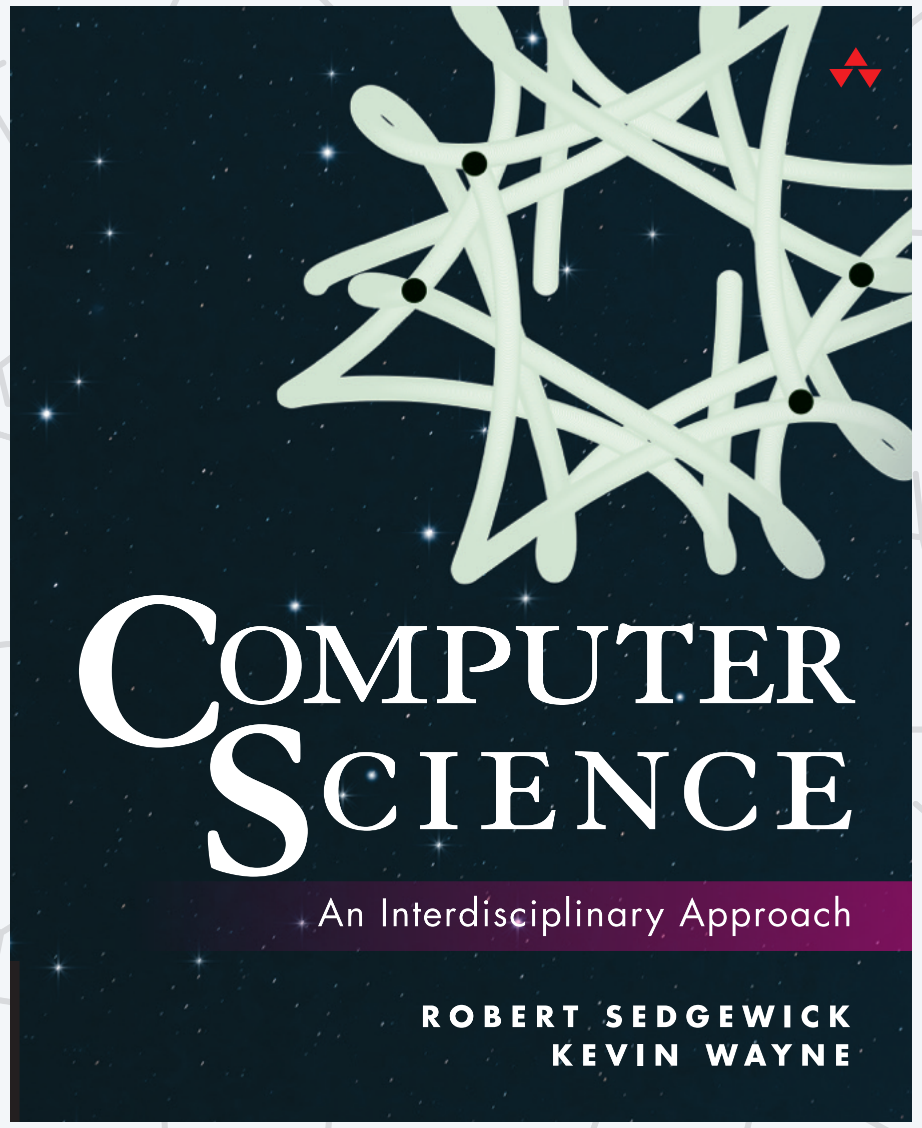


https://introcs.cs.princeton.edu

6. TOY Machine I

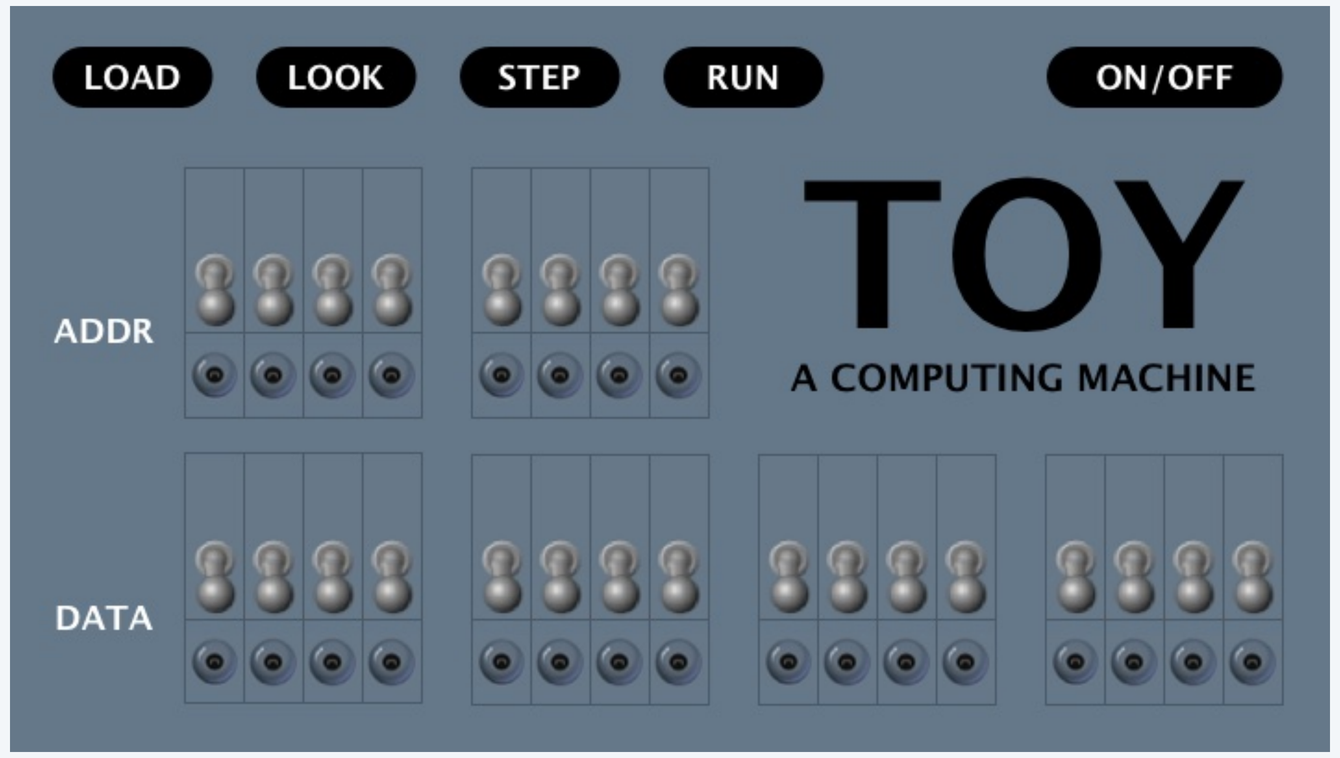
- *overview*
- *data types*
- *instructions*
- *operating the machine*

The TOY computing machine

TOY is an imaginary machine invented for this course.

It is similar in design to:

- Ancient computers.
- Today's microprocessors.
- Countless other devices designed and built over the past 60 years.



TOY machine

PDP-8, 1970s

smartphone processor, 2020s

Reasons to study TOY

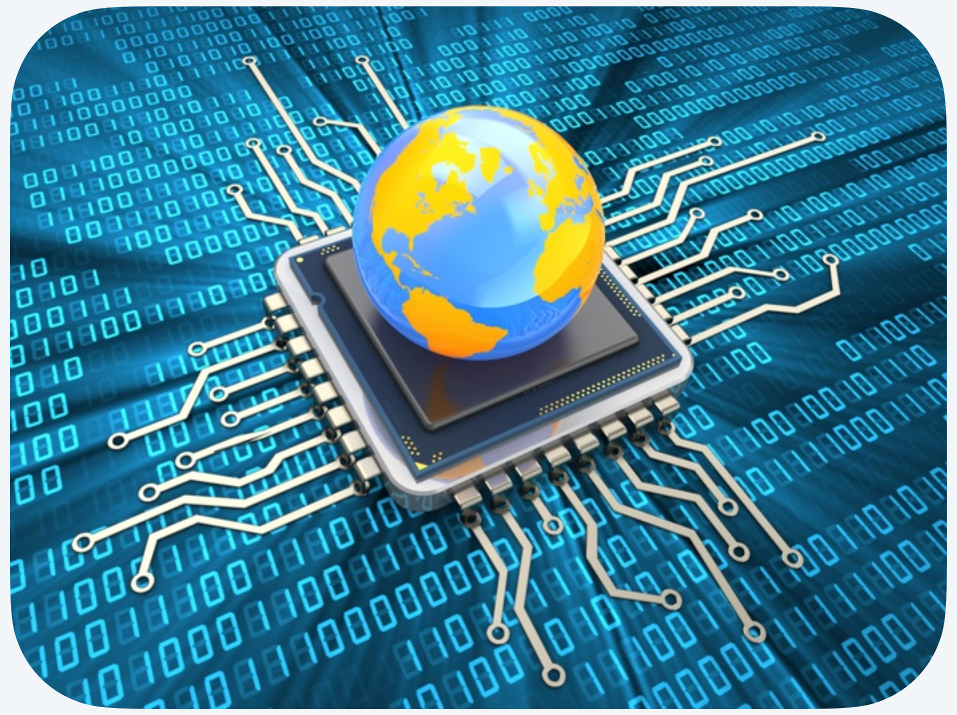
Learn about machine language programming.

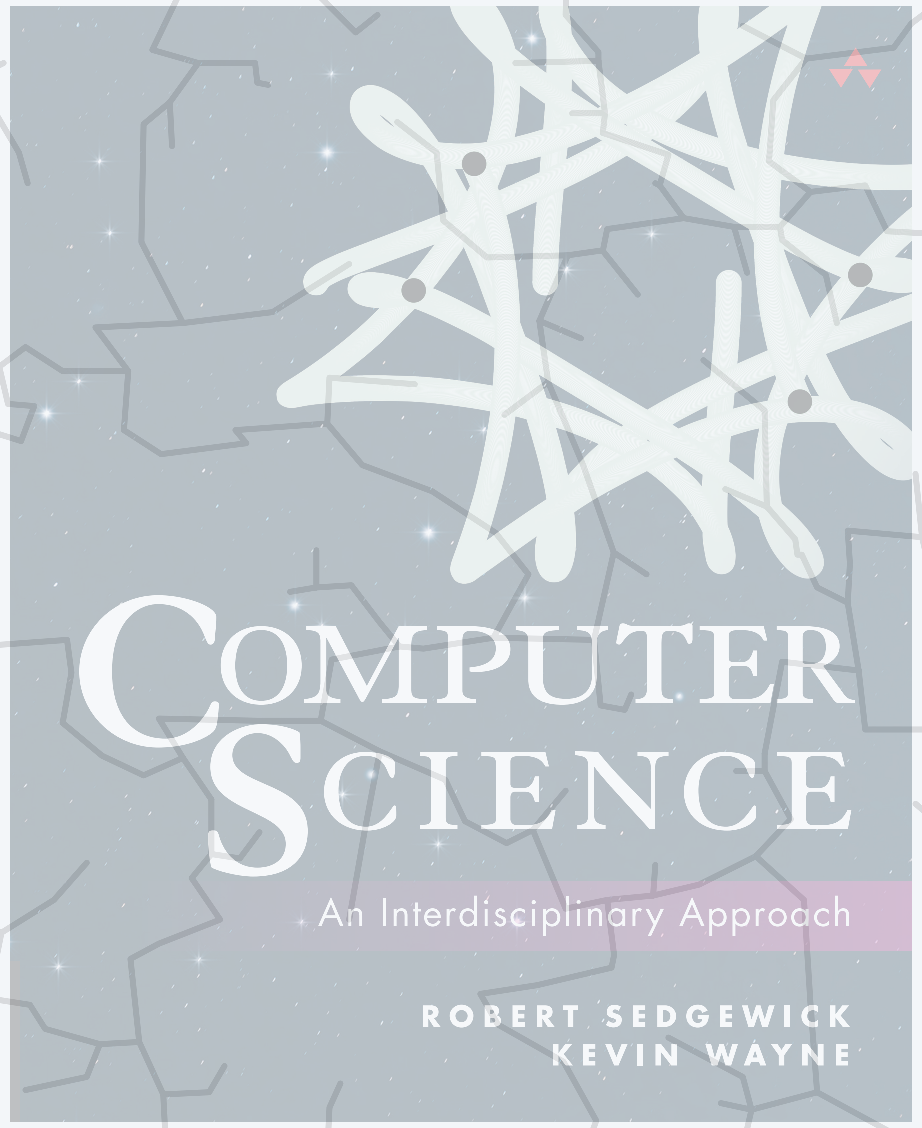
- How do Java programs relate to your computer? ← *see* COS 320
- Key to understanding Java references (and C pointers). ← *see* COS 217
- Still necessary in some modern applications.

multimedia, computer games, embedded devices, scientific computing, …

Prepare to learn about computer architecture. ← *see* COS 375 / ECE 375

- How does your computer's processor work?
- What are its basic components?
- How do they interact?



