Wstęp do informatyki

Wykład 2

Uniwersytet Wrocławski

Instytut Informatyki

Realizacja algorytmu przez komputer

Tydzień temu:

- opis algorytmu w języku zrozumiałym dla człowieka: schemat blokowy, pseudokod.

Dziś:

- schemat logiczny komputera,
- zapis algorytmu w formie umożliwiającej wykonanie go przez komputer.

Schemat logiczny komputera

1. **urządzenia („układy”) wejścia i wyjścia:** komunikacja ze światem
2. **procesor**: przetwarza informacje (wykonuje instrukcje), steruje pozostałymi elementami
3. **pamięć**: przechowuje informacje (dane i programy)
4. **magistrale komunikacyjne:** łączą pozostałe elementy

Maszyna RAM

1a. taśma wejściowa: ciąg liczb całkowitych

1b. taśma wyjściowa : ciąg liczb całkowitych, wypisywanych przez program

2. procesor: wykonuje instrukcje

3a. pamięć: komórki 0, 1, 2, 3,…; przechowują liczby całkowite

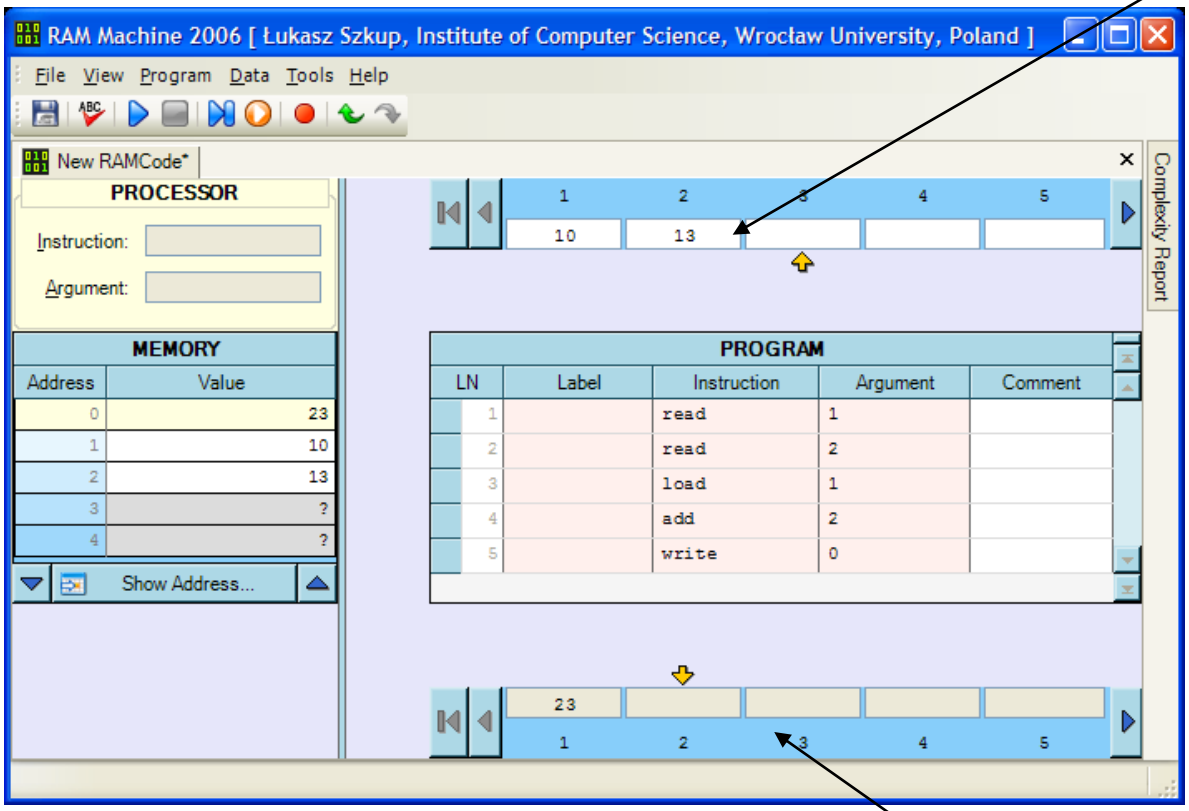
- Komórka 0: akumulator / rejestr

3b. program: ciąg instrukcji do wykonania

Licznik rozkazów: wskazuje na instrukcję, która aktualnie powinna być wykonana

Maszyna RAM

taśma wejściowa



taśma wyjściowa

RAM: szczegóły

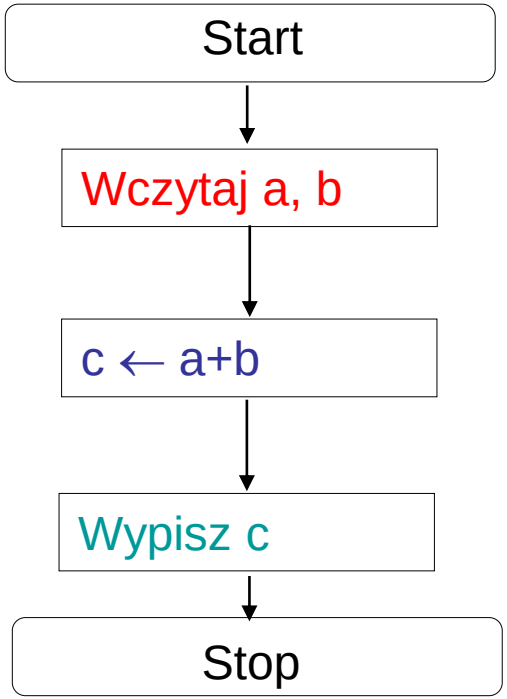
- **Pamięć**: nieskończona liczba komórek etykietowanych 0, 1, 2,..
- Komórka 0 nazywana **rejestrem** lub **akumulatorem**
- Każda komórka może przechowywać **dowolną** liczbę całkowitą
- **Taśma wejściowa** i **wyjściowa** to urządzenia **sekwencyjne**:
  - taśma wejściowa: dane dla programu (wejście)
  - taśma wyjściowa: wyniki programu (wyjście)
  - urządzenie sekwencyjne: dostęp tylko w jednym kierunku, każdy element czytany tylko **jeden** raz.

