

Informatyka 1 (ES1F1002)

Politechnika Białostocka - Wydział Elektryczny
Elektrotechnika, semestr II, studia stacjonarne I stopnia
Rok akademicki 2022/2023

Wykład nr 7 (21.11.2022)

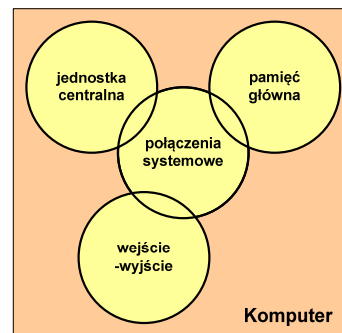
dr inż. Jarosław Forenc

Plan wykładu nr 7

- Struktura i funkcjonowanie komputera
 - procesor, rozkazy, przerwania, magistrala
 - pamięć komputerowa, hierarchia pamięci
 - pamięć podręczna
- Algorytmy komputerowe
 - definicje, podstawowe cechy, sposoby opisu
 - rekurencja, złożoność obliczeniowa
 - algorytmy sortowania (proste wstawianie, proste wybieranie)

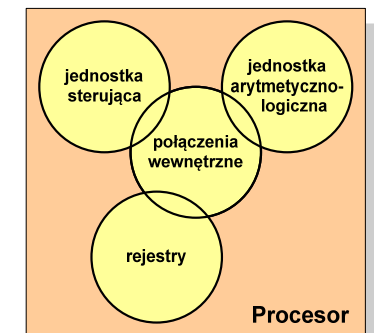
Ogólna struktura systemu komputerowego

- 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



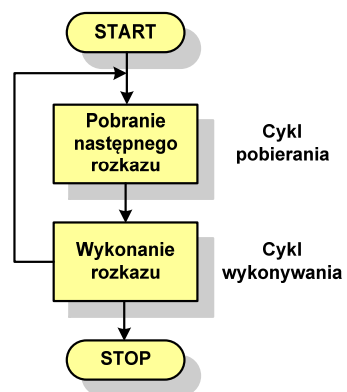
Ogólna struktura procesora

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



Działanie komputera

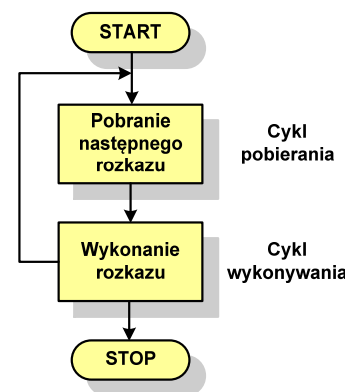
- Podstawowe zadanie komputera to wykonywanie **programu**
- Program składa się z **rozkazów** przechowywanych w pamięci
- Rozkazy są przetwarzane w dwu krokach:



- **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

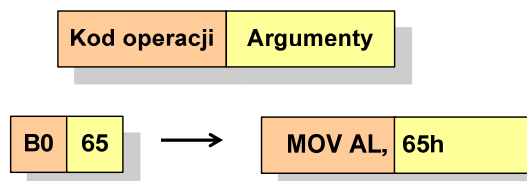
- Podstawowe zadanie komputera to wykonywanie **programu**
- Program składa się z **rozkazów** przechowywanych w pamięci
- Rozkazy są przetwarzane w dwu krokach:



- **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.

