Informatyka 1 (EZ1F1002)

Politechnika Białostocka - Wydział Elektryczny

Elektrotechnika, semestr I, studia niestacjonarne I stopnia

Rok akademicki 2024/2025

Wykład nr 5 (15.11.2024)

dr inż. Jarosław Forenc

Plan wykładu nr 5

- Algorytmy komputerowe
  - złożoność obliczeniowa
  - algorytmy sortowania
- Architektura von Neumanna i architektura harwardzka
- Struktura i funkcjonowanie komputera
  - procesor, rozkazy, przerwania, magistrala
  - pamięć komputerowa, hierarchia pamięci
  - pamięć podręczna
- Język C
  - operator warunkowy, instrukcja switch
  - pętla for, operatory ++ i −−
  - pętle while i do…while

Złożoność obliczeniowa

- Złożoność obliczeniowa algorytmu - funkcja opisującą zależność między liczbą danych a liczbą operacji wykonywanych przez algorytm
- Notacja O („duże O”) - wyraża złożoność matematyczną algorytmu
- Do wyznaczenia złożoności bierze się pod uwagę liczbę dominujących operacji wykonywanych w algorytmie
- Przykład zapisu: $O(n^2)$
  - po literze O występuje wyrażenie w nawiasach zawierające literę n, która oznacza liczbę elementów, na których działa algorytm
- W funkcji opisującej złożoność bierze się pod uwagę tylko najistotniejszy składnik, np.

$f(n) = n^2+2n \rightarrow O(n^2)$ $\quad$ $f(n) = n^2+n-5 \rightarrow O(n^2)$

Notacja O („duże O”)

- Porównanie najczęściej występujących złożoności:

Elementy (n)	O(log n)	O(n)	O(n log n)	$O(n^2)$	$O(n^3)$	$O(2^n)$
10	3	10	33	100	1 000	1024
100	7	100	664	10 000	1 000 000	$1{,}27 \cdot 10^{30}$
1 000	10	1 000	9 966	1 000 000	10^9	$1{,}07 \cdot 10^{301}$
10 000	13	10 000	132 877	10^8	10^{12}	$1{,}99 \cdot 10^{3010}$

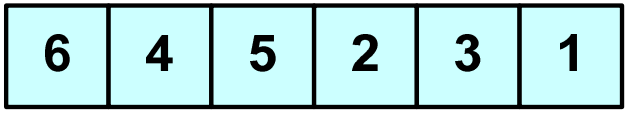
O(log n) - logarytmiczna (np. przeszukiwanie binarne)
O(n) - liniowa (np. porównywanie łańcuchów znaków)
O(n log n) - liniowo-logarytmiczna (np. sortowanie szybkie)
$O(n^2)$ - kwadratowa (np. proste algorytmy sortowania)
$O(n^3)$ - sześcienna (np. mnożenie macierzy)
$O(2^n)$ - wykładnicza (np. problem komiwojażera)

Sortowanie

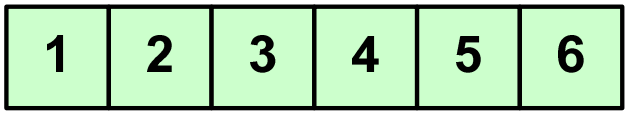
- Sortowanie polega na uporządkowaniu zbioru danych względem pewnych cech charakterystycznych każdego elementu tego zbioru (wartości każdego elementu)
- W przypadku liczb, sortowanie polega na znalezieniu kolejności liczb zgodnej z relacją ≤ lub ≥

Przykład:

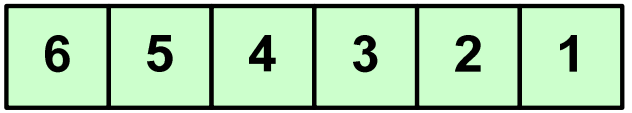
- Tablica nieposortowana:



- Tablica posortowana zgodnie z relacją ≤ (od najmniejszej do największej liczby):



- Tablica posortowana zgodnie z relacją ≥ (od największej do najmniejszej liczby):



Sortowanie

- W przypadku słów sortowanie polega na ustawieniu ich w porządku alfabetycznym (leksykograficznym)

Przykład:

- Tablica nieposortowana:

Paweł	Piotr	Adrian	Ela	Ola	Henryk

- Tablice posortowane:

Adrian	Ela	Henryk	Ola	Paweł	Piotr

Piotr	Paweł	Ola	Henryk	Ela	Adrian

Sortowanie

- W praktyce sortowanie sprowadza się do porządkowanie danych na podstawie porównania - porównywany element to klucz

Przykład:

- Tablica nieposortowana (imię, nazwisko, wiek):

Piotr	Ola	Paweł	Jan	Ela	Magda
Kowalski	Nowak	Wójcik	Kamiński	Król	Mazur
25	18	23	20	22	15

- Tablica posortowana (klucz - nazwisko):

Jan	Piotr	Ela	Magda	Ola	Paweł
Kamiński	Kowalski	Król	Mazur	Nowak	Wójcik
20	25	22	15	18	23

- Tablica posortowana (klucz - wiek):

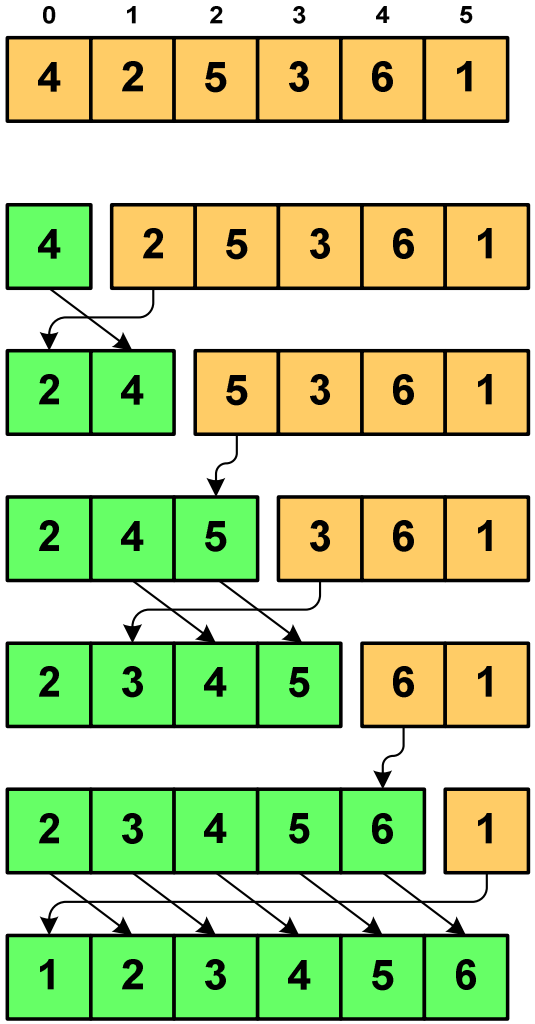
Magda	Ola	Jan	Ela	Paweł	Piotr
Mazur	Nowak	Kamiński	Król	Wójcik	Kowalski
15	18	20	22	23	25

Sortowanie

- Po co stosować sortowanie?
  - posortowane elementy można szybciej zlokalizować
  - posortowane elementy można przedstawić w czytelniejszy sposób
- Przykładowe algorytmy sortowania
  - proste wstawianie (insertion sort)
  - proste wybieranie (selection sort)
  - bąbelkowe (bubble sort)
  - szybkie (quick sort)
  - przez scalanie (merge sort)
  - kubełkowe / przez zliczanie (bucket sort)

Proste wstawianie (insertion sort)

Przykład:

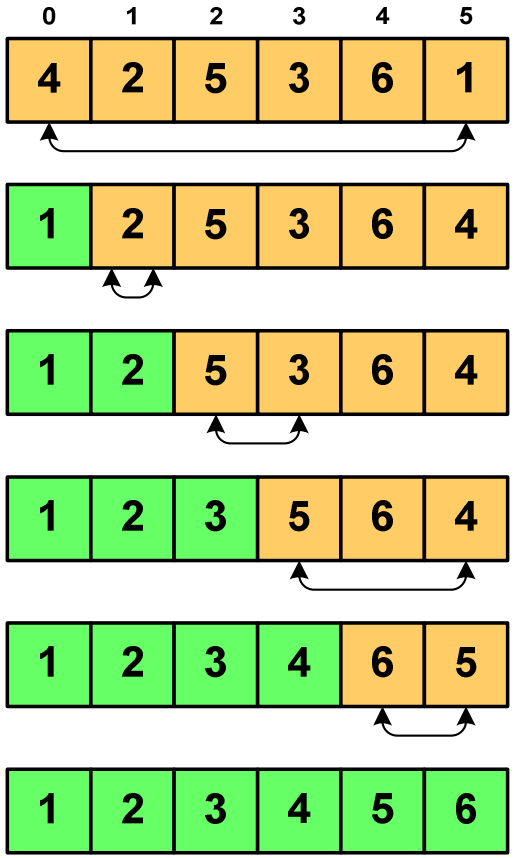


Program w języku C:

```
int main(void)
{
   int tab[N],i,j,tmp;
   // ...
   for (i=1; i<N; i++)
   {
      j=i;
      tmp=tab[i];
      while (tab[j-1]>tmp && j>0)
      {
         tab[j]=tab[j-1];
         j--;
      }
      tab[j]=tmp;
   }
}
```

Proste wybieranie (selection sort)