RAM: cykl główny procesora

- Pobierz instrukcję do wykonania – wskazywaną przez licznik rozkazów
- Pobierz argumenty instrukcji do wykonania
- Wykonaj instrukcję
- Uaktualnij licznik rozkazów (o ile nie został zmieniony przez instrukcję…)

Kod RAM: przykład

Wejście: a, b – liczby
Wyjście: a + b



Skojarzenie zmiennych z komórkami pamięci:
1 – a
2 – b
3 – c

Translacja na kod RAM:

```
Read 1
Read 2
Load 1
Add 2
Store 3
Write 3
Halt
```

Wykonanie programu

Powtarzaj **cykl główny procesora** aż do momentu gdy:

- Licznik rozkazów nie wskazuje na żadną instrukcję

  LUB

- Wykonana została instrukcja HALT

RAM: instrukcje

Składnia: `<etykieta> <instrukcja> <argument>`

Instrukcje wejścia/wyjścia:

Read	Czytaj kolejną liczbę z taśmy wejściowej
Write	Wypisz liczbę na końcu taśmy wyjściowej

Przesłanie do/z akumulatora:

Load	Prześlij do akumulatora
Store	Prześlij zawartość akumulatora do innej komórki

Uwaga: wielkość liter w instrukcjach jest dowolna.

RAM: instrukcje cd.

Instrukcje arytmetyczne:

Add	Dodaj zawartość akumulatora i argument
Sub	Od zawartości akumulatora odejmij argument
Mult	Pomnóż zawartość akumulatora przez argument
Div	Podziel zawartość akumulatora przez argument (dzielenie całkowite, czyli wynik zaokrąglony w dół do liczby całkowitej)

Rezultat jest **zawsze** umieszczany w **akumulatorze**.

RAM: instrukcje cd.

Halt: zakończ działanie programu

Uwaga:

- Halt nie zawsze jest konieczne, gdyż:
  - jeśli licznik rozkazów wskazuje na pozycję niezawierającą żadnej instrukcji, program również się zatrzymuje.

RAM: argumenty

Argument	Znaczenie
⟨liczba⟩	zawartość komórki o numerze liczba
= ⟨liczba⟩	wartość liczba
^ ⟨liczba⟩	**adresowanie pośrednie**: zawartość komórki, której numer znajduje się w komórce o numerze liczba
⟨etykieta⟩	etykieta instrukcji

Argumenty instrukcji: przykład

- load 5:

  pobierz (skopiuj) do akumulatora zawartość komórki numer 5

- load =5:

  umieść liczbę 5 w akumulatorze

- load ^5:

  do akumulatora pobierz (skopiuj) zawartość komórki, której numer znajduje się w komórce 5

Argumenty instrukcji: przykład

div 5: podziel zawartość akumulatora przez zawartość komórki numer 5

div =5: podziel zawartość akumulatora przez liczbę 5

div ^5: podziel zawartość akumulatora przez zawartość komórki, której numer znajduje się w komórce 5

WYNIK jest zawsze umieszczany w akumulatorze!

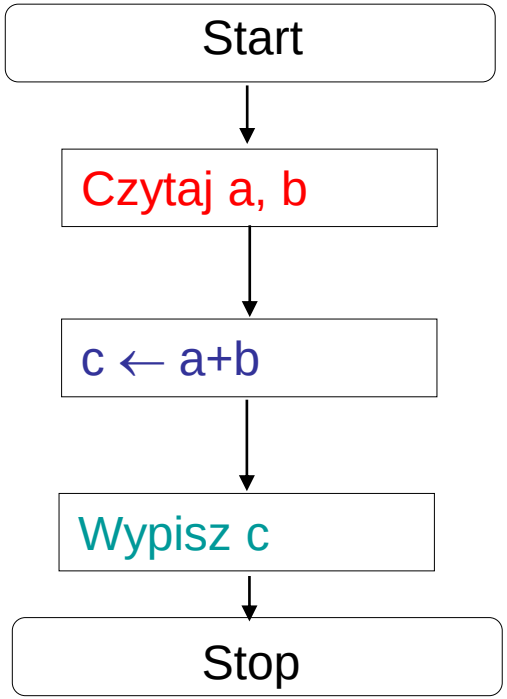
Argumenty instrukcji: przykład

Komórka	Wartość
0	90
1	
2	
3	15
4	
5	3
6	
7	

Instrukcja	Nowa wartość komórki 0
div 5	30 (=90/3)
div =5	18 (=90/5)
div ^5	6 (=90/15)

Kod RAM: przykład

Wejście: a, b – liczby
Wyjście: a + b



Skojarzenie z komórkami pamięci:
1 – a
2 – b
3 – c

Translacja na kod RAM:

```
Read 1
Read 2
Load 1
Add 2
Store 3
Write 3
Halt
```

Instrukcje RAM: skoki

Jump	Skok bezwarunkowy
Jzero	Skok pod warunkiem, że w akumulatorze znajduje się zero
Jgtz	Skok pod warunkiem, że w akumulatorze znajduje się liczba większa od zera

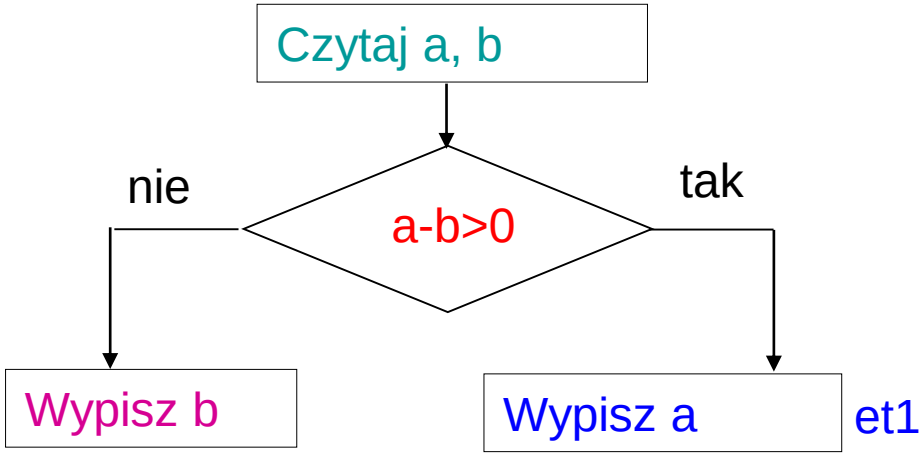
Argument: **etykieta** instrukcji

Etykieta: napis umieszczony „z lewej strony” instrukcji (nie jest wymagana).