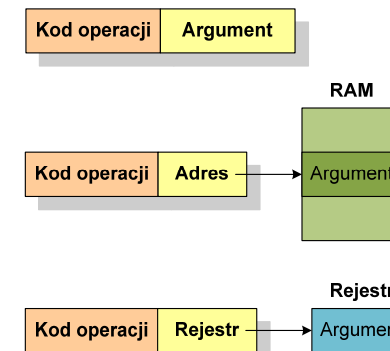
Działanie komputera

- 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
```

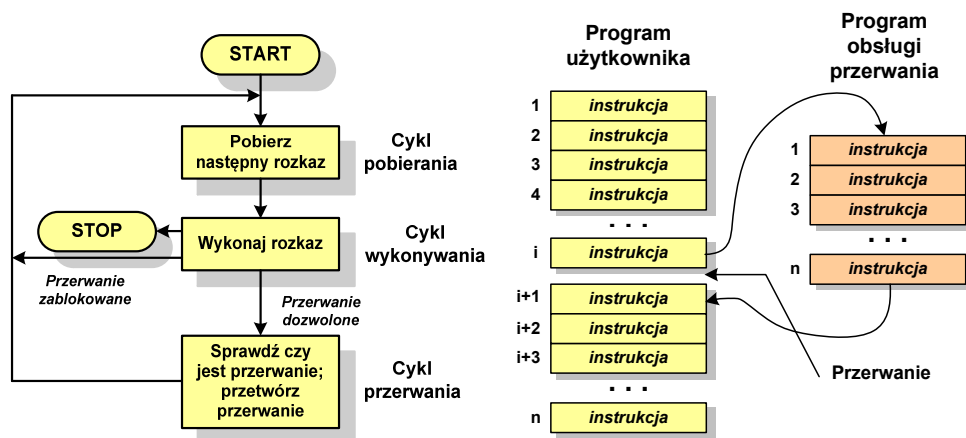
```
cli
mov [di], ax
mov [di+2], cs
sti
mov ax, 3100h
mov dx, (offset begin - offset handler)
inc dx
int 21h
end
start
```

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 przerwania 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 przerwania zapisane są w tablicy wektorów przerwania

Działanie komputera - przerwania

- Implementacja przerwania wymaga dodania cyklu przerwania do cyklu rozkazu

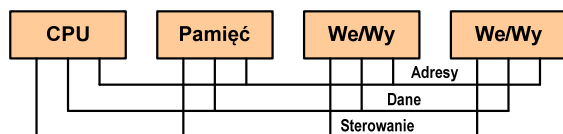


Rodzaje przerwania

- **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 głównie do komunikacji z systemem operacyjnym (DOS - 21h, Windows - 2h, Linux - 80h)

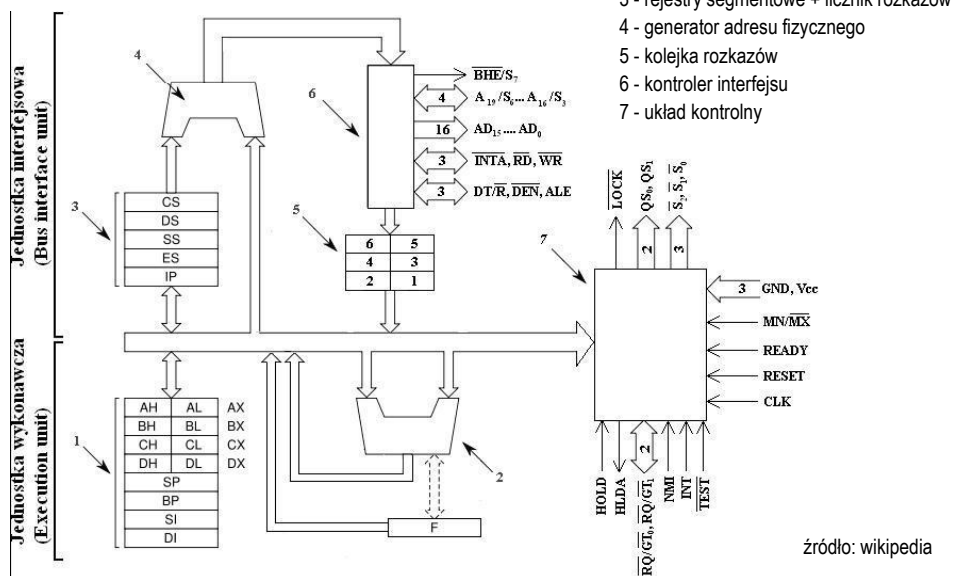
Magistrala

- 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

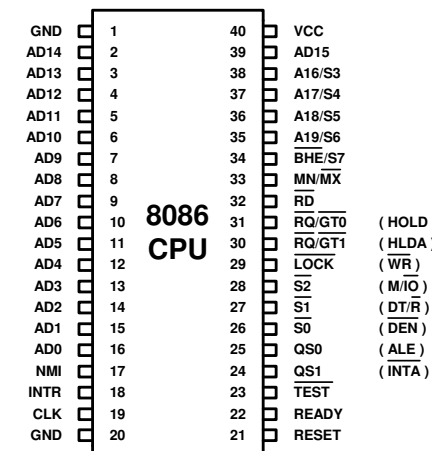
Intel 8086



- rejstry ogólnego przeznaczenia
- ALU + rejestr znaczników (flag)
- rejstry segmentowe + licznik rozkazów
- generator adresu fizycznego
- kolejka rozkazów
- kontroler interfejsu
- układ kontrolny

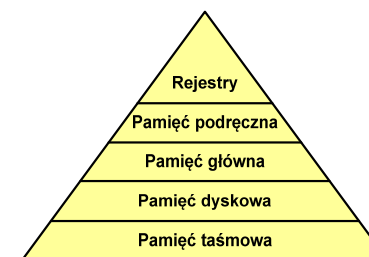
Intel 8086

- 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 μ m



Systemy pamięci komputerowych

- W systemach komputerowych nie stosuje się jednego typu pamięci, ale **hierarchię pamięci**



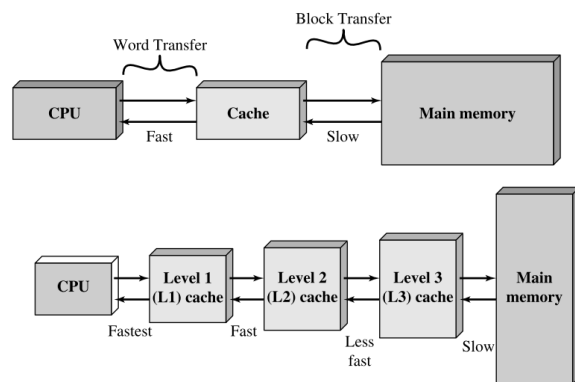
- Rozpatrując hierarchię od góry do dołu obserwujemy zjawiska:
 - malejący koszt na bit
 - rosnąca pojemność
 - rosnący czas dostępu
 - malejącą częstotliwość dostępu do pamięci przez procesor

Półprzewodnikowa pamięć główna

- **RAM** (Random Access Memory) - pamięć o dostępie swobodnym
 - odczyt i zapis następuje za pomocą sygnałów elektrycznych
 - pamięć ulotna - po odłączeniu zasilania dane są traczone
 - **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

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



Półprzewodnikowa pamięć główna

- **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

Algorytm - definicje

Definicja 1

- Skończony, uporządkowany ciąg jasno zdefiniowanych czynności, koniecznych do wykonania pewnego zadania

Definicja 2

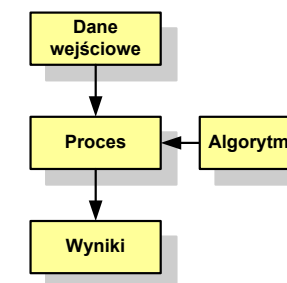
- Opis rozwiązania problemu wyrażony za pomocą operacji zrozumiałych i możliwych do zrealizowania przez wykonawcę

Definicja 3

- Ścisłe określona procedura obliczeniowa, która dla właściwych danych wejściowych zwraca żądane dane wyjściowe zwane wynikiem działania algorytmu

Definicja 4

- Metoda rozwiązania zadania



Algorytmy

- Słowo „algorytm” pochodzi od nazwiska matematyka perskiego z IX wieku - Muhammada ibn-Musy **al-Chuwarizmiego** (po łacinie pisanego jako **Algorismus**)
