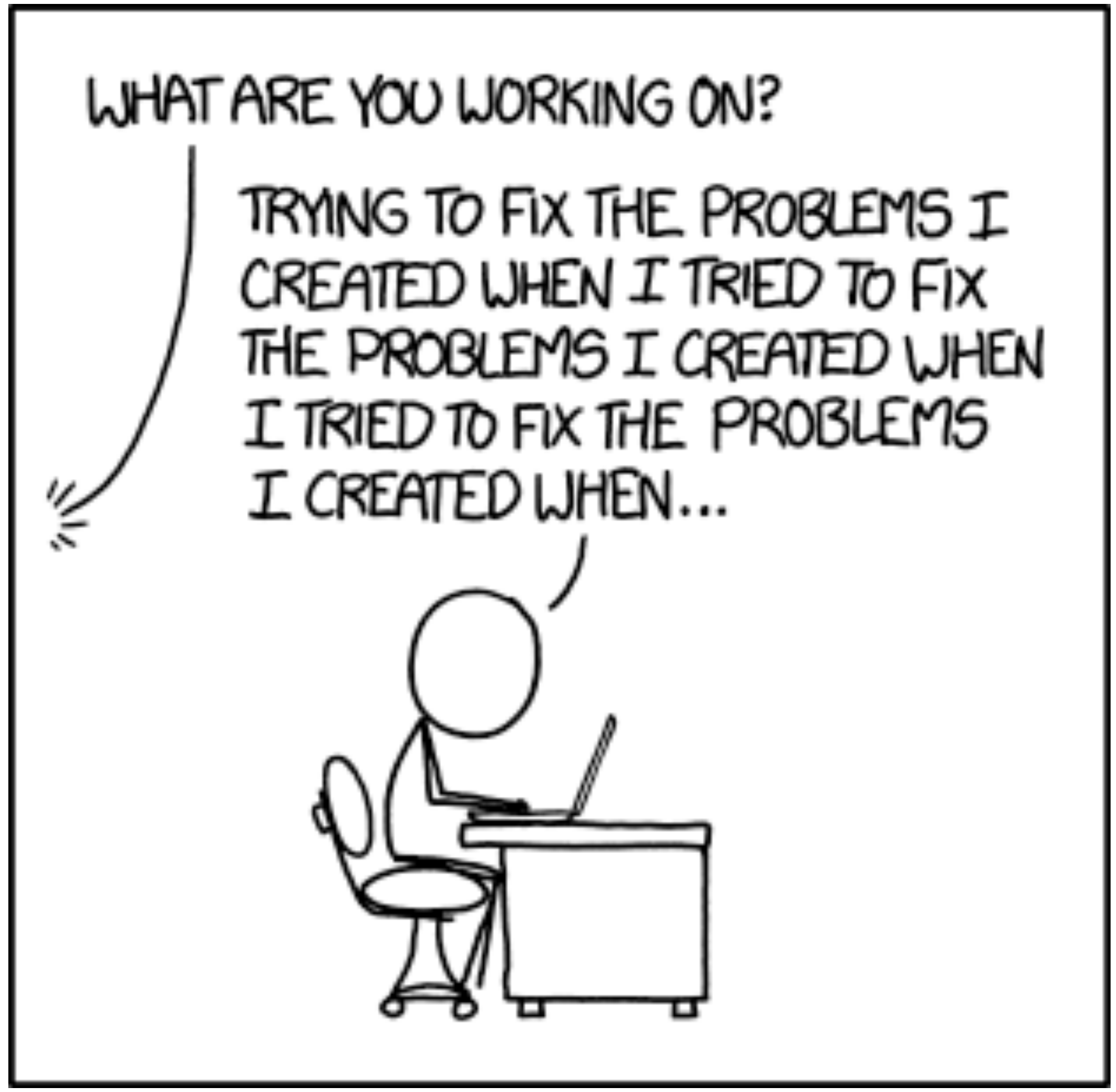


CSCI 141

Lecture 2
Hello World, Computers, Algorithms and Pseudocode

Happenings

- Tuesday, 4/9, 5 pm in CF 316:
  ACM Presents: Open Source Development with Phil Nelson
- Wednesday, 4/10, 4 pm in CF 105:
  Whiteboard Coders Present: How to Land a CS Job
- Tuesday, Apr. 9, 6 pm in MH 105:
  NSBE Presents: Black at Microsoft

Announcements

- Activate your CS Account before lab:
  - The CS department has its own computer network and labs.
  - You will have a separate account for logging into CS labs.
  - The username will be the same. You will set a different password.
  - You **must** activate your CS account from a **non-CS** computer **before** you arrive at your first lab next week.
  - Go to http://password.cs.wwu.edu and follow the instructions there.

Last time: Takeaways

- This course covers the basics of programming, and is the beginning of a journey towards a new way of thinking and solving problems.
- Programming and problem-solving are useful skills, whether you plan to go into computer science or not.
- Making mistakes is an important part of learning. Learn from your own mistakes, and don’t judge other people for theirs. Be empathetic.
- Class participation is an important component of this course.
- Don’t stay stuck on assignments for too long: get help early and often.

Goals

- A slide (or two) like this will appear at the beginning of each lecture.
- This tells you what I want you to get out of the lecture
    - I will use it when writing exams
    - You can use it when studying for exams
- The goal is transparency: you know what I want you to know.