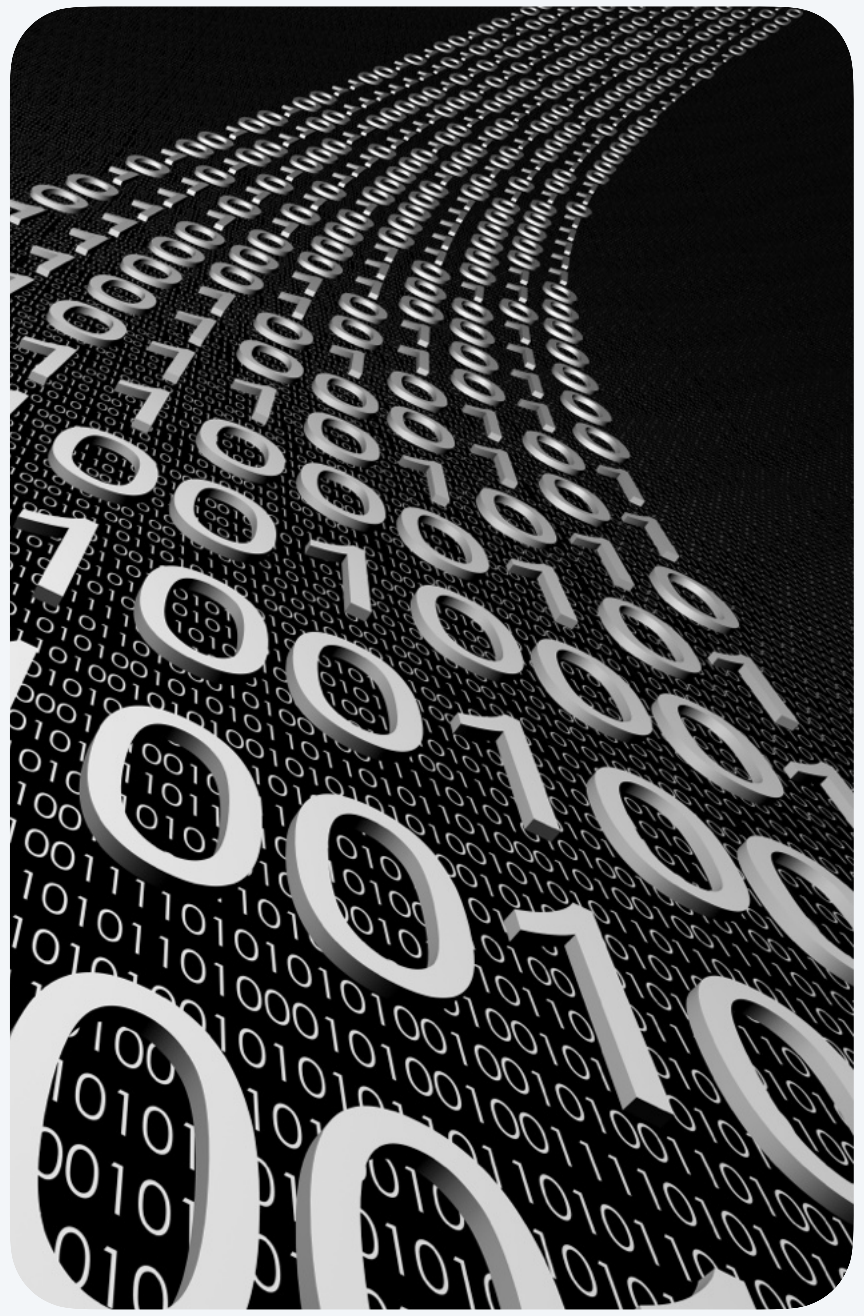


https://introcs.cs.princeton.edu

6. TOY Machine I

- *overview*
- *data types*
- *instructions*
- *operating the machine*

Data and programs are encoded in binary



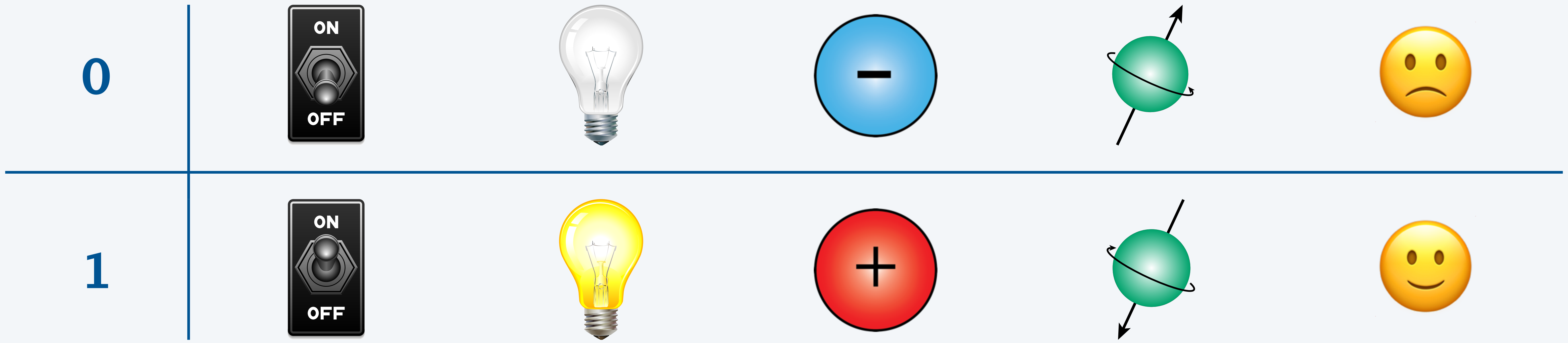
Bit (binary digit). Basic unit of information in computing: either 0 or 1.

Everything stored in a computer is a sequence of bits.

- Data and programs.
- Numbers, text, pictures, songs, movies, biometrics, 3D objects, …

Q. Why binary?

A. Easy to represent two states in physical world.



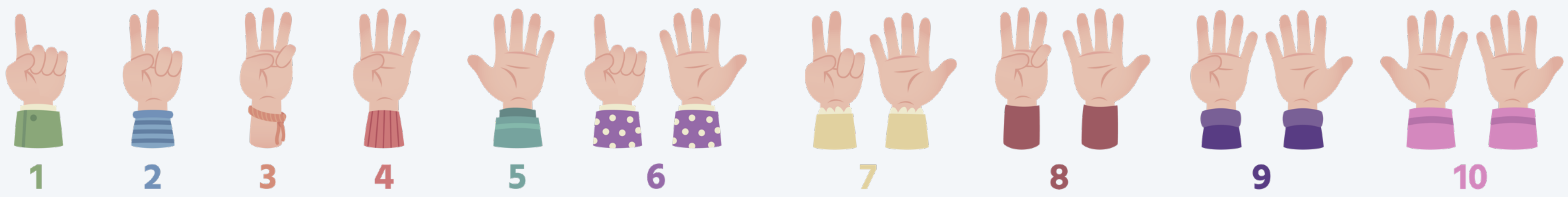
Decimal number system

Decimal number. A number expressed in base 10.

- Place-value notation with ten symbols (0–9).
- Used by most modern cultures.

	3	2	1	0
decimal	6	3	7	5

$6 \cdot 10^3 + 3 \cdot 10^2 + 7 \cdot 10^1 + 5 \cdot 10^0 = 6375_{10}$

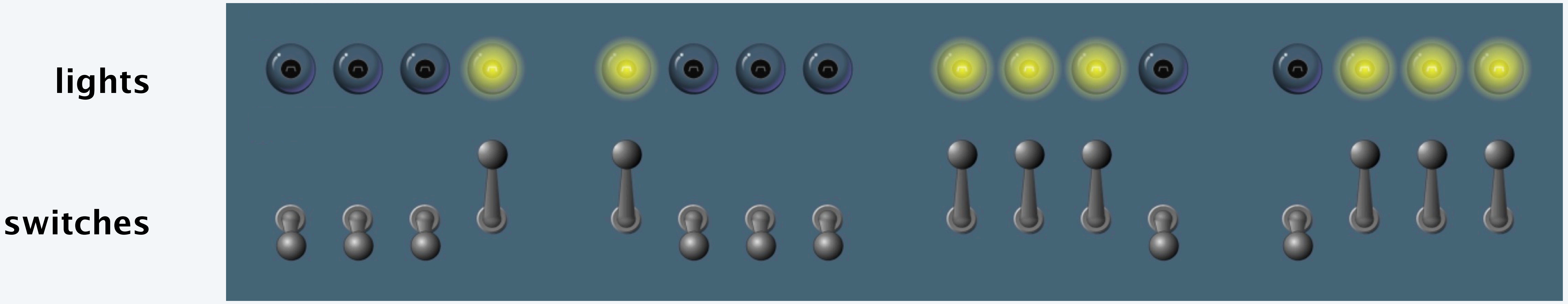


decimal
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Binary number system

Binary number. A number expressed in base 2.

- Place-value notation with two symbols (0 and 1).
- Used by all modern computers.



	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
binary	0	0	0	1	1	0	0	0	1	1	1	0	0	1	1	1

$$2^{12} + 2^{11} + 2^7 + 2^6 + 2^5 + 2^2 + 2^1 + 2^0 = 6375_{10}$$

decimal	binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Binary t-shirt

Hexadecimal number system

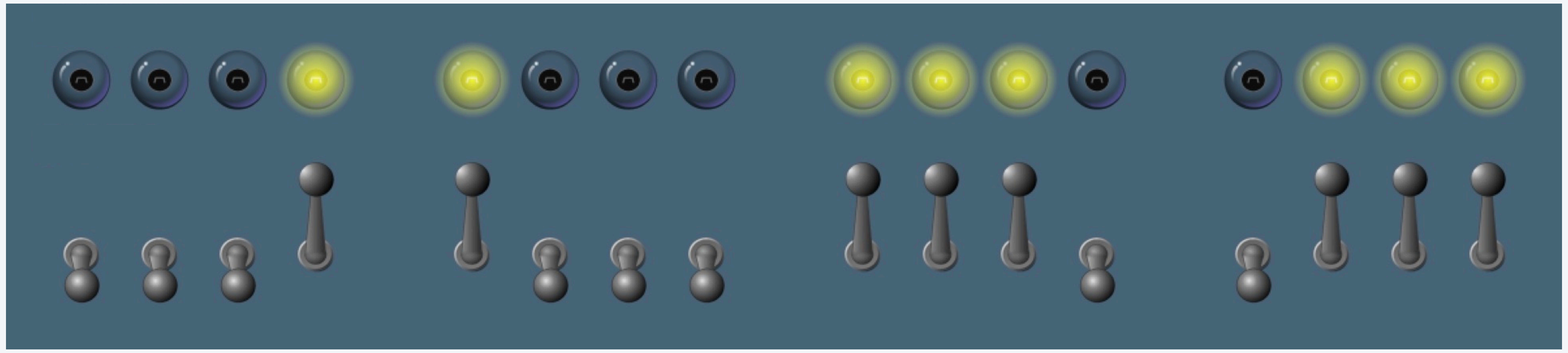
Hexadecimal number. A number expressed in base 16.

- Place-value notation with 16 symbols (0–9, A–F).
- Easy to convert from binary to hex (and vice versa). ← *4 bits per hex digit (because $2^4 = 16$)*
- More convenient for programmers.

decimal	binary	hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

lights

switches



	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
binary	0	0	0	1	1	0	0	0	1	1	1	0	0	1	1	1
hex	1				8				E				7			

$1 \cdot 16^3 \quad 8 \cdot 16^2 \quad 14 \cdot 16^1 \quad 7 \cdot 16^0 \quad = 6375_{10}$

What is 1010111010110 1 in hexadecimal?

