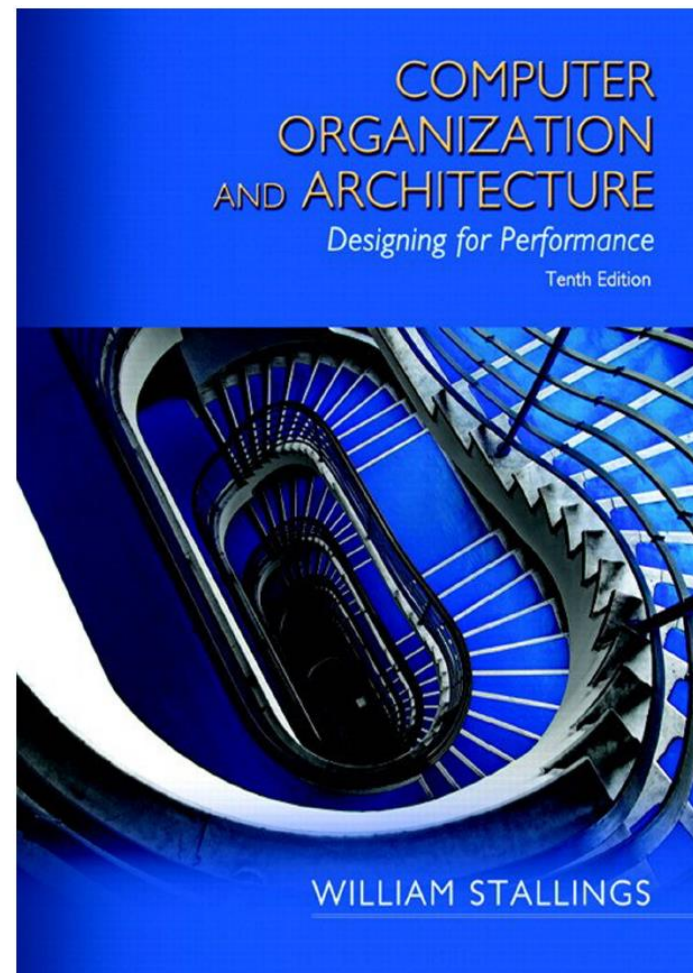
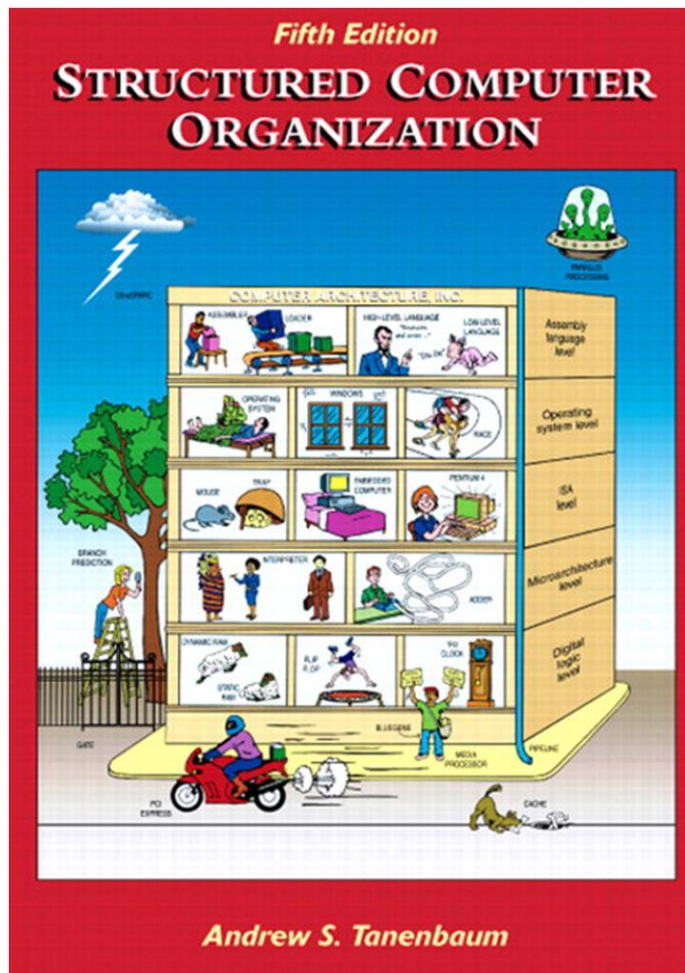


# Architektura Systemów Komputerowych

## dr Marcin Stępnia

Na podstawie wykładów prof. Stanisława Ambroszkiewicza




TEXTS IN COMPUTER SCIENCE

# A Practical Introduction to Computer Architecture



Daniel Page

 Springer

Wiley Series on Parallel and Distributed Computing  
Albert Y. Zomaya, Series Editor

 WILEY



# FUNDAMENTALS OF COMPUTER ORGANIZATION AND ARCHITECTURE

MOSTAFA ABD-EL-BARR

JOHN L. HENNESSY    DAVID A. PATTERSON

# COMPUTER ARCHITECTURE

*A Quantitative Approach*

FIFTH EDITION

MK  
MORGAN KAUFMANN

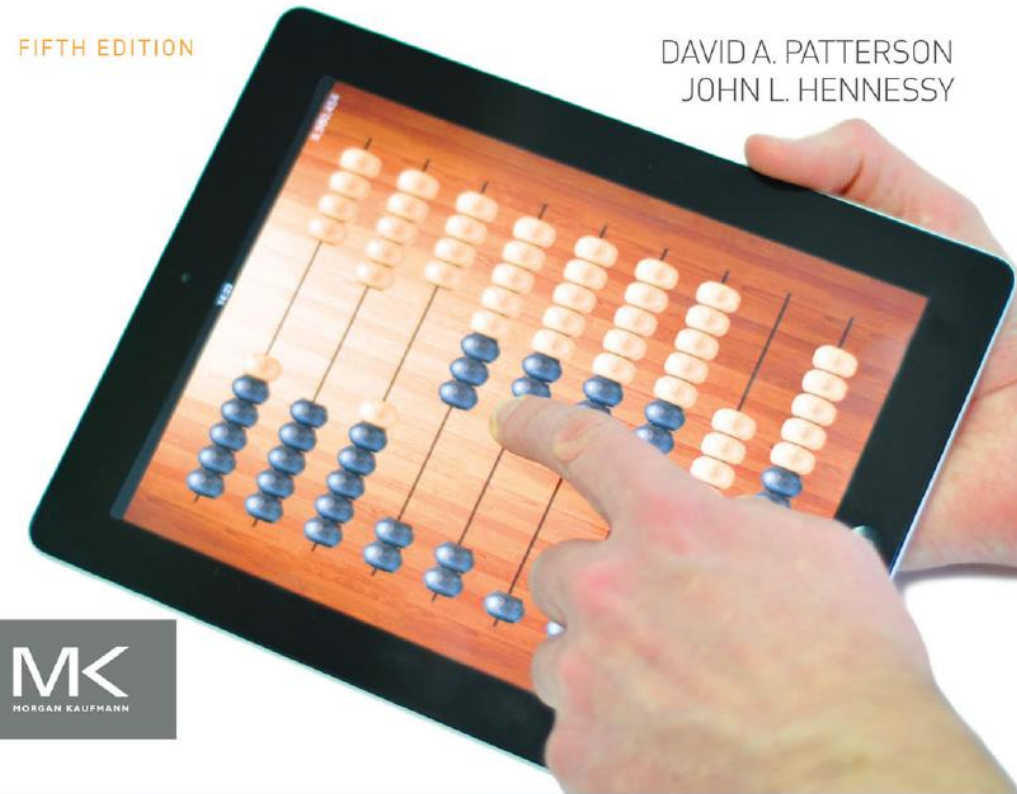


# COMPUTER ORGANIZATION AND DESIGN

THE HARDWARE/SOFTWARE INTERFACE

FIFTH EDITION

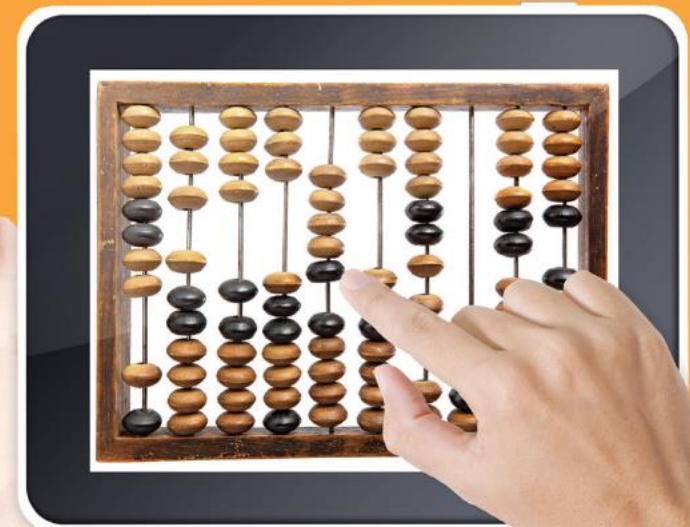
DAVID A. PATTERSON  
JOHN L. HENNESSY



# COMPUTER ORGANIZATION AND DESIGN

THE HARDWARE/SOFTWARE INTERFACE

**ARM® EDITION**



DAVID A. PATTERSON  
JOHN L. HENNESSY

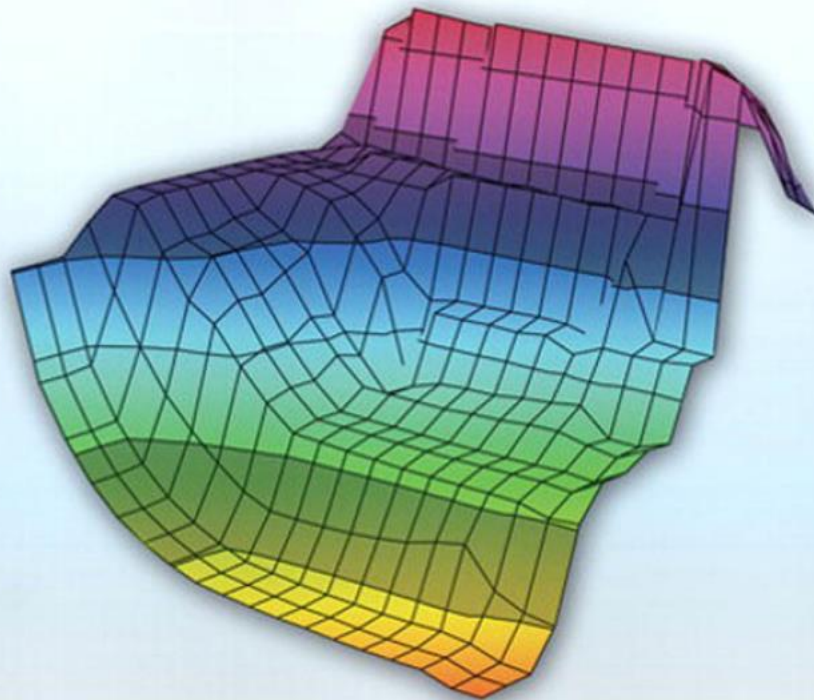
# Architektura Systemów Komputerowych

---

Second Edition

## COMPUTER SYSTEMS

A Programmer's Perspective



Bryant • O'Hallaron

# Pitfalls of Computer Technology Forecasting

---

“DOS addresses only 1 MB of RAM because we cannot imagine any applications needing more.” Microsoft, 1980

“640K ought to be enough for anybody.” Bill Gates, 1981

“Computers in the future may weigh no more than 1.5 tons.” *Popular Mechanics*

“I think there is a world market for maybe five computers.” Thomas Watson, IBM Chairman, 1943

“There is no reason anyone would want a computer in their home.” Ken Olsen, DEC founder, 1977

“The 32-bit machine would be an overkill for a personal computer.” Sol Libes, *ByteLines*

# Architektura Systemów Komputerowych

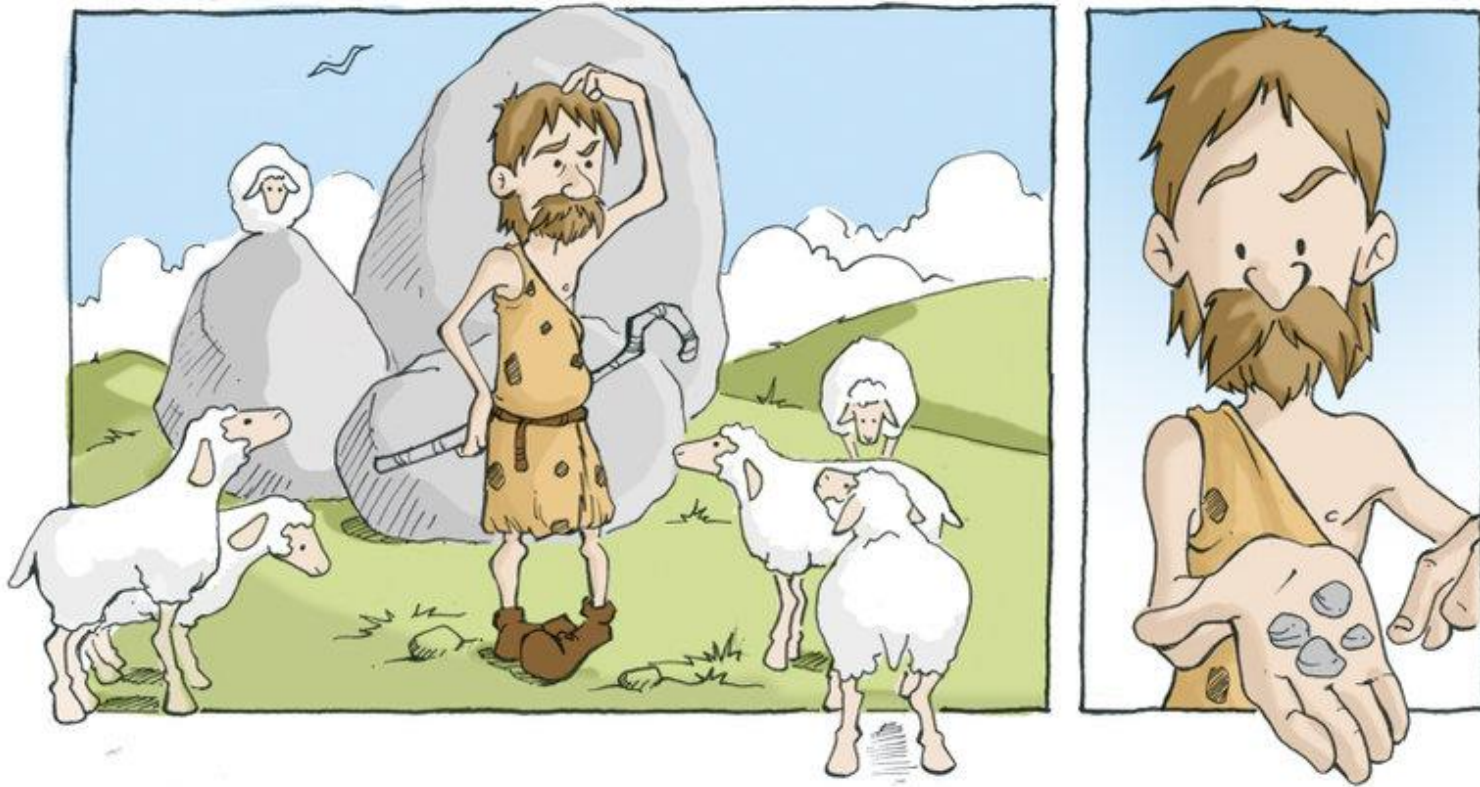
---

- **Komputer** – słowo pochodzące od angielskiego słowa „*computer*”, czyli „*ten który liczy*”
- **Cel:** zrozumieć co to znaczy liczyć i jak można liczyć
- Człowiek potrafi liczyć
- Można te liczenie zautomatyzować, i wówczas maszyna może liczyć
- Komputer to maszyna, która potrafi liczyć. Czy tak jak człowiek?
- Informacja, przetwarzanie informacji
- Algorytmy, programy