Goals: Concepts

- Gain a basic understanding of the components of a computer, and how they interact:
  - Input and output devices
  - CPU
  - Storage
  - Programs
- Understand the distinction between a programming language and an Integrated Development Environment
- Know the definition and purpose of **algorithms** and **pseudocode** and how they fit into the software development process.

Goals: Python

- Understand the basic usage of the Thonny IDE
- Know how to use comments to document your code
- Be able to write a correct “Hello World!” program in Python

Let's write some code already

- Python is our chosen programming language in this course.
- A programming language is a language a computer can "understand" and execute (more on this later today)
- We'll use a program called Thonny to write our Python code.
- Thonny is an example of an "Integrated Development Environment" (IDE): a program that provides all the features you need to write, run, and fix errors in programs.

Hello, world!

- Example code

Hello, world!

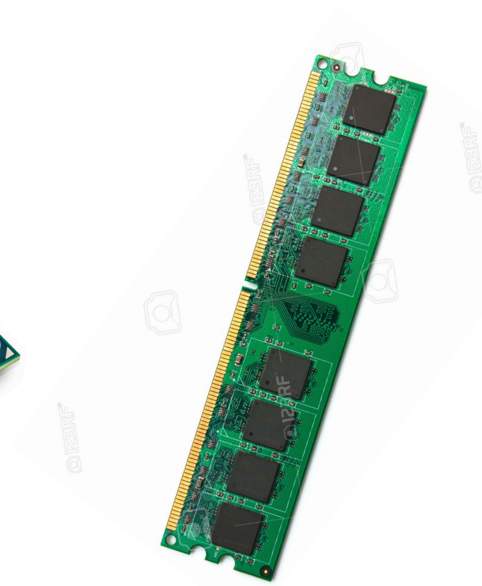
- Example code
- Concepts demonstrated:
  - Comments
  - Print function
  - Single and double quoted strings
  - Input function

What just happened?

- A lot! This course won't get into the details.
- A simple model of a computer:

Input Devices

CPU



Main Memory

Secondary Storage

Output Devices

Hardware

- A simple model of a computer:

Input Devices

Supply input from a user to the computer.

Hardware

- A simple model of a computer:

Output Devices Transmit information back to the user.

Hardware

- A simple model of a computer:

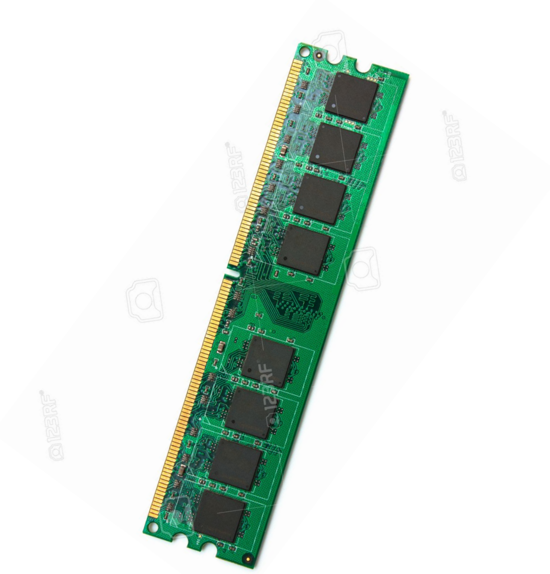
CPU:
Central Processing Unit

Executes instructions to run computer programs.

Hardware

- A simple model of a computer:

Short-term information storage: Information goes away when the computer is turned off or the program quits.



Main Memory

Hardware

- A simple model of a computer:

Long-term information storage:
Stays around even if computer is off, or
if program quits.

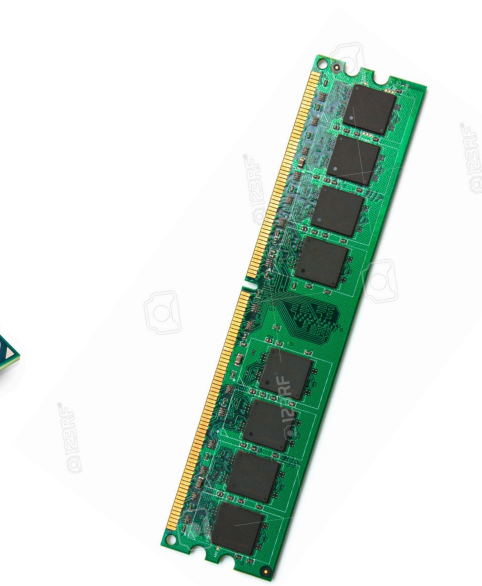
Secondary Storage

Hardware

- A simple model of a computer:

Input Devices

CPU

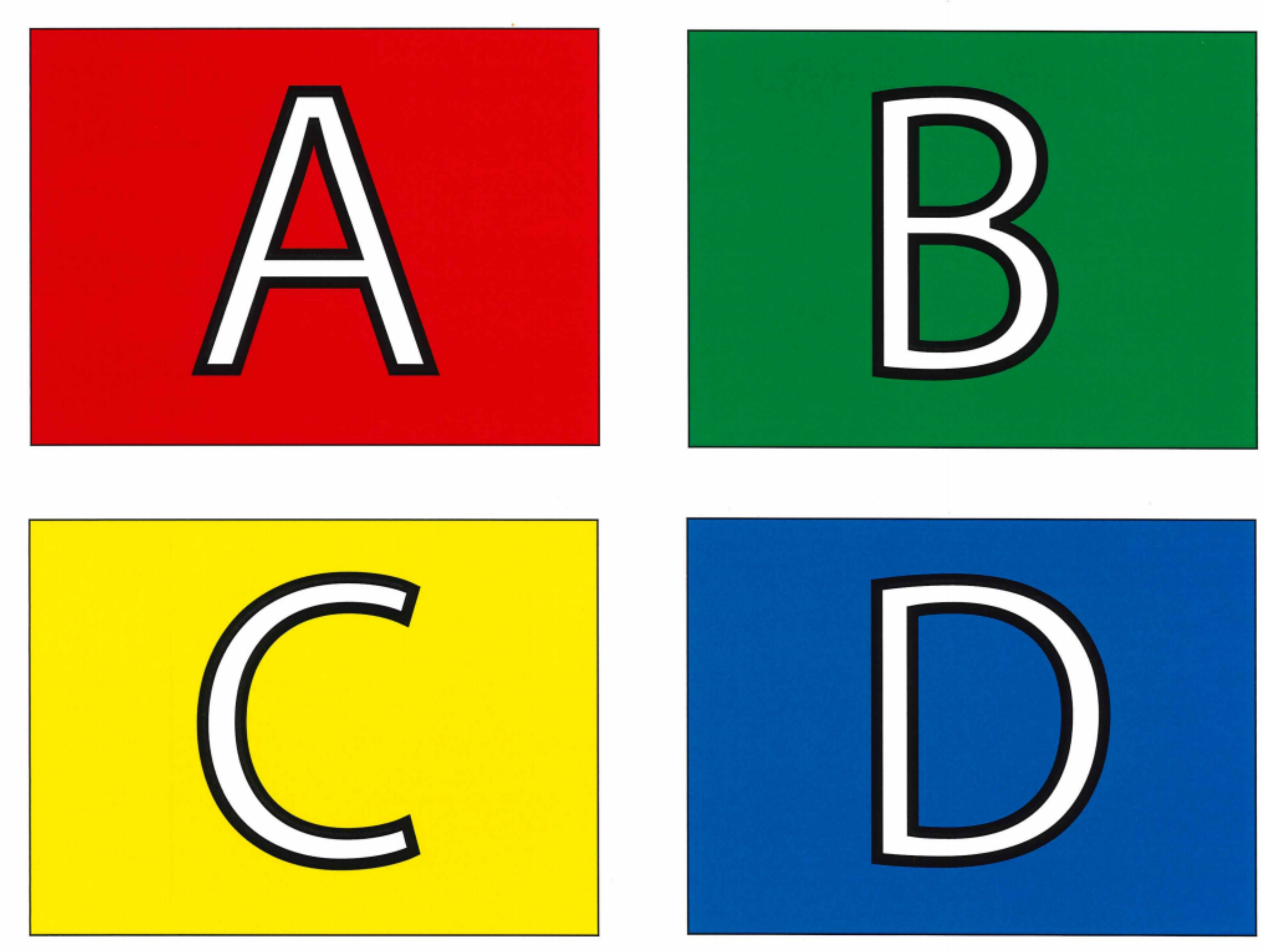


Main Memory

Secondary Storage

Output Devices

ABCD Cards



ABCD Practice

The instructor of this course prefers that you address him as:

A. Professor Wehrwein

B. Scott

C. Dr. Wehrwein

D. Dude

ABCD Practice

CPU stands for:

A. Coronary Pulse Upkeep

B. Critical Process Undertaker

C. Computer Process User

D. Central Processing Unit

ABCD Cards

The CPU is like the _______ of the computer.

A. Foot

B. Bookshelf

C. Brain

D. Treadmill

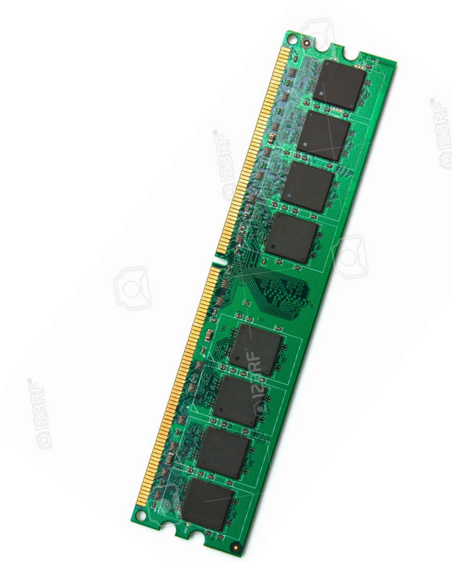
Hardware

- A simple model of a computer:

Input Devices

Output Devices

CPU



Main Memory

Secondary Storage

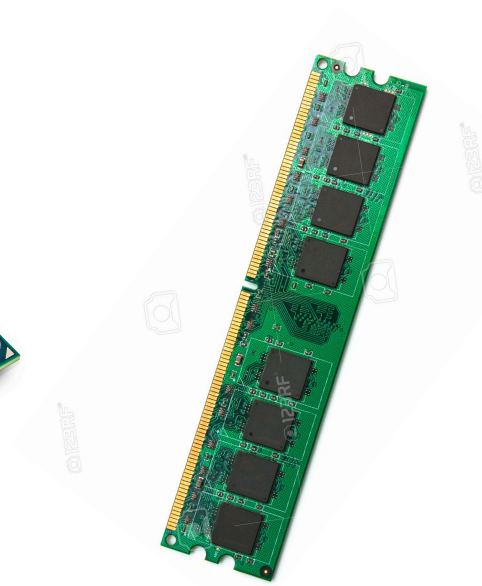
What can computers do?

