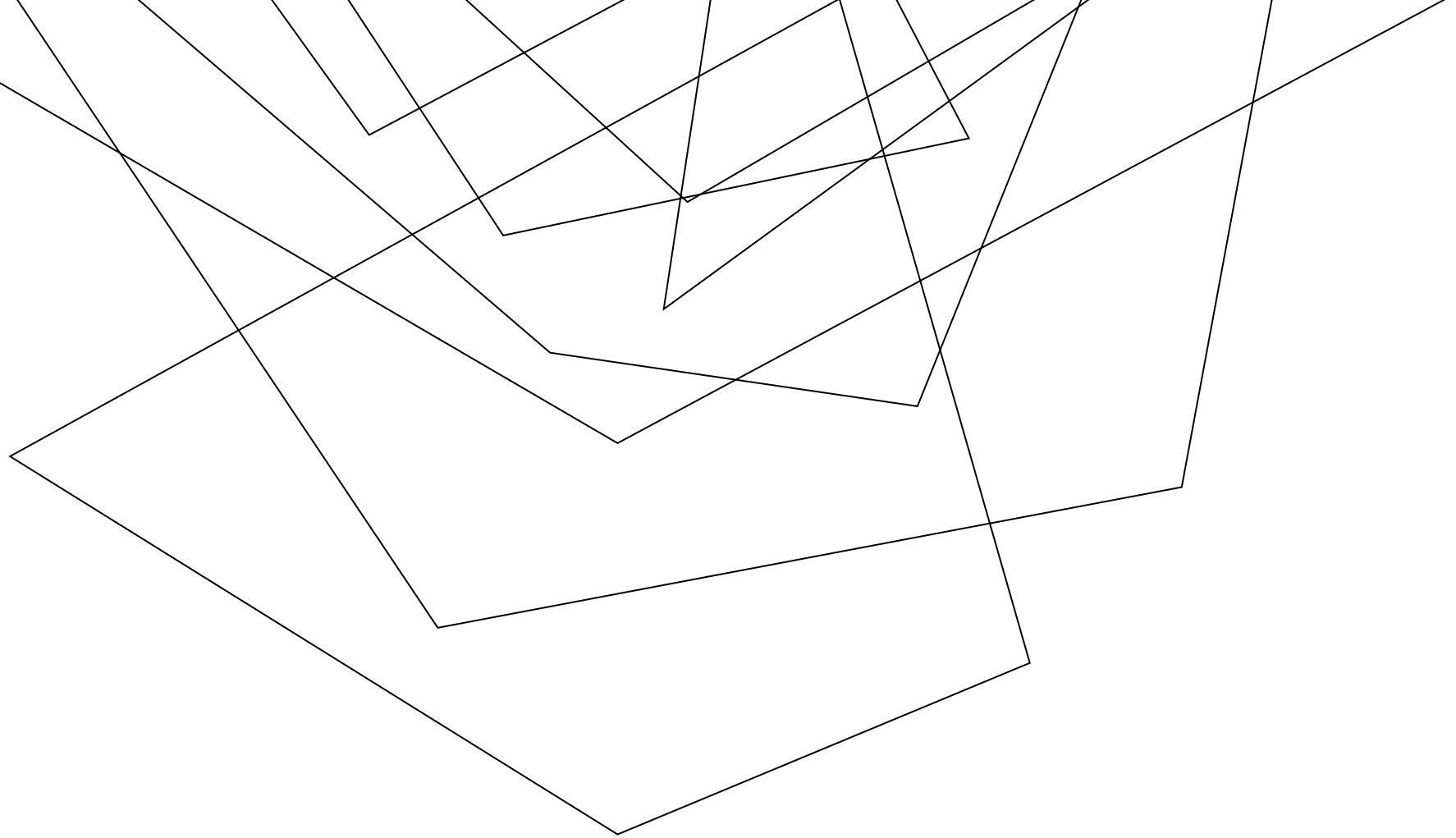
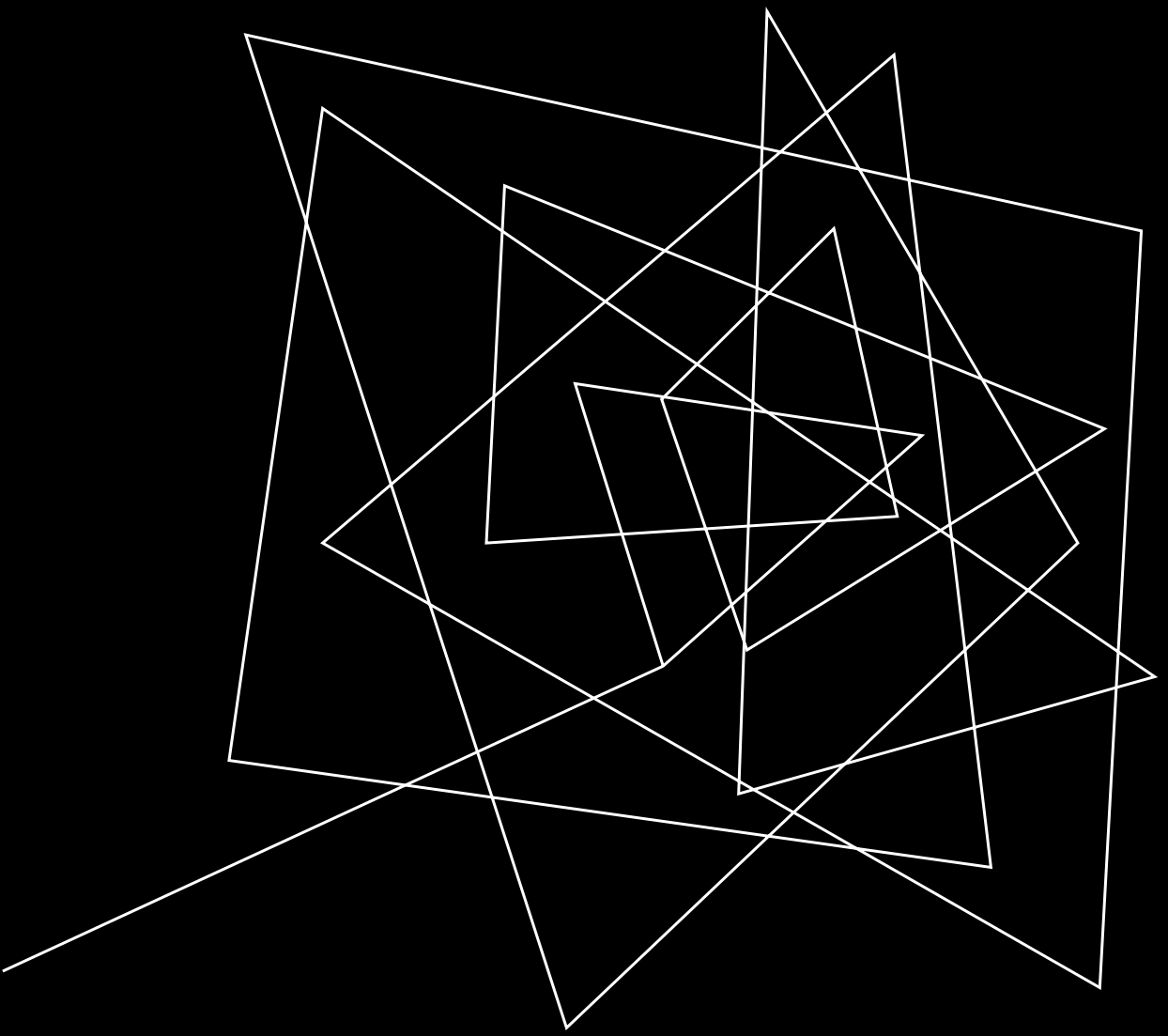


CMSC 240_{10} (11110000_{2}) (360_{8}) ($F0_{16}$) PRINCIPLES OF COMPUTER ORGANIZATION

Pre-requisites: CMSC B 151 or H106 or H107 and CMSC B231

LEARNING OUTCOMES

- Describe the major components of a modern computer (CPU, Memory, I/O) and how they are implemented and interact in hardware.
- Understand how programs and data are represented in hardware, how instructions are executed, and how data is stored in and retrieved from memory.
- Design circuits to implement Boolean functions and basic storage/memory constructs using digital logic.
- The von Neumann Model
- Relate the behavior of high-level languages like C or Java to the underlying low-level assembly language.



WHAT IS A COMPUTER?

A device with

- A processor (CPU – Central Processing Unit)
- Storage/Memory (RAM, Disk, USB Stick)
- A keyboard
- A mouse
- A Monitor
- A printer

THESE ARE ALL COMPUTERS

THIS IS ALSO A COMPUTER

El Capitan

World's Fastest Computer (2024)