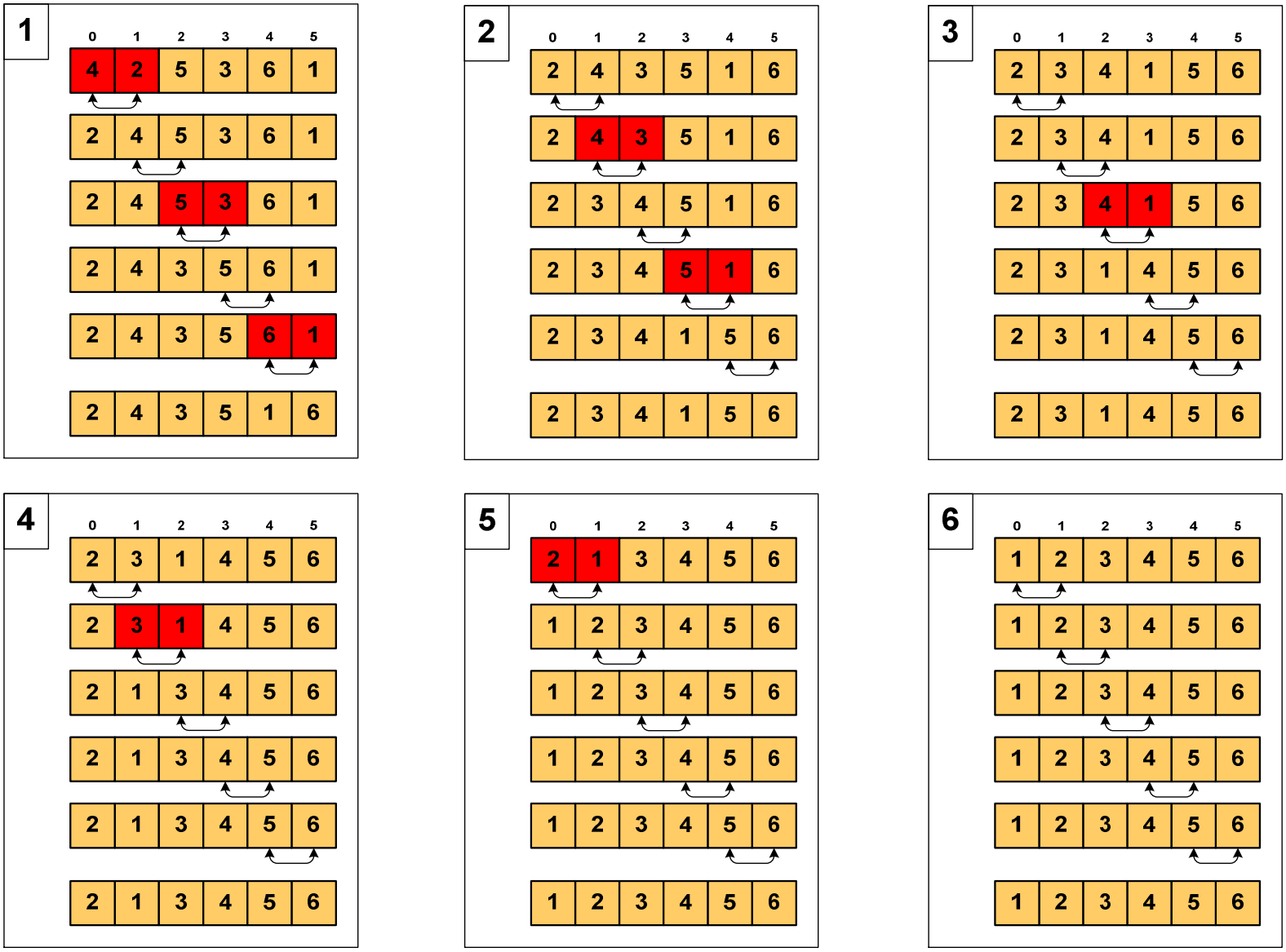
Przykład:



Program w języku C:

```
int main(void)
{
    int tab[N],i,j,k,tmp;
    // ...
    for (i=0;i<N-1;i++)
    {
        k=i;
        for (j=i+1; j<N; j++)
            if (tab[k]>=tab[j])
                k = j;
        tmp = tab[i];
        tab[i] = tab[k];
        tab[k] = tmp;
    }
}
```

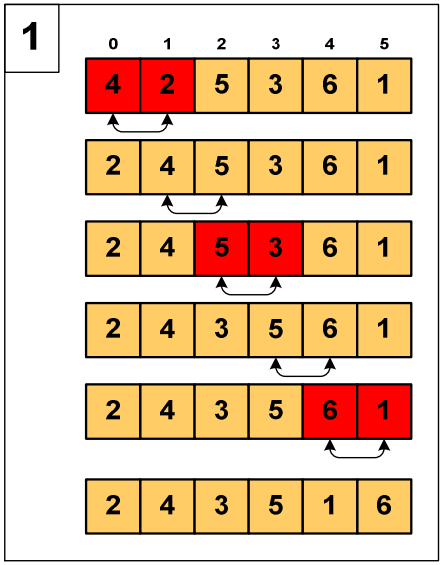
Bąbelkowe (bubble sort)

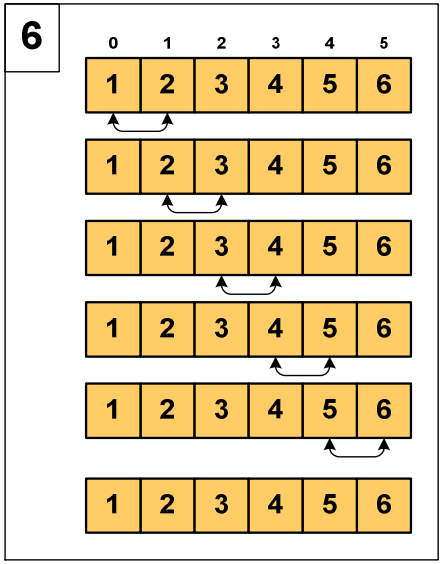


Bąbelkowe (bubble sort)

Program w języku C:

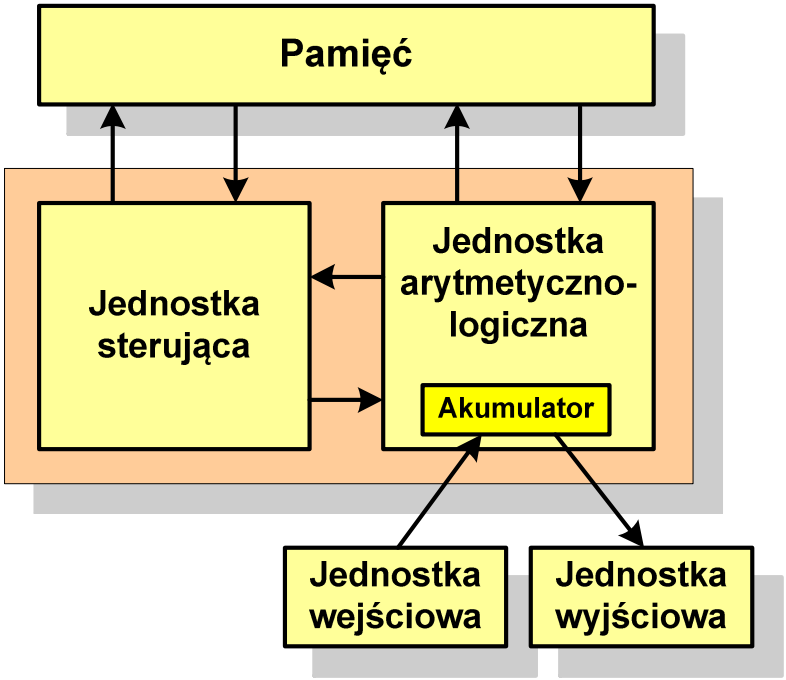
```
int main(void)
{
    int tab[N],i,j,tmp,koniec;
    // ...
    do {
        koniec=1;
        for (i=0;i<N-1;i++)
            if (tab[i]>tab[i+1])
            {
                tmp=tab[i];
                tab[i]=tab[i+1];
                tab[i+1]=tmp;
                koniec=0;
            }
    } while (!koniec);
}
```





Architektura von Neumanna

- Rodzaj architektury komputera, opisanej w 1945 roku przez matematyka Johna von Neumanna
- Inne spotykane nazwy: architektura z Princeton, store-program computer (koncepcja przechowywanego programu)
- Zakłada podział komputera na kilka części:
  - jednostka sterująca (CU - Control Unit)
  - jednostka arytmetyczno-logiczna (ALU - Arithmetic Logic Unit)
  - pamięć główna (memory)
  - urządzenia wejścia-wyjścia (input/output)

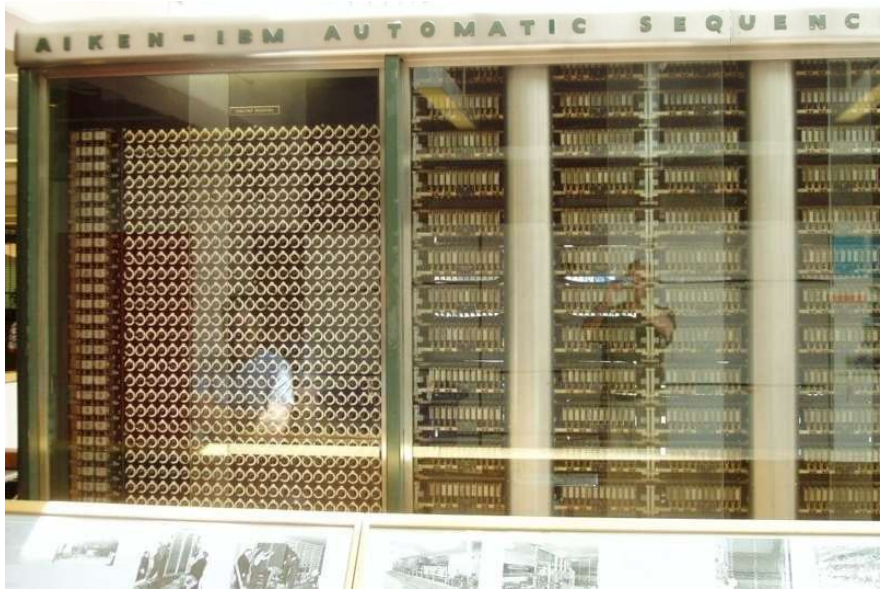


Architektura von Neumanna - podstawowe cechy

- Informacje przechowywane są w komórkach pamięci (cell) o jednakowym rozmiarze, każda komórka ma numer - adres
- Dane oraz instrukcje programu (rozkazy) zakodowane są za pomocą liczb i przechowywane w tej samej pamięci
- Praca komputera to sekwencyjne odczytywanie instrukcji z pamięci komputera i ich wykonywanie w procesorze
- Wykonanie rozkazu:
  - pobranie z pamięci słowa będącego kodem instrukcji
  - pobranie z pamięci danych
  - wykonanie instrukcji
  - zapisanie wyników do pamięci
- Dane i instrukcje czytane są przy wykorzystaniu tej samej magistrali