- Run programs (software).

Input Devices

Output Devices

CPU



Main Memory

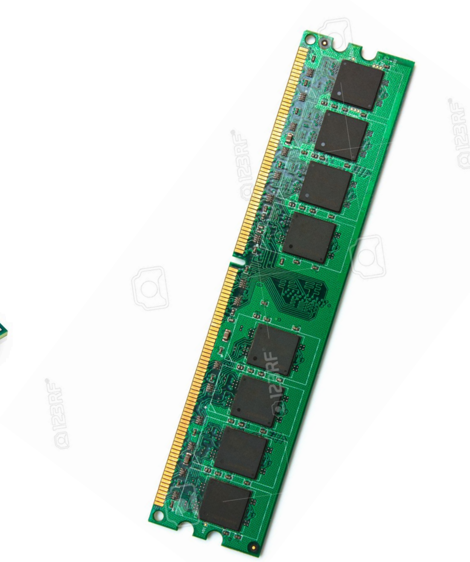
Secondary Storage

What can computers do?

- Run programs (software).
- **That's it!**

Input Devices

CPU



Main Memory

Secondary Storage

Output Devices

What can computers do?

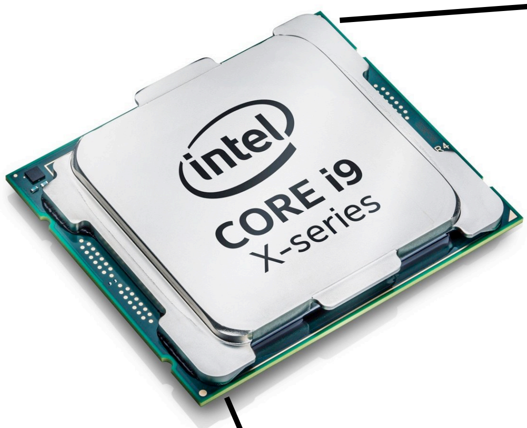
- How does the computer run programs?

CPU

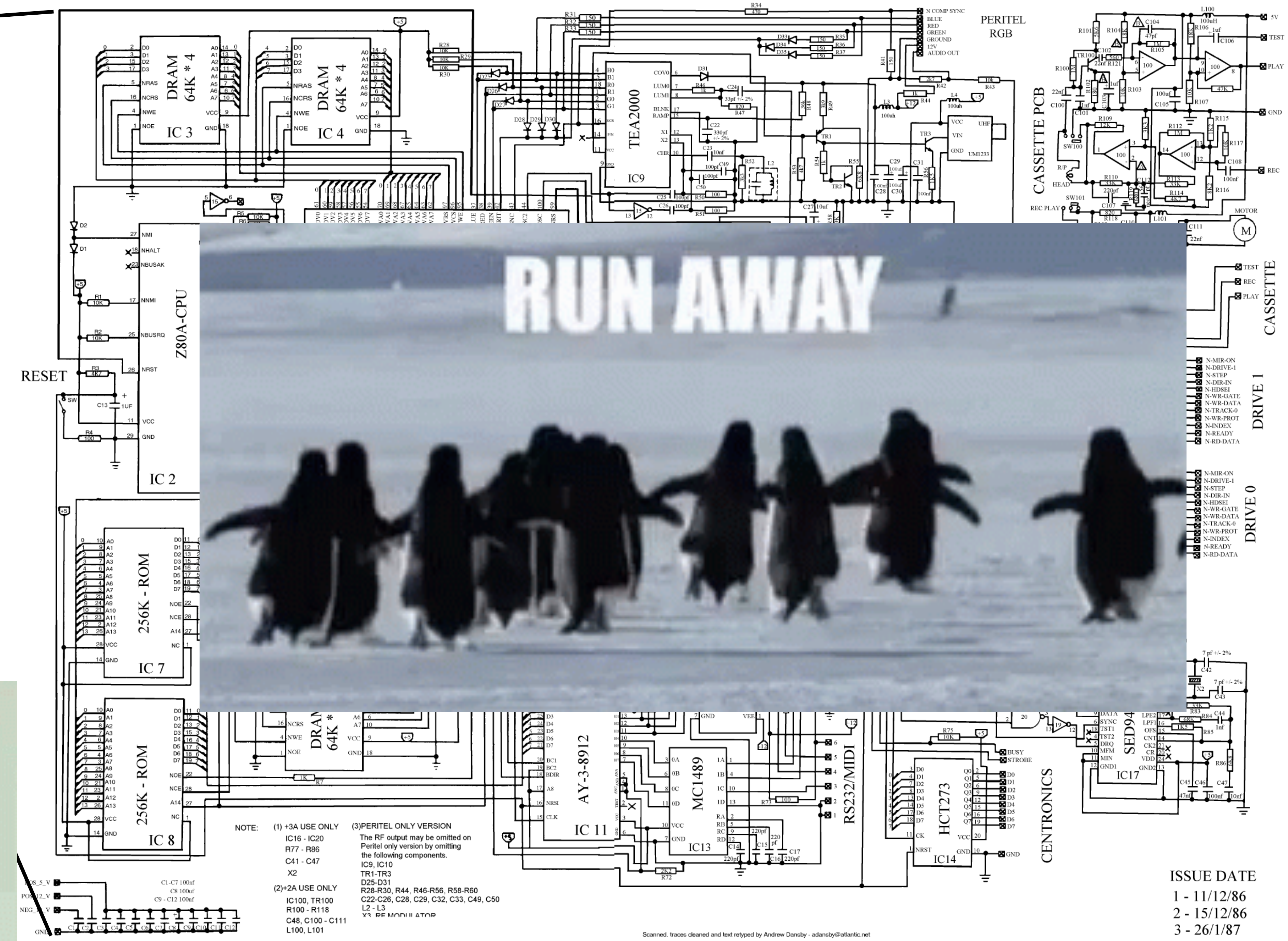
Executes instructions to run computer programs.