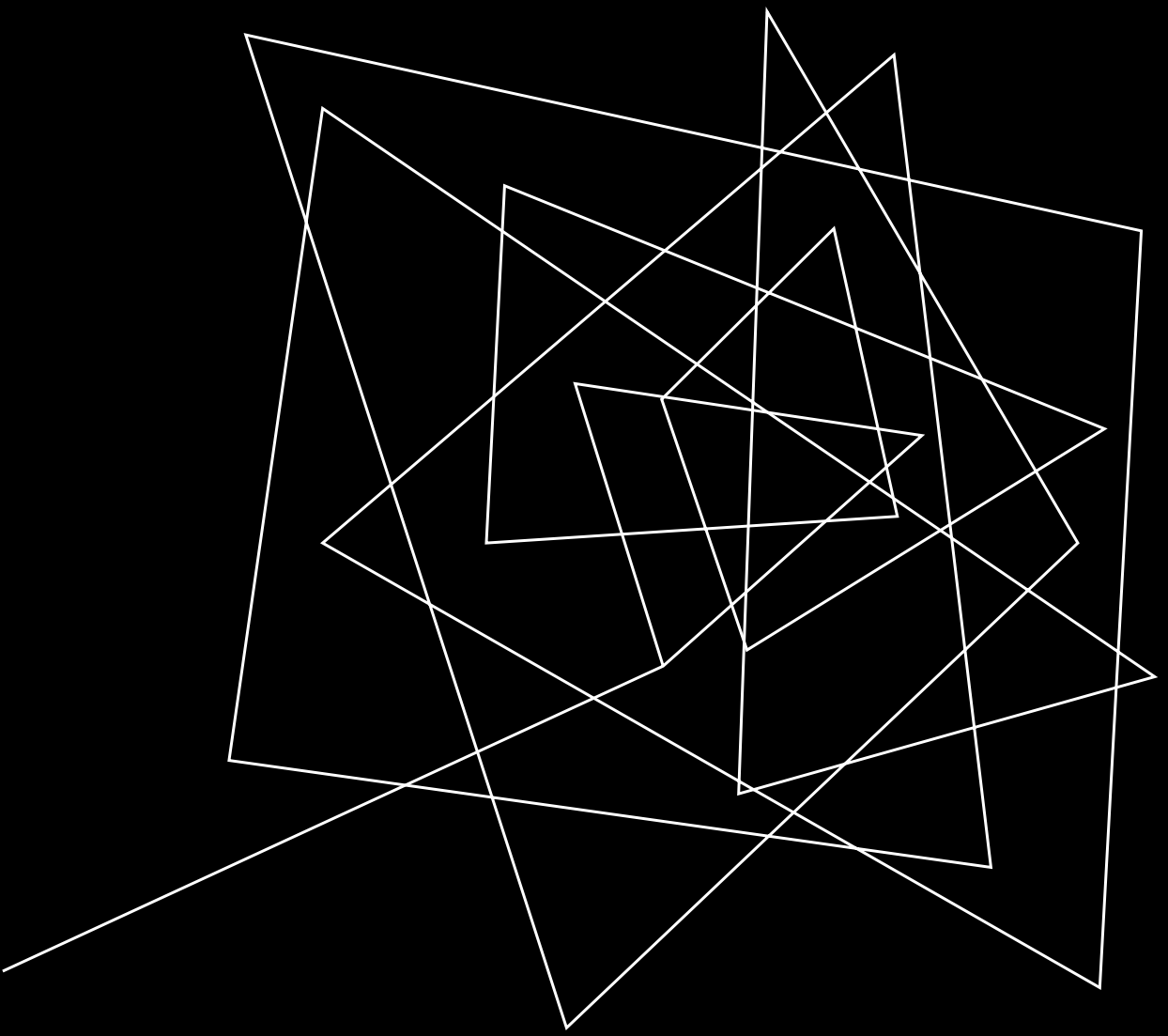
Lawrence Livermore National Labs, CA

Over 1 million Processors

Size of two tennis courts

Costs $600 million

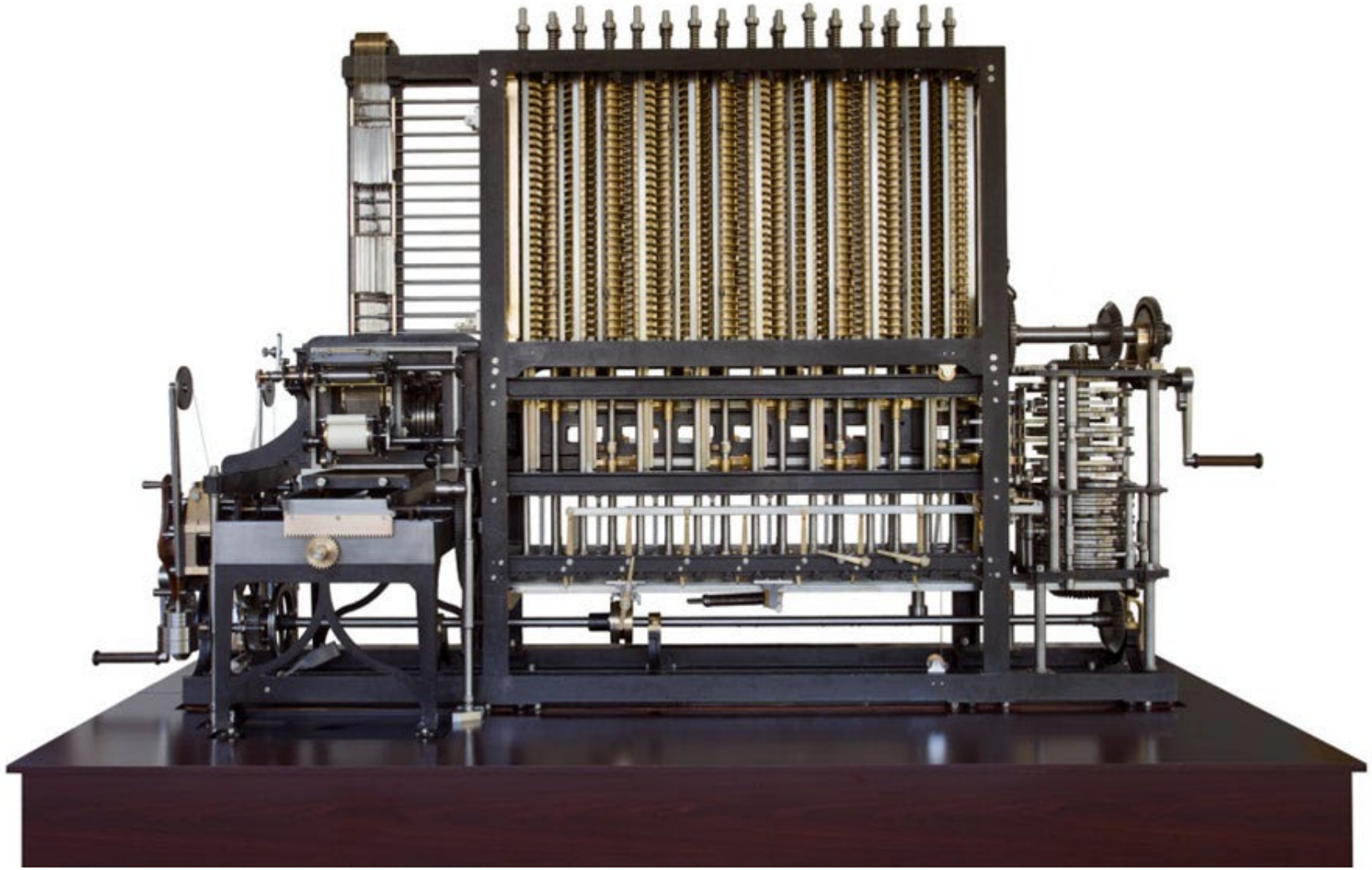
Can execute ~3 ExaFLOPS (10^{18} FLOPS)



WHAT WAS THE FIRST COMPUTER?

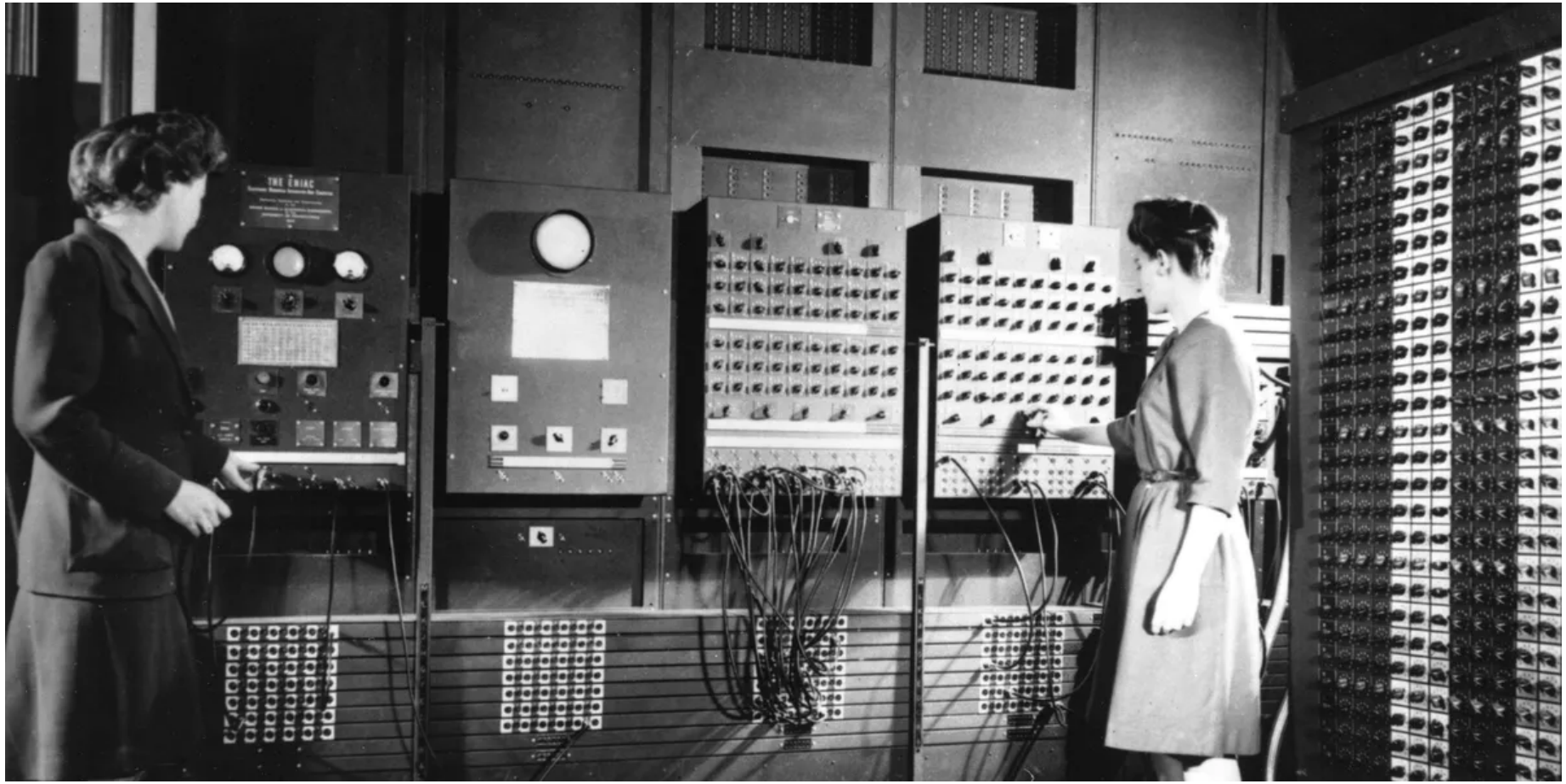
BABBAGE'S DIFFERENCE ENGINE#2 (1832, 2002)

See video at: https://youtu.be/XSkGY6LchJs?si=GzWuDh9yPtsW9DBL

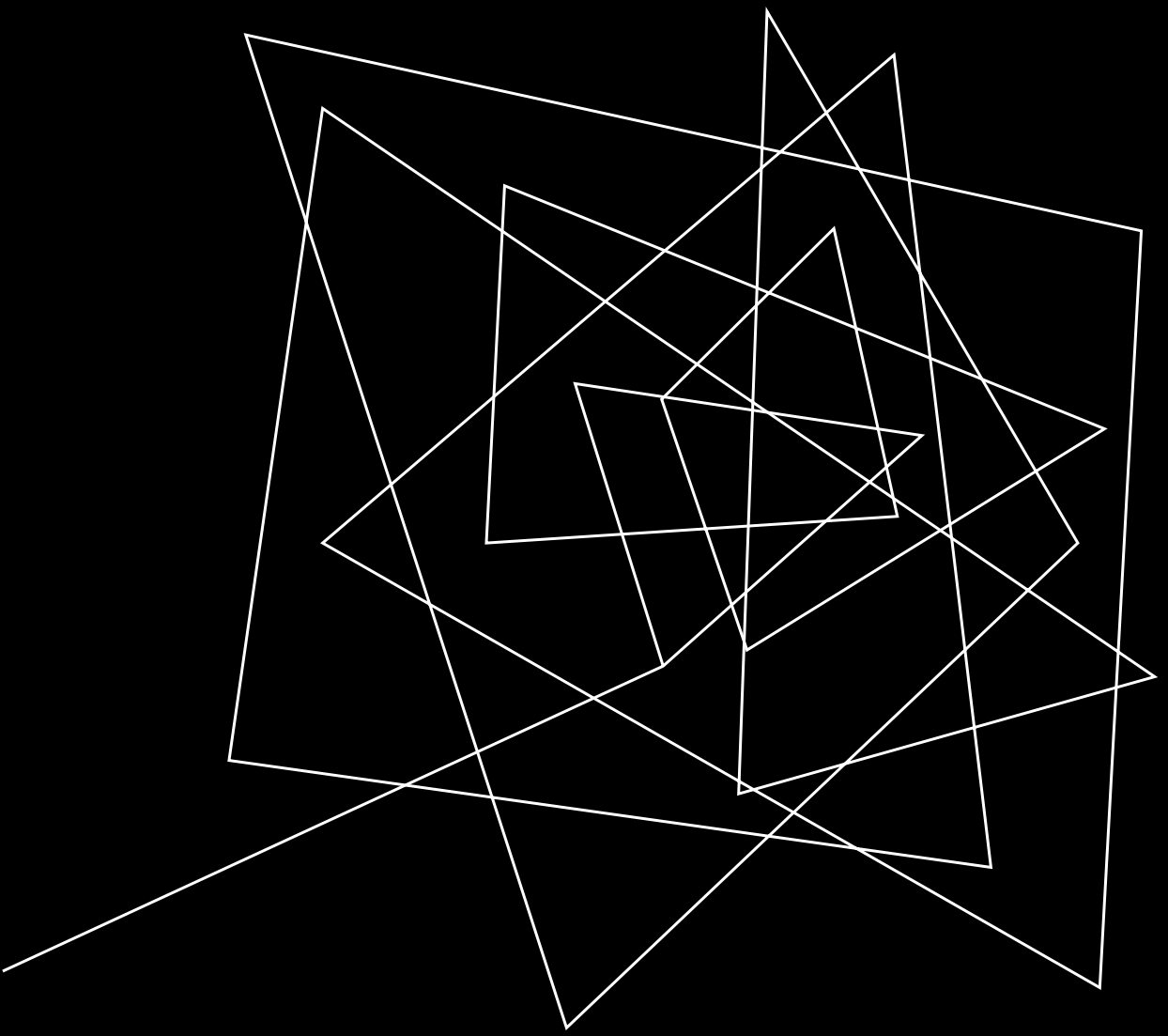


ENIAC FEBRUARY 16, 1946 (U. PENN)

See video at: https://youtu.be/XSkGY6LchJs?si=GzWuDh9yPtsW9DBL



See at: Moore School Building, Corner of 34th & Walnut, Philadelphia.

