PHOT 110: Introduction to programming
LECTURE 01

Michaël Barbier, Spring semester (2023-2024)

COURSE INFORMATION

Instructor

Dr. Michaël Barbier
e-mail: michaelbarbier@iyte.edu.tr
Office – door on the right of Z5
Hours – 9:00-17:00 (appointment)

Teaching Assistants

Hazan Özkan
e-mail: hazanozkan@iyte.edu.tr
Office: Z9B
Office hours: TBD

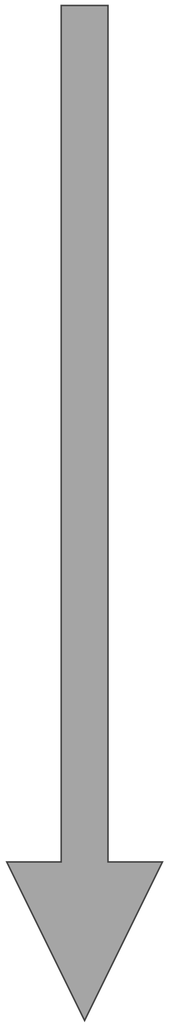
Metin Tan
e-mail: metintan@iyte.edu.tr
Office: Z9B
Office hours: TBD

Course Schedule

Monday	13:30 – 15:15	Ali Küçük Lab (computer lab on the 1st floor)
Thursday	09:45 – 11:30	Ali Küçük Lab

CONTENTS OF THE COURSE

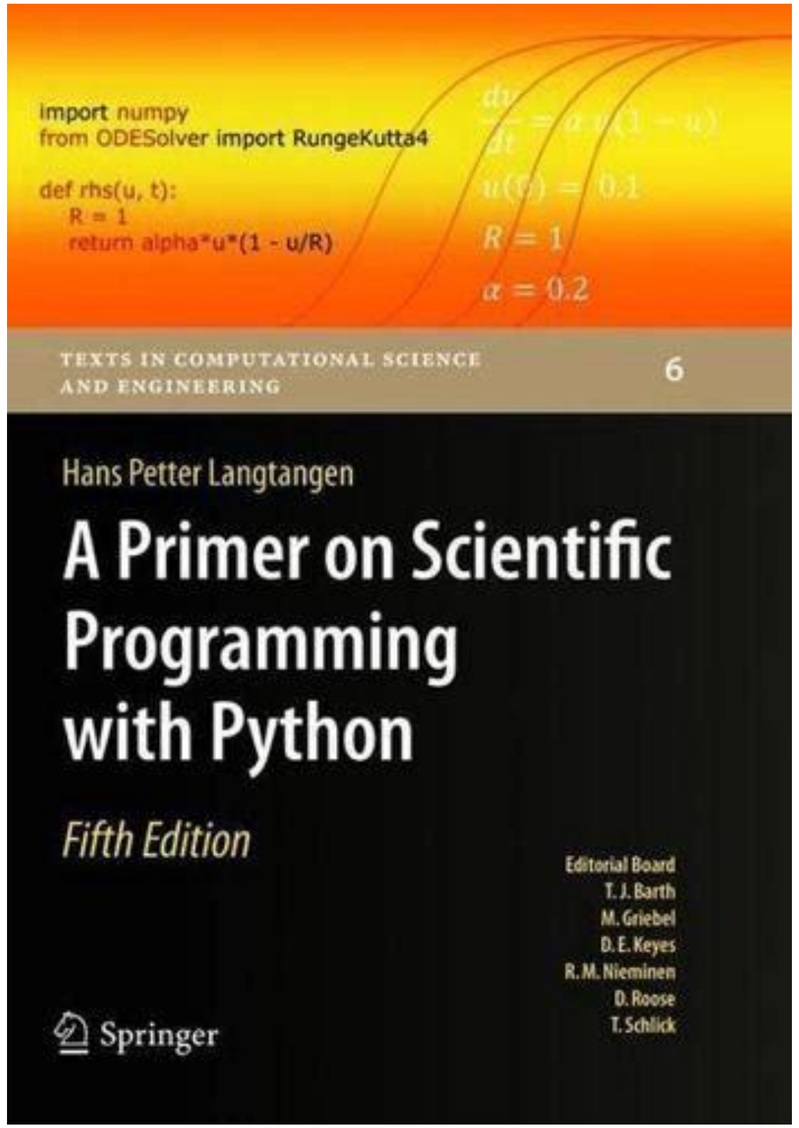
- How computers perform calculations
- How to program in Python
- Implementing numerical methods
- Handling input/output data, plotting graphs, tables
- Debugging, testing, and benchmarking your code



COURSE MATERIALS

Course book

H.P. Langtangen, A primer on scientific programming with Python, Springer



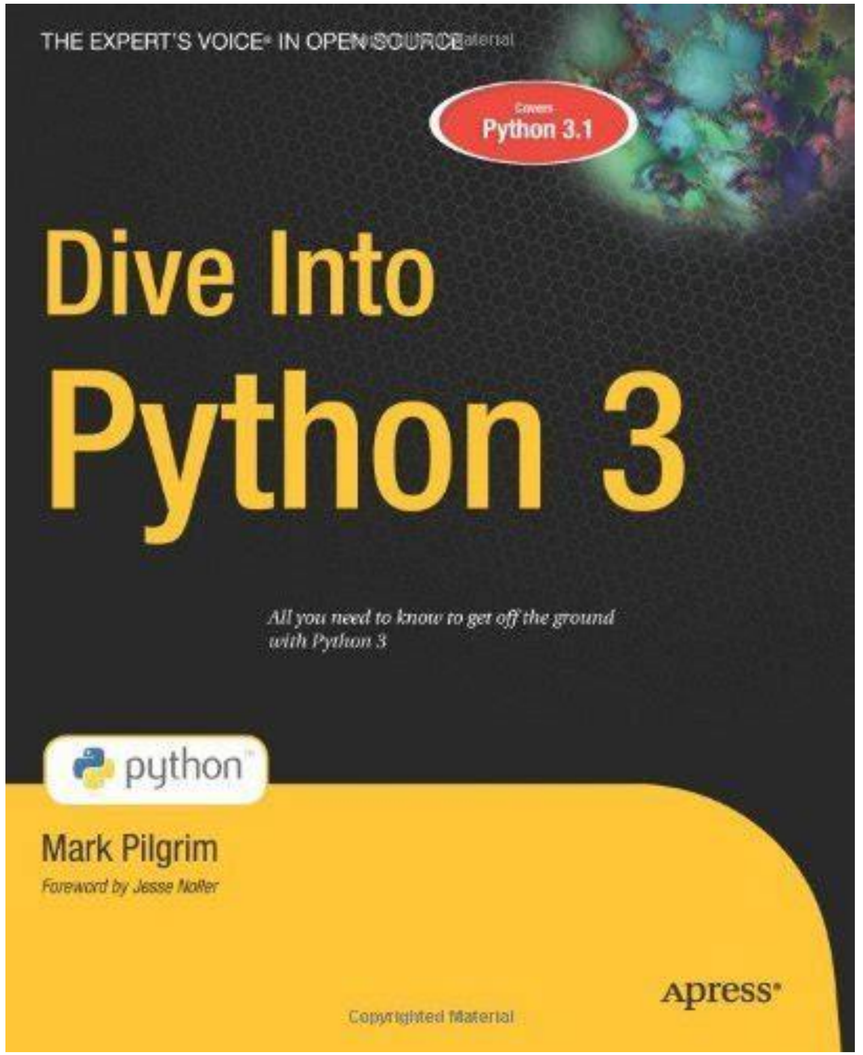
COURSE MATERIALS

Course book

H.P. Langtangen, A primer on scientific programming with Python, Springer

Supplementary material

M. Pilgrim, Dive Into Python 3
https://diveintopython3.net/



OVERVIEW OF THE COURSE

week	topic	Chapter
Week 1	Computers and Python basics: interpreter, IDE, running a script	1
Week 2	Expressions, variables, variable types	1
Week 3	Program control flow: while-loop, for-loop, lists, and ranges	2
Week 4	Program control flow (ctu'd): conditional execution. Further data types: nested lists, Tuples, Arrays	3
Week 5	Functions, variable scope	3
Week 6	Input/output and error handling	4
Week 7	Modules and script documentation	4
Week 8	Vectors, Arrays, implementation of numerical algorithms	5
Week 9	Plotting graphs using Matplotlib	5
Week 10	Dictionaries, Strings, tables with Pandas	6
Week 11	Object Oriented Programming	7
Week 12	Object Oriented Programming ctu'd	9
Week 13	Unit tests and integrated testing	App. F
Week 14	Performance, benchmarking, profiling	
Week 15	Programming good practices and summary	

COURSE SYLLABUS AND CLASS WORKFLOW

Homework/projects

- Small projects
- Working together on solutions allowed
- But .. individual written reports and code (to be explained afterwards, during the exam)

Exams

- Theoretical part & practical part
- ±5 minutes (TBD) individual explanation of the practical homework/projects

SO, LET’S GET STARTED !

PROBLEM SOLVING

PROBLEM SOLVING

Problem solving concerns the design/development of **algorithms** to solve given problems

What is an **algorithm**?

1. Sequence of obvious, logical steps or instructions
2. Solves a target problem
3. Has a definition of when to stop

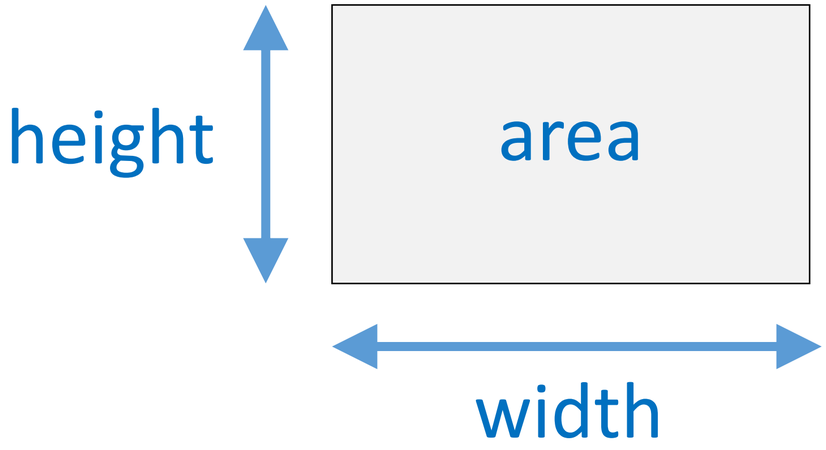
- Algorithms define for every given input an output
- An algorithm can be represented as a “function”

PROBLEM EXAMPLE: CALCULATE AREA

Problem: calculate the area of a rectangle (for any width and height)

Steps:

1. width ← ask input width
2. height ← ask input height
3. area ← width x height
4. output area

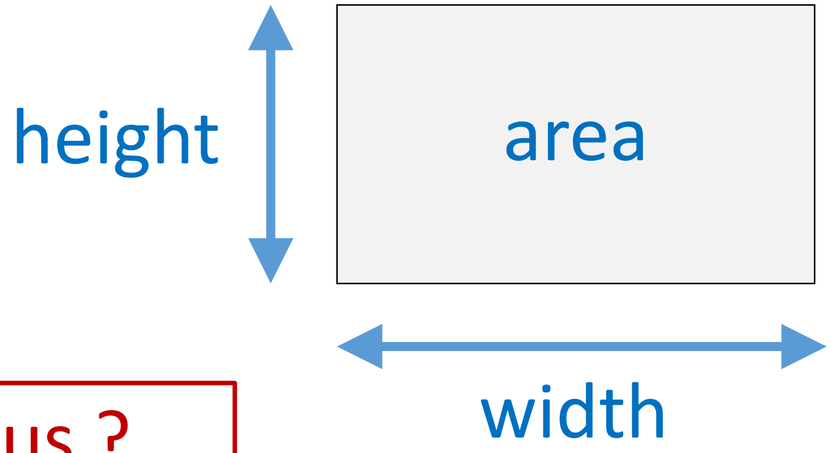


PROBLEM EXAMPLE: CALCULATE AREA

Problem: calculate the area of a rectangle (for any width and height)

Steps:

1. width ← ask input width
2. height ← ask input height
3. area ← width x height
4. output area



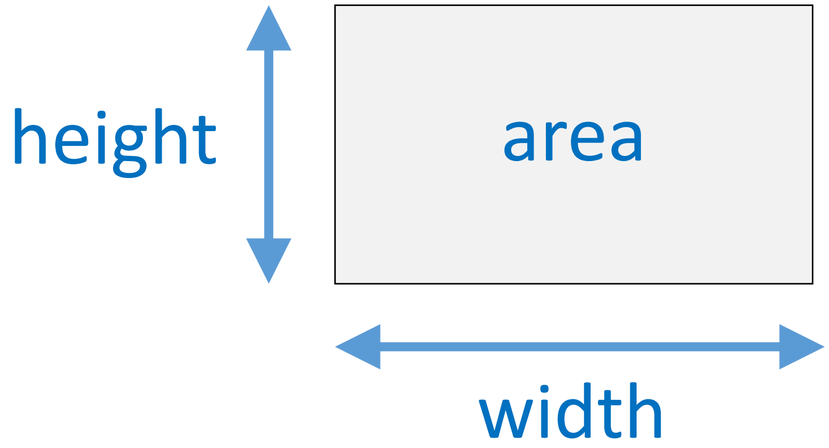
Are the steps obvious ?
Is the solution correct ?
Does the algorithm stop ?

PROBLEM EXAMPLE: CALCULATE AREA

Problem: calculate the area of a rectangle (for any width and height)

Steps:

1. width ← ask input width
2. height ← ask input height
3. area ← width x height
4. output area



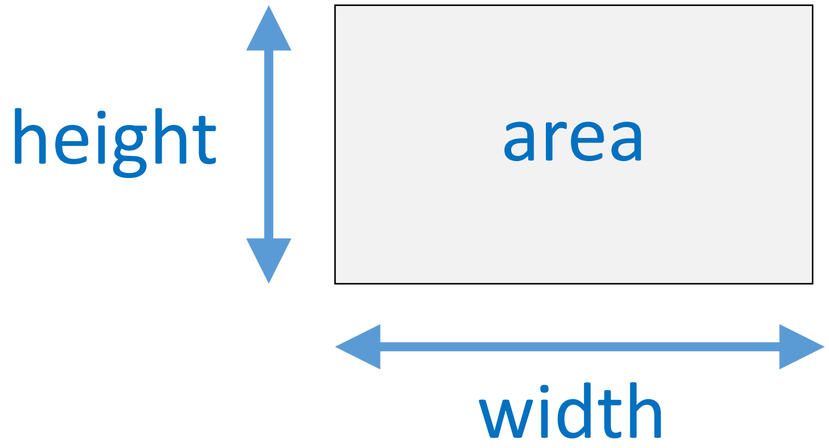
After step	width	height	area
step 1	6	?	?
step 2	6	3	?
step 3	6	3	18
step 4	6	3	18

PROBLEM EXAMPLE: CALCULATE AREA

Problem: calculate the area of a rectangle (for any width and height)

Steps:

1. width ← ask input width
2. height ← ask input height
3. area ← width x height
4. output area



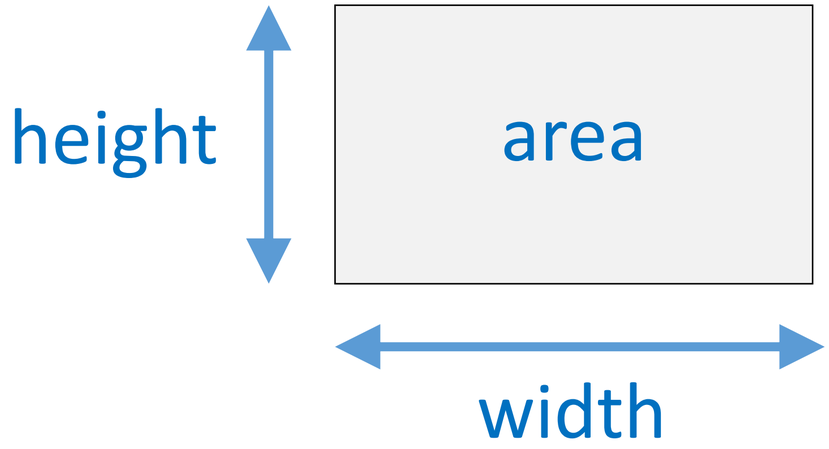
After step	width	height	area
step 1	6	?	?
step 2	6	**3**	?
step 3	6	3	18
step 4	6	3	18

PROBLEM EXAMPLE: CALCULATE AREA

Problem: calculate the area of a rectangle (for any width and height)

Steps:

1. width ← ask input width
2. height ← ask input height
3. area ← width x height
4. output area



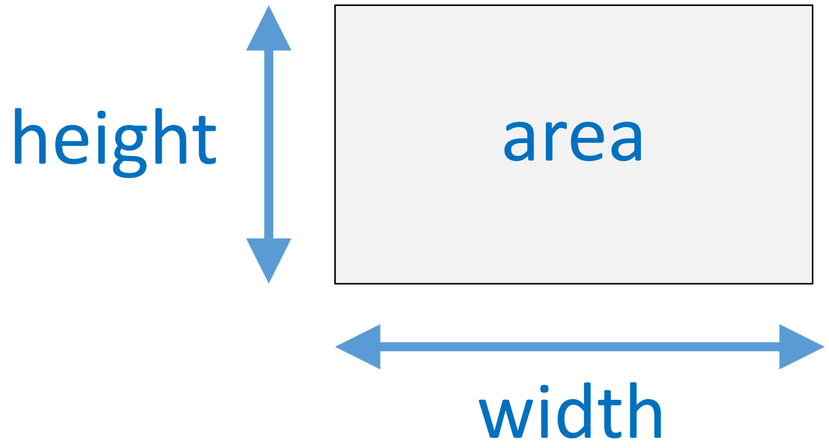
After step	width	height	area
step 1	6	?	?
step 2	6	3	?
step 3	6	3	**18**
step 4	6	3	18

PROBLEM EXAMPLE: CALCULATE AREA

Problem: calculate the area of a rectangle (for any width and height)

Steps:

1. width ← ask input width
2. height ← ask input height
3. area ← width x height
4. output area



After step	width	height	area
step 1	6	?	?
step 2	6	3	?
step 3	6	3	18
step 4	6	3	**18**

PROBLEM EXAMPLE: CALCULATE FACTORIAL N!

Problem: calculate the factorial

$$N! = N \times (N-1) \times \ldots \times 2 \times 1$$

Steps:

1. $N \leftarrow$ input from user
2. $f \leftarrow N$
3. while $N > 1$:
   $N \leftarrow N - 1$
   $f \leftarrow f \times N$
4. output f

PROBLEM EXAMPLE: CALCULATE FACTORIAL N!

Problem: calculate the factorial

$$N! = N \times (N-1) \times \ldots \times 2 \times 1$$

Steps:

1. $N \leftarrow$ input from user
2. $f \leftarrow N$
3. while $N > 1$:
   $N \leftarrow N - 1$
   $f \leftarrow f \times N$
4. output f

Are the steps obvious ?
Is the solution correct ?
Does the algorithm stop ?

PROBLEM EXAMPLE: CALCULATE FACTORIAL N!

Problem: calculate the factorial

$$N! = N \times (N-1) \times \ldots \times 2 \times 1$$

Steps:

1. $N \leftarrow$ input from user
2. $f \leftarrow N$
3. while $N > 1$:
   $N \leftarrow N - 1$
   $f \leftarrow f \times N$
4. output f

Are the steps obvious ?
Is the solution correct ?
Does the algorithm stop ?

Let’s check !

PROBLEM EXAMPLE: CALCULATE FACTORIAL N!