# Liczby – jak dawniej liczono (owce, barany i kozły, monety, ...)

---

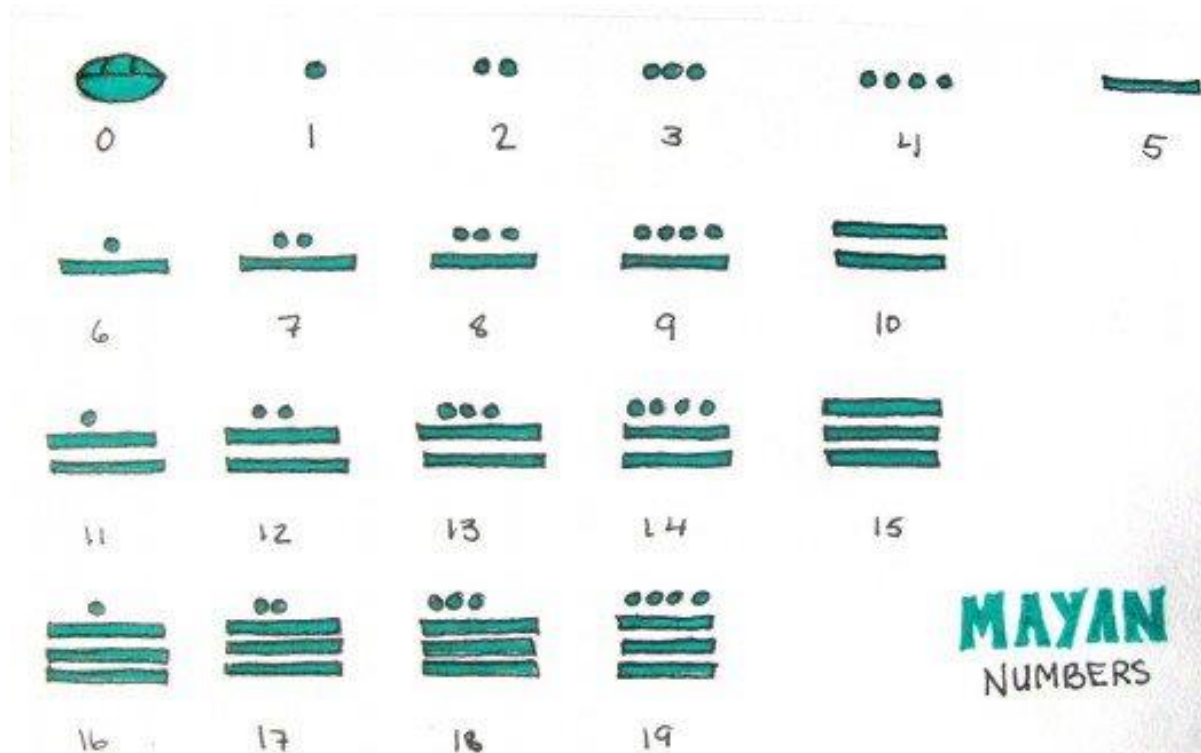




# Liczby i operacje na liczbach

---

- Najprostszy i najbardziej naturalny system pozycyjny
  - | jeden
  - || dwa
  - ||| trzy, itd.
- Dodawanie
- Odejmowanie
- Mnożenie
- Dzielenie



# algorytmy

algorytm (obliczenia, przetwarzanie danych)

Przykłady: pisemne dodawanie, odejmowanie, mnożenie,  
dzielenie (na podstawie dodawania cyfr prostego  
odejmowania) i tabliczki mnożenia  
również przetwarzanie wyrazów, zdań

# DEFINICJA algorytmu

- jednoznaczna metoda obliczenia w skończonym czasie na podstawie danych wejściowych (input) do danych wynikowych (output).
- Te dane to np. liczby naturalne, bajty, (złożone) struktury danych (ciągi bajtów odpowiednio interpretowane)
- Na liczbach naturalnych: 1, 2, 3, 4, ... n, ... to funkcje.
- Rekurencja na podstawie funkcji h definiujemy nową funkcję f:  
 $f(1) := c$ ; stała c  
 $f(n+1) := h(f(n), n+1)$
- Np. suma kolejnych liczb naturalnych do n;  $S(n)$   
 $S(1) = 1$   
 $S(n+1) = S(n) + n+1$   
Gauss jako uczeń podał prostszy wzór  $S(n) := n(n+1)/2$  – jak na to wpadł?



# DEFINICJA algorytmu

- Ciąg Fibonacciego  $\text{Fib}(n)$ :

$$\text{Fib}(0) := 0$$

$$\text{Fib}(1) := 1$$

$$\text{Dla } n > 1, \text{Fib}(n) := \text{Fib}(n - 1) + \text{Fib}(n - 2)$$

$$1, 1, 2, 3, 5, 8, \dots$$

Innym przykładem jest wyliczanie **największego wspólnego dzielnika** za pomocą **algorytmu Euklidesa**:

$$1. \text{gcd}(0, n) = n$$

$$2. \text{gcd}(k, n) = \text{gcd}(n \bmod k, k) \text{ dla } k > 0 \quad (n \bmod k \text{ oznacza tu resztę z dzielenia } n \text{ przez } k).$$

lub inaczej:

$$\text{gcd}(k, n) = \begin{cases} n & \text{dla } k = 0; \\ \text{gcd}(n \bmod k, k) & \text{dla } k > 0. \end{cases}$$

# DEFINICJA algorytmu

- typy danych wejściowych - input:  $T^{\text{in}}$
- typy danych wyjściowych - output:  $T^{\text{out}}$
- funkcje  $f_{ij}: T_i \rightarrow T_j$
- relacje  $R: T_n \rightarrow \text{boolean}$  ( $= \{\text{true}; \text{false}\}$ )
  - np. dla typu liczb naturalnych  $N$  są to relacje  $=$ ,  $>$ , oraz  $<$
- Algorytm to funkcja  $f: T^{\text{in}} \rightarrow T^{\text{out}}$   
skonstruowana z takich funkcji i relacji

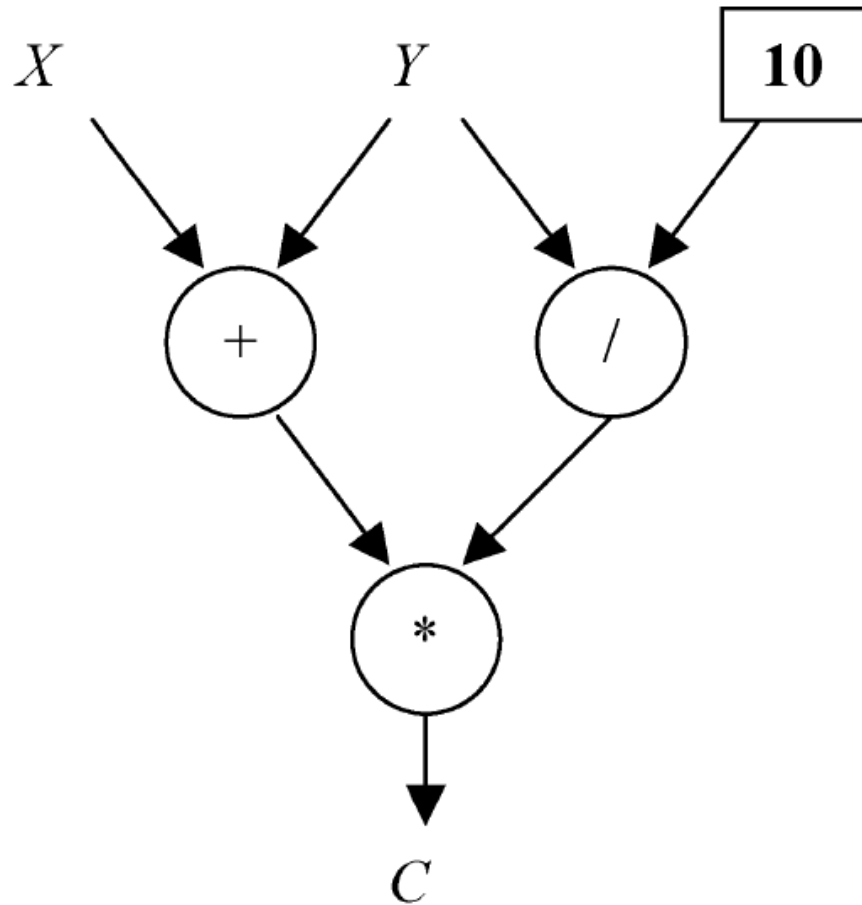
# DEFINICJA algorytmu

- a jak w językach programowania i w komputerze?
- podobnie ale (w architekturze von Neumanna) bardziej skomplikowanie
- wykonanie programu może się zapętlić – konieczna jest zewnętrzna interwencja żeby zakończyć wykonanie



# prosty graf dataflow (graf przepływu danych)

$A := X + Y$   
 $B := Y / 10$   
 $C := A * B$



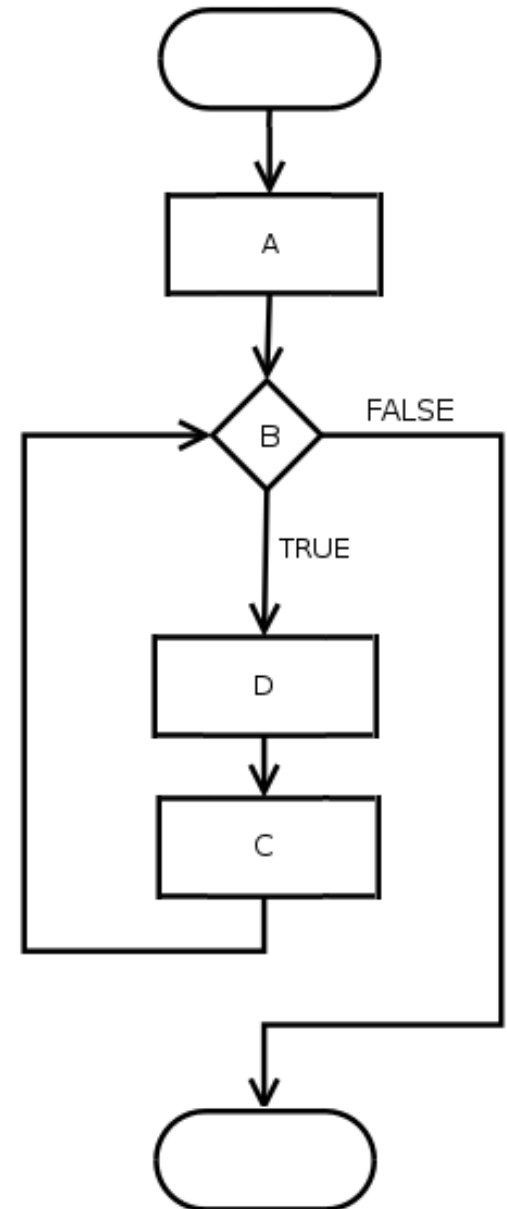
**(a)**

**(b)**

# rozgałęzienia i pętle

- `for(A;B;C)` to formuła
- pętla `while` – wykonuj D dopóki prawdziwa

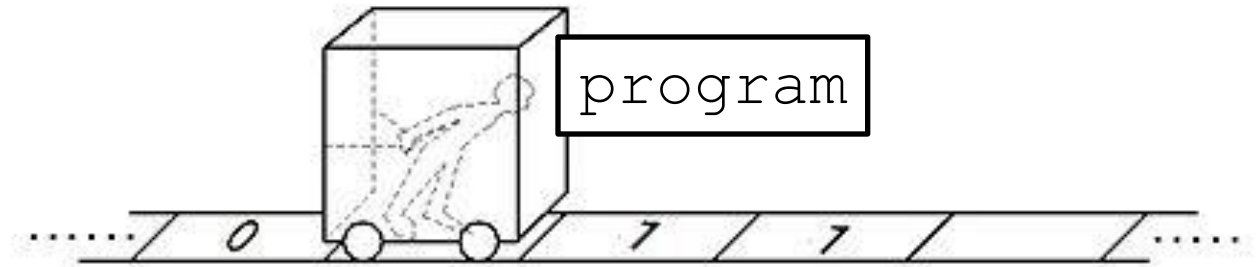
`for(A;B;C)`  
`D;`



# Maszyna Turinga-Posta

## Program

```
1. ins_1  
2. ins_2  
...  
n. ins_n
```



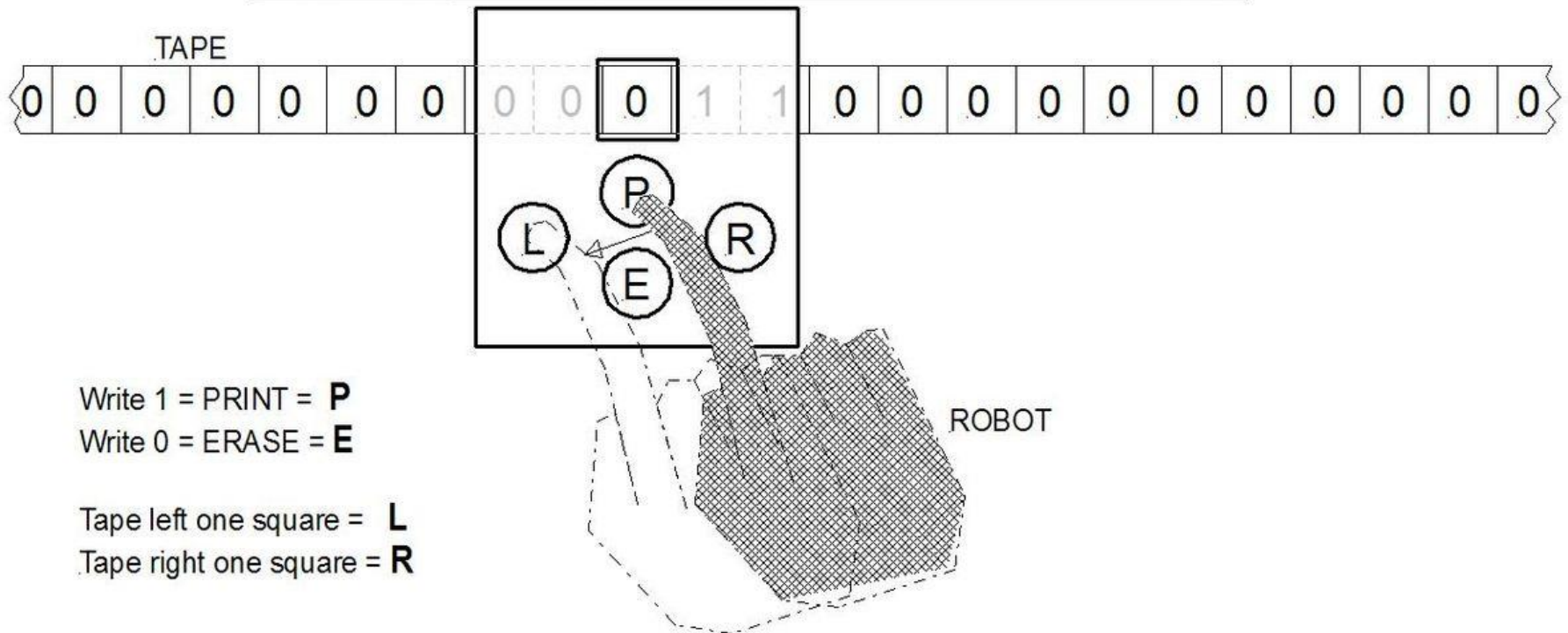
## Instrukcje:

```
1. write 0  
2. write 1  
3. move left  
4. move right  
5. if scanning 0 then goto  
   instruction i  
6. if scanning 1 then goto  
   instruction j
```



