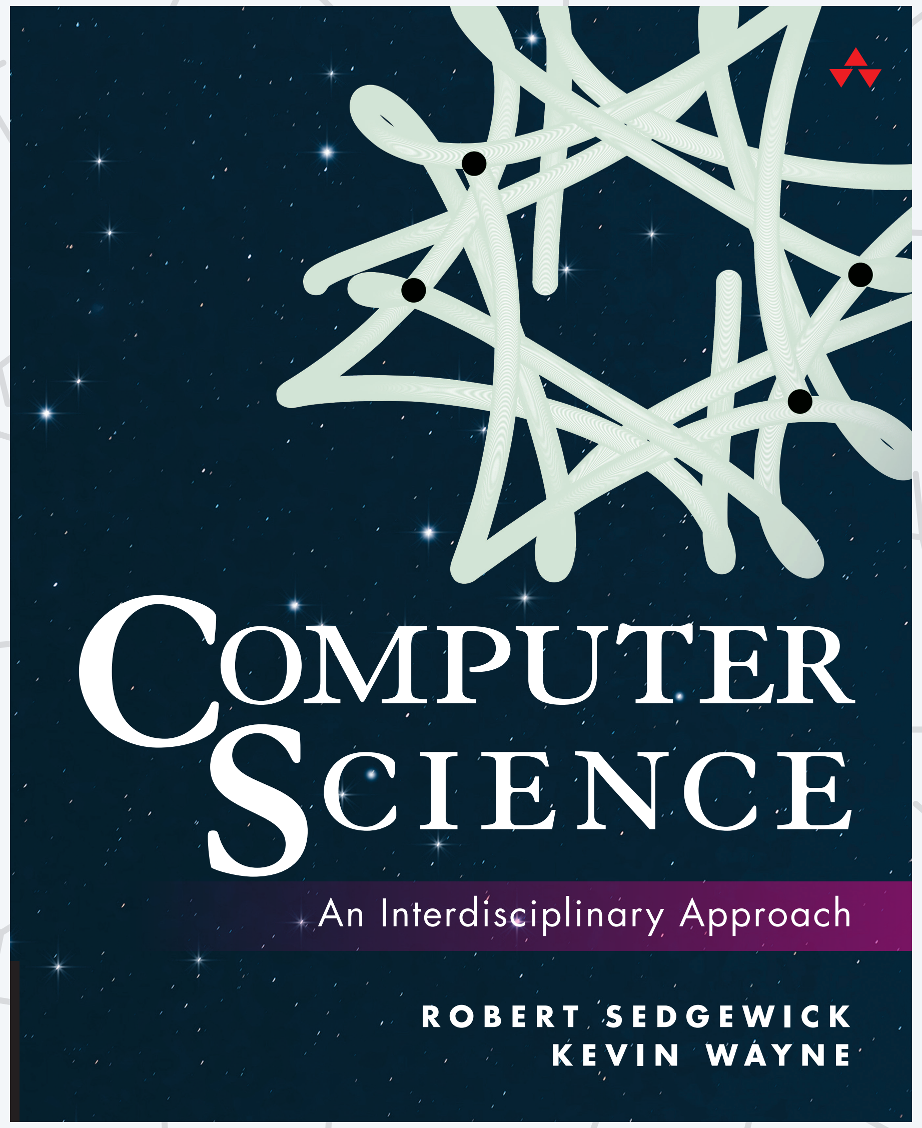


https://introcs.cs.princeton.edu

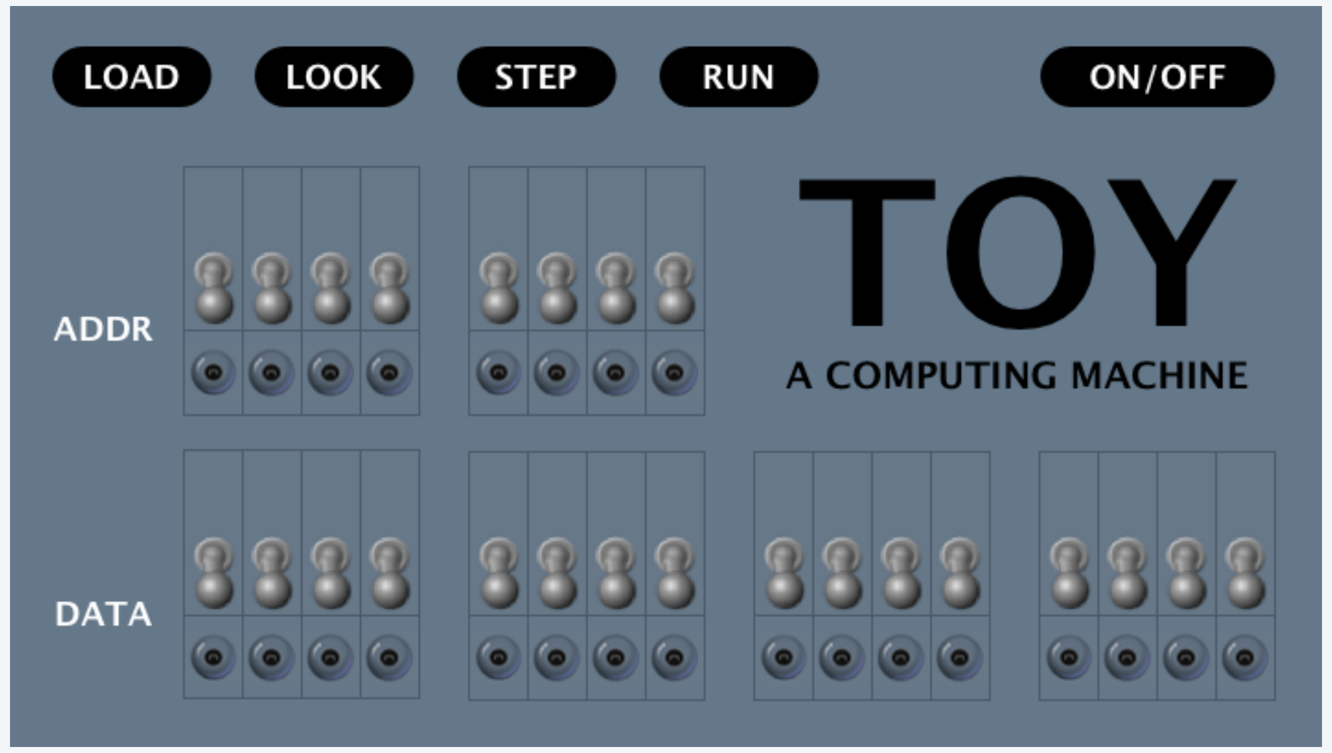
6. TOY Machine II

- *conditionals and loops*
- *input and output*
- *arrays*
- *von Neumann architecture*
- *TOY emulator*

Machine language programming in TOY

TOY machine.

- Arithmetic logic unit (ALU).
- Memory and registers.
- Program counter (PC) and instruction register (IR).
- Lights and switches.



TOY programming.

- Move data between memory and registers.
- Arithmetic/logic operations.

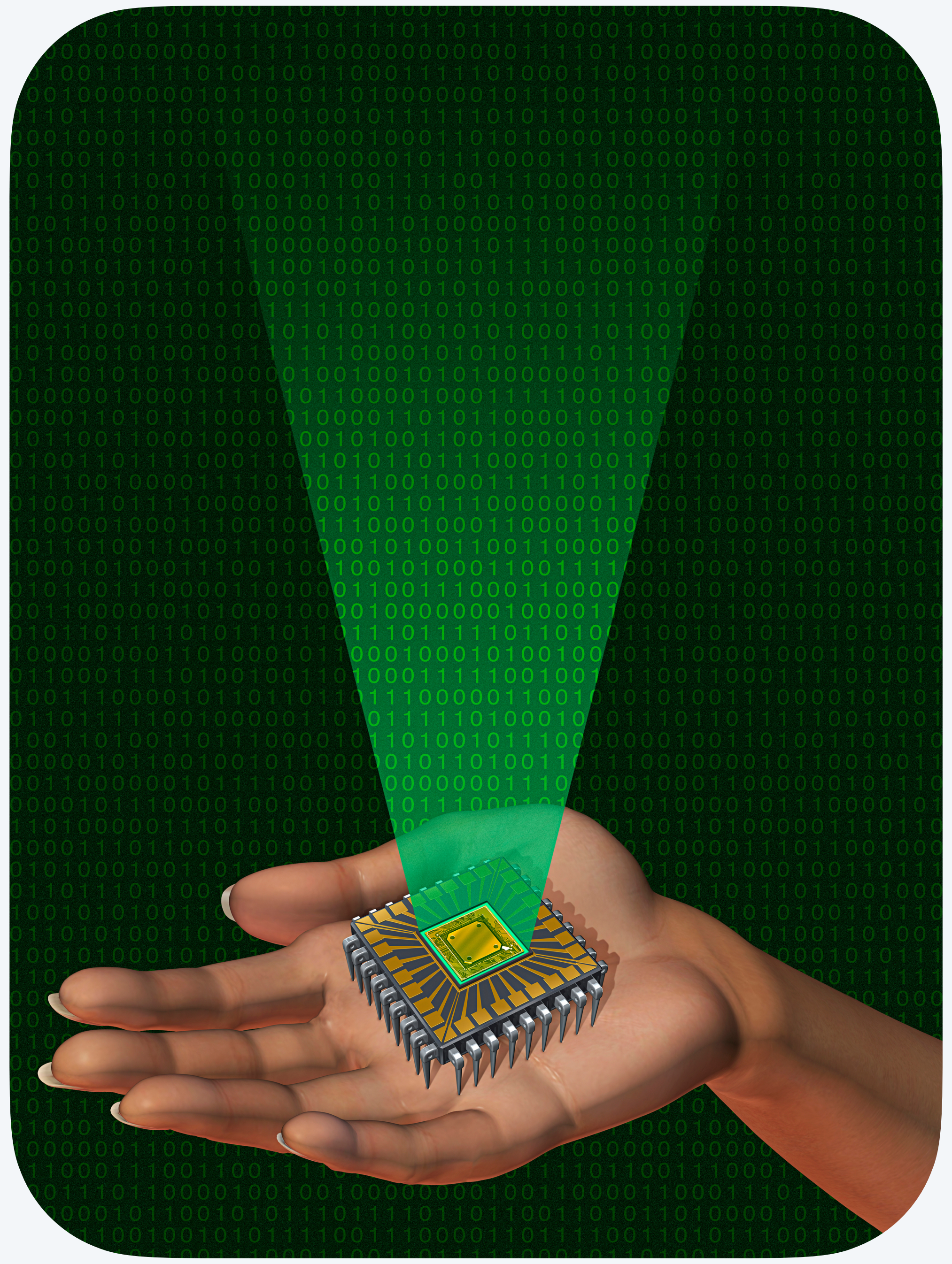
last lecture

- Conditionals and loops.
- Arrays.
- Standard input and output.

this lecture

- Functions.
- Linked structures.

see textbook



Review: your first TOY program

Add two integers.

- Load operands from memory into two registers.
- Add the two registers.
- Store the result in memory.

MEMORY

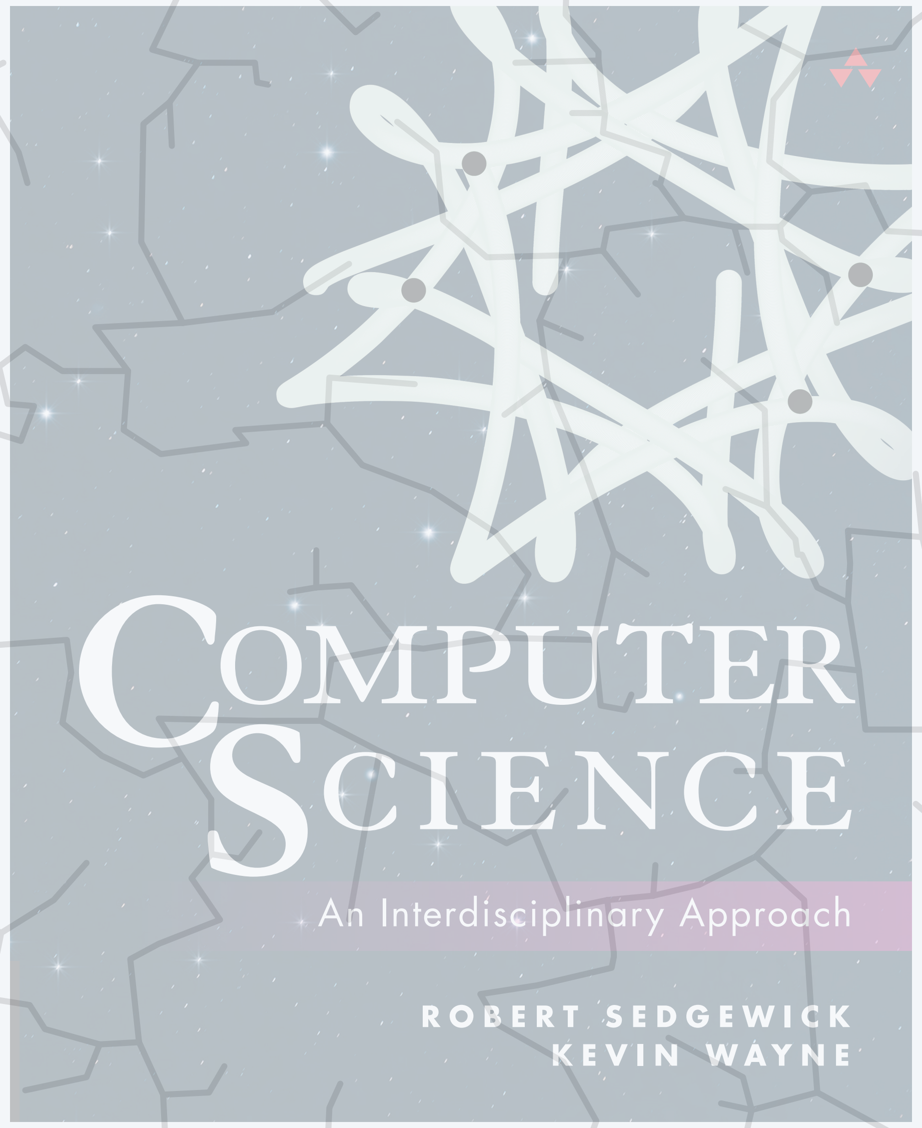
```
 ⋮    ⋮
10: 8A15   R[A] = M[15]
11: 8B16   R[B] = M[16]
12: 1CAB   R[C] = R[A] + R[B]
13: 9C17   M[17] = R[C]
14: 0000   halt
15: 0008   input 1
16: 0005   input 2
17: 0000   output
 ⋮    ⋮
```

REGISTERS

```
  ⋮      ⋮
R[A]   0000
R[B]   0000
R[C]   0000
  ⋮      ⋮
```

PC

10



https://introcs.cs.princeton.edu

6. TOY MACHINE II

- *conditionals and loops*
- *input and output*
- *arrays*
- *von Neumann architecture*
- *TOY emulator*

Conditionals and loops

To control the flow of instruction execution.

