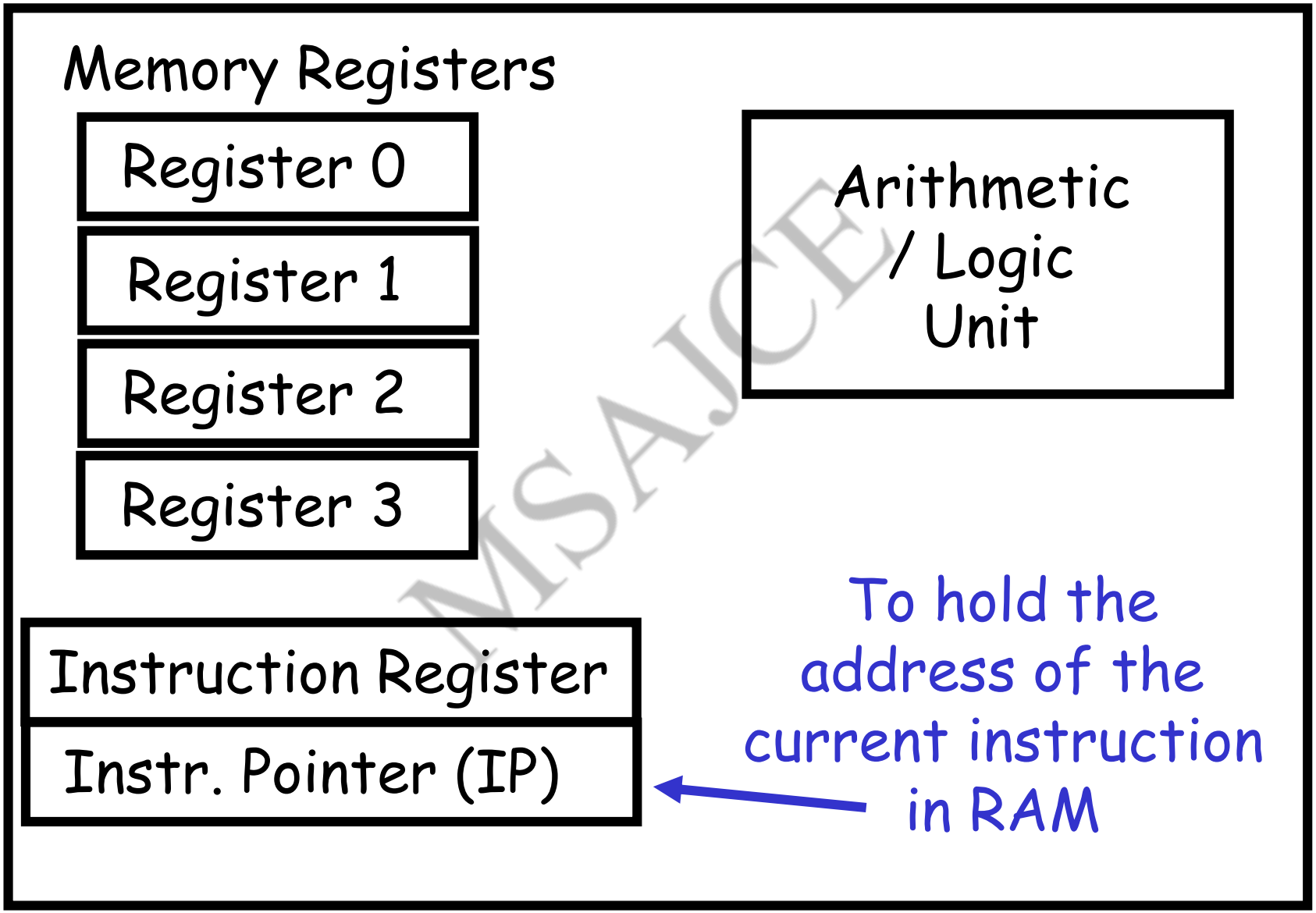
Inside the CPU (cont.)



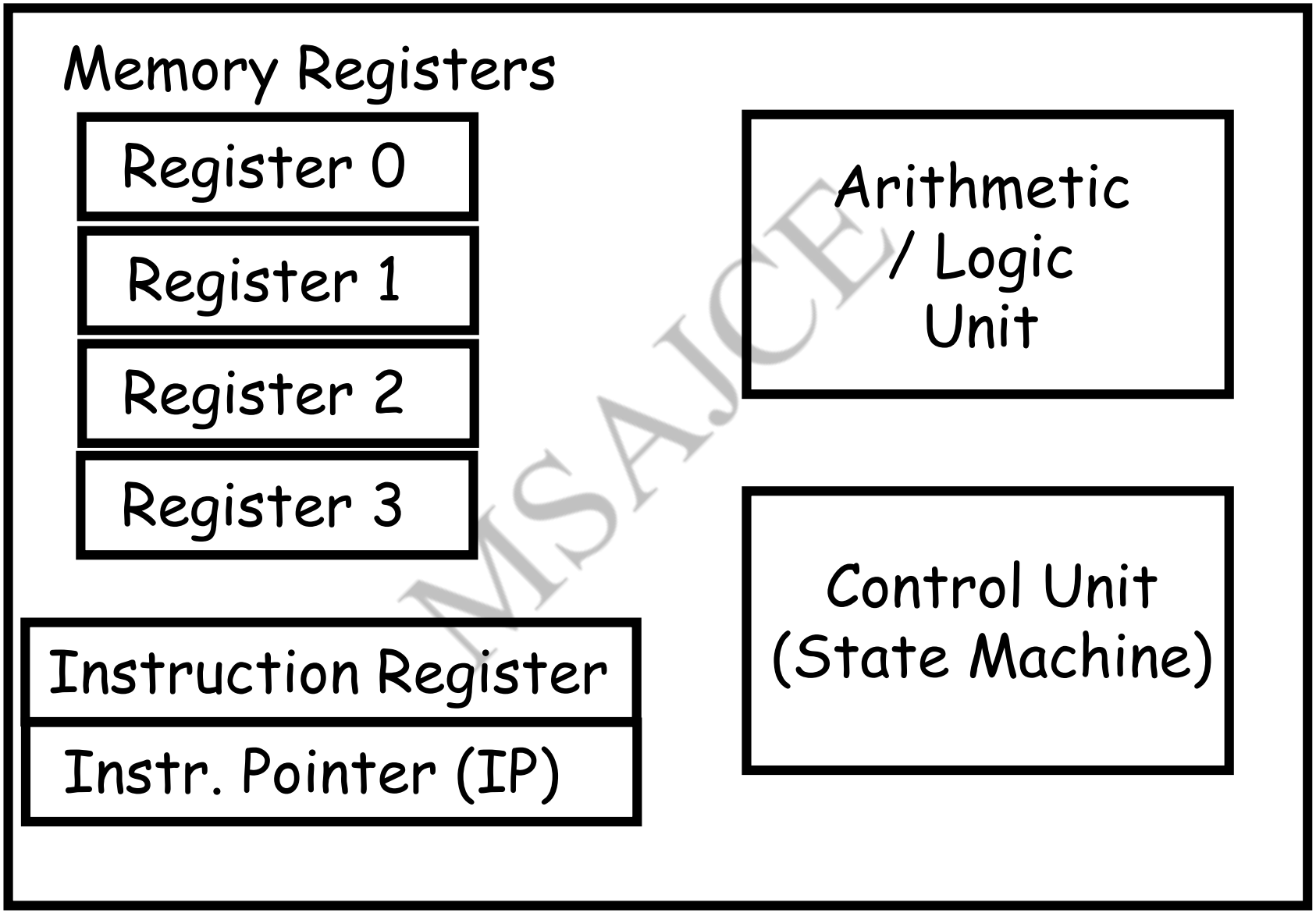
Inside the CPU (cont.)