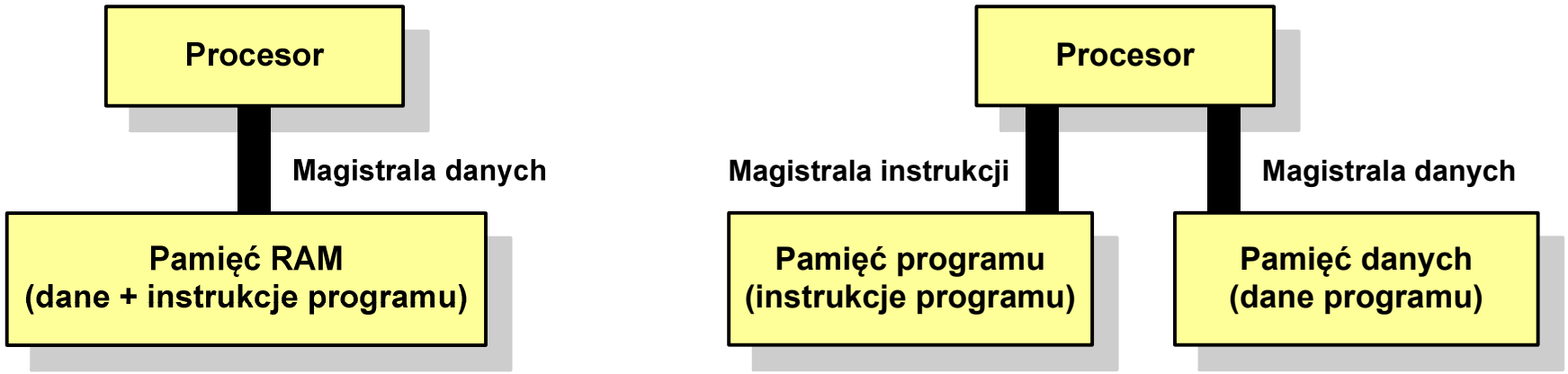
Architektura harwardzka

- Pamięć danych jest oddzielona od pamięci instrukcji
- Procesor może w tym samym czasie czytać instrukcje oraz uzyskiwać dostęp do danych
- Nazwa architektury pochodzi komputera Harward Mark I:
  - pamięć instrukcji - taśma dziurkowana,
    pamięć danych - elektromechaniczne liczniki



Architektura harwardzka i von Neumanna

- W architekturze harwardzkiej pamięć instrukcji i pamięć danych:
  - zajmują różne przestrzenie adresowe
  - mają oddzielne szyny (magistrale) do procesora
  - zaimplementowane są w inny sposób



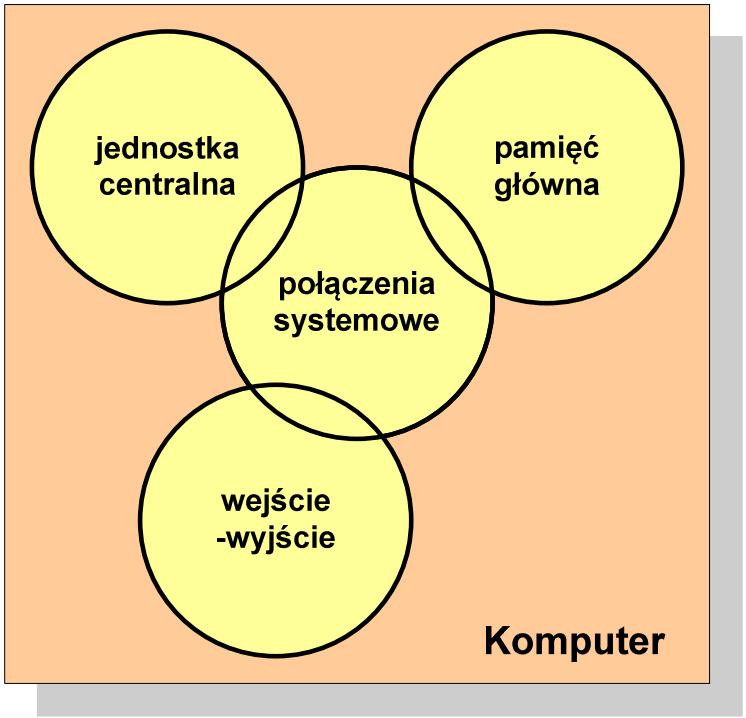
Architektura von Neumanna

Architektura harwardzka

- Zmodyfikowana architektura harwardzka:
  - oddzielone pamięci danych i rozkazów, lecz wykorzystujące wspólną magistralę

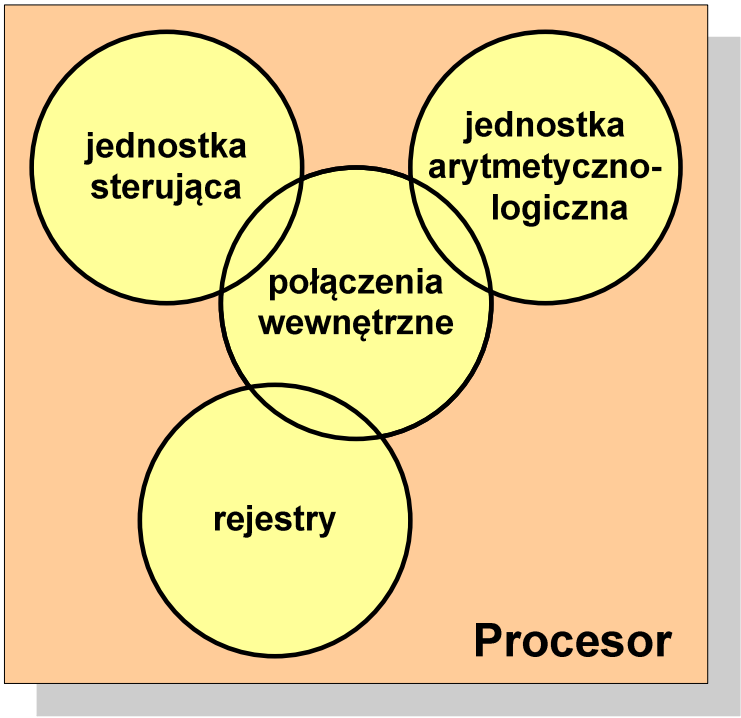
Ogólna struktura systemu komputerowego

- Komputer tworzą cztery główne składniki:
  - procesor (jednostka centralna, CPU) - steruje działaniem komputera i realizuje przetwarzanie danych
  - pamięć główna - przechowuje dane
  - wejście-wyjście - przenosi dane między komputerem a jego otoczeniem zewnętrznym
  - połączenia systemu - mechanizmy zapewniające komunikację między składnikami systemu



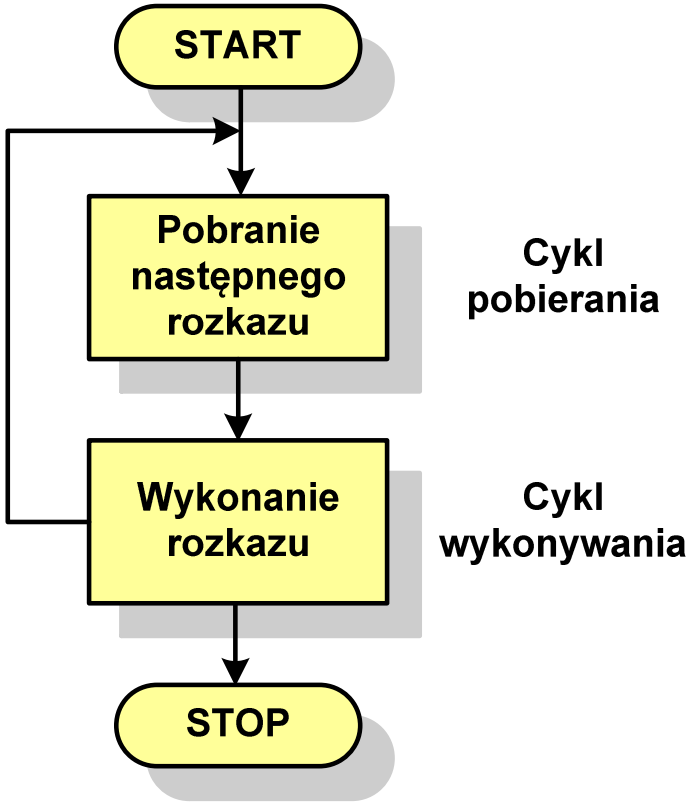
Ogólna struktura procesora

- Główne składniki strukturalne procesora to:
  - jednostka sterująca - steruje działaniem procesora i pośrednio całego komputera
  - jednostka arytmetyczno-logiczna (ALU) - realizuje przetwarzanie danych przez komputer
  - rejestry - realizują wewnętrzne przechowywanie danych w procesorze
  - połączenia procesora - wszystkie mechanizmy zapewniające komunikację między jednostką sterującą, ALU i rejestrami.