# Maszyna Turinga

	Current state A:			Current state B:			Current state C:		
<u>TABLE</u>	Write symbol:	Move tape:	Next state:	Write symbol:	Move tape:	Next state:	Write symbol:	Move tape:	Next state:
tape symbol is 0:	1	R	<b>B</b>	1	L	<b>A</b>	1	L	<b>B</b>
tape symbol is 1:	1	L	<b>C</b>	1	R	<b>B</b>	1	N	HALT



# Maszyna rejestrowa

---

Maszyna rejestrowa (ang. *register machine*) posiada nieograniczoną liczbę rejestrów  $R_n$  (przy czym  $n$  jest liczbą naturalną), z których każdy może zawierać dowolną liczbę naturalną, oznaczona przez  $r_n$ . A więc każdy rejestr jest nieograniczony pamięciowo. Program  $P$  zawiera skończoną liczbę ponumerowanych instrukcji utworzonych na bazie następujących czterech podstawowych instrukcji:

1. zero  $Z(n)$ ; zamień  $r_n$  na 0
2. następnik  $S(n)$ ; zamień  $r_n$  na  $r_n + 1$
3. transfer  $T(m; n)$ ; zamień  $r_n$  na  $r_m$
4. skok  $J(m; n; q)$ ; jeśli  $r_m = r_n$ , to przejdź do instrukcji o numerze  $q$  w  $P$ , w przeciwnym przypadku przejdź do następnej instrukcji w  $P$ .

# Maszyna rejestrowa

---

Maszyna licznikowa – abstrakcyjny model maszyny będący prostą odmianą maszyny rejestrowej.

For a given counter machine model the instruction set is tiny—from just one to six or seven instructions. Most models contain a few arithmetic operations and at least one conditional operation (if *condition* is true, then jump). Three *base models*, each using three instructions, are drawn from the following collection. (The abbreviations are arbitrary.)

- CLR ( $r$ ): CLear register  $r$ . (Set  $r$  to zero.)
- INC ( $r$ ): INCrement the contents of register  $r$ .
- DEC ( $r$ ): DECrement the contents of register  $r$ .
- CPY ( $r_j, r_k$ ): CoPY the contents of register  $r_j$  to register  $r_k$  leaving the contents of  $r_j$  intact.
- JZ ( $r, z$ ): IF register  $r$  contains Zero THEN Jump to instruction  $z$  ELSE continue in sequence.
- JE ( $r_j, r_k, z$ ): IF the contents of register  $r_j$  Equals the contents of register  $r_k$  THEN Jump to instruction  $z$  ELSE continue in sequence.

In addition, a machine usually has a HALT instruction, which stops the machine (normally after the result has been computed).

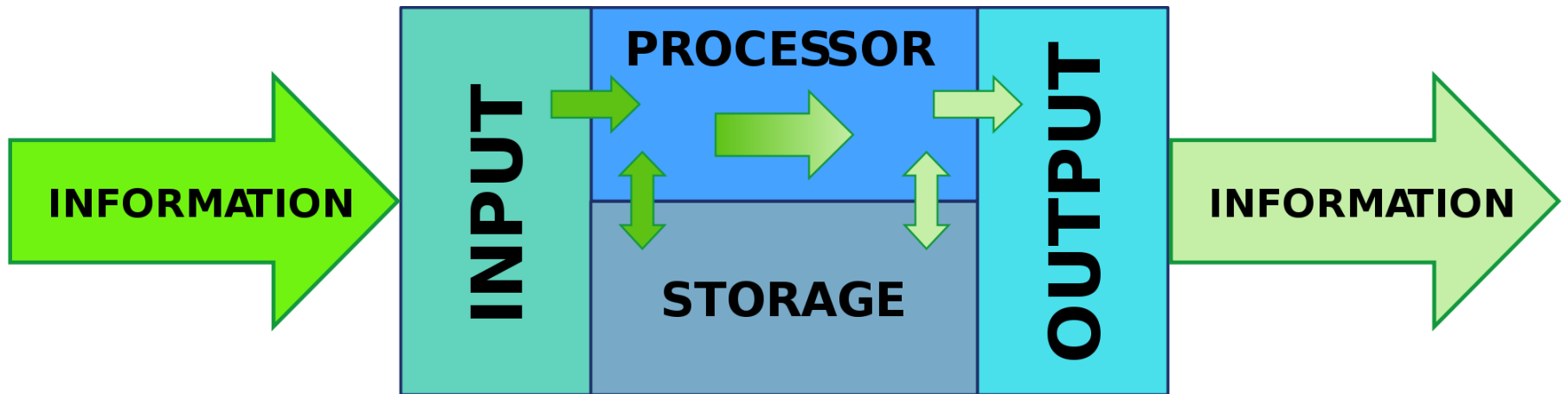
Using the instructions mentioned above, various authors have discussed certain counter machines:

- set 1: { INC ( $r$ ), DEC ( $r$ ), JZ ( $r, z$ ) }, (Minsky (1961, 1967), Lambek (1961))
- set 2: { CLR ( $r$ ), INC ( $r$ ), JE ( $r_j, r_k, z$ ) }, (Ershov (1958), Peter (1958) as interpreted by Shepherdson-Sturgis (1964); Minsky (1967); Schönhage (1980))
- set 3: { INC ( $r$ ), CPY ( $r_j, r_k$ ), JE ( $r_j, r_k, z$ ) }, (Elgot-Robinson (1964), Minsky (1967))

The three counter machine base models have the same computational power since the instructions of one model can be derived from those of another. All are equivalent to the computational power of [Turing machines](#). Due to their unary processing style, counter machines are typically exponentially slower than comparable Turing machines.

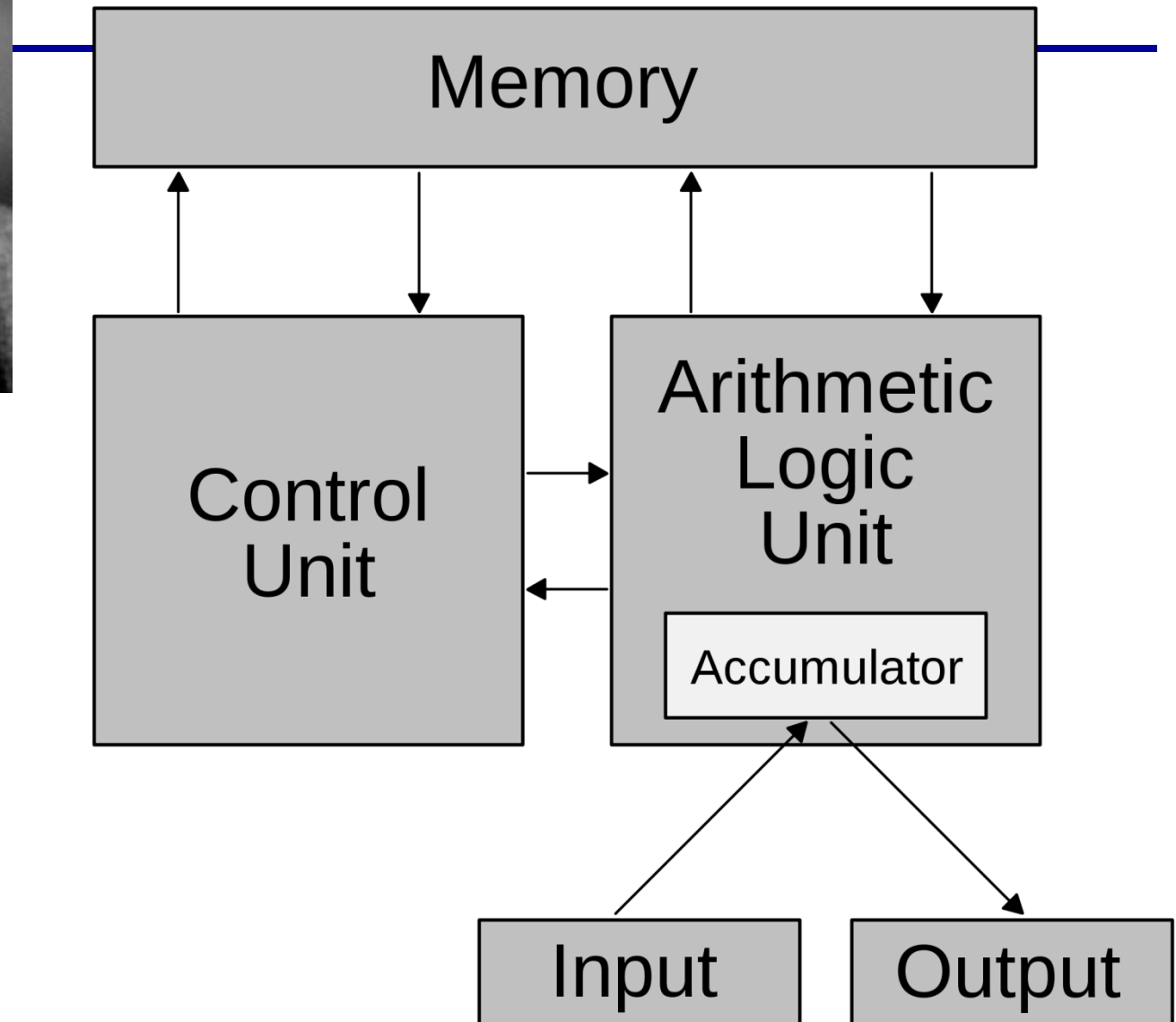
# Co to jest komputer

---



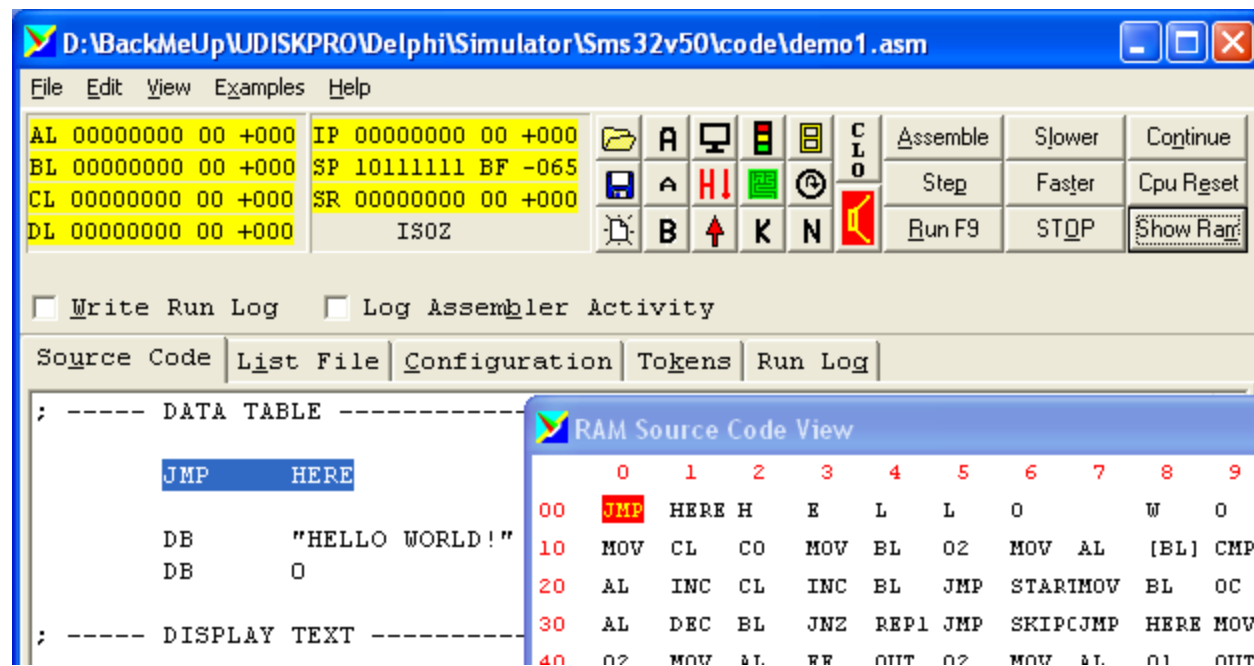


# Komputer von Neumanna



# CEL: poznać architekturę komputera wg. von Neumanna

- Nie tylko poznać ale i programować w prostym asemblerze na prostej maszynie, tj. symulatorze **Microprocessor Simulator V5.0**
- <https://nbest.co.uk/Softwareforeducation/sms32v50/index.php>





File Edit View Examples Help

AL 00010001 11 +017 IP 10001011 8B -117  
 BL 00010100 14 +020 SP 10111111 BF -065  
 CL 11001100 CC -052 SR 00000000 00 +000  
 DL 00000000 00 +000 IS02



Assemble Slower Continue  
 Step Faster Cpu Reset  
 Run F9 STOP Show Ram

☐ Write Run Log ☐ Log Assembler Activity

Source Code List File Configuration Tokens Run Log

```

OUT    02          ; LSB used for multiplexing

MOV     AL,01       ; 0000 0001b
OUT     02          ;
MOV     AL,0        ; 0000 0000b
OUT     02          ;

DEC     BL          ; Count down
JNZ     REP2        ; Jump out of loop on zero

```

```

; ----- HEATER AND THERMOSTAT -----
IN      03          ; Input from thermostat on port 3
CMP     AL,01       ; Is it too warm
JZ      OFF         ; If no then jump to OFF
MOV     AL,80       ; Use MSB to turn heater on.
OUT     03          ; Send 10000000 to port 3
JMP     Skip2       ; Jump past heater-off code

```

```

OFF:
MOV     AL,0        ; Turn o
OUT     03          ; Send o

Skip2:
MOV     BL,20       ; Time D

ON:
DEC     BL          ; BL cou
JNZ     ON          ; Jump o

