Paper Computer Workshop

How Computers *Really* Work (with Paper)

Duration: 2 hours | **Audience:** Beginners to experienced technologists

Outcome: Build → Run → Understand → Modify → Create

LEARNING GOALS

What You'll Master Today

By the end of this workshop, you'll move from confusion to clarity. Computing will stop being magic and start being comprehensible. Here's what you'll actually be able to do:

Explain Core Components

Understand what **memory, registers, inputs, outputs, and programs** actually are—not as abstract concepts, but as concrete parts you can touch and manipulate.

Execute Programs Step-by-Step

Run a program one instruction at a time and **predict what happens next**. You'll develop the ability to think like the computer thinks.

Read Assembly Language

Interpret a **mnemonic instruction set**—the building blocks of all software—and understand how simple commands create complex behaviors.

Modify and Create Programs

Change existing code with intention and **write your own tiny programs**. You'll understand the "why" behind every edit.

Connect Paper to Modern Computing

Map what you build today to real software and hardware. You'll see your laptop, phone, and cloud servers with new eyes.

Workshop Journey

We'll move through five distinct phases, each building on the last. This isn't a race—it's a carefully designed progression from concrete to abstract, from building to understanding to creating.

1. 0:00–0:05
   Welcome & Framing
   Set expectations and understand why this matters

2. 0:05–0:50
   Build the Paper Computer
   Physical assembly of all components

3. 0:50–1:05
   What You Built
   Conceptual overview connecting parts to purpose

4. 1:05–1:30
   Run Your First Program
   Guided execution, step by step

5. 1:30–1:50
   Programming Fundamentals
   Modify existing code and create something new

6. 1:50–2:00
   Wrap-Up & Reflection
   Connect insights to modern computing

SECTION 1

Welcome & Framing (5 minutes)

This is not nostalgia. This is clarity.

Before we dive in, let's be crystal clear about what this workshop is and isn't. Understanding this frame will help you get the most out of the next two hours.

What this workshop *is*:

- A hands-on way to understand how computers actually think and process information
- A tool for demystifying the "black box" of computing
- An experience that makes abstract concepts tangible

What it's *not*:

- A cute toy or novelty project
- A history lesson about old technology
- A shortcut that skips the hard parts

Set Your Expectations

- We will go **slowly**—speed isn't the goal
- Mistakes are part of the point—they reveal how things work
- Confusion → insight is the desired path

> "If you understand this paper computer, modern computers stop being magic."

SECTION 2

Build the Paper Computer (45 minutes)

This is where theory becomes tactile. You'll physically assemble every component, understanding through your hands what programmers grasp through abstraction. This follows the assembly directions and component breakdown from the manual.

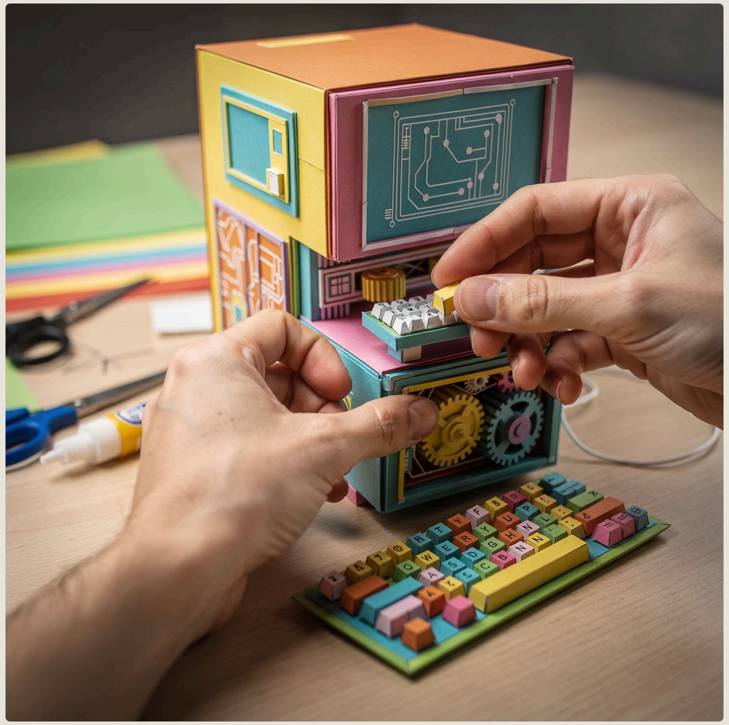
Step 1: Physical Assembly (30 min)

Participants will separate and install all major components. Take your time—precision matters more than speed.

Components to Install:

- Program Storage—where instructions live
- Main Storage—the computer's memory
- Registers A & B—fast temporary holding areas
- Inputs A & B—data entry points
- Outputs A & B—where results appear
- Jump switches & indicators—control flow mechanisms

You'll cut and insert paper strips, then clip in a Main Storage Unit sheet and a Program sheet (folded correctly). Every piece has a purpose.



Step 2: Sanity Check (15 min)

Before we write a single line of code, everyone needs to reset their computer to a known state. This ritual mirrors what real computers do on startup—and it's surprisingly important.

Program Step Indicator
Reset to **00**—start at the beginning

Index Counter
Reset to **00**—no position yet

Registers A & B
Clear to **0**—empty the workspace

Compare Unit
Set to **=**—neutral comparison

Outputs A & B
Clear—blank slate for results

Facilitator Tips

Circulate continuously during assembly. Pair participants if possible—teaching someone else is the best way to learn. Emphasize **legibility** over speed. Messy handwriting leads to debugging headaches later.