A. 2BAD

B. AEB1

C. EBAD

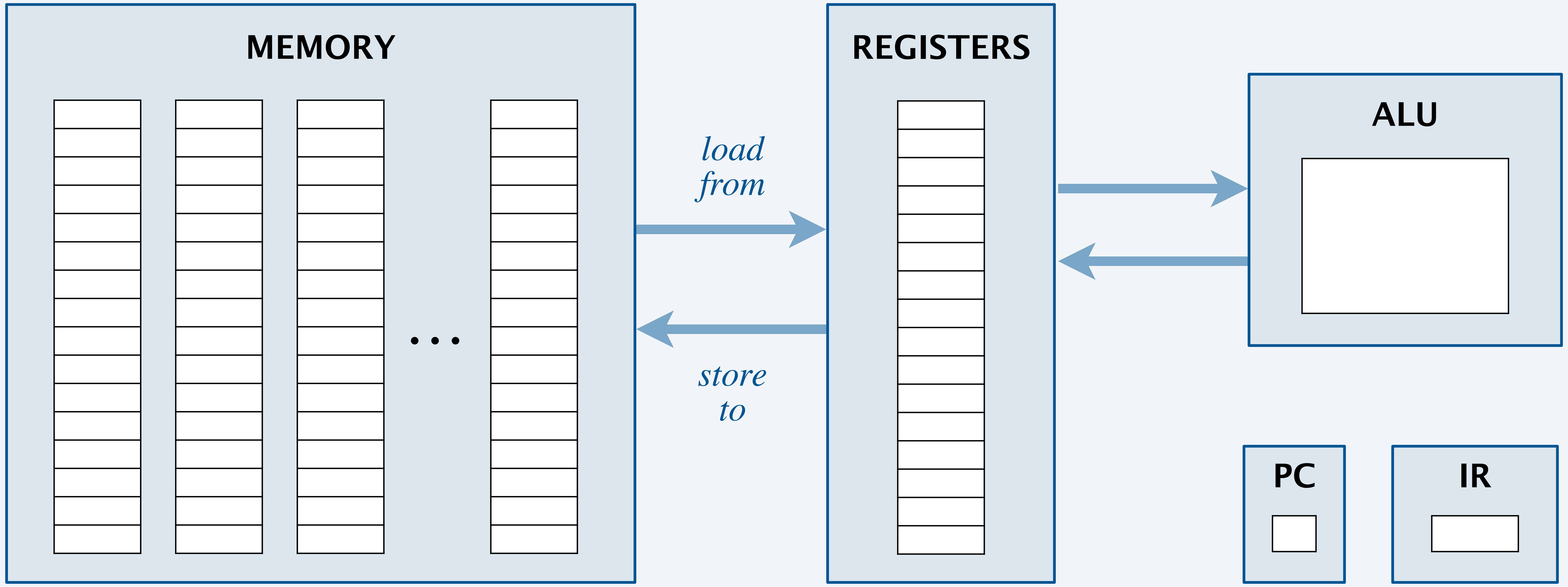
D. DAB2

decimal	binary	hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Inside the box

TOY machine components.

- 256 memory cells.
- 16 registers.
- 1 arithmetic logic unit (ALU).
- 1 program counter (PC).
- 1 instruction register (IR).



Memory

Memory.

- Holds data and instructions.
- 256 words of memory.
- 16 bits per word.

Memory is addressable.

- Specify individual word using array notation.
- Use hexadecimal for addresses: 00 to FF.
- Ex: `M[F2] = C0DE`.

start thinking in hexadecimal

MEMORY

00	0000	10	8A16		F0	CAFE
01	FFFE	11	8B16		F1	ABBA
02	000D	12	1CAB		F2	C0DE
03	0003	13	9C17		F3	8BAD
04	0001	14	0000		F4	F00D
05	0000	15	0008		F5	FACE
06	0000	16	0005		F6	1377
07	0000	17	000D		F7	D1CE
08	0000	18	0000	…	F8	C1A0
09	0000	19	0000		F9	DEAD
0A	0000	1A	C024		FA	BEEF
0B	0000	1B	0000		FB	1DEA
0C	0000	1C	0000		FC	A1DE
0D	0000	1D	0000		FD	0B0E
0E	0000	1E	0000		FE	BEAD
0F	0000	1F	0000		FF	0000

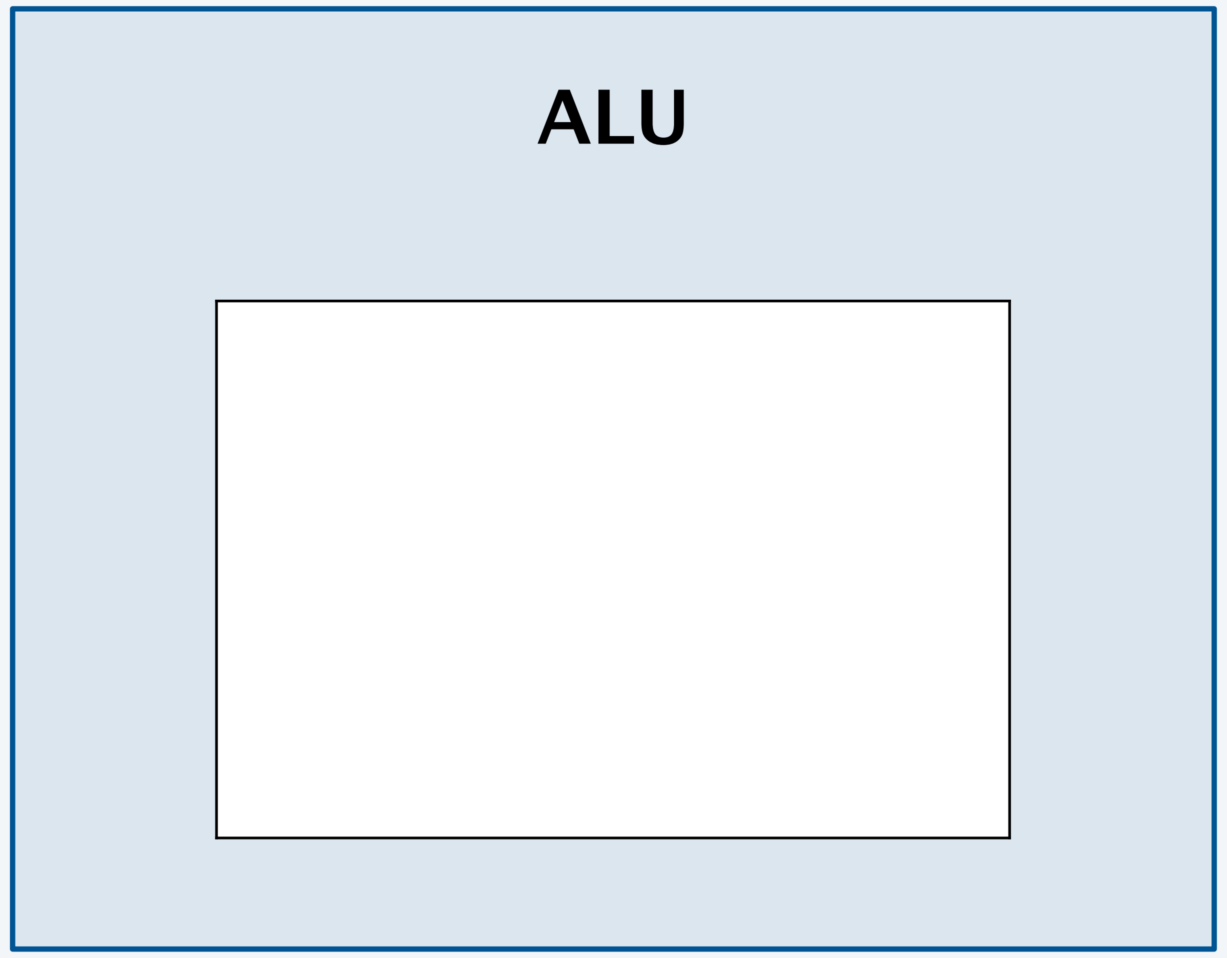
one word of memory (C0DE at F2)

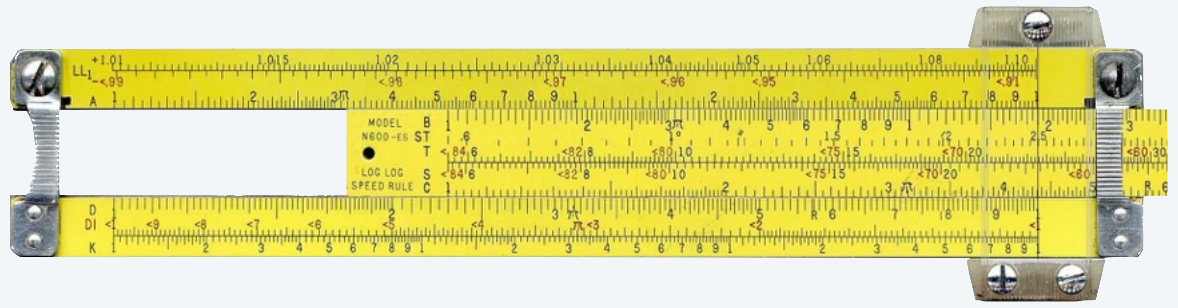
8-bit address — *16-bit word*

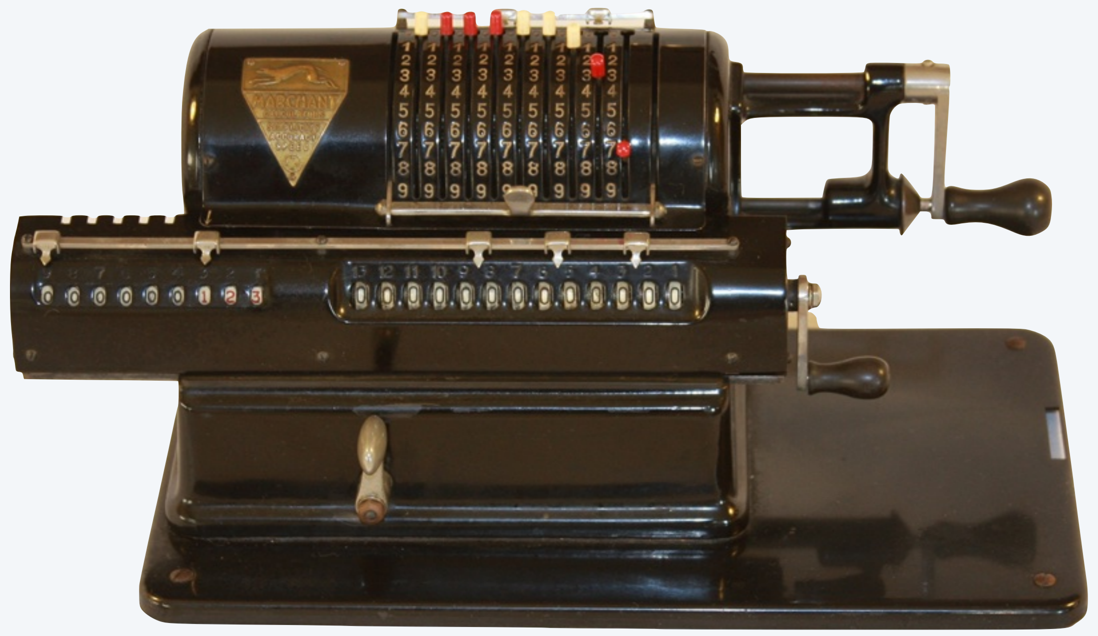
Arithmetic logic unit

Arithmetic logic unit (ALU).

- TOY's computational engine.
- A calculator, not a computer.
- Hardware that implements *all* data-type operations (e.g., add and subtract).







Registers

Registers.

- Scratch space for calculations and data movement.
- 16 registers, each storing one 16-bit word.
- Addressable as R[0] through R[F].
- R[0] always stores 0000.

Q. What's the difference between registers and main memory?

A. Registers are connected directly with ALU.

  – faster than main memory

  – more expensive than main memory

REGISTERS

0	0 0 0 0
1	0 0 0 1
2	0 0 0 2
3	C A F E
4	C 0 D E
5	0 0 0 0
6	F A C E
7	0 0 0 0
8	F 0 0 D
9	0 0 0 0
A	0 0 0 8
B	0 0 0 5
C	0 0 0 D
D	0 0 0 0
E	0 0 0 0
F	0 0 0 0

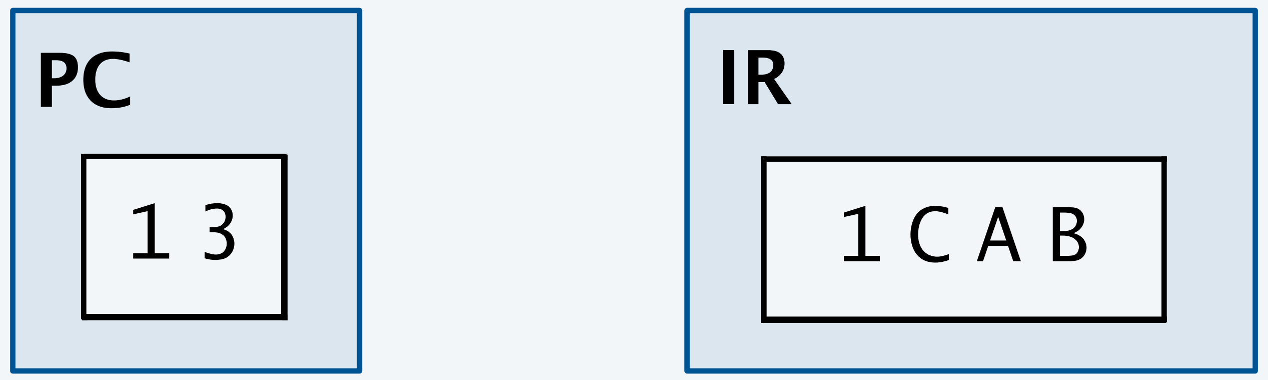
16-bit word

Control

TOY operates by executing a sequence of instructions.