- Test a register's value.
- Change the PC, depending on the value.

opcode	instruction	pseudocode
C	*branch if zero*	`if (R[d] == 0) PC = addr`
D	*branch if positive*	`if (R[d] > 0) PC = addr`

Ex 1. Typical `if` statement.

```
14:  ...
15:  DA17   if (R[A] > 0) goto 17
16:  2A0A   R[A] = -R[A]
17:  ...
```

skips line 16 if R[A] > 0

replace R[A] with absolute value of R[A]

```
if (a <= 0) {
   a = -a;
}
```

replace a with |a|

Conditionals and loops

To control the flow of instruction execution

- Test a register's value.
- Change the PC, depending on the value.

opcode	instruction	pseudocode
C	*branch if zero*	`if (R[d] == 0) PC = addr`
D	*branch if positive*	`if (R[d] > 0)  PC = addr`

Ex 2. Typical `while` loop.

```
12:  ...
13:  CA18   if (R[A] == 0) goto 18
14:  ...
15:  7101   R[1] = 1
16:  2AA1   R[A] = R[A] - 1
17:  C013   goto 13
18:  ...
```

skip lines 14 to 17 if R[A] *is* 0

load address

goto line 13 (R[0] *is always* 0)

line 14 is repeated R[A] times (assuming R[A] is non-negative)

```
while (a != 0) {
   ...
   a--;
}
```

repeat a times

Multiplication

Goal. Compute product of two positive integers: $c = a \times b$.

Algorithm. Initialize $c = 0$; then, add b to c, a times.

opcode	instruction	pseudocode
C	*branch if zero*	`if (R[d] == 0) PC = addr`
D	*branch if positive*	`if (R[d] > 0) PC = addr`

```
10:  8A1A   R[A] = M[1A]
11:  8B1B   R[B] = M[1B]
12:  7C00   R[C] = 0
13:  CA18   if (R[A] == 0) goto 18
14:  1CCB   R[C] = R[C] + R[B]
15:  7101   R[1] = 1
16:  2AA1   R[A] = R[A] - 1
17:  C013   goto 13
18:  9C1A   M[1C] = R[C]
19:  0000   halt
1A:  0007   input a
1B:  0009   input b
1C:  0000   output c = a * b
```

loop template from previous slide

input and output

```
int c = 0;
while (a != 0) {
   c = c + b;
   a--;
}
```

multiplication: c = a × b
(via repeated addition)

TOY II: quiz 1

Upon termination, which value is stored in R[A] **?**

A. 0000

B. 7FFF

C. FFFF

D. Infinite loop

```
10: 7A00    R[A] = 0
11: 7101    R[1] = 1            ← R[1] is always 1
12: 1AA1    R[A] = R[A] + 1
13: C012    goto 12             ← R[0] is always 0
14: 0000    halt
```

TOY II: quiz 2

Upon termination, which value is stored in `R[B]` ?

A. 0000

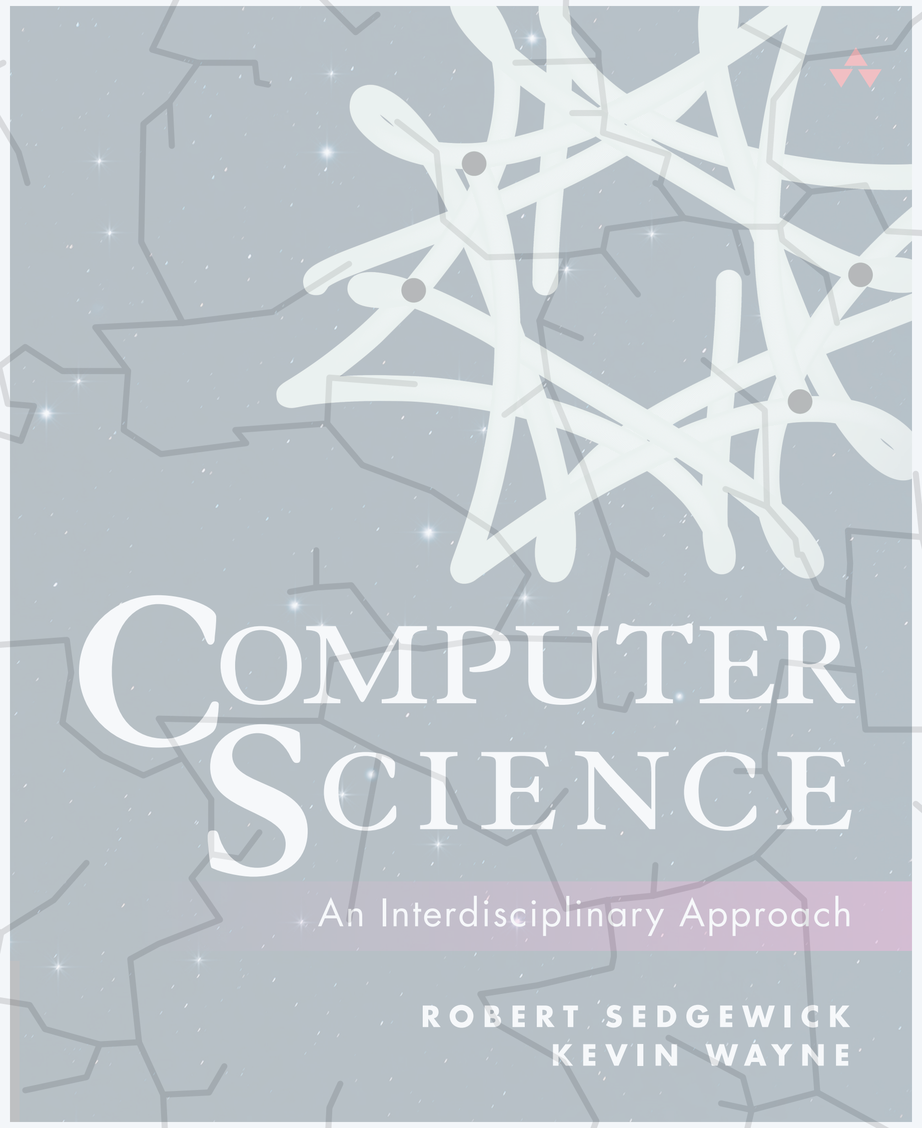
B. 0010

C. 0016

D. 1020

E. 8000

```
10: 7101   R[1] = 1
11: 7A04   R[A] = 4₁₀
12: 7B01   R[B] = 1
13: 2AA1   R[A] = R[A] - 1
14: 1BBB   R[B] = R[B] + R[B]
15: DA13   if (R[A] > 0) goto 13
16: 0000   halt
```