Inside the CPU (cont.)



Inside the CPU (cont.)



The Control Unit

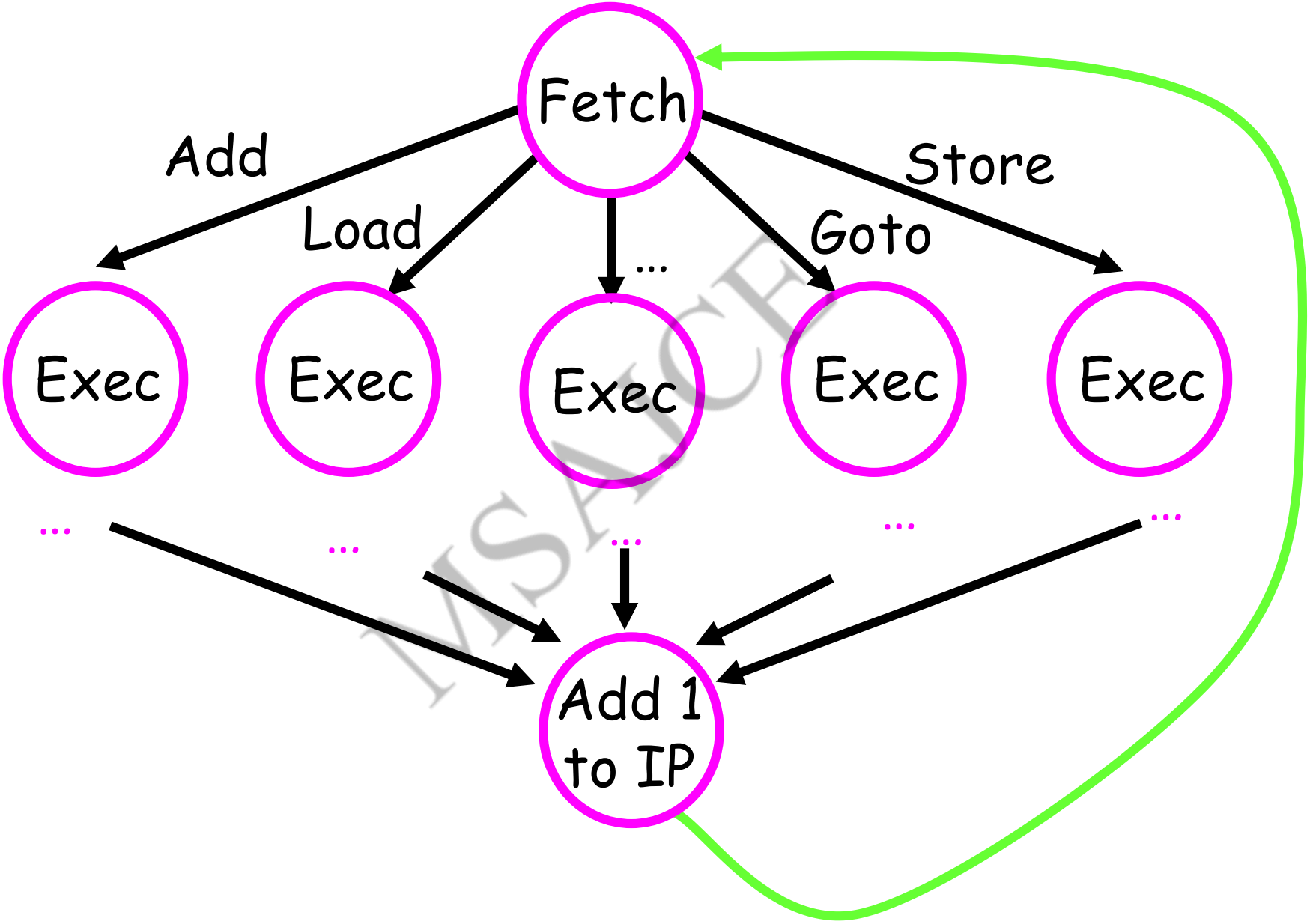
- It all comes down to the Control Unit.
- This is just a State Machine.

- How does it work?

The Control Unit

- Control Unit State Machine has very simple structure:
  - 1) Fetch: Ask the RAM for the instruction whose address is stored in IP.
  - 2) Execute: There are only a small number of possible instructions. Depending on which it is, do what is necessary to execute it.
  - 3) Repeat: Add 1 to the address stored in IP, and go back to Step 1 !