RAM przykład: maksimum z dwóch liczb

Wejście: a, b – liczby

Wyjście: maksimum liczb a, b



Skojarzenie zmiennych z kom. pam.:

1 – a

2 – b

Ćwiczenie. Uzupełnij powyższy schemat o bloki Start i Stop.

```
        READ 1
        READ 2
        LOAD 1
        SUB 2
        JGTZ et1
        WRITE 2
        HALT
et1     WRITE 1
        HALT
```

Translacja schematu blokowego na kod RAM

1. Skojarz zmienne z komórkami pamięci!
2. Translacja każdego bloku schematu blokowego osobno! Zaznacz instrukcje kodu RAM odpowiadające każdemu blokowi!
3. Przekształć warunki sprawdzane w schemacie blokowym do jednej z postaci

   ⟨wyrazenie⟩ = 0

   ⟨wyrazenie⟩ > 0

   Używaj skoków aby zmienić odpowiednio licznik rozkazów (przejść do odpowiedniego fragmentu programu).

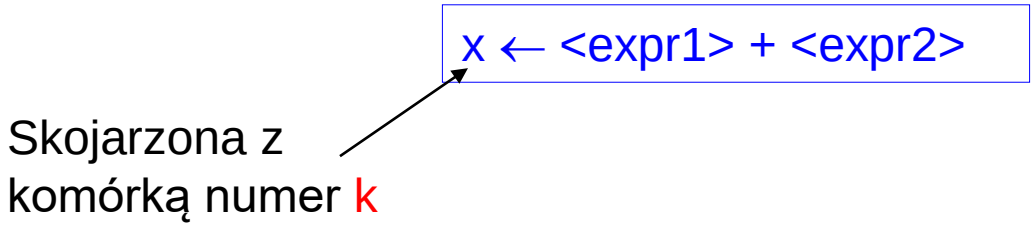
Translacja schematu blokowego na kod RAM

UWAGA!

Na **ćwiczeniach należy** prezentować kod RAM tworząc go w sposób opisany na poprzednim slajdzie:

- Najpierw tworzymy schemat blokowy
- Potem wykonujemy translację schematu blokowego na kod RAM
- Każdy blok schematu „tłumaczymy” osobno na kod RAM!

Schemat blokowy ⇒ kod RAM



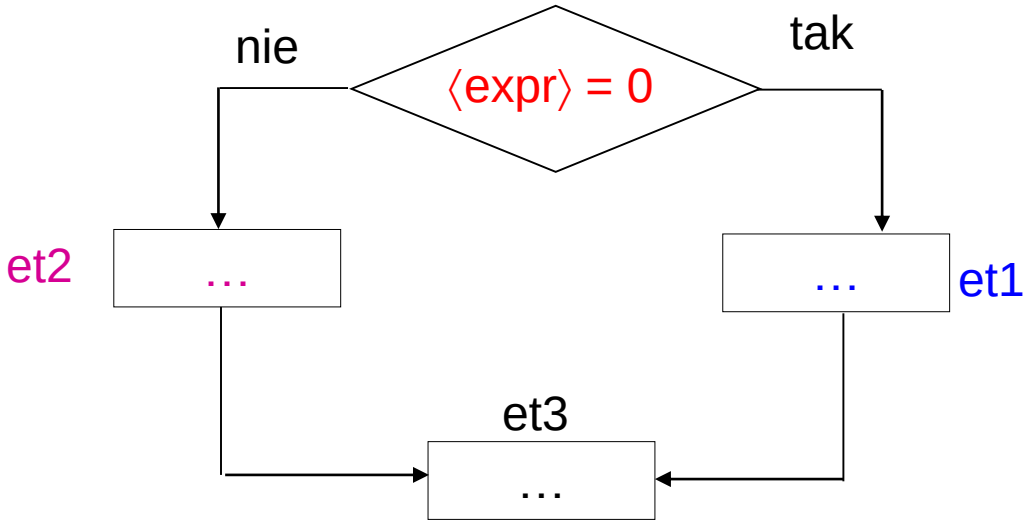
- oblicz ⟨expr1⟩, zachowaj w pewnej komórce i
- oblicz ⟨expr2⟩, zachowaj w pewnej komórce j
- dodaj ⟨expr1⟩ do ⟨expr2⟩ :

  load i

  add j
- zachowaj wynik w komórce k:

  store k

Schemat blokowy ⇒ RAM



- oblicz ⟨expr⟩, wynik masz w akumulatorze;
- wykonaj skok warunkowy:

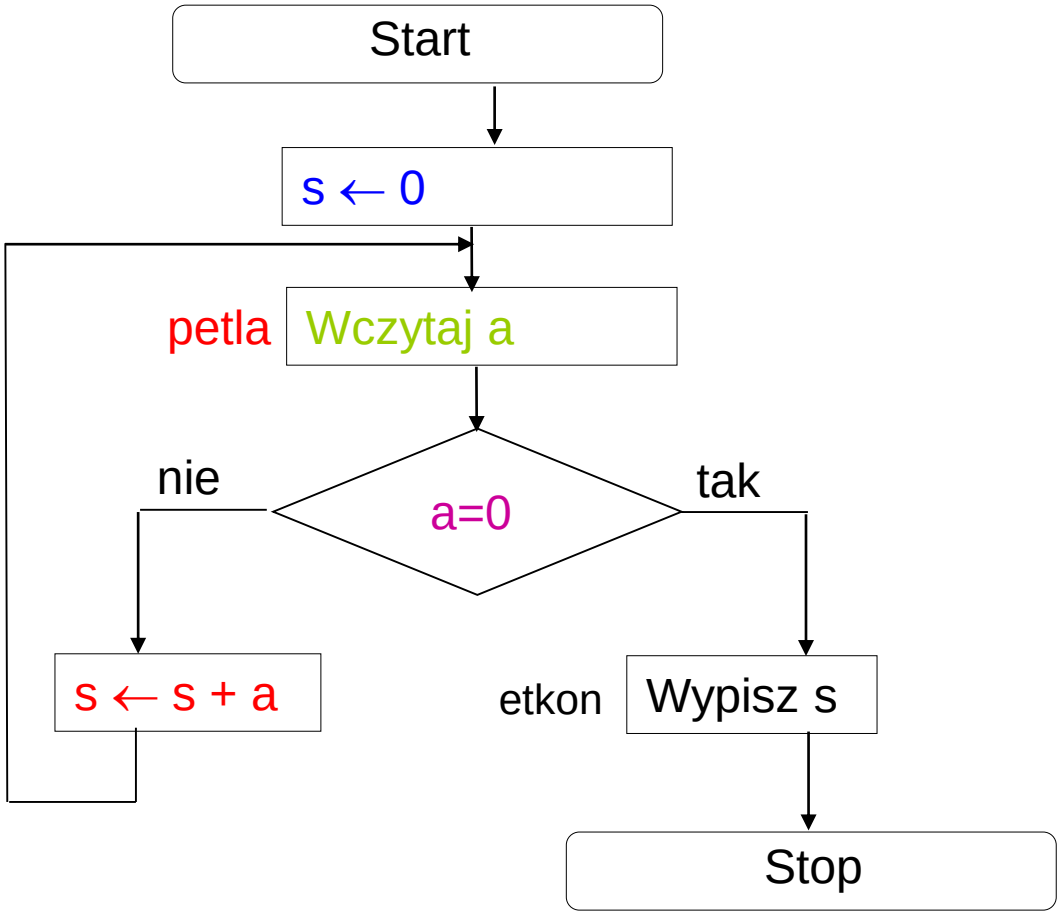
  jzero et1
- Kod odpowiadający „nie” podaj pod instrukcją skoku

```
      … ⟨expr⟩ …
      jzero et1
et2   …
      …
      jump et3
et1   …
      …
      jump et3
et3   …
```

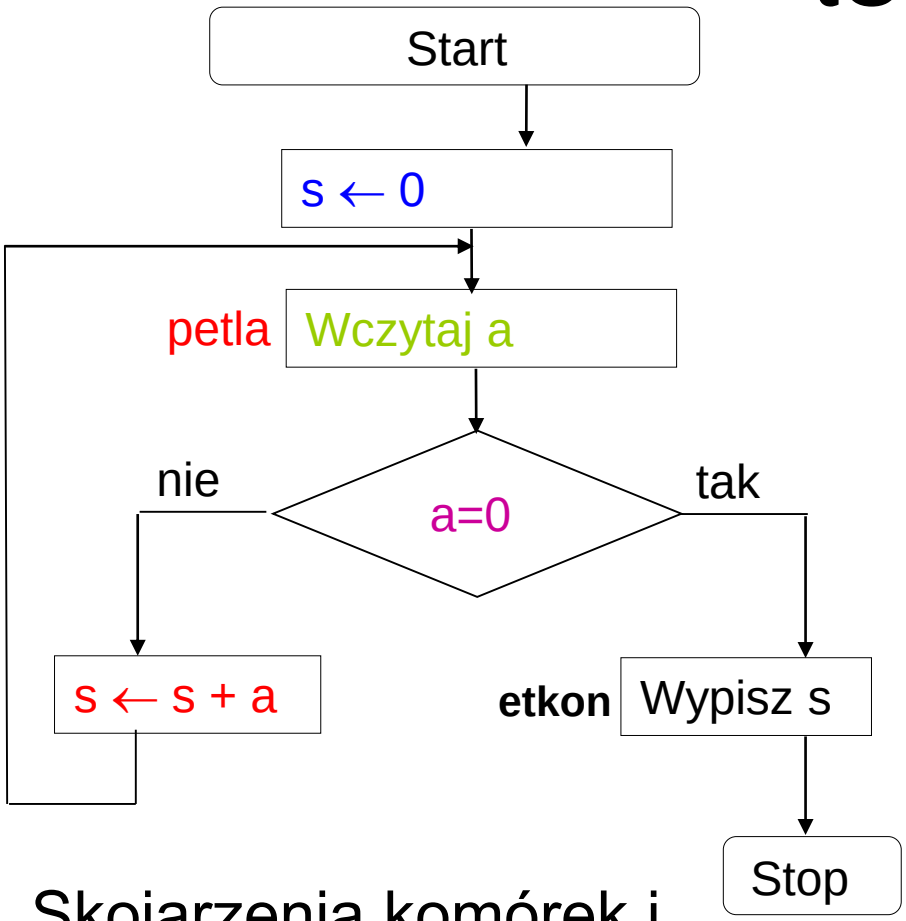
Suma ciągu liczb

Wejście: ciąg liczb zakończony zerem!

Wyjście: suma wszystkich liczb w ciągu



Suma ciągu liczb, cd.




```
        load =0
        store 2
petla   read 1
        load 1
        jzero etkon
        load 2
        add 1
        store 2
        jump petla
etkon   write 2
```

Skojarzenia komórek i zmiennych:

1 – a

2 – s

Suma ciągu liczb – złożoność czasowa

Rozmiar danych:

n – liczba elementów do zsumowania (bez ostatniego zera)

„Złożoność” czasowa schematu blokowego:

1 + 3n + 3 = **3n + 4** **[n – długość ciągu, bez końcowego zera]**

„Złożoność” czasowa programu w kodzie RAM:

2 + 7n + 4 = **7n + 6**

„Która złożoność poprawna”?

Złożoność **asymptotyczna**

Złożoność **asymptotyczna**

Notacja „dużego O”:

- ignorujemy **stałe** (konkretne liczby) zależne od zapisu algorytmu / modelu obliczeń;
- np. $10n$, $n+7$, $112n$ są „podobne” (funkcje liniowe);
- Lecz $100n$ i $2n^2$ istotnie się różnią!

  Pyt.: Która z nich jest „większa”?

  Odp.: $2n^2$

  Pyt.: Dlaczego?

  Odp.: Szybciej rośnie!

Złożoność **asymptotyczna**

Notacja „dużego O” formalnie:

$$f(n) = O(g(n))$$

wtedy i tylko wtedy gdy (wtw)

istnieje stała $c>0$ i liczba naturalna $m>0$ takie, że $f(n) \leq c \cdot g(n)$ dla każdego $n>m$.