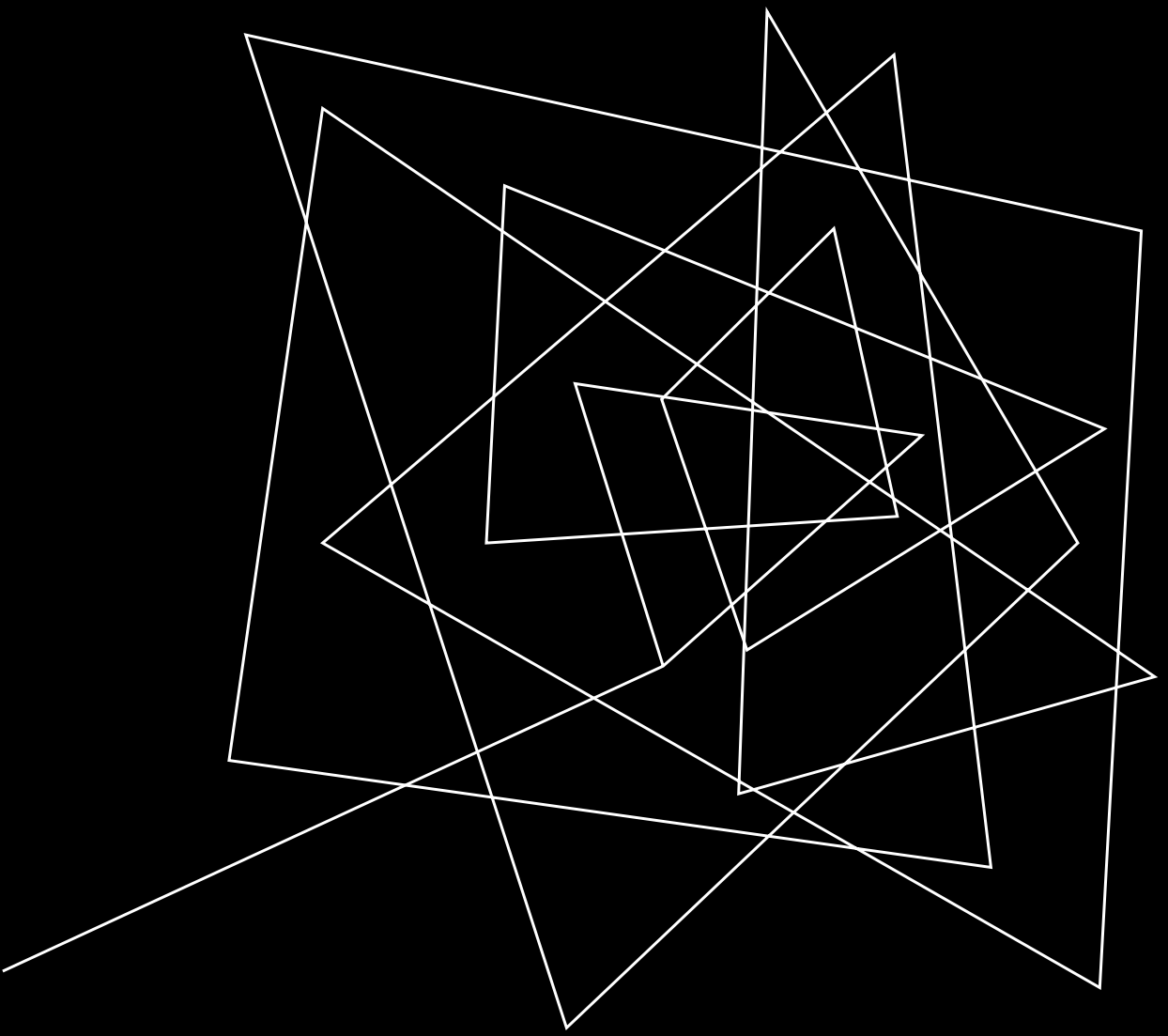


WHAT IS THE DIFFERENCE BETWEEN THE ENIAC, IPHONE, AND FRONTIER?

WHAT IS THE DIFFERENCE BETWEEN THE ENIAC, IPHONE, AND FRONTIER?

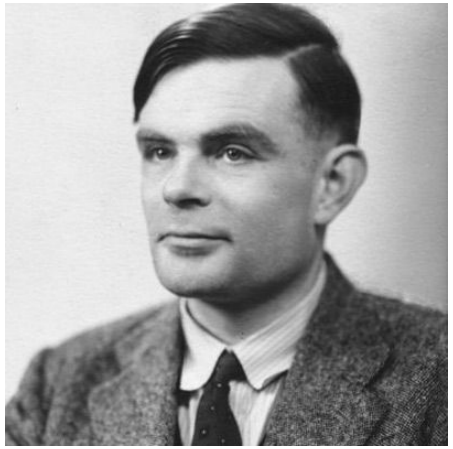
NOTHING!

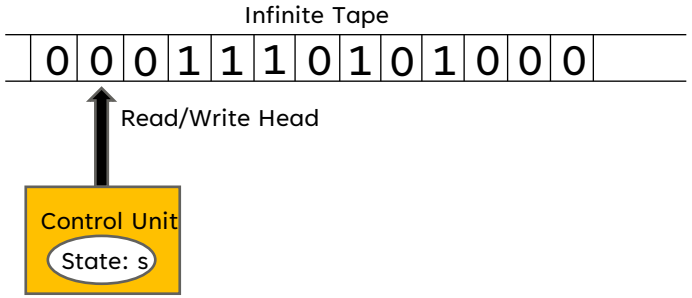
Besides the size, cost, the speed of the processor(s), and amount of memory.

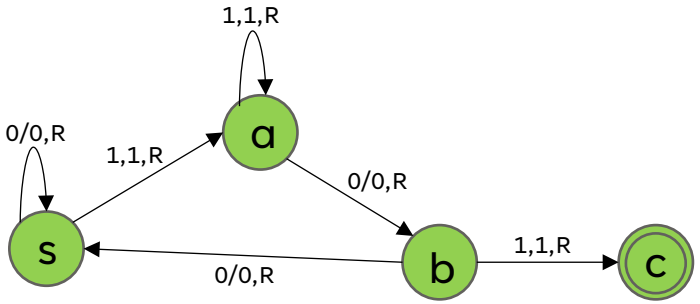


BUT REALLY, WHAT IS A COMPUTER?

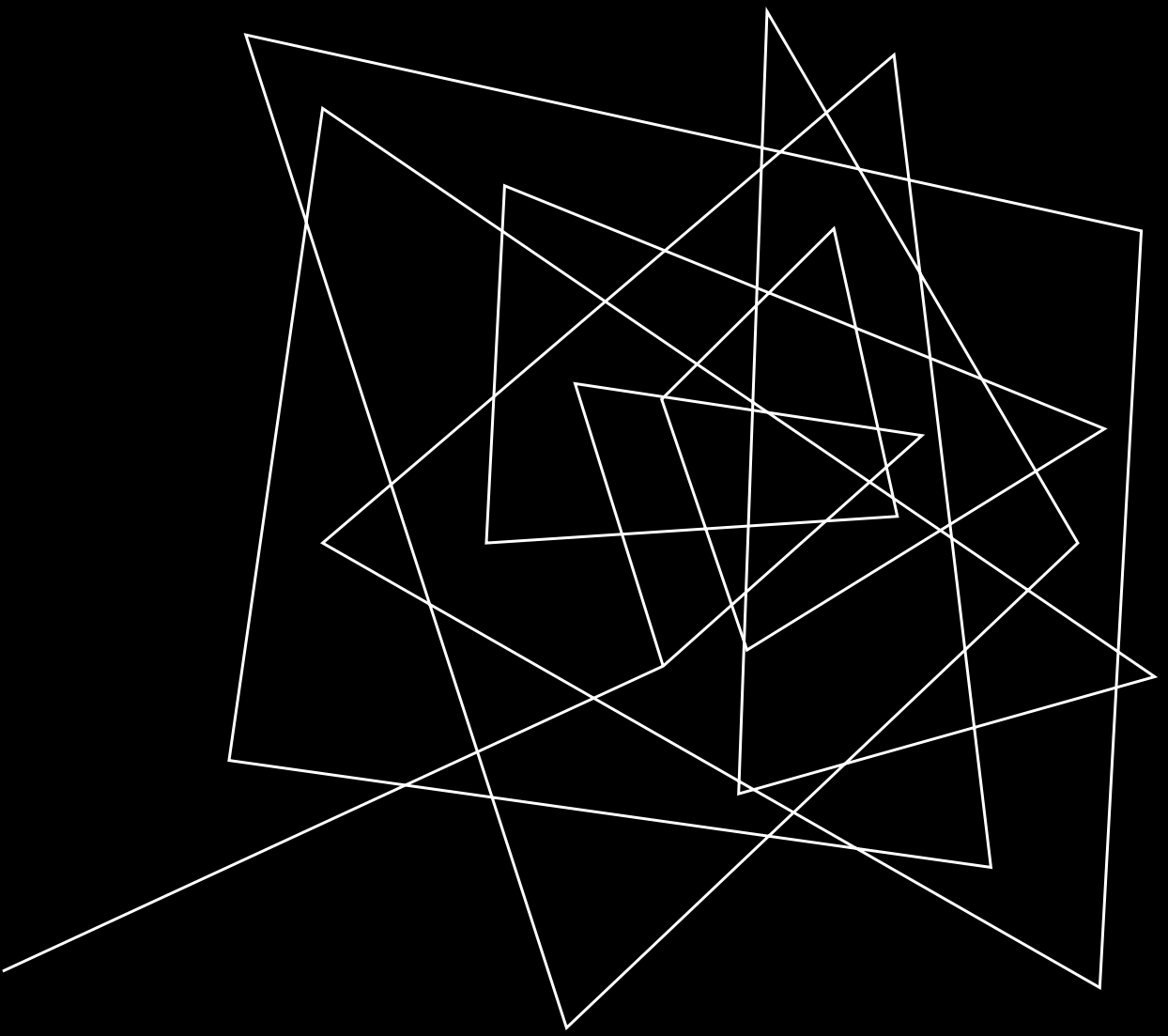
TURING MACHINE: AN IDEALIZED COMPUTER







	0	1
s	0,R, s	1,R, a
a	0,R, b	1,R, a
b	0,R s	1,R c



CHURCH-TURING THESIS* (1936)

EVERY COMPUTATION CAN BE PERFORMED BY SOME TURING MACHINE.