Problem: calculate the factorial

$$N! = N \times (N-1) \times \ldots \times 2 \times 1$$

Steps:

1. $N \leftarrow$ input from user
2. $f \leftarrow N$
3. if $N > 1$:
   - $N \leftarrow N - 1$
   - $f \leftarrow f \times N$
   - go to step 3

   else
   - output f

Are the steps obvious ?
Is the solution correct ?
Does the algorithm stop ?

Let's check !

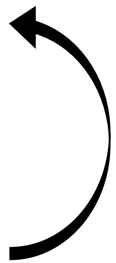
PROBLEM EXAMPLE: CALCULATE FACTORIAL N!

Problem: calculate the factorial

$$N! = N \times (N-1) \times \ldots \times 2 \times 1$$

Steps:

1. $N \leftarrow$ input from user
2. $f \leftarrow N$
3. if $N > 1$:
   $N \leftarrow N - 1$
   $f \leftarrow f \times N$
   go to step 3
   else
   output f



After step	Process line	N	f
step 1	$N \leftarrow 4$	4	?
step 2	$f \leftarrow N$	4	4
step 3 (1.)	$N \leftarrow N - 1$	3	4
step 3 (1.)	$f \leftarrow f \times N$	3	12
step 3 (2.)	$N \leftarrow N - 1$	2	12
step 3 (2.)	$f \leftarrow f \times N$	2	24
step 3 (3.)	$N \leftarrow N - 1$	1	24
step 3 (3.)	$f \leftarrow f \times N$	1	24
step 3 (4.)	Output f	1	24

PROBLEM EXAMPLE: CALCULATE FACTORIAL N!

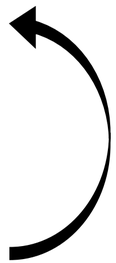
Problem: calculate the factorial

$$N! = N \times (N-1) \times \ldots \times 2 \times 1$$

Steps:

1. $N \leftarrow$ input from user
2. $f \leftarrow N$
3. if $N > 1$:
   - $N \leftarrow N - 1$
   - $f \leftarrow f \times N$
   - go to step 3

   else
   - output f



After step	Process line	N	f
step 1	$N \leftarrow 4$	4	?
step 2	$f \leftarrow N$	4	4
step 3 (1.)	$N \leftarrow N - 1$	3	4
step 3 (1.)	$f \leftarrow f \times N$	3	12
step 3 (2.)	$N \leftarrow N - 1$	2	12
step 3 (2.)	$f \leftarrow f \times N$	2	24
step 3 (3.)	$N \leftarrow N - 1$	1	24
step 3 (3.)	$f \leftarrow f \times N$	1	24
step 3 (4.)	Output f	1	24

PROBLEM EXAMPLE: CALCULATE FACTORIAL N!

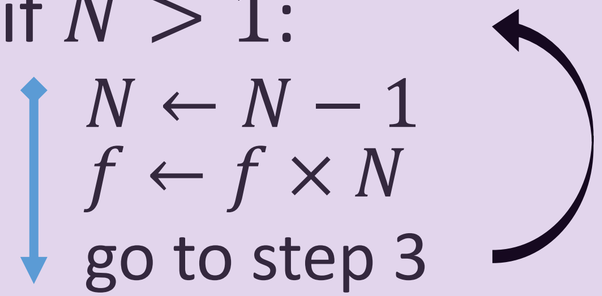
Problem: calculate the factorial

$$N! = N \times (N-1) \times \ldots \times 2 \times 1$$

Steps:

1. $N \leftarrow$ input from user
2. $f \leftarrow N$
3. if $N > 1$:
   - $N \leftarrow N - 1$
   - $f \leftarrow f \times N$
   - go to step 3

   else
   - output f



After step	Process line	N	f
step 1	$N \leftarrow 4$	4	?
step 2	$f \leftarrow N$	4	4
step 3 (1.)	$N \leftarrow N - 1$	**3**	4
step 3 (1.)	$f \leftarrow f \times N$	3	**12**
step 3 (2.)	$N \leftarrow N - 1$	**2**	12
step 3 (2.)	$f \leftarrow f \times N$	2	**24**
step 3 (3.)	$N \leftarrow N - 1$	**1**	24
step 3 (3.)	$f \leftarrow f \times N$	1	24
step 3 (4.)	Output f	1	24

PROBLEM EXAMPLE: CALCULATE FACTORIAL N!

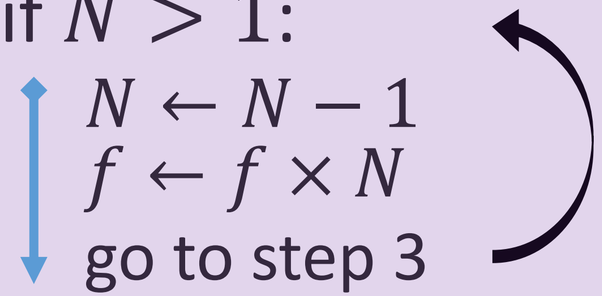
Problem: calculate the factorial

$$N! = N \times (N-1) \times \ldots \times 2 \times 1$$

Steps:

1. $N \leftarrow$ input from user
2. $f \leftarrow N$
3. if $N > 1$:
   - $N \leftarrow N - 1$
   - $f \leftarrow f \times N$
   - go to step 3

   else
   - output f



After step	Process line	N	f
step 1	$N \leftarrow 4$	4	?
step 2	$f \leftarrow N$	4	4
step 3 (1.)	$N \leftarrow N - 1$	**3**	4
step 3 (1.)	$f \leftarrow f \times N$	3	**12**
step 3 (2.)	$N \leftarrow N - 1$	**2**	12
step 3 (2.)	$f \leftarrow f \times N$	2	**24**
step 3 (3.)	$N \leftarrow N - 1$	**1**	24
step 3 (3.)	$f \leftarrow f \times N$	1	24
step 3 (4.)	Output f	1	24

PROBLEM EXAMPLE: CALCULATE FACTORIAL N!

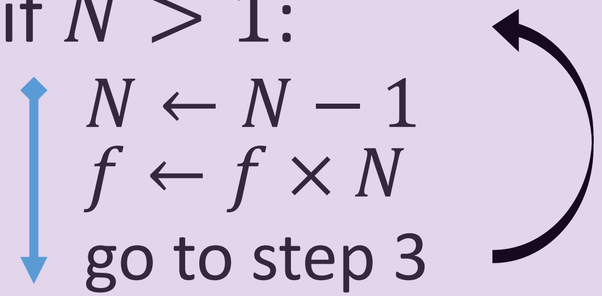
Problem: calculate the factorial

$$N! = N \times (N-1) \times \ldots \times 2 \times 1$$

Steps:

1. $N \leftarrow$ input from user
2. $f \leftarrow N$
3. if $N > 1$:
   - $N \leftarrow N - 1$
   - $f \leftarrow f \times N$
   - go to step 3

   else
   - output f



After step	Process line	N	f
step 1	$N \leftarrow 4$	4	?
step 2	$f \leftarrow N$	4	4
step 3 (1.)	$N \leftarrow N - 1$	**3**	4
step 3 (1.)	$f \leftarrow f \times N$	3	**12**
step 3 (2.)	$N \leftarrow N - 1$	**2**	12
step 3 (2.)	$f \leftarrow f \times N$	2	**24**
step 3 (3.)	$N \leftarrow N - 1$	**1**	24
step 3 (3.)	$f \leftarrow f \times N$	1	24
step 3 (4.)	Output f	1	24

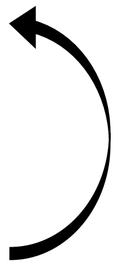
PROBLEM EXAMPLE: CALCULATE FACTORIAL N!

Problem: calculate the factorial

$$N! = N \times (N-1) \times \ldots \times 2 \times 1$$

Steps:

1. $N \leftarrow$ input from user
2. $f \leftarrow N$
3. if $N > 1$:
   $N \leftarrow N - 1$
   $f \leftarrow f \times N$
   go to step 3
   else
   output f



After step	Process line	N	f
step 1	$N \leftarrow 4$	4	?
step 2	$f \leftarrow N$	4	4
step 3 (1.)	$N \leftarrow N - 1$	**3**	4
step 3 (1.)	$f \leftarrow f \times N$	3	**12**
step 3 (2.)	$N \leftarrow N - 1$	**2**	12
step 3 (2.)	$f \leftarrow f \times N$	2	**24**
step 3 (3.)	$N \leftarrow N - 1$	**1**	24
step 3 (3.)	$f \leftarrow f \times N$	1	24
step 3 (4.)	Output f	1	24

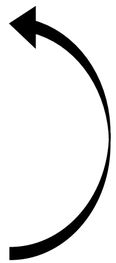
PROBLEM EXAMPLE: CALCULATE FACTORIAL N!

Problem: calculate the factorial

$$N! = N \times (N-1) \times \ \ldots \times 2 \times 1$$

Steps:

1. $N \leftarrow$ input from user
2. $f \leftarrow N$
3. if $N > 2$:
   $N \leftarrow N - 1$
   $f \leftarrow f \times N$
   go to step 3
   else
   output f



Last (4^{th}) time is unnecessary

ALGORITHM PSEUDO CODE

What is pseudo code?

- A more formal way of writing the steps of an algorithm
- Independent from programming language
- Our algorithm steps could be considered **pseudo code !**

Let’s look again:

ALGORITHM PSEUDO CODE

What is pseudo code?

- A more formal way of writing the steps of an algorithm
- Independent from programming language
- Our algorithm steps could be considered **pseudo code !**

Let's look again:

Area of a rectangle

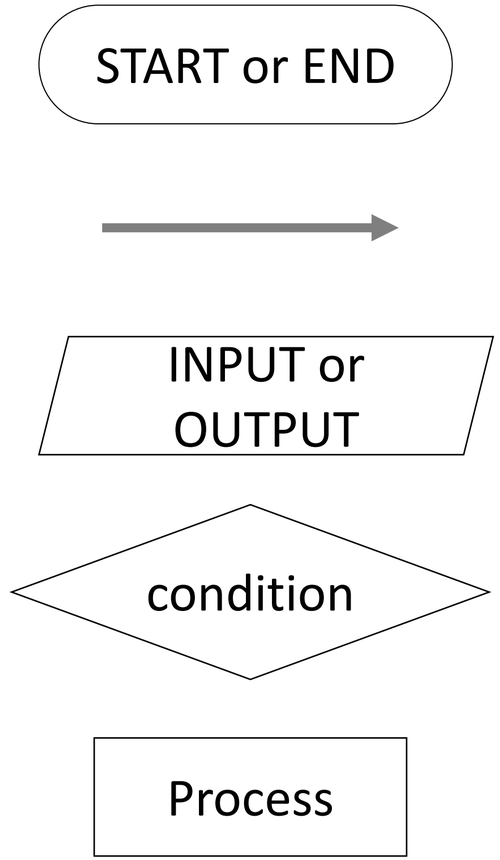
1. width ← ask input width
2. height ← ask input height
3. area ← width x height
4. output area

Factorial $N!$

1. $N \leftarrow$ input from user
2. $f \leftarrow N$
3. while $N > 2$:
   $N \leftarrow N - 1$
   $f \leftarrow f \times N$
4. output f

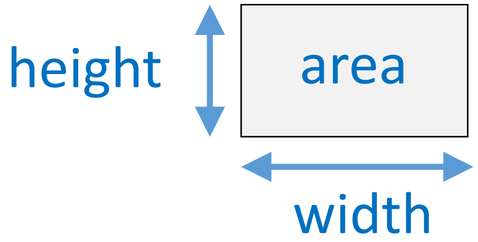
ALGORITHM FLOW CHART

- Visual representation of the steps
- Different blocks
  - **Terminals**
  - **Flow lines** connect blocks
  - **Input/output**
  - **Decisions** according to certain conditions
  - **Operations** on data (processes)



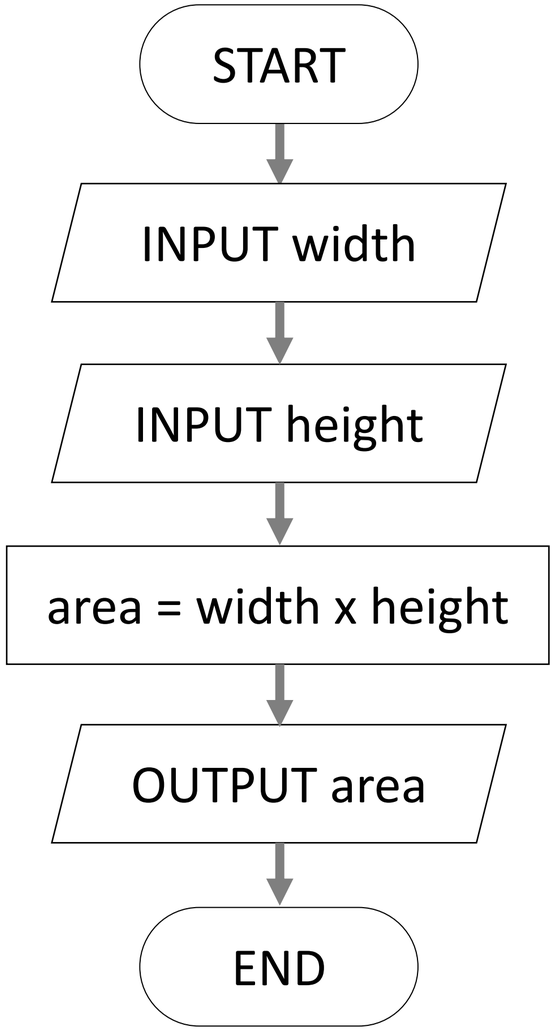
ALGORITHM FLOW CHART

- Visual representation of the steps



Pseudo code steps:

1. width ← ask input width
2. height ← ask input height
3. area ← width x height
4. output area



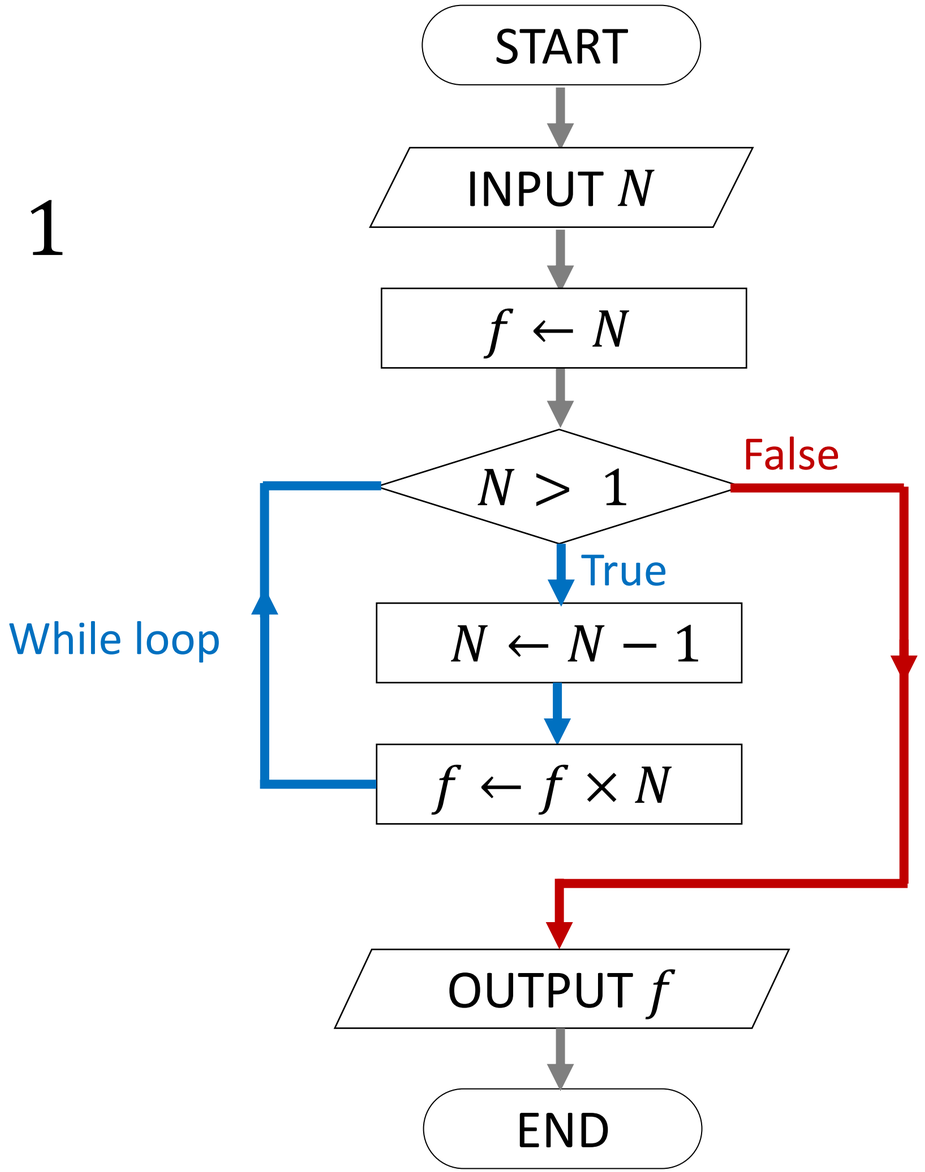
ALGORITHM FLOW CHART

- Visual representation of the steps

$$N! = N \times (N-1) \times \ldots \times 2 \times 1$$

Pseudo code steps:

1. $N \leftarrow$ input from user
2. $f \leftarrow N$
3. while $N > 1$:
   $N \leftarrow N - 1$
   $f \leftarrow f \times N$
4. output f

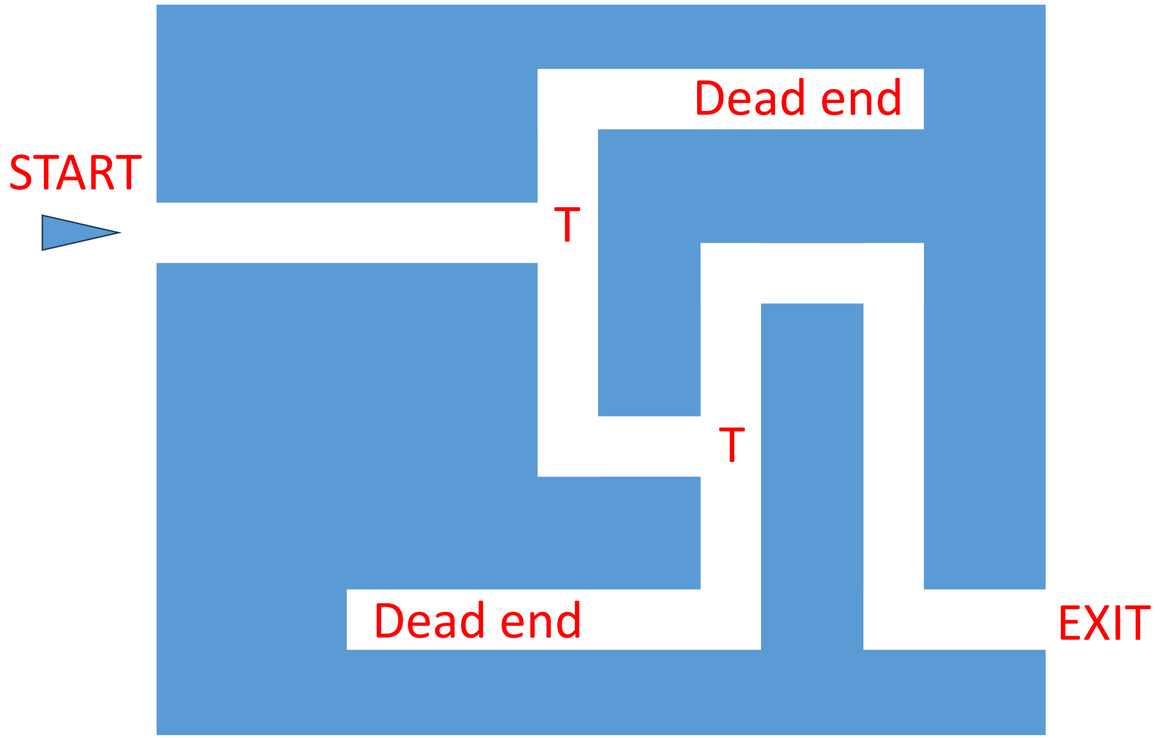


NONDETERMINISTIC ALGORITHMS

Problem: escape from a classical wall maze (T-junctions and dead ends)