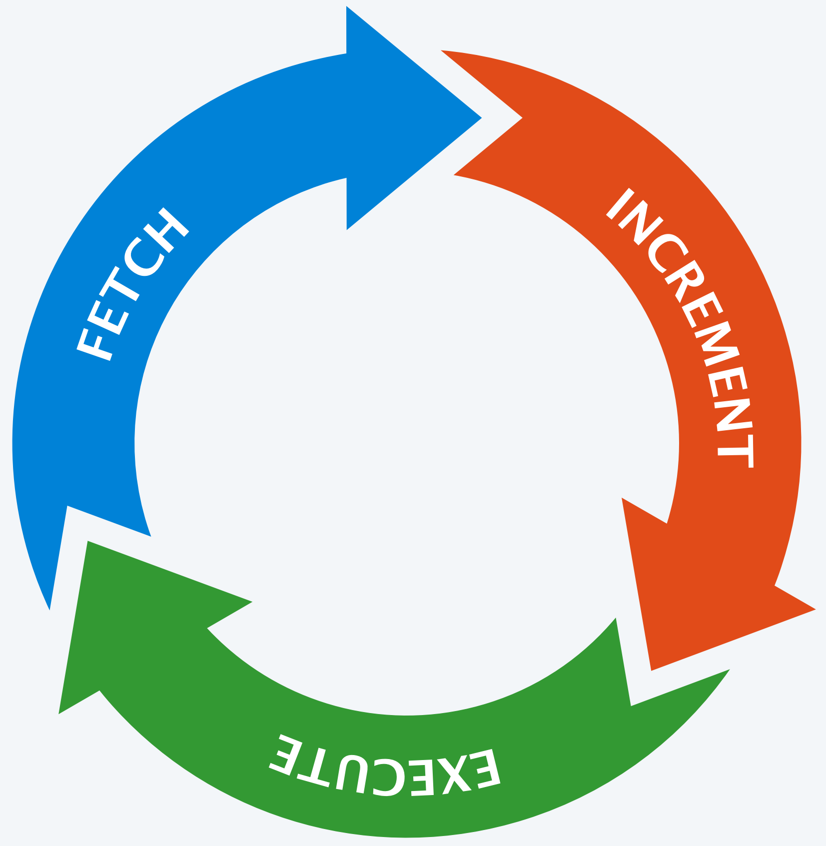
Program counter (PC). Stores memory address (8 bits) of *next* instruction to be executed.

Instruction register (IR). Stores instruction (16 bits) being executed.



Fetch–increment–execute cycle.

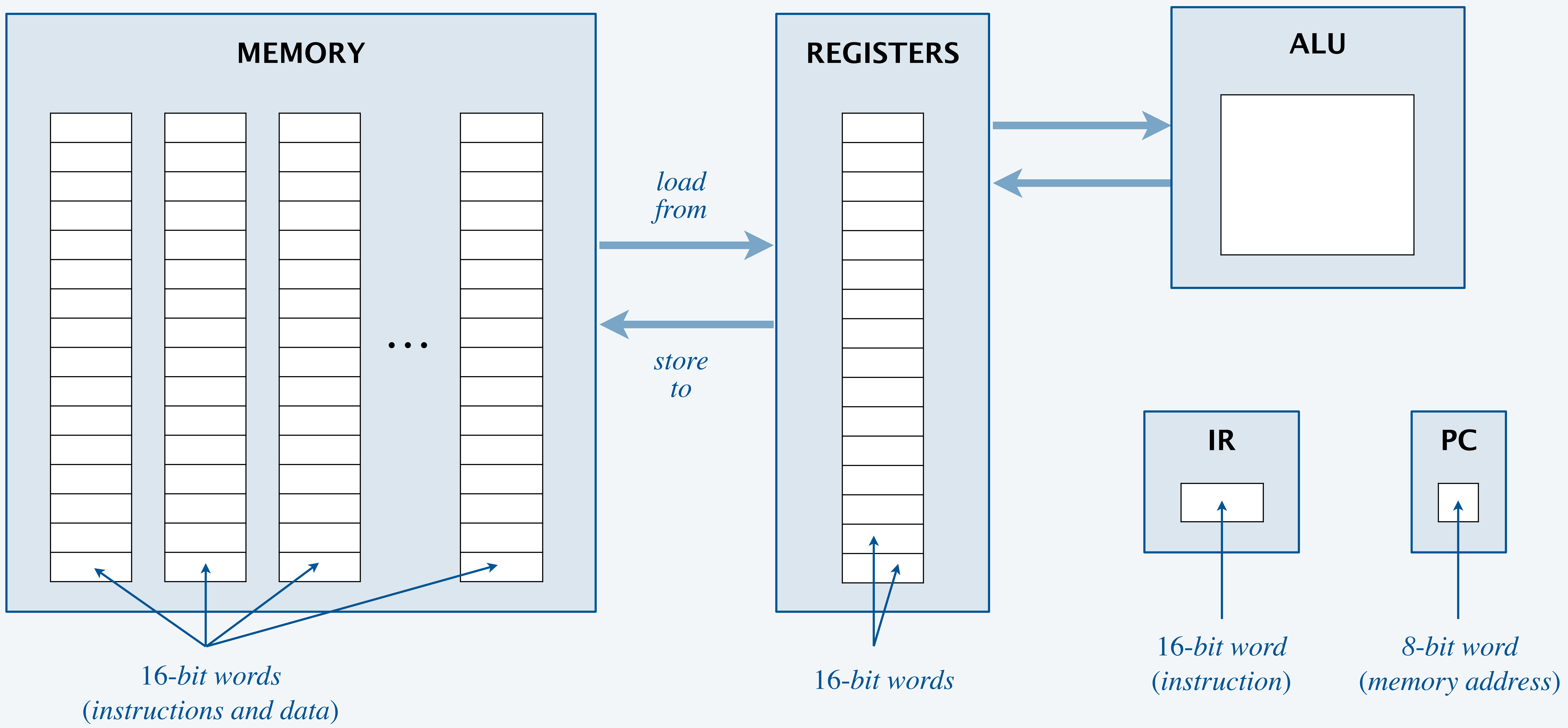
- Fetch: Get instruction (indexed by PC) from memory and store in IR.
- Increment: Update PC to point to next instruction.
- Execute: Move data to (or from) memory; change PC; or perform calculations.



The state of the machine

Contents of memory, registers, and PC at a particular time.

- Provide a record of what a program has done.
- Completely determines the machine will do.



Approximate how many bytes of main memory does TOY machine have?

A. 250 bytes

B. 500 bytes

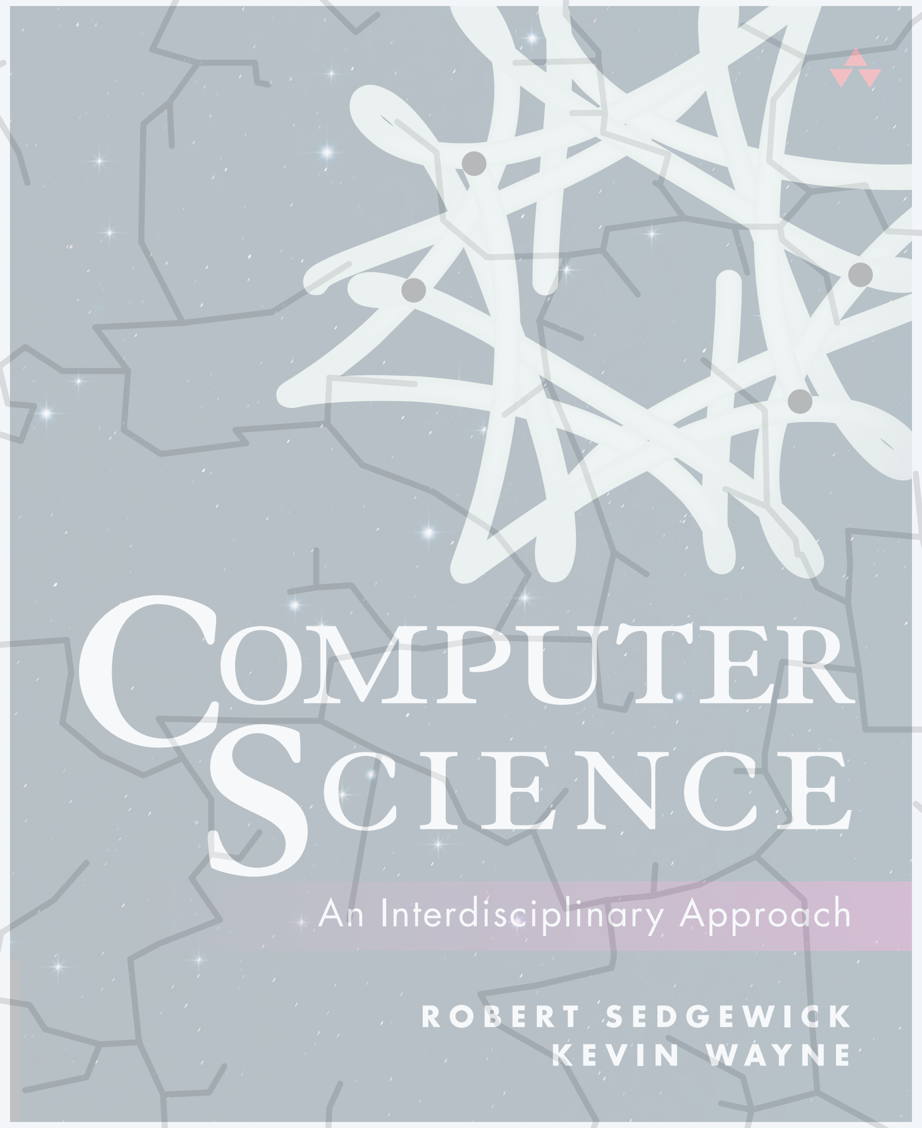
C. 4000 bytes

D. 250 MB

E. 500 GB

term	symbol	quantity
bit	b	1 bit
byte	B	8 bits
kilobyte	KB	1000 bytes
megabyte	MB	1000^2 bytes
gigabyte	GB	1000^3 bytes
terabyte	TB	1000^4 bytes
⋮	⋮	⋮

some define using powers of 2
(MB = 2^{10} *bytes*)



https://introcs.cs.princeton.edu

6. TOY Machine I

- overview
- data types
- instructions
- operating the machine

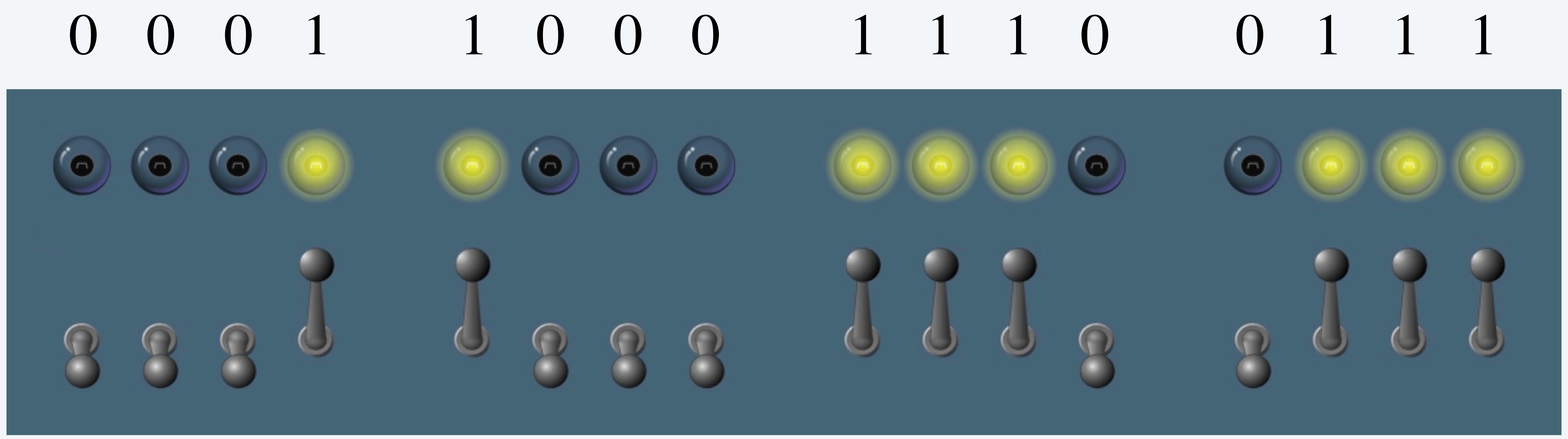
TOY data type

A data type is a set of values and a set of operations on those values.

