COSC 122

Computer Fluency

Computer Organization

Dr. Abdallah Mohamed

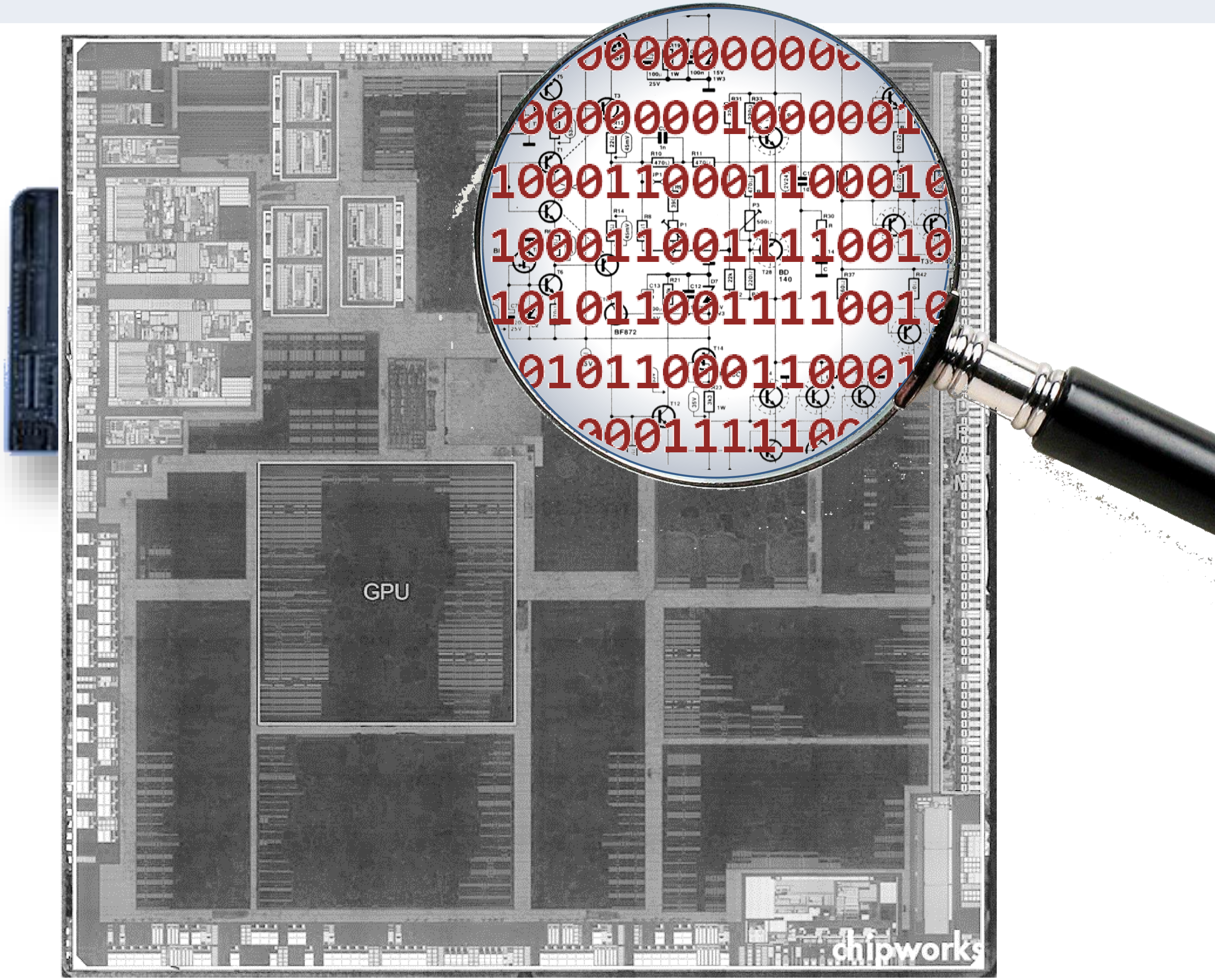
A Computer….

http://allthingsd.com

Let’s [illegible]k Under the Hood

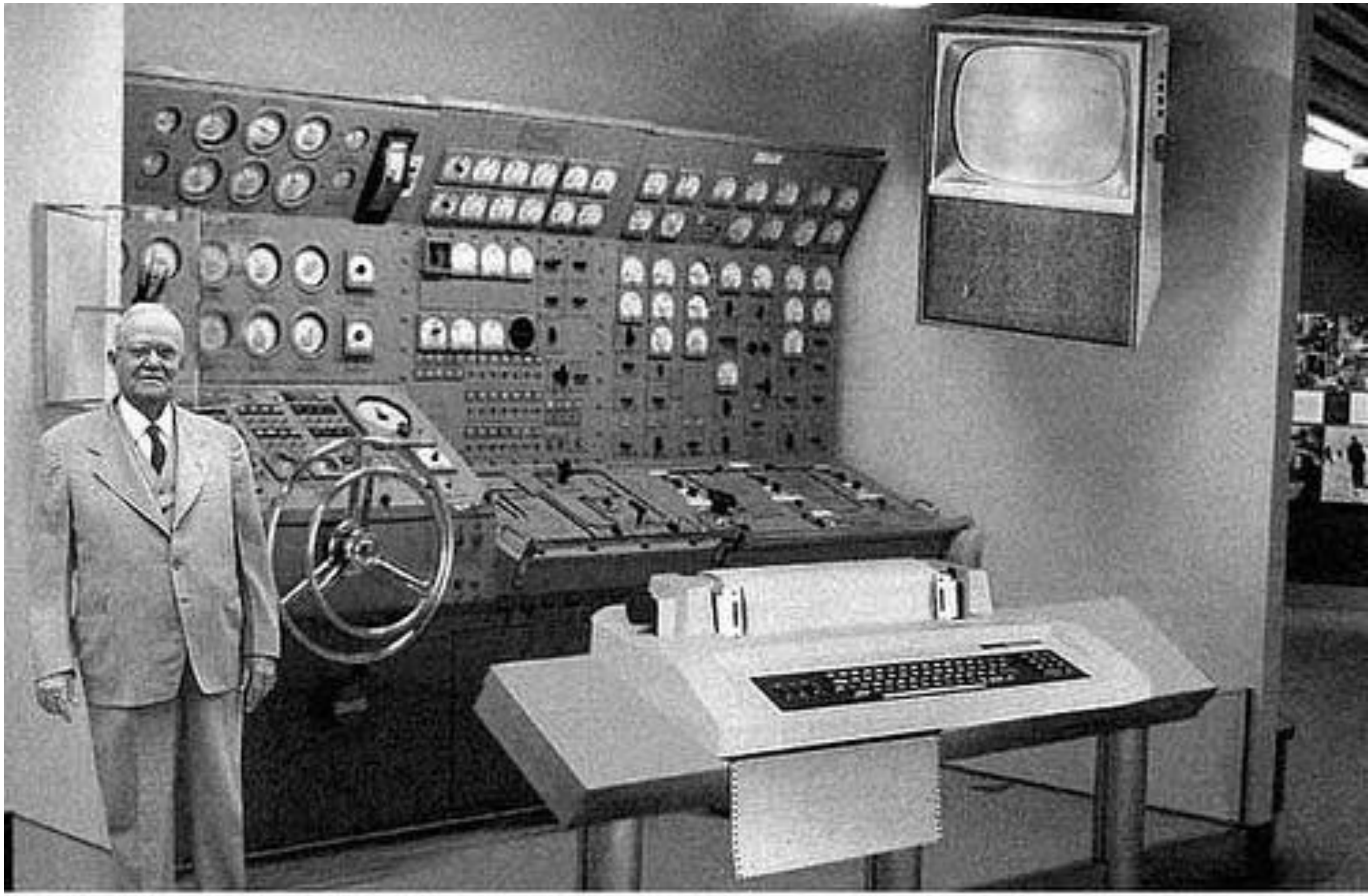
http://allthingsd.com

Let’s Look Under the Hood



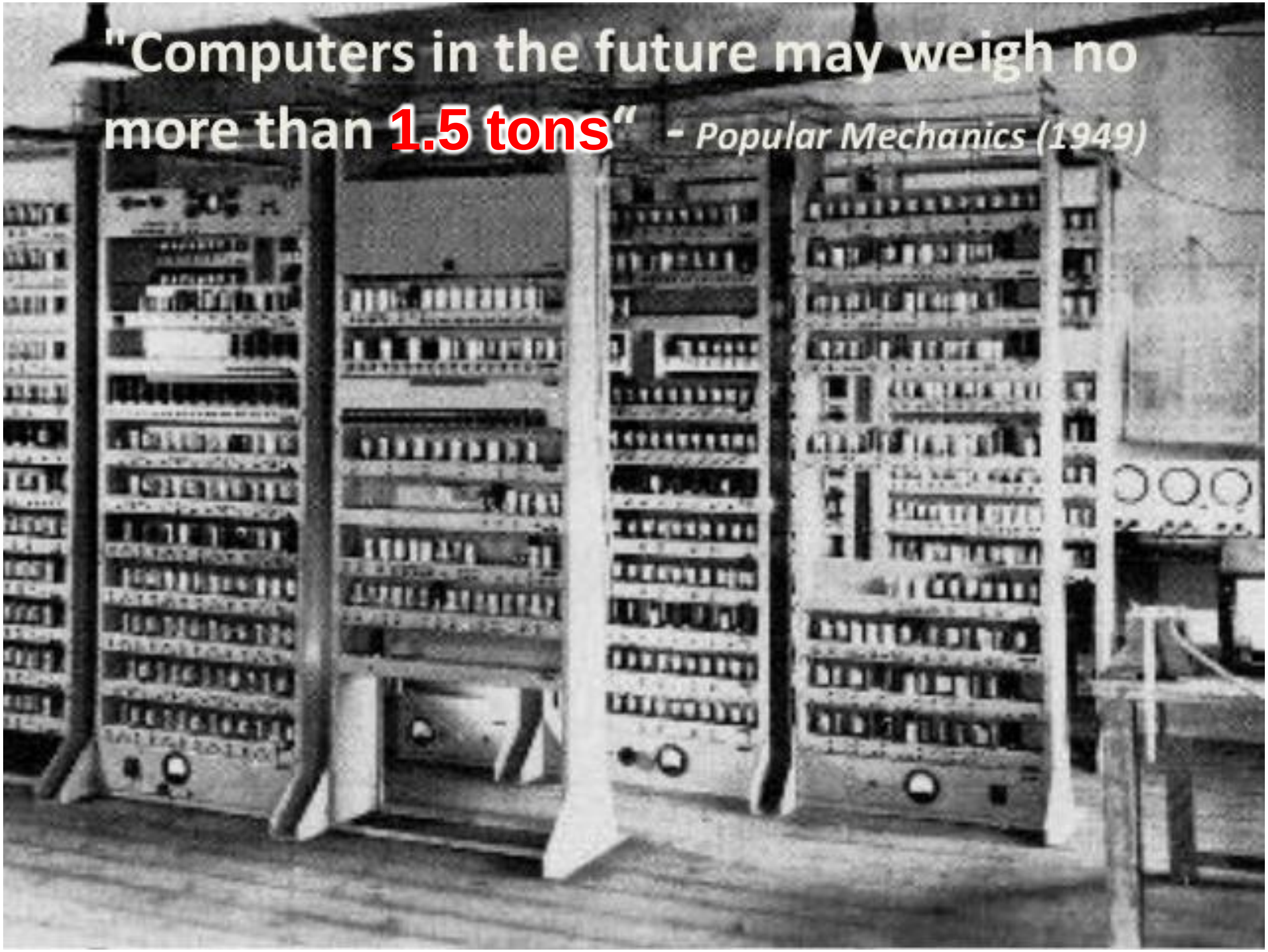
How Computer Was Predicted Back Then!