https://introcs.cs.princeton.edu

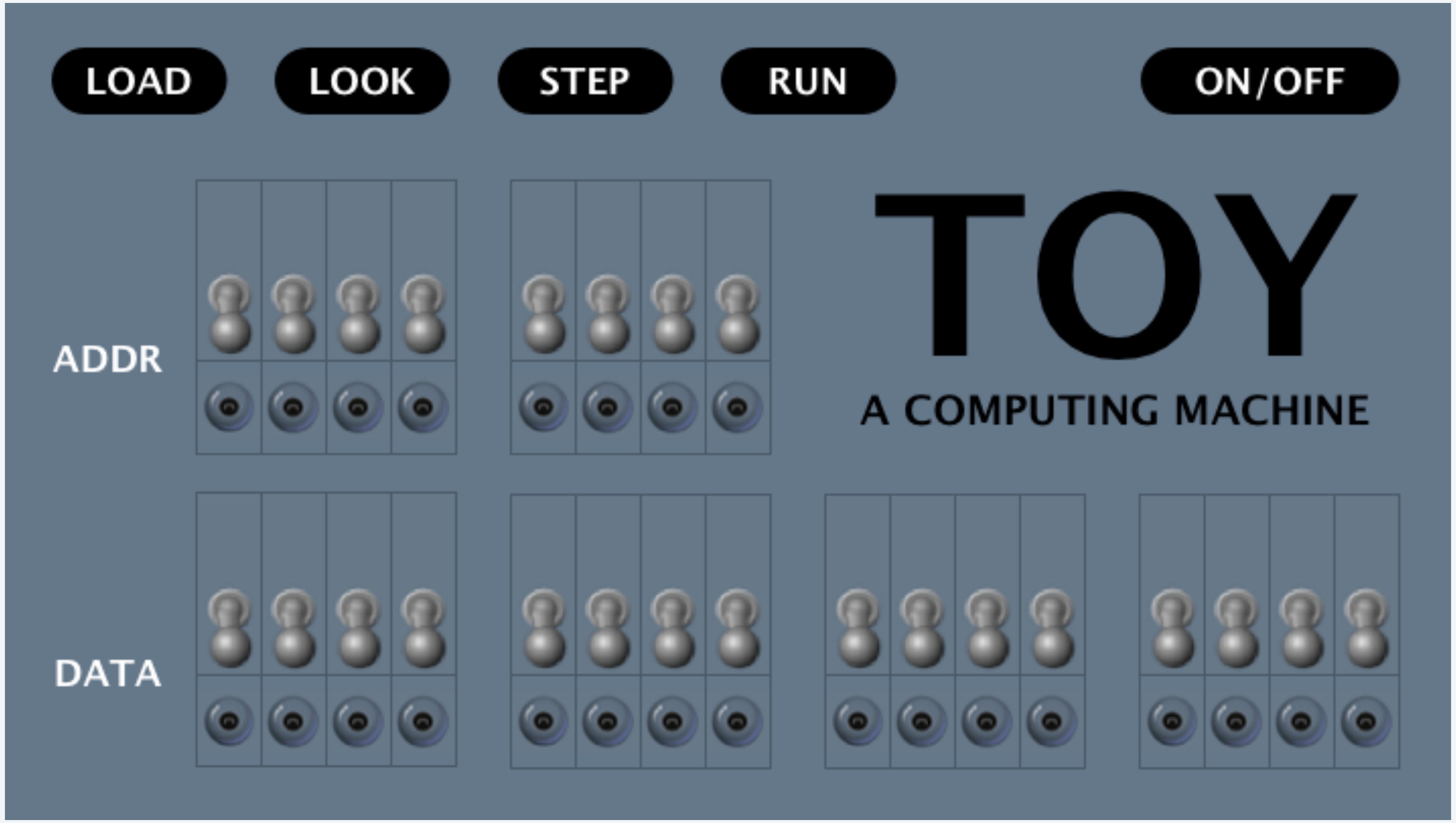
6. TOY Machine II

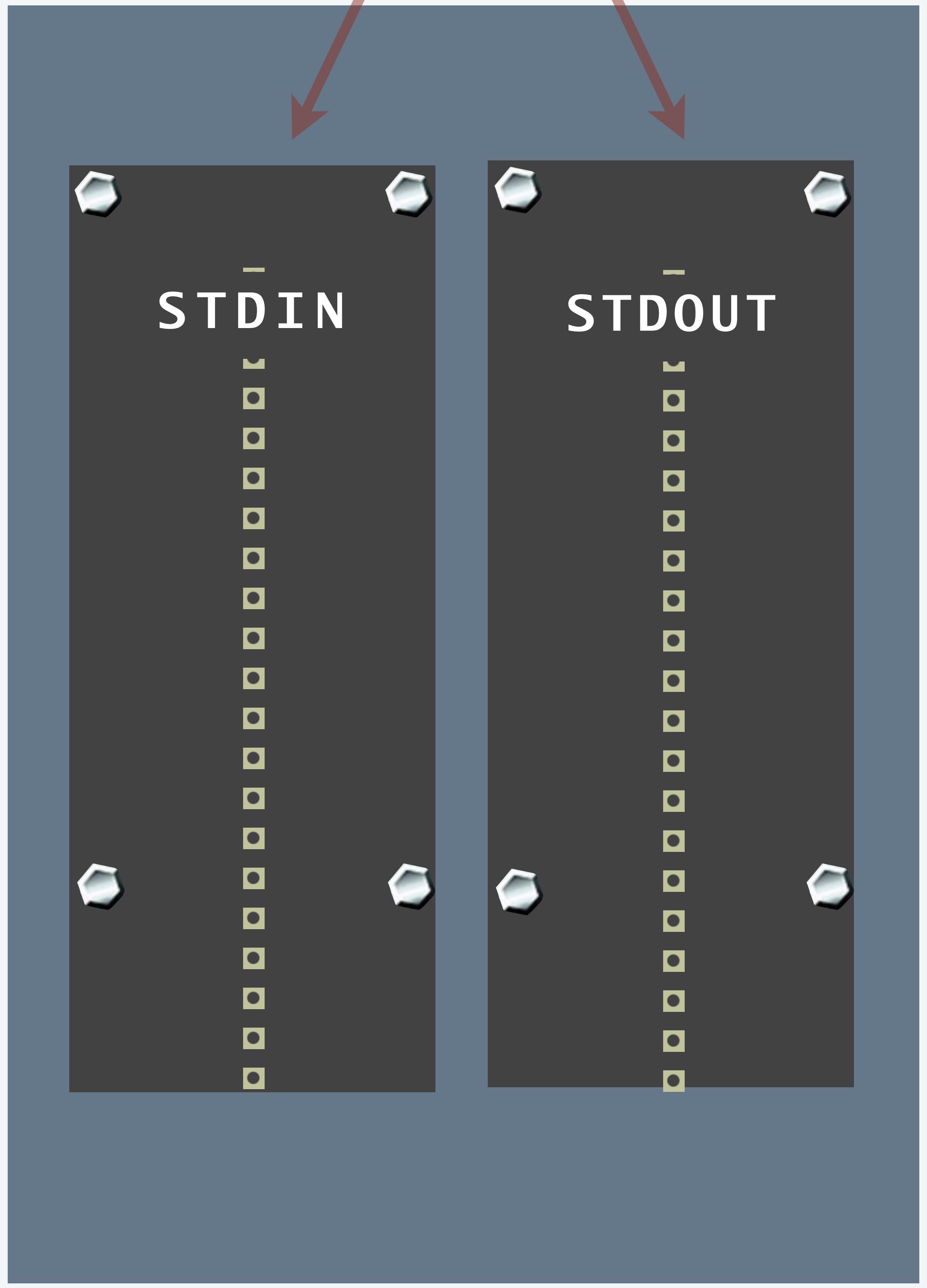
- conditionals and loops
- *input and output*
- arrays
- von Neumann architecture
- TOY emulator

Standard input and output

An immediate problem. Can't address real-world problems with just switches and lights for I/O.

bolt I/O devices to the side of machine





Standard input and output

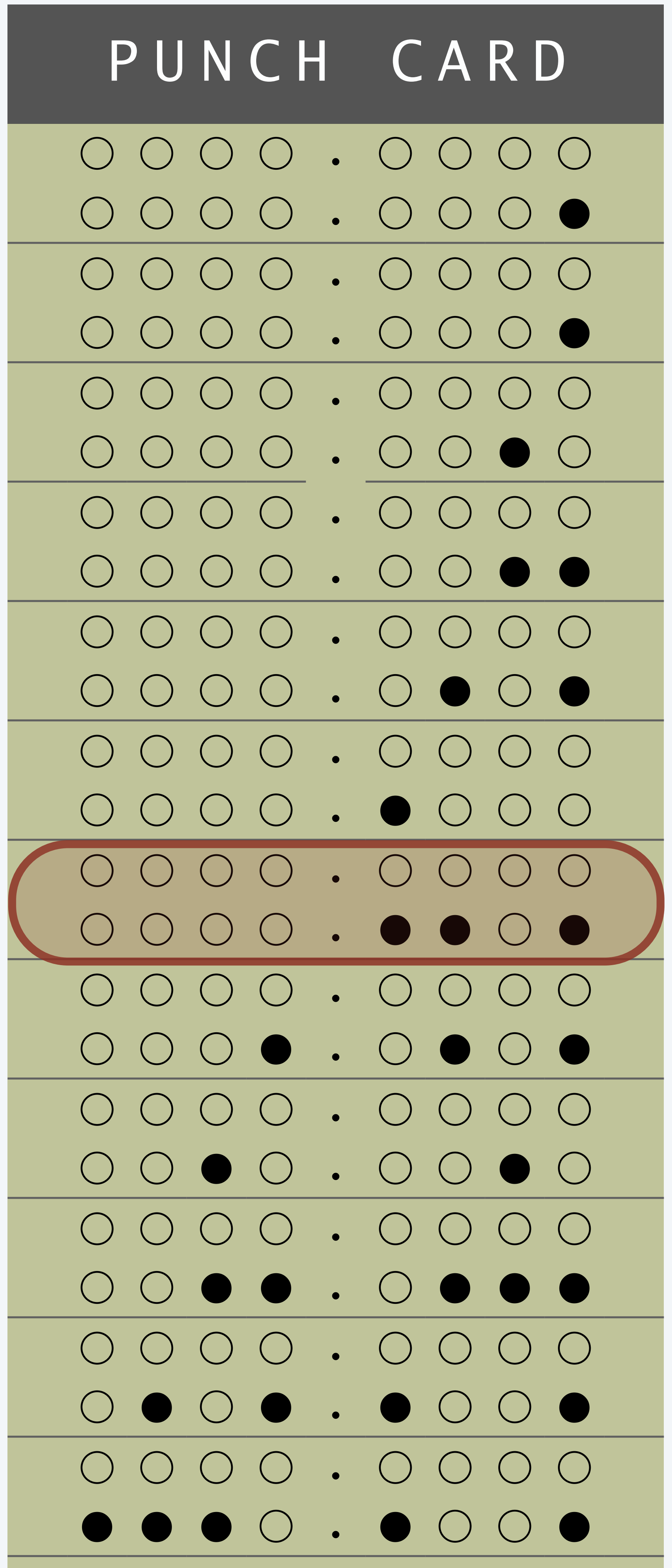
Punched paper tape.

- Encode each 16-bit word in two 8-bit rows.
- To *write* a word, punch a hole for each 1.
- To *read* a word, shine a light behind the tape and sense the holes.

TOY mechanism.

- Connect hardware to memory address `FF`.
- To write the contents of a register to stdout, store to `M[FF]`.
- To read from stdin into a register, load from `M[FF]`.

two 8-bit rows



0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1

16-bit word

Standard input and output: absolute value

Goal. Read integers from standard input (stop on 0000); write absolute value to standard output.

opcode	operation	pseudocode
8	*load*	R[d] = M[addr]
9	*store*	M[addr] = R[d]

```
10: 8AFF   Read R[A] from stdin
11: CA16   if (R[A] == 0) goto 16
12: DA14   if (R[A] >  0) goto 14
13: 2A0A   R[A] = -R[A]
14: 9AFF   write R[A] to stdout
15: C010   goto 10
16: 0000   halt
```

read from standard input (since address is FF)

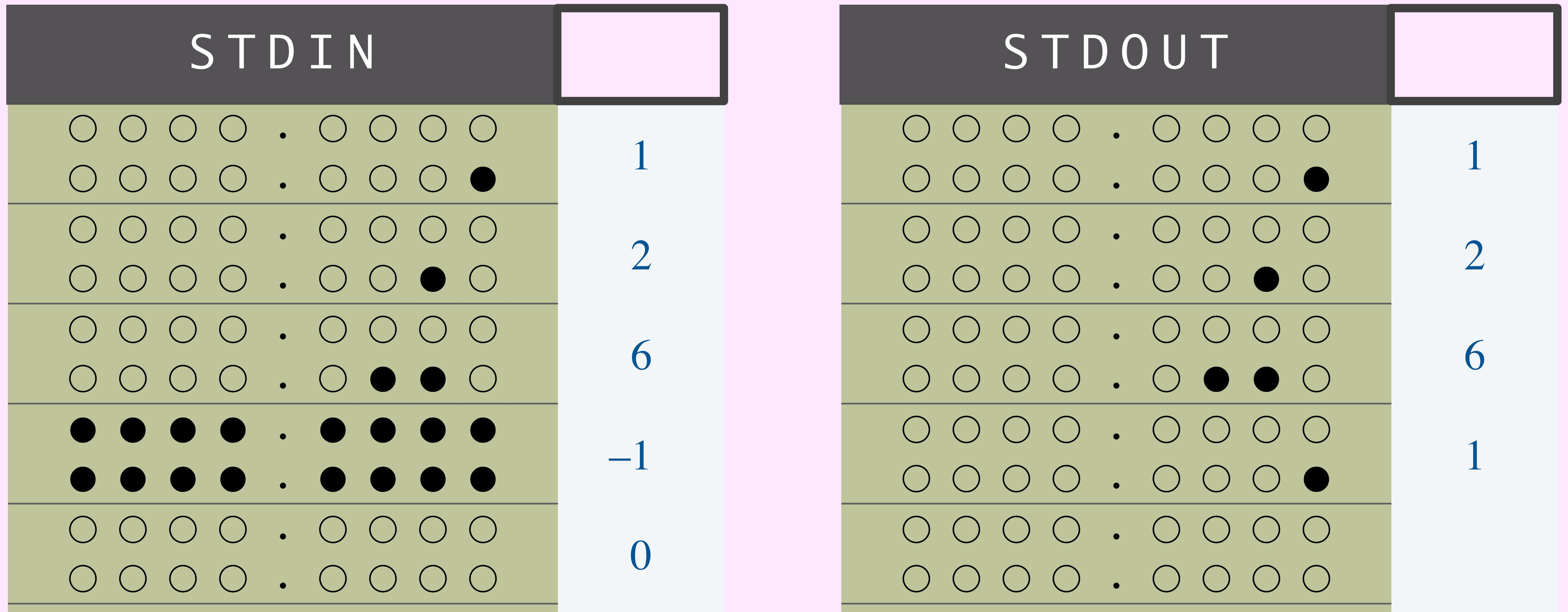
write to standard output (since address is FF)

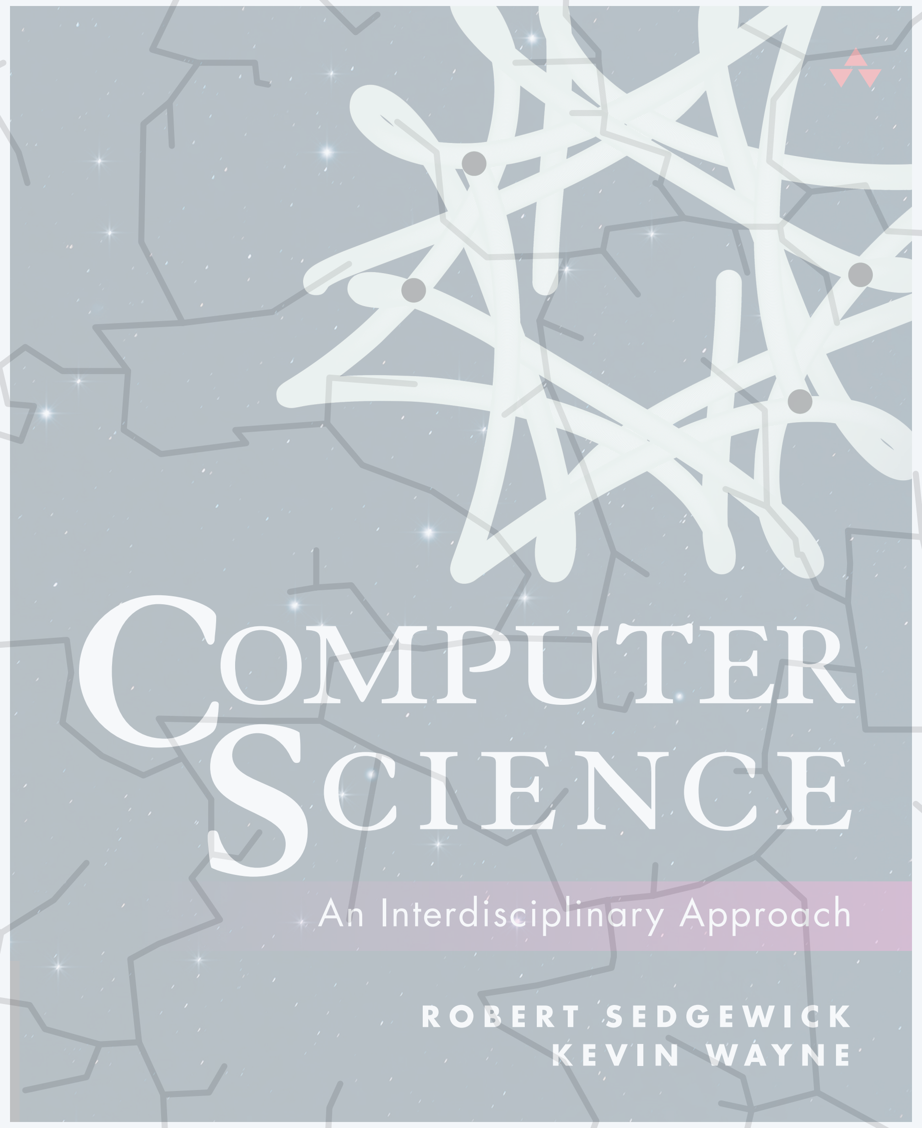
```
while (true) {
   a = StdIn.readInt();
   if (a == 0) break;
   if (a <= 0) a = -a;
   StdOut.println(a);
}
```

Standard input and output trace

Goal. Read integers from standard input (stop on 0000);
write absolute value to standard output.

```
10: 8AFF    Read R[A] from stdin
11: CA16    if (R[A] == 0) goto 16
12: DA14    if (R[A] >  0) goto 14
13: 2A0A    R[A] = -R[A]
14: 9AFF    write R[A] to stdout
15: C010    goto 10
16: 0000    halt
```





https://introcs.cs.princeton.edu

6. TOY Machine II

- conditionals and loops
- input and output
- arrays
- von Neumann architecture
- TOY emulator

TOY II: quiz 3

Upon termination, which value is stored in `R[C]` ?

opcode	operation	pseudocode
7	*load address*	`R[d] = addr`
8	*load*	`R[d] = M[addr]`
A	*load indirect*	`R[d] = M[R[t]]`

A. `000A`

B. `0011`

C. `8B12`

D. `AC0A`

```
10: 7A11    R[A] = 11         ← load address
11: 8B12    R[B] = M[12]      ← load
12: AC0A    R[C] = M[R[A]]    ← load indirect
13: 0000    halt
```

REGISTERS

⋮	⋮
R[A]	0 0 0 0
R[B]	0 0 0 0
R[C]	0 0 0 0
⋮	⋮

PC

1 0

Arrays

To implement an array:

- Keep array elements contiguous in memory, say, starting at 80.
- Access array element i at M[80 + i]. using load/store indirect.