TOY's data type.

- Value: 16-bit two's complement integer.
- Operations: arithmetic (add, subtract) and bitwise (AND, XOR, shift).

Representation. Each value is represented using one 16-bit word.



Note. All other types of data must be implemented with software. ← *32-bit integers, floating-point numbers, booleans, characters, strings, …*

Unsigned integers (16 bit)

Values. Integers 0 to $2^{16} - 1$. ← *only non-negative integers*

Operations.

- Arithmetic: add, subtract.
- Bitwise: AND, XOR, left shift, right shift.

Representation. 16 bits.

decimal	hex	binary
0	0000	0000000000000000
1	0001	0000000000000001
2	0002	0000000000000010
3	0003	0000000000000011
4	0004	0000000000000100
⋮	⋮	⋮
65,533	FFFD	1111111111111101
65,534	FFFE	1111111111111110
65,535	FFFF	1111111111111111

largest integer $(2^{16} - 1)$

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
binary	0	0	0	1	1	0	0	0	1	1	1	0	0	1	1	1

$2^{12} + 2^{11} + 2^7 + 2^6 + 2^5 + 2^2 + 2^1 + 2^0 = 6375_{10}$

hex	1	8	E	7

$1 \times 16^3 + 8 \times 16^2 + 14 \times 16^1 + 7 \times 16^0 = 6375_{10}$

Signed integers (16-bit two's complement)

Values. Integers -2^{15} to $2^{15} - 1$. ← *includes negative integers!*

Operations.

- Arithmetic: add, subtract.
- Bitwise: AND, XOR, left shift, right shift.
- Comparison: test if positive, test if zero.

Representation. 16-bit two's complement.

- For $0 \leq x < 2^{15}$, 16-bit unsigned representation of x.
- For $-2^{15} \leq x < 0$, 16-bit unsigned representation of $2^{16} - |x|$.

"complement" of 2^{16}

decimal	hex	binary
–32,768	8000	1000000000000000
–32,767	8001	1000000000000001
–32,766	8002	1000000000000010
⋮	⋮	⋮
–3	FFFD	1111111111111101
–2	FFFE	1111111111111110
–1	FFFF	1111111111111111
0	0000	0000000000000000
+1	0001	0000000000000001
+2	0002	0000000000000010
+3	0003	0000000000000011
⋮	⋮	⋮
+32,765	7FFD	0111111111111101
+32,766	7FFE	0111111111111110
+32,767	7FFF	0111111111111111

smallest integer (-2^{15})

representation for 0 is 0000000000000000

largest integer ($2^{15} - 1$)

sign bit

Calculations with two's complement integers

Addition. To compute $x + y$:

- Add as unsigned integers.
- Ignore any overflow.

	1111111110000010	-126_{10}
+	0000001111101000	$1{,}000_{10}$
	0000001101101010	874_{10}

ignore carry bit out sign bit

Negation. To convert from x to $-x$ (or vice versa):

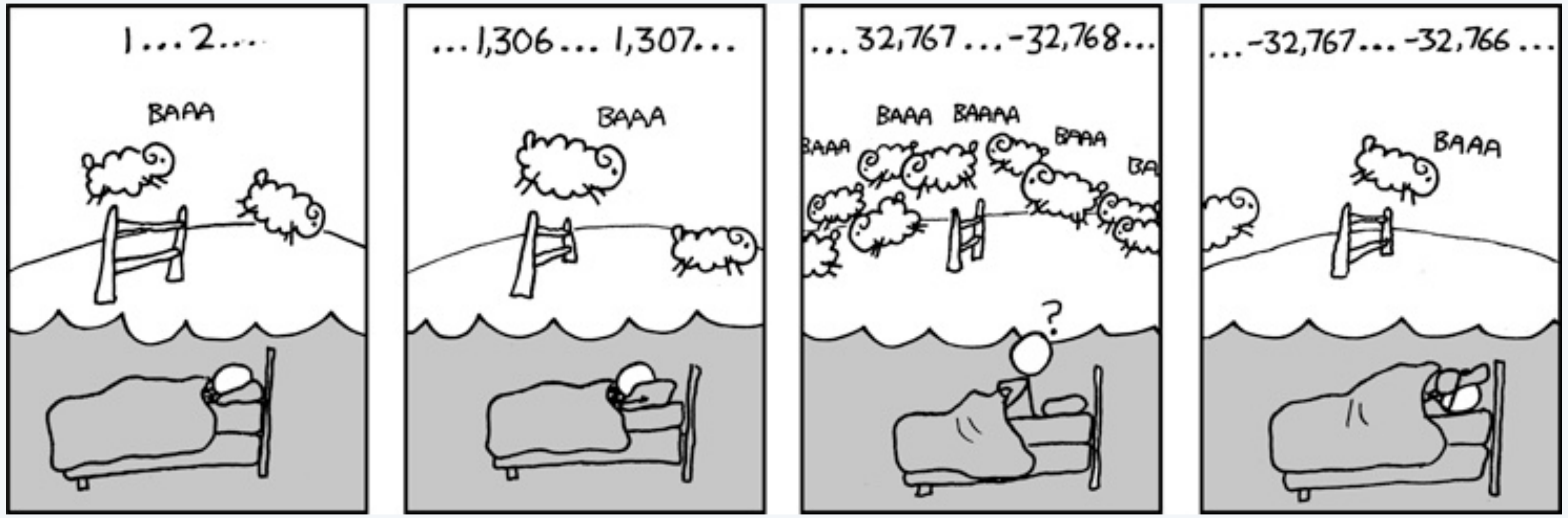
- Flip all bits.
- Add 1.

	0000000001111110	126_{10}
	1111111110000001	*flip bits*
+	0000000000000001	*add* 1
	1111111110000010	-126_{10}

Overflow with two's complement integers

Integer overflow. Result of arithmetic operation is outside prescribed range (too large or small).

```
  0111111111111111     32,767₁₀   ← largest integer (2^15 – 1)
+ 0000000000000001          1₁₀
  ----------------
  1000000000000000    –32,768₁₀   ← smallest integer (–2^15)
  ↑
  overflow (carry into sign bit)
```



https://xkcd.com/571

TOY data type: bitwise operations

Bitwise AND. Apply *and* operation to corresponding bits.

	0	1	0	1	1	0	0	1	0	1	0	0	1	0	0	0
&	0	0	0	1	1	1	1	1	0	0	0	0	0	1	0	1
	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0

x	y	x & y
0	0	0
0	1	0
1	0	0
1	1	1

AND

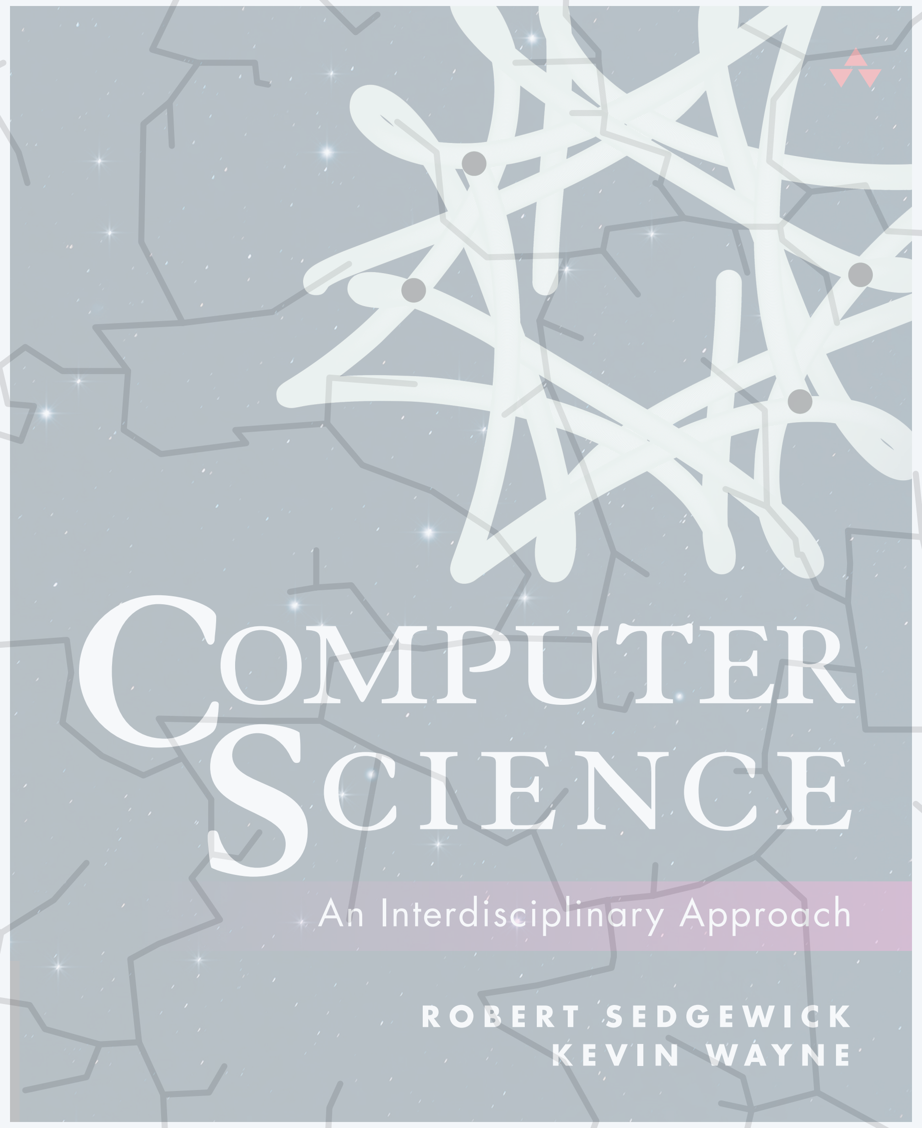
Bitwise XOR. Apply *xor* operation to corresponding bits.

	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
^	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

x	y	x ^ y
0	0	0
0	1	1
1	0	1
1	1	0

XOR

```
~/toy/toy1> jshell
jshell> int a = 3 ^ 5;
a ==> 6
```



https://introcs.cs.princeton.edu

6. TOY Machine I

- overview
- data types
- instructions
- operating the machine

TOY instructions: halt

TOY program. A TOY program is a sequence of TOY instructions.

Instructions. Any 16-bit value can be interpreted as a TOY instruction.

Halt. Stop executing the program.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0				0				0				0			
opcode (halt)				*destination d*				*source s*				*source t*			

TOY instructions: add

TOY program. A TOY program is a sequence of TOY instructions.

Instructions. Any 16-bit value can be interpreted as a TOY instruction.

Add. Add two 16-bit integers from registers and store the sum in a register.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	0	1	0	1	0	1	0	1	1
1				C				A				B			
opcode (add)				*destination d*				*source s*				*source t*			

Pseudocode. `R[C] = R[A] + R[B]` ← *add* R[A] *and* R[B]; *put result in* R[C]

Registers

R[0]	0 0 0 0
R[1]	0 0 0 1
R[2]	0 0 1 0
R[3]	C A F E
R[4]	0 0 0 1
R[5]	0 0 0 0
R[6]	C 0 D E
R[7]	0 0 0 0
R[8]	F 0 0 D
R[9]	0 0 0 0
R[A]	0 0 0 8
R[B]	0 0 0 5
R[C]	0 0 0 D
R[D]	0 0 0 0
R[E]	0 0 0 0
R[F]	0 0 0 0

TOY instructions: load and store

TOY program. A TOY program is a sequence of TOY instructions.

Instructions. Any 16-bit value can be interpreted as a TOY instruction.