Steps:

1. Take a step along the road
2. if at a dead end OR at start:
   Turn back
3. if at a T-junction:
   Turn to **random** arm {left, right}
4. if at exit
   END
   else
   go to step 1

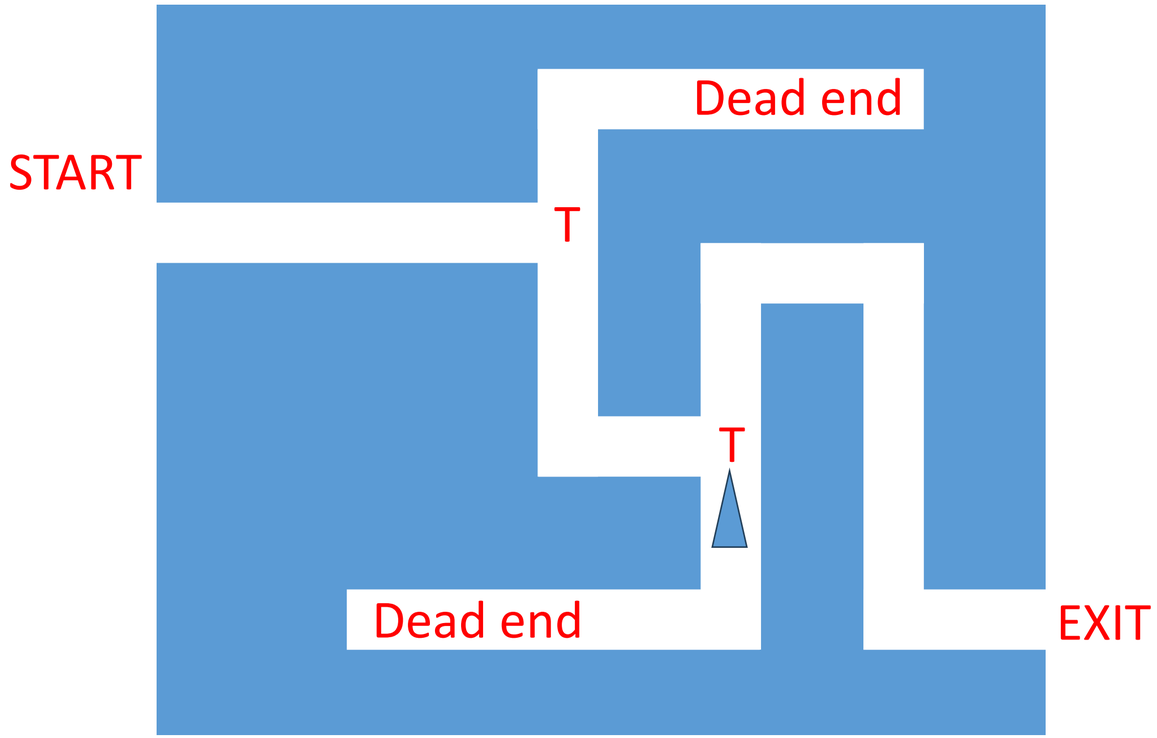


AMBIGUOUS OR NONLOGICAL STEPS

Problem: escape from a classical wall maze (T-junctions and dead ends)

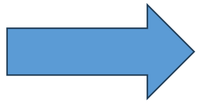
Steps:

1. Take a step along the road
2. if at a dead end OR at start:
   Turn back
3. if at a T-junction:
   Turn to random arm {left, right}
4. if at exit:
   END
   else:
   go to step 1



PROBLEM EXAMPLE: SORT BOOKS ALPHABETICALLY

Not all problems are math or riddles:

Unsorted books		Alphabetically sorted
1. Good Omens		1. Dodger
2. The Color of Magic		2. Good Omens
3. Dodger		3. The Color of Magic
4. Monstrous Regiment		4. The Rincewind Trilogy
5. The Rincewind Trilogy		5. Monstrous Regiment
6. Nanny Ogg's Cookbook		6. Nanny Ogg's Cookbook

SUMMARY OF PROBLEM SOLVING

- **Algorithms**
  - Recipe to solve a problem
  - Step-wise implementation of a function
  - Obvious logical steps
- **Pseudo-code**
  - Representation in code of an algorithm
  - Independent of “real” programming language
- **Flow-charts**
  - Visual representation of an algorithm
  - Blocks for input/output, decisions, and “calculations”

THE COMPUTER & PROGRAMS

PROGRAMS

Program = algorithm executed on a machine

- Steps are arithmetic and logical instructions
- Control flow is defined by instructions
- Stopping mechanism when program finishes

The machine:

- A “calculator” if the program is fixed
- A stored program “computer” if the instructions are stored on the machine

$$A = 10 \times B + 23$$

PROGRAMS

Program = algorithm executed on a machine

- Steps are arithmetic and logical instructions
- Control flow is defined by instructions
- Stopping mechanism when program finishes

$B < C$?
True or false

(1) $B =$ input value
(2) $A = 10 \times B - 100$
(3) if $A < B$ then $A = A + 5$
(4) output value = A

The machine:

- A “calculator” if the program is fixed
- A stored program “computer” if the instructions are stored on the machine

COMPUTERS OF ALL KINDS

Mainframe/cluster

PC / Desktop

Laptop

Smartphone

Smartwatch

ELEMENTS OF A MODERN COMPUTER

Everything connects to the motherboard

- Hard disk / solid state drives
- DVD player
- Memory
- CPU
- USB/sound peripheral connection sockets
- Video cards / ethernet cards

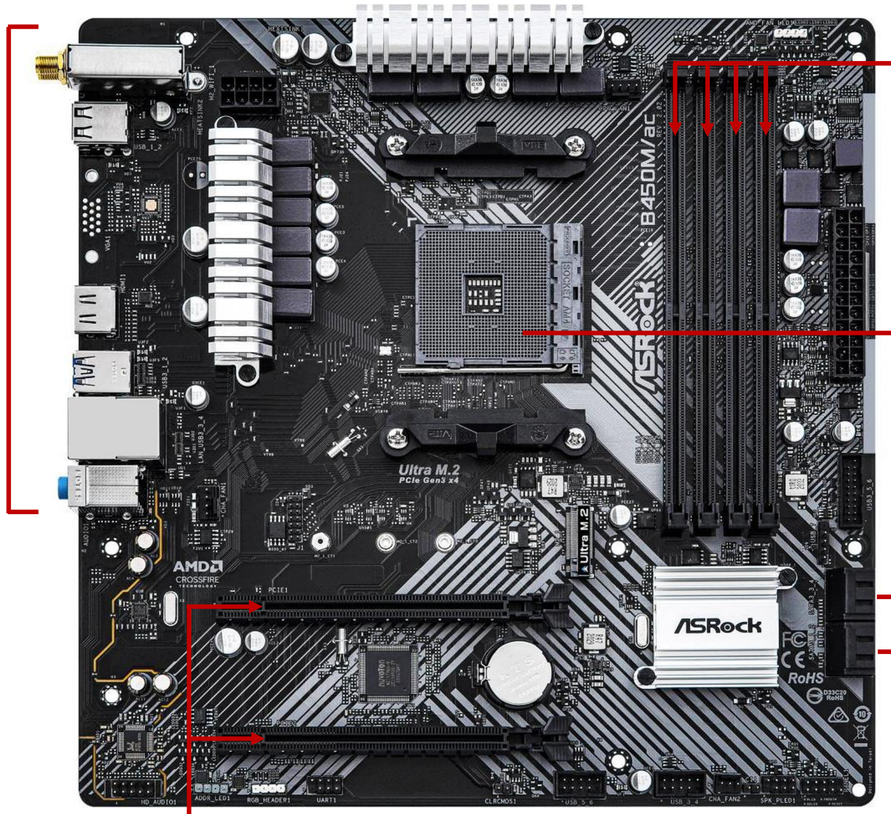
…

ELEMENTS OF A MODERN COMPUTER

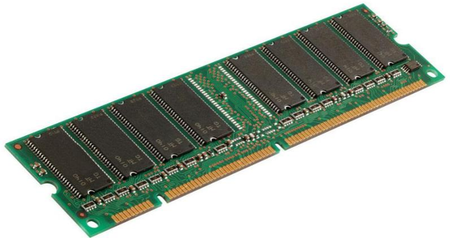
Peripherals:

- Microphone/speakers
- Keyboard/mouse
- Network
- Screen (onboard)

Motherboard



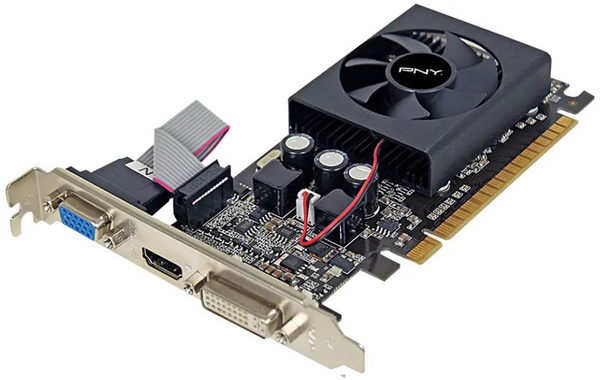
Memory (RAM)



CPU

SATA: HDD, SSD, …

PCI Express slots:
- video card, …



ELEMENTS ARE CONNECTED & CPU CONTROLS

Central Processing Unit (CPU)

BUS

Peripheral devices

Video card

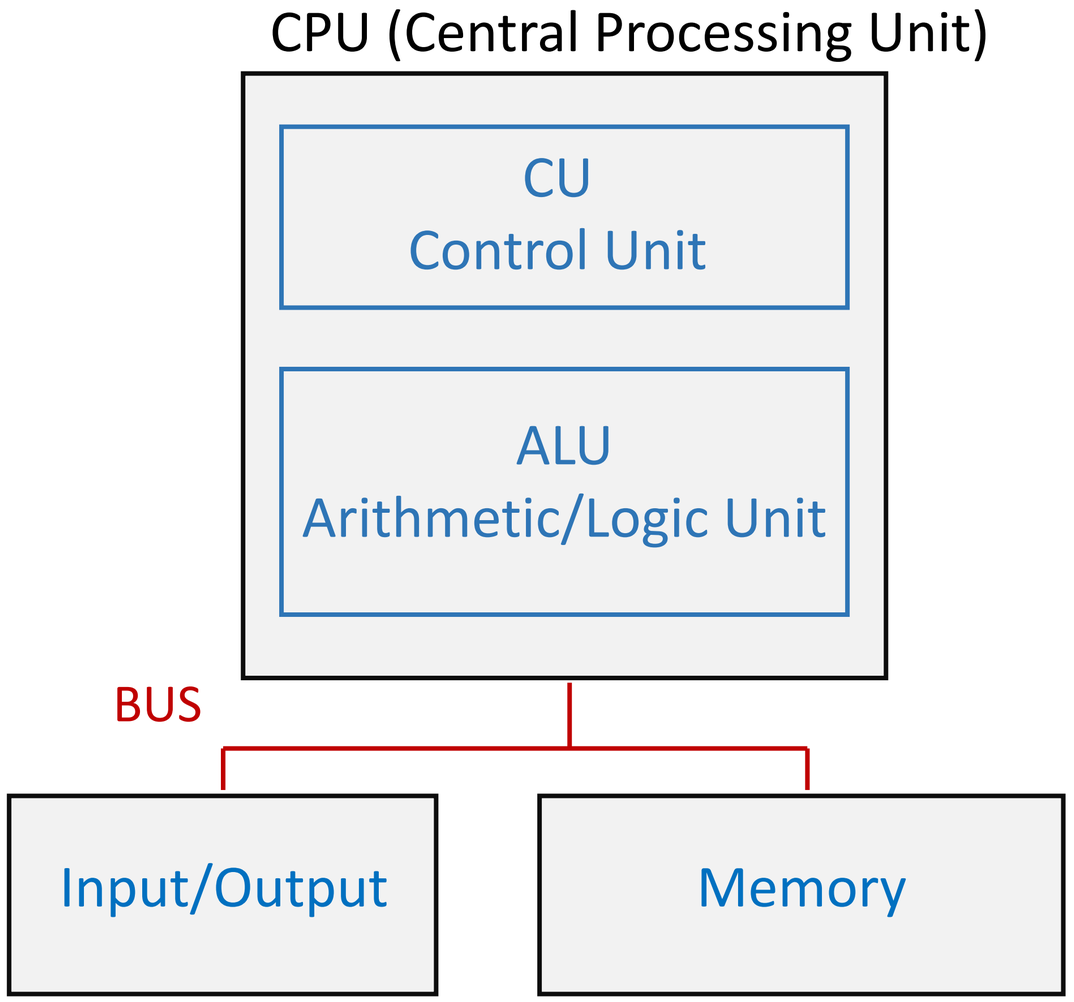
Memory

Hard disk

…

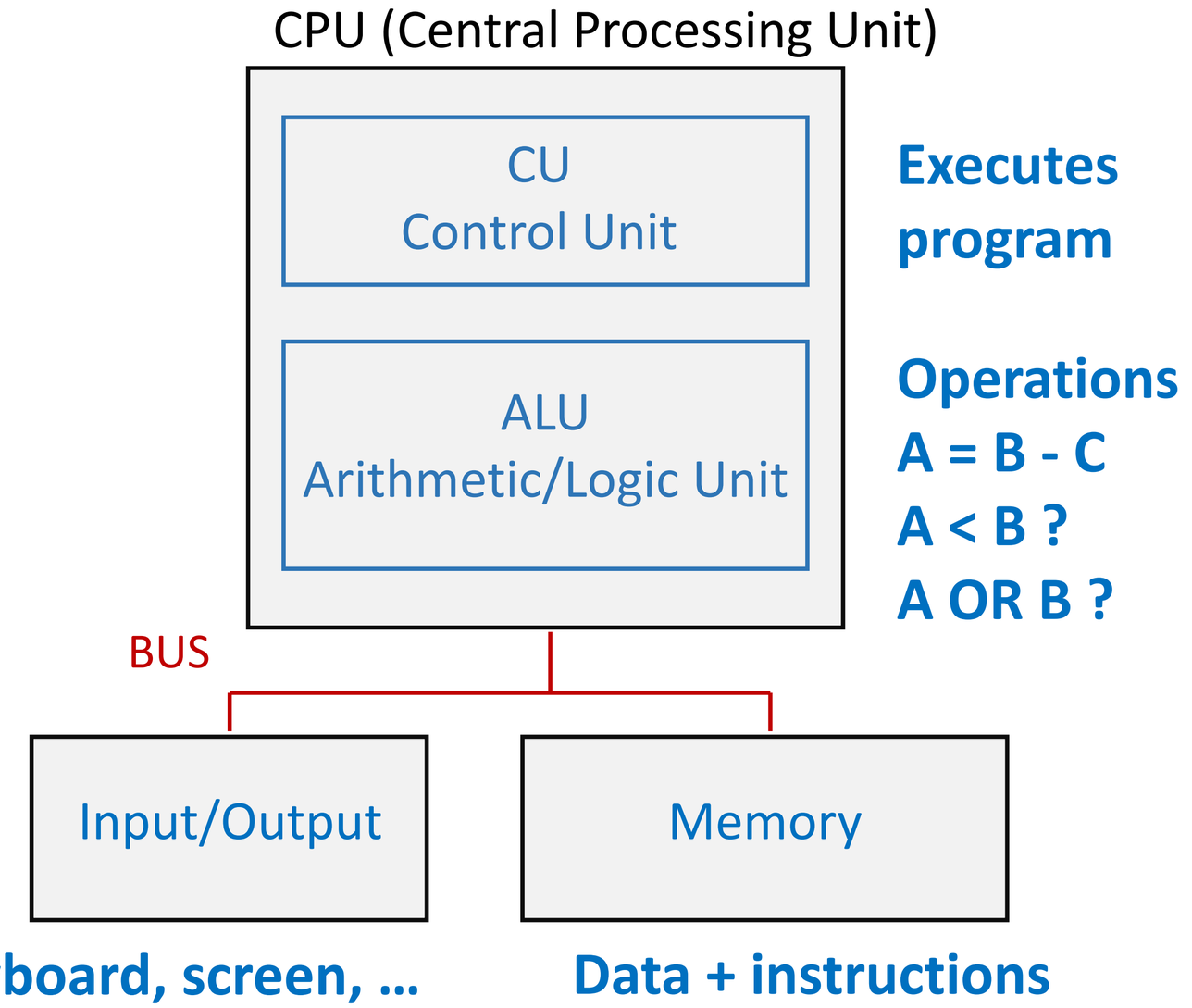
BACK TO THE VON NEUMANN ARCHITECTURE

- Model to design computers
- Defines four parts
  - Memory
  - ALU – Arithmetic/Logic Unit
  - CU – Control Unit
  - Input/Output
- Both data & program instructions stored in memory
- Program instructions executed sequentially



BACK TO THE VON NEUMANN ARCHITECTURE

- Model to design computers
- Defines four parts
  - Memory
  - ALU – Arithmetic/Logic Unit
  - CU – Control Unit
  - Input/Output
- Both data & program instructions stored in memory
- Program instructions executed sequentially



MEMORY

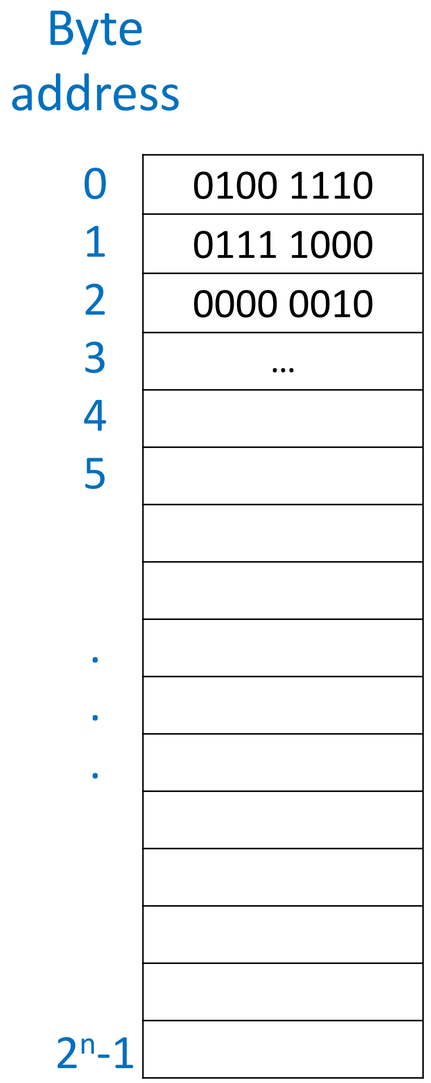
- Random Access Memory (RAM)
- Long array of bits
- Address per Byte
- Largest possible address depend on n-bits addressing:

16-bits address: $2^{16}-1 = 65{,}535 \approx 64$ kB

32–bits address: $232-1 = 4{,}294{,}967{,}295 \approx 4$ GB

- Defines maximum usable memory

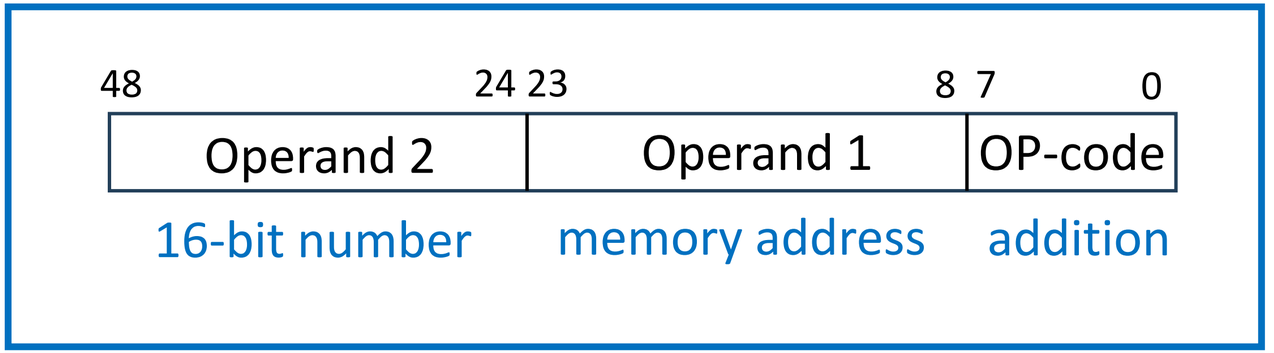
  (if no trick/segmentation applied)



MACHINE INSTRUCTIONS

- Defined by the **Instruction Set Architecture** (ISA)
- Translates into digital signal voltages
- Instructions
  - OP-CODE: ADD, MOV, etc.
  - 0, 1, 2, or 3 operands
  - variable length of bits !
- Operands can refer to
  - “Immediate” numbers/characters
  - A register itself
  - The memory addresses in a register

Example instruction

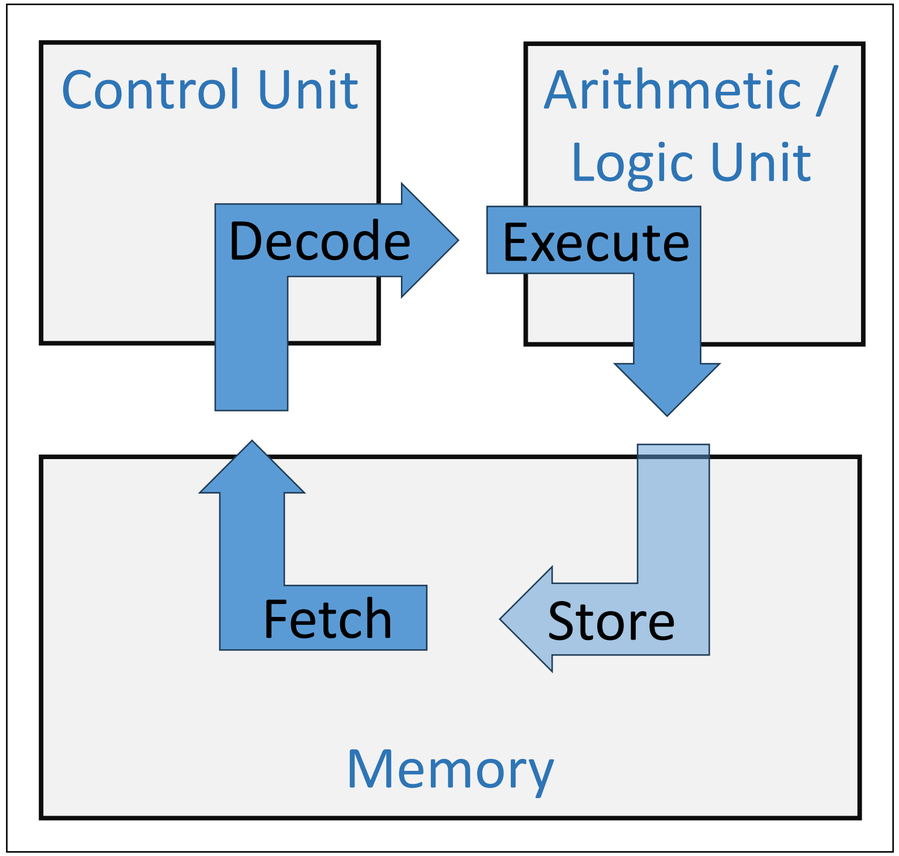


CPU - CENTRAL PROCESSING UNIT

- Executes sequence of instructions (loaded from memory)
  - FDE – **Fetch, Decode, Execute**
  - Clock-speed – #FDE per clock-cycle
  - Pipeline: queue of instructions
- **Instruction Set Architecture** (ISA)

  (= Computer Architecture)
  - Control flow & Arithmetic & logic
  - Load/store data from/on memory
- Few (e.g. 16) **registers**
  - Fast but small in size: e.g. 32-bit

Instruction cycle



CPU - CENTRAL PROCESSING UNIT

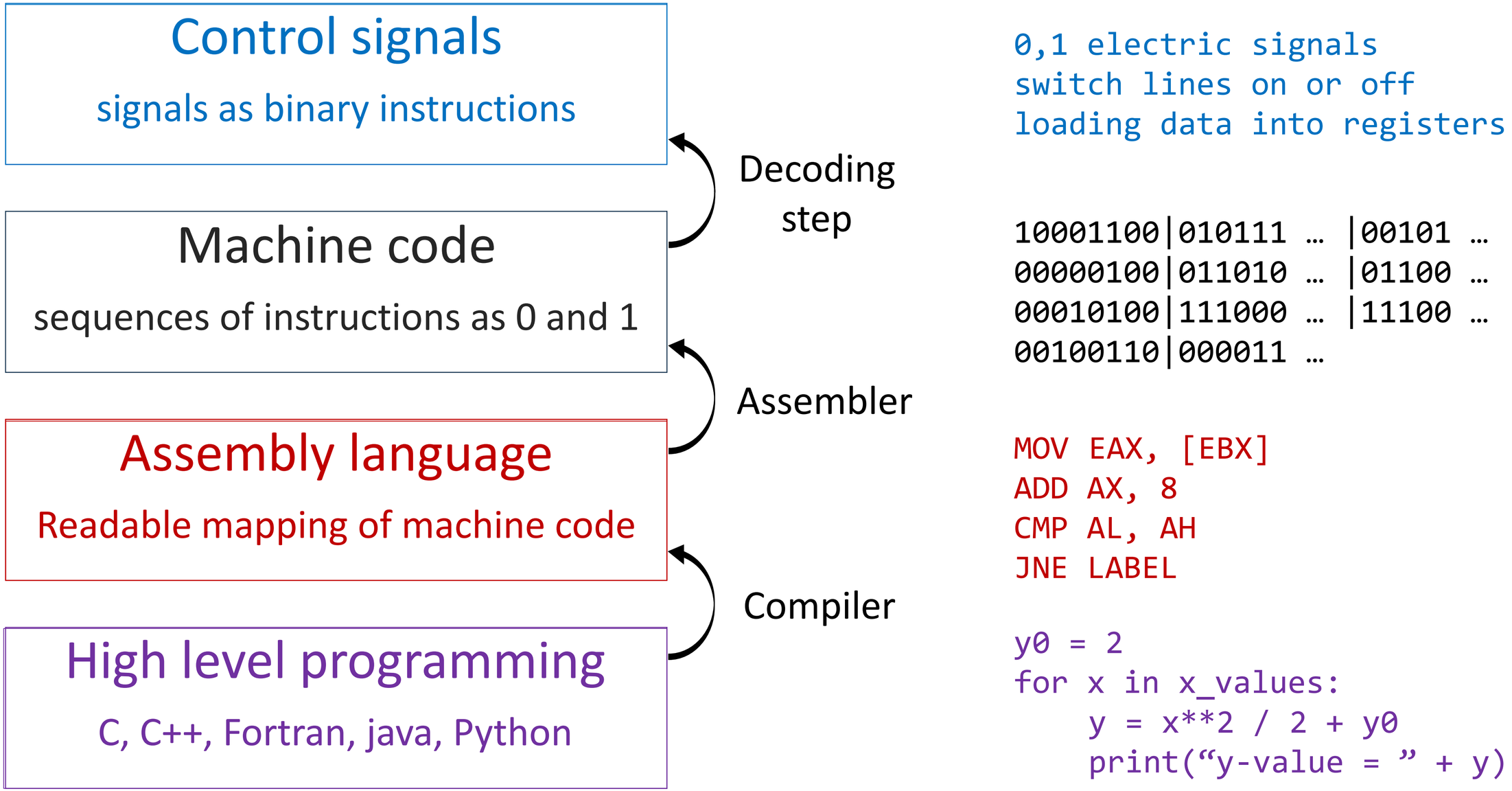
- Executes sequence of instructions (loaded from memory)
  - FDE – **Fetch, Decode, Execute**
  - Clock-speed – #FDE per clock-cycle
  - Pipeline: queue of instructions
- **Instruction Set Architecture** (ISA)

  (= Computer Architecture)
  - Control flow & Arithmetic & logic
  - Load/store data from/on memory
- Few (e.g. 16) **registers**
  - Fast but small in size: e.g. 32-bit

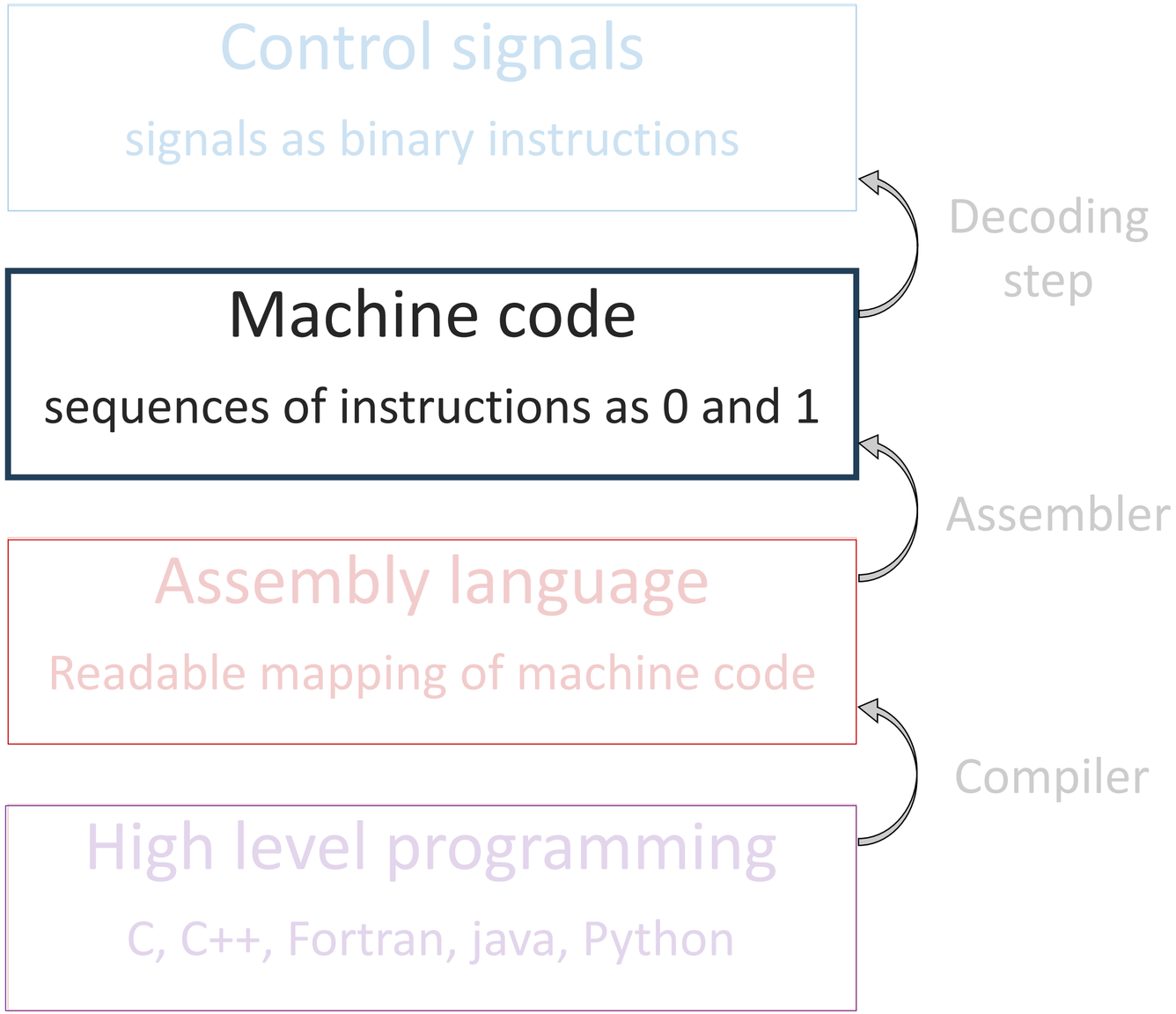
MOS 6502 registers

F E D C B A 9 8 7 6 5 4 3 2 1 0 *(bit position, hex)*	
Main registers	
A	**A**ccumulator
Index registers	
X	**X** index
Y	**Y** index
0 0 0 0 0 0 0 1 S	**S**tack **P**ointer
Program counter	
PC	**P**rogram **C**ounter
Status register	
N V - B D I Z C	**P** Processor flags

PROGRAM LEVELS



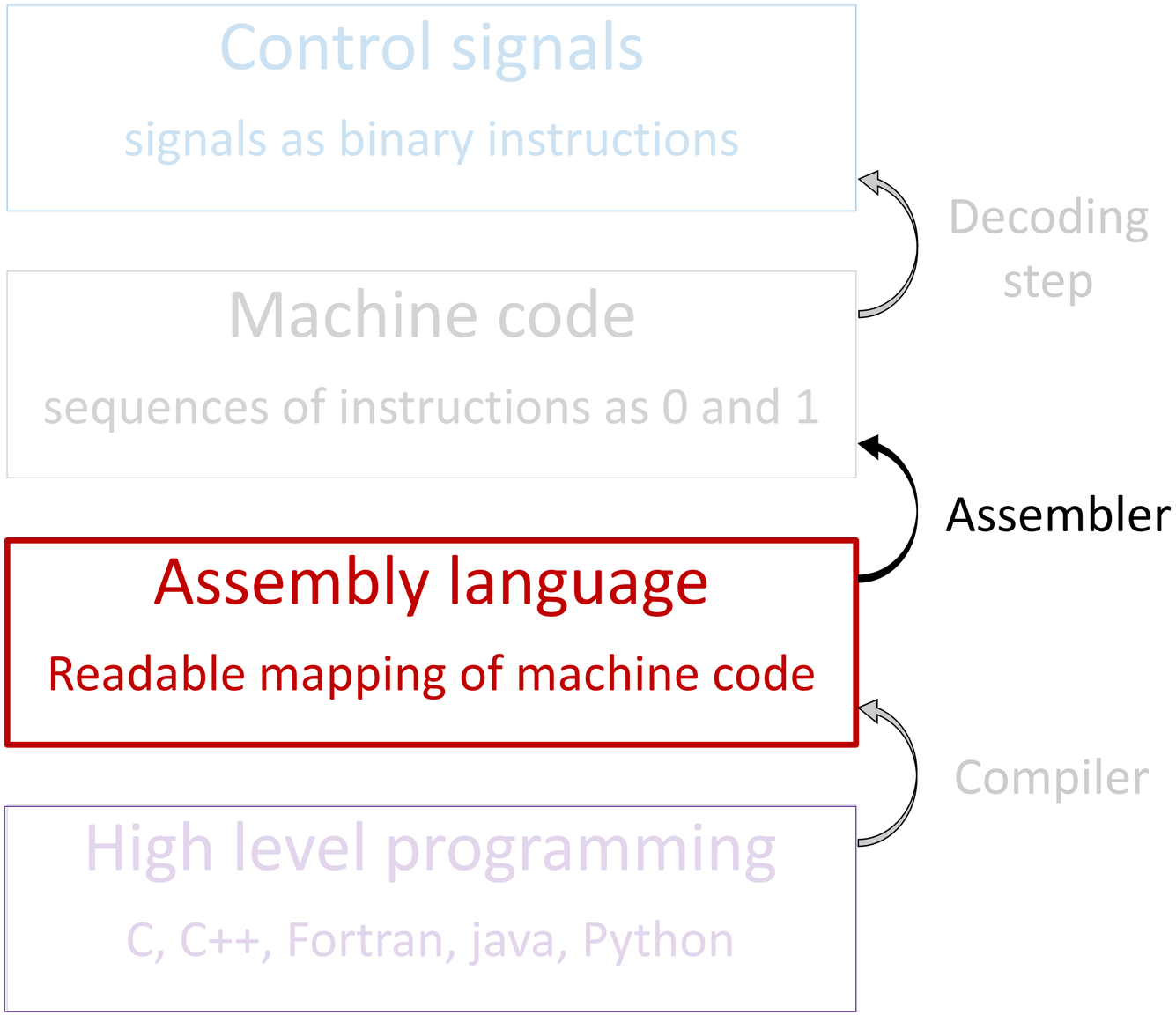
PROGRAM LEVELS




```
10001100|010111 … |00101 …
00000100|011010 … |01100 …
00010100|111000 … |11100 …
00100110|000011 …
```

- **Very** hard to read/write
- Only exact instructions
- Depends on the ISA
  (Instruction Set Architecture)

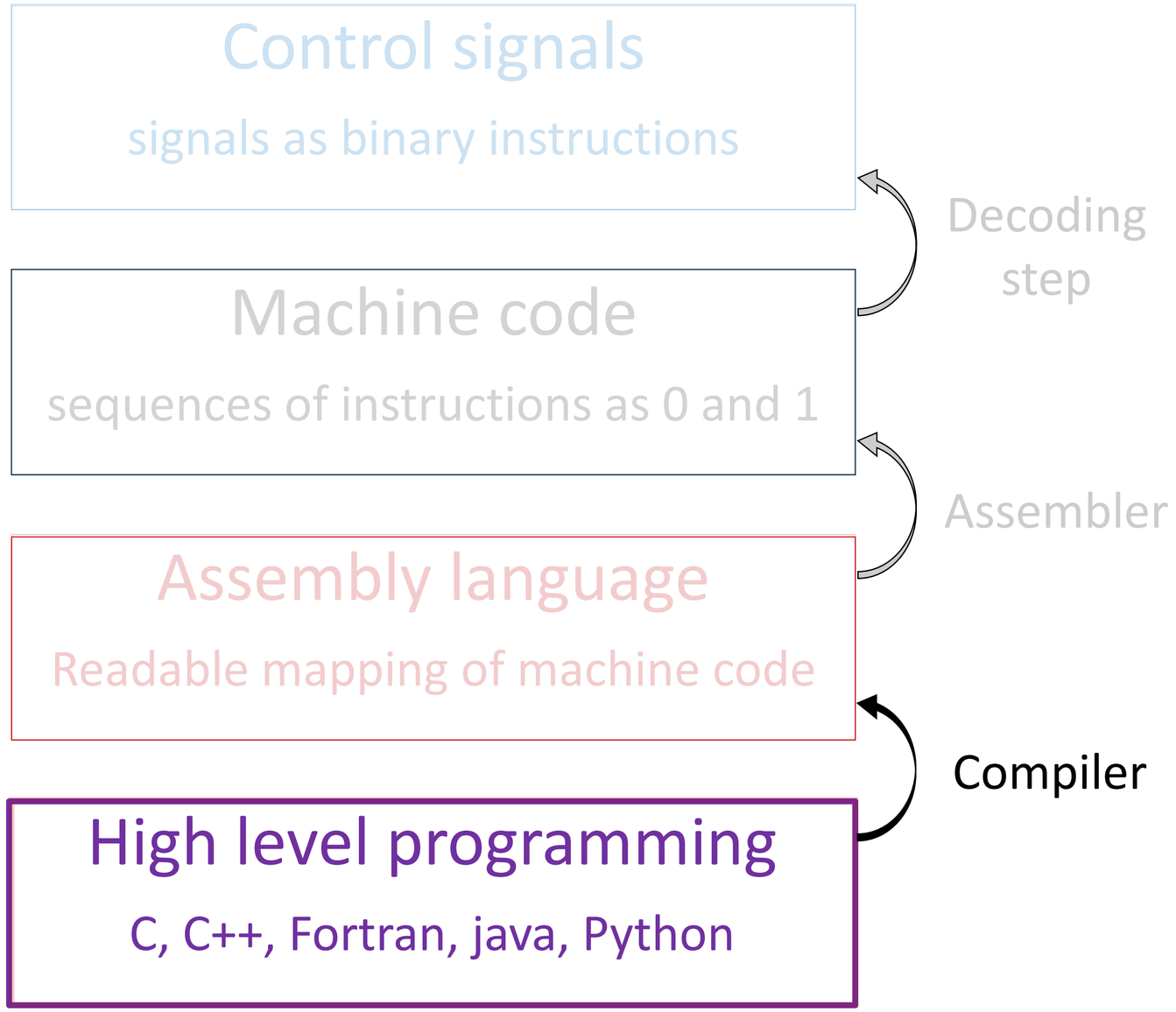
PROGRAM LEVELS




```
  MOV  AX, [BX]      ; move data
label:
  ADD  AX, 8         ; add 8
  CMP  AL, AH        ; compare
  JNE  label         ; jump to
```