```

```

; ----- SNAKE IN THE MAZE -----
MOV     AL,FF       ; Maze r
OUT     04          ; Snake
MOV     BL,0A       ; Count
MOV     AL,4F       ; 4 mean

REP5:
OUT     04          ; Send d
DEC     BL          ; Count
JNZ     REP5        ; Jump o

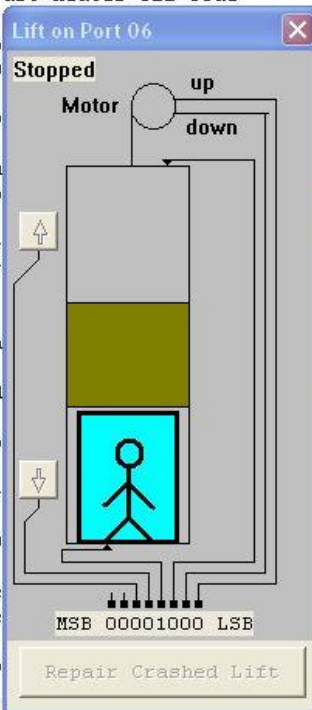
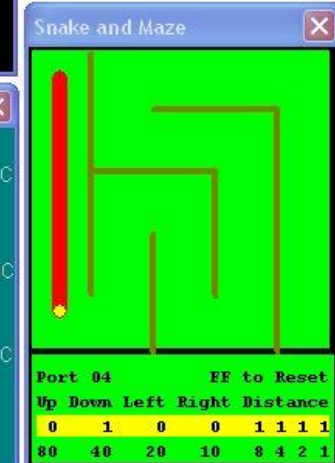
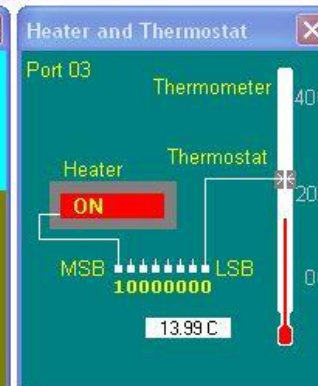
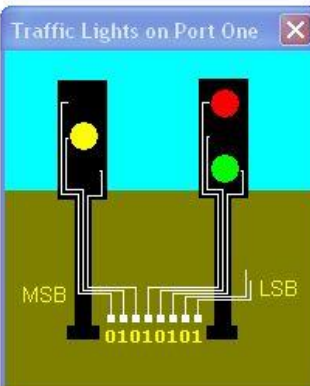
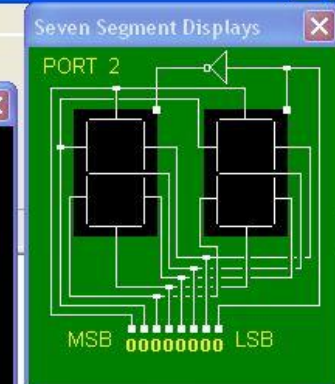
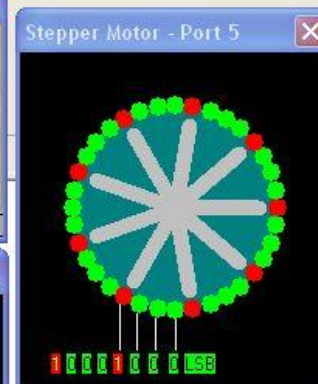
```

```

; ----- SPIN THE STEPPER MOTOR -----
MOV     BL,20       ; Count
MOV     AL,11       ; 0001 0

REP6:
OUT     05          ; Steppe
ROL     AL          ; Rotate
DEC     BL          ; Count
JNZ     REP6        ; Jump o

```



RAM Source Code View

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	JMP	HERE	H	E	L	L	0	W	0	R	L	D	!	0	CLO	
10	MOV	CL	CO	MOV	BL	02	MOV	AL	[BL]	CMP	AL	0	JZ	END1	MOV	[CL]
20	AL	INC	CL	INC	BL	JMP	START	MOV	BL	0C	MOV	AL	AA	OUT	01	NOT
30	AL	DEC	BL	JNZ	REP1	JMP	SKIP0	JMP	HERE	MOV	BL	0C	MOV	AL	FF	OUT
40	02	MOV	AL	FE	OUT											
50	DEC	BL	JNZ	REP2	IN											
60	JMP	SKIP2	MOV	AL	0											
70	FF	OUT	04	MOV	BL											
80	BL	20	MOV	AL	11											
90	MOV	BL	0C	MOV	AL											
A0	MIDDLE	END	END	END	END											
B0	END	END	END	END	END											
C0	48	45	4C	4C	4F											
D0																
E0																
F0																

Key press generates INT 03 - Read key from port 07

INT 03/Port 07

Ctrl Shift Normal

X Hexadecimal Y ASCII Z Source





# Slajdy według **Jaswinder Pal Singh**

---

<http://www.cs.princeton.edu/courses/archive/spr05/cos111/index.php>

# Co robi komputer?

---

Cokolwiek bym mu nie kazał

- Dodaj 2 do 3 i pokaż rezultat
  - Komputer odpowiada: 5
- Pomnóż ten wynik przez 7 i pokaż wynik
  - Komputer odpowiada: 35
- Pokaż mi stronę www II UPH
  - Komputer odpowiada:



# Jak to się dzieje?

---



- Kto to robi?
- W jaki sposób mam mu kazać?

# Dżin w komputerze? Nie

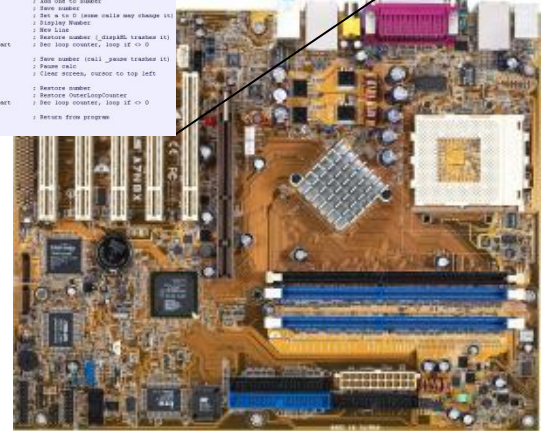


- *wczytaj input i zapamiętaj*
- *wczytaj następny input i zapamiętaj*
- *dodaj i pokaż wynik*
- *wczytaj input i zapamiętaj*
- *pomnóż i pokaż wynik*

```
#include "stdafx.h"
using namespace std;

int main()
{
    int a, b, c;
    cout << "Enter a: ";
    cin >> a;
    cout << "Enter b: ";
    cin >> b;
    c = a + b;
    cout << "Suma: " << c << endl;
    return 0;
}
```

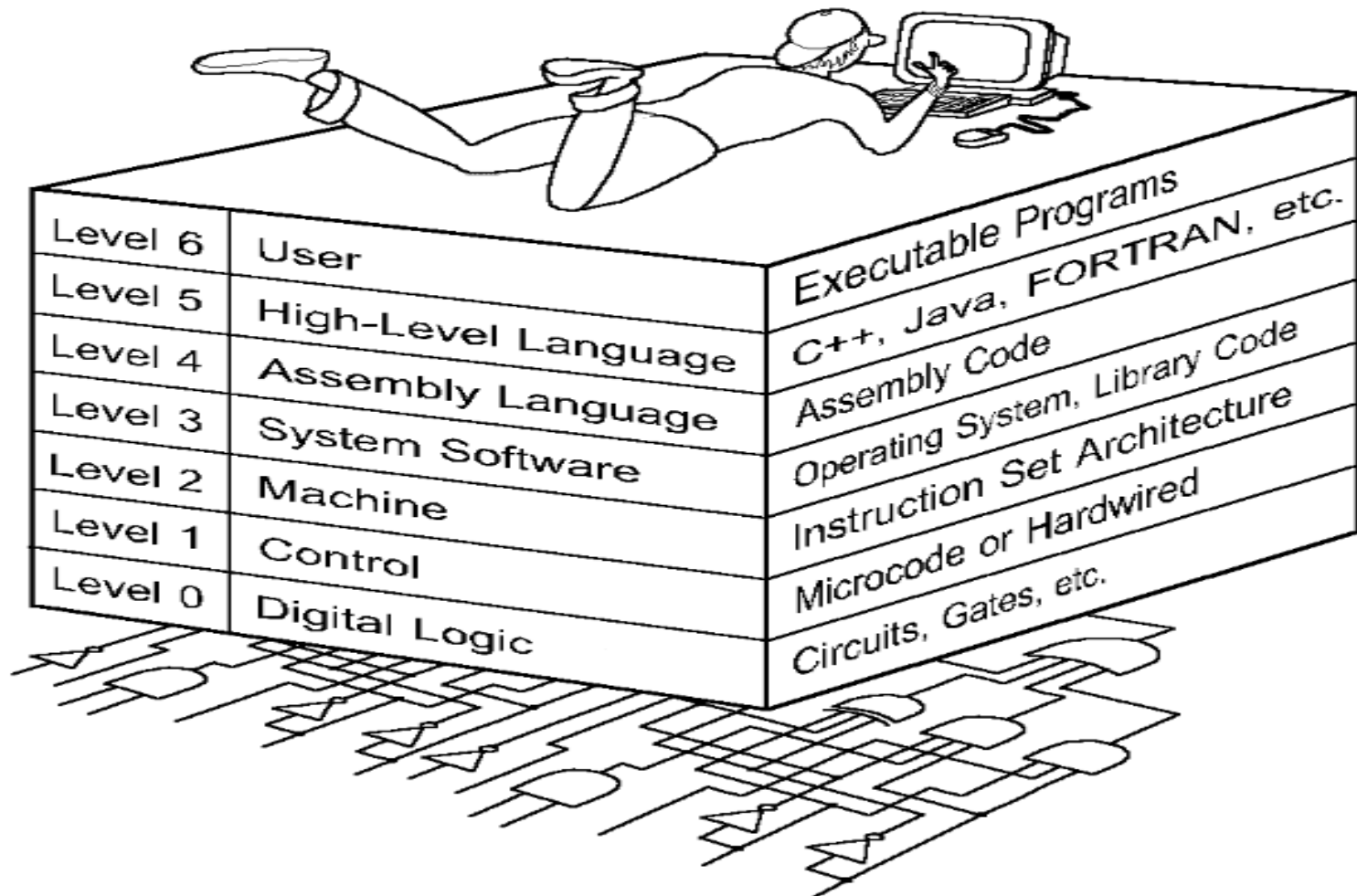
- Hardware (sprzęt) liczy
- Programista pisze kod (program) jak i co ten sprzęt ma liczyć



- Potrzebny jest specjalny język programowania

# Z czego się składa komputer?

## Podójście abstrakcyjne



# Poziomy transformacji

*Problem*

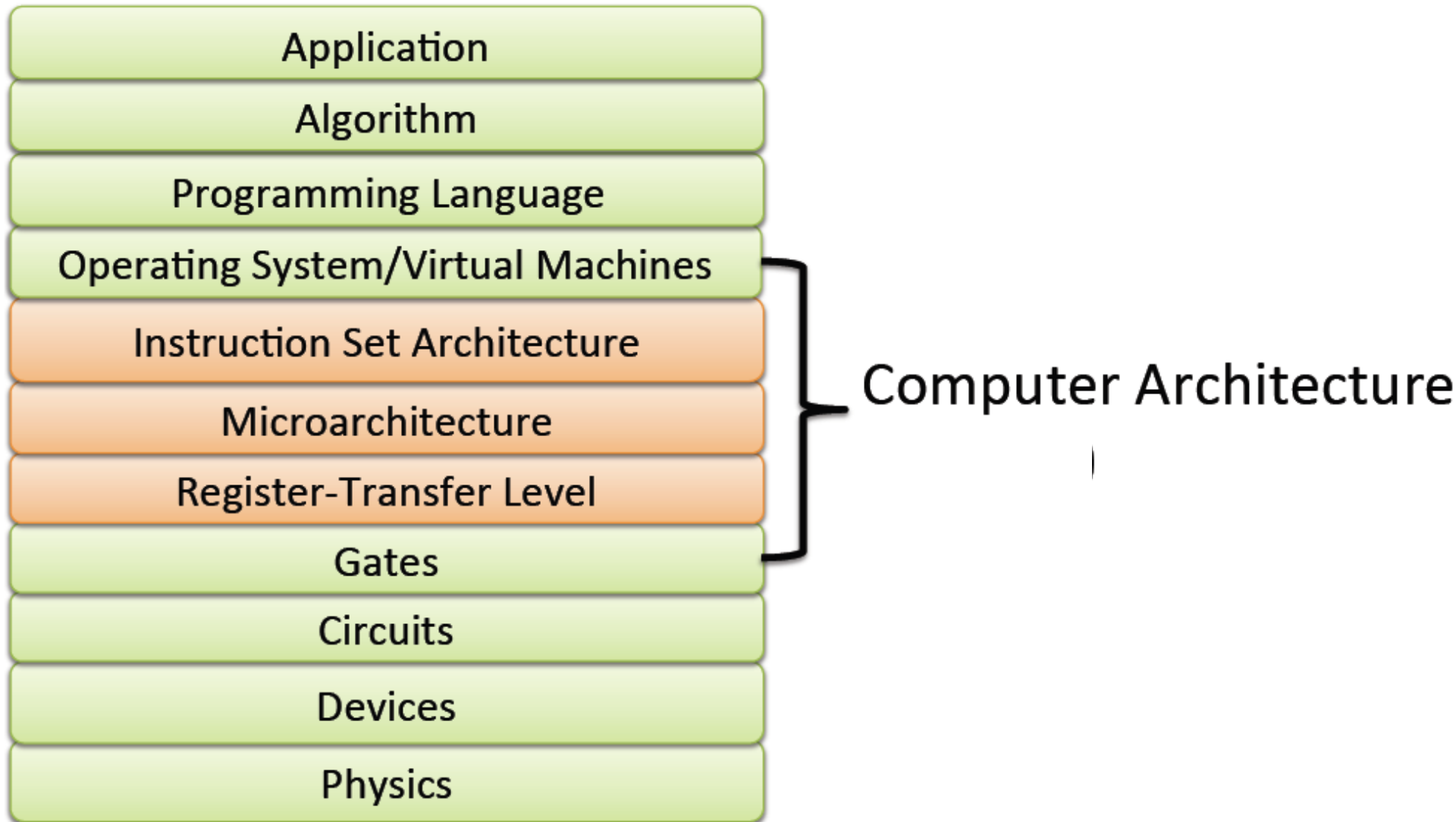


*System operacyjny*

*Interfejs pomiędzy  
Software a Hardware*

*Układy scalone*

# Poziomy transformacji





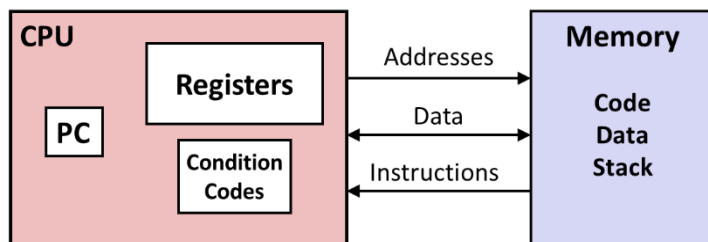
# Poziomy transformacji

C programmer

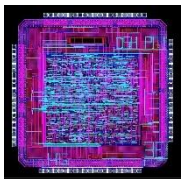
```
#include <stdio.h>
int main() {
    int i, n = 10, t1 = 0, t2 = 1, nxt;
    for (i = 1; i <= n; ++i) {
        printf("%d, ", t1);
        nxt = t1 + t2;
        t1 = t2;
        t2 = nxt; }
    return 0; }
```