Load. Copy a 16-bit integer from a memory cell to a register.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	0	0	1	0	1	0	1
8				A				1				5			
opcode (load)				*destination d*				*addresss*							

load data from `M[15]` *into* `R[A]`

`R[A] = M[15]`

Store. Copy a 16-bit integer from a register to a memory cell.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	1	0	0	0	0	0	1	0	1	1	1
9				C				1				7			
opcode (store)				*destination d*				*addresss*							

store contents of `R[C]` *into* `M[17]`

`M[17] = R[C]`

Your first TOY program

Add two integers.

- Load operands from memory into two registers.
- Add the 16-bit integers in 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

Your first TOY program (with different data)

Q. How can you tell whether a word is an instruction or data?

A. If the PC has its address, it is an instruction.

instruction

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:	8A15	input 1
16:	1CAB	input 2
17:	0000	output
⋮	⋮	

data

REGISTERS

⋮	⋮
R[A]	0000
R[B]	0000
R[C]	0000
⋮	⋮

PC

10

Instruction set

Instruction set. Complete list of machine instructions.

- First hex digit (opcode) specifies which instruction.
- Each instruction changes machine in well-defined way.

category	opcodes	implements	changes
arithmetic and logic operations	1 2 3 4 5 6	data-type operations	registers
data movement	7 8 9 A B	data moves between registers and memory	registers, memory
flow of control	0 C D E F	conditionals, loops, and functions	program counter

opcode	instruction
0	*halt*
1	*add*
2	*subtract*
3	*bitwise and*
4	*bitwise xor*
5	*shift left*
6	*shift right*
7	*load address*
8	*load*
9	*store*
A	*load indirect*
B	*store indirect*
C	*branch if zero*
D	*branch if positive*
E	*jump register*
F	*jump and link*

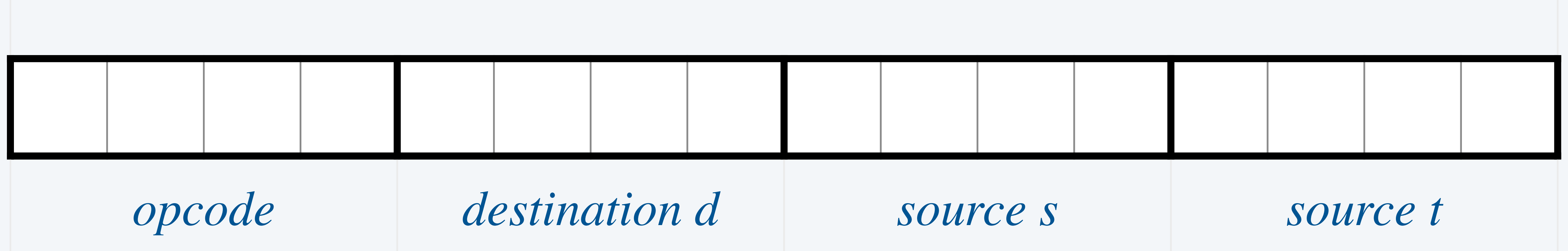
Instruction set

Instruction set. Complete list of machine instructions.

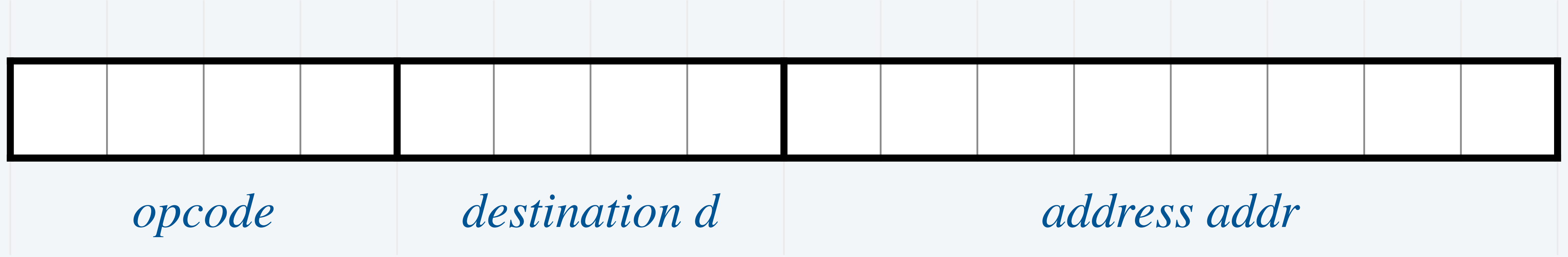
- First hex digit (opcode) specifies which instruction.
- Each instruction changes machine in well-defined way.

Instruction formats. How to interpret a 16-bit instruction?

- Format *RR*: opcode and three registers.



- Format *A*: opcode, one register, and one memory address.



opcode	format	instruction
0	—	*halt*
1	*RR*	*add*
2	*RR*	*subtract*
3	*RR*	*bitwise and*
4	*RR*	*bitwise xor*
5	*RR*	*shift left*
6	*RR*	*shift right*
7	*A*	*load address*
8	*A*	*load*
9	*A*	*store*
A	*RR*	*load indirect*
B	*RR*	*store indirect*
C	*A*	*branch if zero*
D	*A*	*branch if positive*
E	*RR*	*jump register*
F	*A*	*jump and link*

Which instruction copies the values in R[A] to R[B] ?

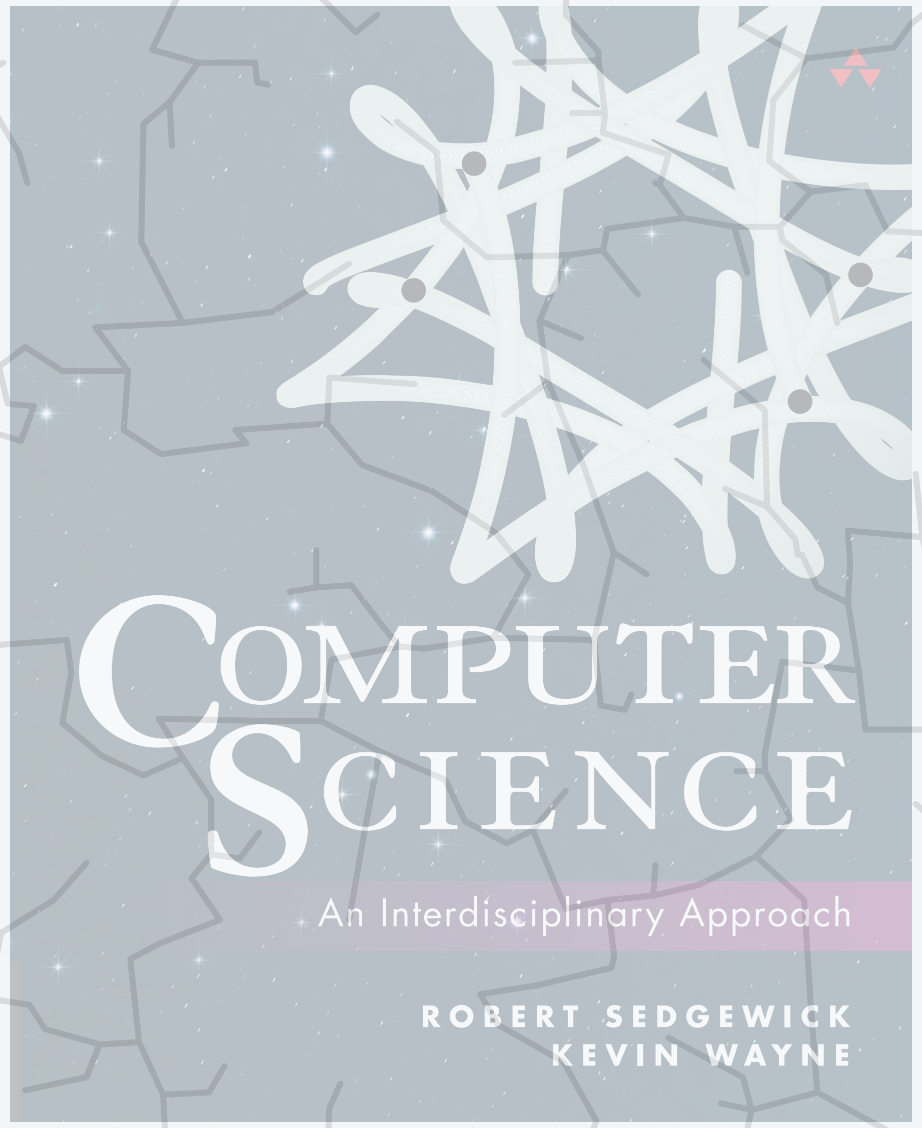
A. 1BA0 R[B] = R[A] + R[0]