Goal. Print elements in an array of length $n > 0$ to standard output.

opcode	operation	pseudocode
7	*load address*	`R[d] = addr`
A	*load indirect*	`R[d] = M[R[t]]`
B	*store indirect*	`M[R[t]] = R[d]`

```
10: 7A80   R[A] = 80
11: 7B09   R[B] = 9

12: 7101   R[1] = 1
13: AC0A   R[C] = M[R[A]]
14: 9CFF   write R[C]
15: 1AA1   R[A] = R[A] + 1
16: 2BB1   R[B] = R[B] - 1
17: CB13   if (R[B] > 0) goto 13
18: 0000   halt
```

array starts at R[A] = 80 *and has length* R[B] = 9

load next array element into R[C]

address of next element in array

array of length 9

```
80: C0DE
81: CAFE
82: ABBA
83: 8BAD
84: F00D
85: FACE
86: 1377
87: D1CE
88: C1A0
```

TOY II: quiz 4

Suppose that we execute the same program, but initialize R[A] to 10. What is the result?

A. Prints 0010, 0011, 0012, …, 0018.

B. Prints 7A10, 7B09, 7101, …, 0000. ← *treats the TOY program as data (and prints the program)*

C. Crashes when R[A] is 0013.

D. Infinite loop.

```
10: 7A10    R[A] = 10
11: 7B09    R[B] = 9

12: 7101    R[1] = 1
13: AC0A    R[C] = M[R[A]]
14: 9CFF    write R[C]
15: 1AA1    R[A] = R[A] + 1
16: 2BB1    R[B] = R[B] - 1
17: CB13    if (R[B] > 0) goto 13
18: 0000    halt
```

array now starts at R[A] = 10

Indirection

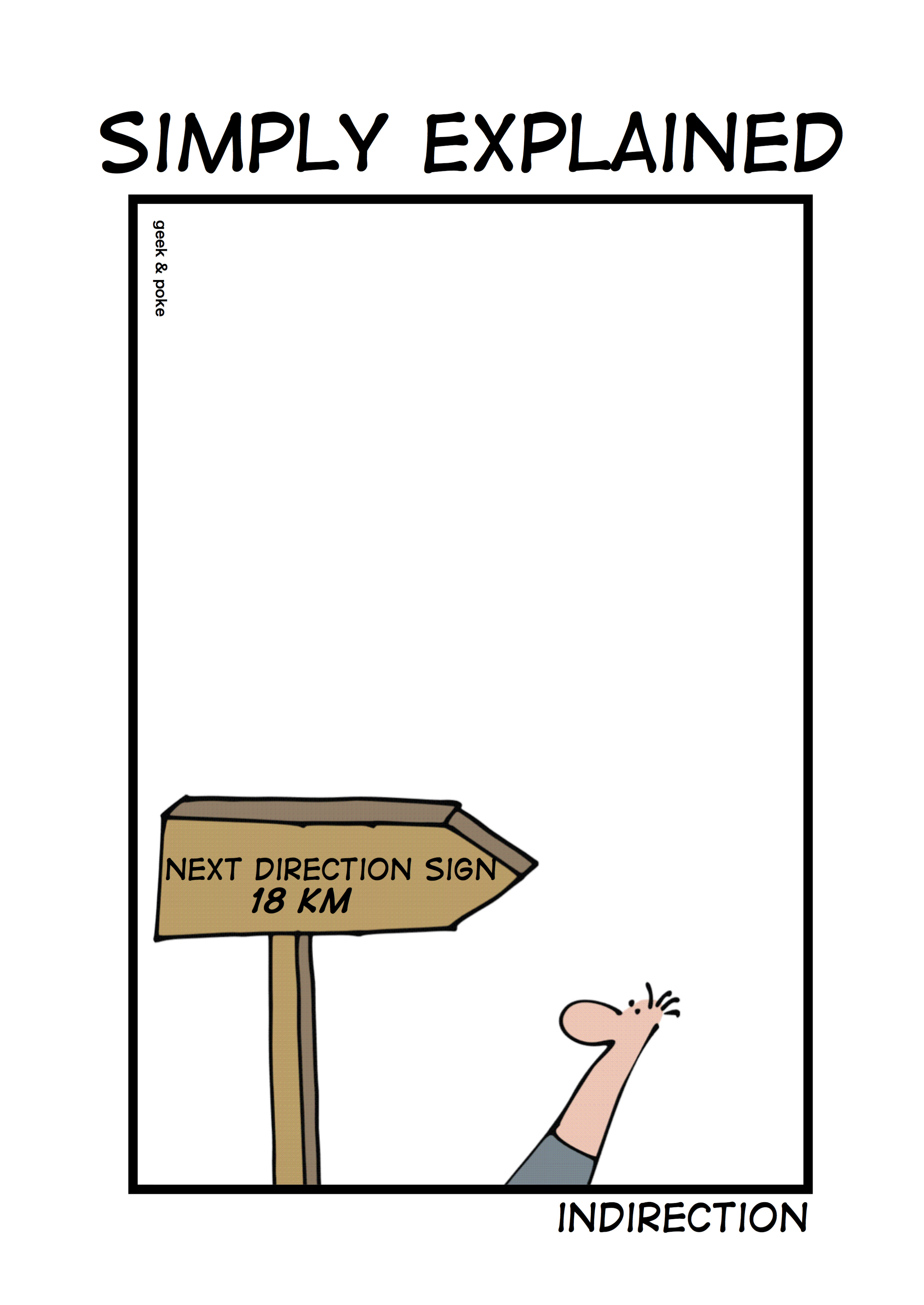
Direct addressing. Specify memory address to access.

Indirect addressing. Specify register containing memory address to access.

Pointer. Variable/register that stores a memory address.

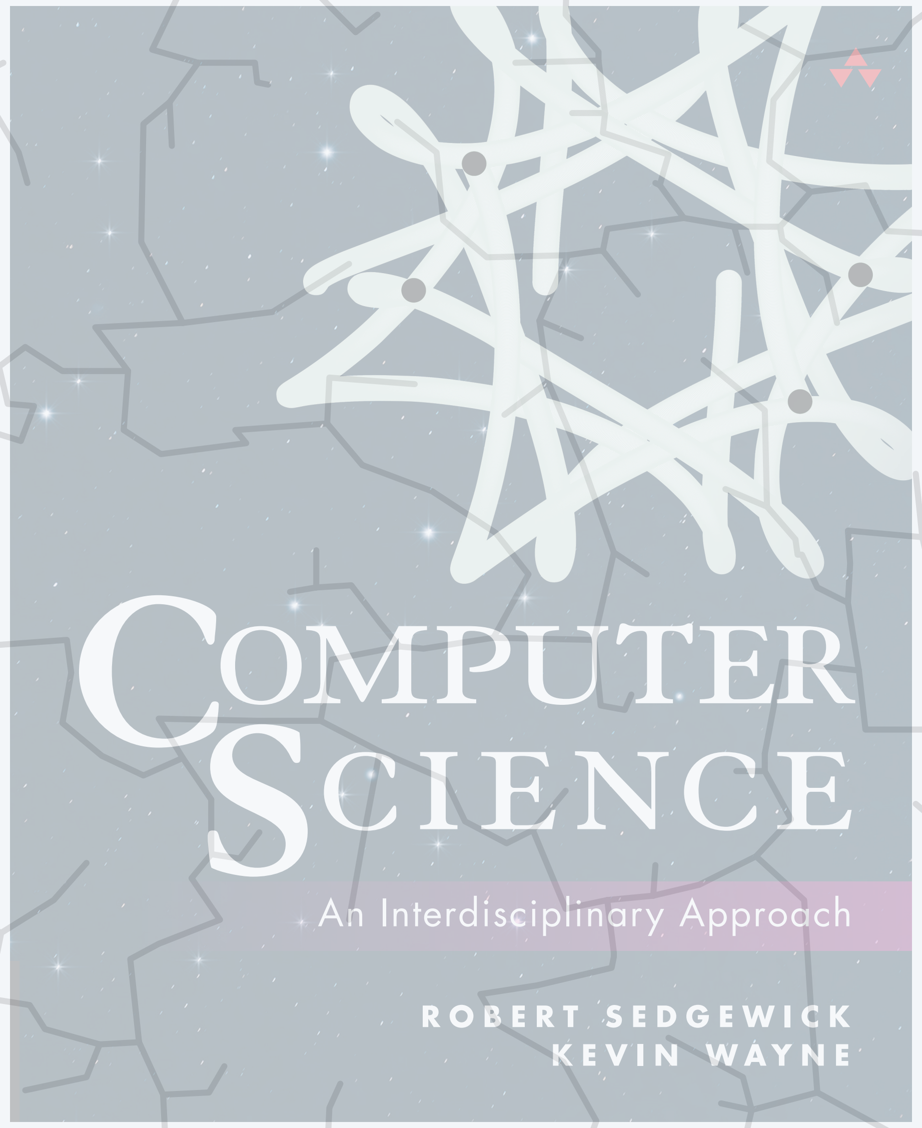
Indirection. Manipulating a value through its memory address.

- TOY arrays.
- Java references.
- C pointers.
- …



" *Any problems in computer science can be solved with another level of indirection.*"

— *attributed to David Wheeler*



https://introcs.cs.princeton.edu

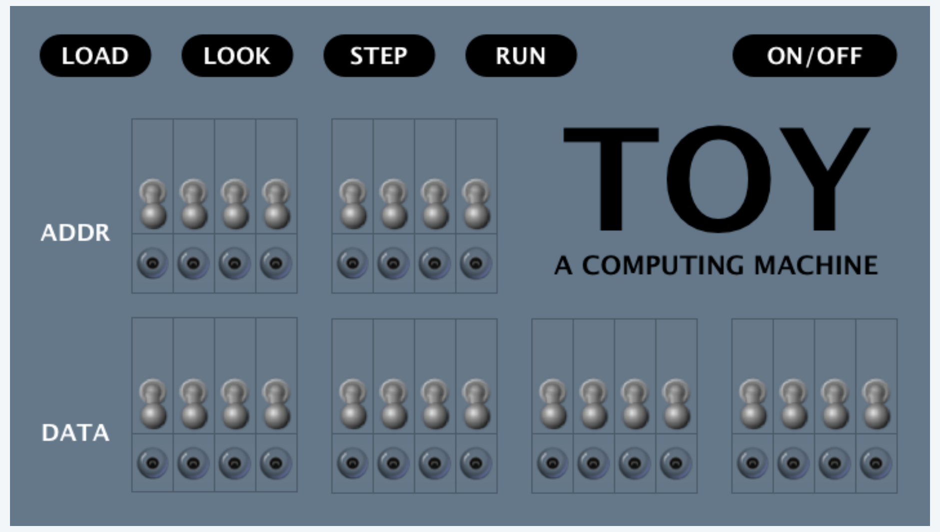
6. TOY Machine II

- conditionals and loops
- input and output
- arrays
- von Neumann architecture
- TOY emulator

TOY vs. your laptop

Two different computing machines.

- Both implement basic data types, conditionals, loops, and other low-level constructs.
- Both can have arrays, functions, linked structures, and other high-level constructs.
- Both have unbounded input and output streams.



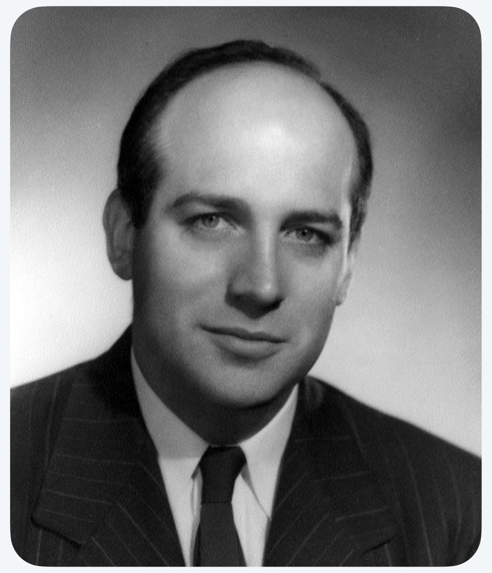
A few key differences.