- This is how some in 1954 predicted a 2004 computer



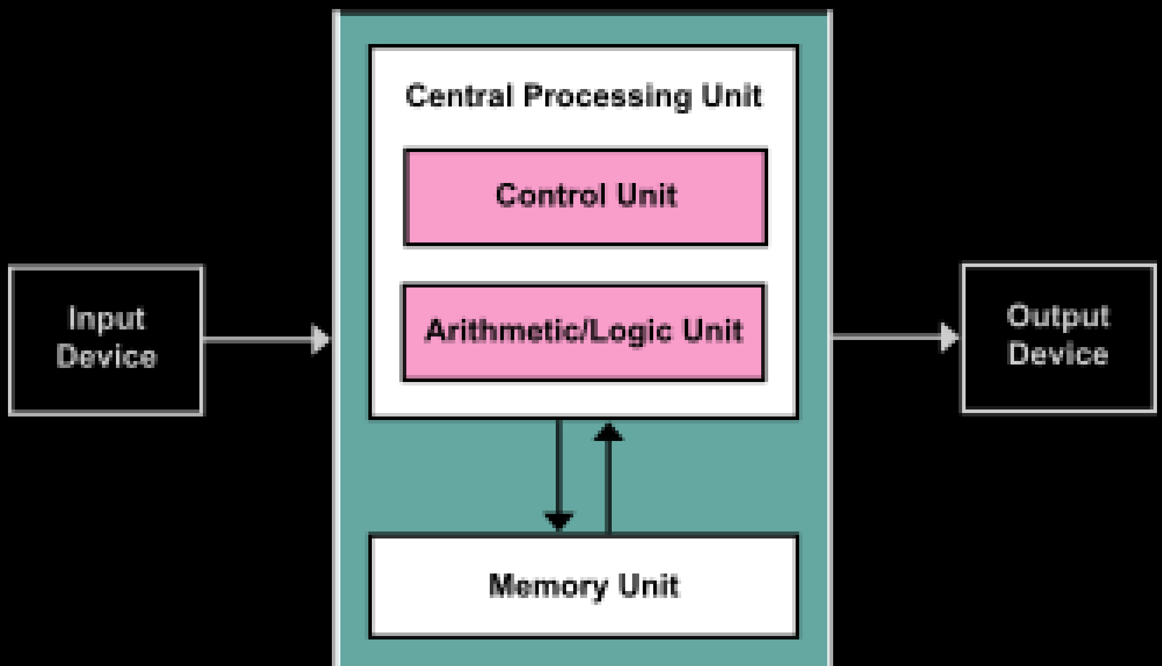
Scientists from the RAND Corporation have created this model to illustrate how a "home computer" could look like in the year 2004. However the needed technology will not be economically feasible for the average home. Also the scientists readily admit that the computer will require not yet invented technology to actually work, but 50 years from now scientific progress is expected to solve these problems. With teletype interface and the Fortran language, the computer will be easy to use.

Some more predictions (back then!)…



Key Points

1) The standard computer (von Neumann) architecture consists of a central processing unit (CPU) and memory that stores both instructions and data.



2) Understand the Fetch/Execute cycle performed by the CPU.

What Computers Can and Cannot Do

Computers can only deterministically **perform or execute instructions** to process information. The computer must have instructions to follow.

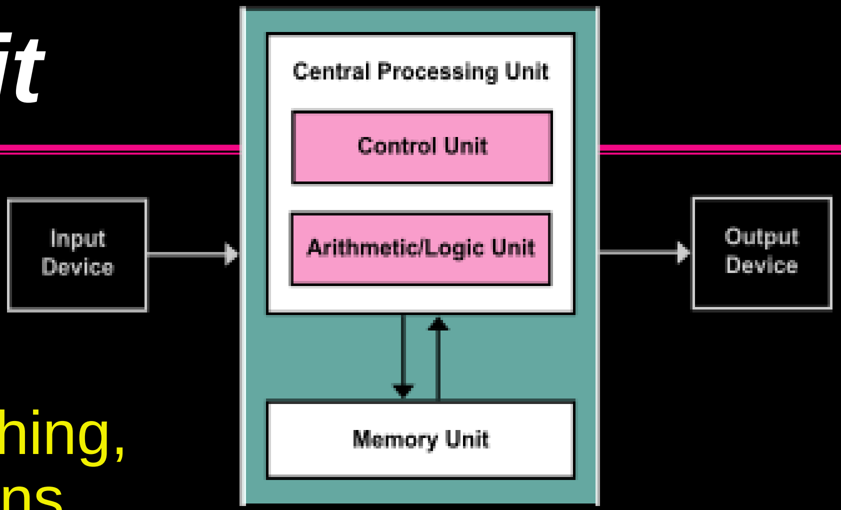
A computer has **no imagination, intuition, or emotions**. It has **no intelligence, free will, or its own purpose**.

We must specify precisely what the computer should do in the form of instructions and the associated data. We will see how a computer processes our instructions.

A computer is useful **not because it is "smart" but because it can do simple operations very quickly**.

A CPU at 1 GHz can perform about 1 billion operations/second.

Central Processing Unit



A computer contains a ***Central Processing Unit*** (***CPU***) with several components:

- ***control unit*** – responsible for fetching, decoding, and executing instructions
- ***arithmetic/logic unit*** (***ALU***) – responsible for performing mathematical functions (addition, subtraction, etc.) and logical operations (AND, OR, NOT)
- ***registers*** – specialized memory locations used during execution

The CPU is connected to input/output devices and main memory using the motherboard ***system bus*** (set of wires).

This architecture is referred to as a ***von Neumann architecture*** because it uses memory to hold both instructions and data.

- The term came from the work of John von Neumann in 1945.

Memory

Memory stores programs and data.

Memory is divided into ***locations***. Each location has an ***address*** (e.g., 32-bits) and can store 1 byte.

- We will show memory with each big cell consisting of 4 bytes (one ***word***).
- Addresses are in decimal and start at 0.

locations

addresses

	0	1	2	3
0	00	00	00	00
4				
8				
12				
16	A	B	C	D
20				
24				
28	12	34	56	78
32				
36				
40				
44				
48				

. . .

Memory

Questions:

1) What byte value is at address 0?
2) What byte value is at address 17?
3) What word value is at address 16?
4) What word value is at address 28?

Aside: Storing Characters in Memory

Recall that an ASCII character occupies one byte of space.

In the memory is the string HELLO WORLD stored at address 0.

- Remember that the computer stores things in binary. To the computer, this looks like: 48 45 4C 4C 4F 20 57 4F 52 4C 44 00.
  - Note that A = 65 = 41 (hex), B = 66 = 42 (hex), etc. Also, we terminated the string with NULL (00).

Questions:

1) Store the string "COMPUTER" in memory starting at address 20. End string with 00.
2) Store your name as a string starting at 16.

	0	1	2	3
0	H	E	L	L
4	O		W	O
8	R	L	D	
12				
16				
20				
24				
28				
32				
36				
40				
44				
48				

...

Memory

Computer Instructions

The CPU has hardwired only a very few basic operations or instructions that it can perform:

- **read** a memory location **into** a register
- **write** a register value **to** a memory location
- add, subtract, multiply, divide values stored in registers
- **shift** bits left or right in a register
- test if a bit is zero or non-zero and jump to new set of instructions based on the outcome
- sense signals from input/output devices

All programs are composed of these basic operations.

The Fetch/Execute Cycle

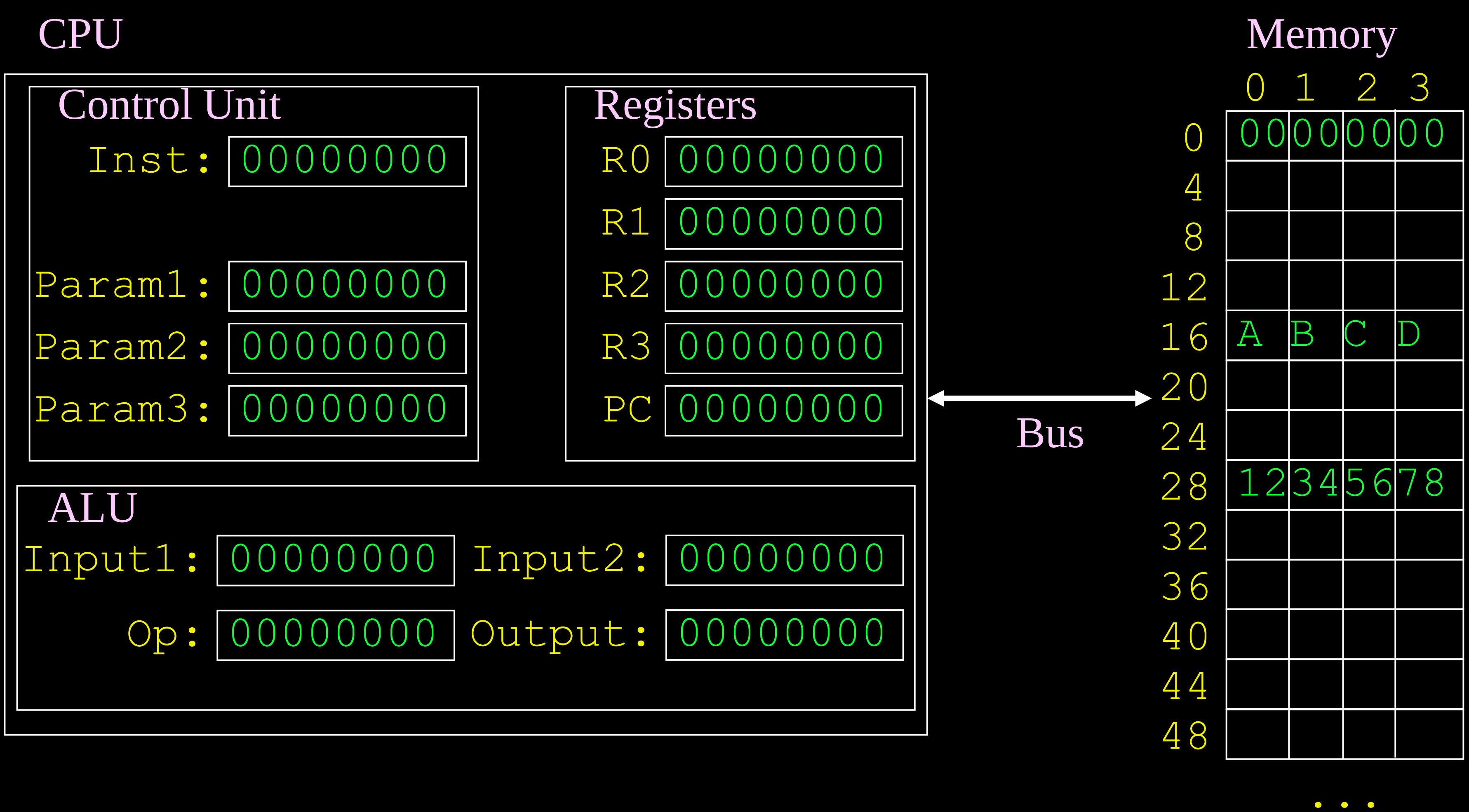
The computer performs the following cycle of operations to process instructions:

- Instruction Fetch (IF) - retrieve instruction from memory
- Instruction Decode (ID) - lookup meaning of instruction
- Data Fetch (DF) - fetch data for instruction
- Instruction Execution (IE) - execute instruction
- Result Return (RR) - return result to register

A special register called the ***program counter*** (PC) stores the address of the next instruction to execute.

- Since each instruction is 4 bytes long, the PC is incremented by 4 every time an instruction is executed unless a *branch* is performed.

CPU and Memory Diagram



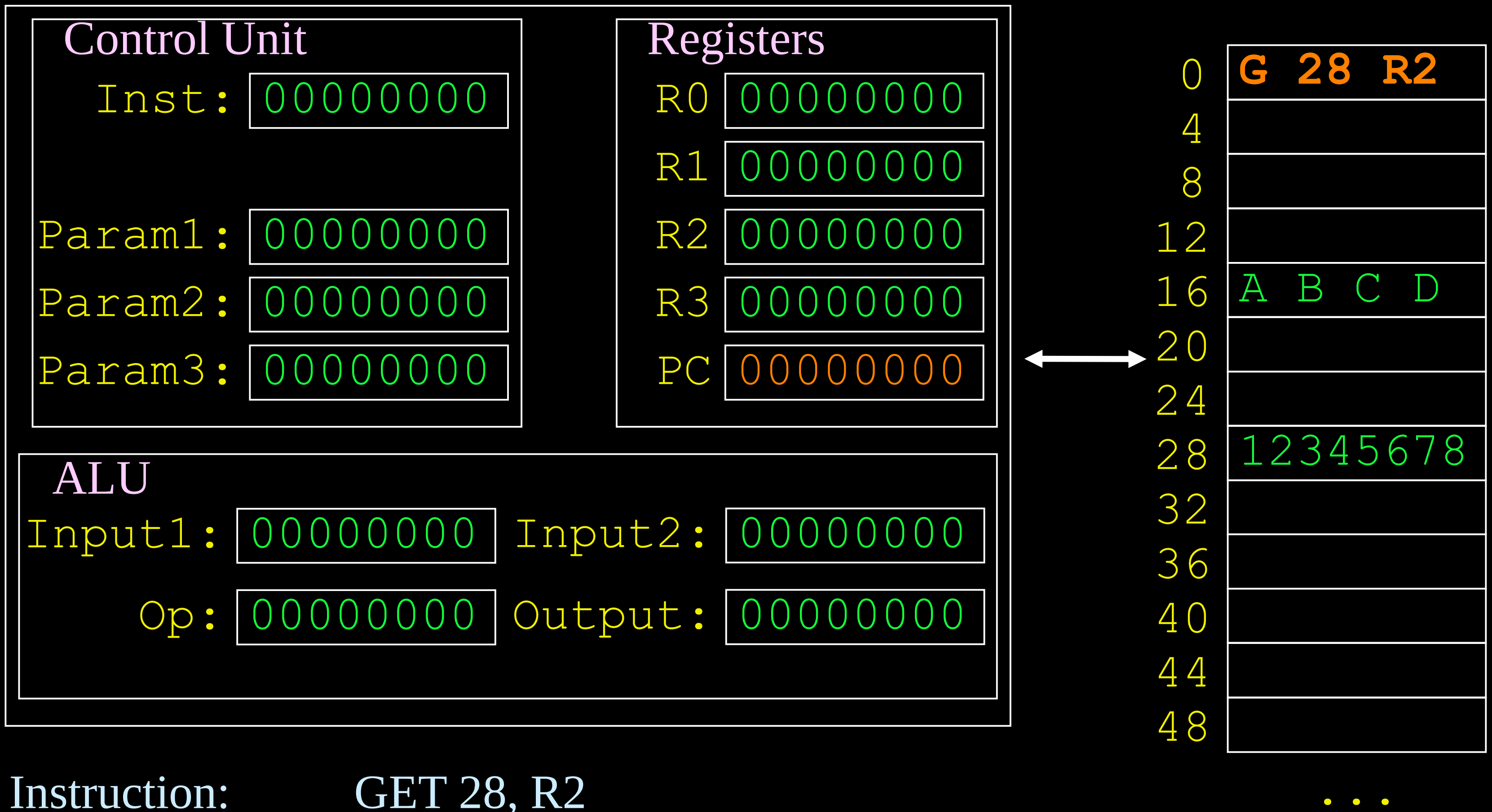
Decoding Computer Instructions

How the instructions are encoded in bits depends on the processor and computer architecture. Just like with the ASCII lookup table, each bit sequence represents some instruction.

We will encode instructions using simple character strings.

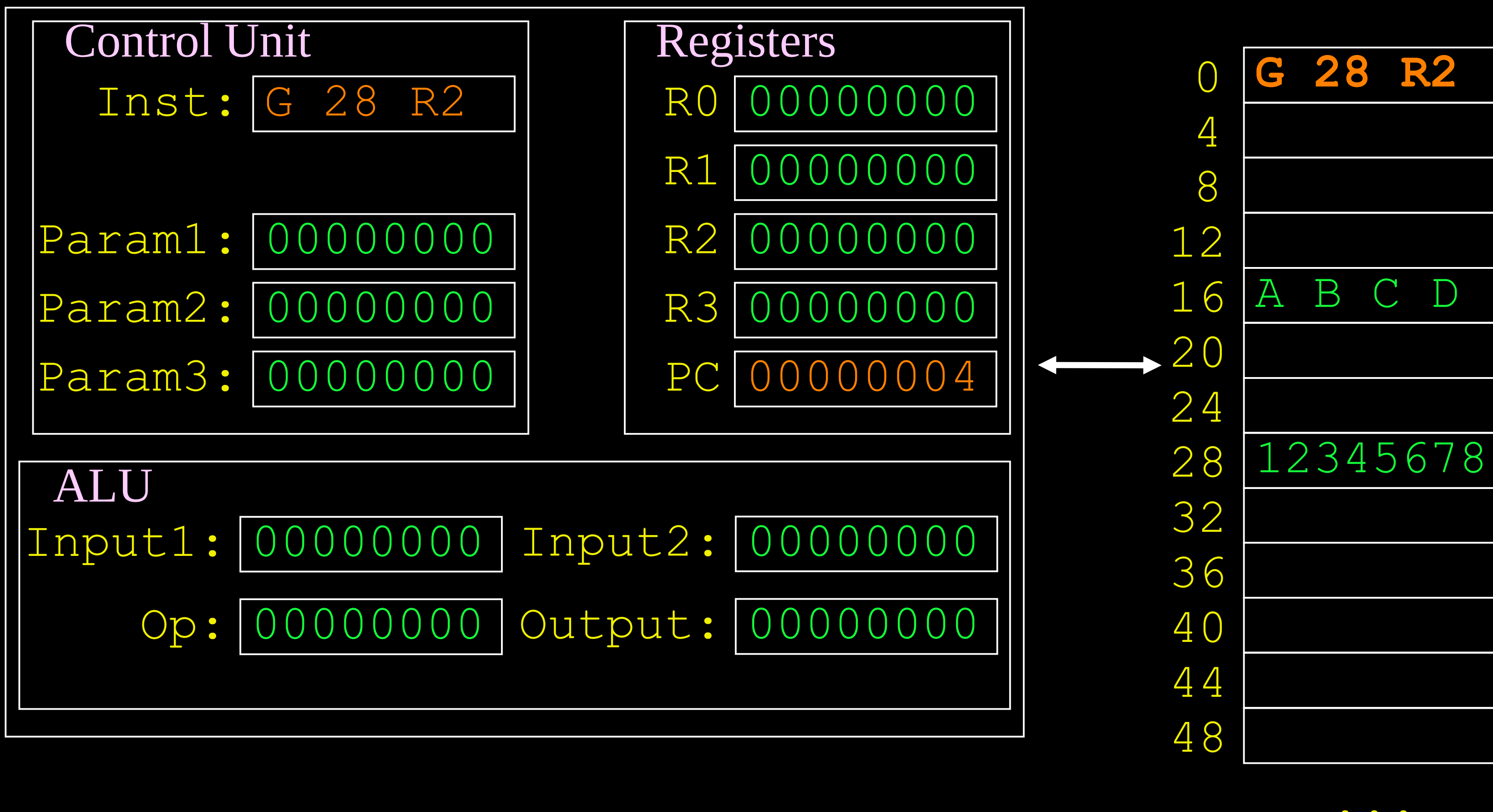
- Each instruction has one, two, or three *parameters* or *operands*.
- **G [address] [register]** – Get data from memory at address and put in to the register
  - e.g. G 16 R1 – Get address 16 and put in register 1
- **P [address] [register]**-put data in register to memory at address
  - e.g. P 20 R2 – Put data in register 2 into memory address 20
- **+ [reg1] [reg2] [reg3]** – Store in reg3 result of reg1+ reg2
- **- [reg1] [reg2] [reg3]** – Store in reg3 result of reg1- reg2
  - e.g. + R0 R1 R2 – Store in register 2 result of register 0 + register 1