B. 2BA0 R[B] = R[A] - R[0]

← R[0] *is always* 0000

C. 3BAA R[B] = R[A] & R[A]

D. All of the above.



https://introcs.cs.princeton.edu

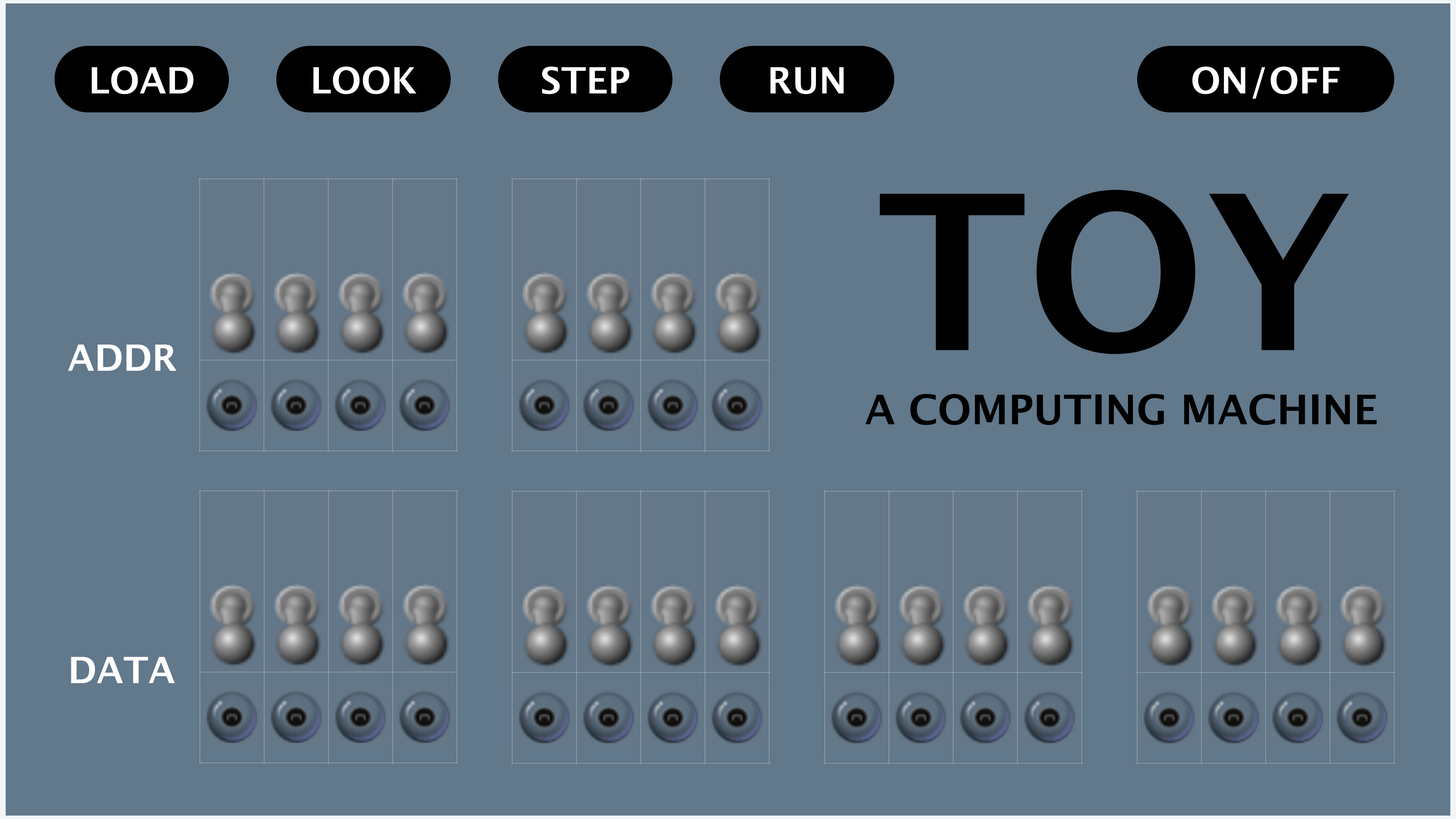
6. TOY MACHINE I

- *overview*
- *data types*
- *instructions*
- *operating the machine*

Outside the box

User interface

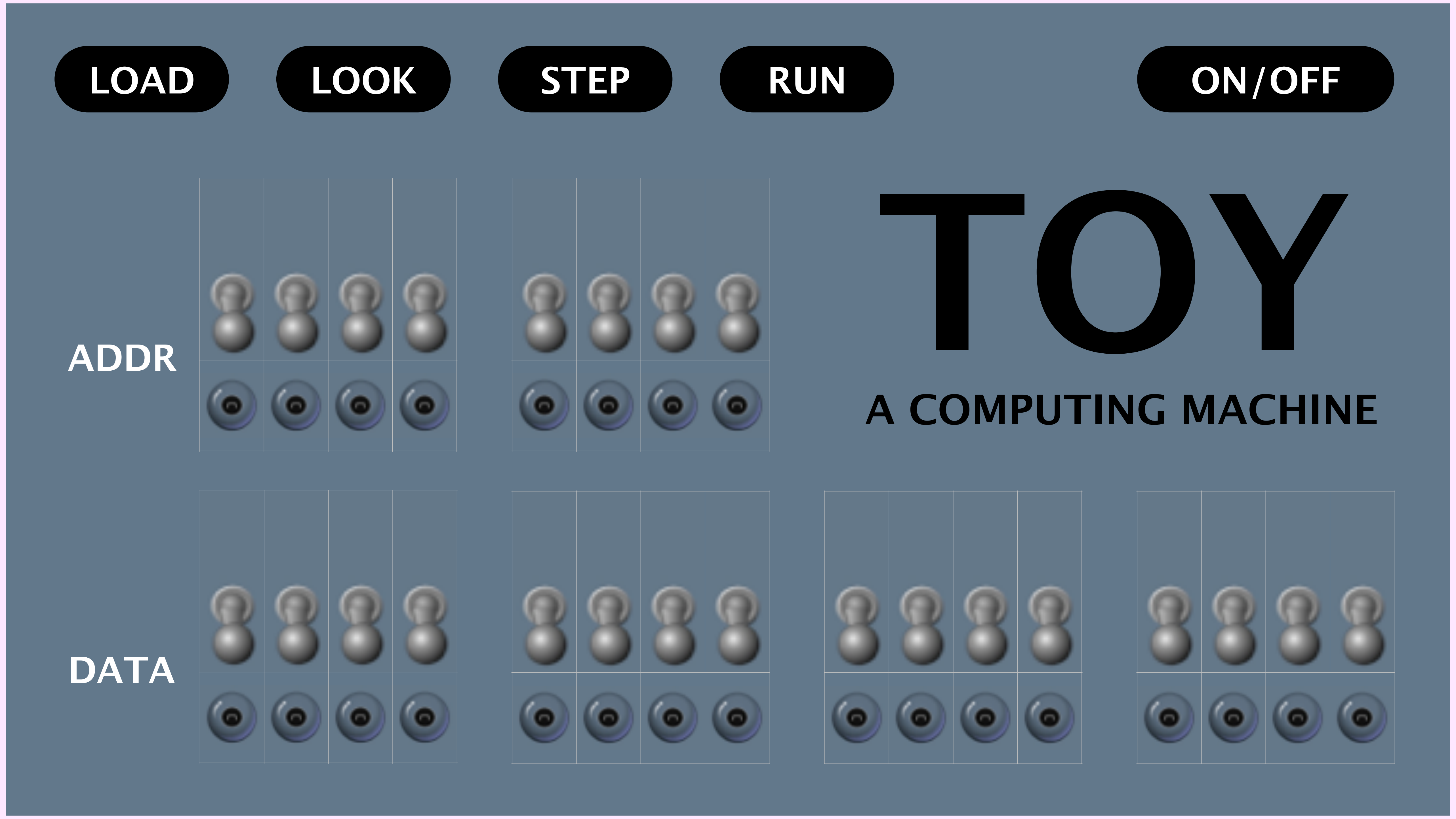
- Switches.
- Lights.
- Control buttons.



TOY machine demo

To load an instruction or data into memory:

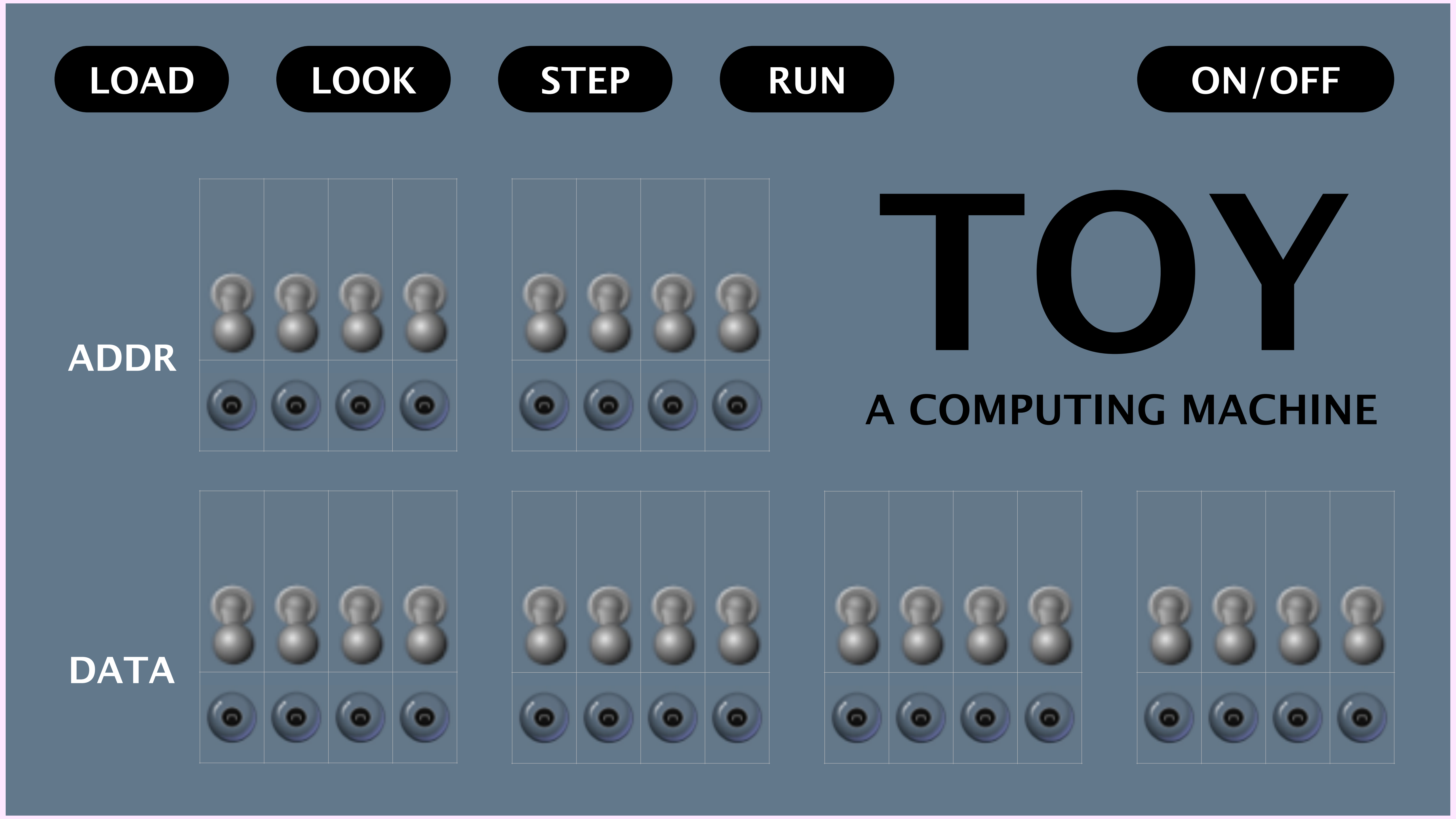
- Set the 8 memory address switches.
- Set the 16 data switches.
- Press LOAD.



TOY machine demo

To view the data in memory:

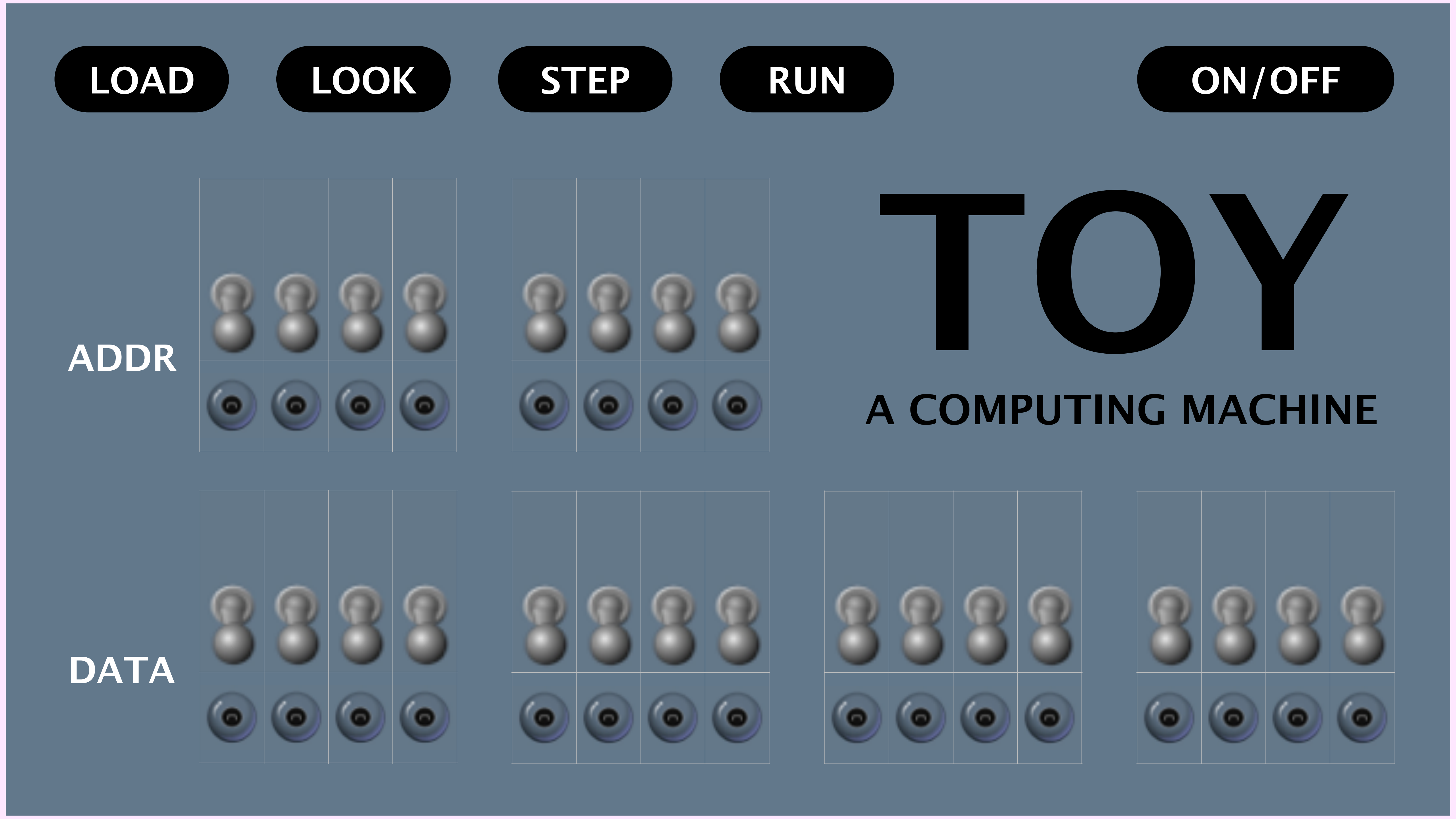
- Set the 8 address switches.
- Press LOOK.