SECTION 3

What You Just Built: The Architecture Revealed

Now that you've assembled the physical computer, let's map what you built to the concepts that power every device you use. This is where the paper model becomes a mental model.

Component Mapping: Paper to Modern Computing

Paper Computer	**Modern Computing Concept**
Program Storage	Code memory / instruction storage where your apps live
Main Storage	RAM—the computer's working memory for active data
Registers A & B	CPU registers—ultra-fast temporary storage inside the processor
Inputs A & B	Keyboard, files, sensors—any way data enters the system
Outputs A & B	Screen, logs, printers—any way data leaves the system
Program Step Indicator	Instruction pointer—tracks which line of code is executing
Jump Instructions	Conditionals: if statements, while loops, goto commands
Compare Unit	Comparison operators for making decisions

Key insight: A computer doesn't "know" anything. It only moves symbols around very precisely, following rules without understanding. That's both its limitation and its superpower.

SECTION 4

Running Your First Program (25 minutes)

This is where everything clicks. We'll execute Practice Program A from the manual together, moving deliberately through each instruction. Speed is the enemy here—understanding is the goal.

Walk It Together (20 min)

We'll do this **slowly** and **out loud**. Every step matters. Here's the rhythm we'll follow:

01

Read the Current Program Step

What instruction are we on?

02

Decode the Mnemonic

What does this instruction mean in plain language?

03

Perform the Action

Execute the instruction physically—move data, do math, check conditions

04

Update the Program Step Indicator

Move to the next instruction (or jump to a different one)

Participant Activities

- **Predict outcomes** before executing each step
- **Physically write** values into registers and memory
- **Observe** how jump instructions change control flow

This is where the "ohhhh" moments happen.

Intentional Error Exercise (5 min)

Now we'll do something counterintuitive: we'll deliberately introduce a bug. This isn't about making mistakes—it's about understanding how computers detect and handle them.

1

Make a Small Arithmetic Mistake

Add two numbers incorrectly

2

Observe Detection

Watch how the program catches the error

3

Discuss Why

Talk about why error handling exists in all software

SECTION 5

Programming Fundamentals: From User to Creator

Now we shift gears. You've been running programs written by someone else. Time to think like a programmer yourself—reading, modifying, and ultimately creating code.

Part 1: Reading Mnemonics (5 min)

Using the Mnemonic List, we'll decode the instruction language. This is assembly language with the volume turned down.

- **LDR*, STR*** → Load and store data (movement)
- **ADD*, SUB*, MUL*** → Arithmetic operations
- **J*** → Jump commands (decisions and control flow)
- **PRO*** → Produce output (display results)

Part 2: Modify a Program (10 min)

Make intentional changes to existing code. Examples include:

- Change the output message to something personal
- Alter a constant stored in memory
- Swap which registers you use (A instead of B)
- Modify a jump condition to change program flow

Critical: Predict the result before running. That prediction is where learning happens.

Part 3: Create a Tiny New Program (5 min)

Write your own micro-program from scratch. Suggested starting points:

- Add two numbers and print the result
- Print a word stored in memory
- Count down from a number

No one needs to finish—the thinking is the win. Starting is succeeding.

CLOSING

Wrap-Up & Reflection: Connecting the Dots

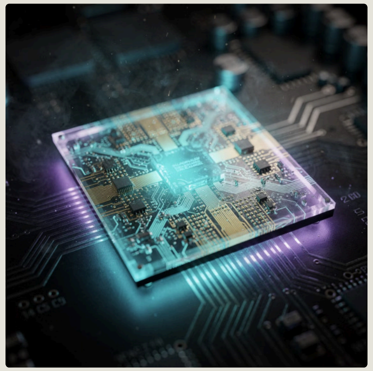
Group Reflection (5 min)

Let's surface what you learned—not just intellectually, but experientially. Consider these questions:

- **What surprised you?** What expectation did the workshop defy?
- **What became clearer?** What concept went from fuzzy to sharp?
- **What modern "black boxes" feel less mysterious now?** Where do you see this model in your daily tech use?

Bridge to Modern Computing

Everything you did today happens billions of times per second in the device you're reading this on. Let's make those connections explicit:



CPU Execution Cycles

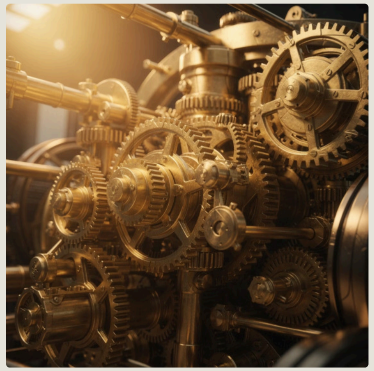
Fetch, decode, execute—exactly what you just did, but at gigahertz speeds

Debugging

When your code breaks, you now know *how* to trace through it step by step

Performance Costs

Every operation has a price—memory access, arithmetic, jumps. You felt that price today.



Software Complexity

Why "simple" software isn't simple at all—layers upon layers of these basic operations

> **"Every computer you use is doing *exactly this*—just very, very fast."**

Thank you for bringing your curiosity and patience to this workshop. You now understand something most people never see: **how computers actually work from the ground up.**