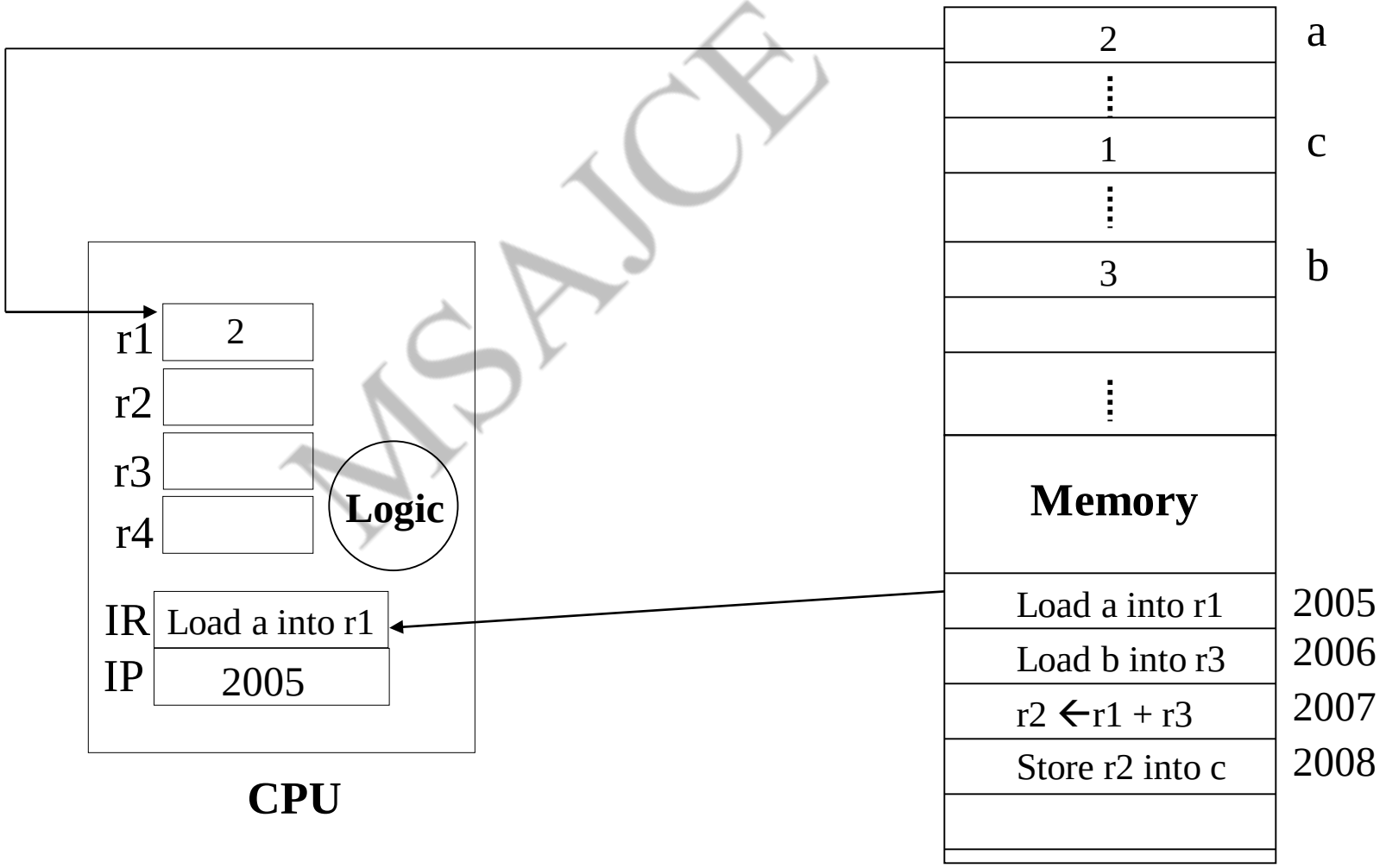
The Control Unit is a State Machine



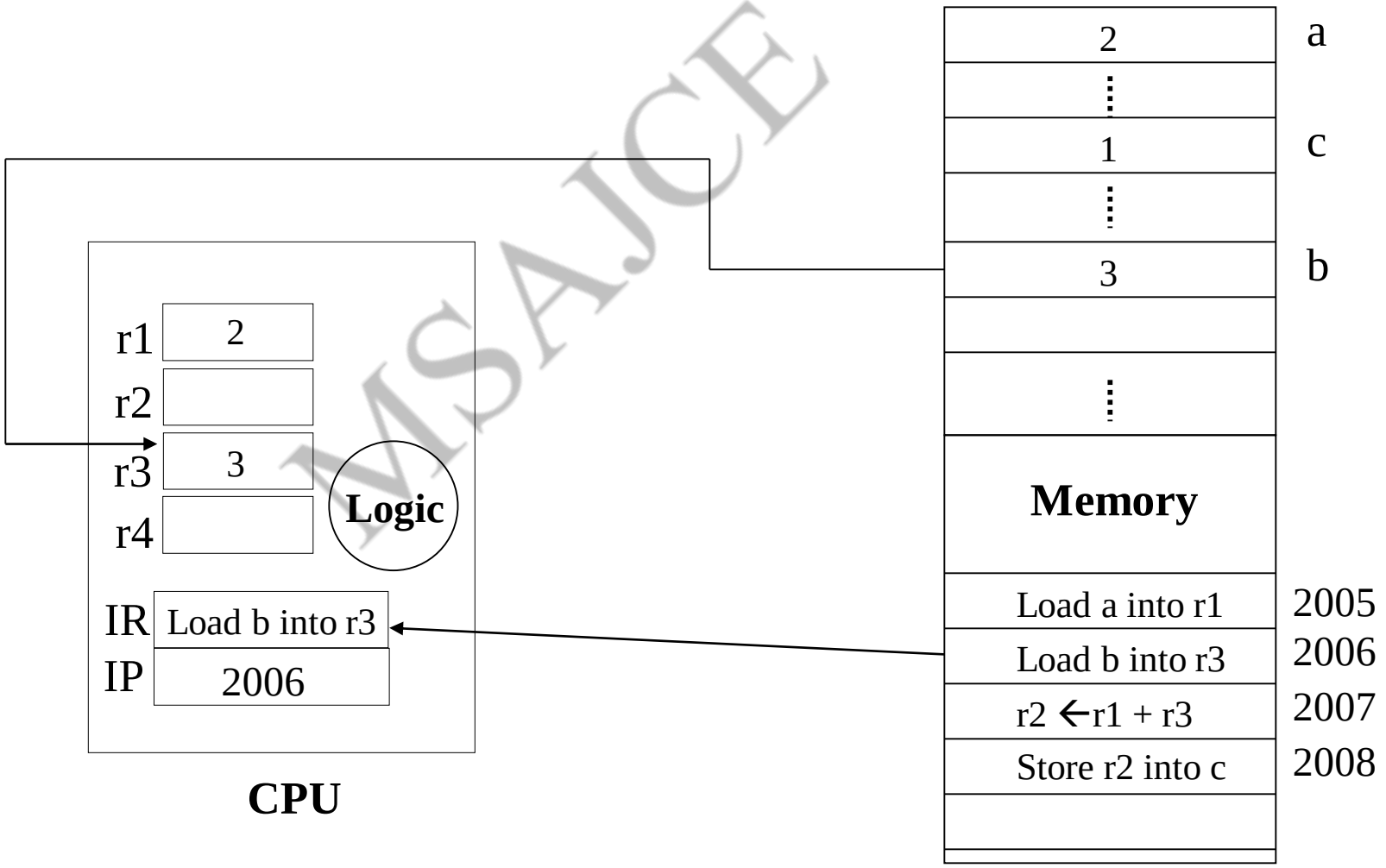
A Simple Program

- Want to add values of variables a and b (assumed to be in memory), and put the result in variable c in memory, I.e. c ← a+b
- Instructions in program
  - Load a into register r1
  - Load b into register r3
  - r2 ← r1 + r3
  - Store r2 in c