- Human readable
- Exact or Low-level instructions
- Depends on the ISA (Instruction Set Architecture)

PROGRAM LEVELS




```
y0 = 2
for x in x_values:
    y = x**2 / 2 + y0
    print("y-value = " + y)
```

- Easy to read/write
- Often high-level instructions
- ISA independent (Instruction Set Architecture)

COMPILATION OR INTERPRETATION

Compilation (C++, Fortran, …)

- Translates high-level programs to assembly / machine code
- Instruction Set Architecture (ISA) specific
- Compiling takes time, required for each ISA
- Can optimize high-level instructions –> for faster execution

Interpretation (Python, MATLAB, …)

- Immediately executable
- Interpreter executes programs “line-by-line”
- Less optimization possible, often slower
- You need the interpreter on your computer

SUMMARY OF COMPUTERS

- Computers
- CPUs
  - Load machine code programs from memory
  - Execute instructions
- Assembly
  - Readable machine code - translates "… 10101 …" into ADD
  - Dependent to the Instruction Set Architecture (ISA)
    - x86, x64, PowerPC, ARM
- Higher programming languages
  - Python, C/C++, Java, …
  - Programs can be compiled or interpreted
  - Independent of the processor architecture and ISA

DATA REPRESENTATION

BINARY, HEXADECIMAL, AND DECIMAL

- **Digital** computers only have 0 and 1 (low and high voltage)
- **Binary** digit or **bit** has value 0 or 1
- Binary number representation:

$$\begin{aligned} 1001_2 &= 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 1^0 \\ &= 8 + 1 \\ &= 9_{10} \end{aligned}$$

- A **Byte** = 8 bits can hold a value of $2^8 = 256$ states
- Possible decimal values in $[0 \,..\, 255]$ where $255 = 2^8 - 1$

$$10000100_2 = 128 + 4 = 132_{10}$$

BINARY, HEXADECIMAL, AND DECIMAL

- **Binary** numbers use 2 digits: {0, 1}
- **Hexadecimal** numbers use 16 digits: {0, 1, .. , 9, A, B, C, D, E, F}

$$\begin{aligned} 1BA_{16} &= 1 \cdot 16^2 + B \cdot 16^1 + A \cdot 16^0 \\ &= 256 + 176 + 10 \\ &= 342_{10} \end{aligned}$$

- A hexadecimal digit bundles 4 binary digits

$$D9_{16} = 1101\ 1001_2$$

- Alternative writing $D9_{16} = 0\text{x}00\text{D}9$ for 16-bits

BINARY NUMBERS & OPERATIONS

- Highest (max. representable) number in 8, 16, 32, or 64 bits ?
- **Negative** numbers ?
- What with **real** numbers and very large/small numbers ?
- **Complex** numbers ?
- How to perform additions, logic comparisons between numbers ?

MAXIMUM REPRESENTABLE INTEGER IN N-BIT

- Highest (max. representable) number in 8, 16, 32, or 64 bits ?
- For N bits the maximum number is $M = 2^N - 1$

$$M_{8\text{bit}} = 255$$
$$M_{16\text{bit}} = 65{,}535$$
$$M_{32\text{bit}} = 4{,}294{,}967{,}295$$
$$M_{64\text{bit}} = 1.84467440737 \times 10^{19}$$

- Use more bits for larger numbers

NEGATIVE INTEGERS

- How to represent negative numbers ?
- One bit to express sign of the number
- Most used representation: **2’s complement**
- Negative number -i = NOT(i) + 1

$$
\begin{aligned}
-3 &= \text{NOT}(3) + 1 \\
&= \text{NOT}(0011_2) + 1 \\
&= 1100_2 + 1 \\
&= 1101_2
\end{aligned}
$$

8-bit example

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

FLOATING POINT NUMBERS

- How to represent real numbers ? in scientific notation !
- More parts:
  - S = sign-bit, 1 bit
  - E = exponent, 8 bits
  - M = mantissa, 23 bits

- Formula to calculate real number x:

$$x = (-1)^S \times M \times 2^{E-127}$$

- Smallest positive nonzero number is $2^{-127} \approx 5.88 \times 10^{-39}$
- Largest number approx. $\pm 2.85 \times 10^{45}$

CHARACTER REPRESENTATION: ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

SUMMARY OF DATA REPRESENTATION

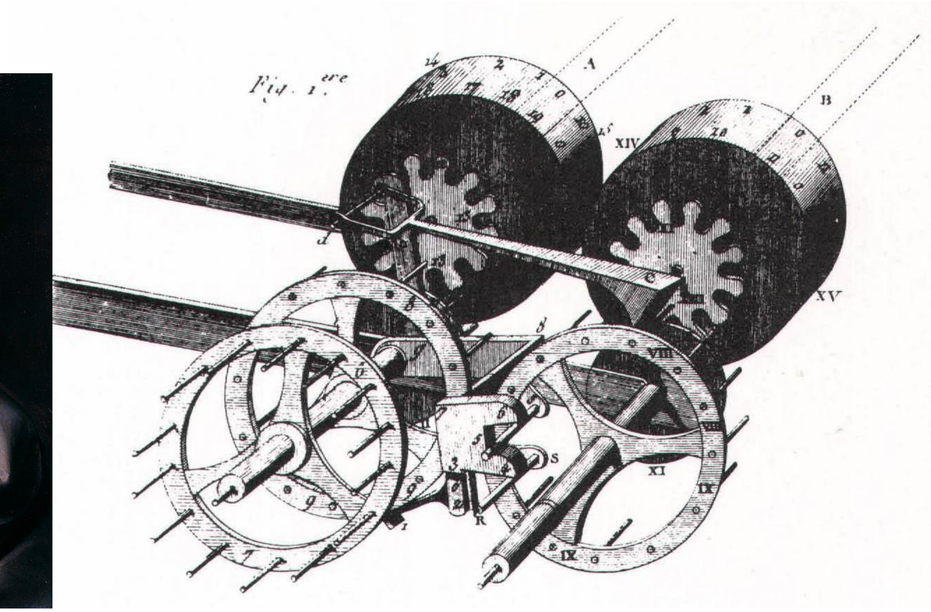
- Numbers
  - Boolean
  - (un)signed integers
  - floating point, scientific notation
- Text
  - Characters
  - String is an array of characters
- datatype mapping to binary numbers

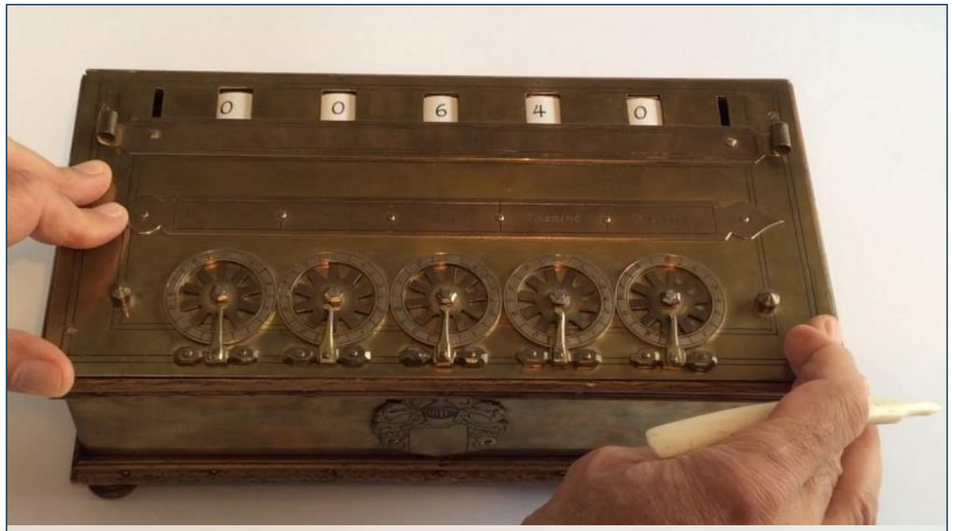
HISTORY OF THE COMPUTER

HISTORY OF THE COMPUTER: MECHANICAL

Pascaline by Blaise Pascal in 1642

- Calculator
- Addition and subtraction





HISTORY OF THE COMPUTER: MECHANICAL

Difference Engine #1 of Charles Babbage in 1832

- Calculation of tables of squares, cubes, logarithms

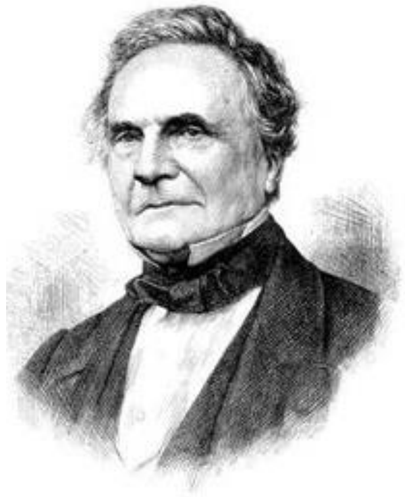
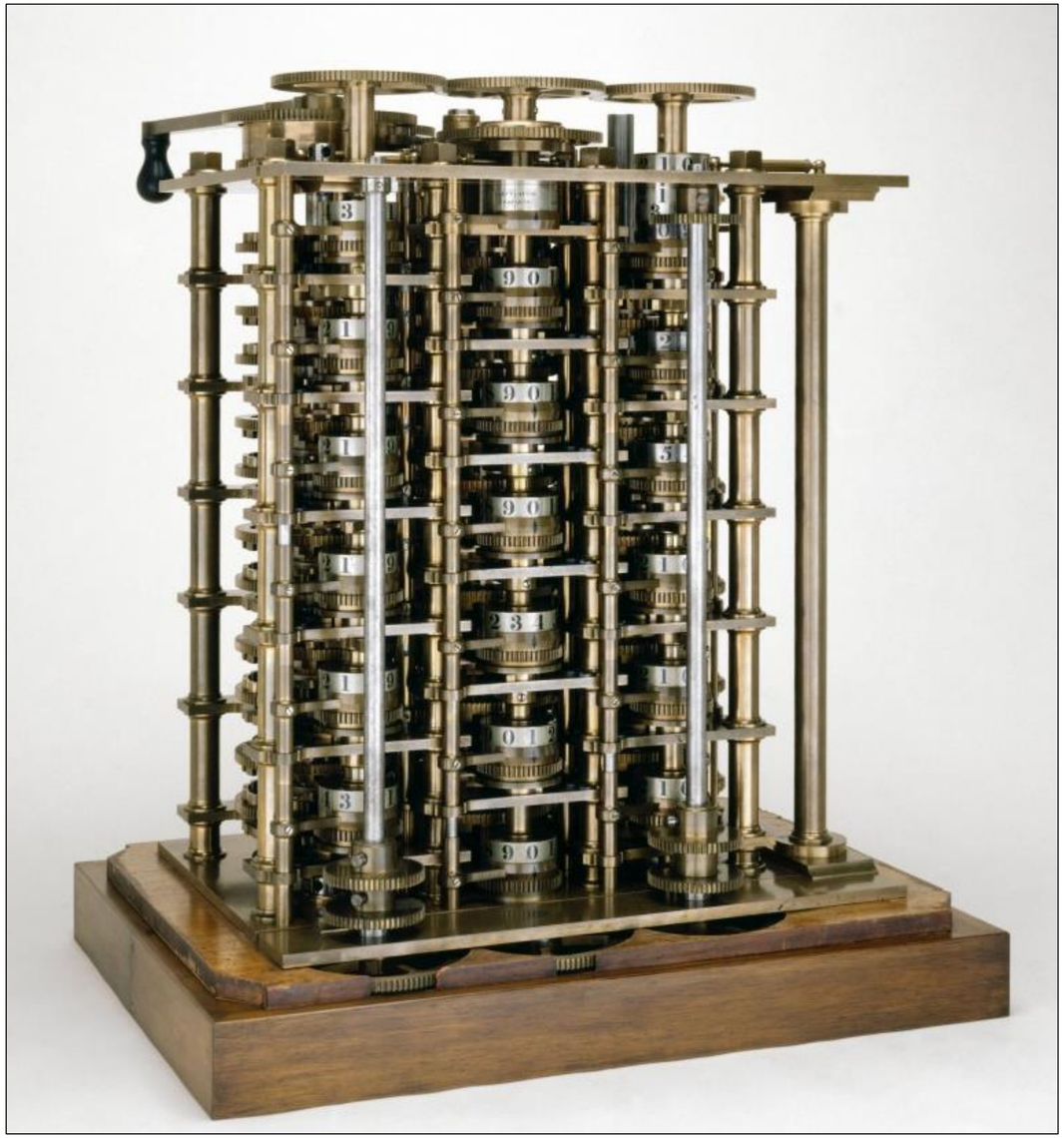


TABLE OF CUBE NUMBERS.

Natural Numbers.	Orders of differences.			Tabular number.
	Third difference.	Second difference.	First difference.	Cubes.
In upper left hand corner of machine.	Bottom wheel on central axis.	Lower wheels on left hand axis.	Wheel on central axis.	Wheels on right hand axis.
1	6	6	1	1
2	6	12	7	8
3	6	18	19	27
4	6	24	37	64
5	6	30	61	125
6	6	36	91	216



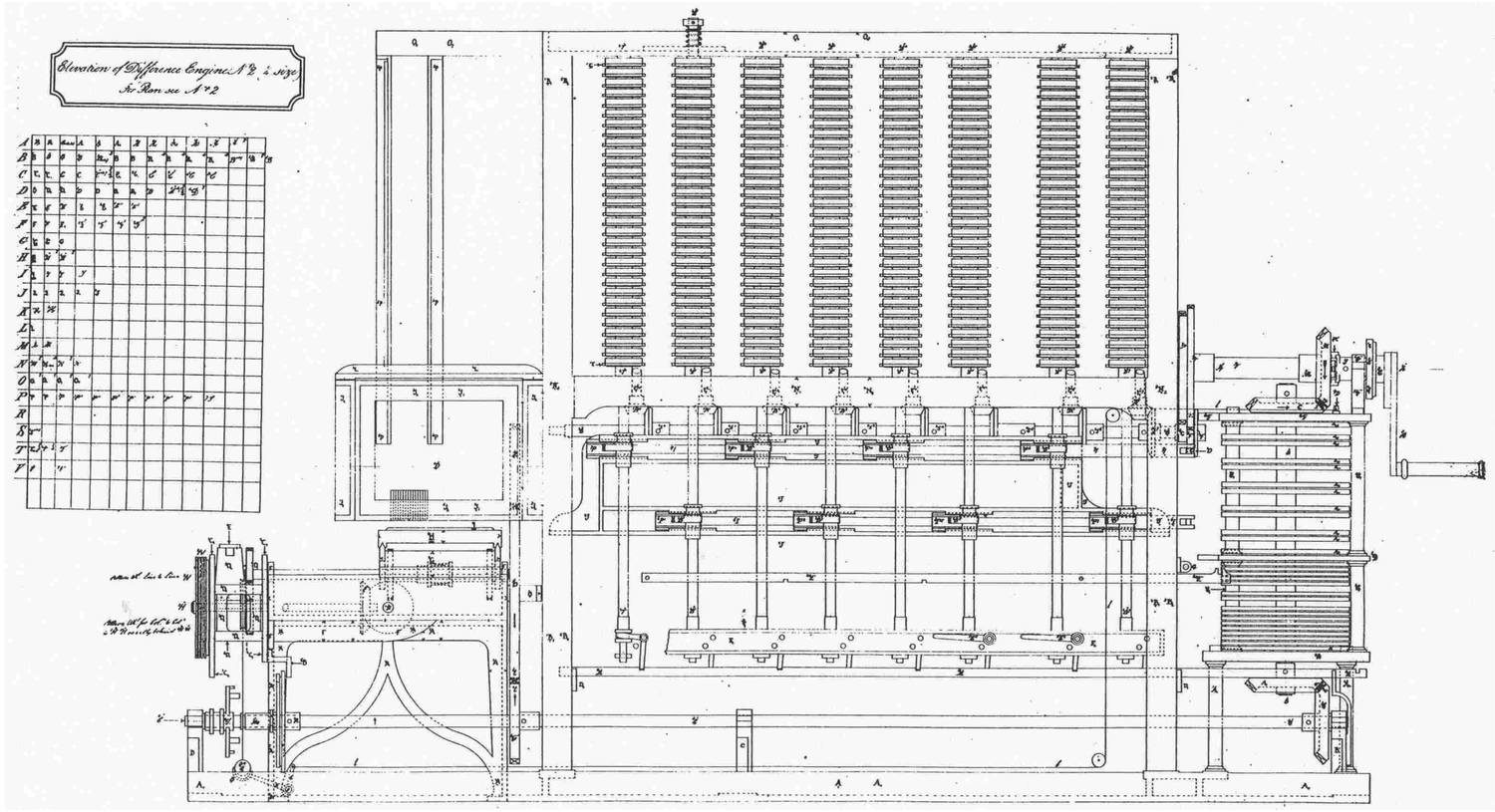
HISTORY OF THE COMPUTER: MECHANICAL

Difference Engine #2
of Charles Babbage in 1857

- Built unfinished
- Built afterwards at museum

Analytical Engine

- Only in design (1842)
- General purpose computer



Difference Engine #2 design (overview)