Złożoność asymptotyczna - intuicja

$f(n) = O(g(n))$

wtw

g(n) rośnie co najmniej tak samo szybko jak f(n) dla $n \to \infty$

Złożoność asymptotyczna – przykłady

$100 \cdot n = O(n^2)$, gdyż

$100 \cdot n \leq 100 \cdot n^2$ dla każdego $n>0$.

$55 \cdot n^2+100n-10 = O(n^2)$, gdyż

$55 \cdot n^2+100n-10 \leq 57 \cdot n^2$ dla każdego $n>60$.

$n^2 = O(2^n)$, gdyż

$n^2 \leq 1 \cdot 2^n$ dla każdego $n>5$. [jak to pokazać?]

Złożoność asymptotyczna – przykłady

$n^2 / 1000 \neq O(n)$, gdyż dla każdego $c>0$ mamy:

$$n^2 / 1000 > c \cdot n$$

dla nieskończenie wielu n.

W szczególności, nierówność

$$n^2 / 1000 > c \cdot n$$

zachodzi dla każdego n spełniającego warunek $n > 2 \cdot 1000 \cdot \max(c,1)$.

Złożoność asymptotyczna – przykłady

$100 \cdot n = O(n)$

$100 \cdot n = O(n^2)$

$n^2 / 1000 \neq O(n)$

$77 \cdot n + 333 = O(n)$

$55 \cdot n^2 + 100n - 10 = O(n^2)$

$55 \cdot n^2 + 100n - 10 = O(n^3)$

$55 \cdot n^2 + 100n - 10 \neq O(n)$

$n^2 = O(2^n)$

$2^n \neq O(n^2)$

Złożoność asymptotyczna – narzędzia

(zał. f(n), g(n)≥0 dla każdego n>0)

Fakt. Jeżeli $\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$ to f(n) = O (g(n)).

Fakt. Jeżeli $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ to f(n) ≠ O (g(n)).

Uwaga: może zachodzić f(n)=O(g(n)) gdy granica wyrażenia f(n)/g(n) nie istnieje.

Suma ciągu liczb – złożoność czasowa

Rozmiar danych:

n – liczba elementów do zsumowania (bez ostatniego zera)

„Złożoność” czasowa schematu blokowego:

1 + 3n + 3 = **3n + 4**

„Złożoność” czasowa programu w kodzie RAM:

2 + 7n + 4 = **7n + 6**

Złożoność asymptotyczna: O(n)

Sortowanie przez zliczanie

Sortowanie przez zliczanie

Problem

Wejście:

n – liczba naturalna;

$a_1,a_2,\ldots,a_n$ – ciąg liczb naturalnych z przedziału [0,9]

Wyjście: elementy $a_1,a_2,\ldots,a_n$ wypisane w porządku niemalejącym

Przykład

Wejście: 11; 0, 7, 9, 9, 2, 1, 6, 7, 6, 5, 5

Wyjście: 0, 1, 2, 5, 5, 6, 6, 7, 7, 9, 9

Sortowanie przez zliczanie

Algorytm

- Nadaj licznikom K[0],…,K[9] wartość zero.
- Wczytaj n
- Dla i=n, n - 1, n - 2,…,1 powtarzaj:
  - Wczytaj a
  - K[a] ← K[a]+1
- Dla i=0, 1, 2, …,9 powtarzaj:
  - Powtórz K[i] razy:
    - Wypisz i

licznik: **„zlicza” liczbę wystąpień pewnego zdarzenia**

Sortowanie przez zliczanie

Implementacja liczników K[0],…,K[9] na maszynie RAM:

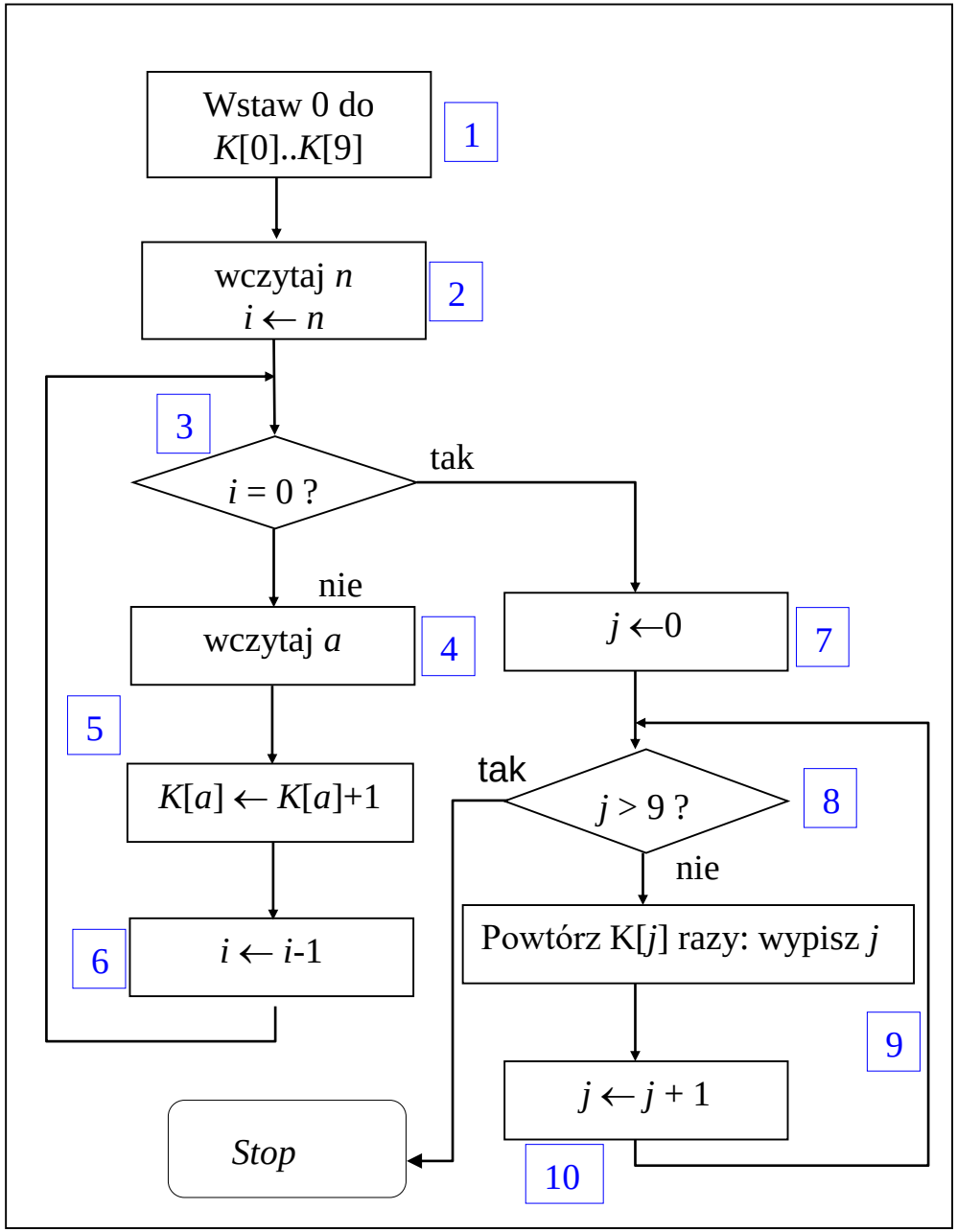
- K[0],…,K[9] to sąsiednie komórki pamięci