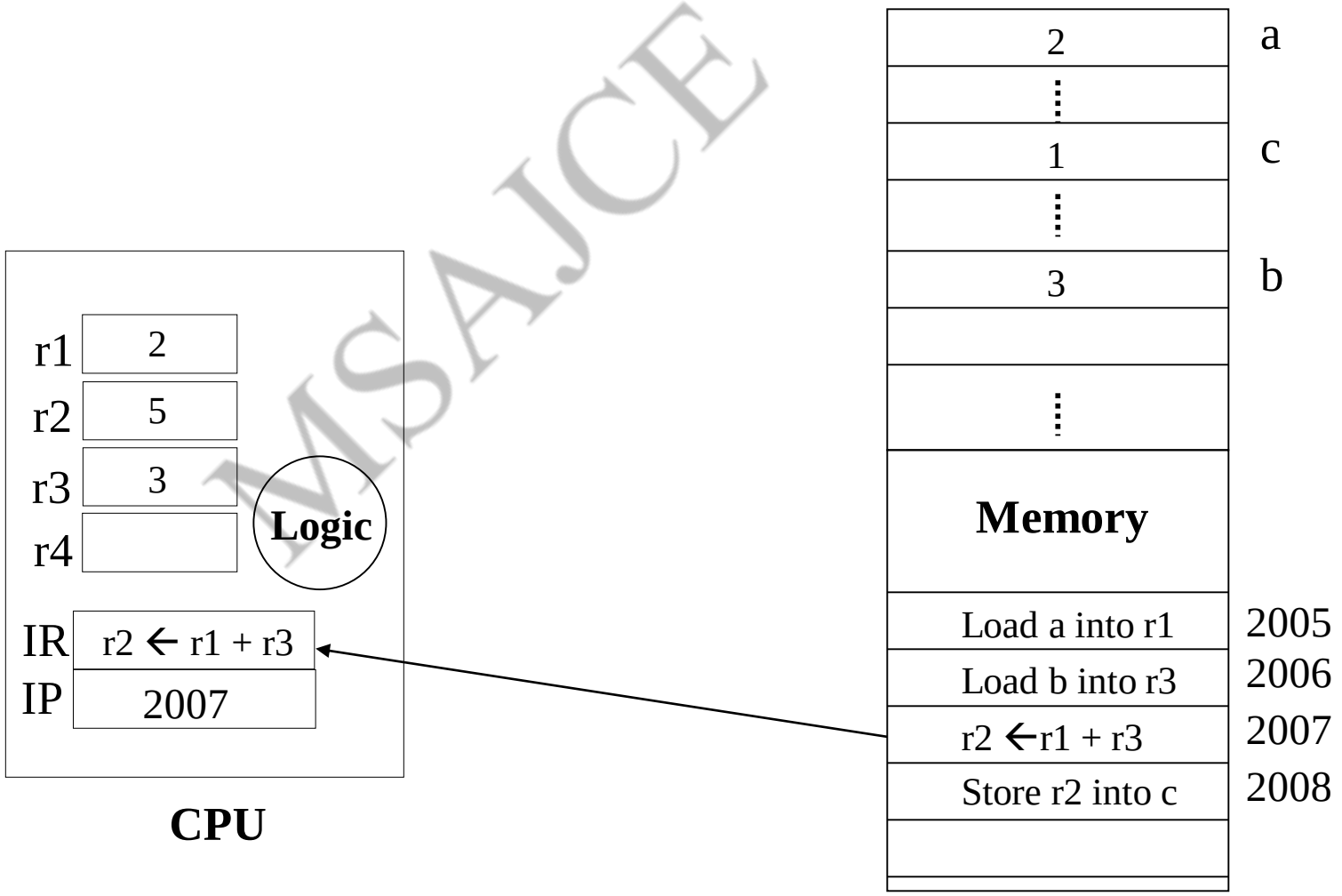
Running the Program



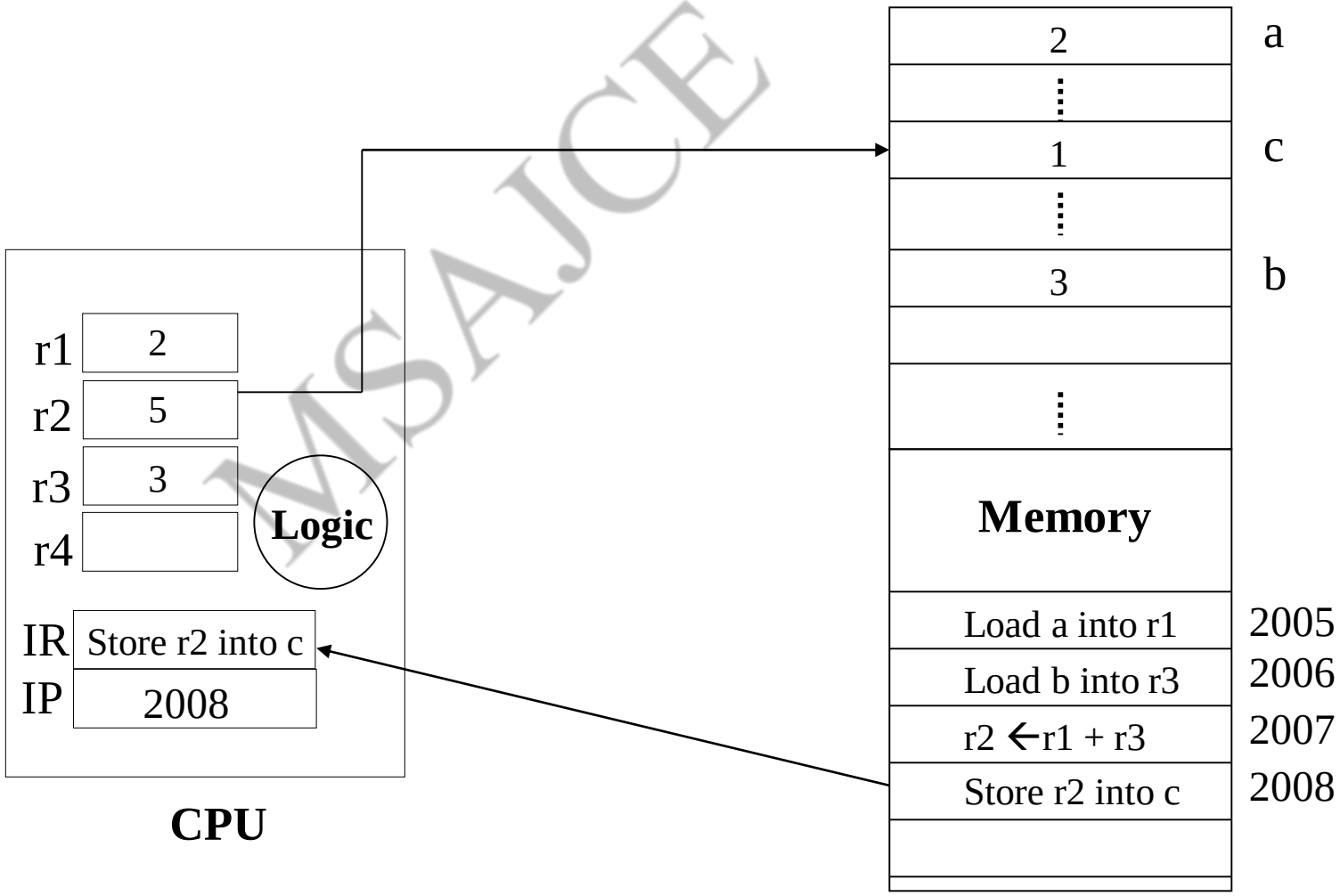
Running the Program



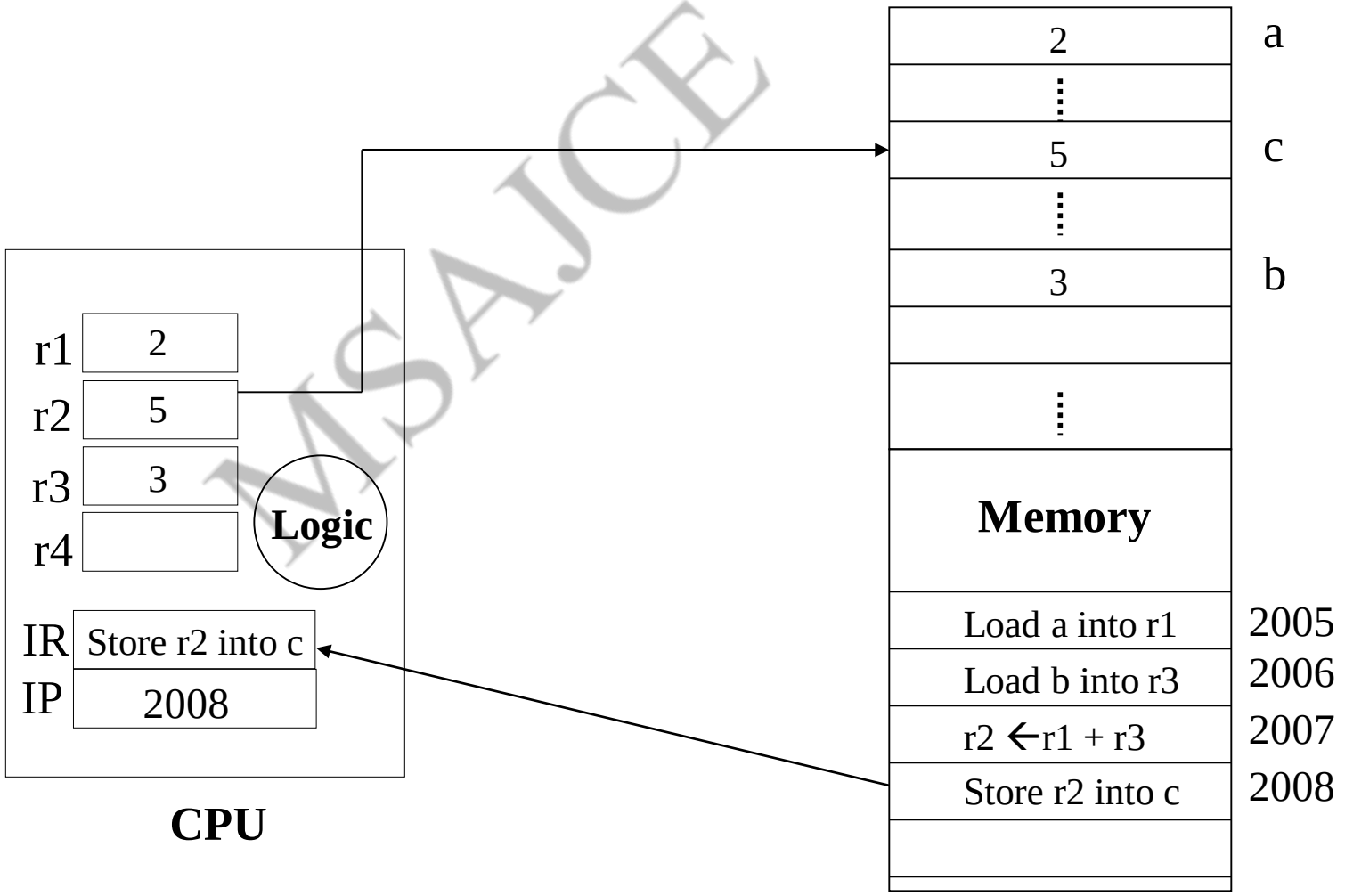
Running the Program