Example:
Executing Move Instruction



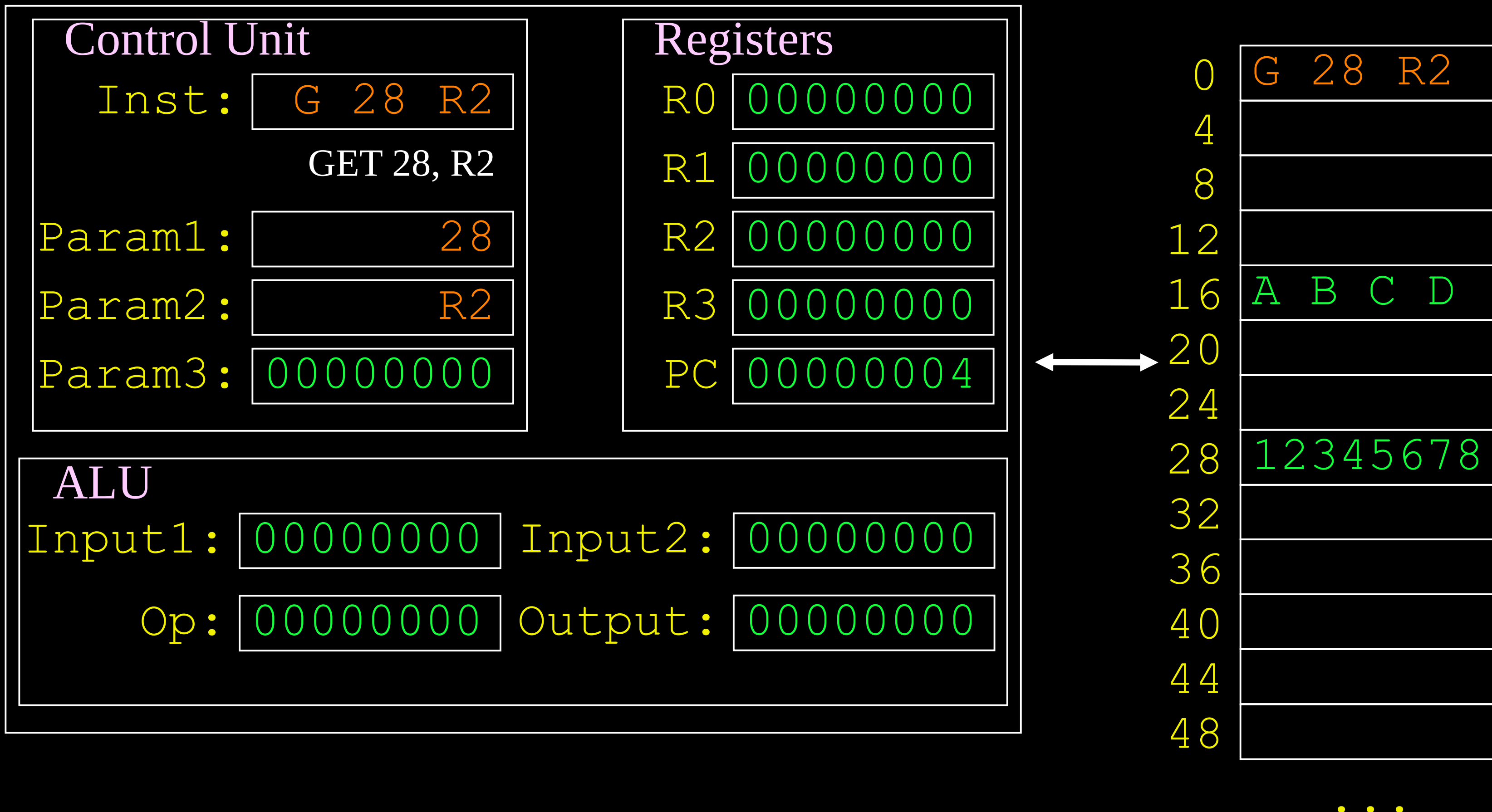
Instruction: GET 28, R2

Example:
Executing Move Instruction - Fetch



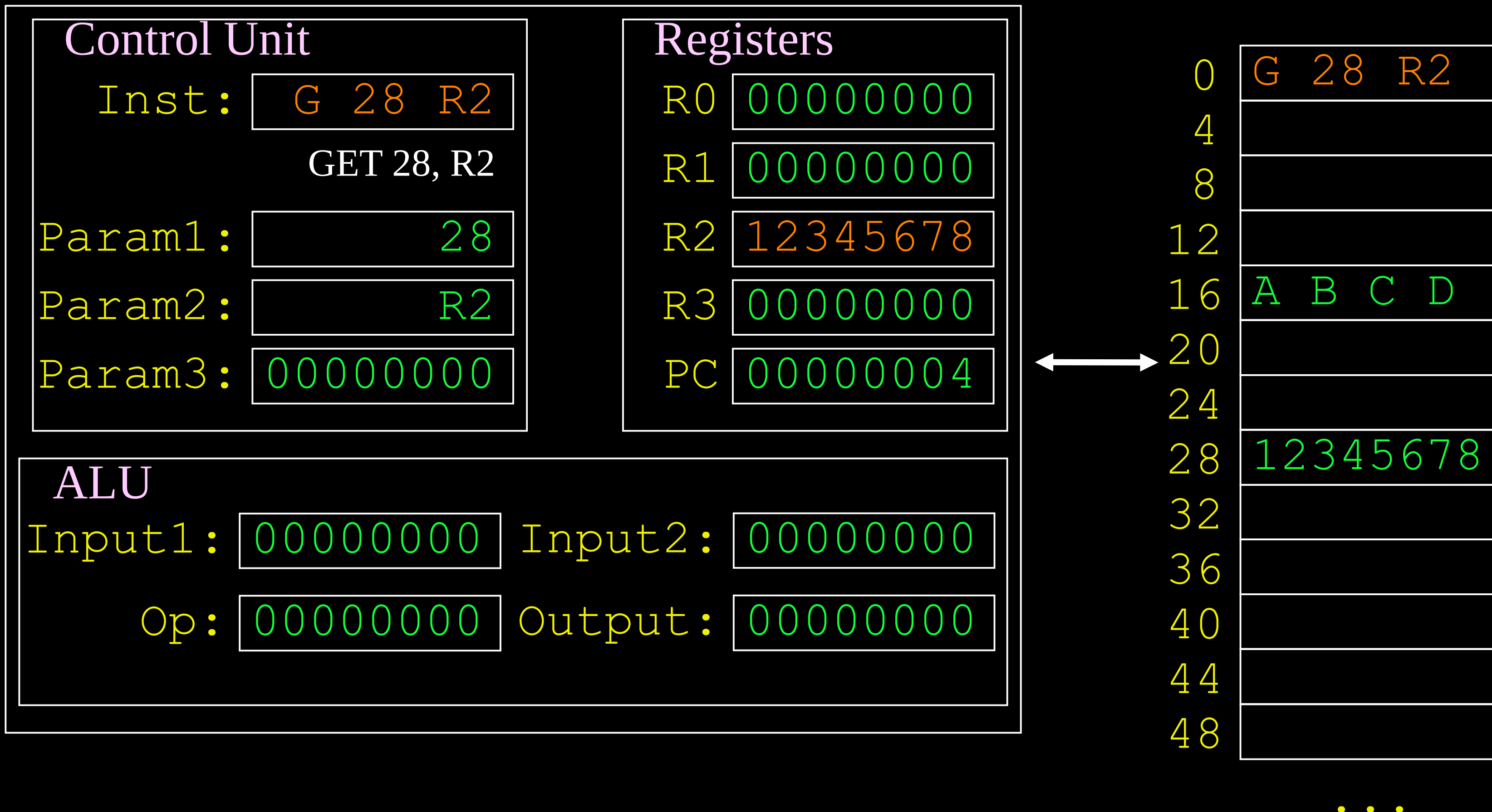
Program counter is for address 0. Fetch from memory.
Increment program counter by 4.

Example:
Executing Move Instruction - Decode



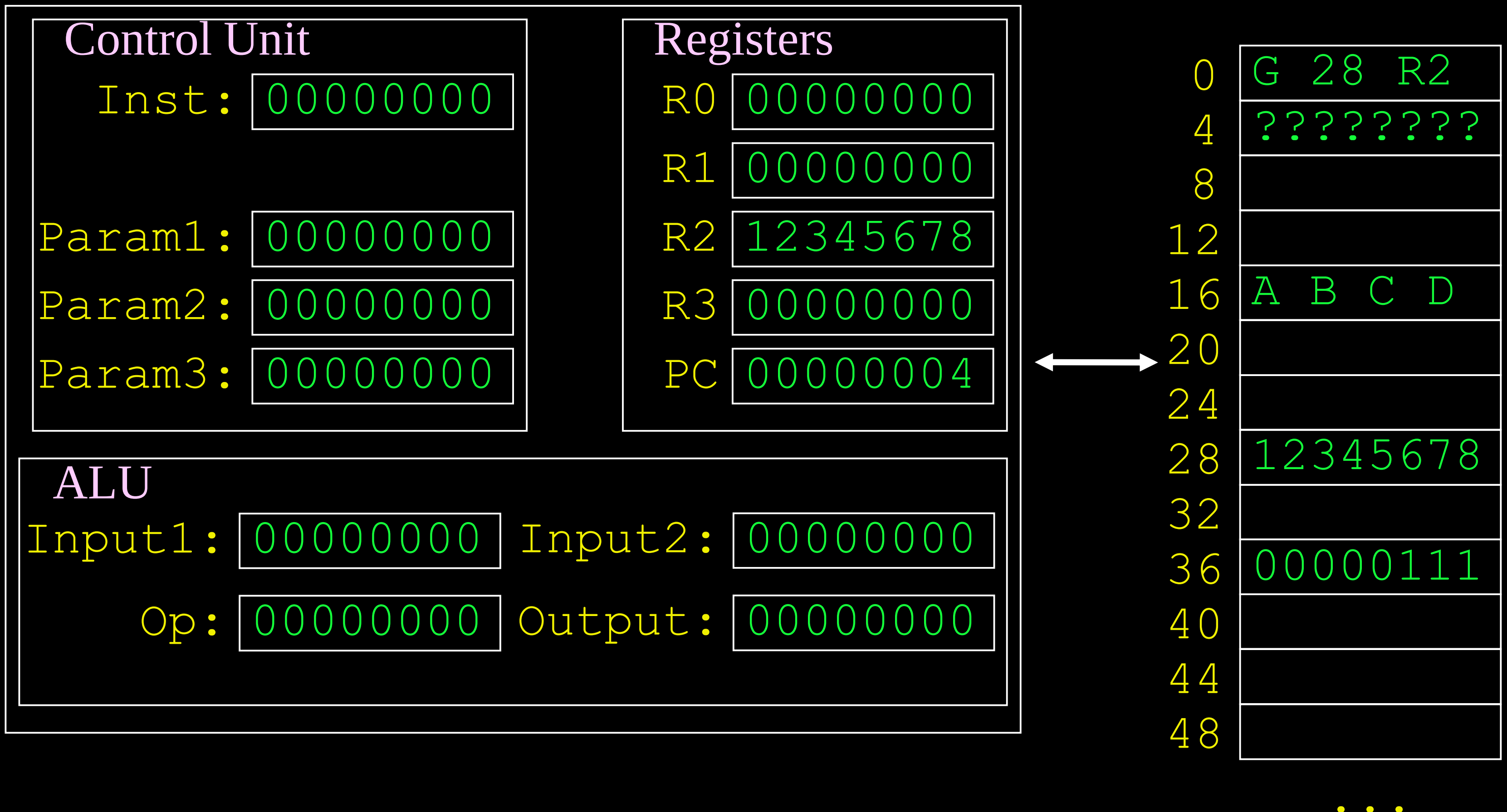
Decode instruction. It is a move instruction.
Set param1 to be 28 for memory address and param2 to be register 2.