UNIVERSAL TURING MACHINE = PROGRAMMABLE COMPUTER

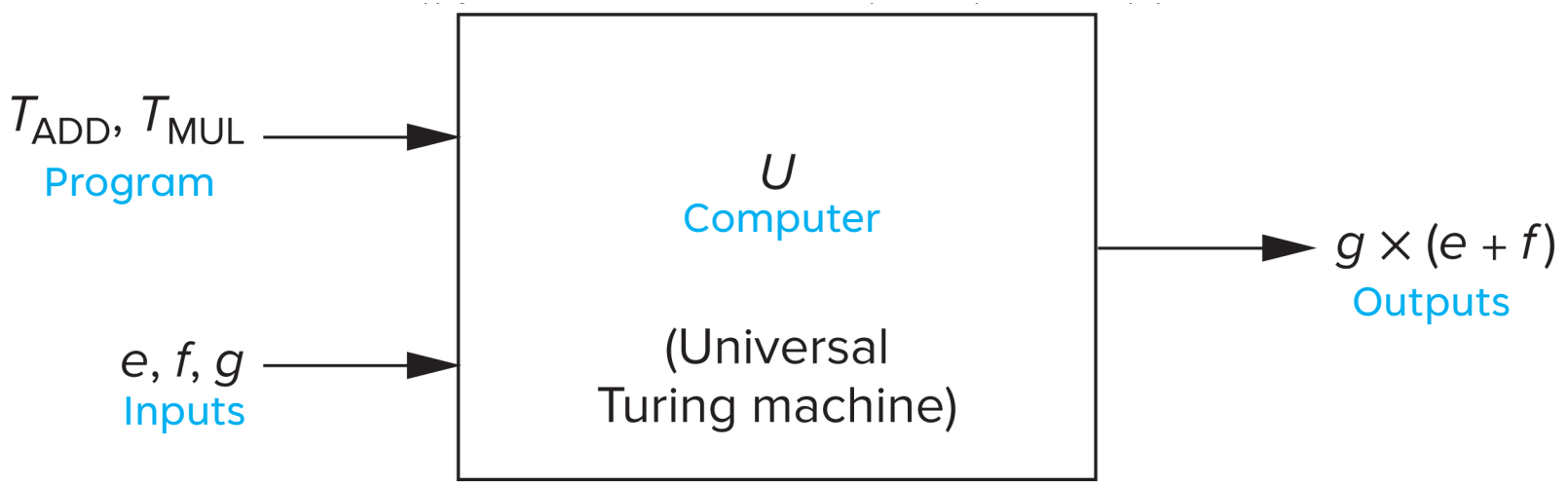
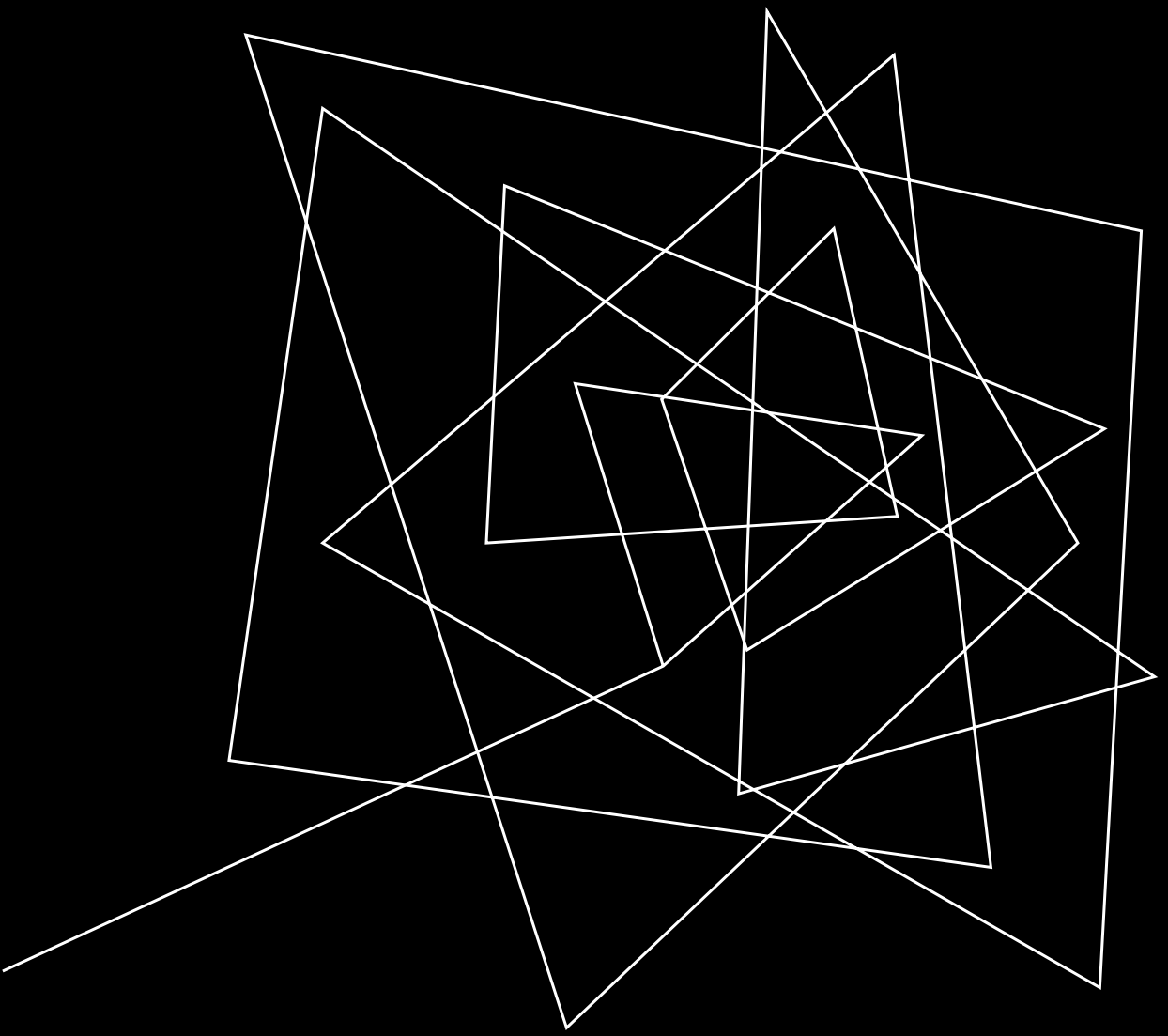
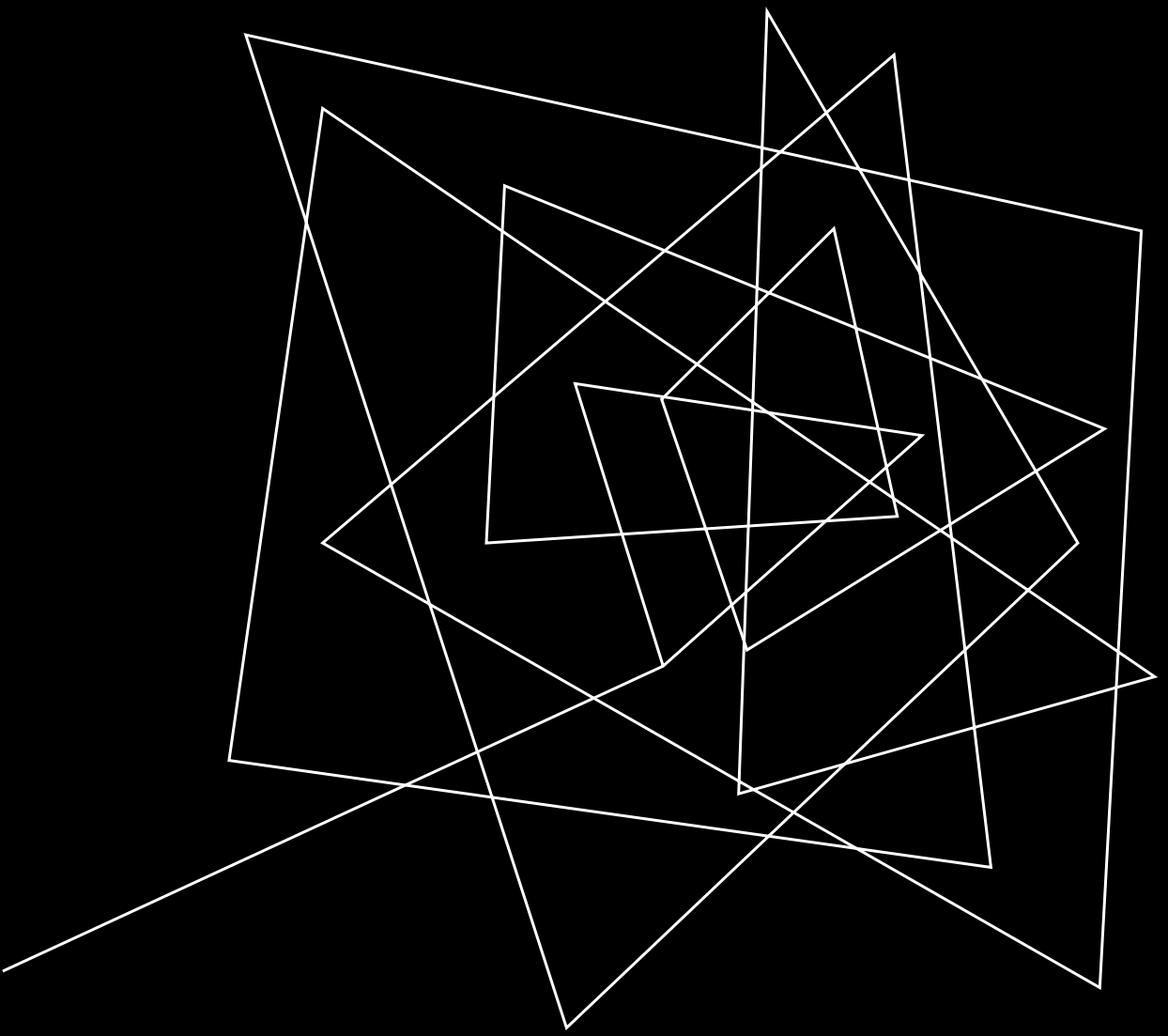


Figure 1.8 Black box model of a universal Turing Machine



KEY IDEA

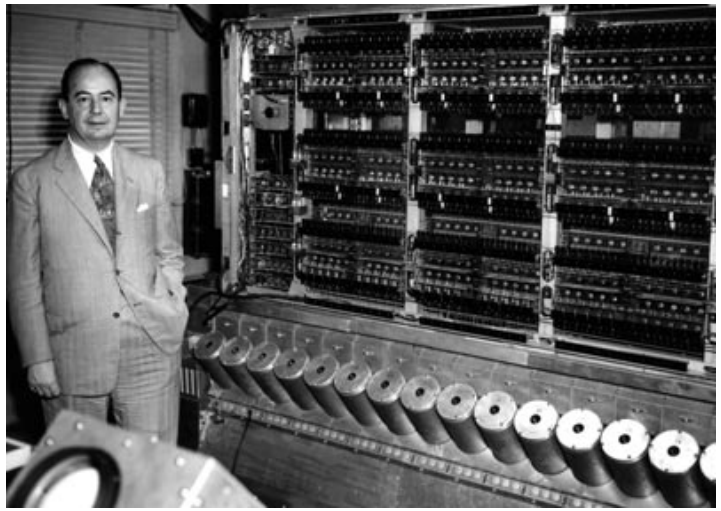
A COMPUTER IS ESSENTIALLY A UNIVERSAL TURING MACHINE.