Działanie komputera

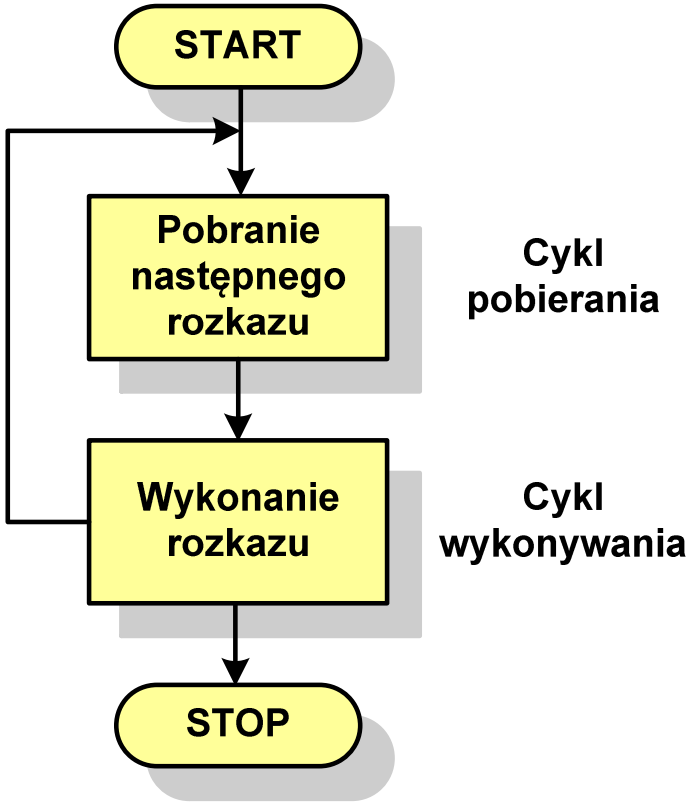
- Podstawowe zadanie komputera to wykonywanie programu
- Program składa się z rozkazów przechowywanych w pamięci
- Rozkazy są przetwarzane w dwu krokach:



- Cykl pobierania (ang. fetch):
  - odczytanie rozkazu z pamięci
  - licznik rozkazów (PC) lub wskaźnik instrukcji (IP) określa, który rozkaz ma być pobrany
  - jeśli procesor nie otrzyma innego polecenia, to inkrementuje licznik PC po każdym pobraniu rozkazu.

Działanie komputera

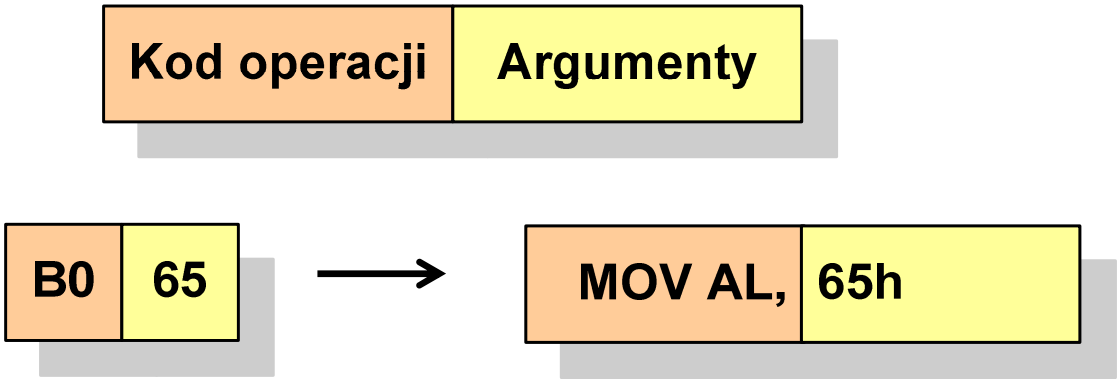
- Podstawowe zadanie komputera to wykonywanie programu
- Program składa się z rozkazów przechowywanych w pamięci
- Rozkazy są przetwarzane w dwu krokach:



- Cykl wykonywania (ang. execution):
  - pobrany rozkaz jest umieszczany w rejestrze rozkazu (IR)
  - rozkaz określa działania, które ma podjąć procesor
  - procesor interpretuje rozkaz i przeprowadza wymagane operacje.

Działanie komputera

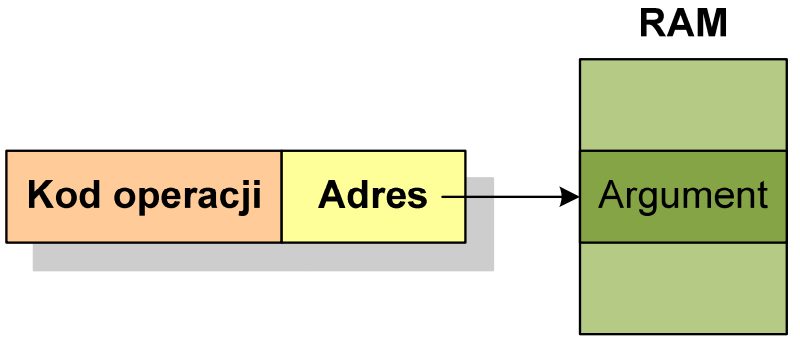
- Rozkaz:
  - przechowywany jest w postaci binarnej
  - ma określony format
  - używa określonego trybu adresowania
- Format - sposób rozmieszczenia informacji w kodzie rozkazu
- Rozkaz zawiera:
  - kod operacji (rodzaj wykonywanej operacji)
  - argumenty (lub adresy argumentów) wykonywanych operacji



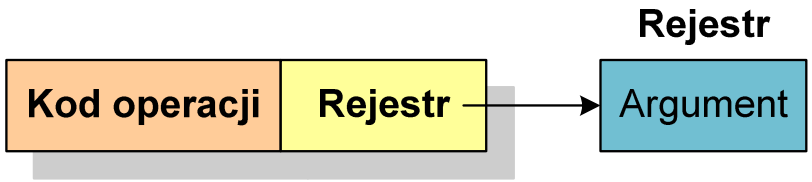
Działanie komputera

- Tryb adresowania - sposób określania miejsca przechowywania argumentów rozkazu (operandów)
- Przykładowe rodzaje adresowania:
  - natychmiastowe - argument znajduje się w kodzie rozkazu

  - bezpośrednie - kod rozkazu zawiera adres komórki pamięci, w której znajduje się argument



  - rejestrowe - kod rozkazu zawiera oznaczenie rejestru, w którym znajduje się argument



Program w asemblerze

```
.model SMALL
.286
.stack 100h
.code
   start:
      jmp begin

   handler:
      pusha
      push ds
      pop ds
      popa
      iret

   begin:
      mov ax,0000h
      mov ds,ax
      mov di,0070h
      lea ax,handler
```

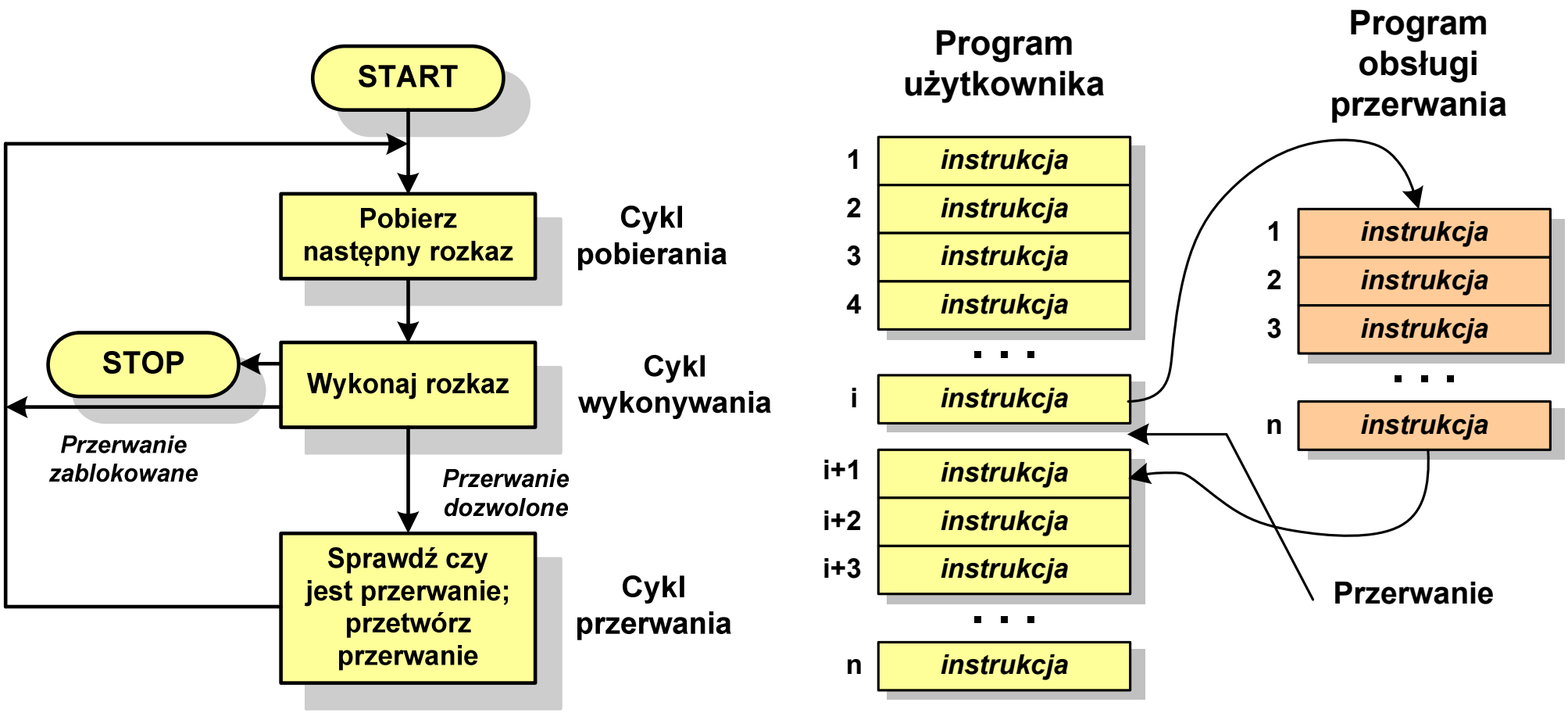
```
cli
mov [di],ax
mov [di+2],cs
sti
mov ax,3100h
mov dx,(offset begin - offset handler)
inc dx
int 21h
end
   start
```

Działanie komputera - przerwania

- Wykonywanie kolejnych rozkazów przez procesor może zostać przerwane poprzez wystąpienie tzw. przerwania (interrupt)
- Przerwanie jest to sygnał pochodzący od sprzętu lub oprogramowania informujący procesor o wystąpieniu jakiegoś zdarzenia (np. wciśnięcie klawisza na klawiaturze)
- Bez przerwań procesor musiałby ciągle kontrolować wszystkie urządzenia zewnętrzne, np. klawiatura, port szeregowy
- Każde przerwanie posiada procedurę obsługi przerwania, która jest wykonywana w momencie jego wystąpienia
- Adresy procedur obsługi przerwań zapisane są w tablicy wektorów przerwań