**Klarowne poziomy,  
ale uważaj...**

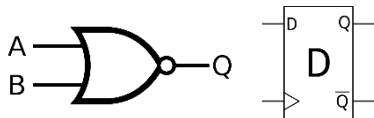
Assembly programmer



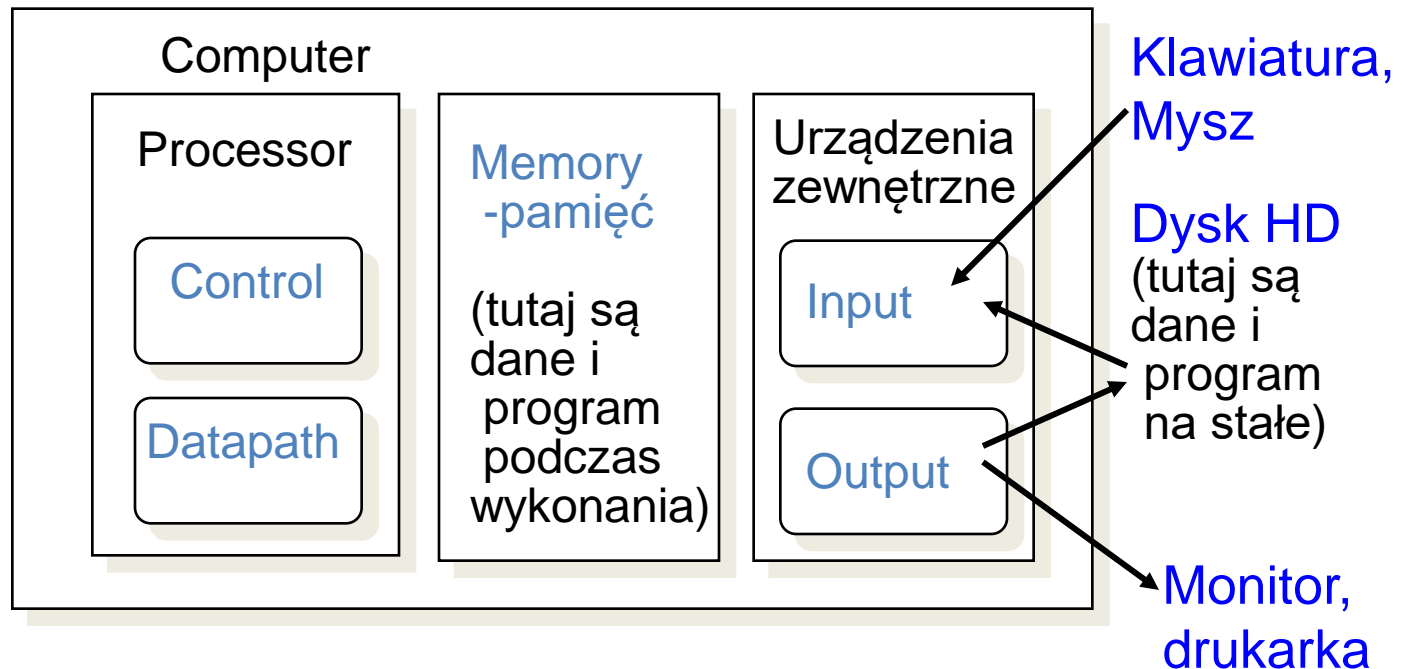
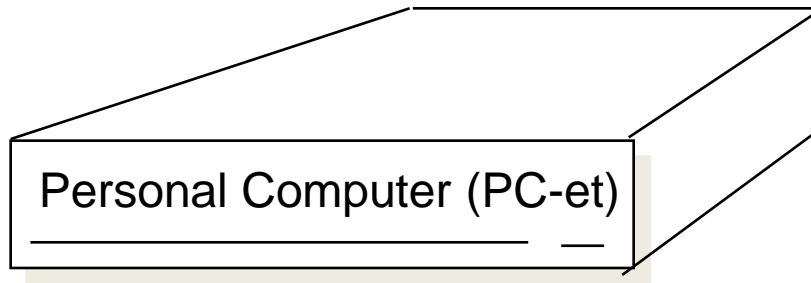
Computer Designer



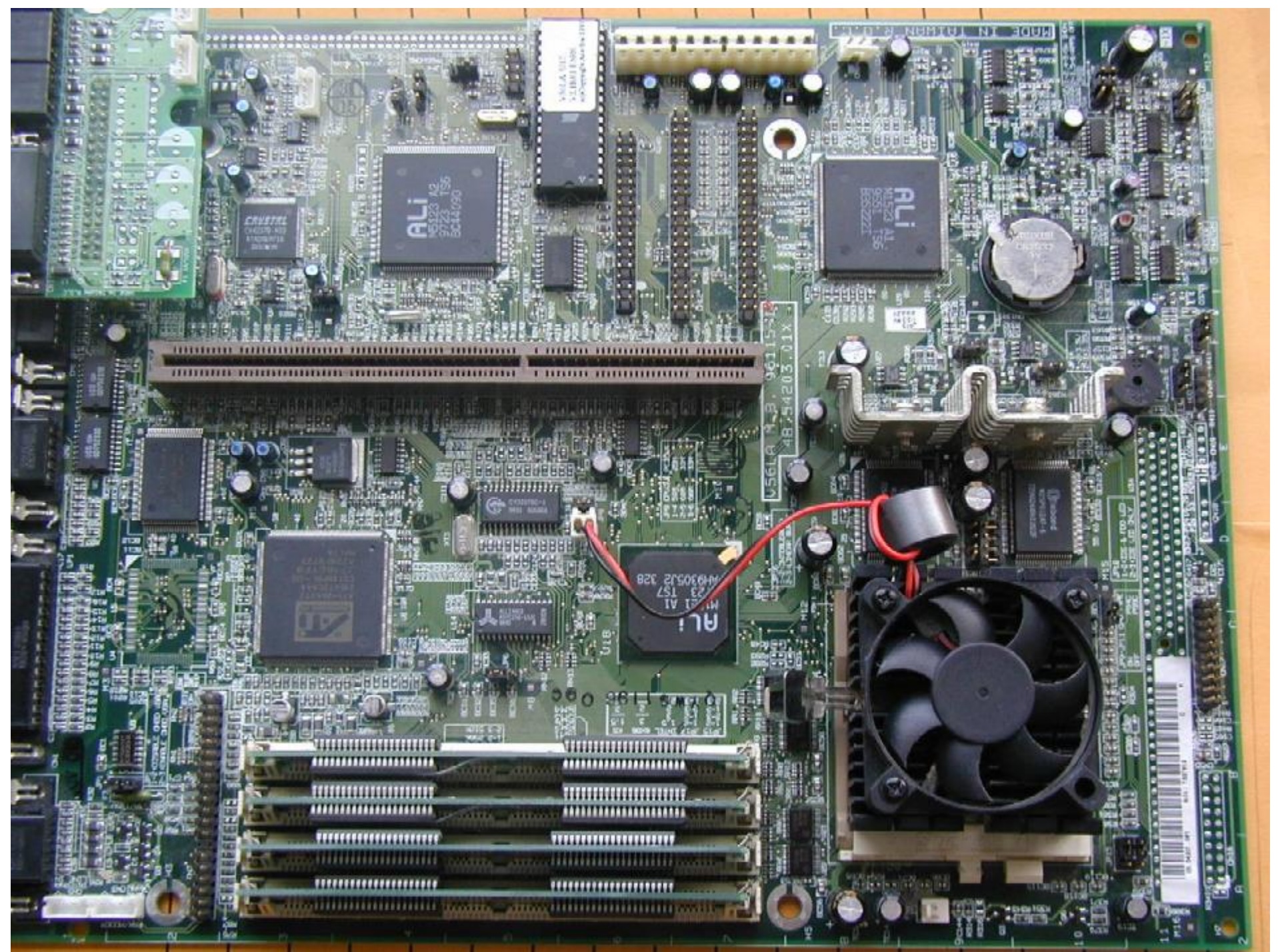
Gates, clocks, circuit layout, ...



# 5 komponentów komputera







# Program

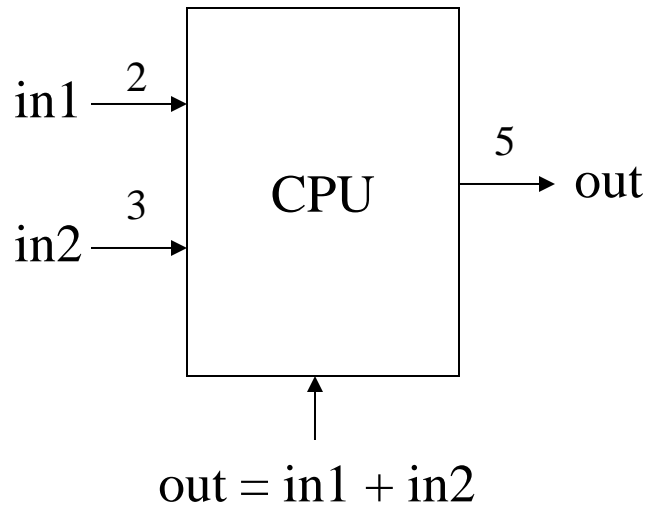
---

```
read (a)      ; wczytaj pierwsza wartość i zapisz na zmienną a
read (b)      ; wczytaj drugą wartość i zapisz na zmienną b
c = a + b     ; dodaj a do b a wynik zapisz na zmiennej c
read (d)      ; wczytaj trzecią wartość i zapisz na zmienną d
e = d * c     ; pomnóż c przez d; rezultat zapisz w zmiennej e
print (e)     ; wyświetl wartość zmiennej e na ekran
```

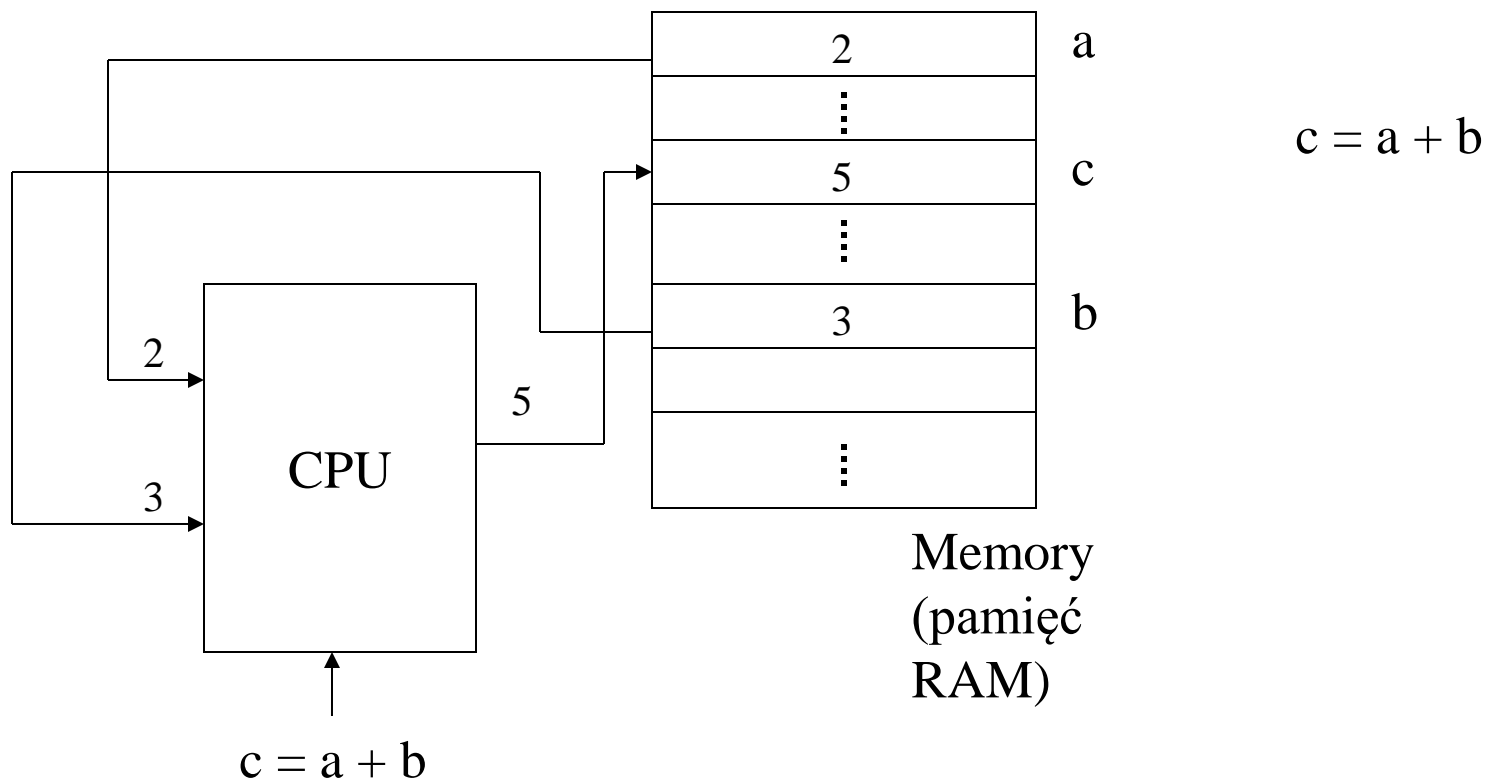
- Zmienna to konkretny obszar w pamięci komputera
- Komputer wykonuje kolejno te instrukcje