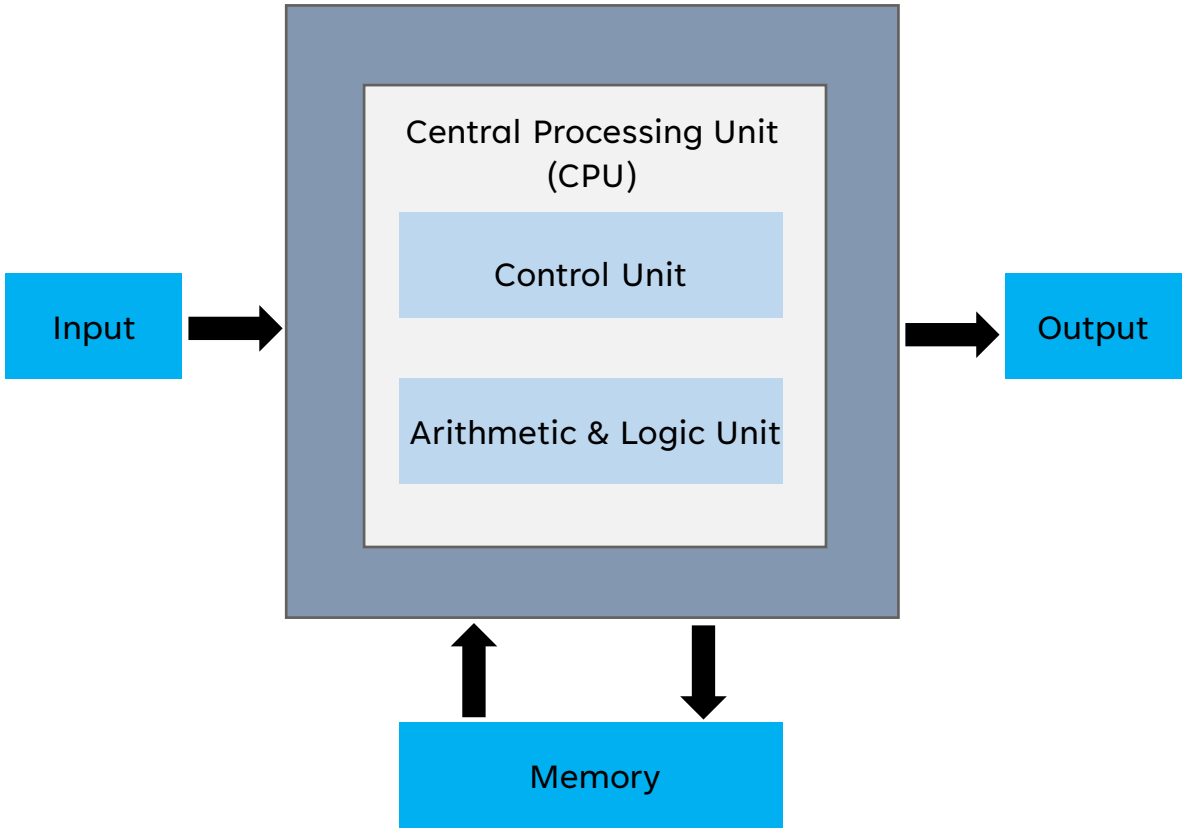


OK, SO...

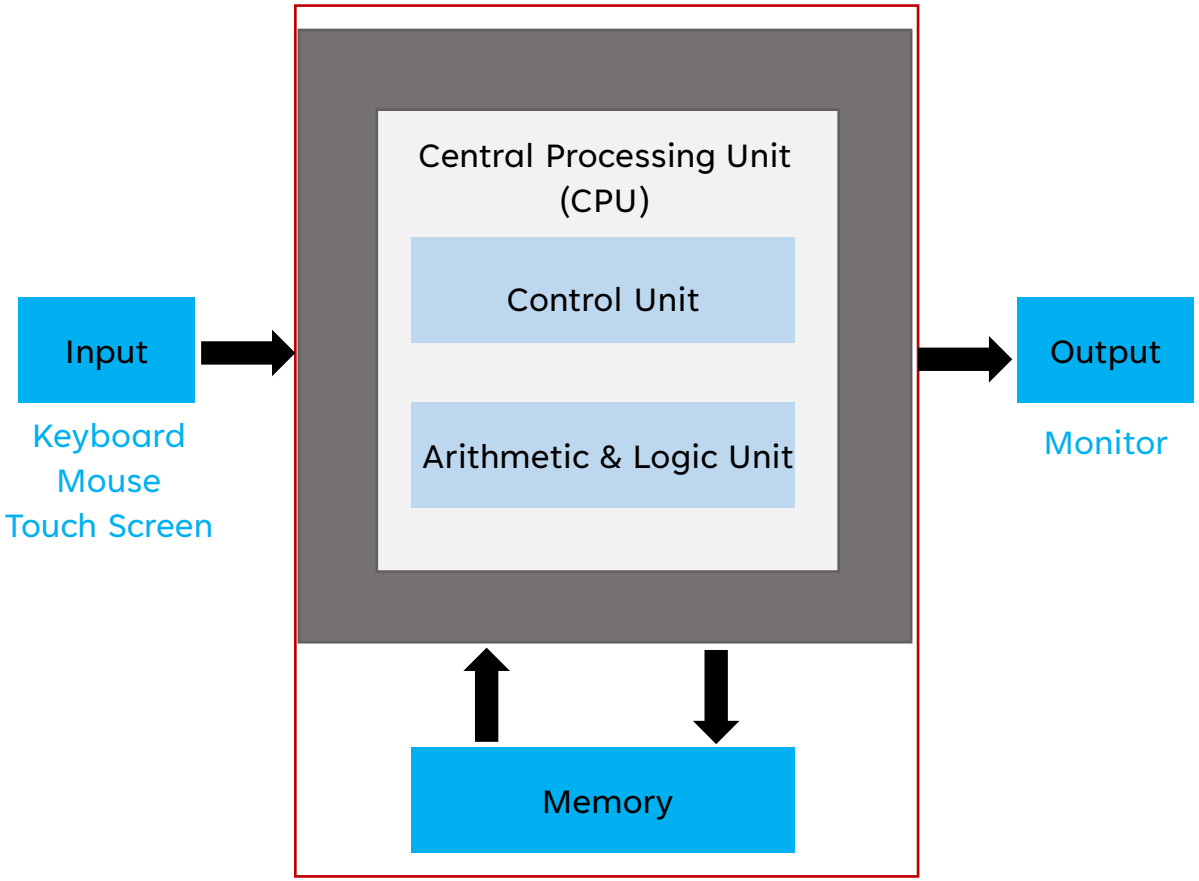
HOW DO WE ACTUALLY BUILD A UNIVERSAL TURING MACHINE (COMPUTER)???

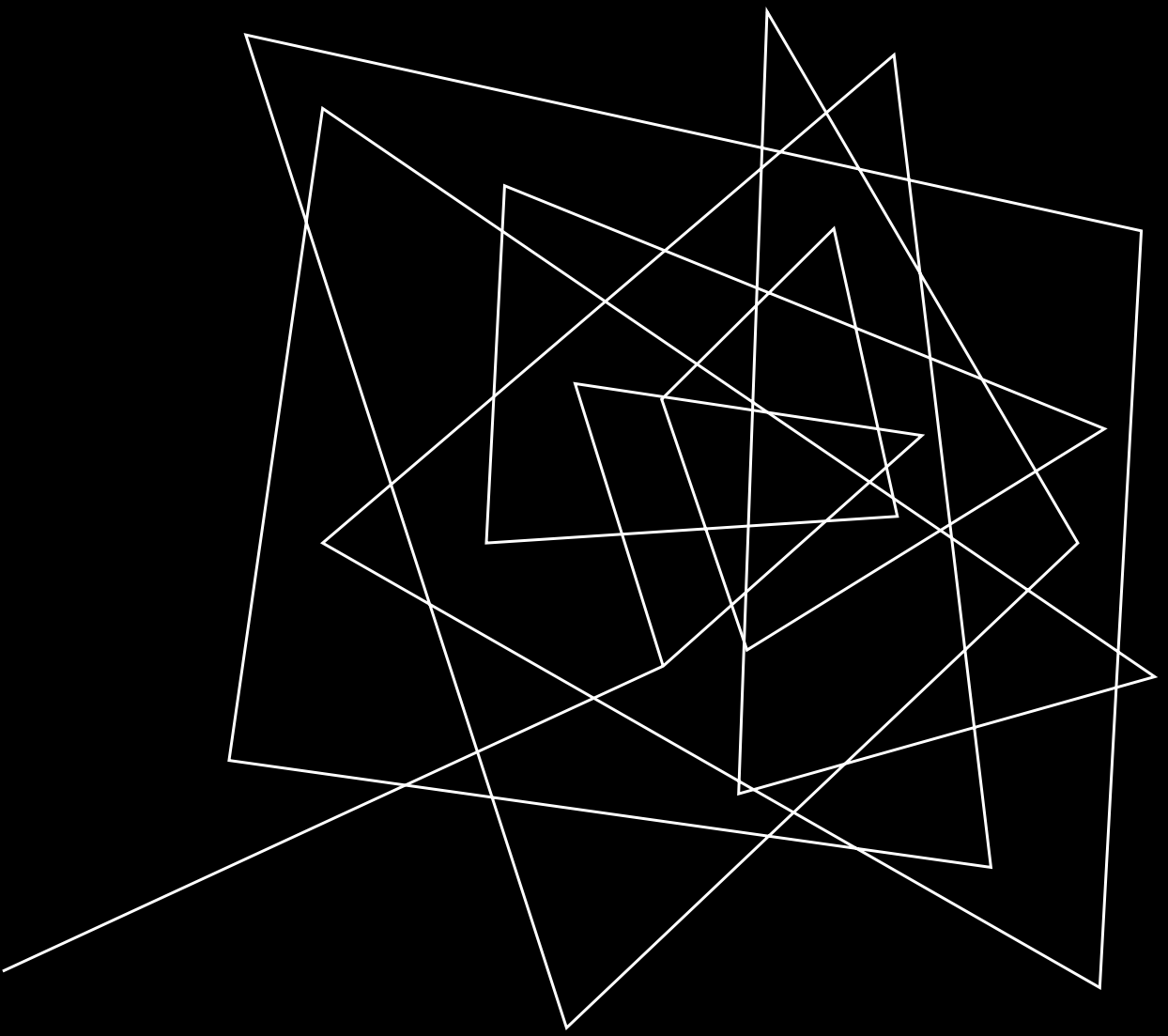
VON NEUMAN ARCHITECTURE (1945)





VON NEUMAN ARCHITECTURE





HOW DO WE GET THE COMPUTER TO WORK???