Działanie komputera - przerwania

- Implementacja przerwań wymaga dodania cyklu przerwania do cyklu rozkazu

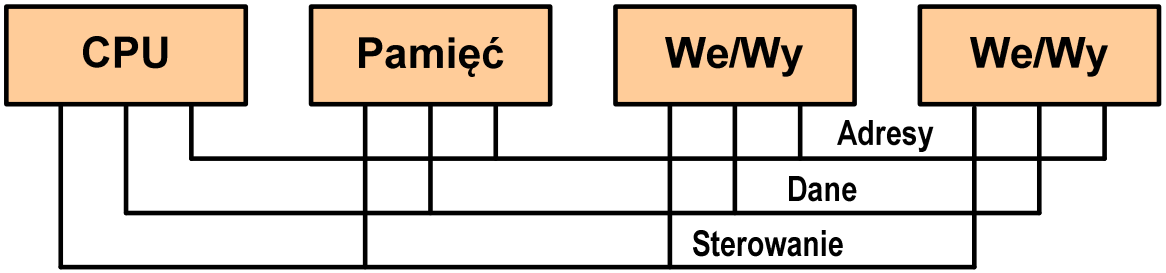


Rodzaje przerwań

- Sprzętowe
  - zewnętrzne - sygnały pochodzące z urządzeń zewnętrznych i służące do komunikacji z nimi, np. 08H - zegar, 09h - klawiatura
  - wewnętrzne - wywoływane przez procesor w celu zasygnalizowania sytuacji wyjątkowych (faults, traps, aborts)
- Programowe
  - instrukcje programu wywołują przerwanie - tym samym wykonywana jest procedura obsługi przerwania
  - służą głownie do komunikacji z systemem operacyjnym (DOS - 21h, Windows - 2h, Linux - 80h)

Magistrala

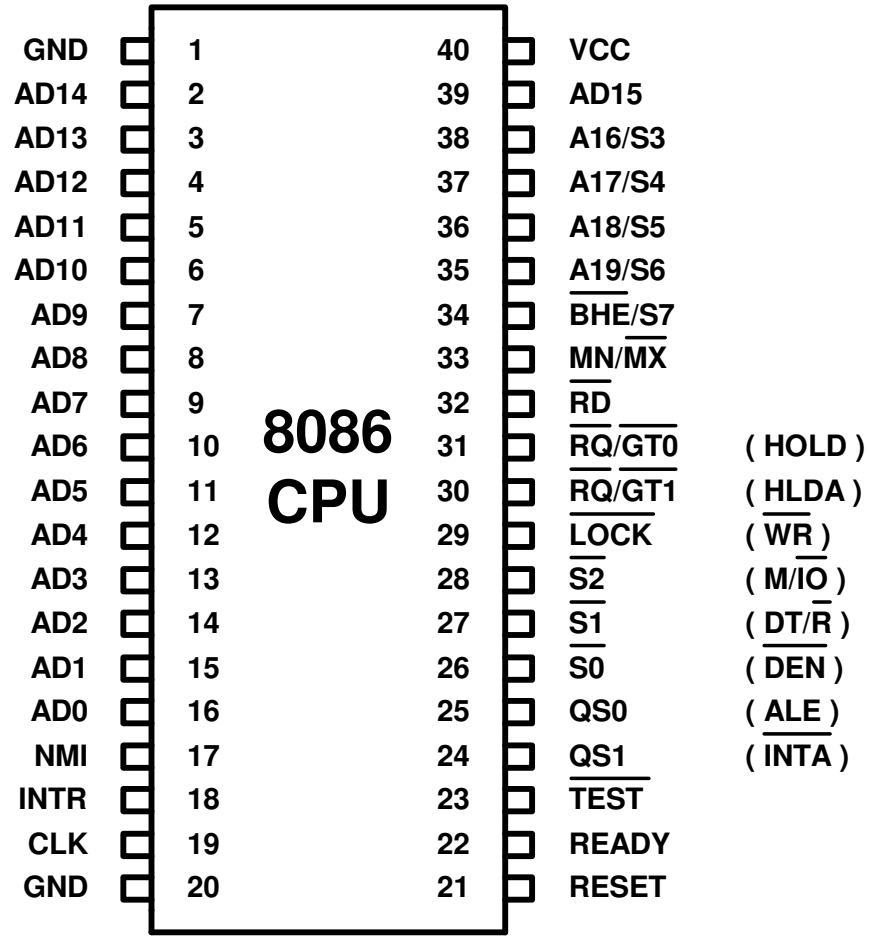
- Najczęściej stosowana struktura połączeń to magistrala, składająca się z wielu linii komunikacyjnych, którym przypisane jest określone znaczenie i określona funkcja



  - linie danych (szyna danych) - przenoszą dane między modułami systemu, liczba linii określa szerokość szyny danych (8, 16, 32, 64 bity)
  - linie adresowe - służą do określania źródła i miejsca przeznaczenia danych przesyłanych magistralą; liczba linii adresowych określa maksymalną możliwą pojemność pamięci systemu
  - linie sterowania - służą do sterowania dostępem do linii danych i linii adresowych

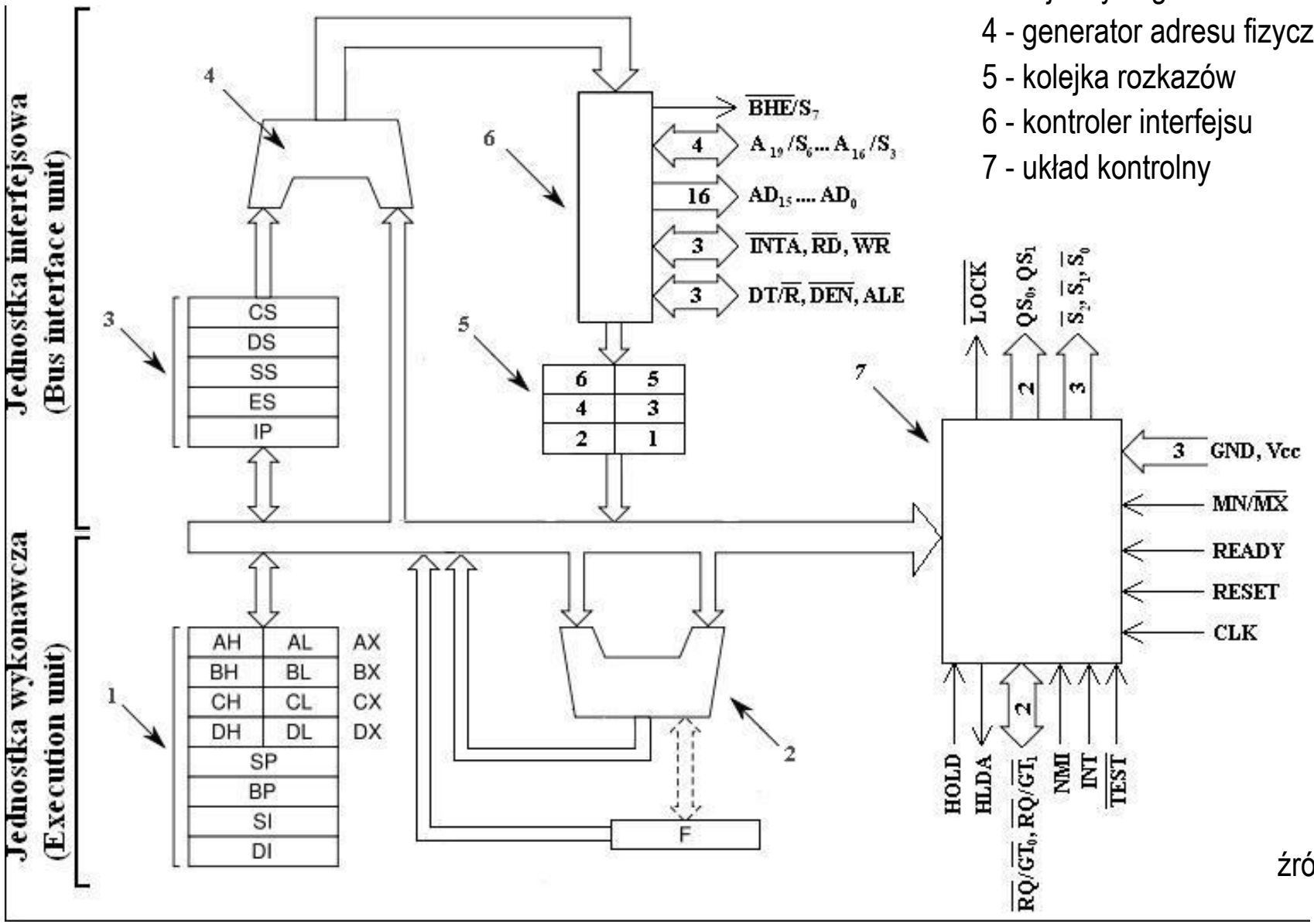
Intel 8086

- 1978 rok
- Procesor 16-bitowy
- 16-bitowa magistrala danych
- 20-bitowa magistrala adresowa
- Adresowanie do 1 MB pamięci
- Częstotliwość: 10 MHz
- Multipleksowane magistrale: danych i adresowa
- Litografia: 3 μm



Intel 8086

1 - rejestry ogólnego przeznaczenia
2 - ALU + rejestr znaczników (flag)
3 - rejestry segmentowe + licznik rozkazów
4 - generator adresu fizycznego
5 - kolejka rozkazów
6 - kontroler interfejsu
7 - układ kontrolny