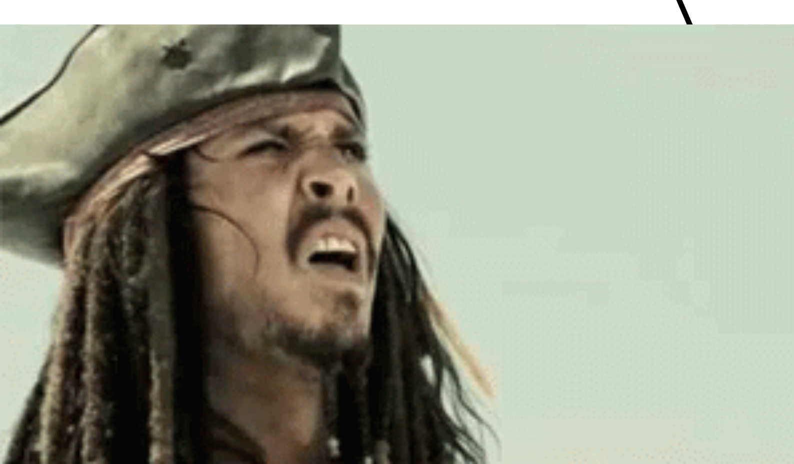
What can computers do?

- How does that work? Let's take a closer look…



CPU





What can computers do?

- How does that work? Let’s **not** take a closer look.

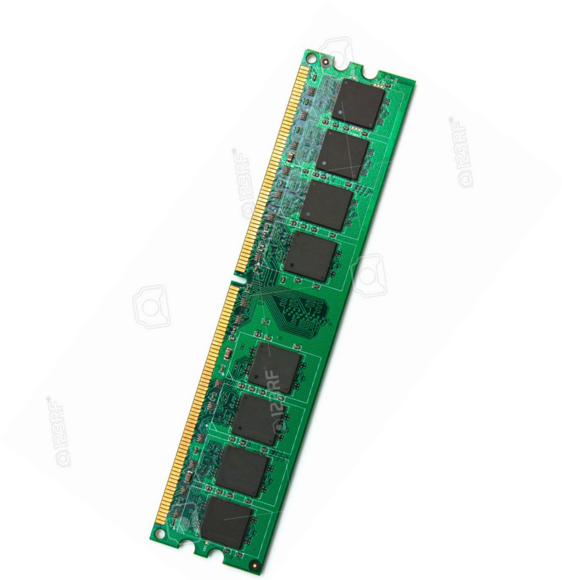
CPU

We don’t need to know the hardware details!
This is an example of **abstraction**.

What can computers do?

- How does the computer run programs?

CPU



Main Memory

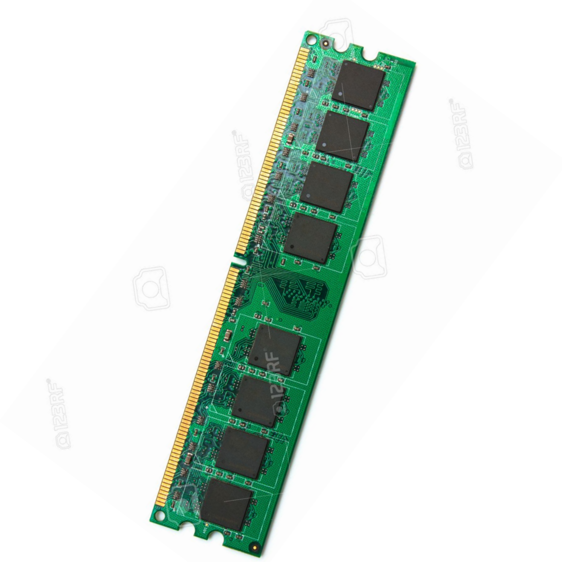
Here's how we'll think about it:
A program is stored in main memory.

1. **Fetch** an instruction from memory
2. **Decode** that instruction
3. **Execute** the instruction

What can computers do?

- How does the computer run programs?

CPU



Main Memory

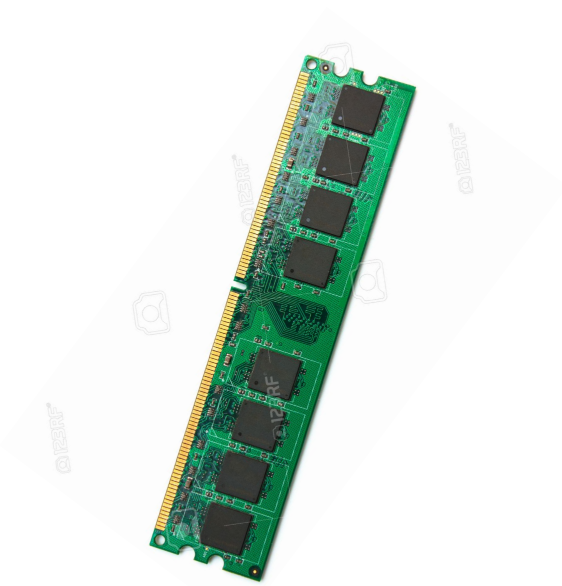
Here's how we'll think about it:
A program is stored in main memory.