- Badaniem algorytmów zajmuje się **algorytmika**
- „Przetłumaczenie” algorytmu na wybrany język programowania:
 - **implementacja** algorytmu
 - **kodowanie** algorytmu
- Sposoby opisu algorytmów
 - opis słowny w języku naturalnym lub lista kroków (opis w punktach)
 - schemat blokowy
 - pseudokod (nieformalna odmiana języka programowania)
 - wybrany język programowania

Lista kroków

- Uporządkowany opis wszystkich czynności, jakie należy wykonać podczas realizacji algorytmu
- **Krok** jest to pojedyncza czynność realizowana w algorytmie
- Kroki w algorytmie są numerowane, operacje wykonywane są zgodnie z rosnącą numeracją kroków
- Jedynym odstępstwem od powyższej reguły są operacje skoku (warunkowe lub bezwarunkowe), w których jawnie określa się numer kolejnego kroku
- **Przykład** (instrukcja otwierania wózka-specerówki):
 - Krok 1:** Zwolnij element blokujący wózek
 - Krok 2:** Rozkładaj wózek w kierunku kółek
 - Krok 3:** Naciskając nogą dolny element blokujący aż do zatrzaśnięcia, rozłóż wózek do pozycji przewozowej

Opis słowny algorytmu

- Podanie kolejnych czynności, które należy wykonać, aby otrzymać oczekiwany efekt końcowy
- Przypomina przepis kulinarny z książki kucharskiej lub instrukcję obsługi urządzenia, np.

Algorytm: Tortilla („Podróże kulinarne” R. Makłowicza)

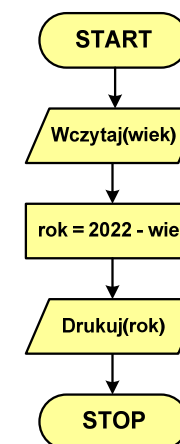
Dane wejściowe: 0,5 kg ziemniaków, 100 g kiełbasy Chorizo, 8 jajek

Dane wyjściowe: gotowa Tortilla

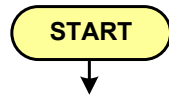
Opis algorytmu: Ziemniaki obrać i pokroić w plasterki. Kiełbasę pokroić w plasterki. Ziemniaki wrzucić na gorącą oliwę na patelni i przyrumienić z obu stron. Kiełbasę wrzucić na gorącą oliwę na patelni i przyrumienić z obu stron. Ubić jajka i dodać do połączonych ziemniaków i kiełbasy. Dodać sól i pieprz. Usmażyć z obu stron wielki omlet nadziewany chipsami ziemniaczanymi z kiełbaską.

Schemat blokowy

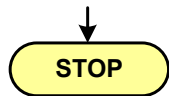
- Zawiera plan algorytmu przedstawiony w postaci graficznej
- Na schemacie umieszczane są **bloki** oraz **linie przepływu** (strzałki)
- Blok zawiera informację o wykonywanej operacji
- Linie przepływu (strzałki) określają kolejność wykonywania bloków algorytmu
- Przykład: wyznaczenie roku urodzenia na podstawie wieku (**algorytm liniowy**)



Schemat blokowy - symbole graficzne

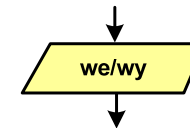


- **blok startowy**, początek algorytmu
- wskazuje miejsce rozpoczęcia algorytmu
- ma jedno wyjście
- może występować tylko jeden raz

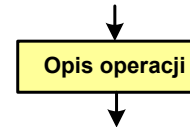


- **blok końcowy**, koniec algorytmu