Example:
Executing Move Instruction – Execute



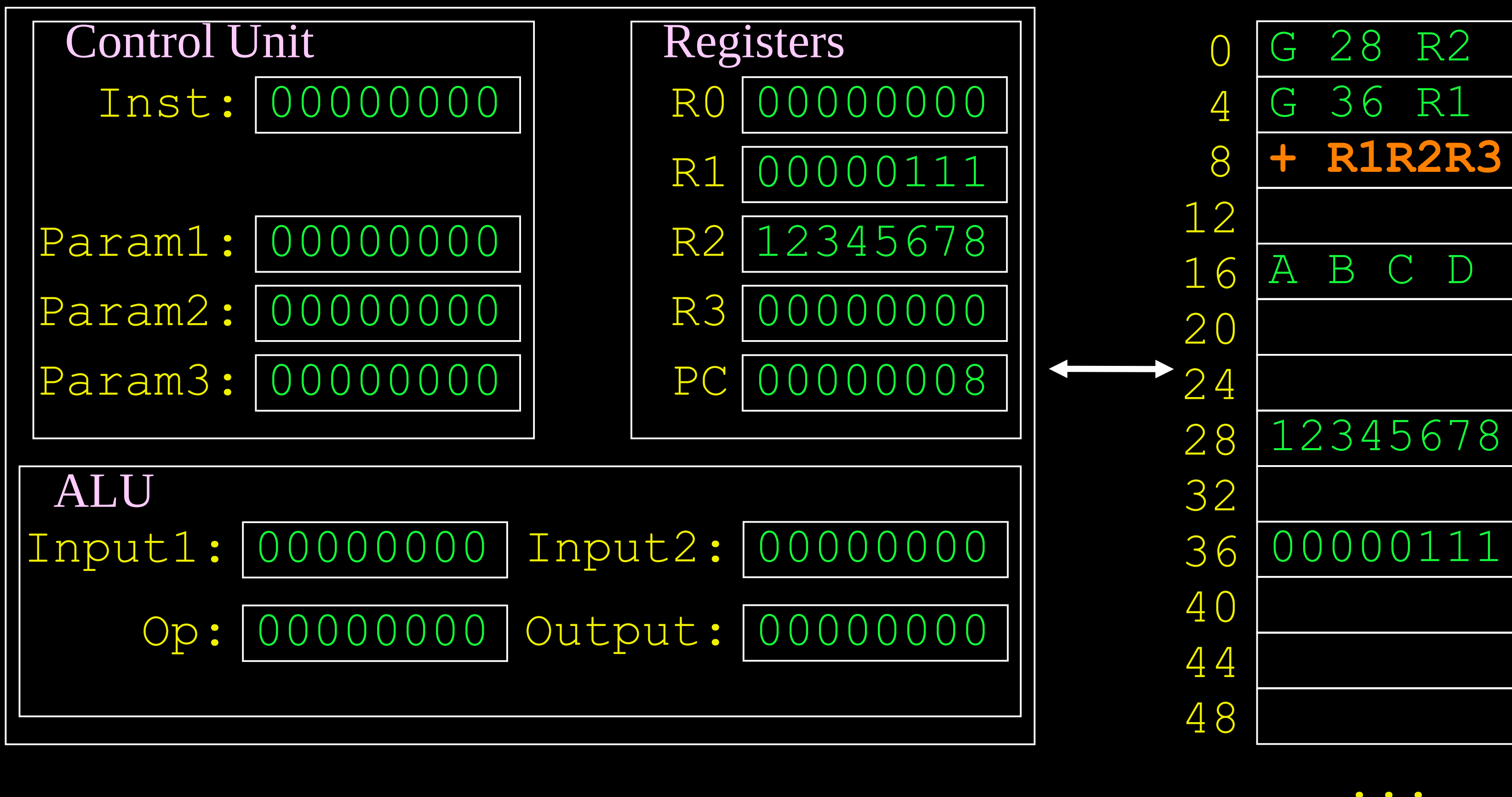
No data fetch (performed during execute).
Instruction execution: Fetch memory location 28 and put in R2.
No result return.

Question: Instruction Execution



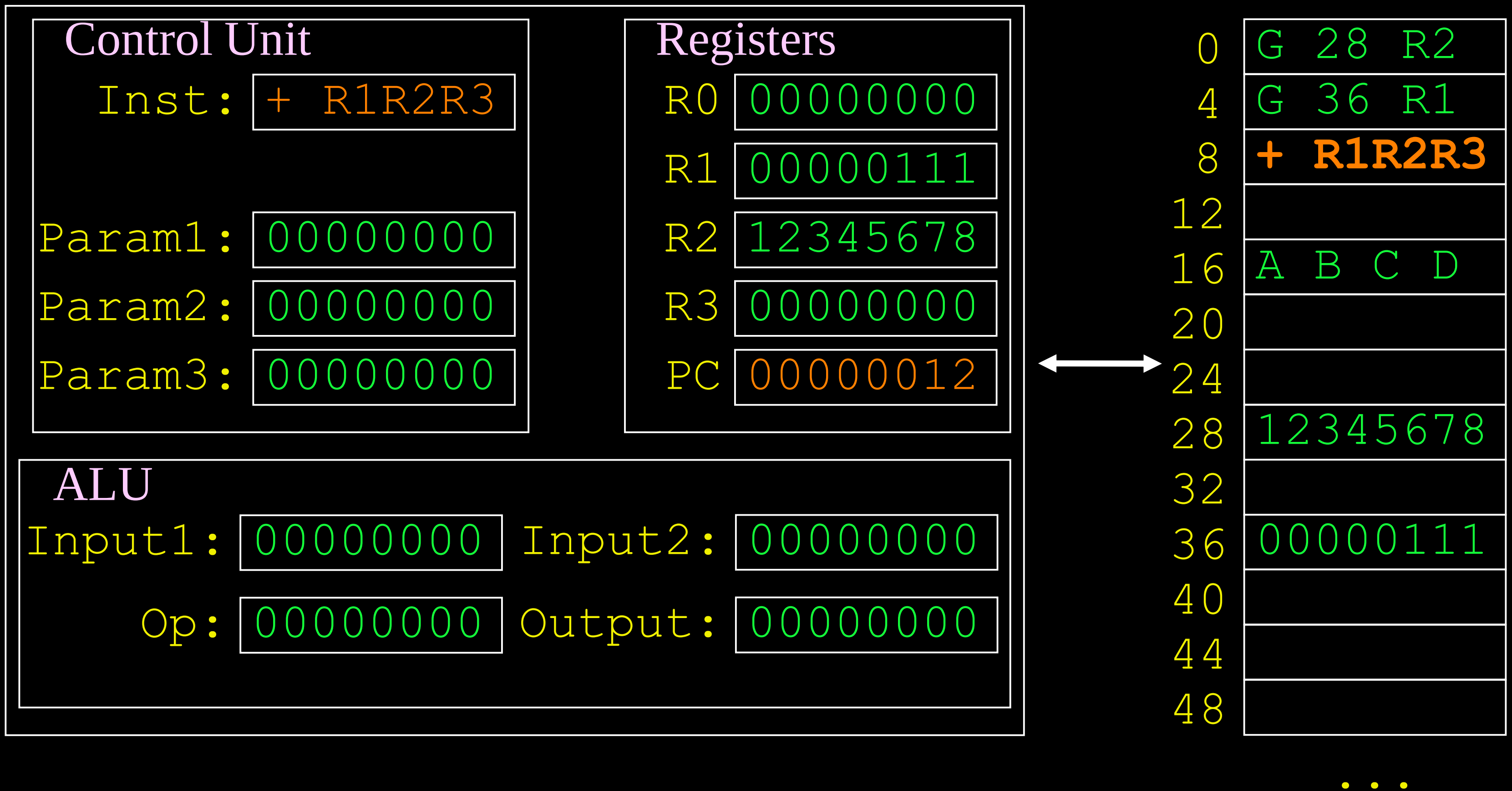
Question: Write the instruction GET 36, R1.
Put this instruction in location 4 and explain how it gets executed.