LEVELS OF TRANSFORMATION

Problems

Algorithms

Language

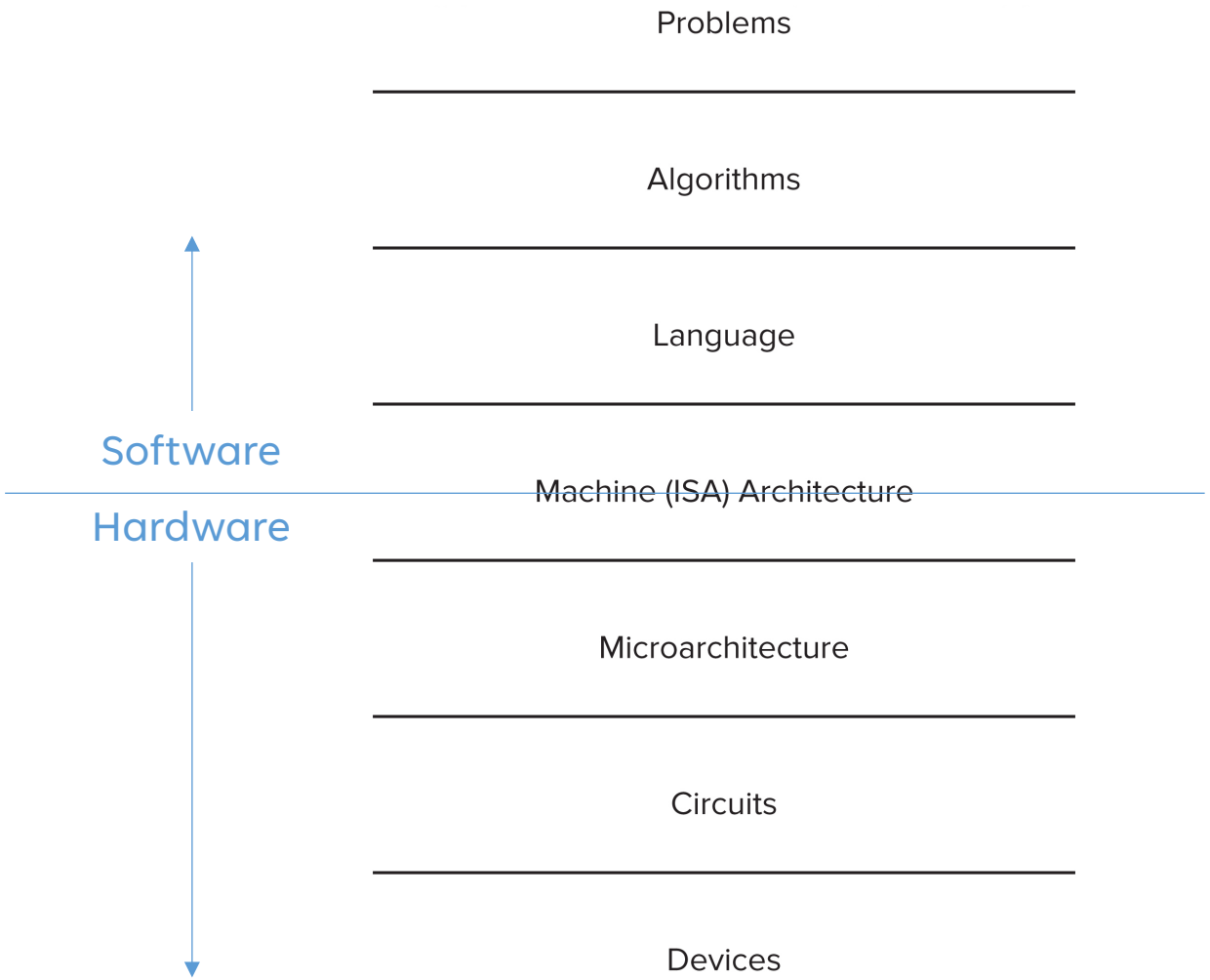
Machine (ISA) Architecture

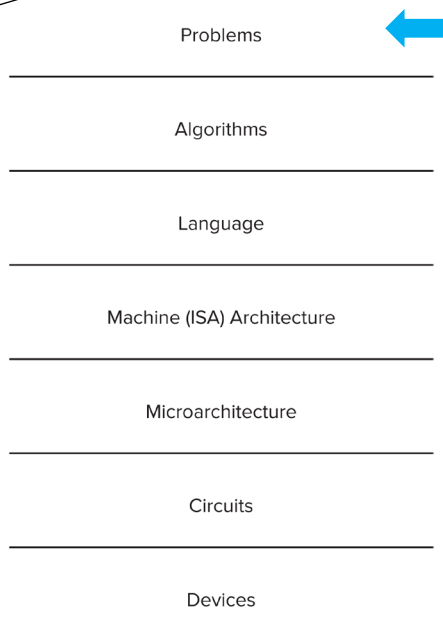
Microarchitecture

Circuits

Devices

LEVELS OF TRANSFORMATION





LEVELS OF TRANSFORMATION

PROBLEM

Compute the square root of a given number, a.

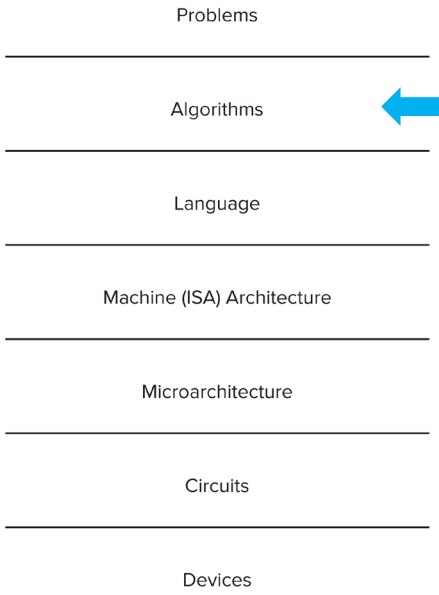
A SOLUTION

To compute the square root $x = \sqrt{a}$ do the following:

1. Start with some guess $x_1 > 0$
2. Compute a sequence of guesses x_1, x_2, ..., x_n using the equation

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$$

until the numbers produced converge.

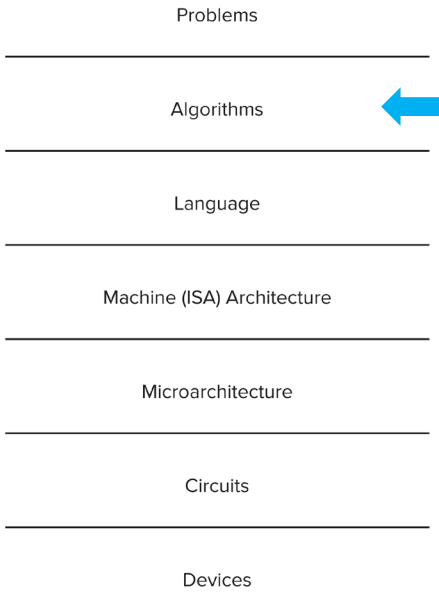


LEVELS OF TRANSFORMATION

ALGORITHM

1. To compute $\sqrt{a}$
2. Start with some guess $x_i = 1$. This is our initial guess.
3. Compute the next guess $x_{i+1} = \frac{1}{2}\left(x_i + \frac{a}{x_i}\right)$
4. If $x_{i+1} \neq x_i$
   Set x_i to be same as x_{i+1}
   And then repeat from Step 3.

   Otherwise, because $x_{i+1} = x_i$, they have converged.
   Therefore, $\sqrt{a} = x_{i+1}$



LEVELS OF TRANSFORMATION

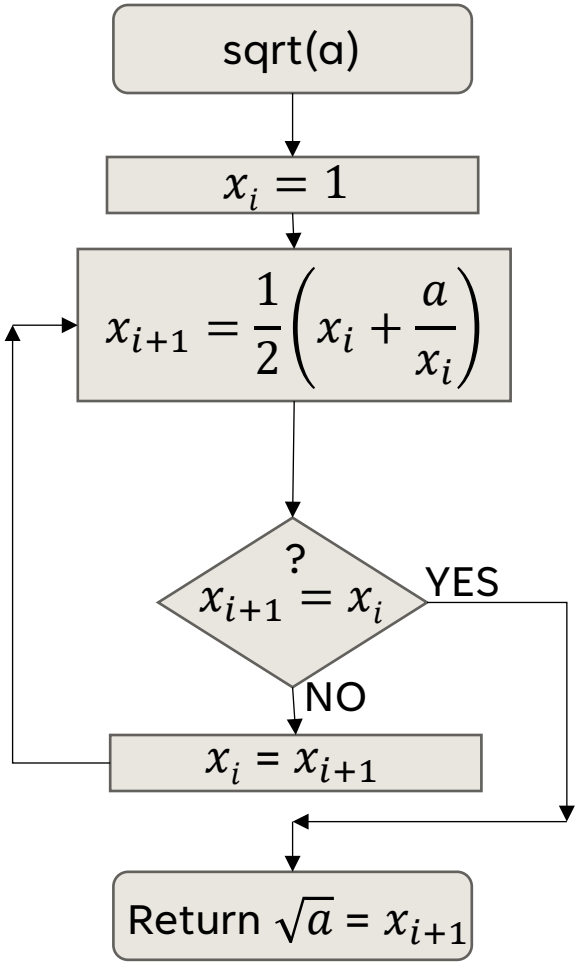
ALGORITHM

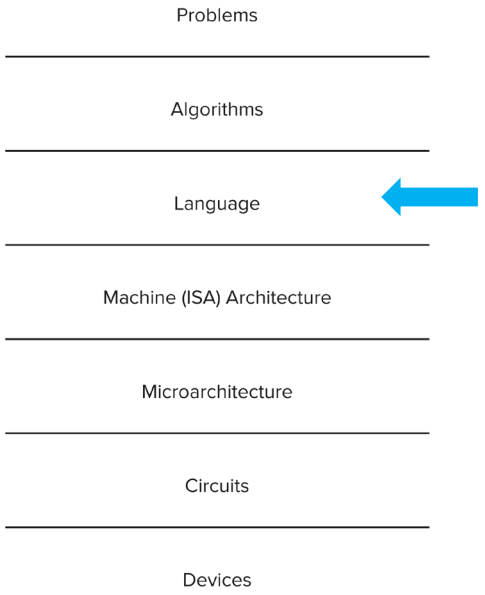
1. To compute $\sqrt{a}$
2. Start with some guess $x_i = 1$. This is our initial guess.
3. Compute the next guess $x_{i+1} = \frac{1}{2}\left(x_i + \frac{a}{x_i}\right)$
4. If $x_{i+1} \neq x_i$
   Set x_i to be same as x_{i+1}
   And then repeat from Step 3.

   Otherwise because $x_{i+1} = x_i$, they have converged.
   Therefore, $\sqrt{a} = x_{i+1}$

Definition:
An algorithm is a *precise*, *unambiguous*, and *effective* procedure.

FLOWCHART


sqrt(a)
$x_i = 1$
$x_{i+1} = \frac{1}{2}\left(x_i + \frac{a}{x_i}\right)$
? $x_{i+1} = x_i$
YES
NO
$x_i = x_{i+1}$
Return $\sqrt{a} = x_{i+1}$



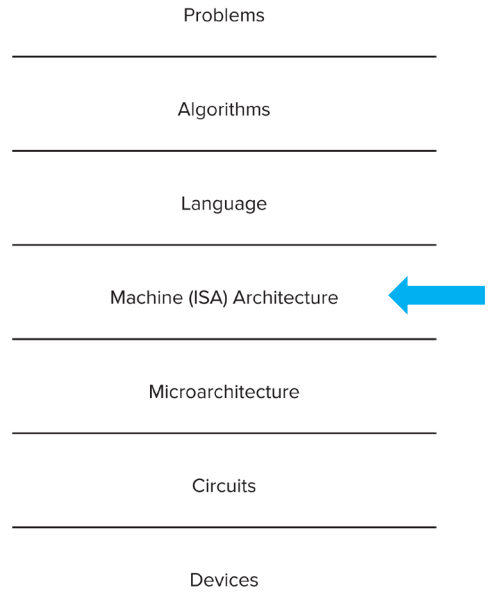
LEVELS OF TRANSFORMATION

PROGRAM

```
double sqrt (double a) {
   if (a <= 0) return 0;

   double x0 = 1;
   double x1 = (x0 + a/x0)/2.0;

   while (x0 != x1) {
      x0 = x1;
      x1 = (x1 + (a/x1))/2.0;
   }
   return x1;
} // sqrt()
```