# Komputer: liczy output na podstawie input

---



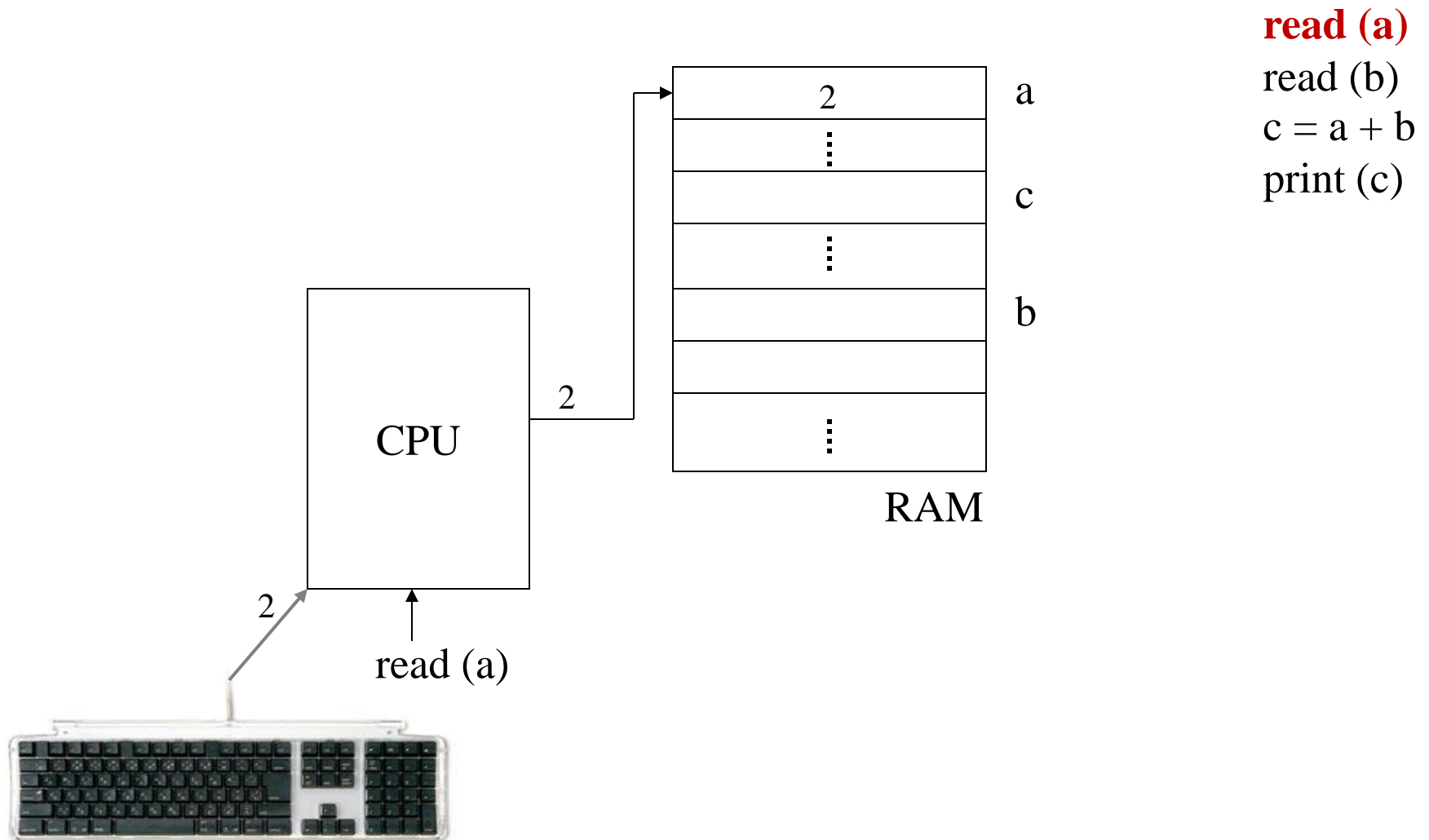
# Komputer: pobiera/zapisuje wartość z/do zmiennych, tj. z/do pamięci





# Komputer: Input (wejście) oraz Output (wyjście)

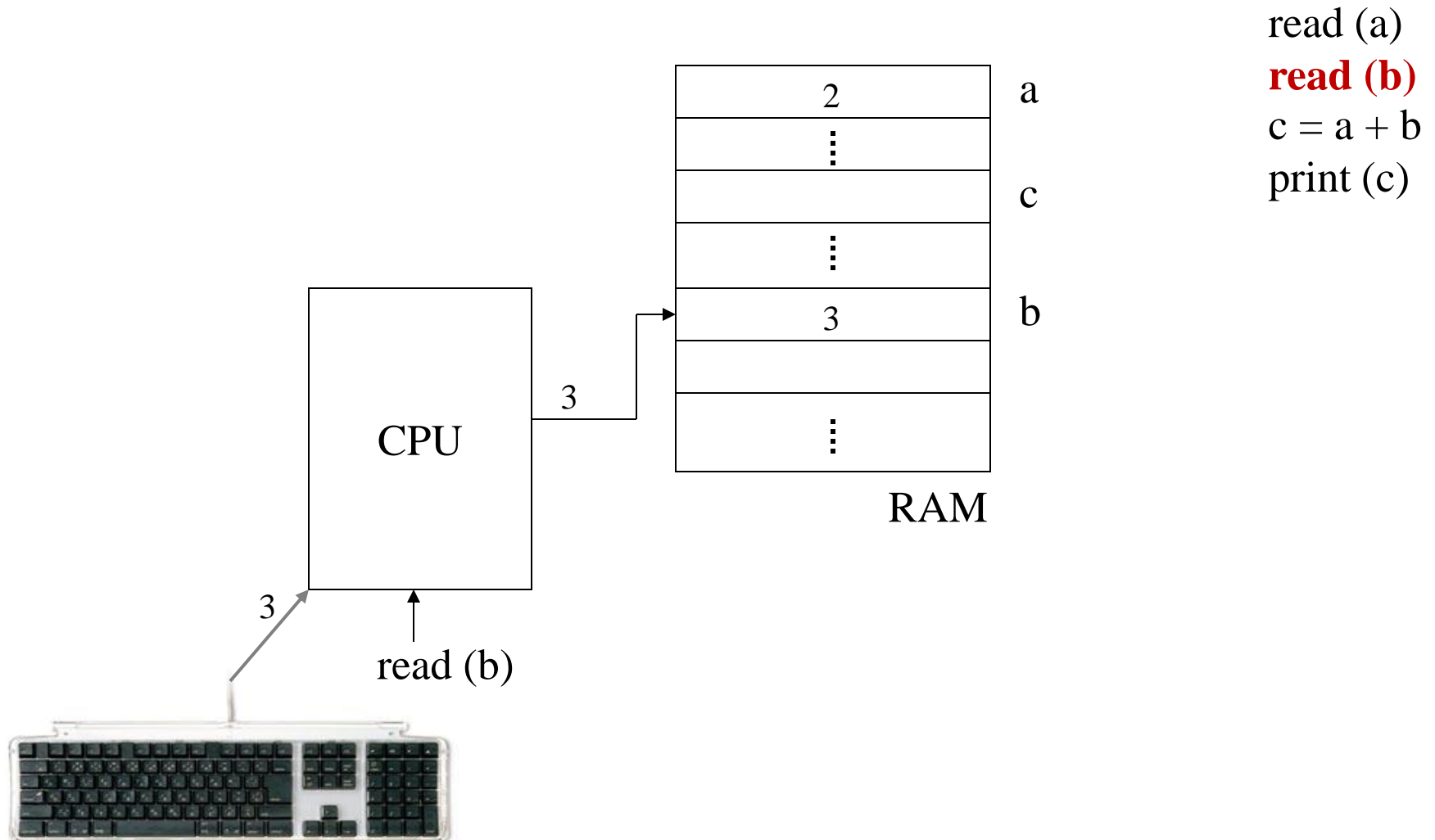
---



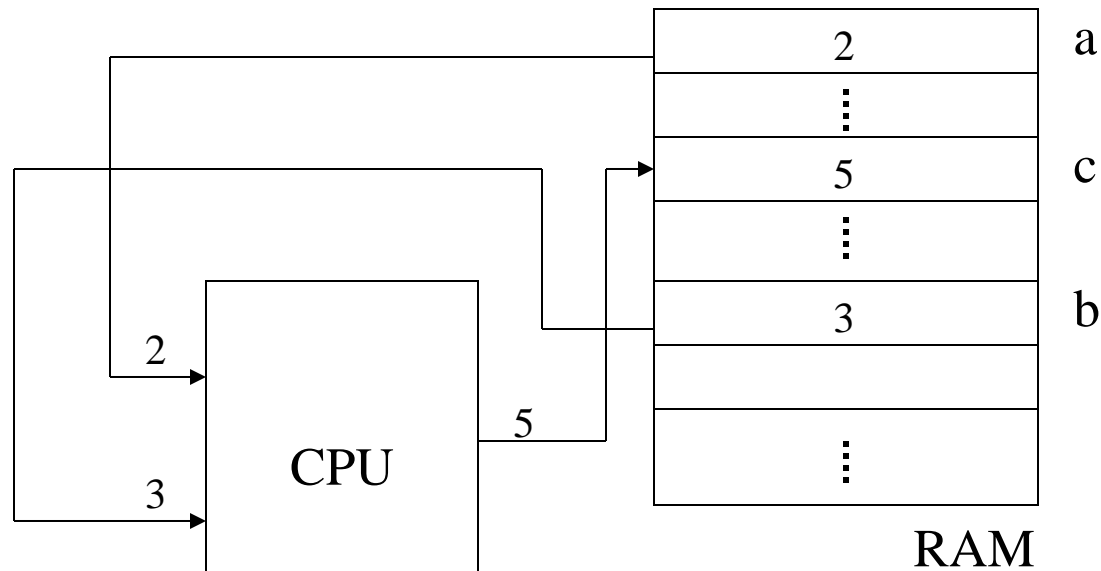


# Komputer: Input oraz Output

---



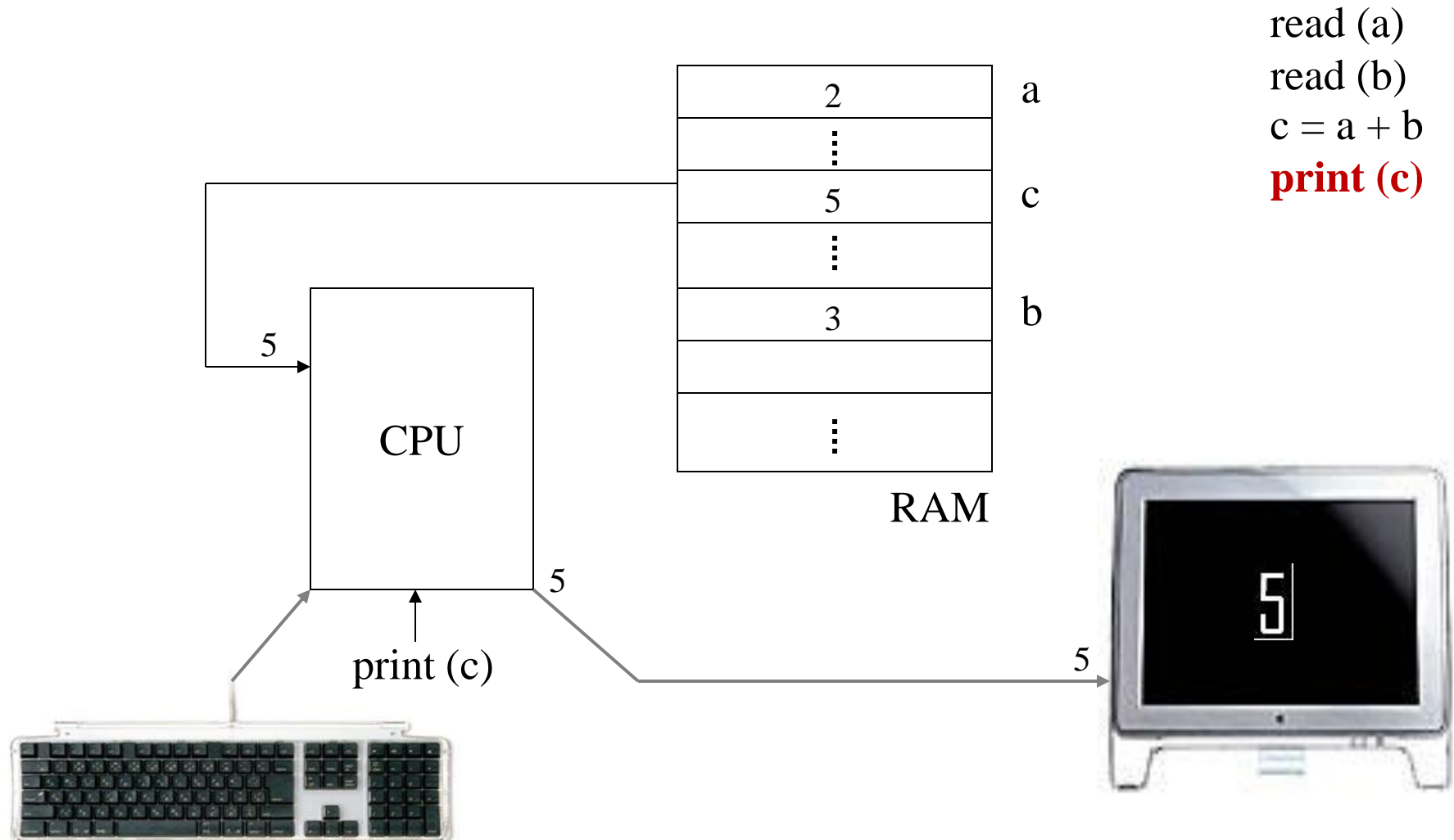
# Komputer: Input oraz Output



read (a)  
read (b)  
 **$c = a + b$**   
print (c)



# Komputer: Input oraz Output

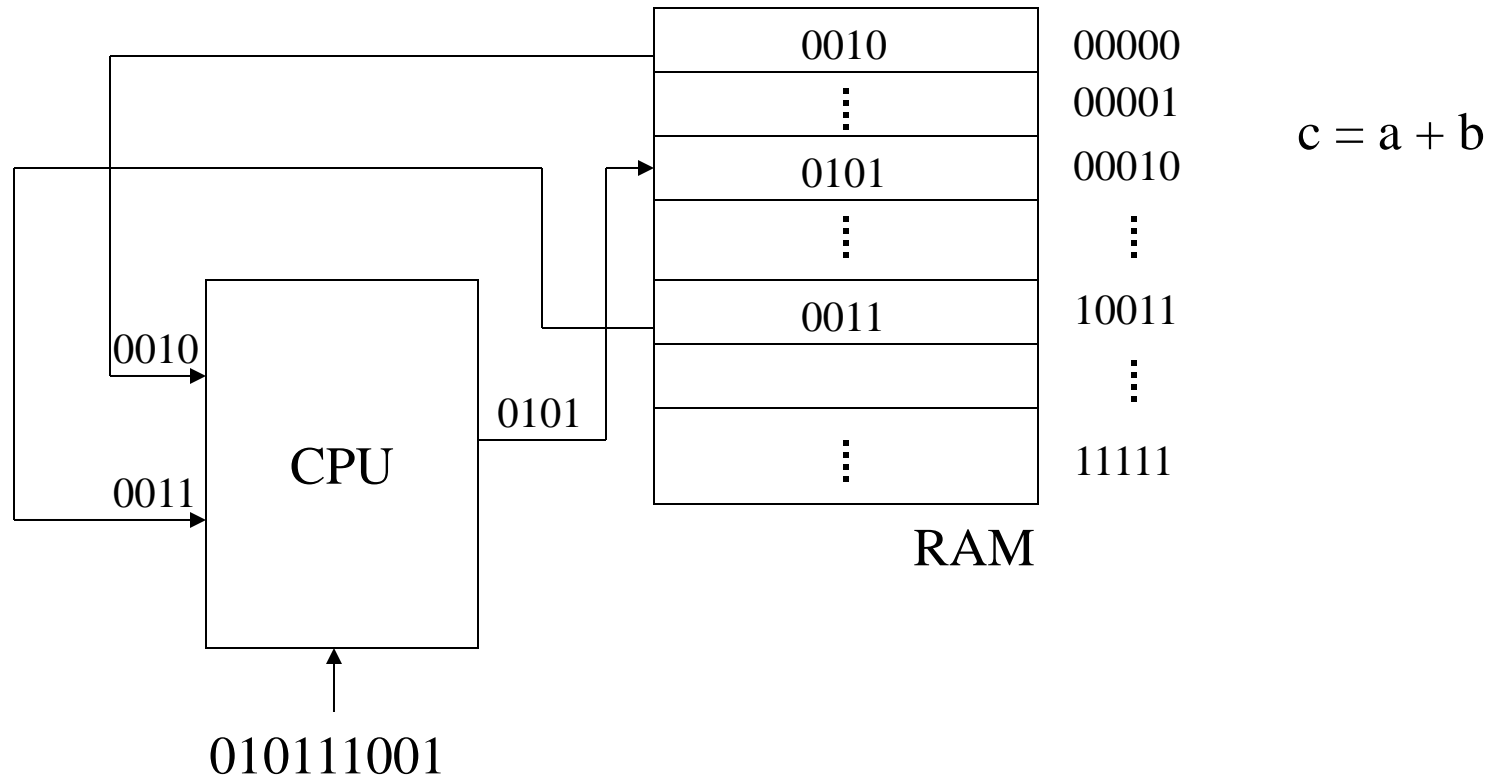


# Komputer: dane i przetwarzanie danych

---

- Co dla komputera znaczą “a”, “b”, “4”, etc.
  - “a” oraz “b” to adresy komórek w pamięci komputera
  - “4” to liczba cztery
- Jak komputer rozumie instrukcje (rozказы), np. “read (a)”
- Jak i gdzie jest zapisany program, tj. sekwencja rozkazów
- Jak komputer pobiera i wykonuje kolejne instrukcje z programu
- **ODPOWIEDŹ:** wszystko to jest zapisane jako bity: zero **0** oraz jeden **1** . Dokładniej, jako ciągi bitów.