Example:
Add Instruction Execution



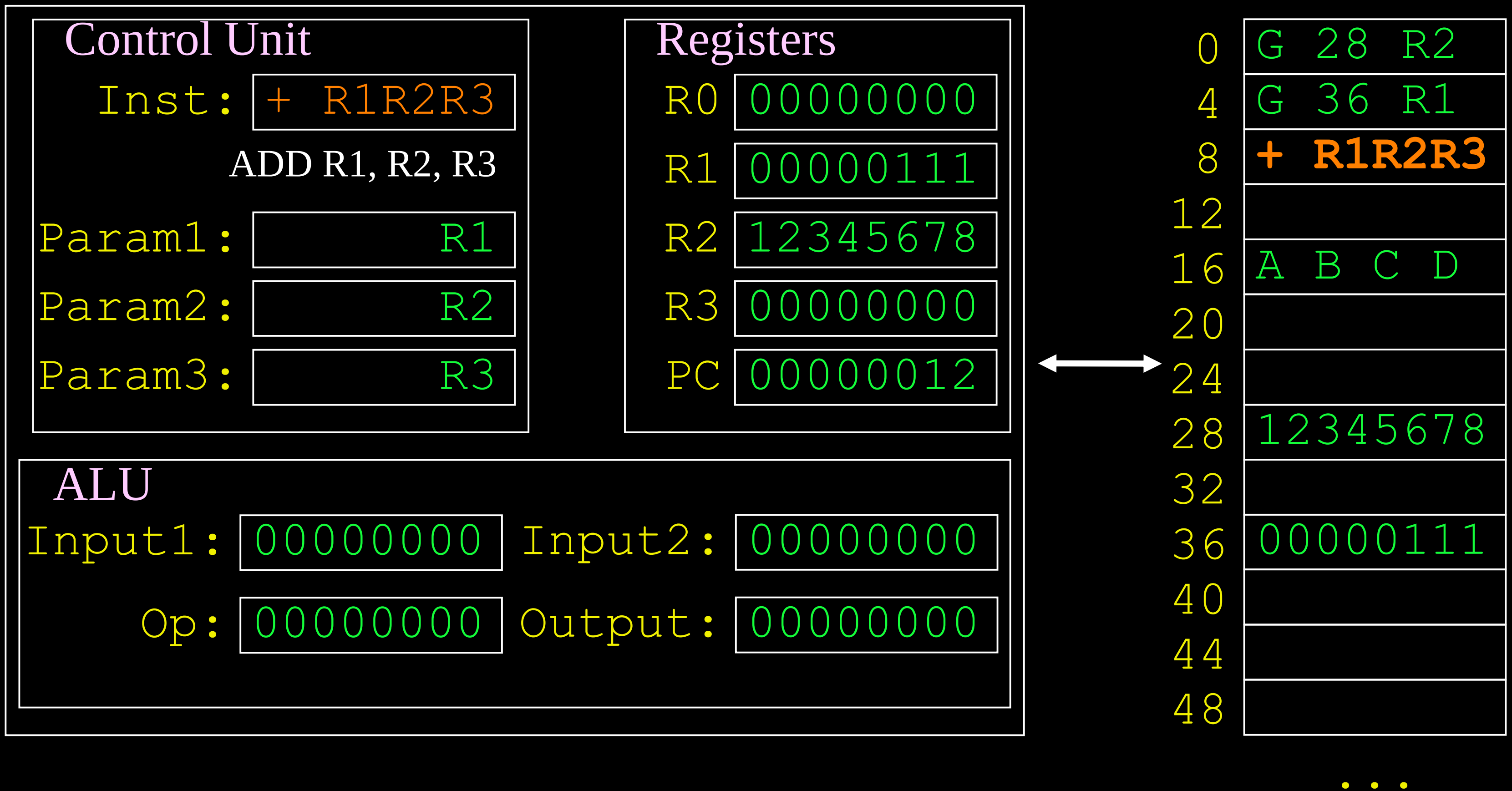
Instruction: ADD R1, R2, R3

Example:
Add Instruction Execution - Fetch



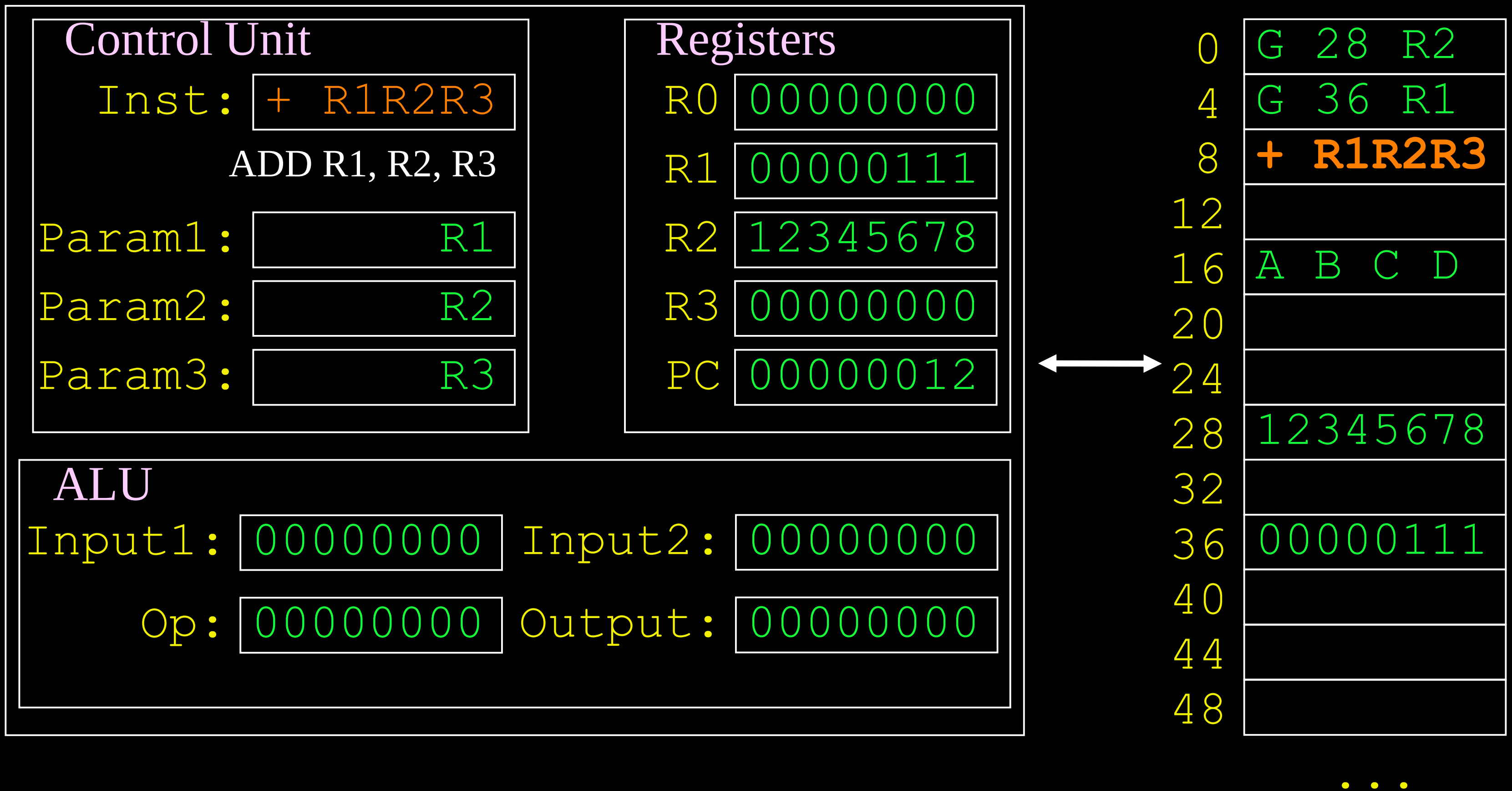
Program counter is for address 8. Fetch from memory.
Increment program counter by 4.

Example: Add Instruction Execution - Decode



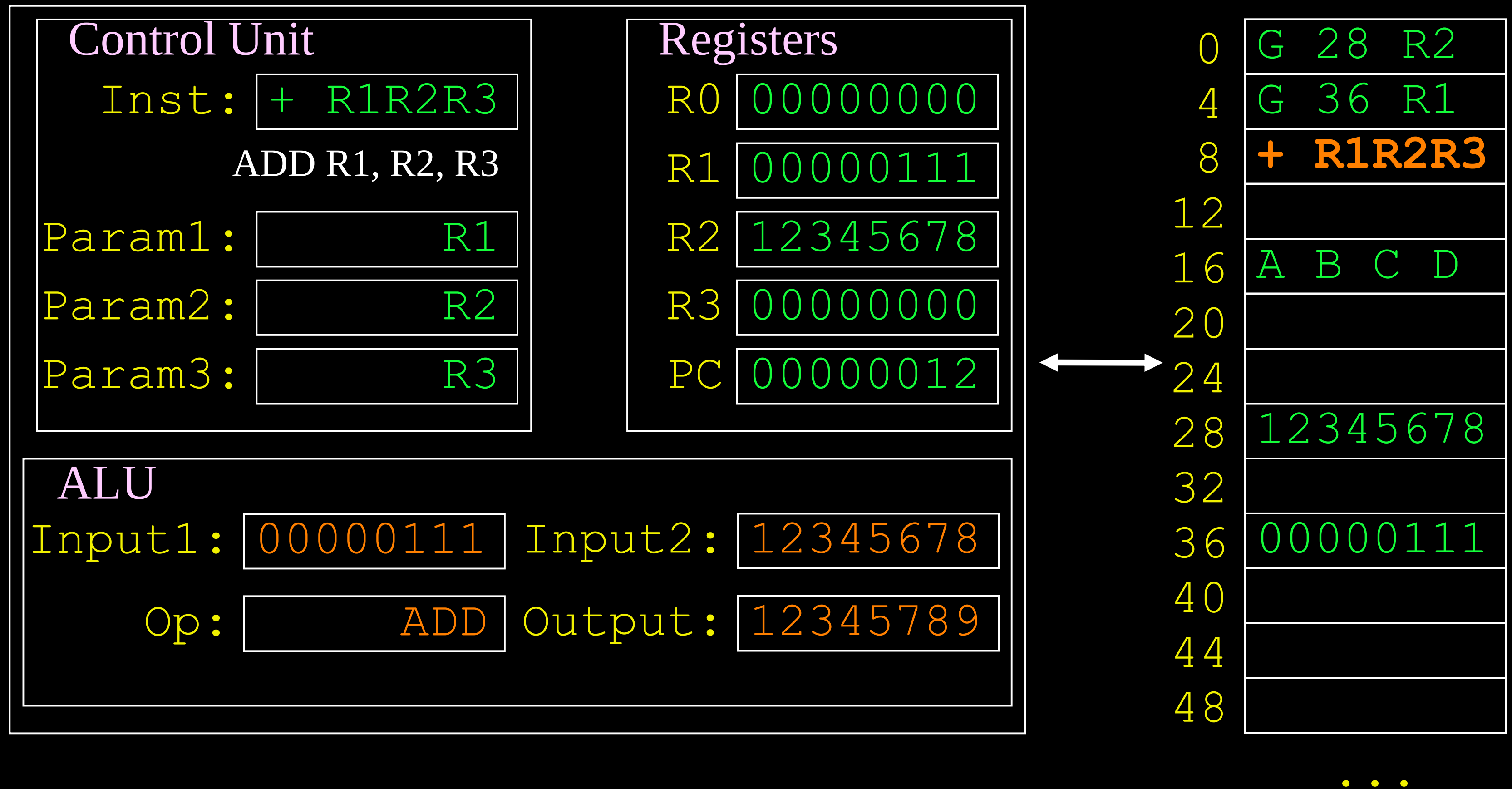
Decode instruction to determine it is an add. Set param1 to R1 (register 1), param2 to R2 (register 2), and param3 to R3 (register 3).
Prepare ALU to receive command and inputs.