Co to daje?

- Adres komórki odpowiadającej licznikowi K[i] jest równy adresowi K[0] plus i

x	K[0]
x+1	K[1]
x+2	K[2]
x+3	K[3]
x+4	K[4]
x+5	K[5]
x+6	K[6]
x+7	K[7]
x+8	K[8]
x+9	K[9]

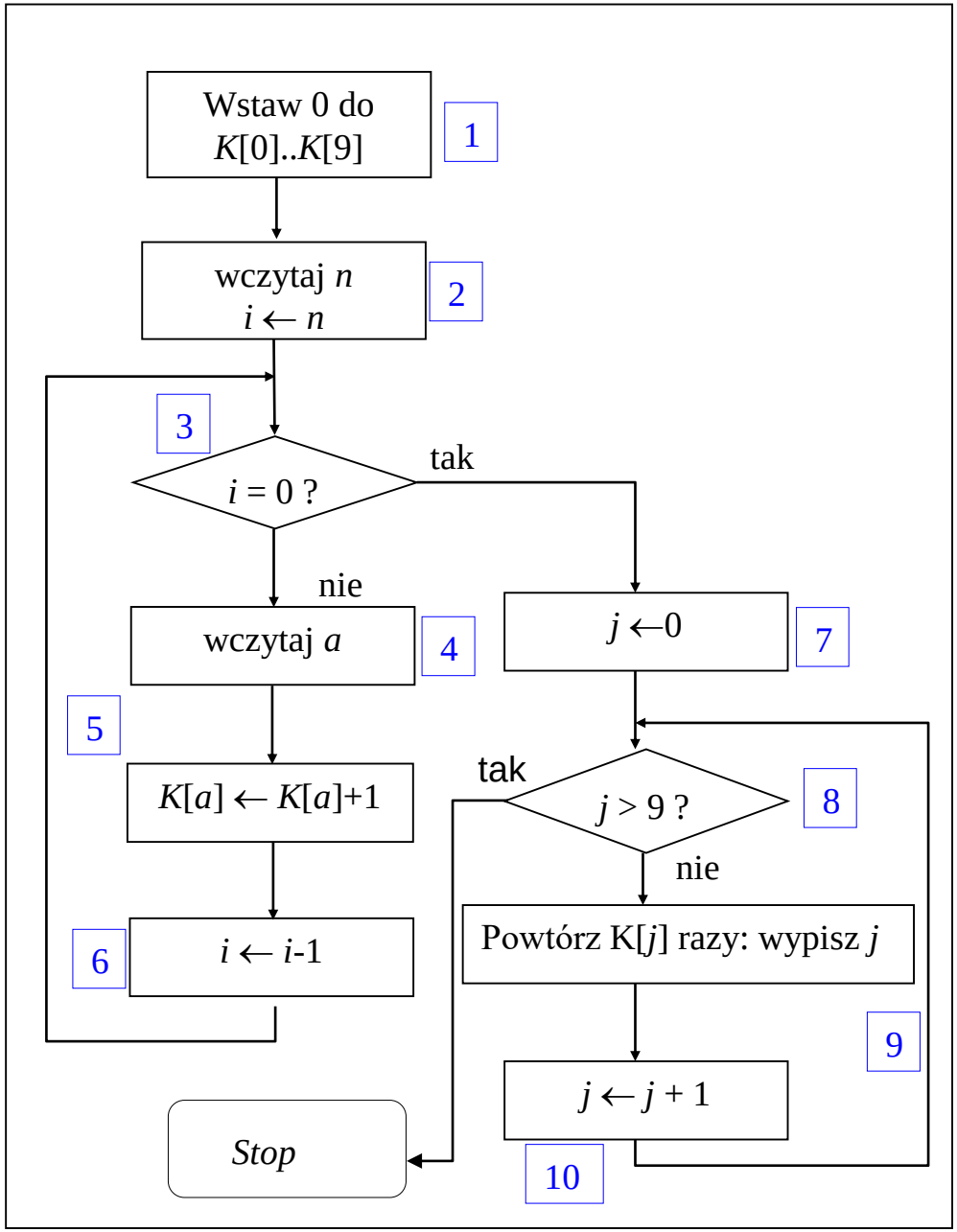
Sortowanie przez zliczanie



Algorytm

- Nadaj licznikom K[0],…,K[9] wartość zero.
- Wczytaj n
- Dla i=n, n - 1, n - 2,…,1 powtarzaj:
  - Wczytaj a
  - K[a] ← K[a]+1
- Dla j=0, 1, 2, …,9 powtarzaj:
  - Powtórz K[j] razy:
    - Wypisz j