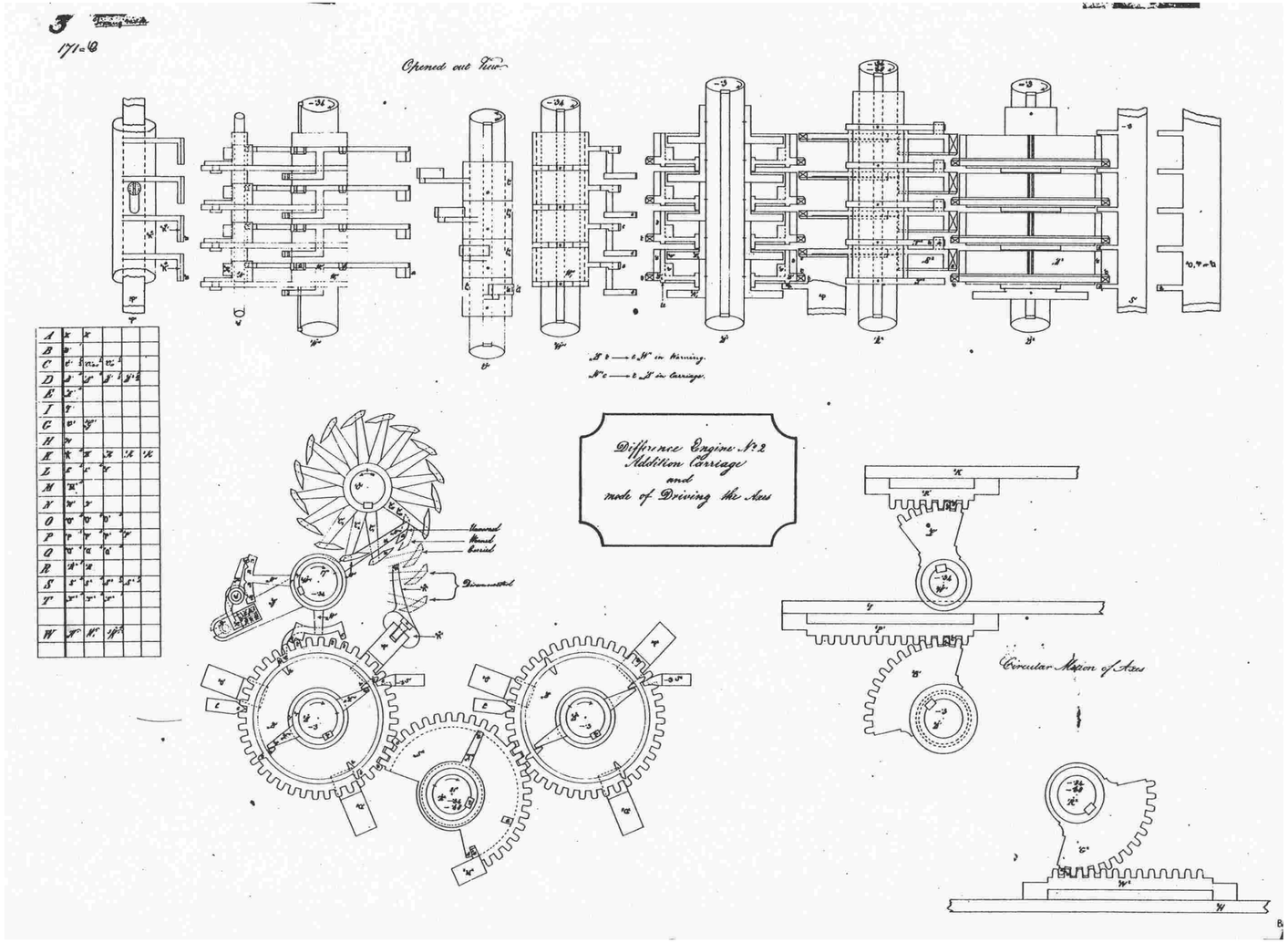
HISTORY OF THE COMPUTER: MECHANICAL

Difference Engine #2 of Charles Babbage in 1857

- Built unfinished
- Built afterwards at museum

Analytical Engine

- Only in design (1842)
- General purpose computer



Difference Engine #2 (addition and carry)

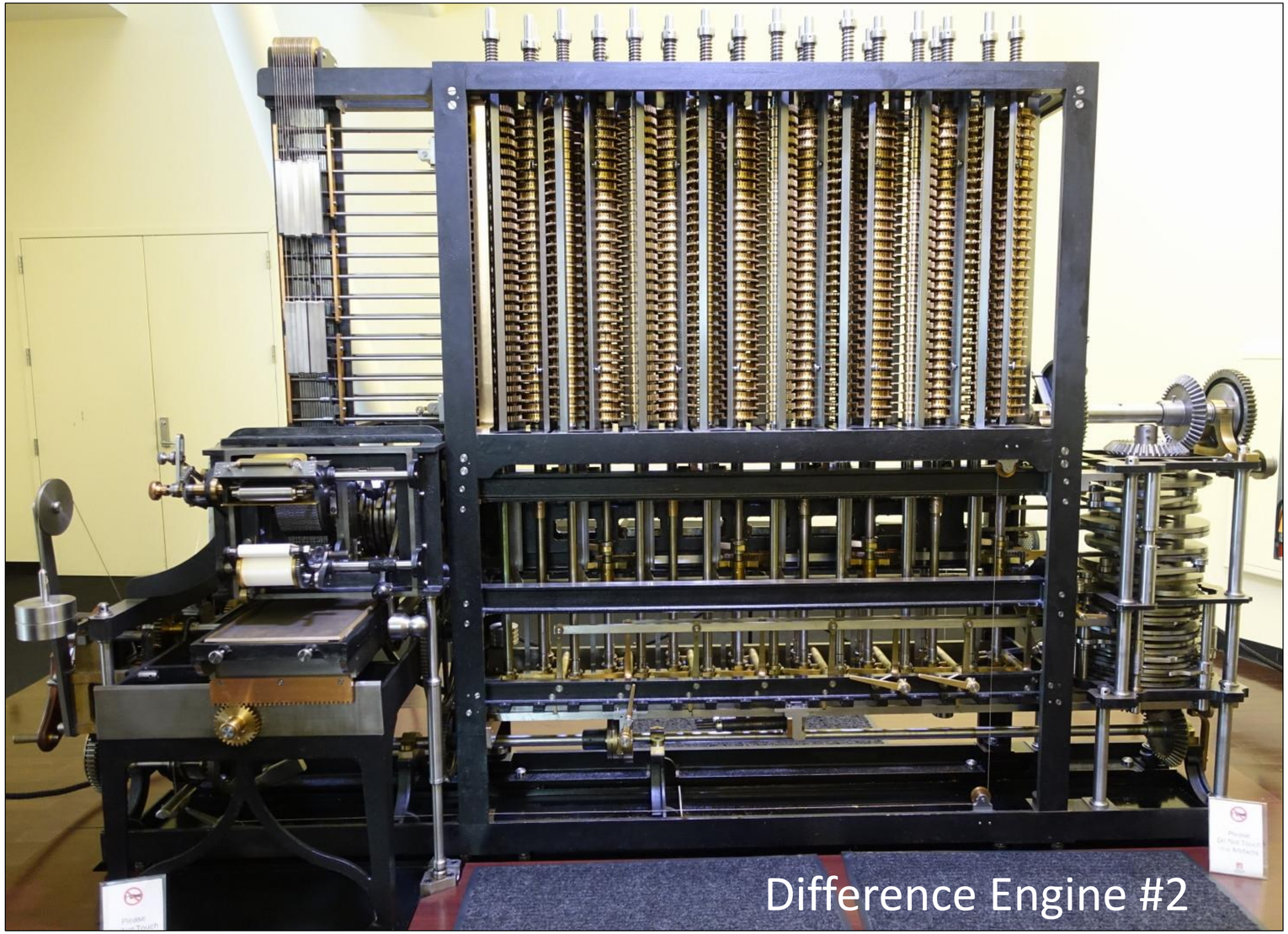
HISTORY OF THE COMPUTER: MECHANICAL

Difference Engine #2 of Charles Babbage in 1857

- Built unfinished
- Built afterwards at museum

Analytical Engine

- Only in design (1842)
- General purpose computer


Difference Engine #2

HISTORY OF THE COMPUTER: MECHANICAL

Note on the Analytical Engine by Ada Lovelace (1843)

- **Program** (to calculate Bernoulli numbers)
- Shows its general-purpose design

Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.	Data. $^1V_1$ 1	Data. $^1V_2$ 2	Data. $^1V_3$ n	Working Variables. $^0V_4$	$^0V_5$	$^0V_6$	$^0V_7$	$^0V_8$	$^0V_9$	$^0V_{10}$	$^0V_{11}$	$^0V_{12}$	$^0V_{13}$	Result Variables. $^1V_{21}$ B_1	$^1V_{22}$ B_2	$^1V_{23}$ B_5	$^0V_{24}$ ... B_7
1	×	$^1V_2 \times ^1V_3$	$^1V_4, ^1V_5, ^1V_6$	$^1V_2 = ^1V_2$, $^1V_3 = ^1V_3$	$= 2n$	...	2	n	$2n$	$2n$	$2n$											
2	−	$^1V_4 - ^1V_1$	$^2V_4$	$^1V_4 = ^2V_4$, $^1V_1 = ^1V_1$	$= 2n-1$	1	...	...	$2n-1$													
3	+	$^1V_5 + ^1V_1$	$^2V_5$	$^1V_5 = ^2V_5$, $^1V_1 = ^1V_1$	$= 2n+1$	1	...	...	...	$2n+1$												
4	÷	$^2V_5 \div ^2V_4$	$^1V_{11}$	$^2V_5 = ^0V_5$, $^2V_4 = ^0V_4$	$= \frac{2n-1}{2n+1}$	...	...	...	0	0	...	...	...	...	...	$\frac{2n-1}{2n+1}$						
5	÷	$^1V_{11} \div ^1V_2$	$^2V_{11}$	$^1V_{11} = ^2V_{11}$, $^1V_2 = ^1V_2$	$= \frac{1}{2} \cdot \frac{2n-1}{2n+1}$	...	2	...	...	...	...	...	...	...	...	$\frac{1}{2} \cdot \frac{2n-1}{2n+1}$						
6	−	$^0V_{13} - ^2V_{11}$	$^1V_{13}$	$^2V_{11} = ^0V_{11}$, $^0V_{13} = ^1V_{13}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$	...	...	...	...	...	...	...	...	...	...	0	.....	$-\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$				
7	−	$^1V_3 - ^1V_1$	$^1V_{10}$	$^1V_3 = ^1V_3$, $^1V_1 = ^1V_1$	$= n-1\ (=3)$	1	...	n	...	...	...	...	...	...	$n-1$							
8	+	$^1V_2 + ^0V_7$	$^1V_7$	$^1V_2 = ^1V_2$, $^0V_7 = ^1V_7$	$= 2+0=2$	...	2	...	...	...	...	2										
9	÷	$^1V_6 \div ^1V_7$	$^3V_{11}$	$^1V_6 = ^1V_6$, $^0V_{11} = ^3V_{11}$	$= \frac{2n}{2} = A_1$	...	...	...	...	...	$2n$	2	...	...	...	$\frac{2n}{2} = A_1$						
10	×	$^1V_{21} \times ^3V_{11}$	$^1V_{12}$	$^1V_{21} = ^1V_{21}$, $^3V_{11} = ^3V_{11}$	$= B_1 \cdot \frac{2n}{2} = B_1 A_1$	...	...	...	...	...	...	...	...	...	...	$\frac{2n}{2} = A_1$	$B_1 \cdot \frac{2n}{2} = B_1 A_1$	.....	B_1			
11	+	$^1V_{12} + ^1V_{13}$	$^2V_{13}$	$^1V_{12} = ^0V_{12}$, $^1V_{13} = ^2V_{13}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}$ ...	...	...	...	...	...	...	...	...	...	...	.....	0	$\left\{-\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}\right\}$				
12	−	$^1V_{10} - ^1V_1$	$^2V_{10}$	$^1V_{10} = ^2V_{10}$, $^1V_1 = ^1V_1$	$= n-2\ (=2)$	1	...	...	...	...	...	...	...	...	$n-2$							
13		$^1V_6 - ^1V_1$	$^2V_6$	$^1V_6 = ^2V_6$, $^1V_1 = ^1V_1$	$= 2n-1$	1	...	...	...	...	$2n-1$											
14		$+ ^1V_1 + ^1V_7$	$^2V_7$	$^1V_1 = ^1V_1$, $^1V_7 = ^2V_7$	$= 2+1=3$	1	...	...	...	...	...	3										
15		$\div ^2V_6 \div ^2V_7$	$^1V_8$	$^2V_6 = ^2V_6$, $^2V_7 = ^2V_7$	$= \frac{2n-1}{3}$	...	...	...	...	...	$2n-1$	3	$\frac{2n-1}{3}$									
16		$\times ^1V_8 \times ^3V_{11}$	$^4V_{11}$	$^1V_8 = ^0V_8$, $^3V_{11} = ^4V_{11}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3}$	...	...	...	...	...	...	...	0	...	...	$\frac{2n}{2} \cdot \frac{2n-1}{3}$						
17		$^2V_6 - ^1V_1$	$^3V_6$	$^2V_6 = ^3V_6$, $^1V_1 = ^1V_1$	$= 2n-2$	1	...	...	...	...	$2n-2$											
18		$+ ^1V_1 + ^2V_7$	$^3V_7$	$^2V_7 = ^3V_7$, $^1V_1 = ^1V_1$	$= 3+1=4$	1	...	...	...	...	...	4										
19		$\div ^3V_6 \div ^3V_7$	$^1V_9$	$^3V_6 = ^3V_6$, $^3V_7 = ^3V_7$	$= \frac{2n-2}{4}$	...	...	...	...	...	$2n-2$	4	...	$\frac{2n-2}{4}$	...	$\left\{\frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3}\right\} = A_3$							
20		$\times ^1V_9 \times ^4V_{11}$	$^5V_{11}$	$^1V_9 = ^0V_9$, $^4V_{11} = ^5V_{11}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{4} = A_3$	...	...	...	...	...	...	...	...	0	...							
21	×	$^1V_{22} \times ^5V_{11}$	$^0V_{12}$	$^1V_{22} = ^1V_{22}$, $^0V_{12} = ^2V_{12}$	$= B_3 \cdot \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3} = B_3 A_3$	...	...	...	...	...	...	...	...	...	...	0	$B_3 A_3$	.....	.....	B_3		
22	+	$^2V_{12} + ^2V_{13}$	$^3V_{13}$	$^2V_{12} = ^0V_{12}$, $^2V_{13} = ^3V_{13}$	$= A_0 + B_1 A_1 + B_3 A_3$	...	...	...	...	...	...	...	...	...	...	.....	0	$\{A_3 + B_1 A_1 + B_2 A_2\}$				
23	−	$^2V_{10} - ^1V_1$	$^3V_{10}$	$^2V_{10} = ^3V_{10}$, $^1V_1 = ^1V_1$	$= n-3\ (=1)$	1	...	...	...	...	...	...	...	...	$n-3$							
Here follows a repetition of Operations thirteen to twenty-three.																						
24		$+ ^4V_{13} + ^0V_{24}$	$^1V_{24}$	$^4V_{13} = ^0V_{13}$, $^0V_{24} = ^1V_{24}$	$= B_7$	...	...	...	...	...	...	...	...	...	...	.....	.....	.....	.....	.....	.....	B_7
25		$+ ^1V_1 + ^1V_3$	$^1V_3$	$^1V_1 = ^1V_1$, $^1V_3 = ^0V_6$, $^4V_6 = ^0V_7$, $^5V_7 = ^0V_7$	$= n+1 = 4+1 = 5$ ..... by a Variable-card. by a Variable card.	1	...	$n+1$	...	...	0	0										

HISTORY OF THE COMPUTER: MECHANICAL

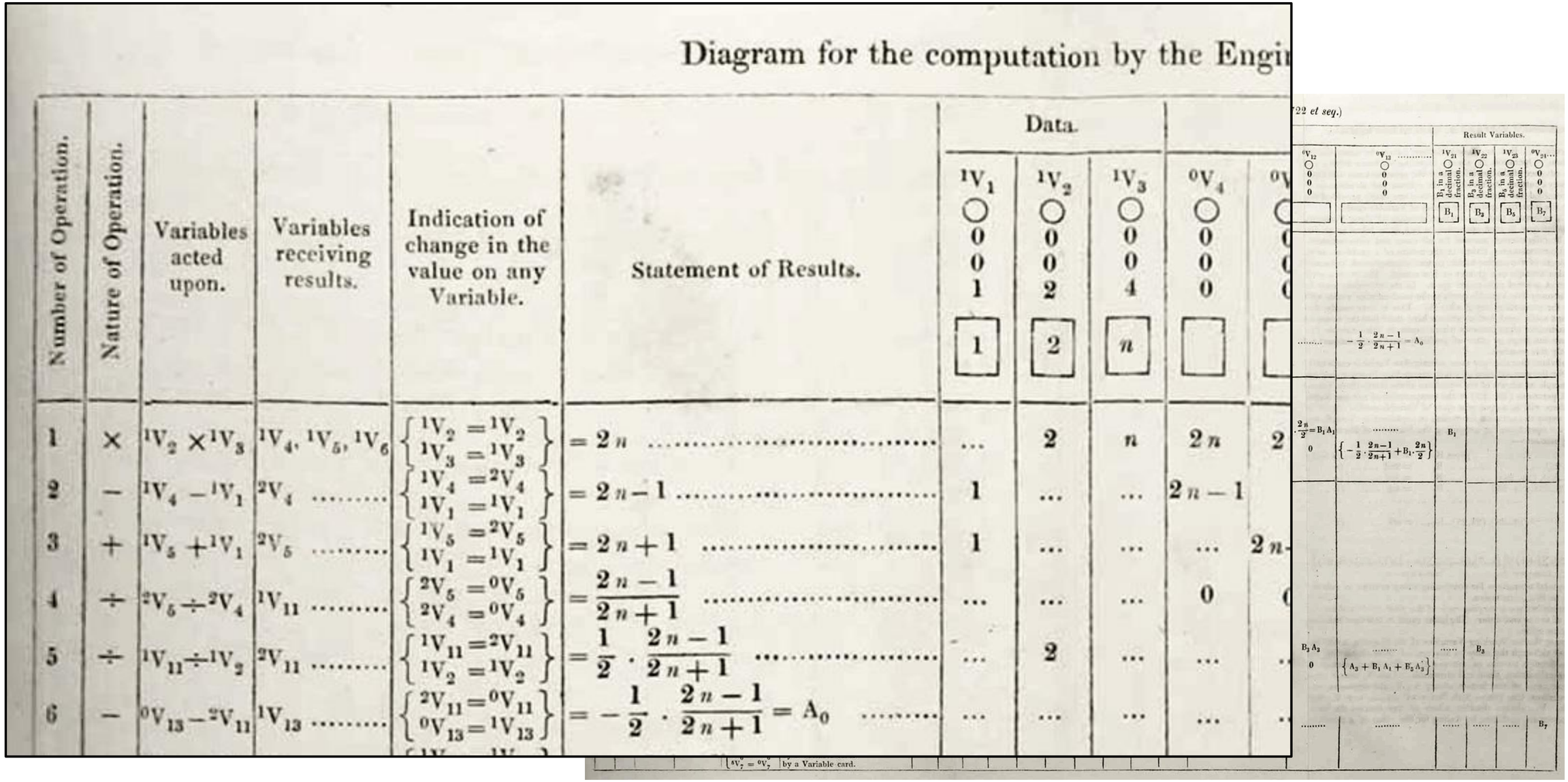


Diagram for the computation by the Engi

Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.	Data. $^1V_1$	Data. $^1V_2$	Data. $^1V_3$	Data. $^0V_4$	Data. ^0V
						0 0 1	0 0 2	0 0 4	0 0 0	0 0 0
						1	2	n		
1	×	$^1V_2 \times {}^1V_3$	$^1V_4, {}^1V_5, {}^1V_6$	$^1V_2 = {}^1V_2$, $^1V_3 = {}^1V_3$	$= 2n$	...	2	n	$2n$	2
2	−	$^1V_4 - {}^1V_1$	$^2V_4$	$^1V_4 = {}^2V_4$, $^1V_1 = {}^1V_1$	$= 2n-1$	1	...	...	$2n-1$	
3	+	$^1V_5 + {}^1V_1$	$^2V_5$	$^1V_5 = {}^2V_5$, $^1V_1 = {}^1V_1$	$= 2n+1$	1	...	...	...	$2n$
4	÷	$^2V_5 \div {}^2V_4$	$^1V_{11}$	$^2V_5 = {}^0V_5$, $^2V_4 = {}^0V_4$	$= \frac{2n-1}{2n+1}$	...	...	...	0	0
5	÷	$^1V_{11} \div {}^1V_2$	$^2V_{11}$	$^1V_{11} = {}^2V_{11}$, $^1V_2 = {}^1V_2$	$= \frac{1}{2} \cdot \frac{2n-1}{2n+1}$	...	2	...	...	...
6	−	$^0V_{13} - {}^2V_{11}$	$^1V_{13}$	$^2V_{11} = {}^0V_{11}$, $^0V_{13} = {}^1V_{13}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$	...	...	...	...	...