Running the Program

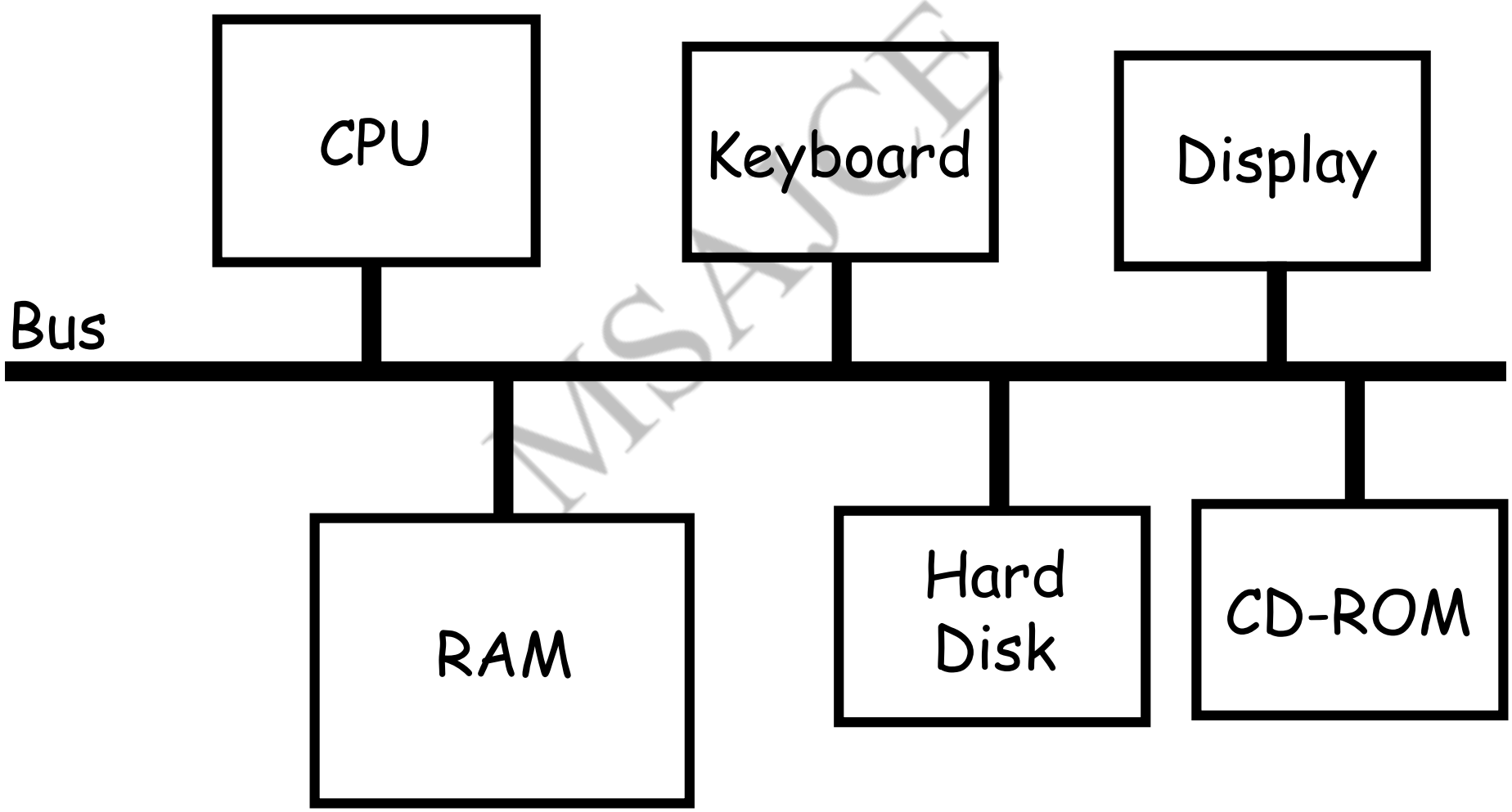


Running the Program



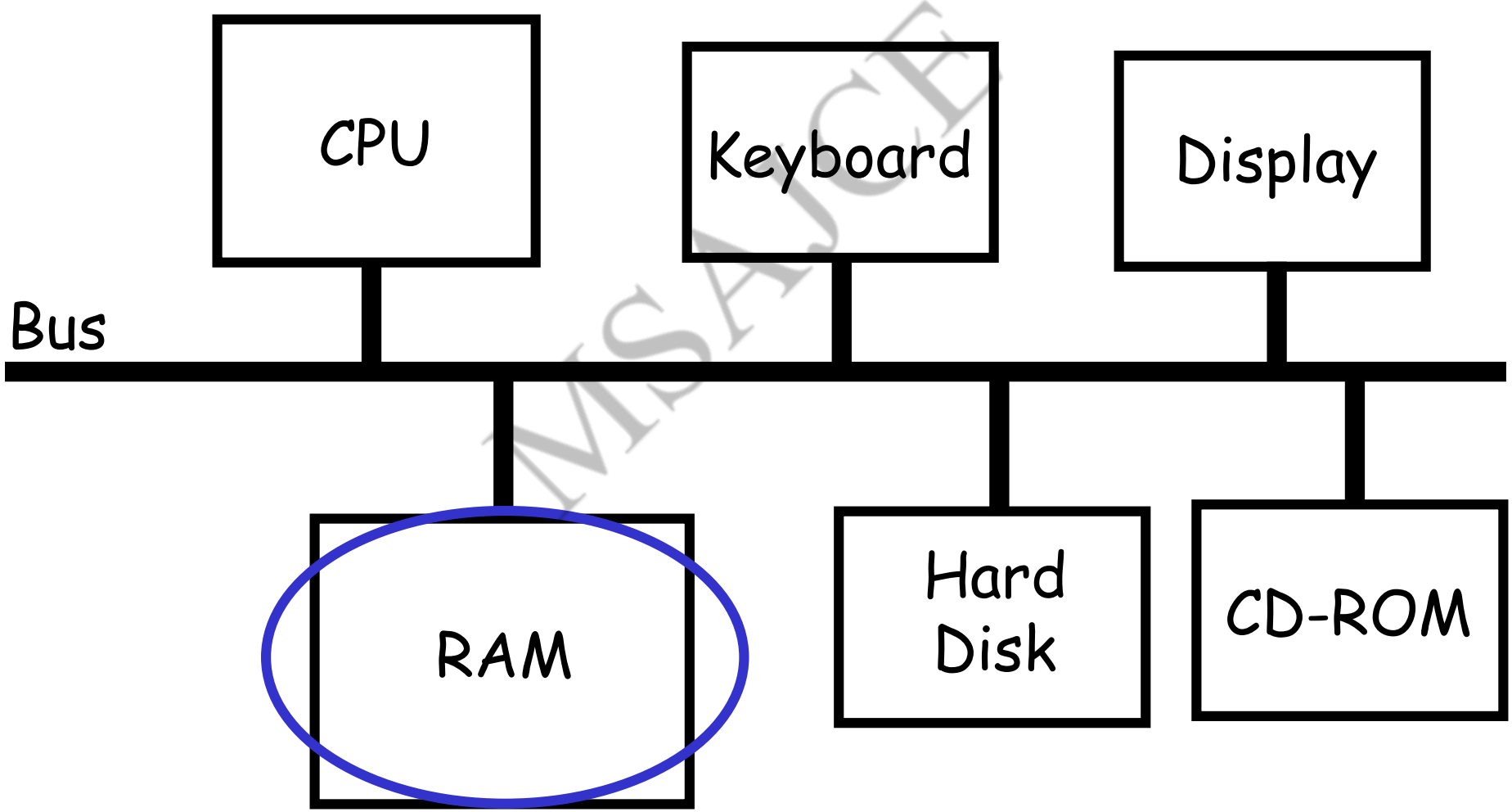
Putting it all together