źródło: wikipedia

Typy pamięci komputerowej

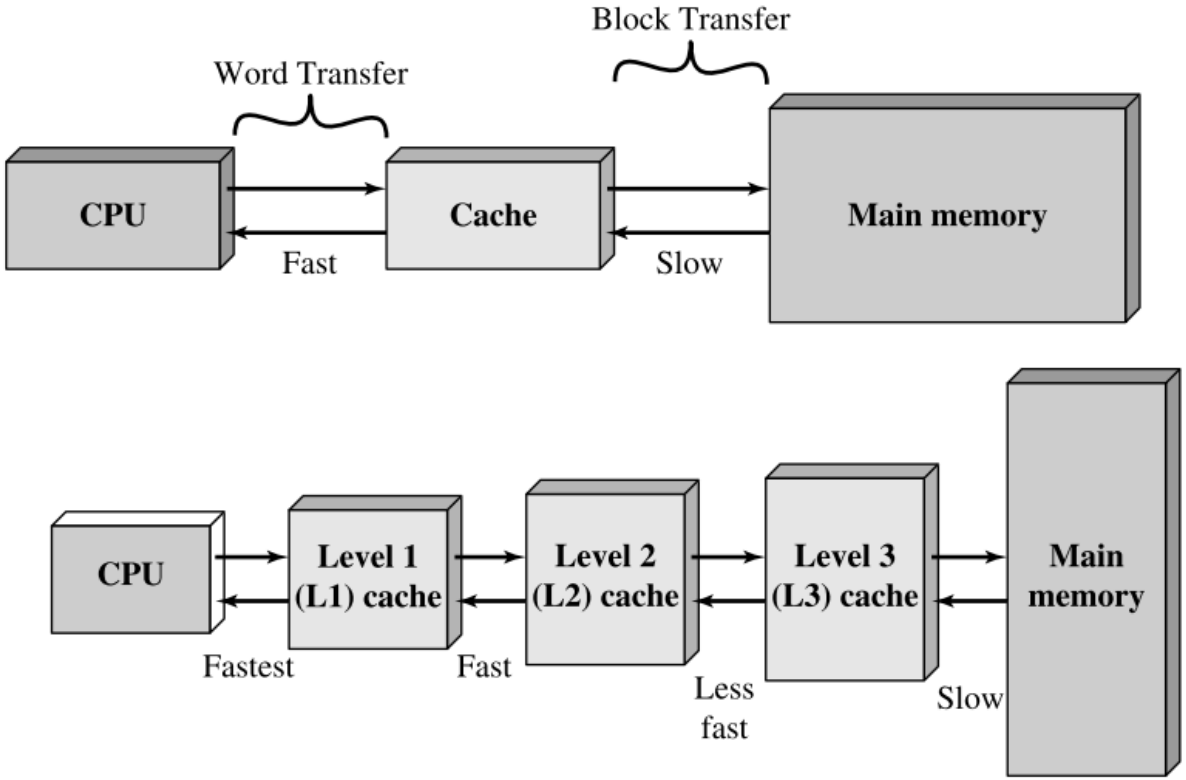
- RAM (Random Access Memory) - pamięć o dostępie swobodnym
  - odczyt i zapis następuje za pomocą sygnałów elektrycznych
  - pamięć ulotna - po odłączeniu zasilania dane są tracone
  - DRAM - pamięć dynamiczna:
    - przechowuje dane podobnie jak kondensator ładunek elektryczny
    - wymaga operacji odświeżania
    - jest mniejsza, gęściej upakowana i tańsza niż pamięć statyczna
    - stosowana jest do budowy głównej pamięci operacyjnej komputera
  - SRAM - pamięć statyczna:
    - przechowuje dane za pomocą przerzutnikowych konfiguracji bramek logicznych
    - nie wymaga operacji odświeżania
    - jest szybsza i droższa od pamięci dynamicznej
    - stosowana jest do budowy pamięci podręcznej

Typy pamięci komputerowej

- ROM (ang. Read-Only Memory) - pamięć stała
  - pamięć o dostępie swobodnym przeznaczona tylko do odczytu
  - dane są zapisywane podczas procesu wytwarzania, pamięć nieulotna
- PROM (ang. Programmable ROM) - programowalna pamięć ROM
  - pamięć nieulotna, może być zapisywana tylko jeden raz
  - zapis jest realizowany elektrycznie po wyprodukowaniu
- EPROM - pamięć wielokrotnie programowalna, kasowanie następuje przez naświetlanie promieniami UV
- EEPROM - pamięć kasowana i programowana na drodze elektrycznej
- Flash - rozwinięcie koncepcji pamięci EEPROM, możliwe kasowanie i programowanie bez wymontowywania pamięci z urządzenia

Pamięć podręczna (cache)

- Dodatkowa, szybka pamięć (SRAM) umieszczana pomiędzy procesorem a pamięcią główną
- Zastosowanie pamięci podręcznej ma na celu przyspieszenie dostępu procesora do pamięci głównej



źródło: W. Stallings, Computer Organization and Architecture

Przykład: suma kolejnych 10 liczb: 1+2+...+10

```
#include <stdio.h>

int main(void)
{
    int suma;

    suma = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10;

    printf("Suma wynosi: %d\n",suma);

    return 0;
}
```

```
Suma wynosi: 55
```

Przykład: suma kolejnych 100 liczb: 1+2+...+100

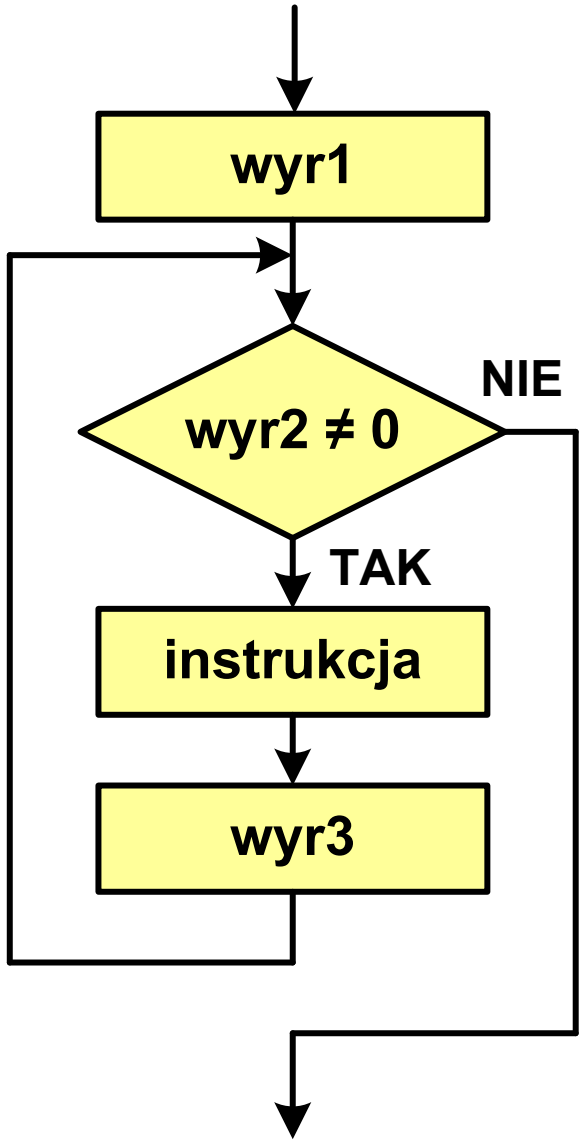
```
#include <stdio.h>

int main(void)
{
    int suma=0, i;

    for (i=1; i<=100; i=i+1)
        suma = suma + i;

    printf("Suma wynosi: %d\n",suma);

    return 0;
}
```

```
Suma wynosi: 5050
```

Język C - pętla for

```
for (wyr1; wyr2; wyr3)
    instrukcja;
```

- wyr1, wyr2, wyr3 - dowolne wyrażenia w języku C
- Instrukcja:
  - prosta - jedna instrukcja zakończona średnikiem
  - złożona - jedna lub kilka instrukcji objętych nawiasami klamrowymi



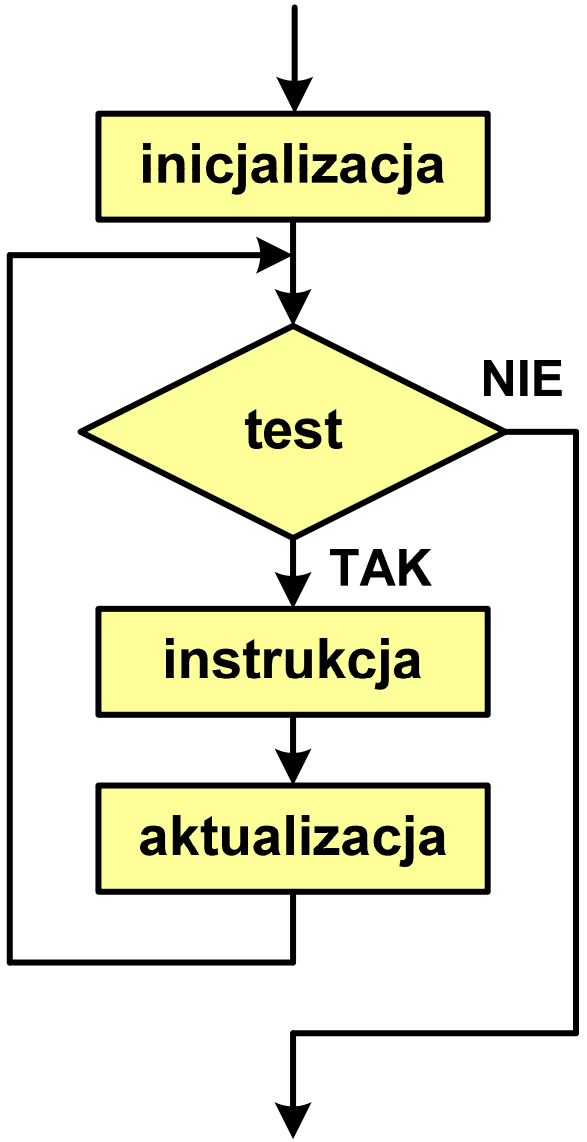
Język C - pętla for

- Najczęściej stosowana postać pętli for

```
int i;
for (i = 0; i < 10; i = i + 1)
    instrukcja;
```