- Performance: 1Hz vs. 3.5 GHz.
- Memory: 512 bytes vs. 32GB.
- Input/output devices: display, keyboard, trackpad, speakers, webcam, …

An early computer

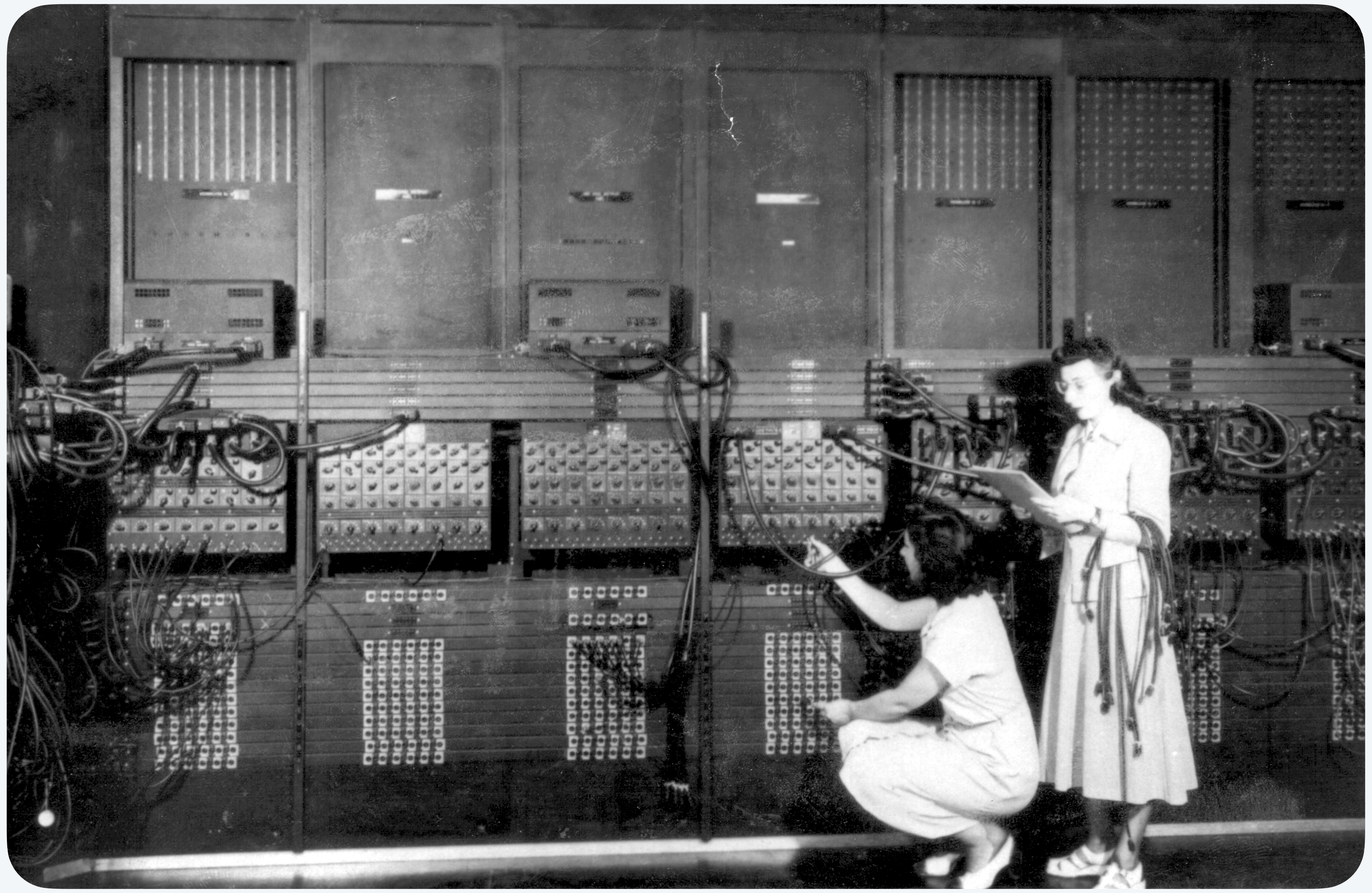
Electronic Numerical Integrator and Calculator (ENIAC).

- First widely-known general-purpose electronic computer.
- "Programmable", but no memory.
- Programming: change switches and cable connections.
- Data: enter numbers using punch cards.



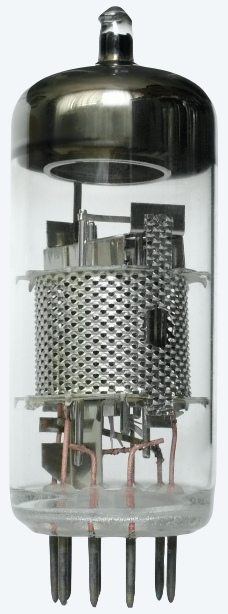
J. Presper Eckert

John W. Mauchly



two programmers "programming" the ENIAC (1946)

facts and figures
30 *tons*
$30 \times 50 \times 8.5$ *feet*
17,468 *vacuum tubes*
300 *multiply/sec*

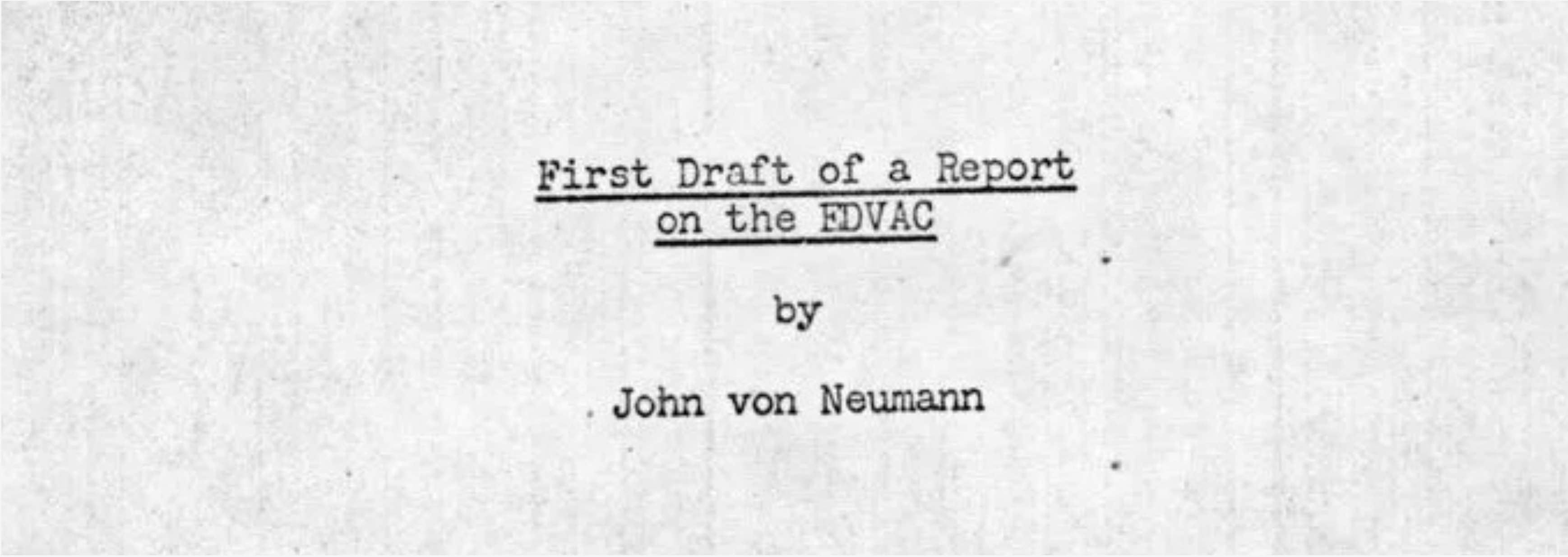


one bit
(vacuum tube)

Von Neumann architecture

First Draft of a Report on the EDVAC (1945).

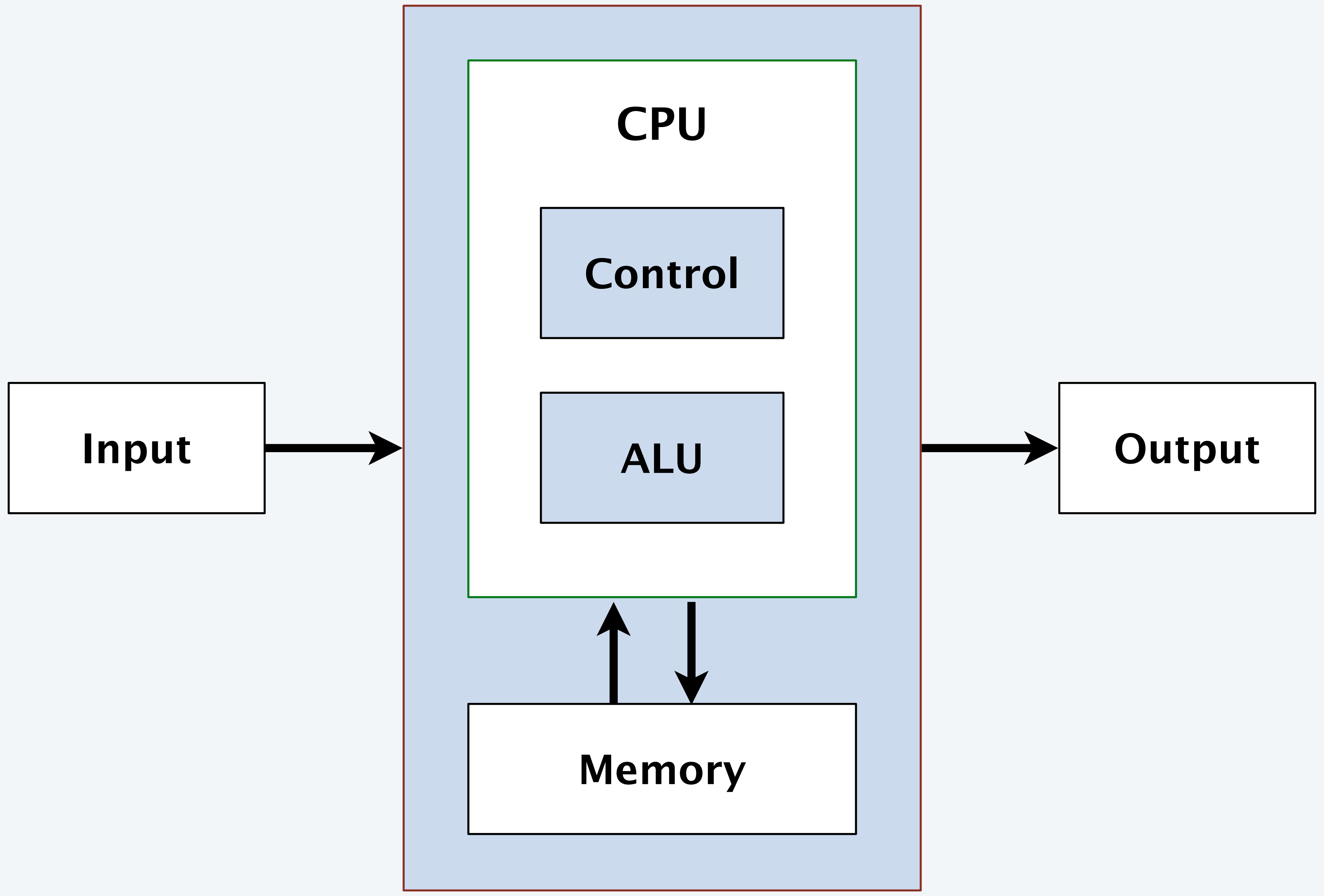
- Brilliant summation of a stored-program machine.
- Written by John von Neumann on a train.
- Based upon EDVAC design of Eckert-Mauchly; influenced by Turing.



First Draft of a Report
on the EDVAC

by

John von Neumann



Keys elements.

- Data and instructions encoded in binary.
- Store both data and instructions in same computer memory.
- ALU, control, memory, registers, and input/output.

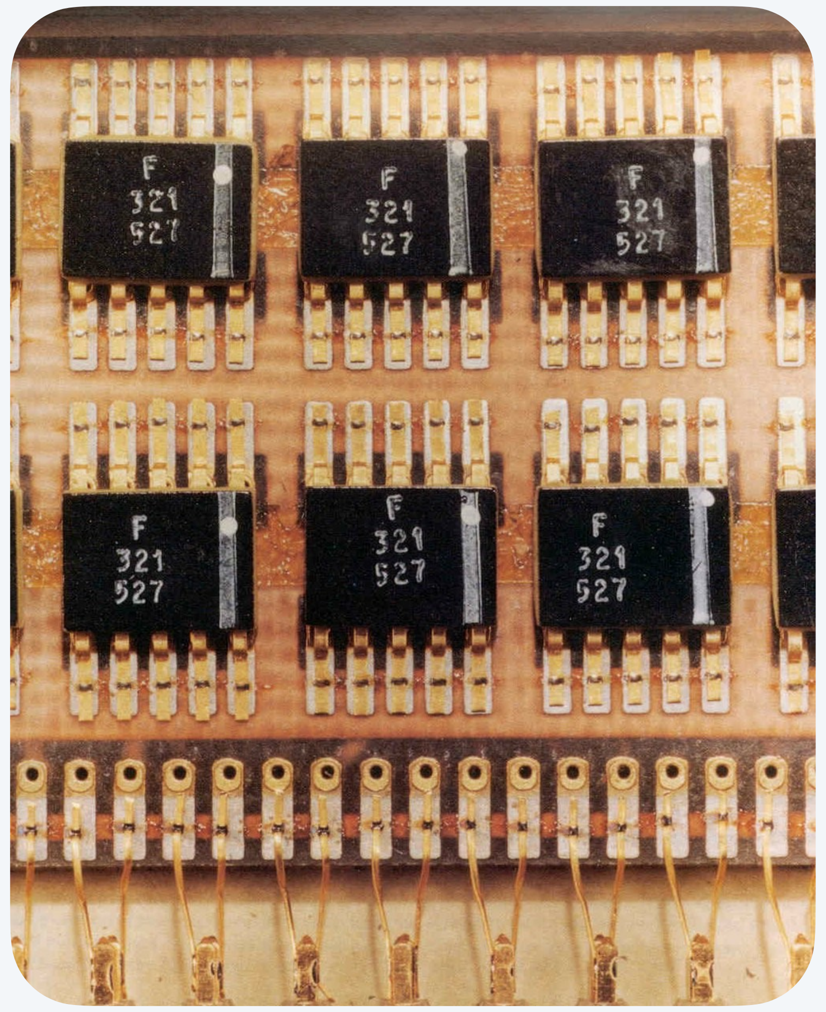
Apollo Guidance Computer

Apollo Guidance Computer. For guidance, navigation, and control of the spacecraft.