(722 *et seq.*)

Result Variables.

$^0V_{12}$	$^0V_{13}$	$^1V_{21}$	$^1V_{22}$	$^1V_{23}$	$^0V_{24}$ ...
0 0 0 0	0 0 0 0	B_1 in a decimal fraction.	B_2 in a decimal fraction.	B_5 in a decimal fraction.	0 0 0 0
		B_1	B_2	B_5	B_7
.....	$-\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$				
$\frac{2n}{2} = B_1 A_1$ 0	..... $\left\{-\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}\right\}$	B_1			
$B_2 A_2$ 0	..... $\left\{A_2 + B_1 A_1 + B_2 A_2\right\}$	.....	B_2		
.....	.....	.....	.....	.....	B_7

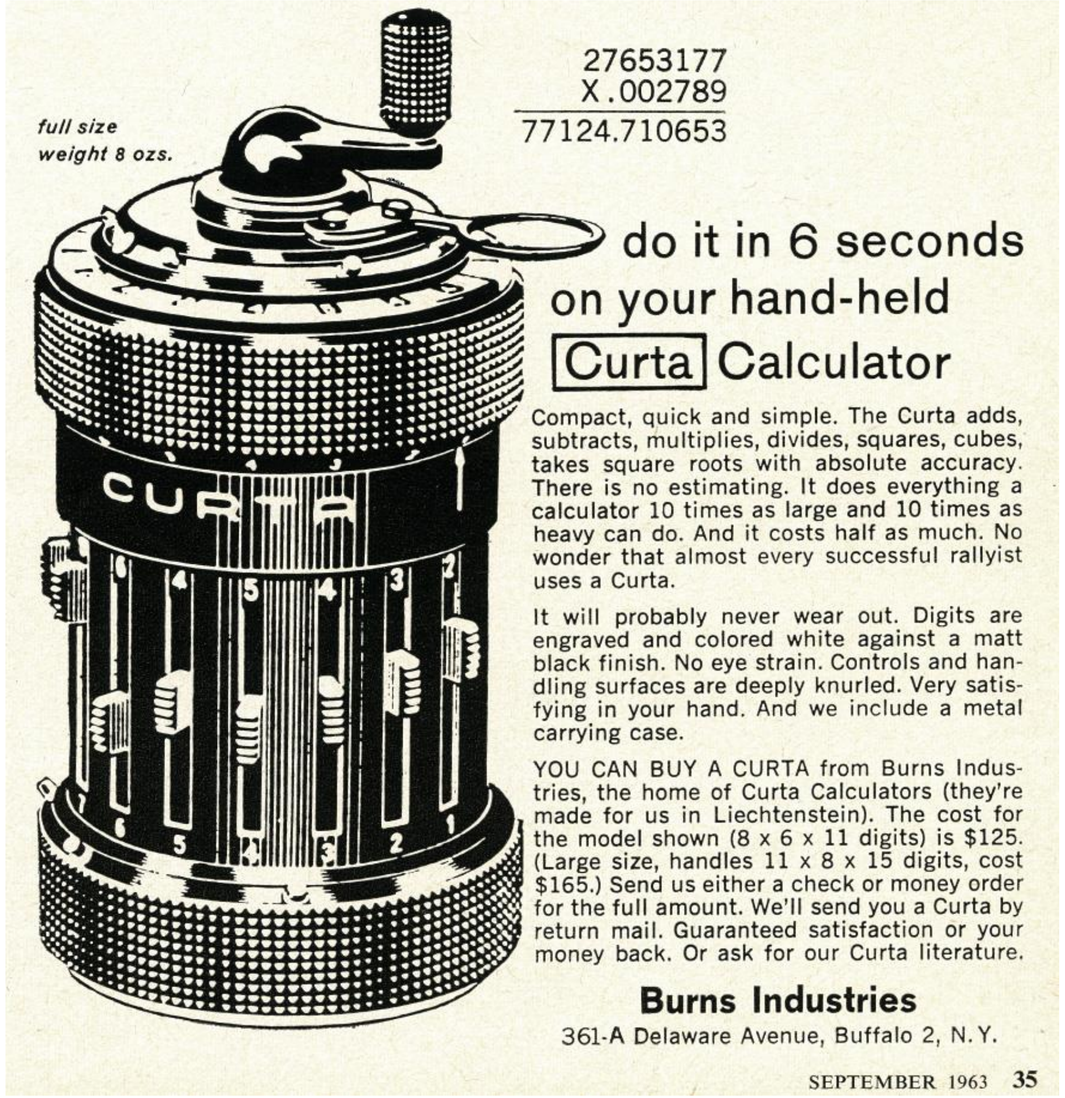
$\{{}^5V_7 = {}^0V_7\}$ by a Variable card.

HISTORY OF THE COMPUTER: MECHANICAL

Curta by Curt Herzstark

- Designed in 1943

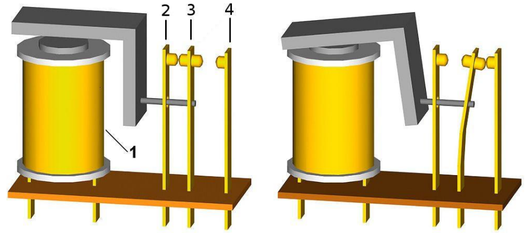
Adv. (1963)

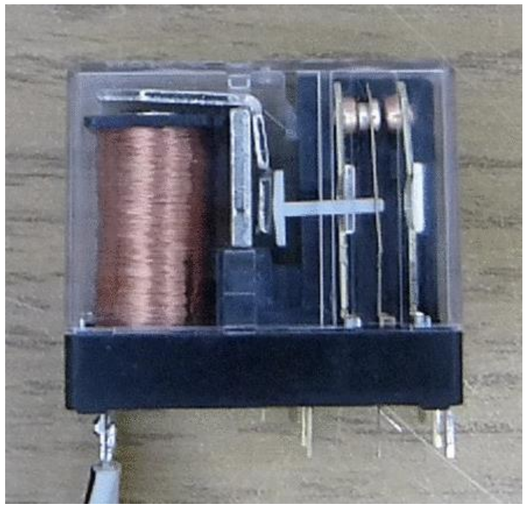


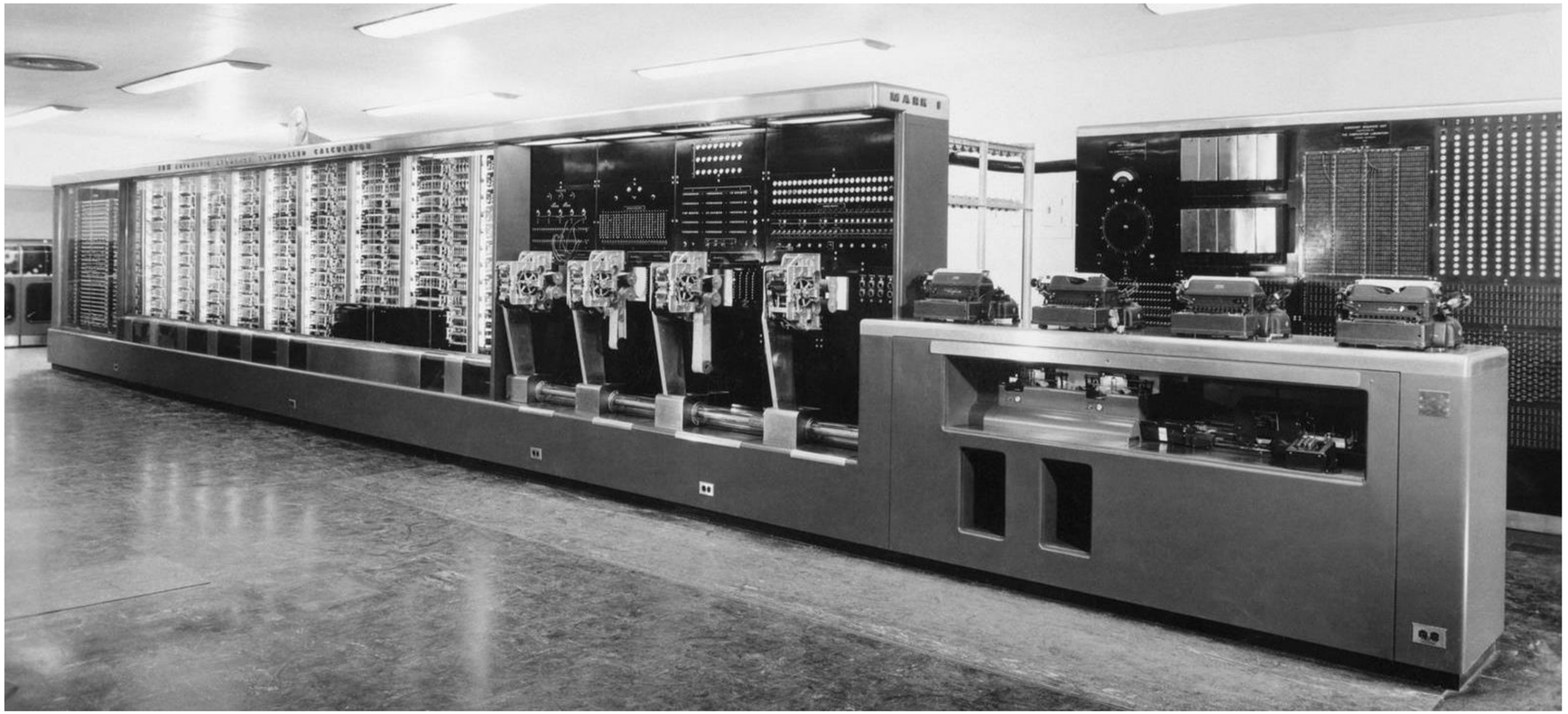
HISTORY OF THE COMPUTER: ELECTRO-MECHANICAL