- Computer has many parts, connected by a Bus:



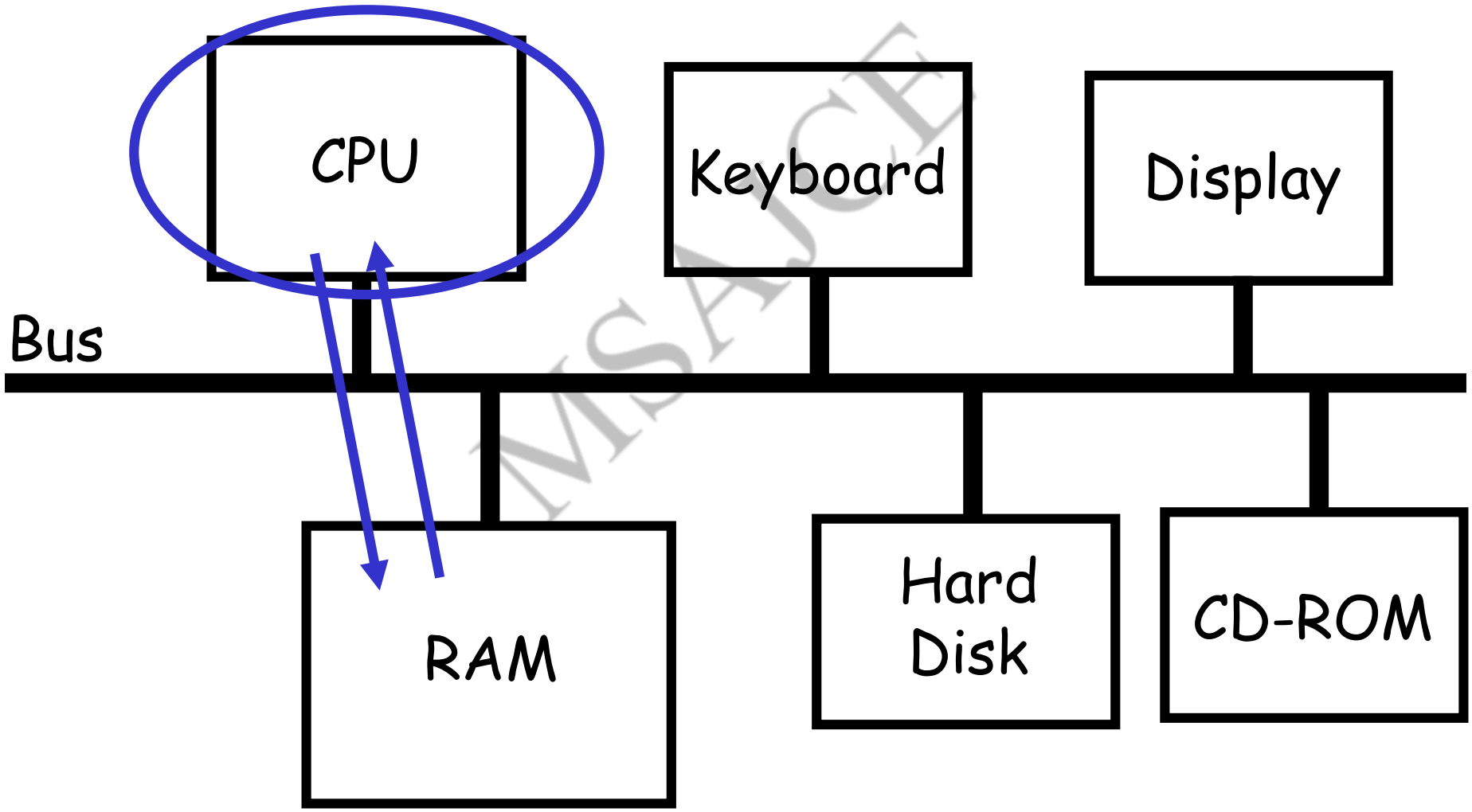
Putting it all together

- The RAM is the computer's main memory.
- This is where programs and data are stored.