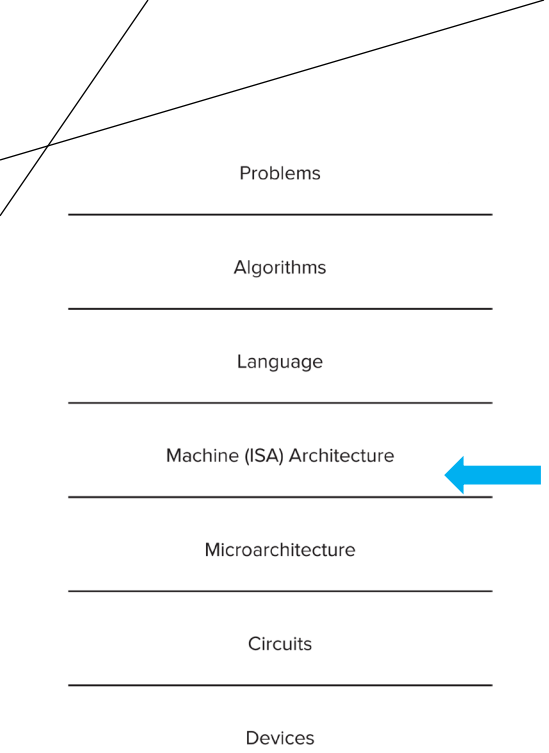
LEVELS OF TRANSFORMATION

ASSEMBLY PROGRAM (ISA)

Instruction Set Architecture (e.g. x86, ARM, LC3, etc.)

opcode operands

```
        .globl sqrt
sqrt:
.LFB0:
        .cfi_startproc
        pushq %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq %rsp, %rbp
        .cfi_def_cfa_register 6
        movsd %xmm0, -24(%rbp)
        pxor %xmm0, %xmm0
        comisd -24(%rbp), %xmm0
        jb .L8
        pxor %xmm0, %xmm0
        jmp .L4
.L8:
        movsd .LC1(%rip), %xmm0
        movsd %xmm0, -16(%rbp)
        movsd -24(%rbp), %xmm0
        divsd -16(%rbp), %xmm0
        addsd -16(%rbp), %xmm0
        movsd .LC2(%rip), %xmm1
        divsd %xmm1, %xmm0
        movsd %xmm0, -8(%rbp)
        jmp .L5
.L6:
        movsd -8(%rbp), %xmm0
        movsd %xmm0, -16(%rbp)
        movsd -24(%rbp), %xmm0
        divsd -8(%rbp), %xmm0
        addsd -8(%rbp), %xmm0
        movsd .LC2(%rip), %xmm1
        divsd %xmm1, %xmm0
        movsd %xmm0, -8(%rbp)
.L5:
        movsd -16(%rbp), %xmm0
        jp .L6
        movsd -16(%rbp), %xmm0
        ucomisd -8(%rbp), %xmm0
        jne .L6
        movsd -8(%rbp), %xmm0
.L4:
        popq %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
        .LFE0:
        .size sqrt, .-sqrt
        .section .rodata
        .align 8
.LC1:
        .long 0
        .long 1072693248
        .align 8
.LC2:
        .long 0
        .long 1073741824
```



LEVELS OF TRANSFORMATION

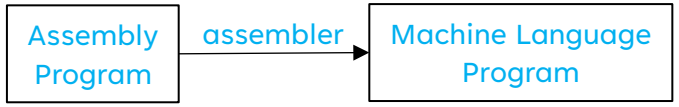
MACHINE LANGUAGE PROGRAM

Binary representation of assembly program

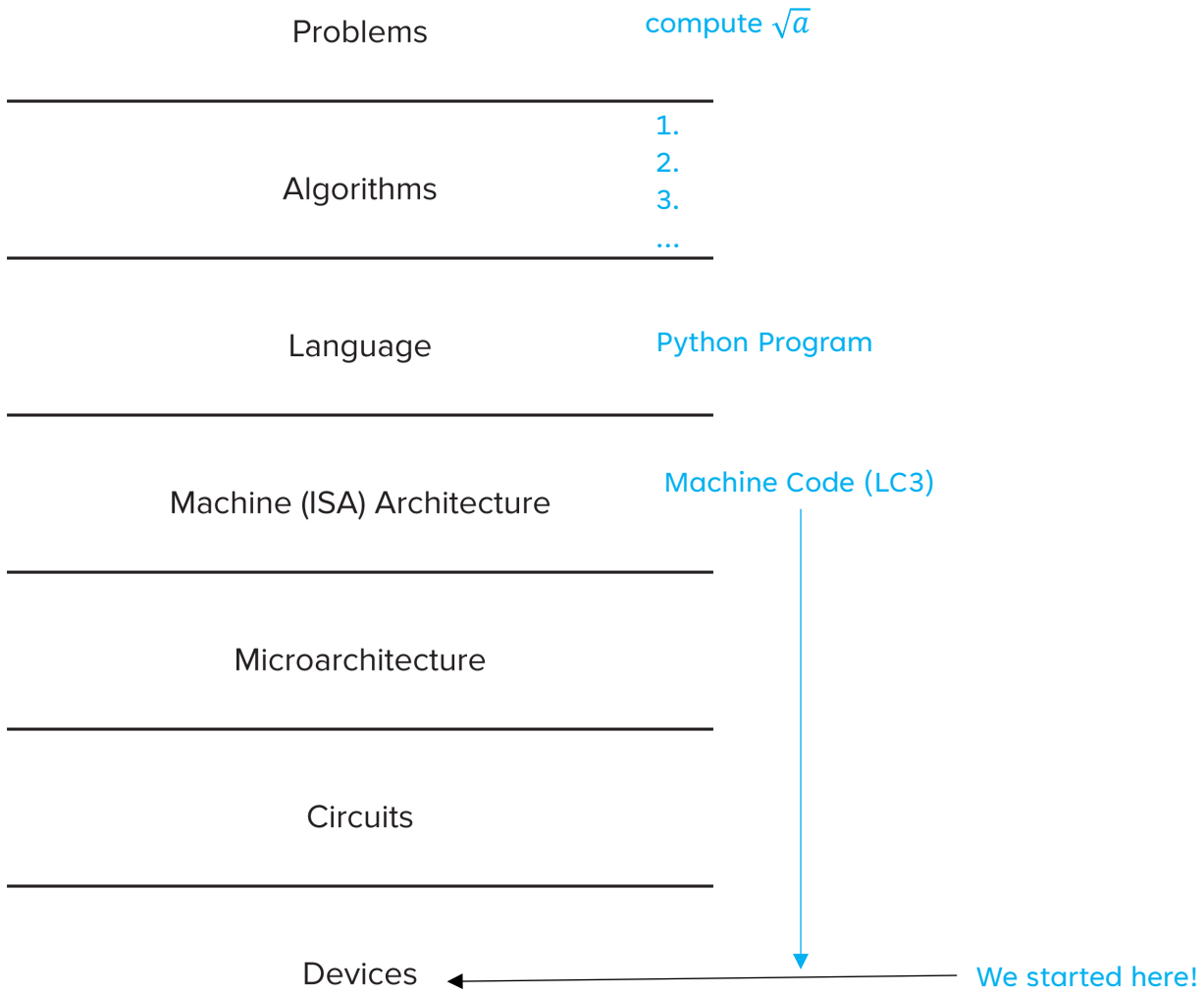
```
0101000000100000
0001000000100001
0101001001100000
0001001001111011
0101011011100000
0001011011101010
0010100000001001
0110010100000000
0001010010000001
0000010000000101
0001100100100001
0001011011111111
0110010100000000
0000001111111010
0101000000100000
1111000000100101
0011000100000000
```

What the CPU does:

```
do forever
   fetch the next instruction
   decode it
   carry it out
```



LEVELS OF TRANSFORMATION



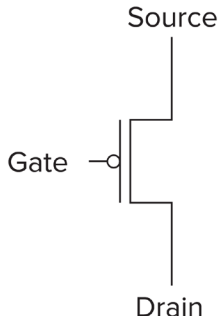
THE BINARY WORLD OF COMPUTERS

- Computers use standardized representations for basic data:

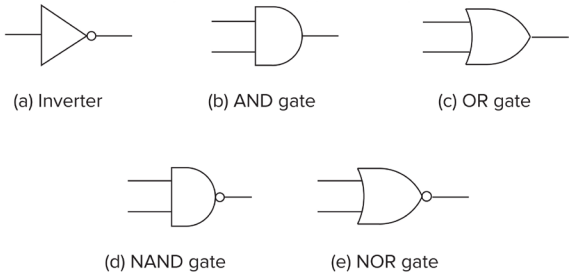
  integers: 2's complement representation
  arithmetic operations on these

  floating point: IEEE Floating Point representation

  characters: ASCII, Unicode, UTF-8, etc.



DIGITAL CIRCUITS

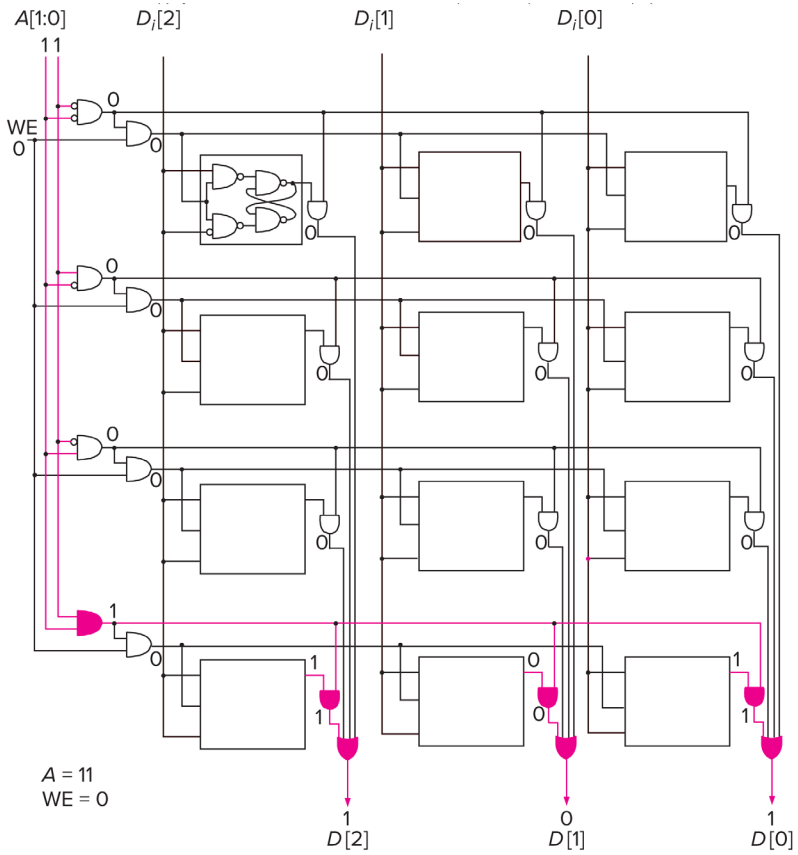


- P and N-type **transistors**
- Transistors are put together to build **logic gates** (**AND**, **OR**, **NOT**)
- AND, OR, NOT are the basic (and complete) set of logical operations implemented in a CPU using logic gates.
- **Adders**, **Decoders**, **Multiplexers**, and **Programmable Logic Arrays** make up the functional units in a CPU.
- Basic circuits in a CPU are **combinational logic circuits** and **sequential logic circuits**.