# Komputer żyje w świecie bitów



# Programowanie na komputerze

High-level  
language  
program  
(in C)

```
swap(int v[], int k)
{
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

C compiler

Assembly  
language  
program

```
swap:
    muli $2, $5, 4
    add $2, $4, $2
    lw $15, 0($2)
    lw $16, 4($2)
    sw $16, 0($2)
    sw $15, 4($2)
    jr $31
```

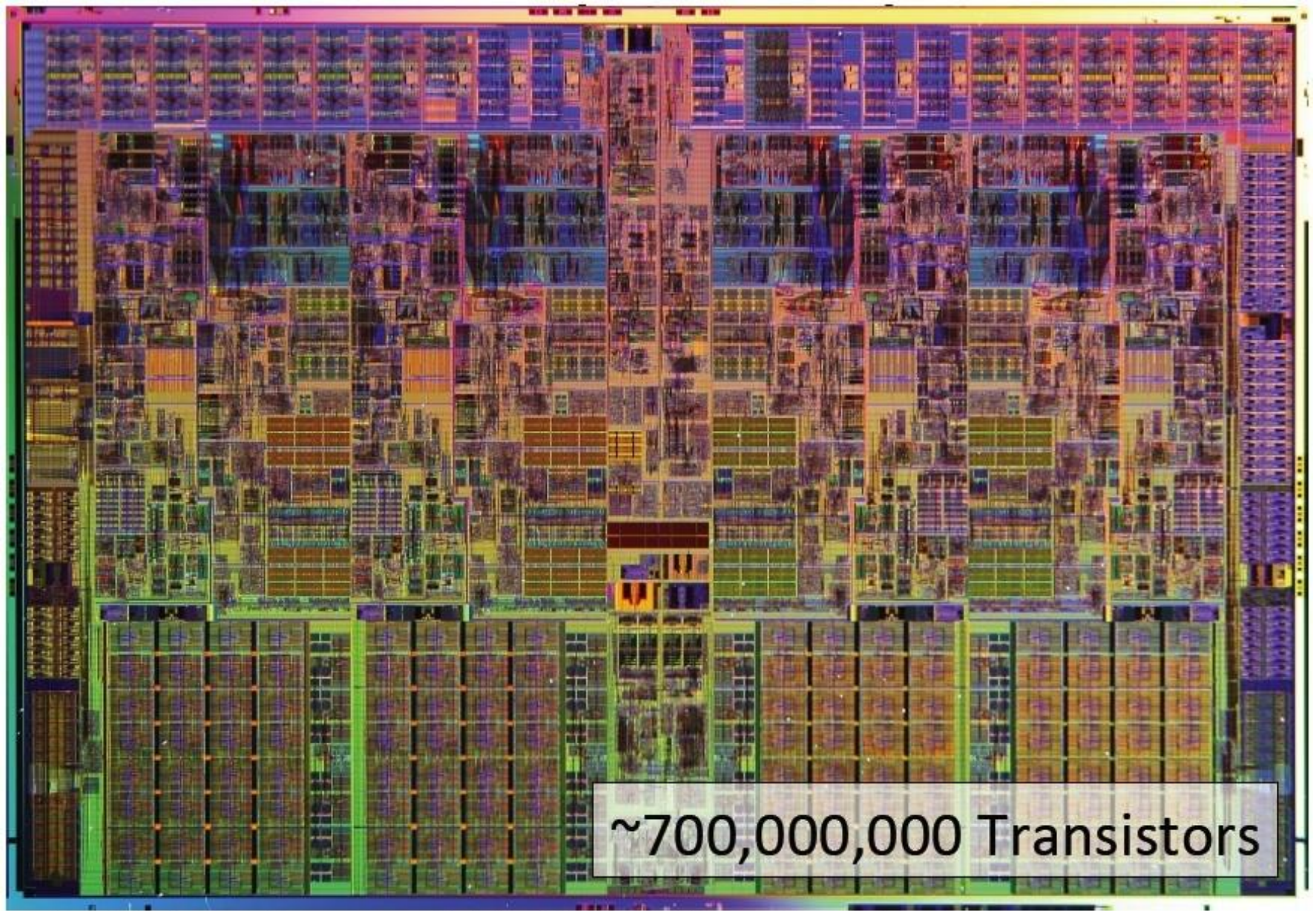
Assembler