TOY machine demo

To run a program:

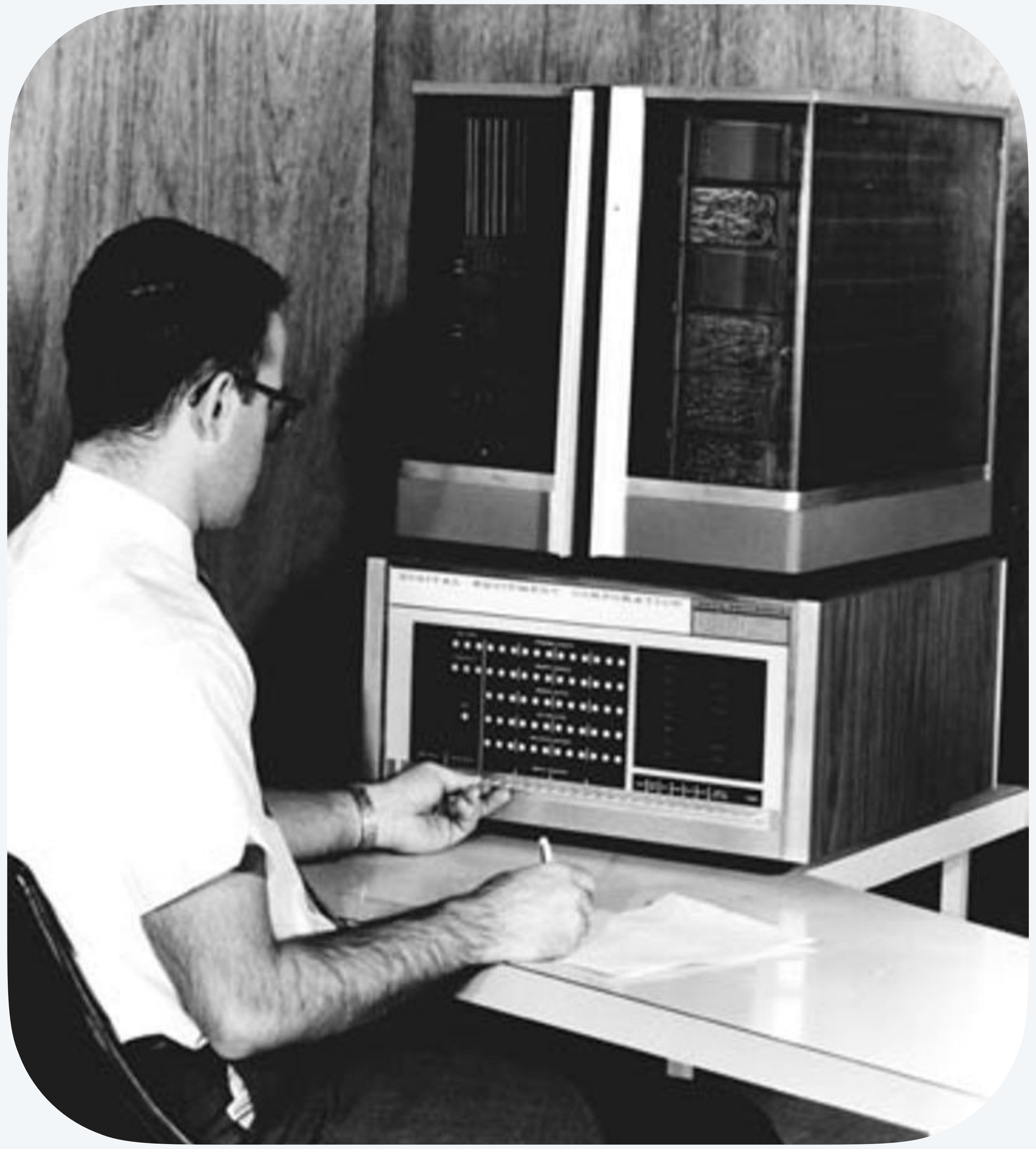
- Set the 8 address switches to the address of first instruction.
- Press RUN.



Switches and lights

Q. Did people really program this way?

A. Yes! We have it good.



DEC PDP-8 (1964)

TOY summary

TOY machine has same basic architecture as modern CPUs:

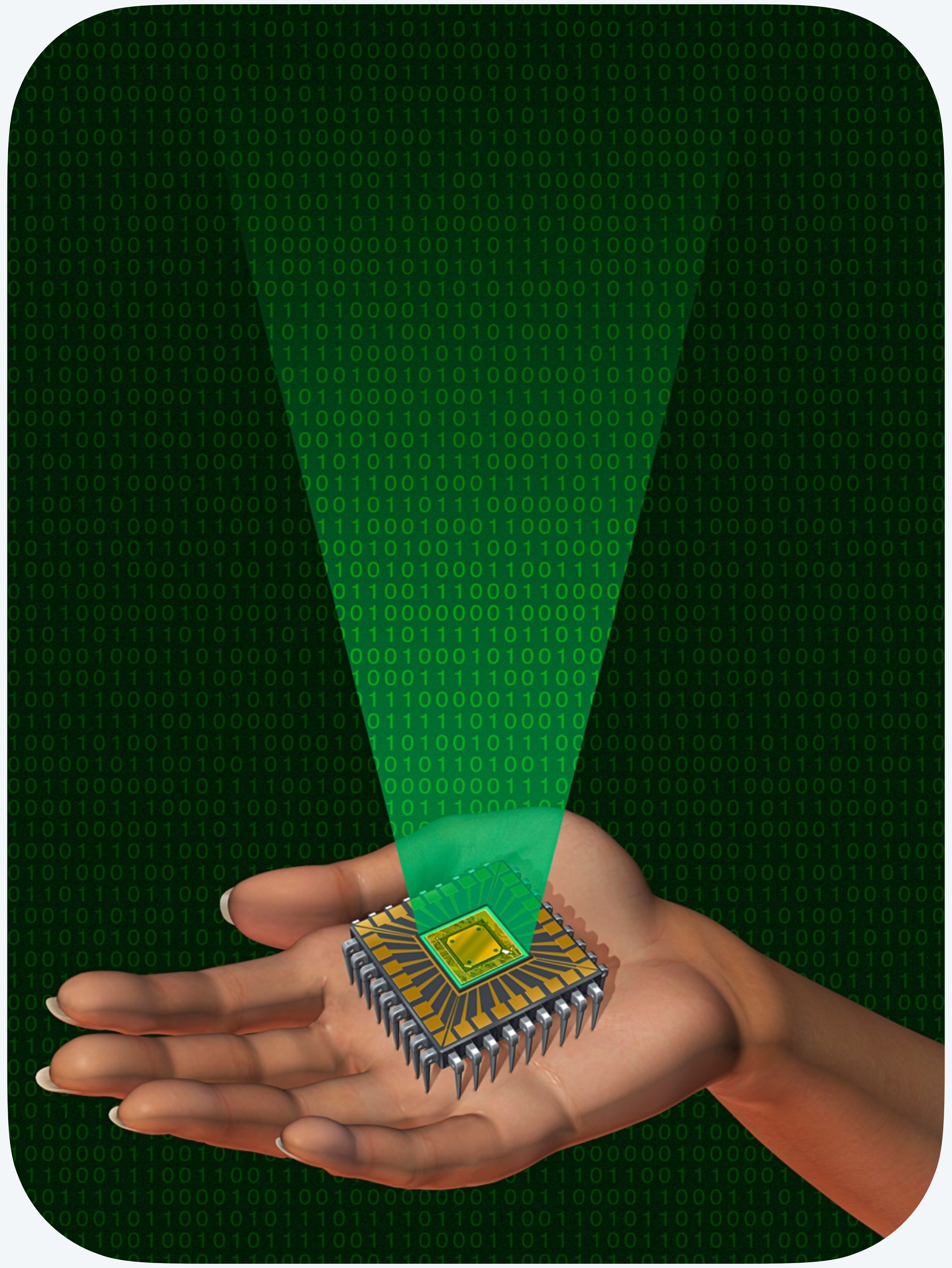
- Arithmetic logic unit (ALU).
- Memory and registers.
- Program counter (PC) and instruction register (IR).
- Input and output.

TOY supports same basic programming constructs as Java:

- Primitive data types.
- Arithmetic/logic operations.
- Conditionals and loops.
- Input and output.
- Arrays.
- Functions.
- Linked structures.

next lecture (Conditionals and loops, Input and output, Arrays)

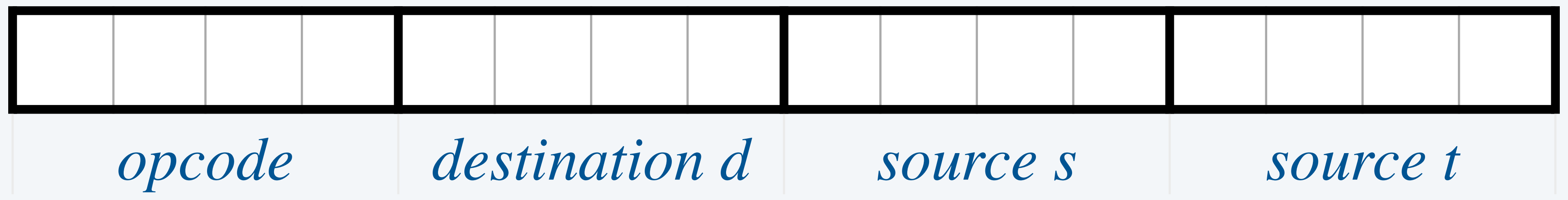
see textbook (Functions, Linked structures)



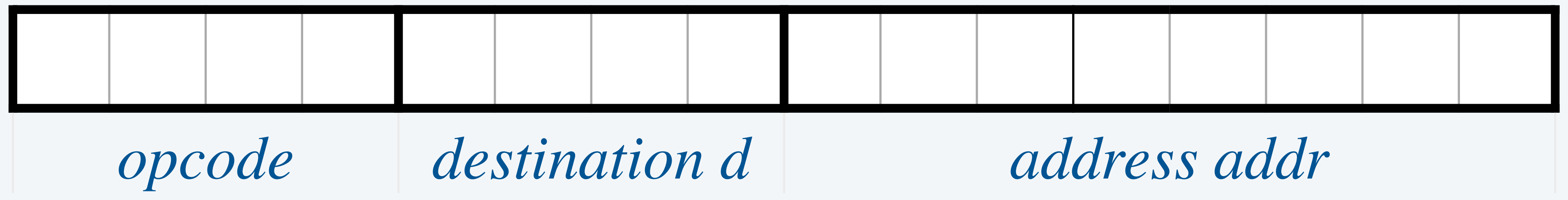
TOY reference sheet

opcode	operation	format	pseudo-code
0	*halt*	—	`halt`
1	*add*	*RR*	`R[d] = R[s] + R[t]`
2	*subtract*	*RR*	`R[d] = R[s] - R[t]`
3	*bitwise and*	*RR*	`R[d] = R[s] & R[t]`
4	*bitwise xor*	*RR*	`R[d] = R[s] ^ R[t]`
5	*shift left*	*RR*	`R[d] = R[s] << R[t]`
6	*shift right*	*RR*	`R[d] = R[s] >> R[t]`
7	*load address*	*A*	`R[d] = addr`
8	*load*	*A*	`R[d] = M[addr]`
9	*store*	*A*	`M[addr] = R[d]`
A	*load indirect*	*RR*	`R[d] = M[R[t]]`
B	*store indirect*	*RR*	`M[R[t]] = R[d]`
C	*branch zero*	*A*	`if (R[d] == 0) PC = addr`
D	*branch positive*	*A*	`if (R[d] > 0) PC = addr`
E	*jump register*	*RR*	`PC = R[d]`
F	*jump and link*	*A*	`R[d] = PC; PC = addr`

format RR



format A



zero `R[0]` *is always* `0000`.

standard input *Load from* `M[FF]`.

standard output *Store to* `M[FF]`.

Credits

image	source	license
PDP-8	Philipp Hachtmann	
A16 Bionic	Apple Inc.	
Silhouette Detective	publicdomainvectors.org	CC0 1.0
Modern Laptop	Adobe Stock	education license
Computer Chip and Earth	Adobe Stock	education license
3D 0s and 1s	Adobe Stock	education license
Toggle Switch	Adobe Stock	education license
Light Bulb	Adobe Stock	education license
Electron Nuclei	Adobe Stock	education license
Quantum Spin	Adobe Stock	education license
Counting to Ten	Adobe Stock	education license

Credits

image	source	license
10 Types of People	Zazzle	
Slide Rule	sliderulemuseum.com	
Abacus	Wikimedia	CC BY-SA 3.0
Marchant XLA Calculator	Wikimedia	public domain
Marchant SCM Calculator	Wikimedia	CC BY 3.0
Casio fx-85WA Calculator	Wikimedia	CC BY-SA 3.0
Fetch–Increment–Execute	Adobe Stock	education license
iPhone 14 Pro Max	Apple	
Can't Sleep	xkcd	CC BY-NC 2.5
Halt Sign	Wikimedia	public domain
Programming a PDP-8	computerhistory.org	
Computer Chip and Binary	Adobe Stock	education license