1. **Fetch** an instruction from memory
2. **Decode** that instruction
3. **Execute** the instruction

What can computers do?

Consider a simple program:
Multiply 3 by 4, add 2, and print to screen

CPU



Main Memory

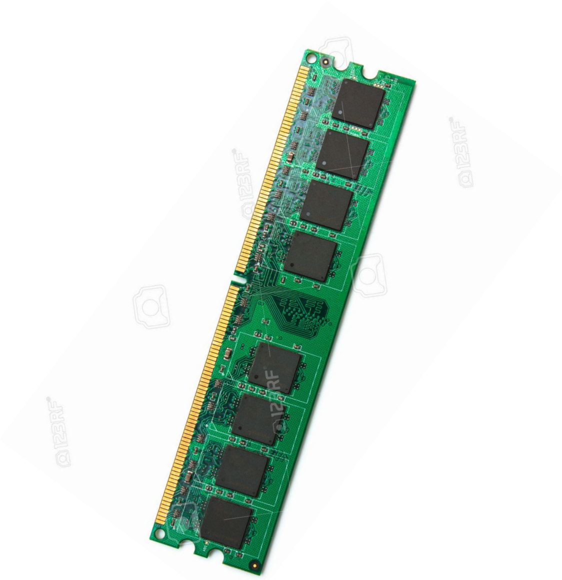
What can computers do?

Consider a simple program:
Multiply 3 by 4, add 2, and print to screen

1. **Fetch** first instruction ("multiply 3 by 4)"

(move it from memory to the CPU)

CPU



Main Memory

What can computers do?

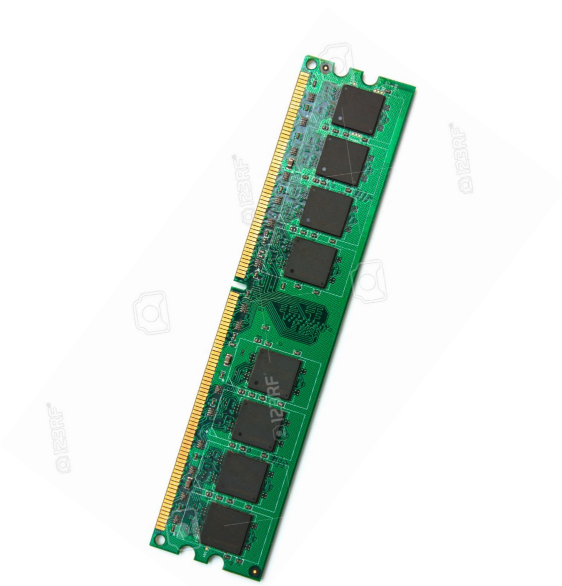
Consider a simple program:

Multiply 3 by 4, add 2, and print to screen

1. **Fetch** first instruction ("multiply 3 by 4)"
2. **Decode:** convert to CPU instructions

(translate it into instructions the CPU can execute)

CPU

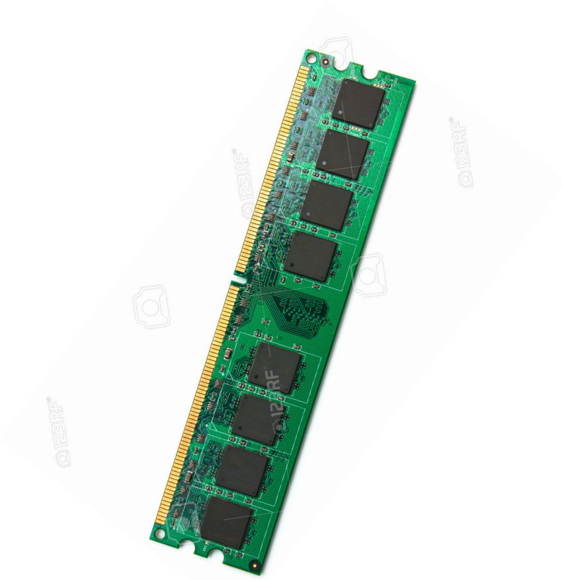


Main Memory

What can computers do?

Consider a simple program:
Multiply 3 by 4, add 2, and print to screen

CPU



Main Memory

1. **Fetch** first instruction (“multiply 3 by 4)”
2. **Decode:** convert to CPU instructions
3. **Execute** the instruction using CPU circuitry

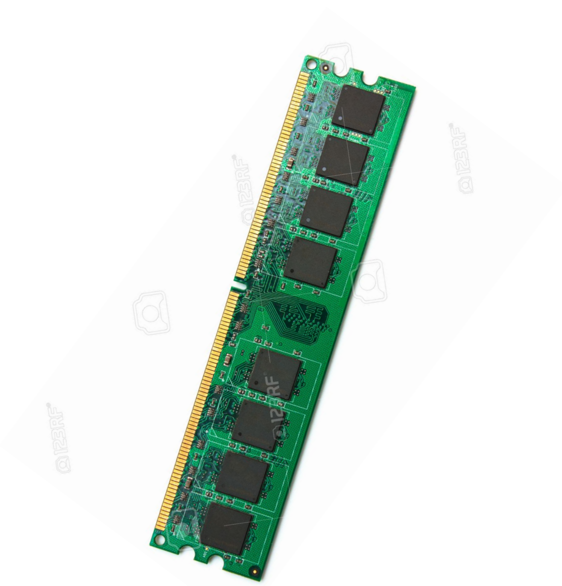
(actually multiply 3 by 4, and
save the result (12) to memory)

What can computers do?