- Instrukcja zostanie wykonana 10 razy (dla i = 0, 1, 2, … 9)
- Funkcje pełnione przez wyrażenia

```
for (inicjalizacja;test;aktualizacja)
    instrukcja;
```



Przykład: wyświetlenie tekstu 5 razy

```
#include <stdio.h>

int main(void)
{
    int i;

    for (i=0; i<5; i=i+1)
        printf("Programowanie nie jest trudne\n");

    return 0;
}
```

```
Programowanie nie jest trudne
Programowanie nie jest trudne
Programowanie nie jest trudne
Programowanie nie jest trudne
Programowanie nie jest trudne
```

Język C - pętla for (przykłady)

```
for (i=0; i<10; i++)
    printf("%d ",i);
```

```
0 1 2 3 4 5 6 7 8 9
```

```
for (i=0; i<10; i++)
    printf("%d ",i+1);
```

```
1 2 3 4 5 6 7 8 9 10
```

```
for (i=1; i<=10; i++)
    printf("%d ",i);
```

```
1 2 3 4 5 6 7 8 9 10
```

Język C - pętla for (przykłady)

```
for (i=1; i<10; i=i+2)
    printf("%d ",i);
```

```
1 3 5 7 9
```

```
for (i=10; i>0; i--)
    printf("%d ",i);
```

```
10 9 8 7 6 5 4 3 2 1
```

```
for (i=-9; i<=9; i=i+3)
    printf("%d ",i);
```

```
-9 -6 -3 0 3 6 9
```

Język C - pętla for (break, continue)

- W pętli for można stosować instrukcje skoku: break i continue

```
int i;

for (i=1; i<10; i++)
{
    if (i%2==0)
        continue;
    if (i%7==0)
        break;
    printf("%d\n",i);
}
```

```
1 3 5
```

- continue przerywa bieżącą iterację i przechodzi do obliczania wyr3
- break przerywa wykonywanie pętli

Język C - pętla for (najczęstsze błędy)

- Postawienie średnika na końcu pętli for

```
int i;
for (i=0; i<10; i++);
printf("%d ",i);
```

```
10
```

- Przecinki zamiast średników pomiędzy wyrażeniami

```
int i;
for (i=0, i<10, i++)
    printf("%d ",i);
```

Błąd kompilacji!

Język C - pętla for (najczęstsze błędy)

- Błędny warunek - brak wykonania instrukcji

```
int i;
for (i=0; i>10; i++)
    printf("%d ",i);
```

- Błędny warunek - pętla nieskończona

```
int i;
for (i=1; i>0; i++)
    printf("%d ",i);
```

```
1 2 3 4 5 6 7 8 9 ...
```

Język C - pętla nieskończona

```
for (wyr1; wyr2; wyr3)
    instrukcja;
```

- Wszystkie wyrażenia (wyr1, wyr2, wyr3) w pętli for są opcjonalne

```
for ( ; ; )
    instrukcja;
```

- pętla nieskończona

- W przypadku braku wyr2 przyjmuje się, że jest ono prawdziwe

Język C - zagnieżdżanie pętli for

- Jako instrukcja w pętli for może występować kolejna pętla for

```
int i, j;
for (i=1; i<=3; i++)          // pętla zewnętrzna
    for (j=1; j<=2; j++)      // pętla wewnętrzna
        printf("i: %d   j: %d\n",i,j);
```

```
i: 1   j: 1
i: 1   j: 2
i: 2   j: 1
i: 2   j: 2
i: 3   j: 1
i: 3   j: 2
```

Język C - operator inkrementacji (++)

- Jednoargumentowy operator ++ zwiększa wartość zmiennej o 1 (nie wolno stosować go do wyrażeń)
- Operator ++ może występować jako przedrostek lub przyrostek

Zapis	Nazwa	Znaczenie
`++x`	preinkrementacji	wartość zmiennej jest modyfikowana przed jej użyciem
`x++`	postinkrementacji	wartość zmiennej jest modyfikowana po użyciu jej poprzedniej wartości

Język C - operator inkrementacji (++)

- Przykład

```
int x = 1, y;
y = 2 * ++x;
```

```
int x = 1, y;
y = 2 * x++;
```

- Kolejność operacji

```
++x                 x = 2
2 * ++x             2 * 2
y = 2 * ++x         y = 4
```

```
2 * x               2 * 1
y = 2 * x           y = 2
x++                 x = 2
```

- Wartości zmiennych

```
x = 2     y = 4
```

```
x = 2     y = 2
```

Język C - operator inkrementacji (++)

- Miejsce umieszczenia operatora ++ nie ma znaczenia w przypadku instrukcji typu:

```
x++;
++x;
```

równoważne

```
x = x + 1;
```

- Nie należy stosować operatora ++ do zmiennych pojawiających się w wyrażeniu więcej niż jeden raz

```
x = x++;
x = ++x;
```

- Zgodnie ze standardem języka C wynik powyższych instrukcji jest niezdefiniowany

Język C - operator dekrementacji (--)

- Jednoargumentowy operator -- zmniejsza wartość zmiennej o 1 (nie wolno stosować go do wyrażeń)
- Operator -- może występować jako przedrostek lub przyrostek

Zapis	Nazwa	Znaczenie
`--x`	predekrementacji	wartość zmiennej jest modyfikowana przed jej użyciem
`x--`	postdekrementacji	wartość zmiennej jest modyfikowana po użyciu jej poprzedniej wartości

Język C - priorytet operatorów ++ i --

Priorytet	Operator / opis
1	`++` `--` (przyrostki) `()` `[]` `.` `->`
2	`++` `--` (przedrostki) `sizeof` `(typ)` `+` `-` `!` `~` `*` `&` (jednoargumentowe)
3	`*` `/` `%`
4	`+` `-` (dwuargumentowe)
5	`<<` `>>`
6	`<` `>` `<=` `>=`
7	`==` `!=`
8	`&` (bitowy)
9	`^`

Przykład: pierwiastek kwadratowy

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    float x, y;

    printf("Podaj liczbe: ");
    scanf("%f",&x);

    if (x>=0)
    {
        y = sqrt(x);
        printf("Pierwiastek liczby: %f\n",y);
    }
    else
        printf("Blad! Liczba ujemna\n");

    return 0;
}
```

```
Podaj liczbe: -3
Blad! Liczba ujemna
```

```
Podaj liczbe: 3
Pierwiastek liczby: 1.732051
```

Przykład: pierwiastek kwadratowy (pętla while)

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    float x, y;

    printf("Podaj liczbe: ");
    scanf("%f",&x);
    while (x<0)
    {
        printf("Blad! Liczba ujemna\n\n");
        printf("Podaj liczbe: ");
        scanf("%f",&x);
    }
    y = sqrt(x);
    printf("Pierwiastek liczby: %f\n",y);

    return 0;
}
```

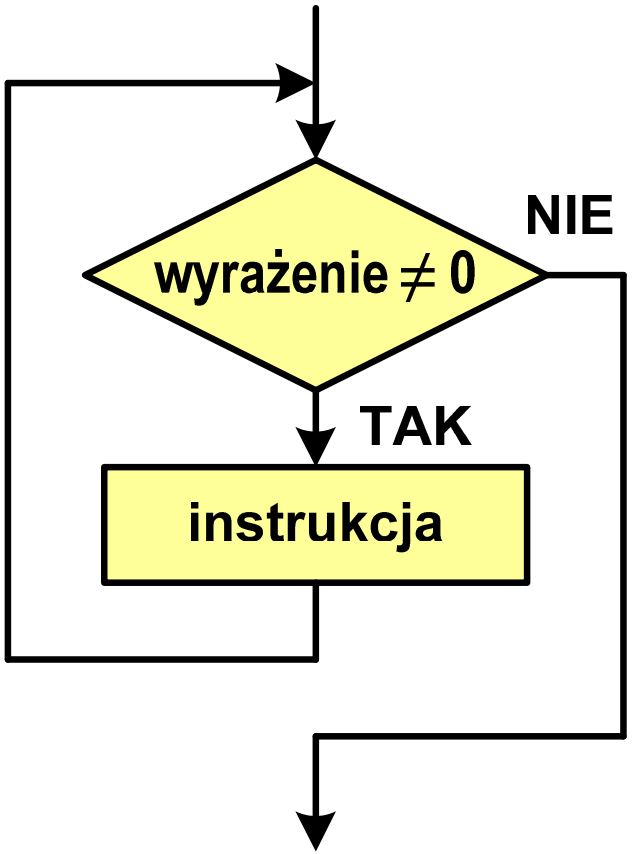
```
Podaj liczbe: -3
Blad! Liczba ujemna

Podaj liczbe: -5
Blad! Liczba ujemna

Podaj liczbe: 3
Pierwiastek liczby: 1.732051
```

Język C - pętla while

```
while (wyrażenie)
    instrukcja;
```

- „dopóki wyrażenie w nawiasach jest prawdziwe wykonuj instrukcję”

- Wyrażenie w nawiasach:
  - prawdziwe - gdy jego wartość jest różna od zera
  - fałszywe - gdy jego wartość jest równa zero
- Jako wyrażenie najczęściej stosowane jest wyrażenie logiczne



Język C - pętla while

```
while (wyrażenie)
    instrukcja;
```

- Instrukcja:
  - prosta - jedna instrukcja zakończona średnikiem
  - złożona - jedna lub kilka instrukcji objętych nawiasami klamrowymi

```
int x = 10;
while (x>0)
    x = x - 1;
```

```
int x = 10;
while (x>0)
{
    printf("%d\n",x);
    x = x - 1;
}
```

Przykład: suma liczb dodatnich

```c
#include <stdio.h>

int main(void)
{
    int x, suma = 0;

    printf("Podaj liczbe: ");
    scanf("%d",&x);

    while(x>0)
    {
        suma = suma + x;
        printf("Podaj liczbe: ");
        scanf("%d",&x);
    }
    printf("Suma liczb: %d\n",suma);

    return 0;
}
```

```
Podaj liczbe: 4
Podaj liczbe: 8
Podaj liczbe: 2
Podaj liczbe: 3
Podaj liczbe: 5
Podaj liczbe: -2
Suma liczb: 22
```