```
000000001010000100000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
100011001111001000000000000000100
10101100111100100000000000000000
101011000110001000000000000000100
00000011111000000000000000001000
```

Binary machine  
language  
program





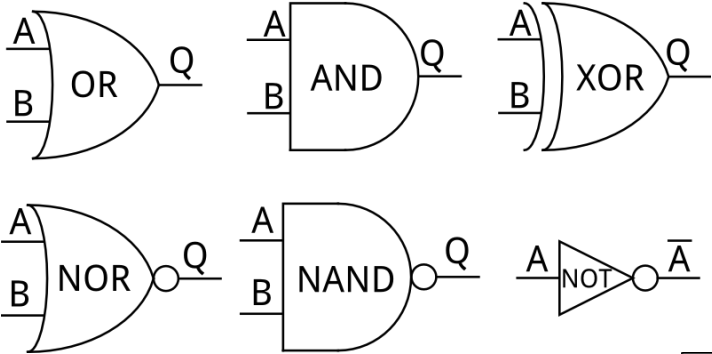


~700,000,000 Transistors

Intel Nehalem Processor, Original Core i7, Image Credit Intel:

[http://download.intel.com/pressroom/kits/corei7/images/Nehalem\\_Die\\_Shot\\_3.jpg](http://download.intel.com/pressroom/kits/corei7/images/Nehalem_Die_Shot_3.jpg)

# Bramki logiczne



- Każda bramka to kilka tranzystorów

		A	
		0	1
B	0	0	1
	1	1	1

		A	
		0	1
B	0	0	0
	1	0	1

		A	
		0	1
B	0	0	1
	1	1	0

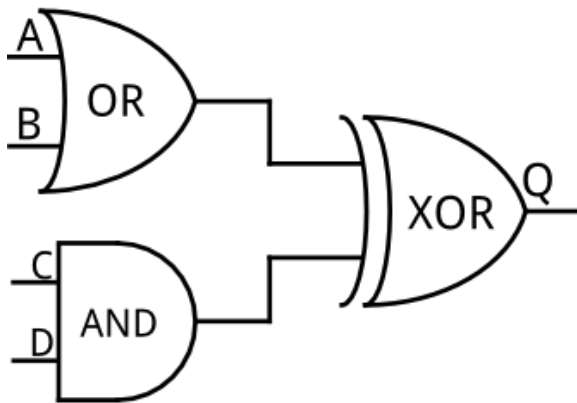
		A	
		0	1
B	0	1	0
	1	0	0

		A	
		0	1
B	0	1	1
	1	1	0

A	
0	1
1	0

# Bramki logiczne

- Współczesny procesor zawiera > 100 mln bramek



		AB			
		00	01	10	11
CD	00	0	1	1	1
	01	0	1	1	1
	10	0	1	1	1
	11	1	0	0	0

# Praca komputera: taktowanie

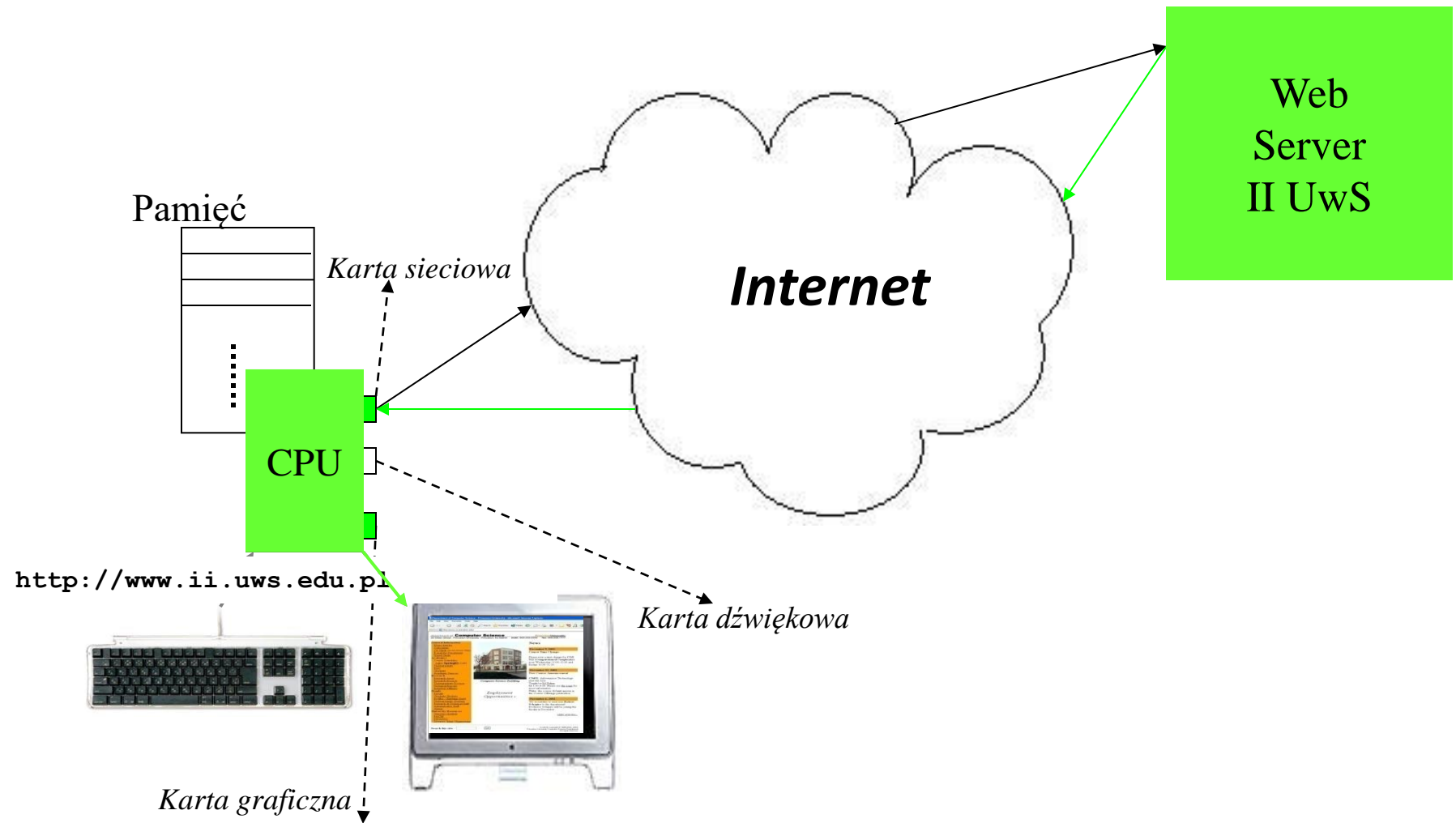
---

- Jeden **takt**, to jeden wykonany rozkaz przez CPU
- **Takty** są wyznaczone przez **zegar** (clock)
- Stan CPU jest zapisywany w specjalnych rejestrach
- Procesor (tj. CPU) ma więc wiele wejść różnych typów:
  - do pobierania / zapisywania danych (np. 2, 3, etc)
  - pobierania kolejnych instrukcji (np. `read a; c=a+b;` etc)
  - dla zegara do wyznaczania kolejnych taktów



# Komputer w sieci (Interneecie)

---



# Komputer: kompilacja i wykonywanie programów

---

- Programy są pisane przez (np. studentów) w języku (np. prosty assembler) dla nich zrozumiałym.
- Muszą być przetłumaczone (skompilowane) do języka maszynowego do rozkazów w postaci bajtów, tzw. *binary code*
- System operacyjny wykonuje *binarny kod* zarządzając hardwarem. W tym samym czasie może być wykonywane wiele programów (aplikacji, np. przeglądarka www, email, MS Word, MP3 Player) chociaż CPU (procesor) wykonuje rozkazy sekwencyjnie, tj. jeden rozkaz w jednym takcie.

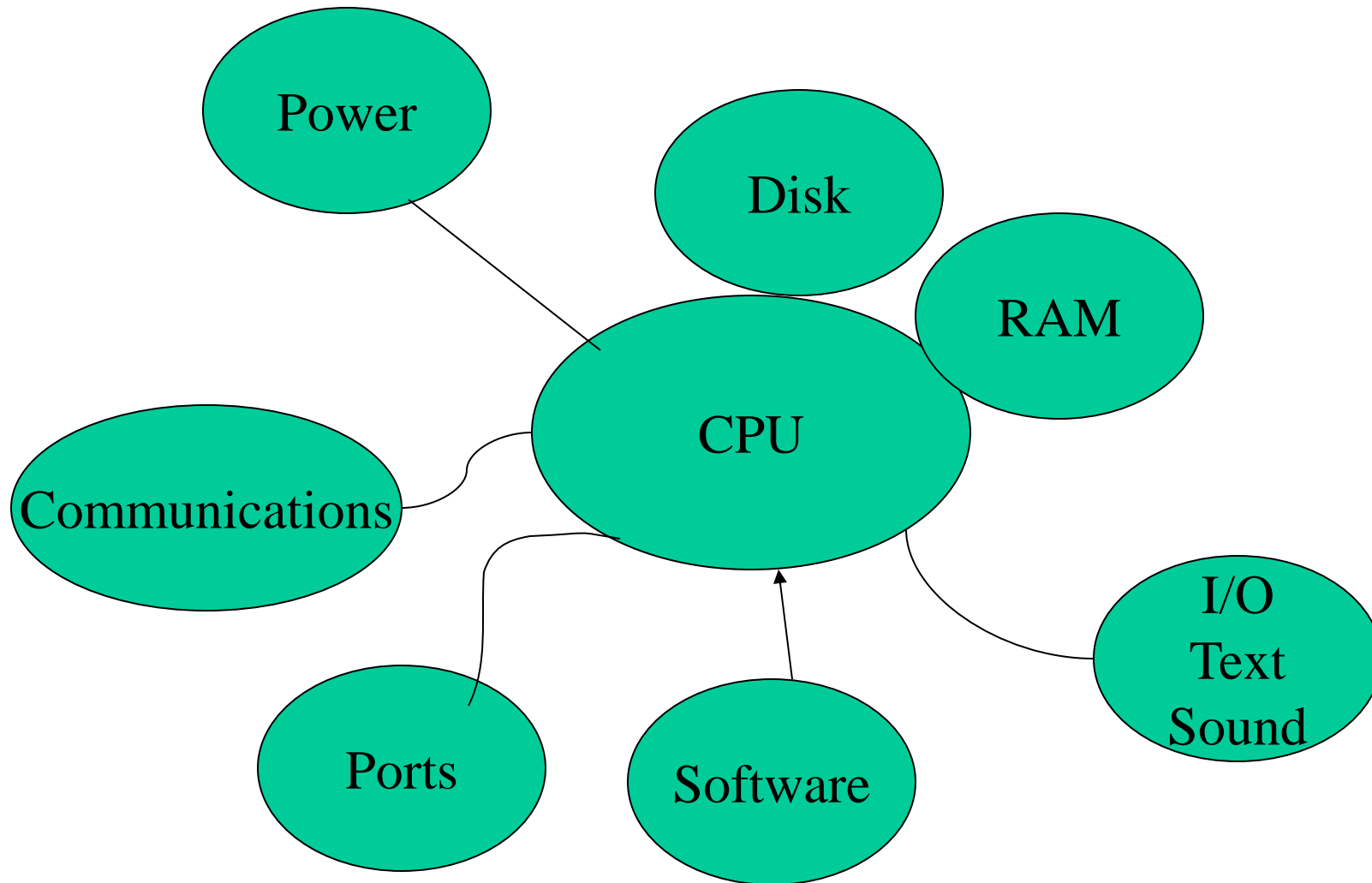
# Co jest w Komputerze?

---

- Processor            *mózg*
- Memory             *notatki*
- Disk                 *trwała pamięć*
- I/O                  *komunikacja (poprzez zmysły)*
  
- Software            *sposoby, metody, algorytmy*

# Z czego jest zbudowany komputer?

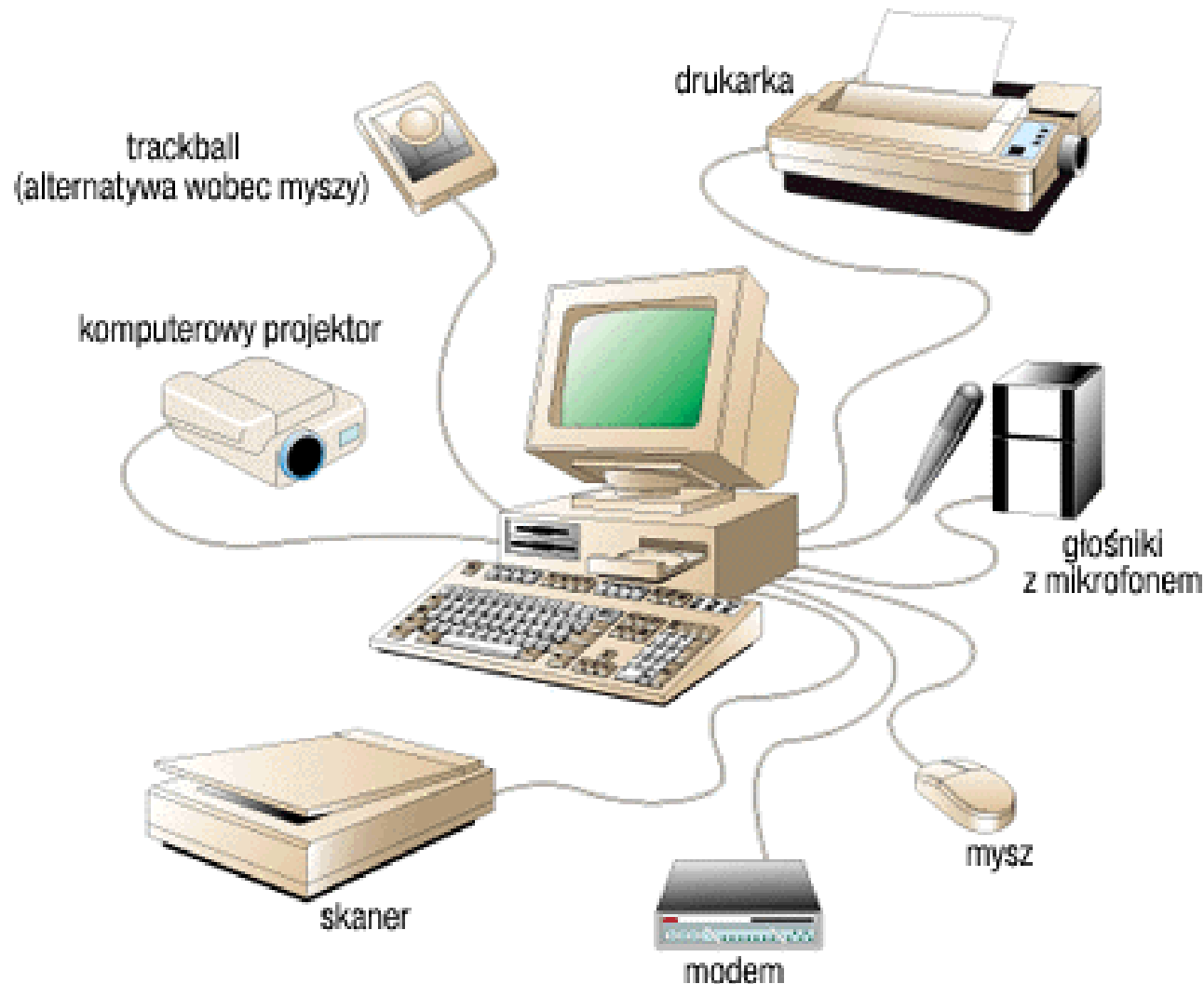
---





# Z czego jest zbudowany komputer?

---



# Luty 2018: Najlepszy zestaw PC do 3650 zł

---

- **Procesor:** Intel Core i5-7500 - 1000 zł
- **Płyta główna:** MSI B150M PRO-VDH - 310 zł
- **Pamięć:** 2 x Patriot 4 GB 2400 MHz Viper Elite - 255 zł
- **Karta graficzna:** ZOTAC GeForce GTX 1060 3GB AMP! - 1020 zł
- **SSD:** GoodRam Iridium Pro 240 GB - 360 zł
- **Dysk twardy:** Western Digital Blue 1 TB - 235 zł
- **Zasilacz:** Thermaltake Smart SE 530W - 250 zł
- **Obudowa:** Zalman Z3 Plus - 180 zł
- **RAZEM:** 3620 zł

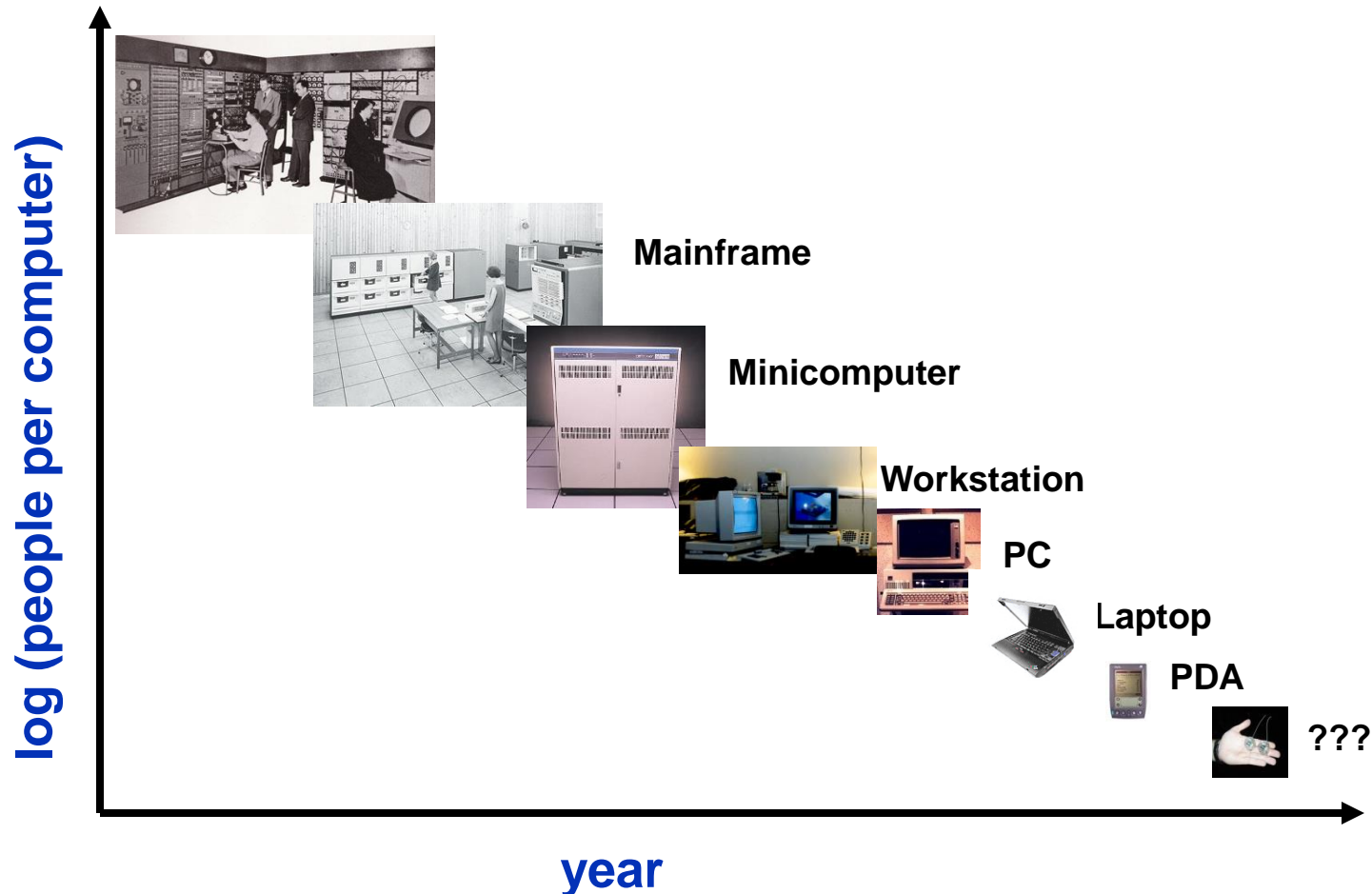
**2020 rok: procesory CPU staniały, a karty graficzne GPU zdrożały -- dlaczego?**

# More for Less --Moore's Law

---

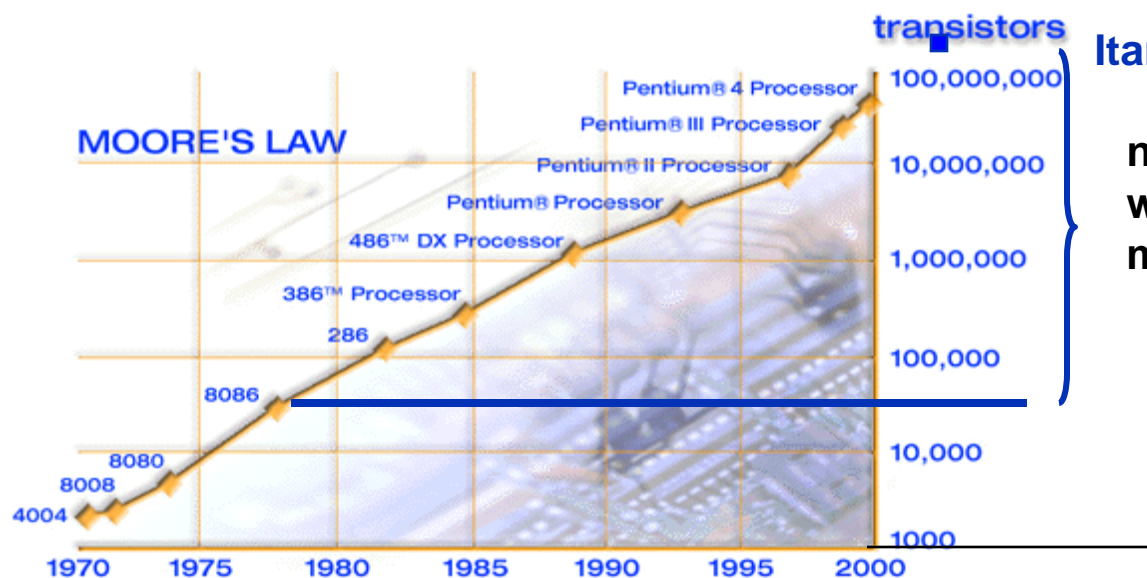
- 1985 (w Polsce komuna-PRL)
  - Komputer w laboratorium (teraz w domu a nawet w komórce)
    - Cena \$ 150 000 (teraz < \$ 1 000)
    - Procesor (CPU) 700 KHz (teraz ~3 GHz i wiele rdzeni)
    - Pamięć RAM 1 MB (teraz 8 GB+)
    - Dysk twardy 80 MB (teraz 1TB +, oraz dyski SSD)
    - CD-ROM właśnie wynaleziony (teraz DVD+)
    - Połączenie z Internetem 9600 bps (teraz 100 Mbps+)

# Moore's Law (contd)



\* This slide and the next lifted from a presentation by David Culler

# Moore's Law (stan na 2005 rok)



Itanium2 (241M )

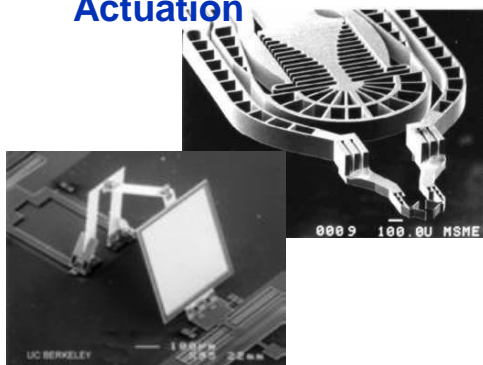
nearly a thousand 8086's  
would fit in a modern  
microprocessor

Wkrótce:

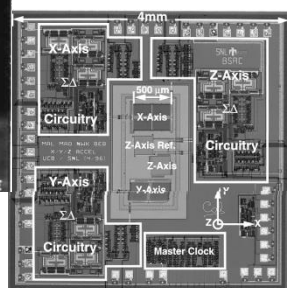


moneta 1 cent  
USA

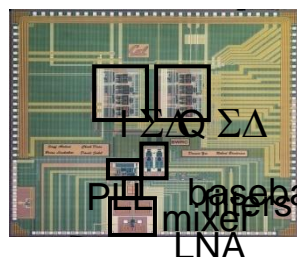
**Actuation**



**Sensing**



**Communication**



**Processing & Storage**