Consider a simple program:
Multiply 3 by 4, add 2, and print to screen

CPU

1. **Fetch** first instruction ("multiply 3 by 4)"
2. **Decode:** convert to CPU instructions
3. **Execute** the instruction using CPU circuitry
4. **Fetch** next instruction ("add 2")

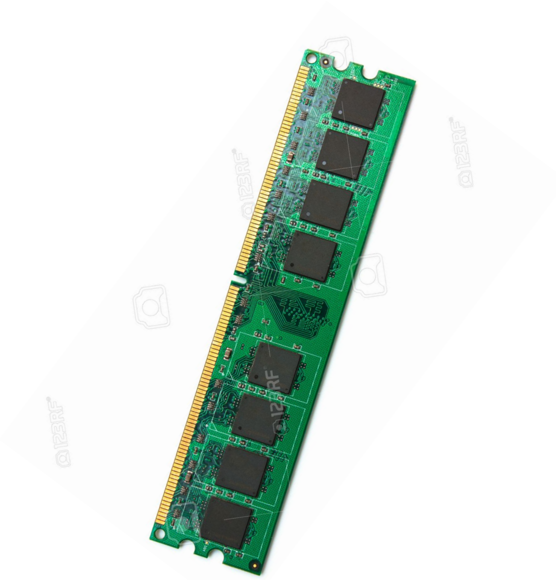


Main Memory

What can computers do?

Consider a simple program:
Multiply 3 by 4, add 2, and print to screen

CPU



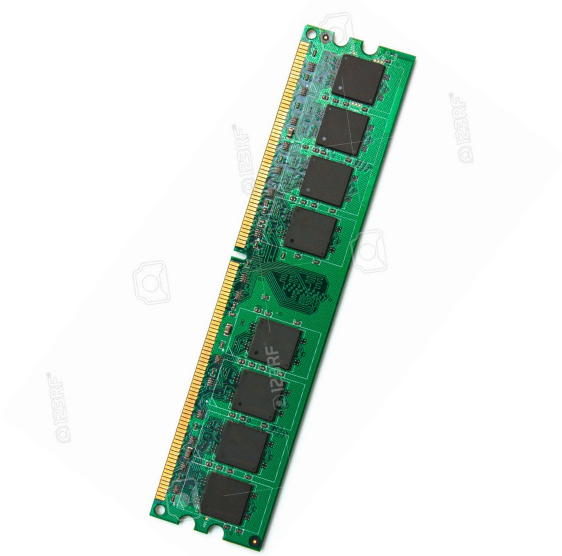
Main Memory

1. **Fetch** first instruction ("multiply 3 by 4)"
2. **Decode:** convert to CPU instructions
3. **Execute** the instruction using CPU circuitry
4. **Fetch** next instruction ("add 2")
5. **Decode**

What can computers do?

Consider a simple program:
Multiply 3 by 4, add 2, and print to screen

CPU



Main Memory

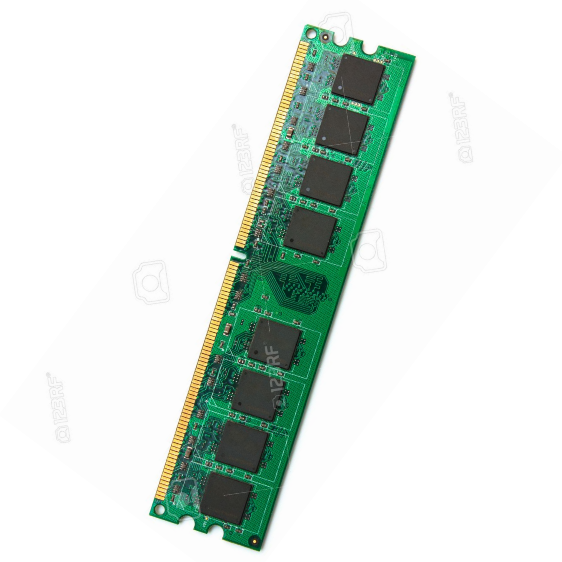
1. **Fetch** first instruction ("multiply 3 by 4)"
2. **Decode:** convert to CPU instructions
3. **Execute** the instruction using CPU circuitry
4. **Fetch** next instruction ("add 2")
5. **Decode**
6. **Execute:** add 2 to the result in memory

(add 2 to 12, and store the result (14) to memory again)

What can computers do?

Consider a simple program:
Multiply 3 by 4, add 2, and print to screen

CPU



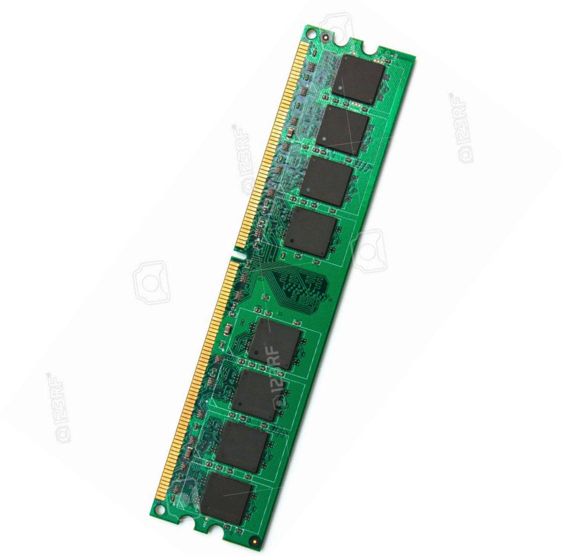
Main Memory

1. **Fetch** first instruction (“multiply 3 by 4)”
2. **Decode:** convert to CPU instructions
3. **Execute** the instruction using CPU circuitry
4. **Fetch** next instruction (“add 2”)
5. **Decode**
6. **Execute:** add 2 to the result in memory
7. **Fetch** the next instruction (“print to screen”)

What can computers do?