Sortowanie przez zliczanie

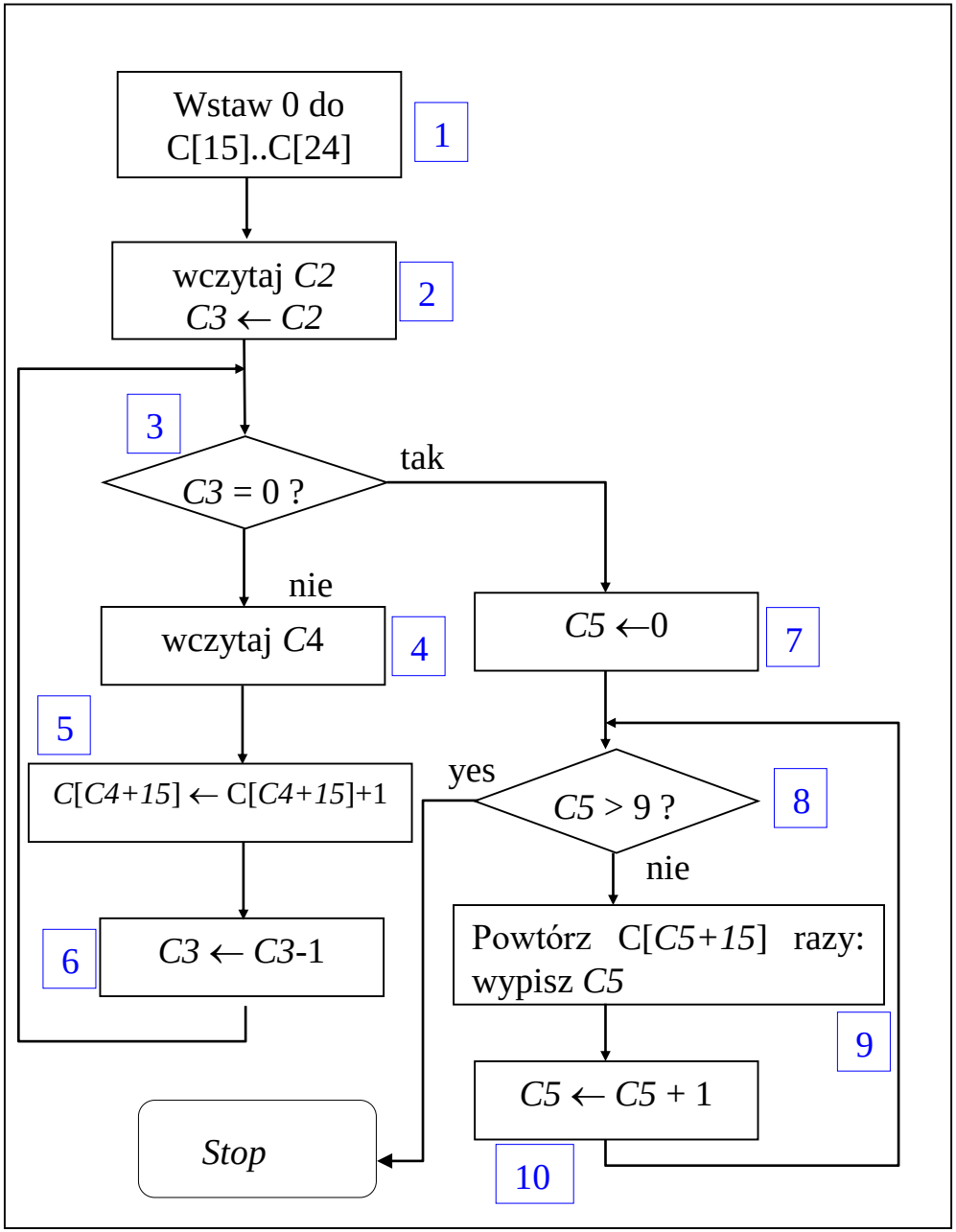


Skojarzenia:

`K[0]..K[9]:`
`  komórki 15..24`
`n  : C2`
`i  : C3`
`a  : C4`
`j  : C5`

Uwaga: komórkę C_i będziemy też oznaczać przez `C[i]`

Sortowanie przez zliczanie



Skojarzenia:

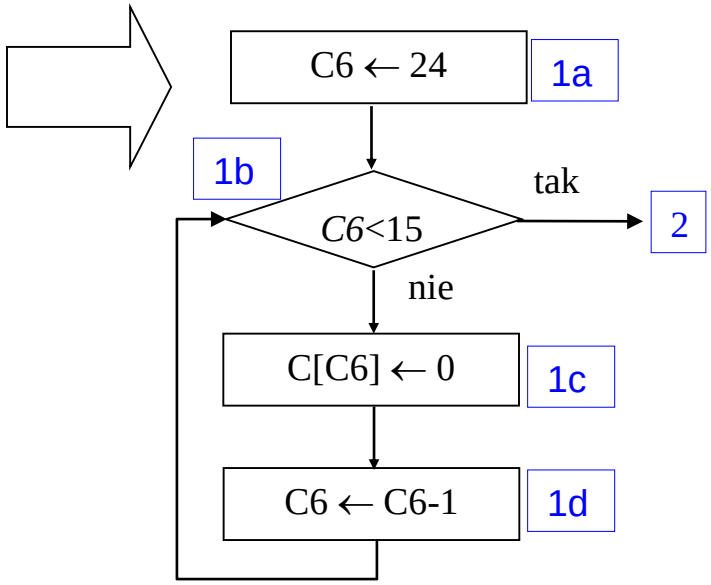
```
K[0]..K[9]:
   komórki 15..24
n   : C2
i   : C3
a   : C4
j   : C5
```

Uwaga: komórkę C_i będziemy też oznaczać przez `C[i]`

Sortowanie przez zliczanie

Wstaw 0 do C[15]..C[24]

1




```
      load  =24 #1a
      store 6
L1b:  load  =15 #1b
      sub   6
      jgtz  L2
      load  =0  #1c
      store ^6
      load  6   #1d
      sub   =1
      store 6
      jump  L1b
```