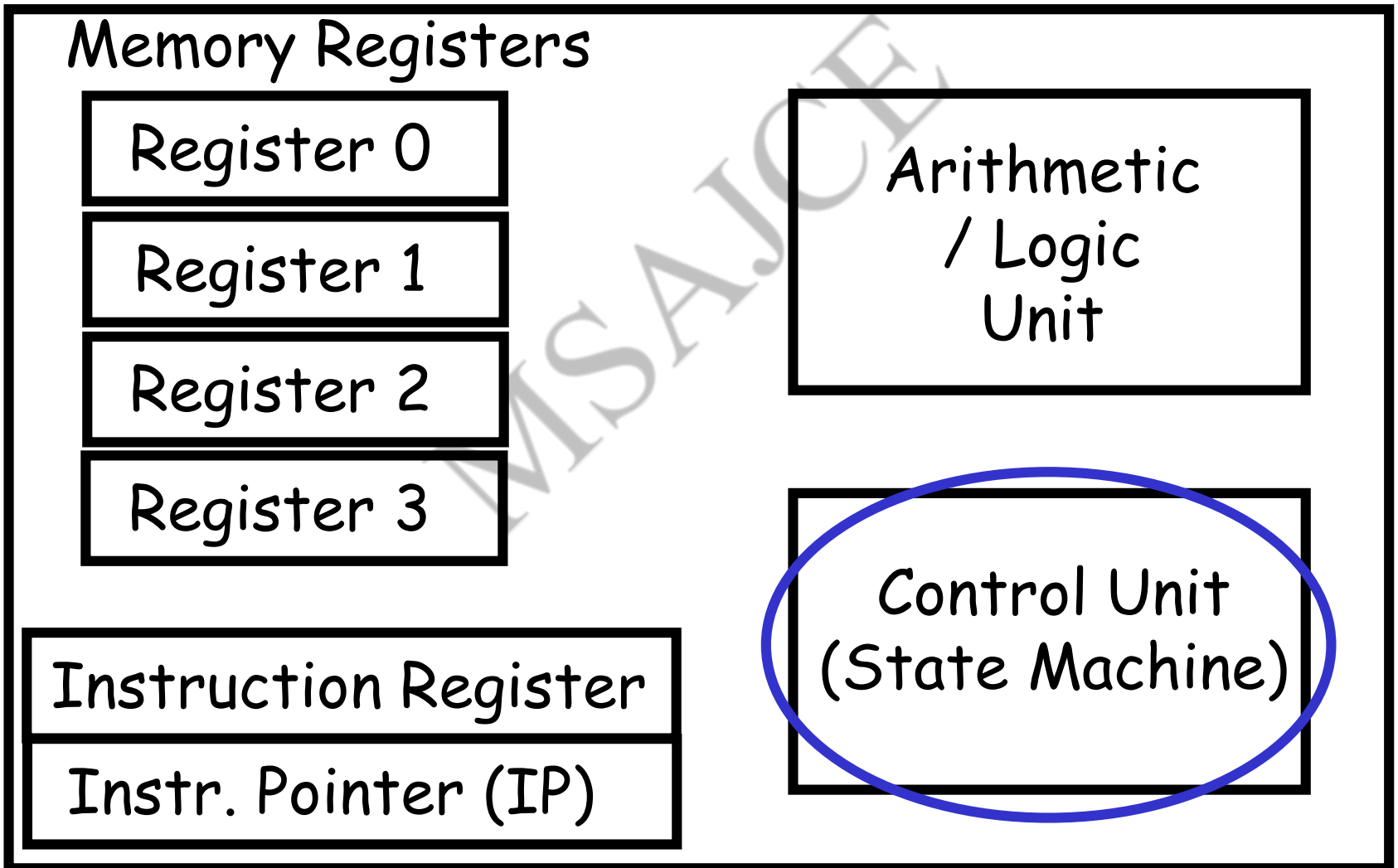
Putting it all together

- The CPU goes in a never-ending cycle, reading instructions from RAM and executing them.



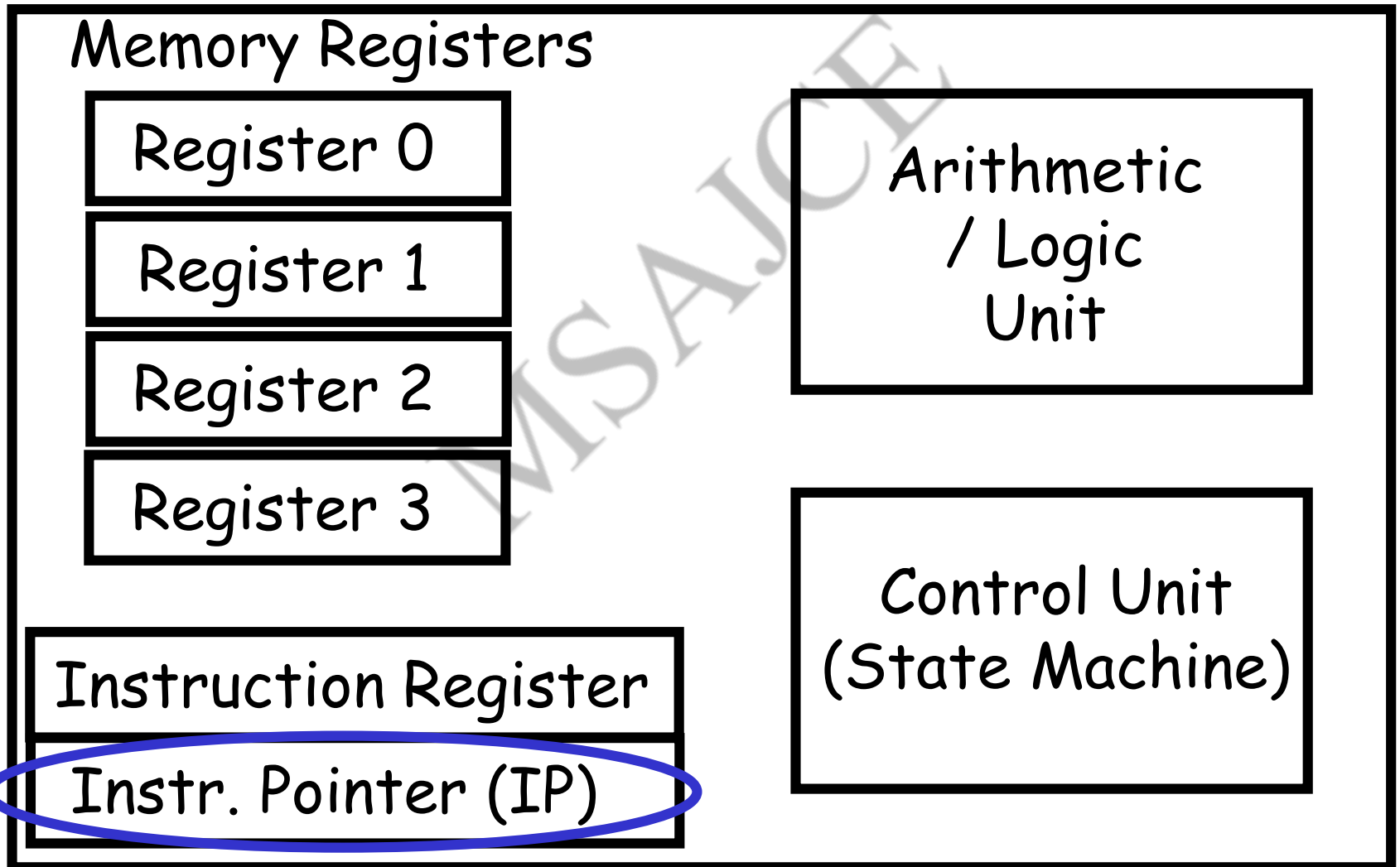
Putting it all together

- This cycle is orchestrated by the Control Unit in the CPU.