Consider a simple program:
Multiply 3 by 4, add 2, and print to screen

CPU



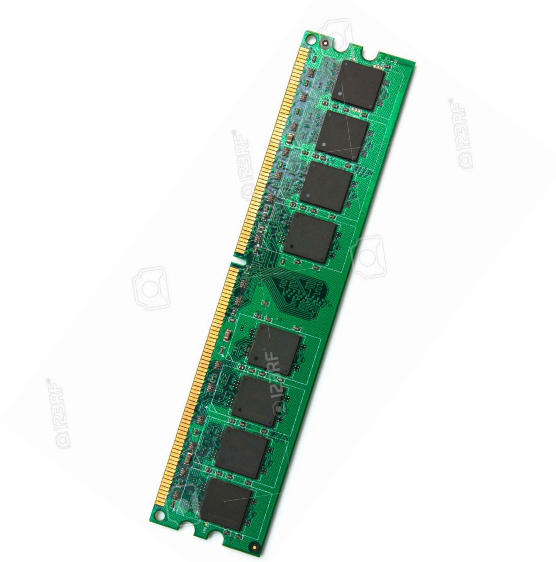
Main Memory

1. **Fetch** first instruction (“multiply 3 by 4)”
2. **Decode:** convert to CPU instructions
3. **Execute** the instruction using CPU circuitry
4. **Fetch** next instruction (“add 2”)
5. **Decode**
6. **Execute:** add 2 to the result in memory
7. **Fetch** the next instruction (“print to screen”)
8. **Decode**

What can computers do?

Consider a simple program:
Multiply 3 by 4, add 2, and print to screen

CPU



Main Memory

1. **Fetch** first instruction (“multiply 3 by 4)”
2. **Decode:** convert to CPU instructions
3. **Execute** the instruction using CPU circuitry
4. **Fetch** next instruction (“add 2”)
5. **Decode**
6. **Execute:** add 2 to the result in memory
7. **Fetch** the next instruction (“print to screen”)
8. **Decode**
9. **Execute:** print the result in memory to the screen

ABCD

While running, all program instructions are stored in:

A. The CPU

B. The recycle bin

C. Input/Output devices

D. Main Memory

ABCD

Which of the following is not an important part of how computers execute everyday programs?

A. Arithmetic Logic Units

B. SIMD Registers

C. Cache Hierarchies

D. The Call Stack

ABCD

Which of the following is not an important part of how computers execute everyday programs?

A. Arithmetic Logic Units

B. SIMD Registers

C. Cache Hierarchies

D. The Call Stack

Remember: “I don’t know” is a valid ABCD response!

Our Simple Program

- We just executed this:

Multiply 3 by 4, add 2, and print to screen

- Is this a Python program?

Our Simple Program

- We just executed this:

  Multiply 3 by 4, add 2, and print to screen

- Is this a Python program? Let’s find out…

Our Simple Program

- We just executed this:

Multiply 3 by 4, add 2, and print to screen

- Is this a Python program? No!
- What Python program accomplishes the same thing?

```
print(3 * 4 + 2)
```

Our Simple Program

- This is an an example of an **algorithm:**

Multiply 3 by 4, add 2, and print to screen

An algorithm is a sequence of steps that solve a problem.

Our Simple Program

- This is an an example of an **algorithm:**

  Multiply 3 by 4, add 2, and print to screen

- The algorithm is written in **pseudocode**

  Pseudocode is a way of expressing algorithms independent of any specific programming language: think of it as an **informal but precise** description of an algorithm.

Our Simple Program

- This is an an example of an **algorithm:**

Multiply 3 by 4, add 2, and print to screen

- The algorithm is written in **pseudocode**

- This is an **implementation** of the algorithm written in Python:

```
print(3 * 4 + 2)
```

Python is high-level programming language that can be translated into instructions that can be executed on a CPU.

Solving Problems with Computers

An algorithm for solving problems:

1. Devise an **algorithm** to solve the problem
2. Write the algorithm in **pseudocode**.
3. **Translate** the pseudocode into a programming language to implement the algorithm.
4. Execute and **test** the program, fixing errors until it solves the problem correctly.

True or False

- When you are presented with a problem that can be solved using a computer, the first step is to start writing a program.

A. False

B. True

Example:
“I need a tool that adds three numbers”

True or False

- When you are presented with a problem that can be solved using a computer, the first step is to start writing a program.

A. False

B. True

Example:
“I need a tool that adds three numbers”

Think about the problem,
sketch out some pseudocode,
then start writing code!

True or False

- When you are presented with a problem that can be solved using a computer, the first step is to start writing a program.

A. False

B. True

Example:
"I need a tool that adds three numbers"
"I need a tool to sort 3 million social security numbers"

Think about the problem, sketch out some pseudocode, **then** start writing code!