- First computer based on silicon integrated circuits.
- Weighed only 70 pounds!
- 1.024 MHz processor speed.
- 4 KB memory.

less powerful than a USB-C charger

Apollo 11
(landed on moon)



integrated circuit

Margaret Hamilton
(lead NASA software engineer)

What does the following program print to standard output?

- **A.** 0000
- **B.** 0088
- **C.** 0088, 0088, 0088, 0088, ...
- **D.** Nothing.

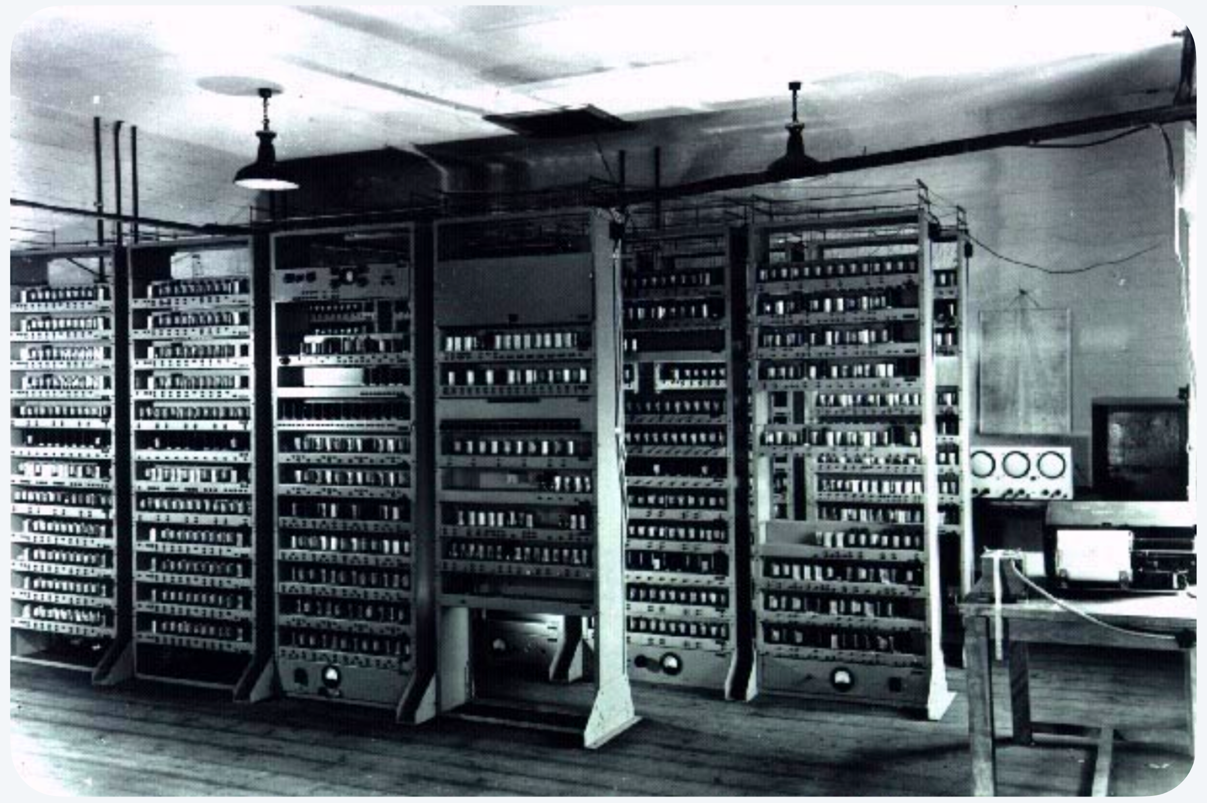
```
10: 7202    R[2] = 2
11: 7A88    R[A] = 88
12: AB16    R[B] = M[16]          ← R[B] stores C017
13: 2BB1    R[B] = R[B] - 2       ← R[B] stores C015
14: BB16    M[16] = R[B]
15: 9AFF    write R[A] to stdout
16: C017    goto 17
17: 0000    halt
```

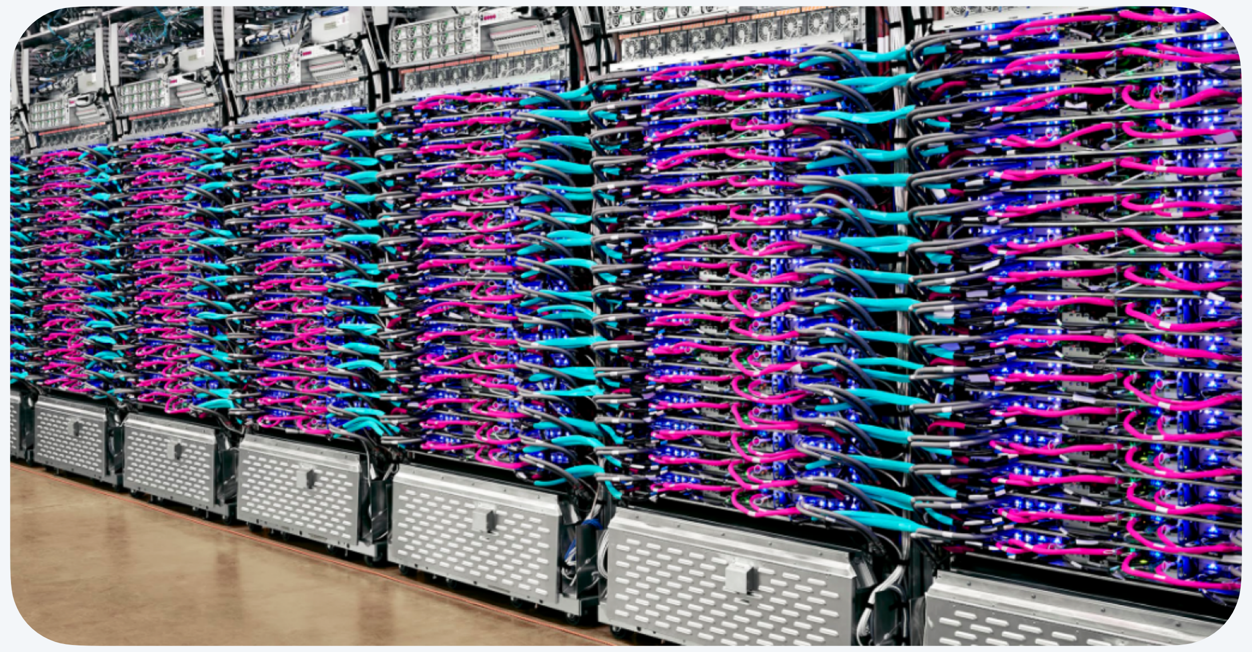
Implications

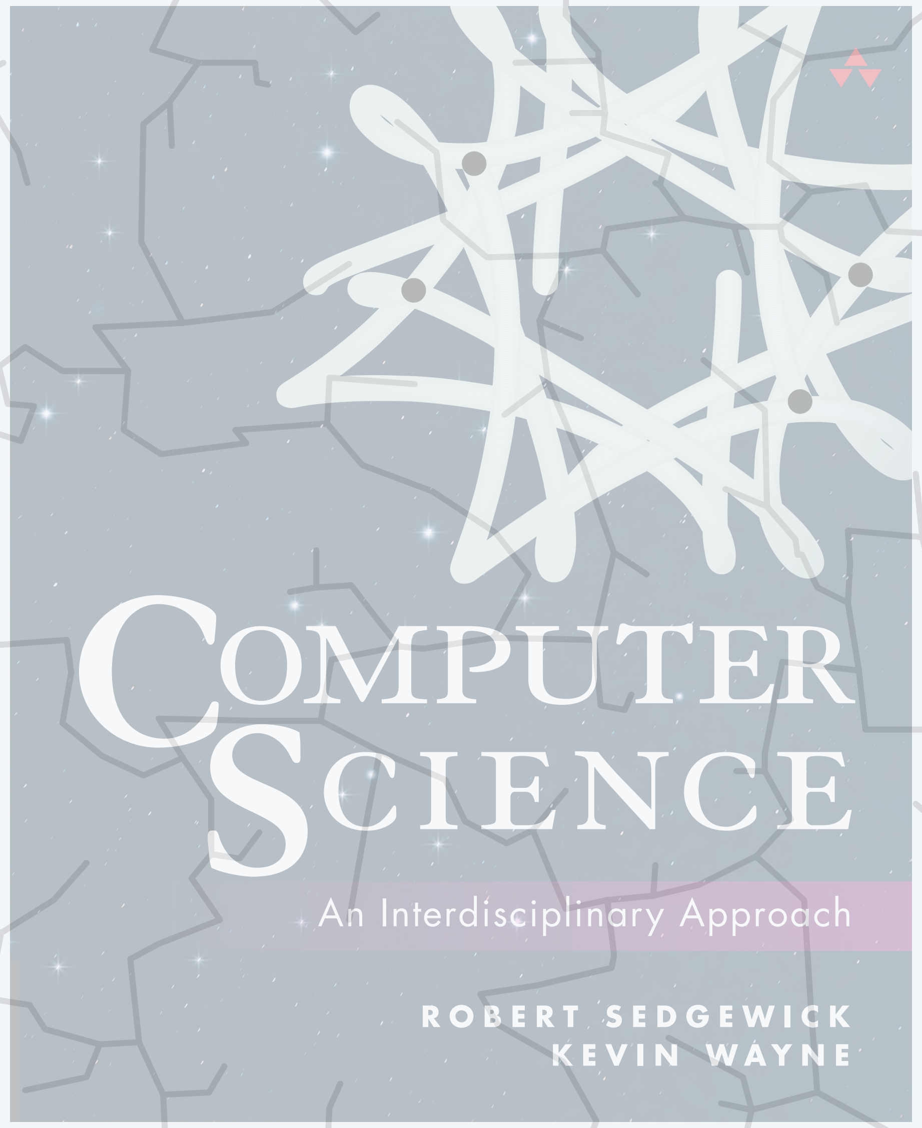
Stored-program (von Neumann) architecture is the basis of nearly all computers since the 1950s.

Practical implications.

- Programming: develop programs without rewiring.
- Download apps: load programs, not just data, into memory.
- Compilers: write programs that take programs as input (and produce programs as output).
- Code-injection attacks: trick program into treating input data as code.







https://introcs.cs.princeton.edu

6. TOY MACHINE II

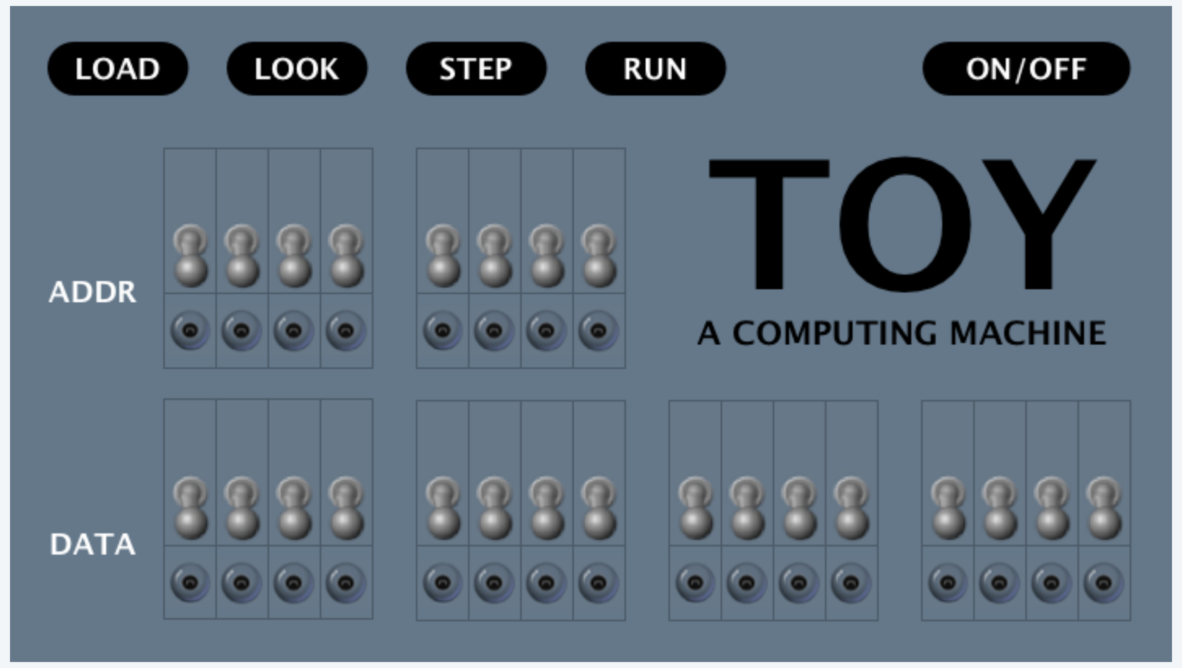
- conditionals and loops
- input and output
- arrays
- von Neumann architecture
- TOY emulator

TOY emulator

Q. How did we debug all our TOY programs?

A. We wrote a Java program to emulate a TOY machine.

Emulator. Hardware or software that enables one computer system to behave like another.



estimated number of Android devices: 1 billion+

estimated number of TOY devices: 1 billion+

Remarks.

- YOU could write a TOY emulator (ahead).
- We designed TOY by refining this code.
- All computers are designed in this way.