Harvard Mark I in 1944

- Based or **relay switches**, slower than vacuum tubes



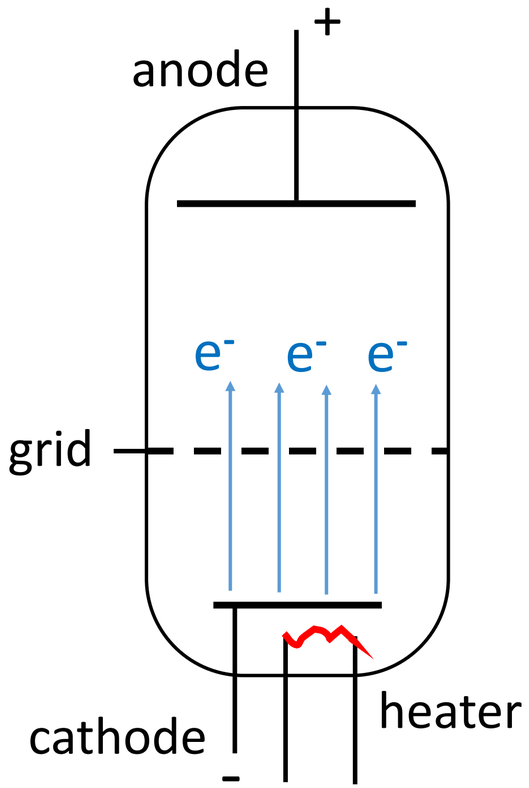


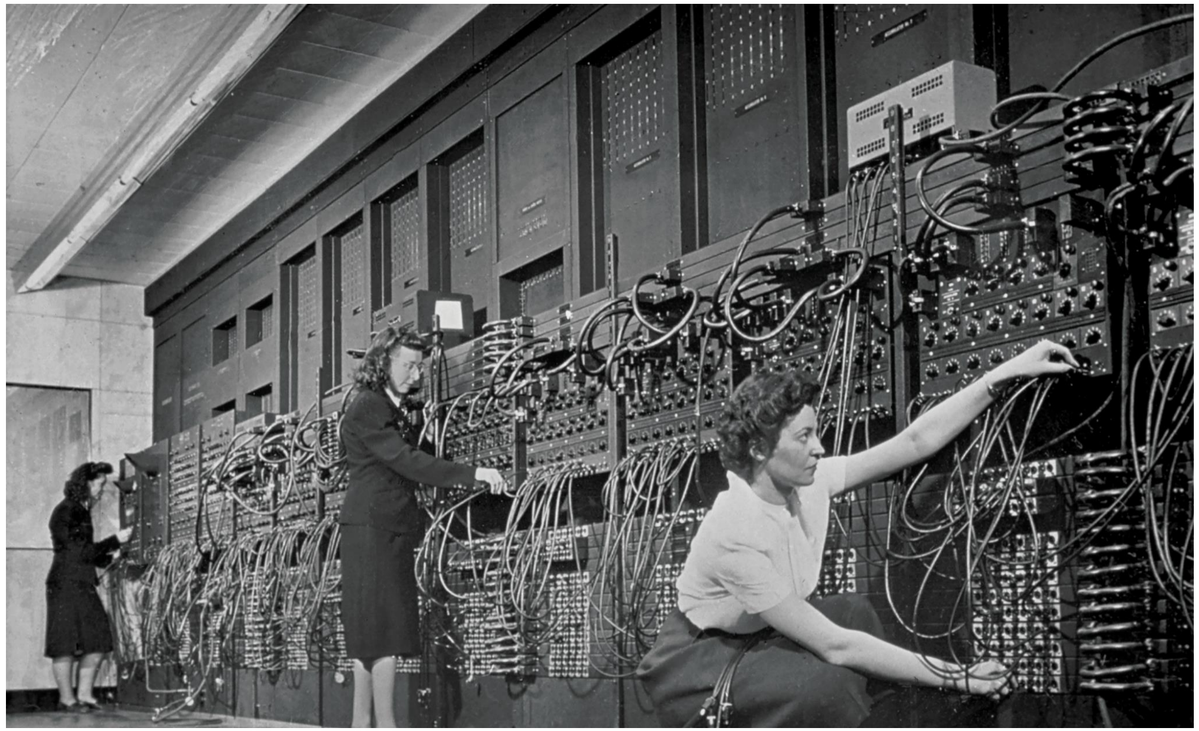


HISTORY OF THE COMPUTER: ELECTRONIC

ENIAC in 1943

- **Vacuum tubes** as electronic switches
- Manually programming by rewiring

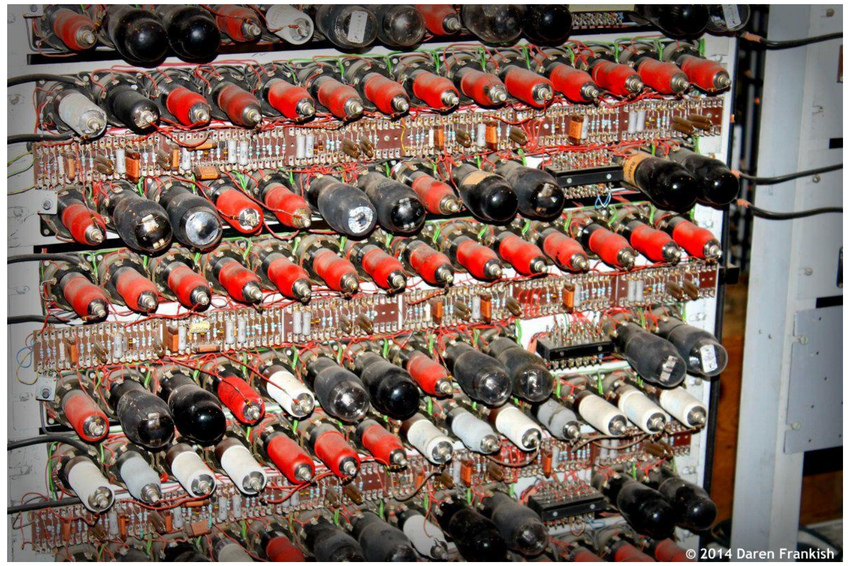


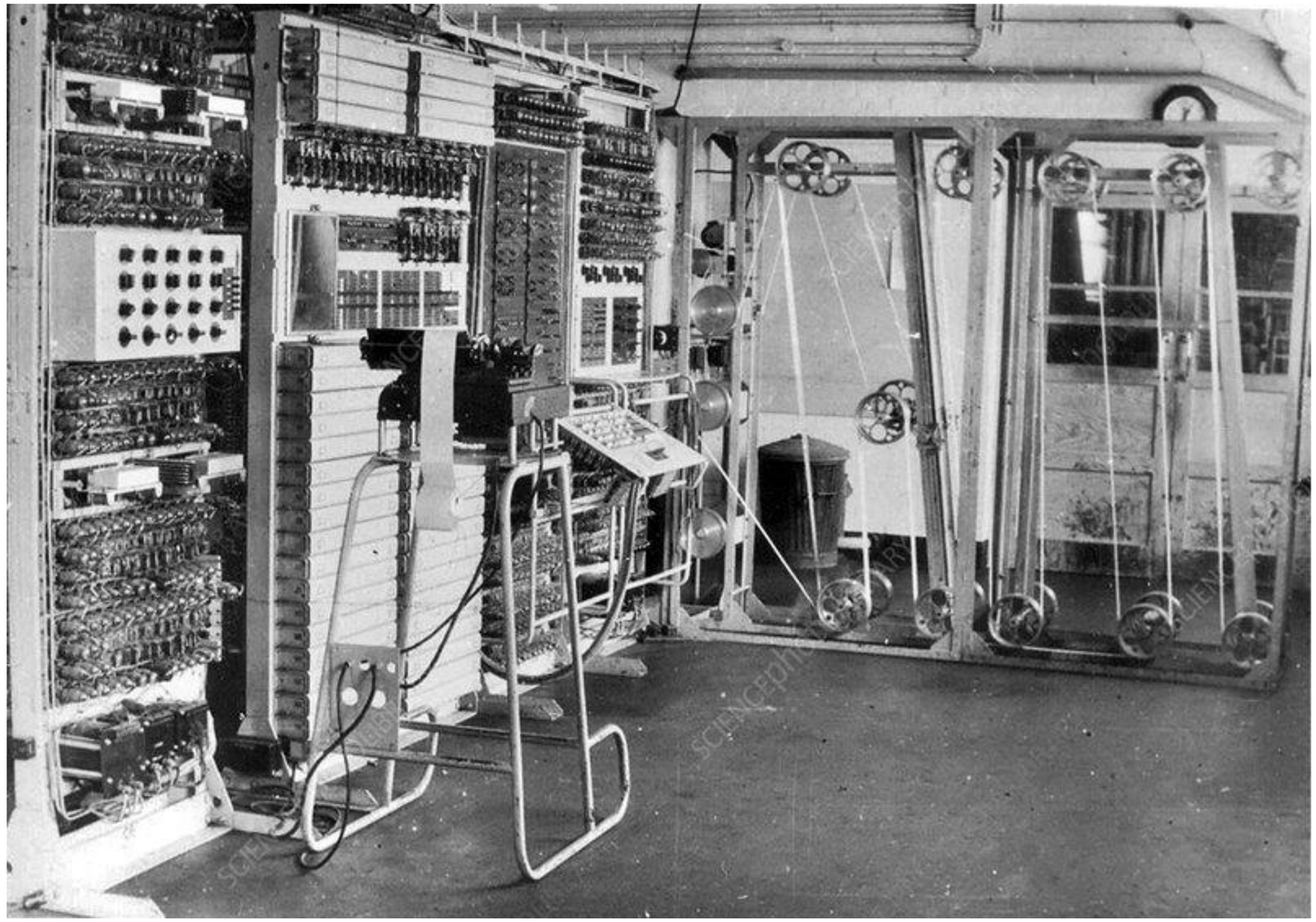


HISTORY OF THE COMPUTER: ELECTRONIC

Colossus in 1944

- **Vacuum tubes**
- Code decryption during WW2

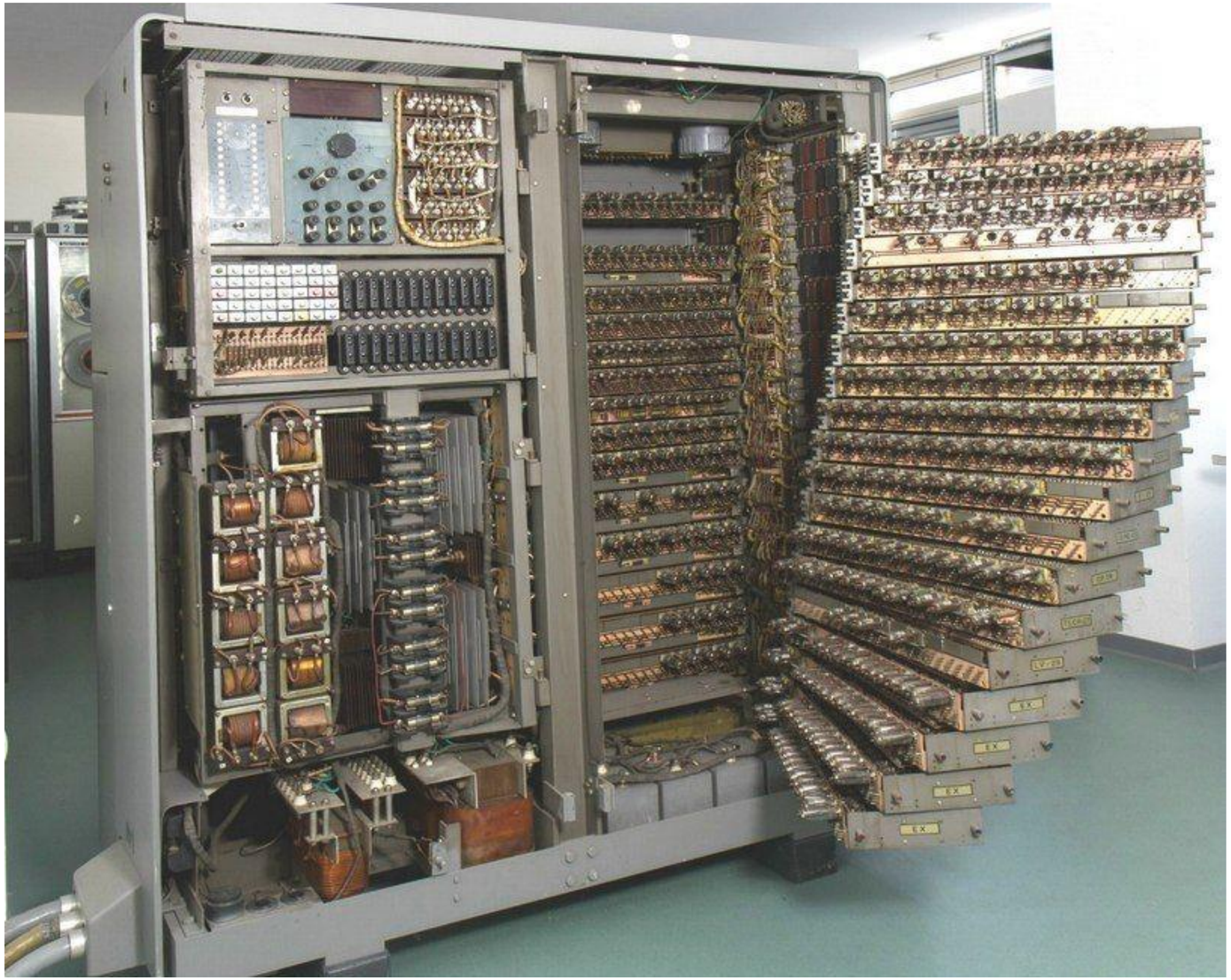




HISTORY OF THE COMPUTER: ELECTRONIC

Bull Gamma 3 in 1952

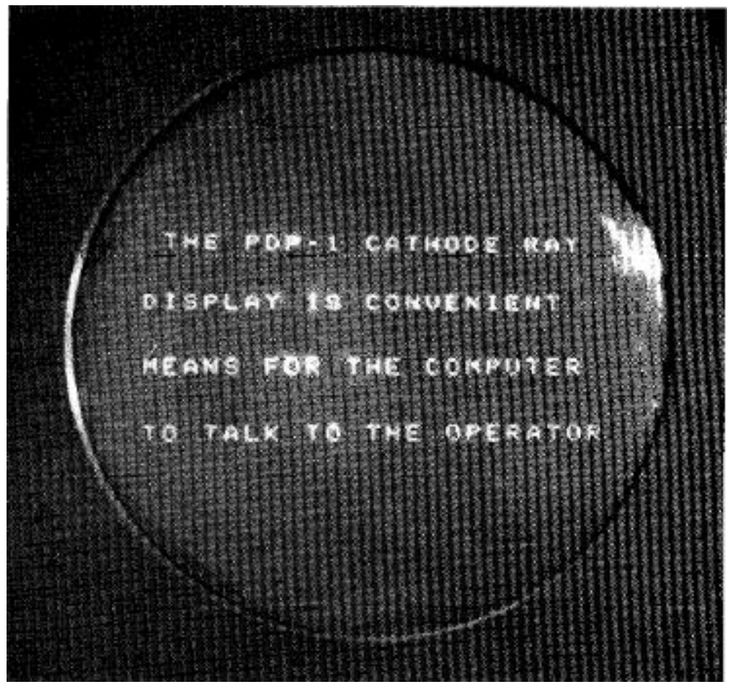
- **Vacuum tubes**
- Commercial

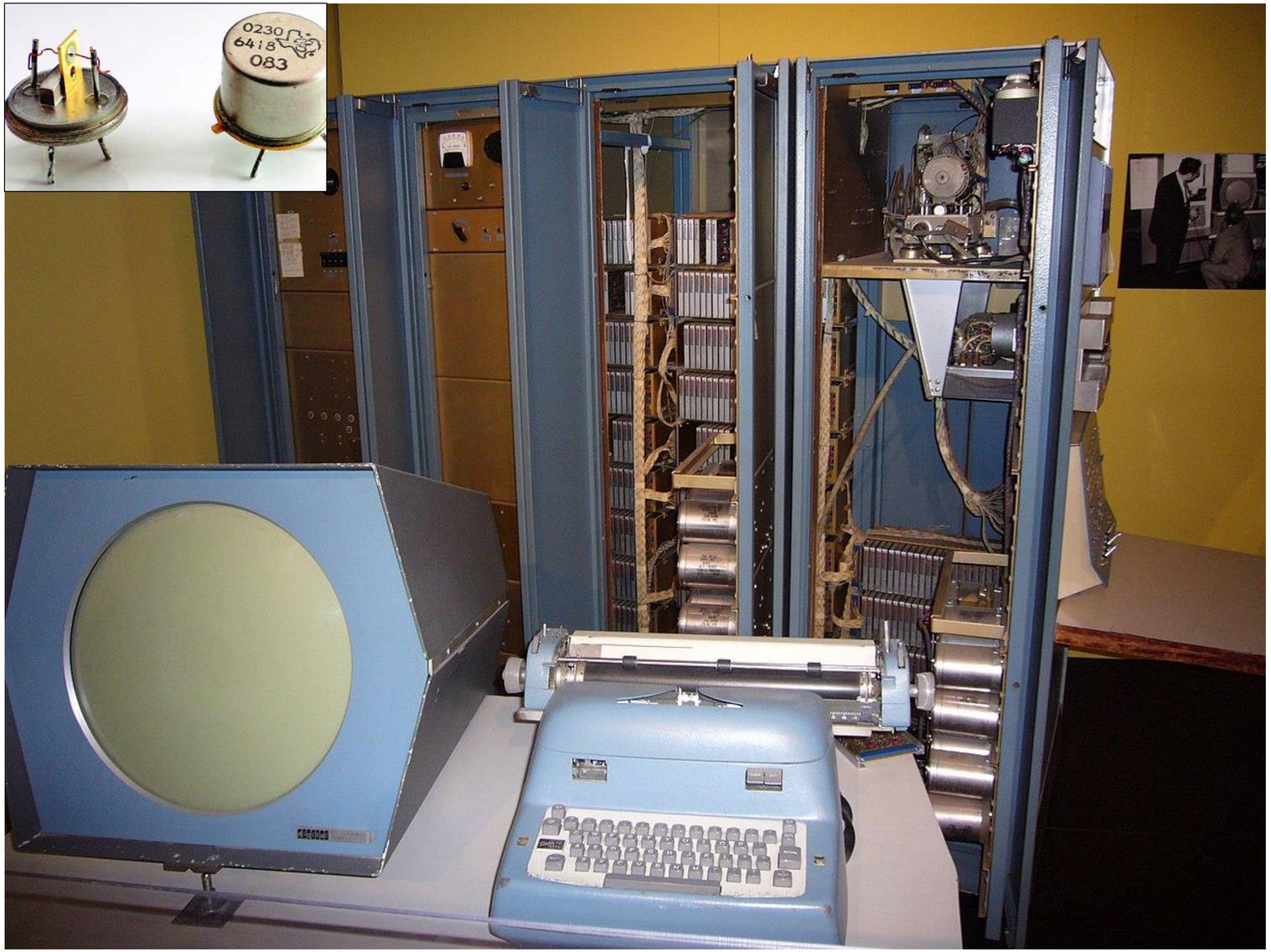


HISTORY OF THE COMPUTER: ELECTRONIC

PDP-1 in 1959

- **Transistors**
- User-friendly





HISTORY OF THE COMPUTER: ELECTRONIC

PDP-1 in 1959

- “Modern” computer design
- Assembly programming
- Electronic typewriter input

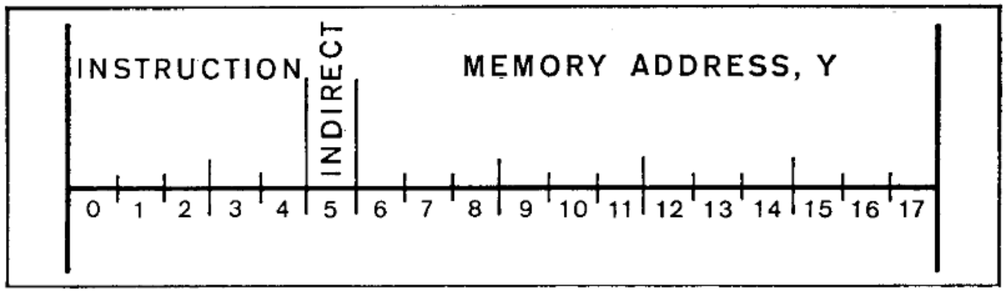
Alphanumeric Typewriter

The typewriter will operate in the input mode or the output mode.

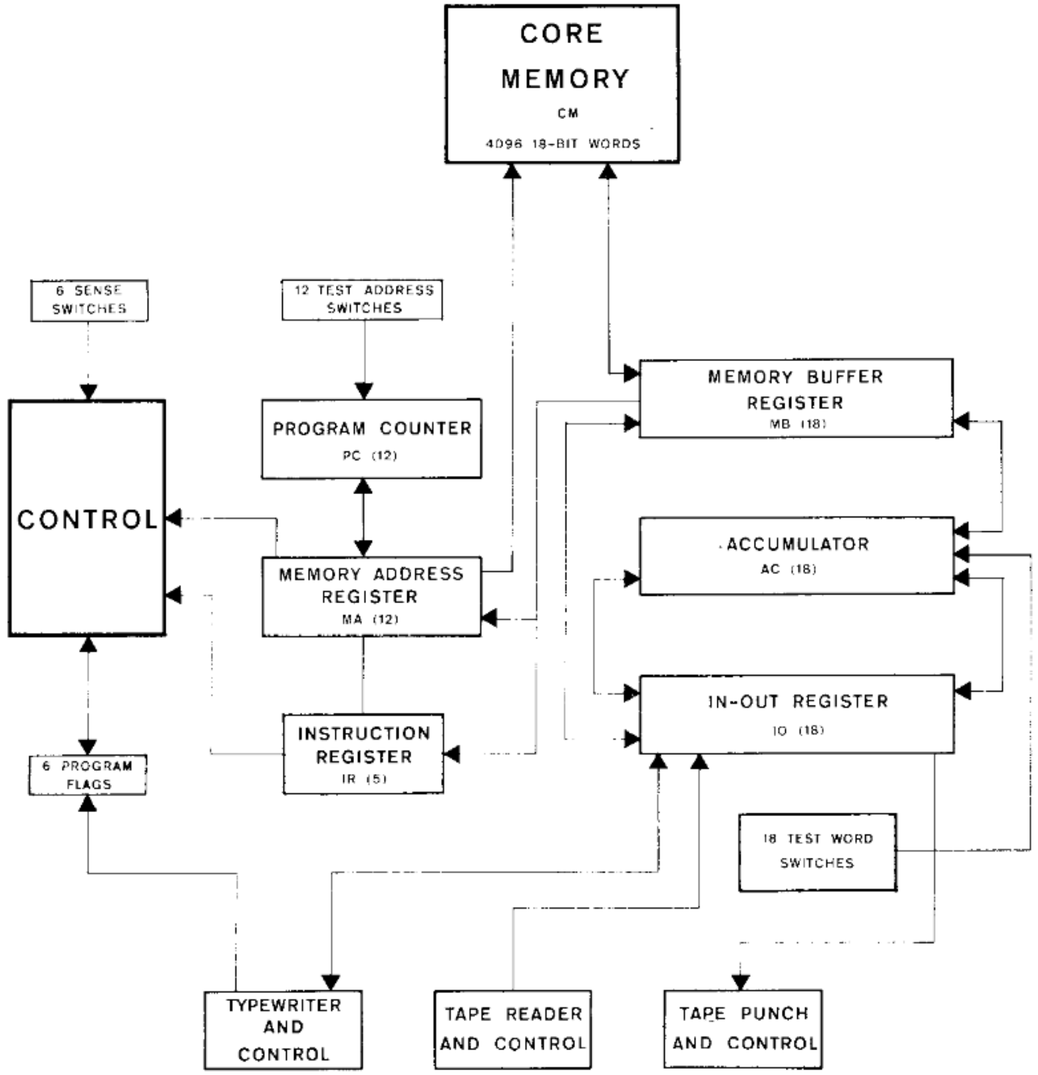
Type Out

tyo Address 0003

For each In-Out Transfer instruction one character is typed. The character is specified by the right six bits of the In-Out Register.



PDP-1 Instruction Format



PDP-1 System Block Diagram

HISTORY OF THE COMPUTER: ELECTRONIC

8800 minicomputer-kit in 1974

- Early personal computer
- No screen/keyboard – LEDs/switches

HISTORY OF THE COMPUTER: ELECTRONIC

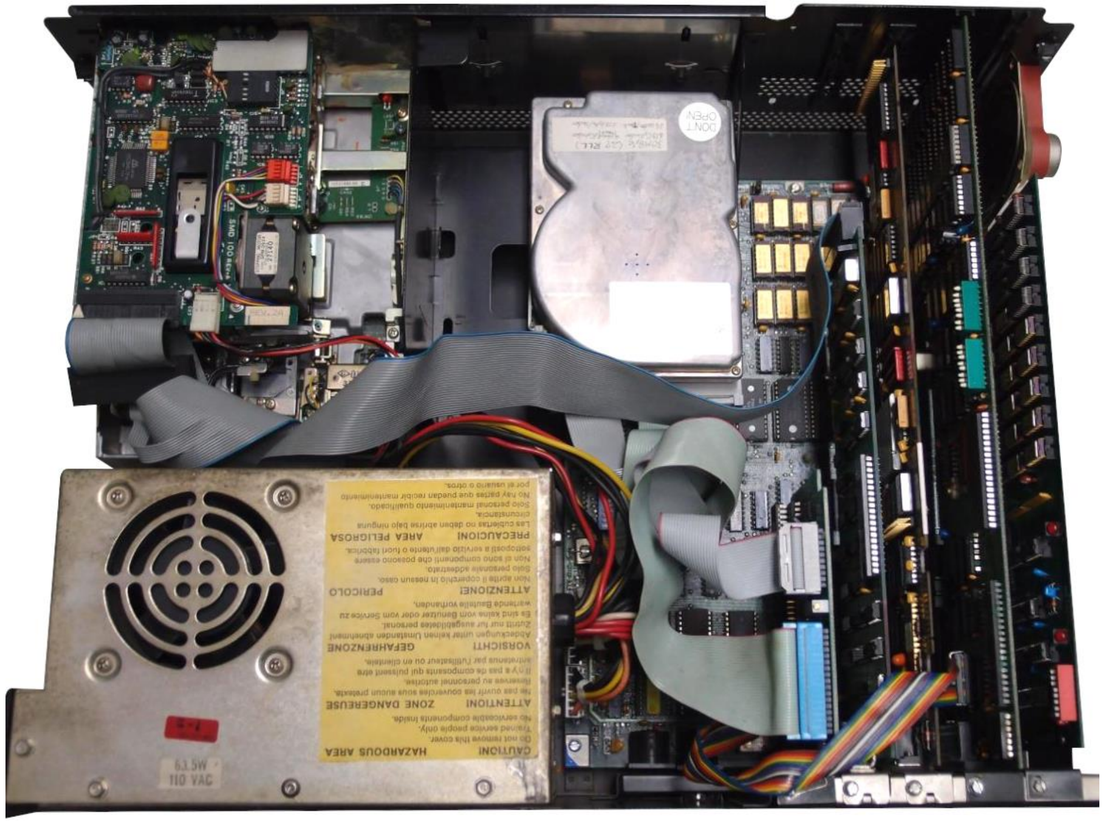
APPLE II in 1977

- Color graphics
- Modern keyboard
- Easy to extend with screen, cassette deck, …
- Programming in Basic

HISTORY OF THE COMPUTER: ELECTRONIC

IBM 5150 PC in 1981

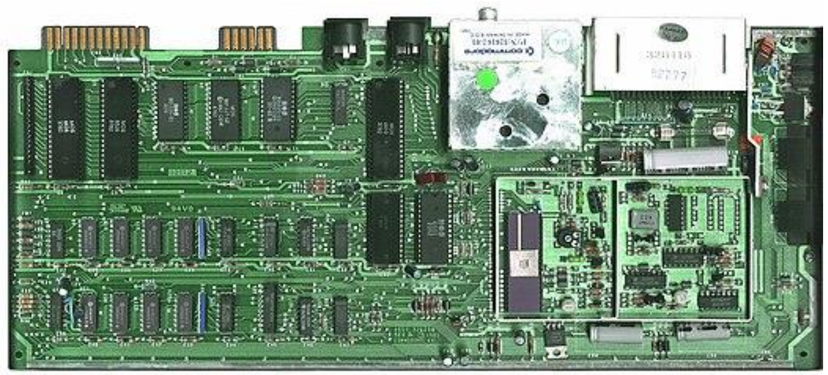
- Focused on business usage



HISTORY OF THE COMPUTER: ELECTRONIC

Commodore 64 in 1982

- Graphics: Color sprites
- Sound: free waveform
- MOS transistors (5 micron)
- 8-bit processor, 64 kB RAM
- Programming in Basic



SUMMARY OF COMPUTER HISTORY

- Calculators vs. computers
  - Sequences of operations (data memory)
  - Programs stored in memory ?
- Basic technique
  - Mechanical
  - Relays, vacuum tubes, or transistors as switch
  - Memory: Magnetic drums
- Manual rewiring, machine code, or assembly available
- Input: switches, typewriters, punched cards/paper
- Output: light indicators, screen, …