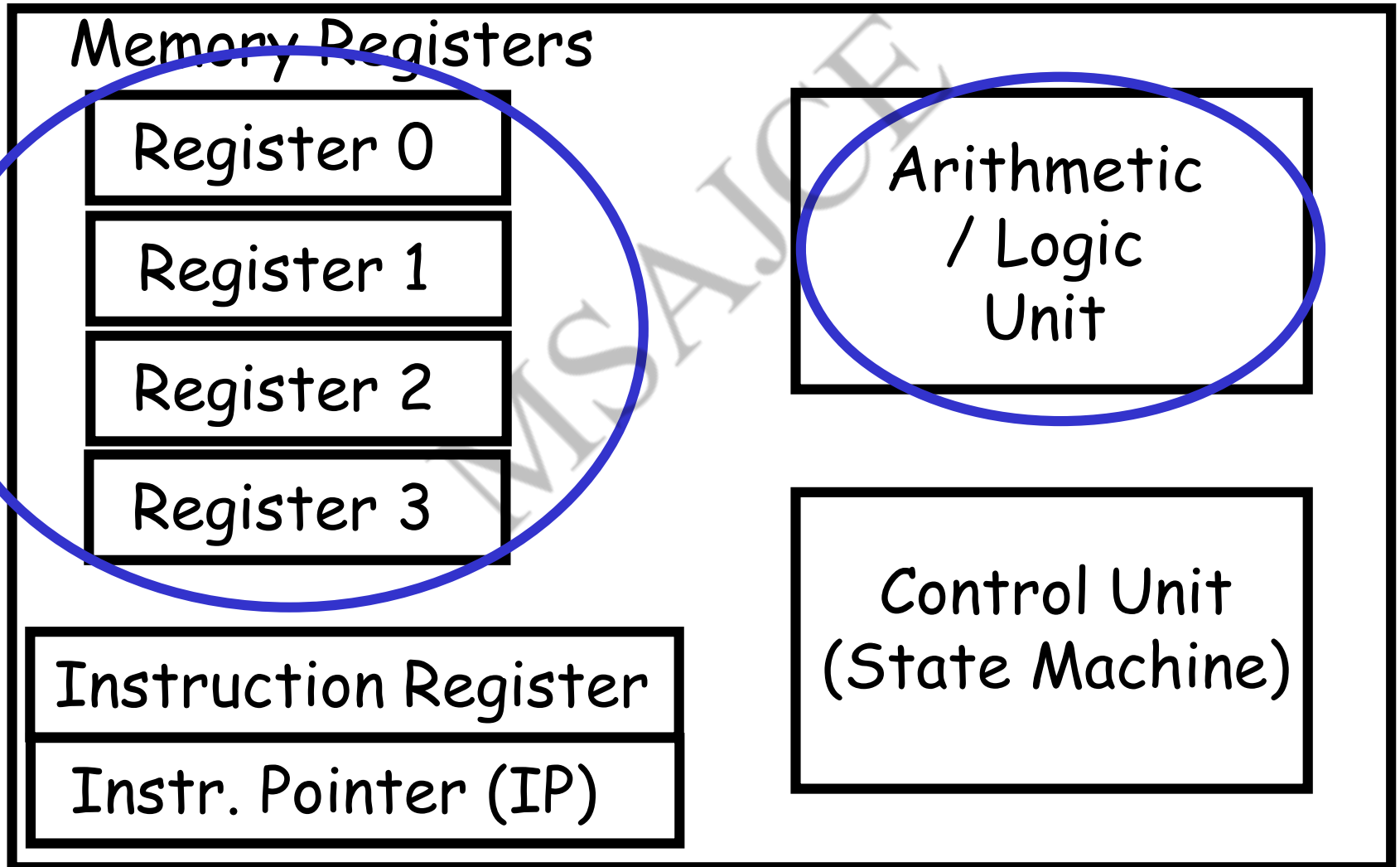
Back to the Control Unit

- It simply looks at where IP is pointing, reads the instruction there from RAM, and executes it.



Putting it all together

- To execute an instruction, the Control Unit uses the ALU plus Memory and/or the Registers.



Programming

Where we are

- Examined the hardware for a computer
  - Truth tables
  - Logic gates
  - States and transitions in a state machine
  - The workings of a CPU and Memory

- Now, want to program the hardware

Programs and Instructions

- Programs are made up of instructions
- CPU executes one instruction every clock cycle
  - Modern CPUS do more, but we ignore that
- Specifying a program and its instructions:
  - Lowest level: Machine language
  - Intermediate level: Assembly language
  - Typically today: High-level programming language

Specifying a Program and its Instructions

- High-level programs: each statement translates to many instructions
  - E.g. $c \leftarrow a + b$ to:

Load a into r1
Load b into r3
r2 ← r1 + r3
Store r2 into c

- Assembly language: specify each machine instruction, using mnemonic form
  - E.g. Load r1, A
- Machine language: specify each machine instruction, using bit patterns
  - E.g. 1101101000001110011

Machine/Assembly Language

- We have a machine that can execute instructions
- Basic Questions:
  - What instructions?
  - How are these instructions represented to the computer hardware?

Complex vs Simple Instructions

- Computers used to have very complicated instruction sets – this was known as:
  - CISC = Complex Instruction Set Computer
  - Almost all computers 20 years ago were CISC.
- 80s introduced RISC:
  - RISC = Reduced Instruction Set Computer

Complex vs Simple Instructions

- RISC = Reduced Instruction Set Computer
  - Fewer, Less powerful basic instructions
  - But Simpler, Faster, Easier to design CPU's
  - Can make "powerful" instructions by combining several wimpy ones

- Shown to deliver better performance than Complex Instruction Set Computer (CISC) for several types of applications.

Complex vs Simple Instructions

- Nevertheless, Pentium is actually CISC !
- Why?

Complex vs Simple Instructions

- Nevertheless, Pentium is actually CISC !
- Why: Compatibility with older software
- Newer application types (media processing etc) perform better with specialized instructions
- The world has become too complex to talk about RISC versus CISC

Typical Assembly Instructions

- Some common assembly instructions include:
  - 1) "Load" – Load a value from RAM into one of the registers
  - 2) "Load Direct" – Put a fixed value in one of the registers (as specified)
  - 3) "Store" - Store the value in a specified register to the RAM
  - 4) "Add" - Add the contents of two registers and put the result in a third register

Typical Assembly Instructions

- Some common instructions include:
  - 5) "Compare" - If the value in a specified register is larger than the value in a second register, put a "0" in Register r0
  - 6) "Jump" - If the value in Register r0 is "0", change Instruction Pointer to the value in a given register
  - 7) "Branch" - If the value in a specified register is larger than that in another register, change IP to a specified value

Machine Languages

- Different types of CPU's understand different instructions
  - Pentium family / Celeron / Xeon / AMD K6 / Cyrix … (Intel x86 family)
  - PowerPC (Mac)
  - DragonBall (Palm Pilot)
  - StrongARM/MIPS (WinCE)
  - Many Others (specialized or general-purpose)
- They represent instructions differently in their assembly/machine languages (even common ones)
- Let's look instructions for a simple example CPU