TOY emulator in Java: high-level design

Goal. Write a Java program that emulates the TOY machine.

hex literal (starts with 0x)

```
public class TOYLite {
   public static void main(String[] args) {
      int pc = 0x10;            // program counter
      int[] R = new int[16];    // registers
      int[] M = new int[256];   // main memory

      In in = new In(args[0]);
      for (int i = pc; !in.isEmpty(); i++)
         M[i] = Integer.parseInt(in.readString(), 16);

      while (true) {
         // 1. fetch instruction and increment PC
         // 2. decode instruction
         // 3. execute instruction
      }
   }
}
```

base 16

```
~/cos126/toy> more add.toy
8AFF
8BFF
1CAB
9CFF
0000

~/cos126/toy> java-introcs TOYLite add.toy
0008
0005
000D
```

read R[A] and R[B] from standard input

print R[C] to standard output

takes TOY program as input

emulates TOY program and produces same output

TOY emulator: fetch and increment

Fetch. Get instruction from memory location indexed by PC.

Increment. Increment PC by 1.

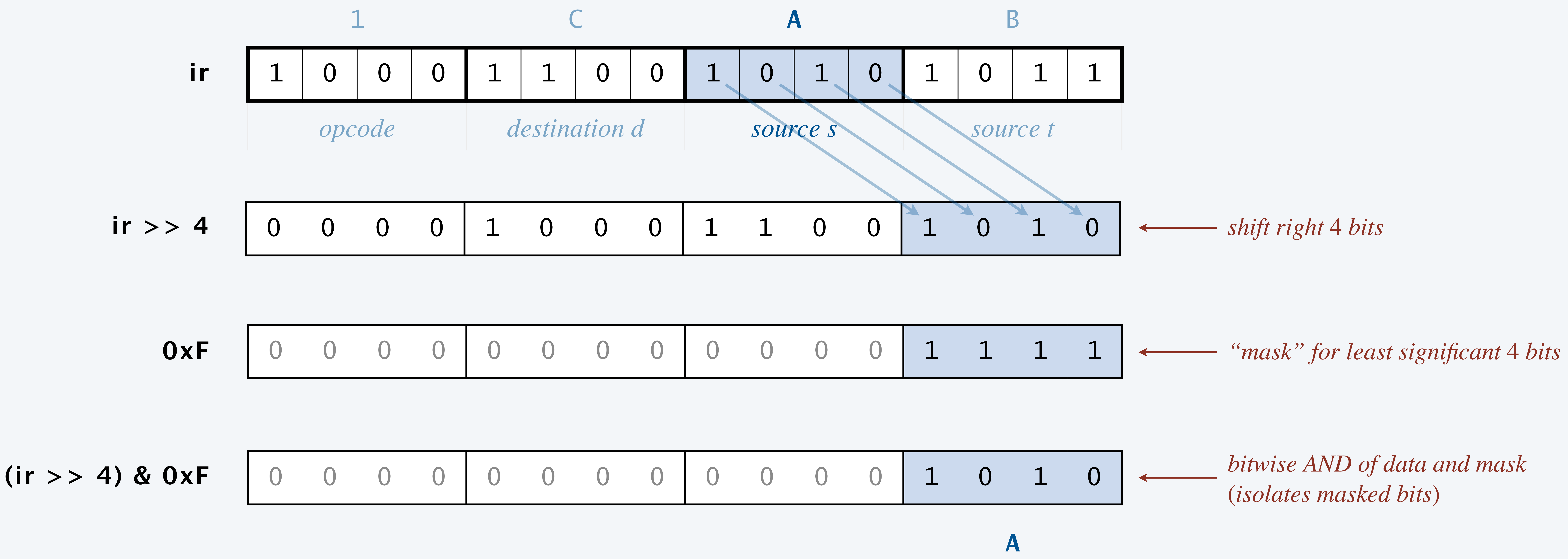
```
int ir = M[pc];     // fetch
pc++;               // increment
```

TOY emulator: decode instruction

Decode. Extract relevant components from instruction register (IR).

- Bitwise operations are the same in Java and TOY.
- Use shift-and-mask technique.

Ex. Extract source s from `1CAB`.



TOY emulator: decode instruction

Decode. Extract relevant components from instruction register (IR).

- Bitwise operations are the same in Java and TOY.
- Use shift-and-mask technique.

```
int ir = M[pc];    // fetch
pc++;              // increment

int op   = (ir >> 12) &  0xF;    // opcode
int d    = (ir >>  8) &  0xF;    // destination d
int s    = (ir >>  4) &  0xF;    // source s
int t    = (ir >>  0) &  0xF;    // source t
int addr = (ir >>  0) & 0xFF;    // address
```

TOY emulator: execute instruction

Execute. Use Java `switch` statement to implement state change for each of 16 instructions.

```
if (op == 0) break;        // halt
```

```
switch (op) {
   case  1:  R[d] = R[s] +  R[t];        break;
   case  2:  R[d] = R[s] -  R[t];        break;
   case  3:  R[d] = R[s] &  R[t];        break;
   case  4:  R[d] = R[s] ^  R[t];        break;
   case  5:  R[d] = R[s] << R[t];        break;
   case  6:  R[d] = R[s] >> R[t];        break;
   case  7:  R[d] = addr;                break;
   case  8:  R[d] = M[addr];             break;
   case  9:  M[addr] = R[d];             break;
   case 10:  R[d] = M[R[t]];             break;
   case 11:  M[R[t]] = R[d];             break;
   case 12:  if (R[d] == 0) pc = addr;   break;
   case 13:  if (R[d] >  0) pc = addr;   break;
   case 14:  pc = R[d];                  break;
   case 15:  R[d] = pc; pc = addr;       break;
}
```

TOY emulator in Java

A few missing details.

- R[0] is always 0000.
- TOY standard input/output.
- 16-bit TOY word vs. 32-bit Java int.
- More flexible TOY program input format.

Full implementation. See booksite.

Implications.

- Can run any TOY program!
- Can develop TOY code on another machine.
- Easy to change TOY design.

```java
public class TOYLite {
   public static void main(String[] args) {
      int pc   = 0x10;              // program counter
      int[] R = new int[16];        // registers
      int[] M = new int[256];       // main memory

      In in = new In(args[0]);
      for (int i = pc; !in.isEmpty(); i++)
         M[i] = Integer.parseInt(in.readString(), 16);

      while (true) {
         int ir = M[pc++];      // fetch
         pc++;                  // increment

         int op   = (ir >> 12) &  0xF;    // opcode
         int d    = (ir >>  8) &  0xF;    // destination d
         int s    = (ir >>  4) &  0xF;    // source s
         int t    = (ir >>  0) &  0xF;    // source t
         int addr = (ir >>  0) & 0xFF;    // address

         if (op == 0) break;
         switch (op) {
            case  1:  R[d] = R[s] +  R[t];        break;
            case  2:  R[d] = R[s] -  R[t];        break;
            case  3:  R[d] = R[s] &  R[t];        break;
            case  4:  R[d] = R[s] ^  R[t];        break;
            case  5:  R[d] = R[s] << R[t];        break;
            case  6:  R[d] = R[s] >> R[t];        break;
            case  7:  R[d] = addr;                break;
            case  8:  R[d] = M[addr];             break;
            case  9:  M[addr] = R[d];             break;
            case 10:  R[d] = M[R[t]];             break;
            case 11:  M[R[t]] = R[d];             break;
            case 12:  if (R[d] == 0) pc = addr;   break;
            case 13:  if (R[d] >  0) pc = addr;   break;
            case 14:  pc = R[d];                  break;
            case 15:  R[d] = pc; pc = addr;       break;
         }
      }
   }
}
```

state

parse input file

fetch, increment

decode instruction

execute instruction

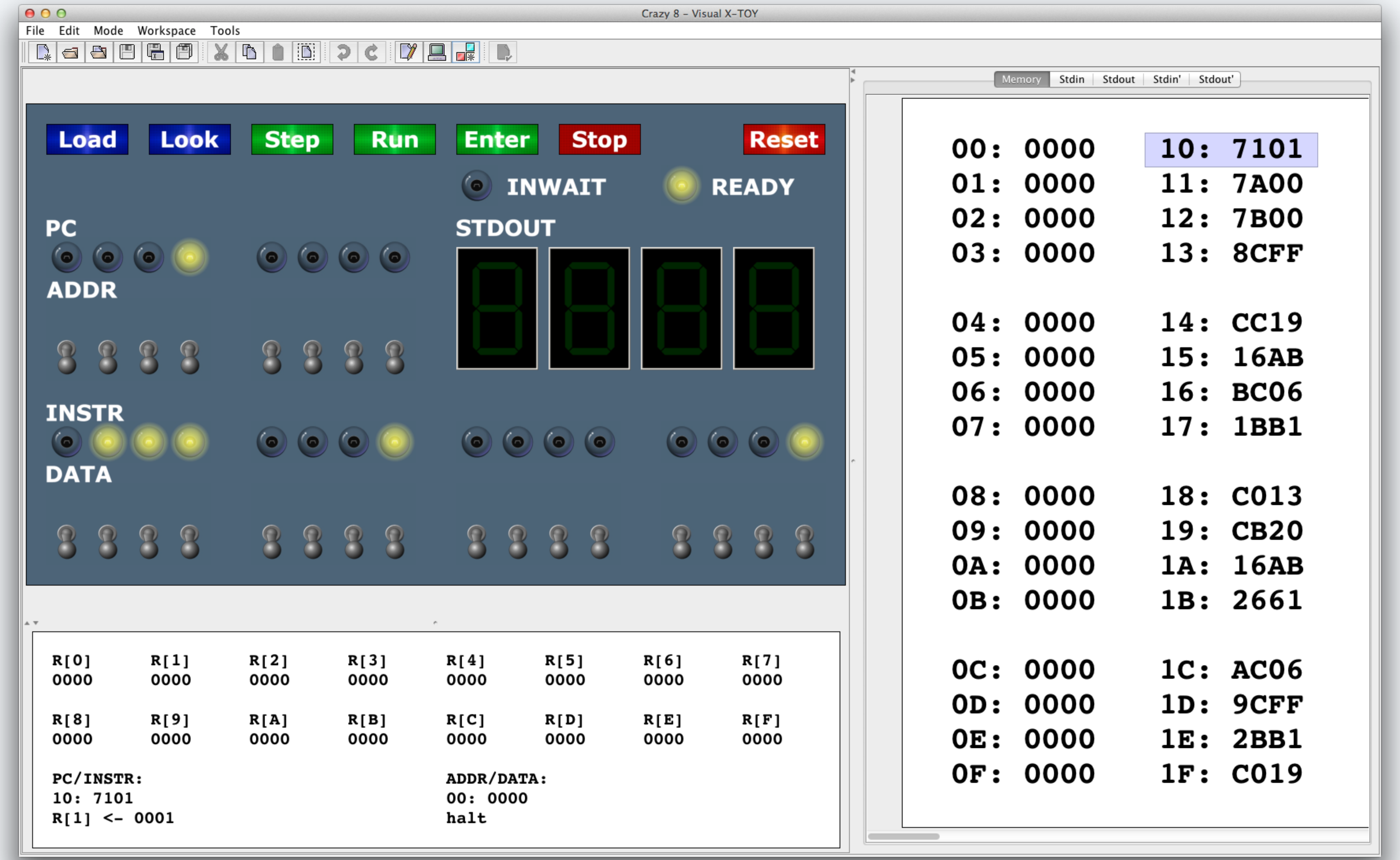
Visual X-TOY

Visual X–TOY. A Java IDE that emulates the TOY machine.

- GUI, text editor, auto-comments, debugger, many other features.
- Written by Brian Tsang ’04 (using Java 1.3). ← *and still works 20 years later!*
- Available on the booksite.
- YOU can develop TOY software. ← *Assignment 8*

Same approach used for all new systems.

- Build simulator and development environment.
- Develop and test software.
- Build and sell hardware.



Backward compatibility

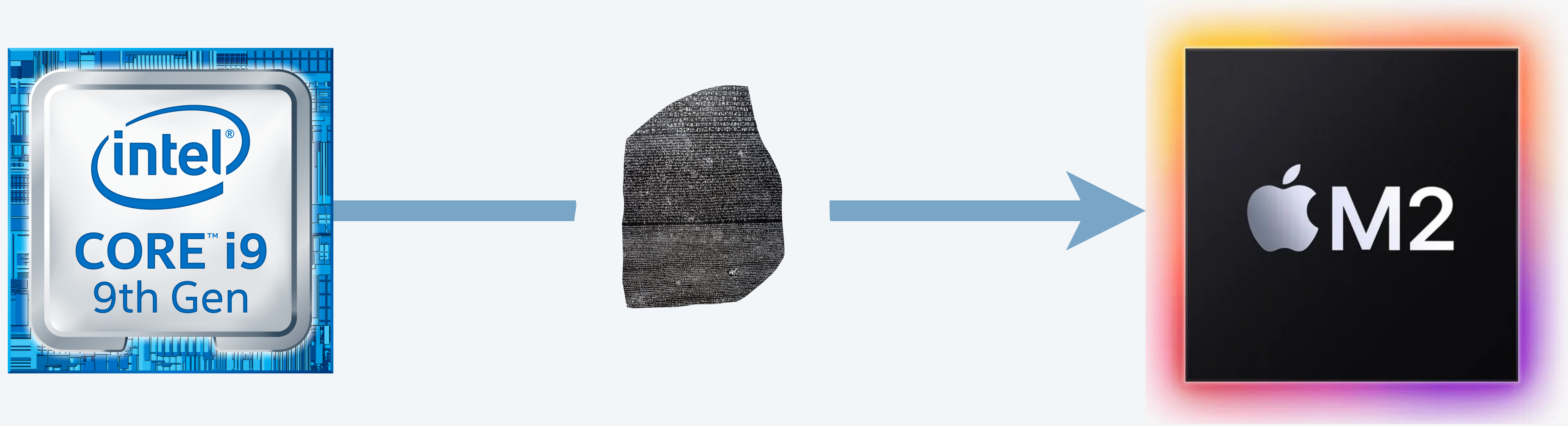
Q. How to run old software on a new machine architecture?