Example: Add Instruction Execution – Data Fetch



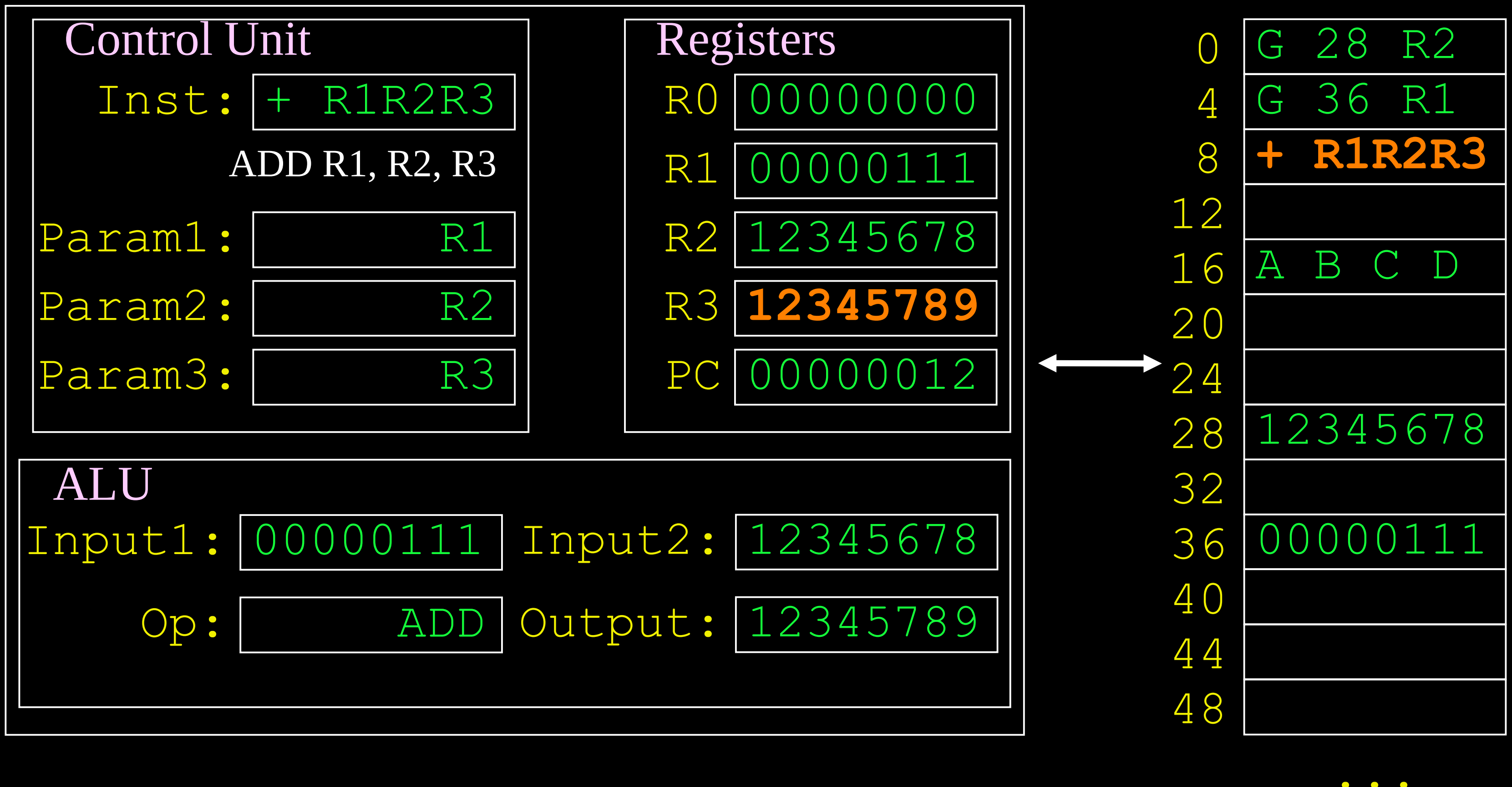
No data must be fetched from memory. Nothing to do in this step.

Example:
Add Instruction Execution – Execute



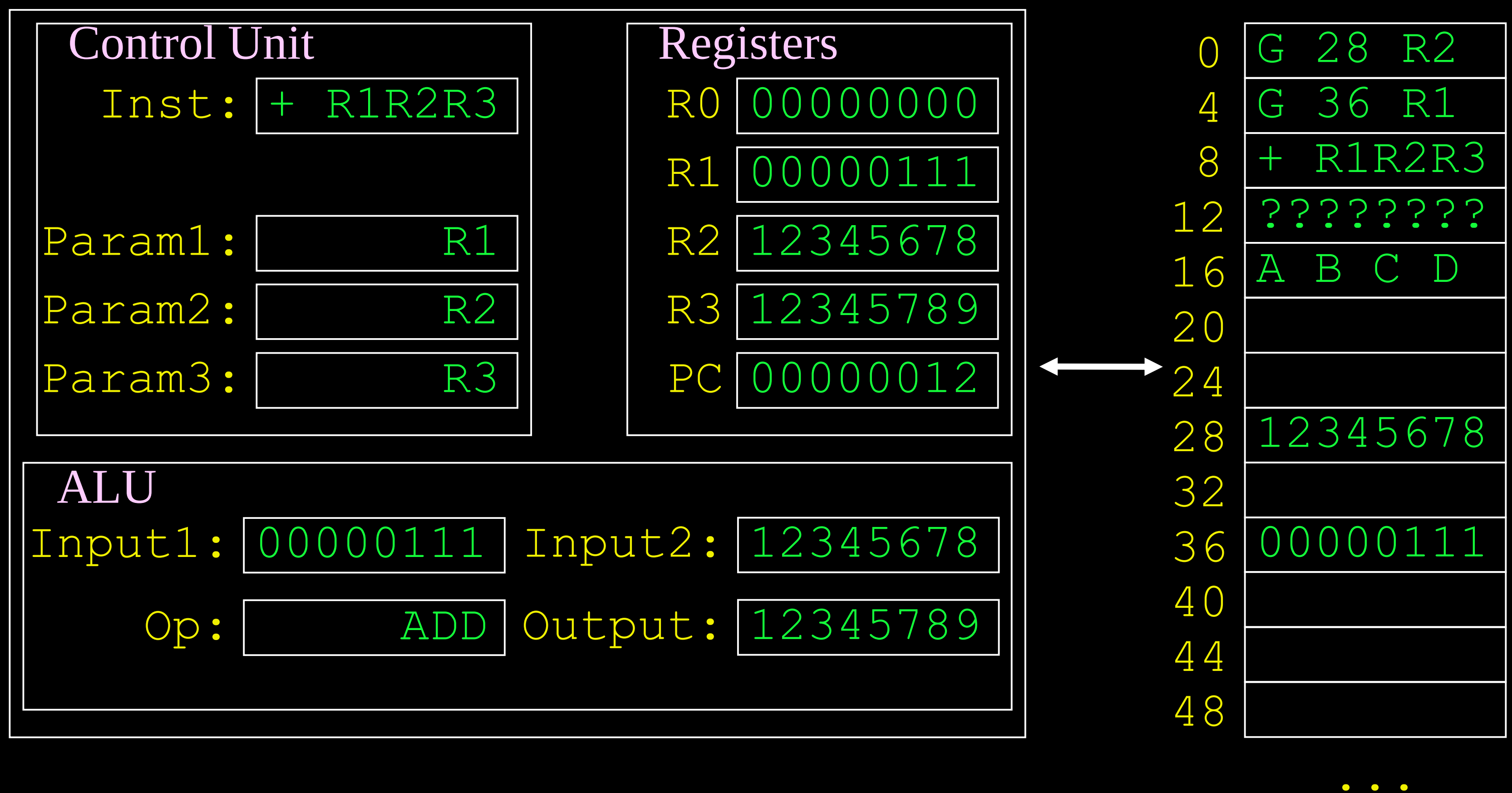
Execute instruction by passing operation and parameters to ALU.
Assume ALU knows operation is an ADD.
ALU executes the add (which may take some time) and result is in output.

Example:
Add Instruction Execution – Return



Output result from ALU is returned into register 3 as required.

Question: Instruction Execution



Question: Encode the instruction to put the data in register 3 into memory address 40. Instruction goes in address 12.

Explain how this instruction gets executed.

CPU

Question: Which of these is ***NOT*** a component of the CPU?

A) control unit

B) arithmetic logic unit

C) bus

D) registers

Fetch/Execute Cycle

Question: Put the steps in order for the Fetch/Execute cycle:

1) Data Fetch (DF)

2) Result Return (RR)

3) Instruction Execution (IE)

4) Instruction Fetch (IF)

5) Instruction Decode (ID)