MEMORY

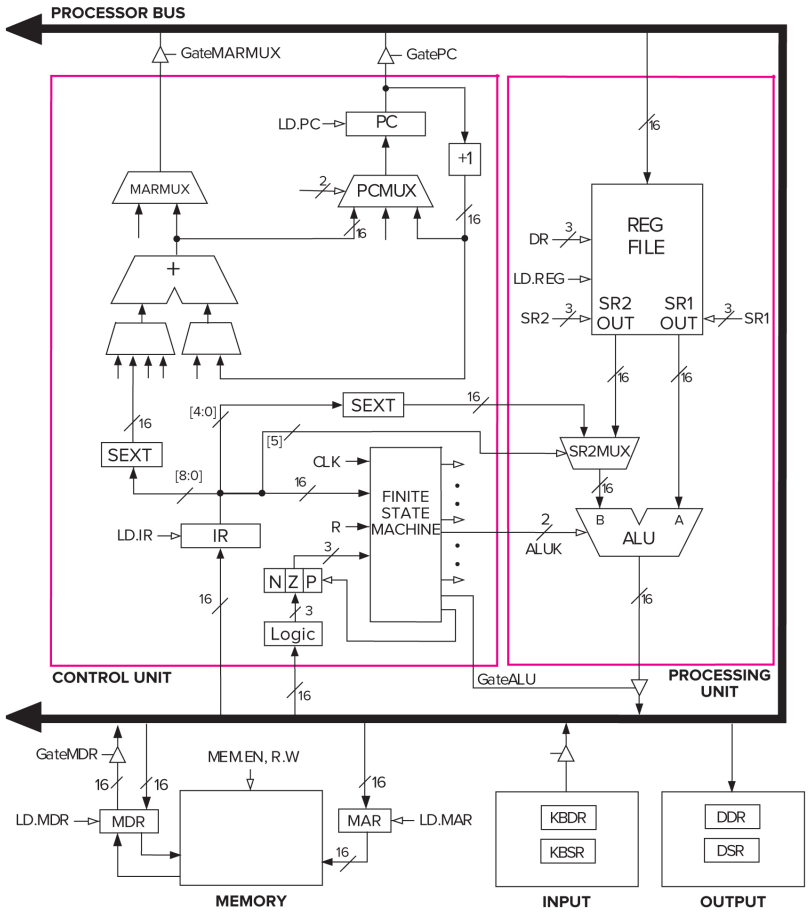
- Designed using **synchronous flip-flops** (**gated latches**), **decoders**, and **multiplexers**. **Addressability** and **address space** are the basic characteristics of any memory.

- Other attributes: error correction, refresh rate, and speed of retrieval.



CPU

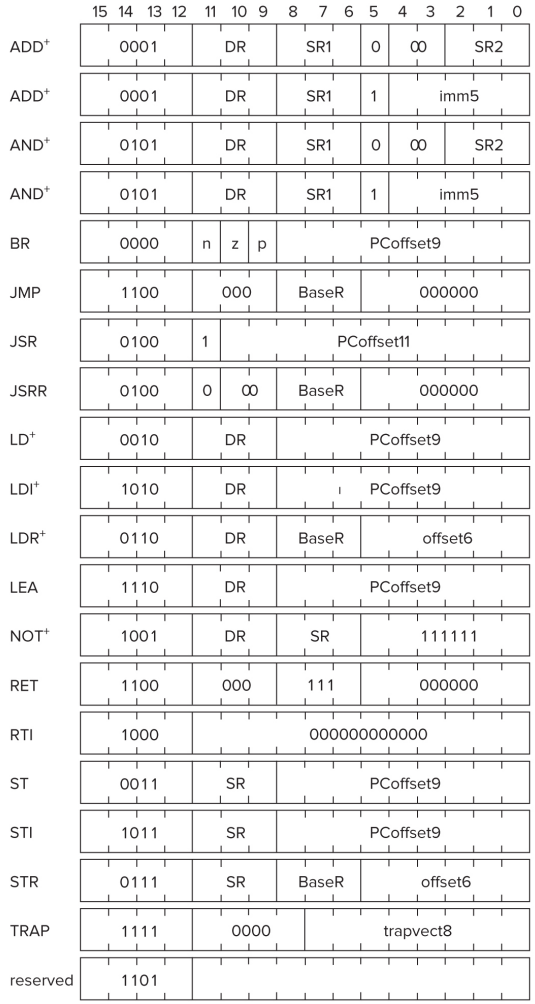
- A CPU is a **synchronous finite state machine**.
- It has a **control unit**, an **arithmetic and logic unit** (**ALU**), and several **registers** (including PC, IR, PSR, SSP, USP, etc).
- Synchronization is done by a **clock**.



INSTRUCTION SET ARCHITECTURE

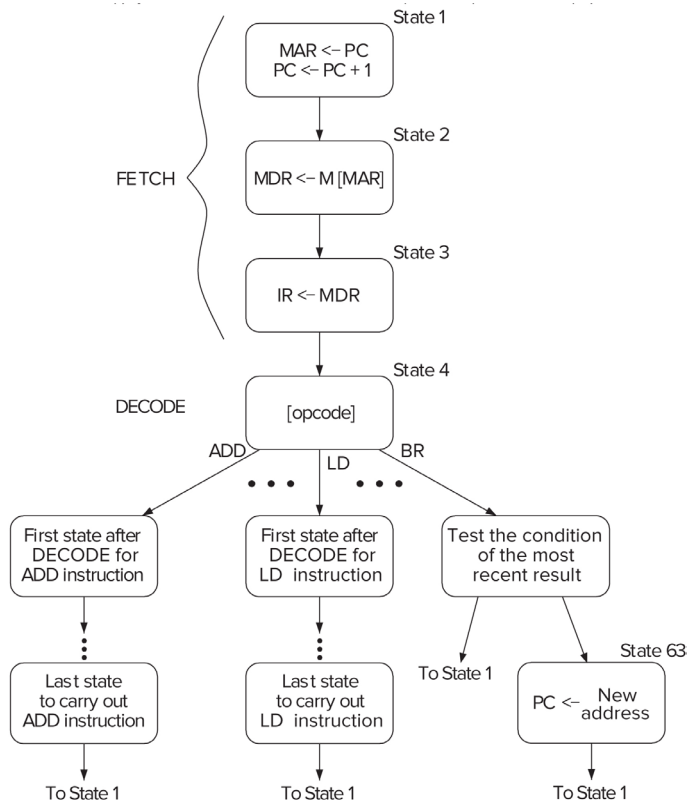
- Designers of the CPU specify an **instruction set architecture** that includes instructions for doing basic **arithmetic**, **loading**, and **storing** data, **branching**, **subroutine call** and **return**, and **TRAP** instructions.

- The instruction has an **opcode**, **operands**, and **addressing modes**.



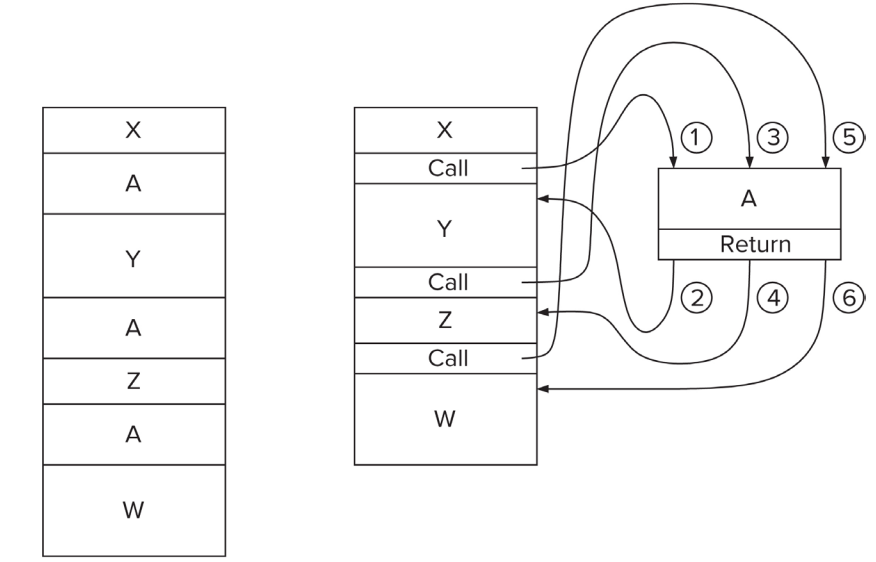
INSTRUCTIONS & INSTRUCTION CYCLE

- The **Instruction Cycle** of a CPU defines the basic tasks: **fetch**, **decode**, **evaluate address**, **fetch operands**, **execute**, **store result** that needs to be carried out in each instruction.
- **Machine language programming** involves writing programs using the ISA instructions in binary.
- An **assembler** makes the machine-level programming more palatable by defining symbols for **opcodes**, **registers**, and **labels**. An assembler also provides **Assembler Directives** that help organize and define a machine language program. **The assembler translates an assembly program into an equivalent machine language program**.



SUBROUTINES & TRAPS

- **Subroutines** enable organizing a program into functional pieces.
- TRAP instructions can be used to write **I/O Service Routines**. Simple **I/O devices can be mapped to memory addresses**.
- I/O service routines can use a **polling** or an **interrupt driven** model for doing I/O.

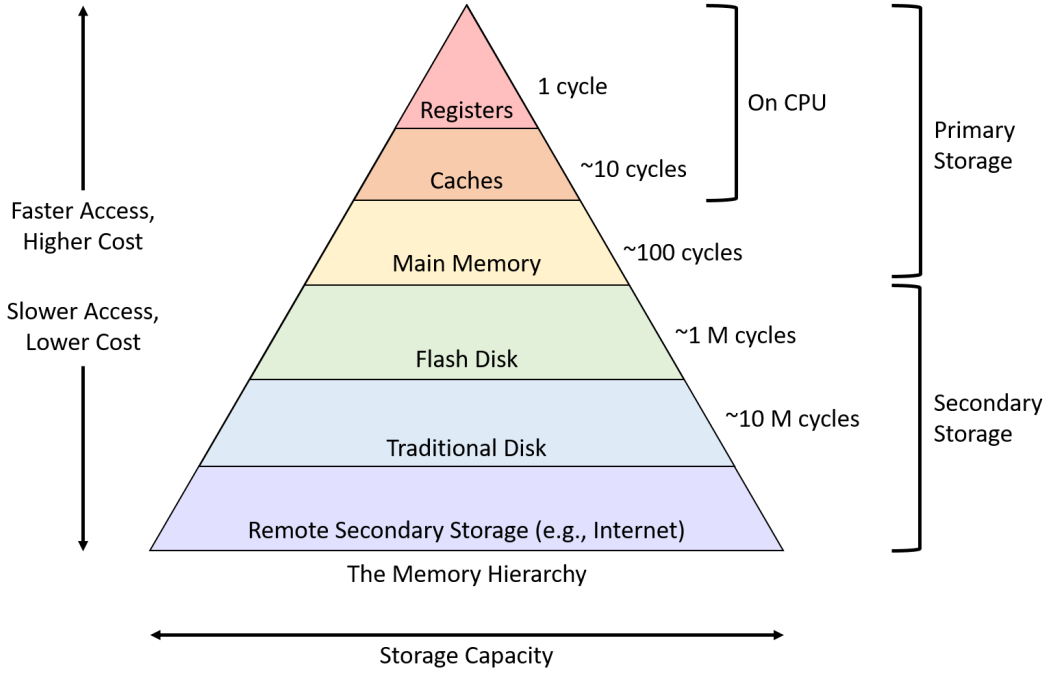


(a) Without subroutines

(b) With subroutines

CPU PROCESSING POWER

- **Processing power** of a basic CPU can be defined in terms of number of **instructions/second**.
- **FLOPS** represent a more practical measure of CPU power.
- Most CPUs have **multiple cores** running one or more **threads** of computation in each core.
- Most computers utilize a **memory hierarchy** to speed up processing and memory access using different levels of **cache memory**.
- Most computers also have **co-processors** for doing: floating-point arithmetic, graphics processing, etc.



From: https://diveintosystems.org/book/C11-MemHierarchy/mem_hierarchy.html

TOPICS

- What is a Computer?
- Instruction Set Architecture (ISA)
- Bits, Data Types, and Operations
- The von Neumann Model
- The LC3 ISA
- Programming in Assembly using LC3 ISA
- Subroutines and the Stack
- I/O Operations: Service Routines, Traps and Interrupts
- Memory Hierarchy & Caching
- Non-von Neumann Architectures
- Multi-core processors
- Co-processors & GPUs

THANK YOU