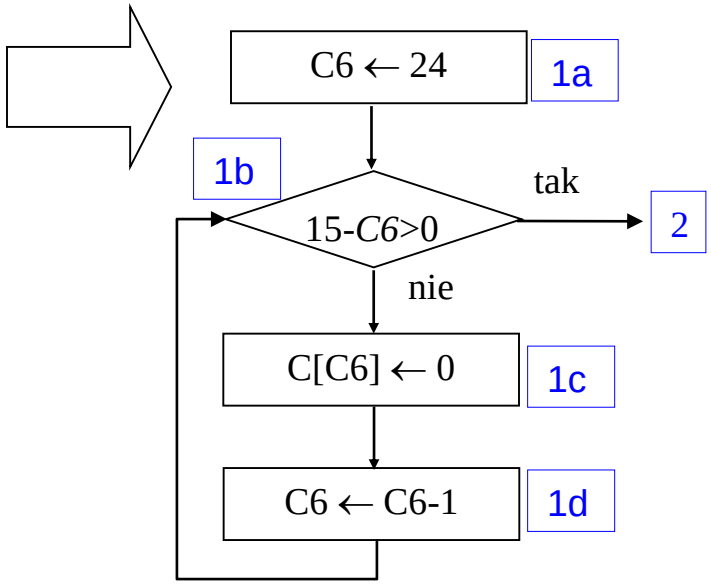
Skojarzenia:

```
K[0]..K[9]:
   komórki 15..24
```

Sortowanie przez zliczanie

Wstaw 0 do C[15]..C[24]

1




```
      load  =24 #1a
      store 6
L1b:  load  =15 #1b
      sub   6
      jgtz  L2
      load  =0  #1c
      store ^6
      load  6   #1d
      sub   =1
      store 6
      jump  L1b
```

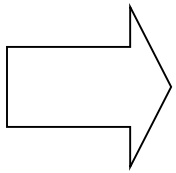
Skojarzenia:

```
K[0]..K[9]:
   komórki 15..24
```

Sortowanie przez zliczanie

$C[C4+15] \leftarrow C[C4+15]+1$

5

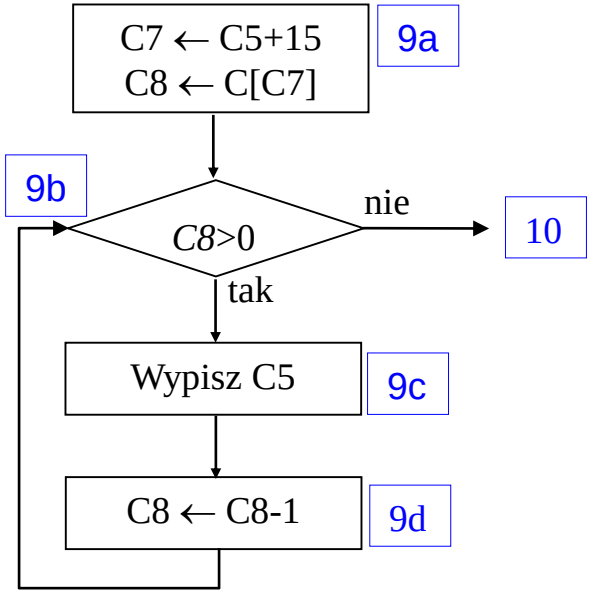


```
load  4   #5
add   =15
store 7
load  ^7
add   =1
store ^7
```

Sortowanie przez zliczanie

9 Powtórz C[*C5+15*] razy: wypisz *C5*

```
      load  5    #9a
      add   =15
      store 7
      load  ^7
      store 8
L9b:  load  8    #9b
      jgtz  L9c
      jump  L10
L9c:  write 5    #9c
      load  8    #9d
      sub   =1
      store 8
      jump  L9b
```



Sortowanie przez zliczanie ogólniej

Problem
Wejście:
n,k – liczby naturalne;
$a_1,a_2,\ldots,a_n$ – ciąg liczb całkowitych z zakresu [0,k].

Wyjście:
elementy ciągu $a_1,a_2,\ldots,a_n$ wypisane w porządku niemalejących

Rozwiązanie
Zamiast **liczników** K[0],…K[9], użyj liczników K[0],…K[k].

Sortowanie przez zliczanie – złożoność

Rozmiar danych:
n

Pamięć:
proporcjonalna do k, czyli O(k)

Czas:
proporcjonalny do n+k, czyli O(n+k) :
- gdy $n > k$: $O(n + k) = O(n)$
- gdy $n \leq k$: $O(n + k) = O(k)$

Sortowanie przez zliczanie – złożoność

Czas:

proporcjonalny do n+k, czyli O(n+k) :

gdy k = O(n): O(n + k) = O(n)

gdy n = O(k): O(n + k) = O(k)

Uwaga

Jeśli k >> n (czyli k jest „istotnie większe” od n):

- potrzebujemy większej pamięci niż długość ciągu!
- czas obliczeń proporcjonalny do k, czyli znacznie większy od n (czy taki algorytm jest „praktyczny”?).

Podsumowanie (1)

1. Maszyna RAM – model komputera:
   - „Zmienna”– komórka w pamięci
   - Operacje arytmetyczne – wykonaj operację na akumulatorze i (ewentualnie) zawartości innej komórki
   - Cykl główny procesora
   - Pętle – skoki – licznik rozkazów.
2. Adresowanie pośrednie i jego zastosowania!
3. Notacja asymptotyczna w szacowaniu złożoności
4. Problem sortowania; sortowanie przez zliczanie: jak działa? kiedy przydatne?

Podsumowanie (2)

Pamiętaj na ćwiczeniach, aby kod blokowy tłumaczyć na kod RAM w następujący sposób:

- Skojarz zmienne z komórkami pamięci
- „Tłumacz” każdy element schematu blokowego osobno! Zaznacz instrukcje kodu RAM odpowiadające każdemu blokowi!
- Przekształć warunki sprawdzane w schemacie blokowym do jednej z postaci

  `⟨wyrazenie⟩ = 0`

  `⟨wyrazenie⟩ > 0`

  Używaj skoków aby zmienić odpowiednio licznik rozkazów (przejść do odpowiedniego fragmentu programu).

Podsumowanie (3)

Rozmiar danych:

- Rozmiar danych wynika ze specyfikacji problemu, nie konkretnego algorytmu
- Rozmiar danych można ustalić na podstawie opisu wejścia w specyfikacji podanego problemu

Złożoność pamięciowa:

- Złożoność pamięciowa algorytmu to rozmiar wykorzystanej pamięci (liczba zmiennych lub komórek maszyny RAM)
- Złożoność pamięciowa może być zarówno mniejsza od rozmiaru danych jak i od niego większa