Approach 1. Rewrite it all: time-consuming, expensive, error-prone.

Approach 2. Write an emulator for old computer on the new one.

Ex 1. Pac-Man.

Ex 2. Rosetta 2. ←— *run 64-bit Intel on Apple Silicon*



64-bit Intel

Apple Silicon

Pac-Man 1980s
(arcade machine)

Pac-Man 2020s
(Android phone)

Impact. Old software remains available.

Virtual machines

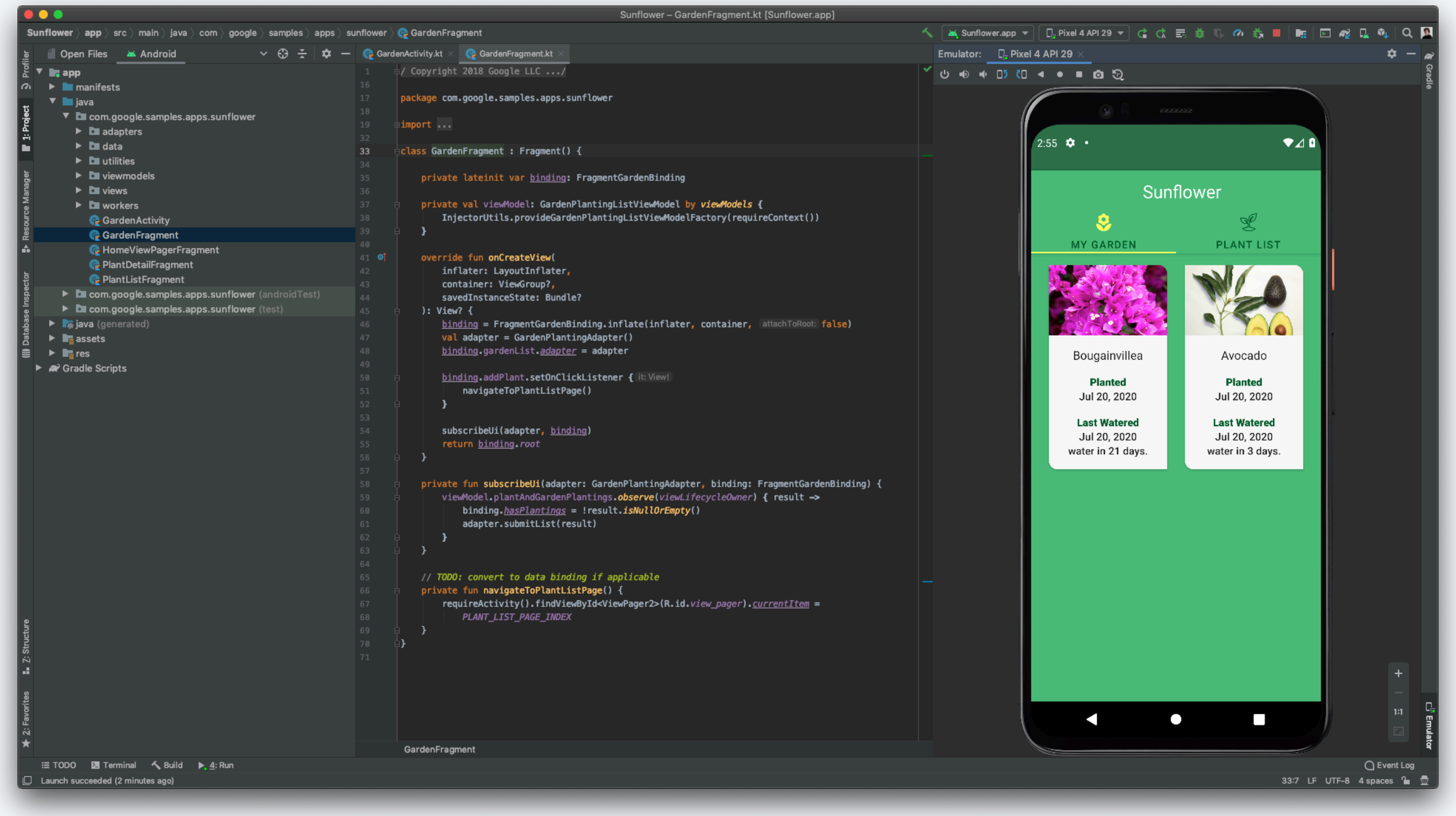
Virtual machine. Software-based emulation of a physical computer.

- Can run/develop software without having physical computer.
- Provides portability, scalability, flexibility, and security.

"write once, run anywhere"

Java virtual machine (JVM). Abstract machine that can execute Java `.class` files.

Mobile app IDEs. Provide emulator for Android, iPhone, Apple watch, …



Cloud computing. Virtual CPU, memory, storage, OS, and network.

Amazon EC2

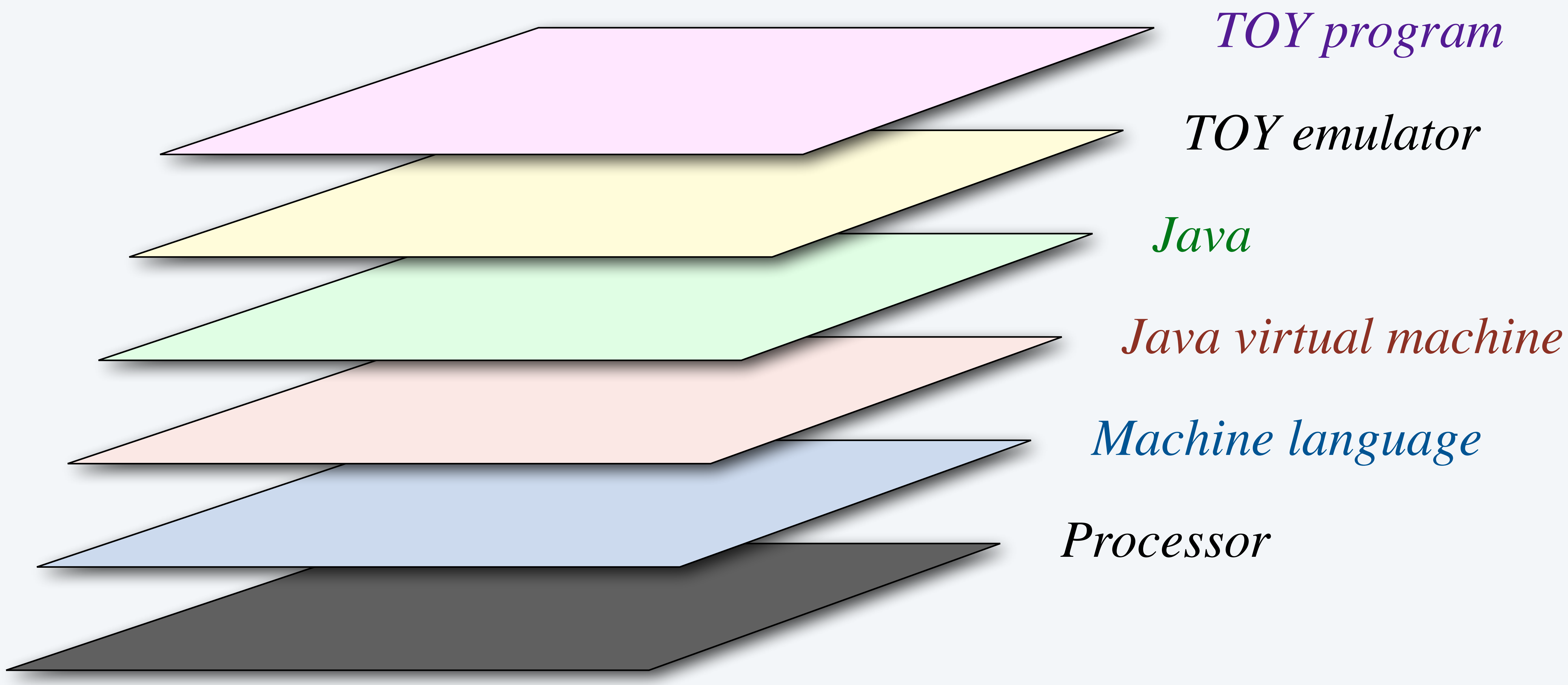
Google CE

Microsoft Azure

Layers of abstraction

Computer systems are built by accumulating layers of abstraction.

Ex. Running a TOY program.



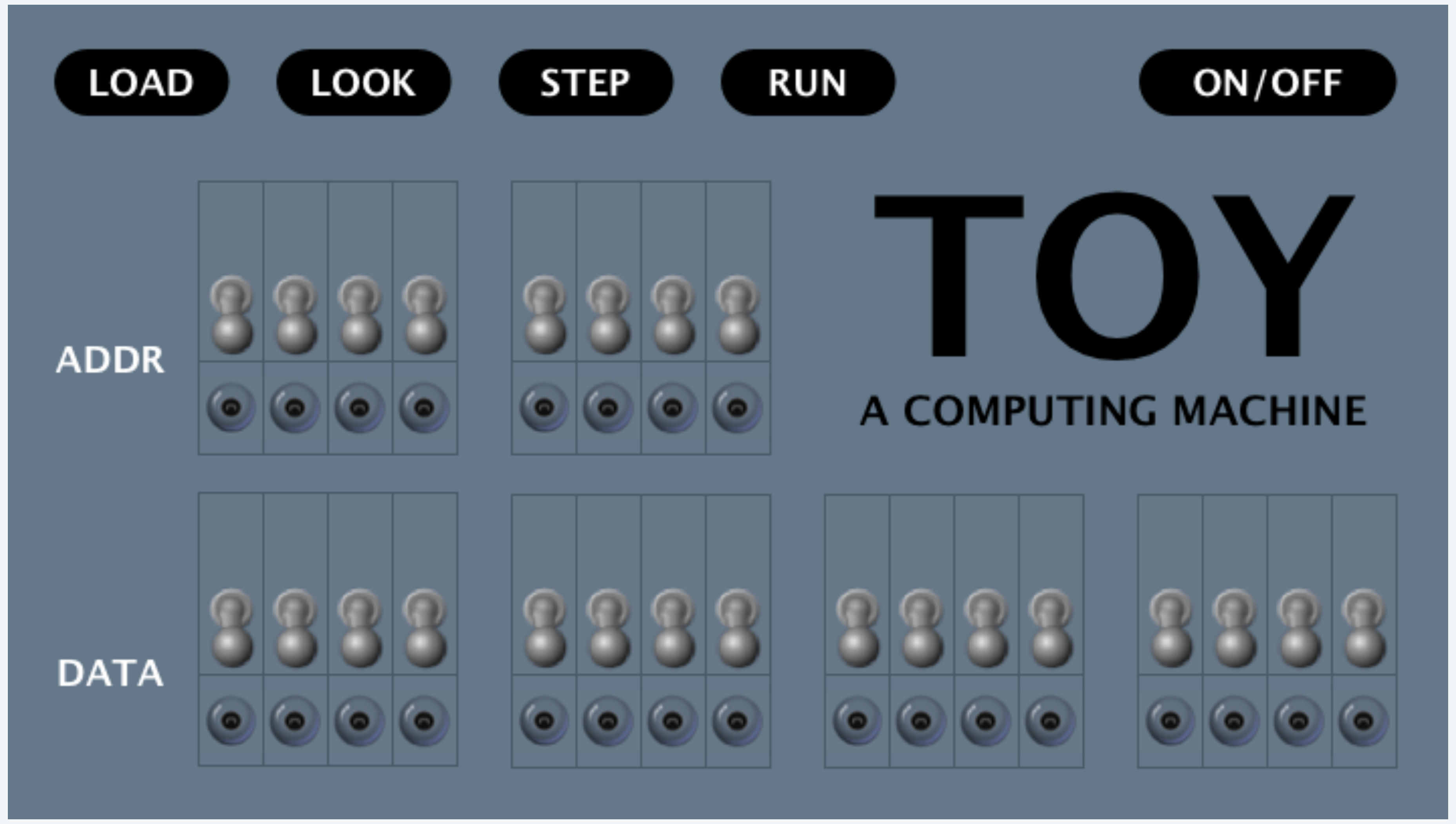
Big ideas

Digital computers. Encode "everything" in binary, including programs and data.

von Neumann machine. Store programs and data in same memory.

Indirection. Manipulate a value through its memory address.

Emulation. Make one system imitate another.



Credits

image	source	license
Microprocessor and Binary	Adobe Stock	education license
David Wheeler	Computer Laboratory, Cambridge	CC BY 2.0
Simply Explained Indirection	Geek and Poke	
J. Presper Eckert	Michael Denning	
John Mauchly	Encyclopædia Britannica	
Programming ENIAC	U.S. Army	public domain
Vacuum Tube	Adobe Stock	education license
John von Neumann	Los Alamos National Labs	LANL

Credits

image	source	license
Apollo 11	NASA	public domain
Integrated Circuit	NASA	public domain
Margaret Hamilton	MIT Museum	
Light Bulb	openclipart.com	CC0 1.0
EDSAC	University of Cambridge	CC BY 2.0
Old Personal Computer	Adobe Stock	education license
Sprint Phone	www.zdnet.com	
Google Data Center	Alphabet / Google	
Pac-Man Arcade Machine	NAMCO	
Pac-Man on Android	Leslie Wong	
Rosetta Stone	Adobe Stock	education license