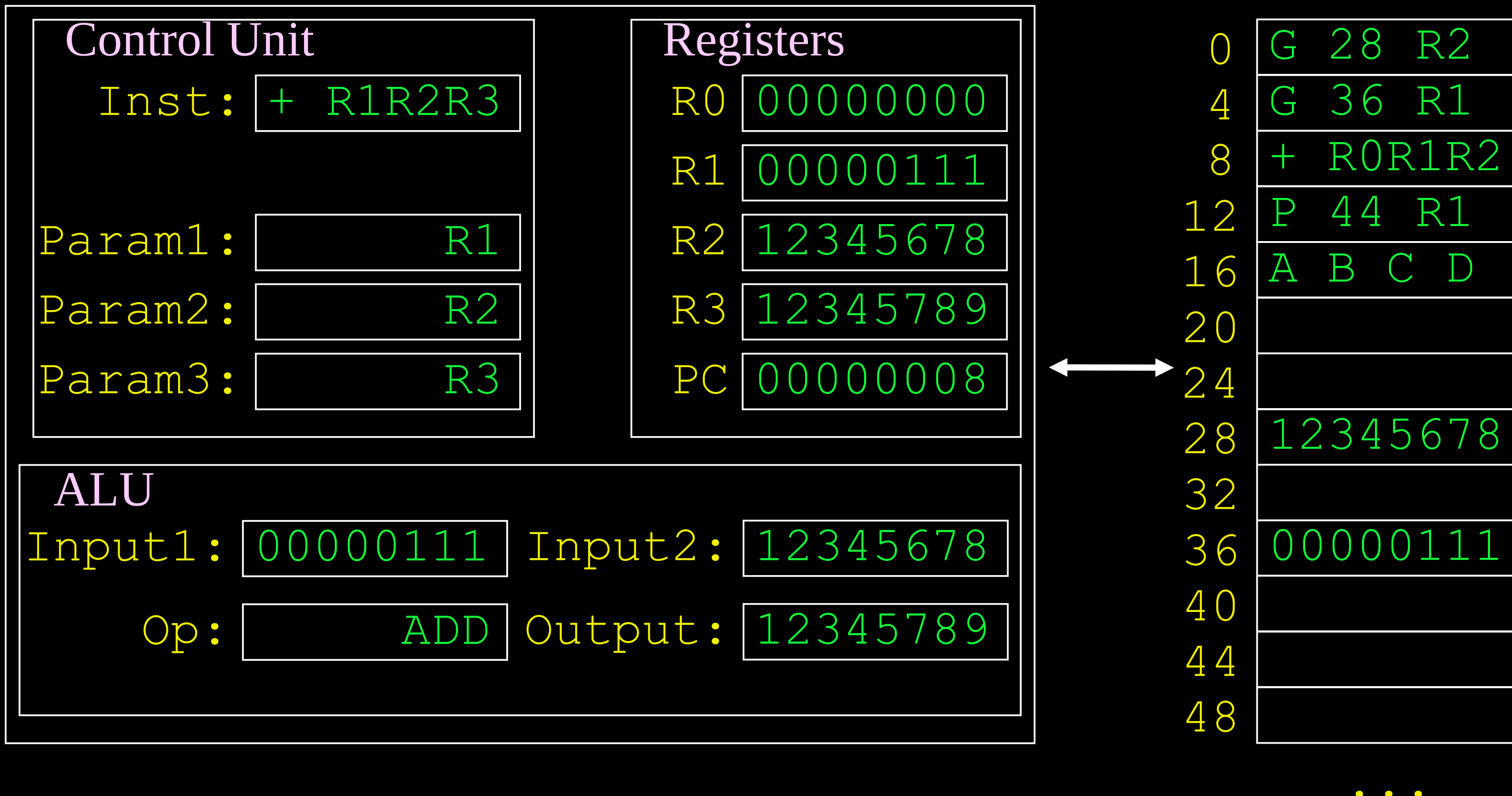
a) IF,ID,DF,IE,RR

b) ID,IF,DF,RR,IE

c) IF,IE,ID,DF,RR

d) IF,DF,IE,ID,RR

Instruction Execution Result



Question: What is the value of R2 after executing the statement at location 8?

A) 00000000 **B)** 00000111 **C)** 12345678 **D)** 12345789 **E)** None of the above

Challenge Question: Writing a Simple Program

Write a simple program that computes the following:

result = (A + B) * C

52		A
56		B
60		C
64		result

- Assume A is at location 52, B is at 56, C is at 60.
- Let A=5, B=2, C=10 then the result should be 70.
- Store result at location 64.
- Your program instructions should begin at address 0.

Be prepared to explain how the program works when executed.

HINT: You will need 6 instructions:

- 3 `GET`,
- 1 `ADD`,
- 1 `MULTIPLY`,
- 1 `PUT`. Use the “*” to denote a multiply expression.

Assembly and Machine Programming

The previous examples are similar to assembly programming.

Machine programming involves specifying commands **directly in binary form**.

- e.g., 10011010100100101010010

Assembly language is a slightly higher level of commands which look more like English commands that are then translated to machine language before execution.

- e.g., move, add, sub, etc.

Most programmers **do not write code** in assembly or machine language because it is too low-level and time-consuming.

Higher-Level Programming

Higher-level programming languages (such as HTML and JavaScript) are more powerful and easier to use because they have more powerful features and functions.

- The programmer does not have to specify all the details at a low-level and can use more general commands.
- Note that this is another form of *abstraction*.

Every language for communicating instructions to the computer **must ultimately be translated to machine language** for execution.

- The tools that translate to machine language are called ***compilers***. Compilers verify that code has correct syntax before performing the translation.

Branch and Jump Instructions

One type of instruction that is available in **all languages** is called a ***branch*** or ***jump instruction***.

A branch instruction allows the program to execute different parts of code depending on certain conditions. Example:

```
IF (you are late) THEN
   take the bus to work
ELSE
    walk to work
```

A branch instruction is implemented by making a decision whether or not to branch (usually a comparison) then setting the program counter to the address of the next instruction.

Computer Speed

The speed that a computer can execute a program depends on many things including:

- the speed of the CPU
- the speed of the bus, memory, and other devices
- the type of program and its characteristics
- the amount of parallelism and pipelining in the CPU

Historical example:

Apollo Guidance Computer had 2.048 MHz processor, 32KB of RAM, 4KB of ROM, and 8 16-bit registers.

Computer Speed in GHz

The most basic measurement is the speed of the ***CPU clock*** because it is a rough estimate of the number of instructions that can be executed per second.

CPU speed is measured in ***hertz*** or ***cycles per second***. The clock of typical CPUs perform billions (giga-) cycles per second, so the measurement is in giga-hertz (GHz).

- A computer with a 2 GHz CPU has the potential for executing 2 billion instructions per second.

Note that measuring computer performance simply on clock speed has been used as a **marketing tool**. As computers have become faster and more complex, CPU clock speed in GHz is not the best measurement.

Aside: Advanced Processor Issues

Our explanation of how a processor works is a high-level abstraction of how they work in practice.

Processors may have multiple dedicated hardware, complex pipelining features, cache memory, and other optimizations.

Some other terminology:

- ***dual/quad core*** – means that there are two/four **processing units** on the **same chip**. The units may **share** subcomponents.
- ***dual processor*** – means that there **are two separate processing units** on different chips. Each processor appears distinct to the operating system.
- ***32-bit or 64-bit*** – describes the size of the basic memory unit and is also related to the bus size.

Operating Systems

An operating system (OS) is software written to perform the basic operations that are necessary for the effective use of the computer that are not built into the hardware.

Three most widely used Operating Systems:

- Microsoft Windows, Apple's Mac OS X, and Linux/Unix

The OS performs

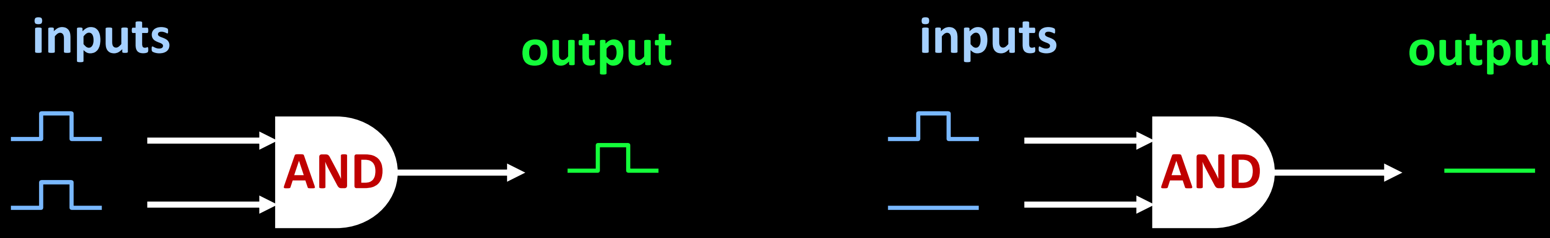
- **booting**,
- **memory management,**
- **device management,**
- **file management**,
- **Internet connection,** and
- provides a platform for the execution and development of programs.

Computers and Electricity

Computer components consist of gates and circuits that control the flow of electricity.

A ***gate*** is a **device** that performs a **basic operation** on electrical signals.

- Common gates: `AND`, `OR`, `NOT`, `XOR`



A ***circuit*** is a combination of gates that performs a more complicated task.

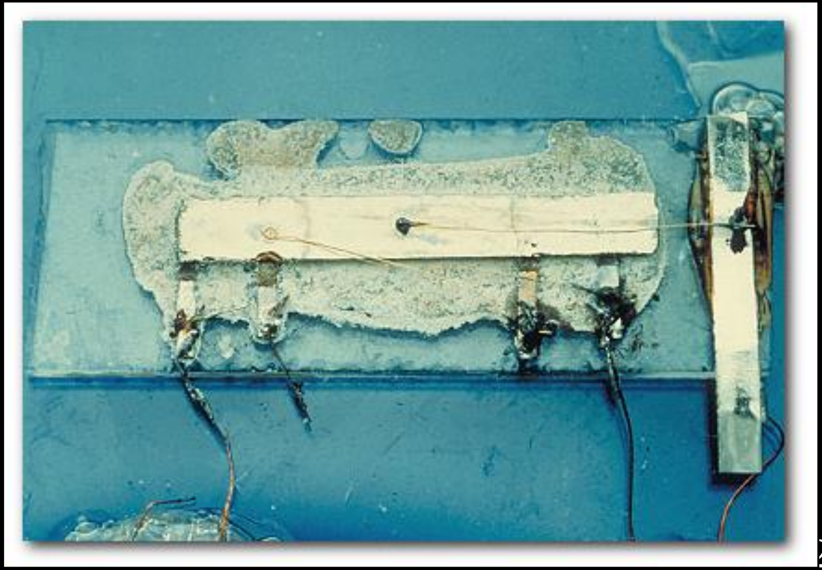
Constructing Gates using Transistors

A ***transistor*** may either conduct or block flow of electricity based on input voltage (functions like a switch).

- Made of ***semiconductor*** material such as silicon.

An ***integrated circuit*** **(IC)** is contains both transistors and wires that connect them. manufactured during same process.

- Invented by Jack Kilby and others at Texas Instruments in 1958. They received the Nobel Prize in Physics in 2000.
- First integrated circuit:



Summary: Putting it All Together

- An application is written by a programmer to solve a task using a programming language.
- The application uses features of the operating system to perform certain functions.
- The program is translated (compiled) into machine language for the computer to use. This form is simply a sequence of bytes.
- The byte sequence (binary file) is read from the hard drive into memory by the operating system when executed.
- The commands are executed using the fetch/execute cycle.
- The commands are implemented in hardware on silicon on integrated circuits that are produced using photolithography.
- The CPU contains a control unit and arithmetic logic unit that performs the basic operations. By controlling the flow of electricity, different states and operations are performed.

Conclusion

The standard computer (von Neumann) architecture consists of a CPU, memory, a bus, and input/output devices.

The five basic steps of the ***fetch/execute cycle*** are:

- Instruction Fetch
- Instruction Decode
- Data Fetch
- Instruction Execution
- Result Return

Hardware commands are encoded on integrated circuits using gates that consist of transistors etched on silicon (semiconductor).

Objectives

- Describe the von Neumann architecture (computer anatomy). Draw the diagram, and list and explain its main components.
- Explain the organization of memory in terms of locations and addresses.
- Define and list examples of: input/output device, peripheral
- List and explain the three major components of the CPU.
- Advanced: Explain the key feature of the von Neumann architecture.
- List some of the basic CPU instructions.
- List and explain the five steps of the fetch/execute cycle.
- Explain the purpose of the program counter register.
- Advanced: Explain how instruction decoding works and be able to decode an instruction using our format.

Objectives (2)

- Be able to explain and demonstrate the fetch/execute cycle for a small program.
- Define: machine language, assembly language
- Explain the difference between a high-level programming language and assembly/machine language.
- Define: compiler
- Define: branch instruction
- List some factors in determining a computer's speed.
- Define: gate, circuit, integrated circuit, transistor, semiconductor