Język C - pętla while

- Program pokazany na poprzednim slajdzie zawiera typowy schemat przetwarzania danych z wykorzystaniem pętli while

```
printf("Podaj liczbe: ");
scanf("%d",&x);

while(x>0)
{
    suma = suma + x;

    printf("Podaj liczbe: ");
    scanf("%d",&x);
}
```

wczytanie danych

operacje na danych

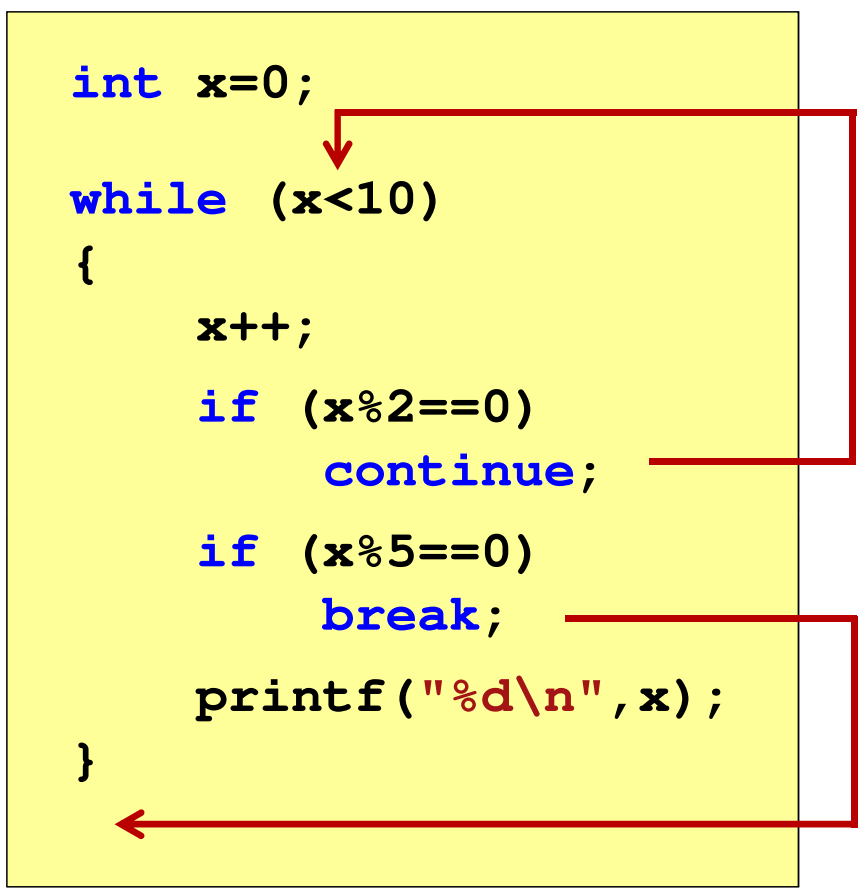
wczytanie danych

- Dane mogą być wczytywane z klawiatury, pliku, itp.

Język C - pętla while (break, continue)

- break i continue są to instrukcje skoku

```
int x=0;

while (x<10)
{
    x++;
    if (x%2==0)
        continue;
    if (x%5==0)
        break;
    printf("%d\n",x);
}
```



- continue przerywa bieżącą iterację
- break przerywa wykonywanie pętli

Język C - pętla while (najczęstsze błędy)

- Postawienie średnika po wyrażeniu w nawiasach powoduje powstanie pętli nieskończonej - program zatrzymuje się na pętli

```
int x = 10;
while (x>0);
    printf("%d ",x--);
```

- Brak aktualizacji zmiennej powoduje także powstanie pętli nieskończonej - program wyświetla wielokrotnie tę samą wartość

```
int x = 10;
while (x>0)
    printf("%d ",x);
```

```
10 10 10 10 10 ...
```

Język C - pętla while (pętla nieskończona)

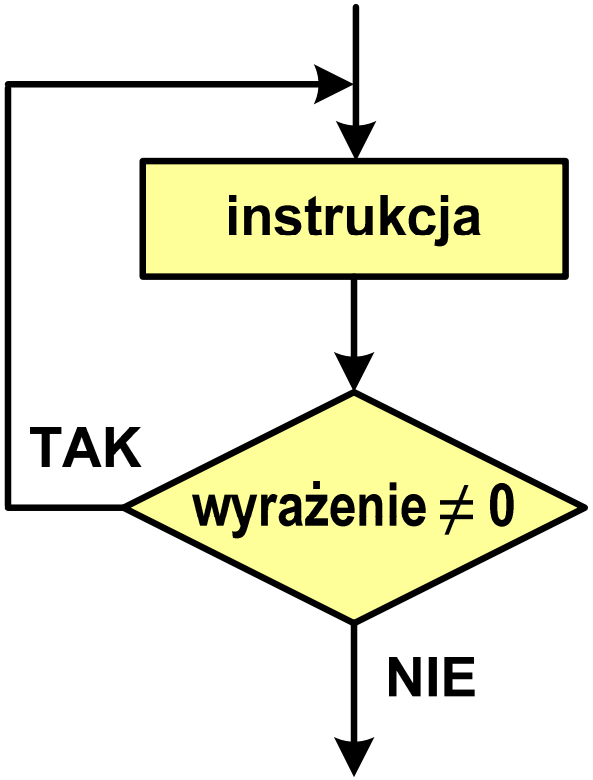
- W pewnych sytuacjach celowo stosuje się pętlę nieskończoną (np. w mikrokontrolerach)

```
while (1)
{
    instrukcja;
    instrukcja;
    ...
}
```

Język C - pętla do ... while

```
do
    instrukcja;
while (wyrażenie);
```

- „wykonuj instrukcję dopóki wyrażenie w nawiasach jest prawdziwe”



- Wyrażenie w nawiasach:
  - prawdziwe - gdy jego wartość jest różna od zera
  - fałszywe - gdy jego wartość jest równa zero

Język C - pętla do ... while

```
do
    instrukcja;
while (wyrażenie);
```

- Instrukcja:
  - prosta - jedna instrukcja zakończona średnikiem
  - złożona - jedna lub kilka instrukcji objętych nawiasami klamrowymi

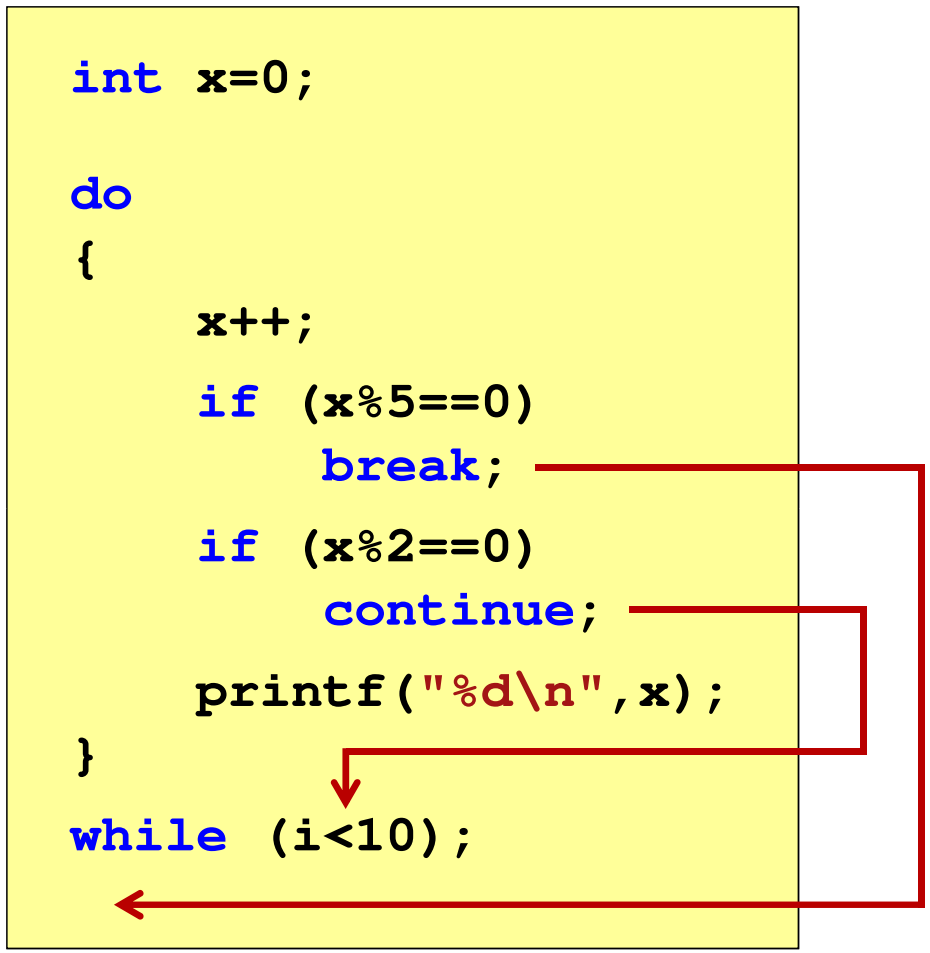
```
int x = 10;
do
    x = x - 1;
while (x>0);
```

```
int x = 10;
do
{
    printf("%d\n",x);
    x = x - 1;
}
while (x>0);
```

Język C - pętla do ... while (break, continue)

- break i continue są to instrukcje skoku

```
int x=0;

do
{
    x++;
    if (x%5==0)
        break;
    if (x%2==0)
        continue;
    printf("%d\n",x);
}
while (i<10);
```



- break przerywa wykonywanie pętli
- continue przerywa bieżącą iterację

Przykład: pierwiastek kwadratowy (pętla do...while)

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    float x, y;

    do
    {
        printf("Podaj liczbe: ");
        scanf("%f",&x);
    }
    while (x<0);

    y = sqrt(x);
    printf("Pierwiastek liczby: %f\n",y);

    return 0;
}
```

```
Podaj liczbe: -3
Podaj liczbe: -5
Podaj liczbe: 3

Pierwiastek liczby: 1.732051
```

Koniec wykładu nr 5

Dziękuję za uwagę!