- wskazuje miejsce zakończenia algorytmu
- ma jedno wejście
- musi występować przynajmniej jeden raz

Schemat blokowy - symbole graficzne



- **blok wejścia-wyjścia**
- poprzez ten blok wprowadzane są (czytane) dane wejściowe i wyprowadzane (zapisywane) wyniki
- ma jedno wejście i jedno wyjście

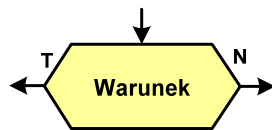


- **blok wykonawczy**, blok funkcyjny, opis procesu
- zawiera jedno lub kilka poleceń (elementarnych instrukcji) wykonywanych w podanej kolejności
- instrukcją może być np. operacja arytmetyczna, podstawienie
- ma jedno wejście i jedno wyjście

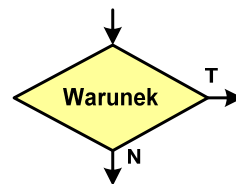
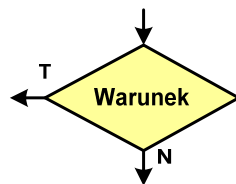
Schemat blokowy - symbole graficzne



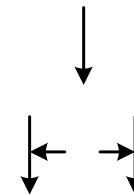
- **blok warunkowy** (decyzyjny, porównujący)
- wewnątrz bloku umieszcza się warunek logiczny
- na podstawie warunku określana jest tylko jedna droga wyjściowa



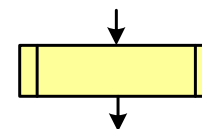
- połączenia wychodzące z bloku:
 - **T** lub **TAK** - gdy warunek jest prawdziwy
 - **N** lub **NIE** - gdy warunek nie jest prawdziwy
- wyjścia mogą być skierowane na boki lub w dół



Schemat blokowy - symbole graficzne



- **linia przepływu**, połączenie, linia
- występuje w postaci linii zakończonej strzałką
- określa kierunek przemieszczania się po schemacie
- linie pochodzące z różnych części algorytmu mogą zbiegać się w jednym miejscu



- **podprogram**
- wywołanie wcześniej zdefiniowanego fragmentu algorytmu (podprogramu)
- ma jedno wejście i jedno wyjście



- **komentarz**
- dodanie do schematu dodatkowego opisu

Pseudokod i język programowania

Pseudokod:

- Pseudokod (pseudojęzyk) - uproszczona wersja języka programowania
- Często zawiera zwroty pochodzące z języków programowania
- Zapis w pseudokodzie może być łatwo przetłumaczony na wybrany język programowania

Opis w języku programowania:

- Zapis programu w konkretnym języku programowania
- Stosowane języki: Pascal, C, C++, Matlab, Python (kiedyś - Fortran, Basic)

Algorytm Euklidesa - lista kroków

Dane wejściowe: niezerowe liczby naturalne a i b

Dane wyjściowe: $NWD(a,b)$

Kolejne kroki:

1. Czytaj liczby a i b
2. Dopóki a i b są większe od zera, powtarzaj **krok 3**, a w przeciwnym przypadku przejdź do **kroku 4**
3. Jeśli a jest większe od b , to weź za a resztę z dzielenia a przez b , w przeciwnym przypadku weź za b resztę z dzielenia b przez a
4. Przyjmij jako największy wspólny dzielnik tę z liczb a i b , która pozostała większa od zera
5. Drukuj $NWD(a,b)$

Największy wspólny dzielnik - algorytm Euklidesa

- NWD - największa liczba naturalna dzieląca (bez reszty) dwie (lub więcej) liczby całkowite

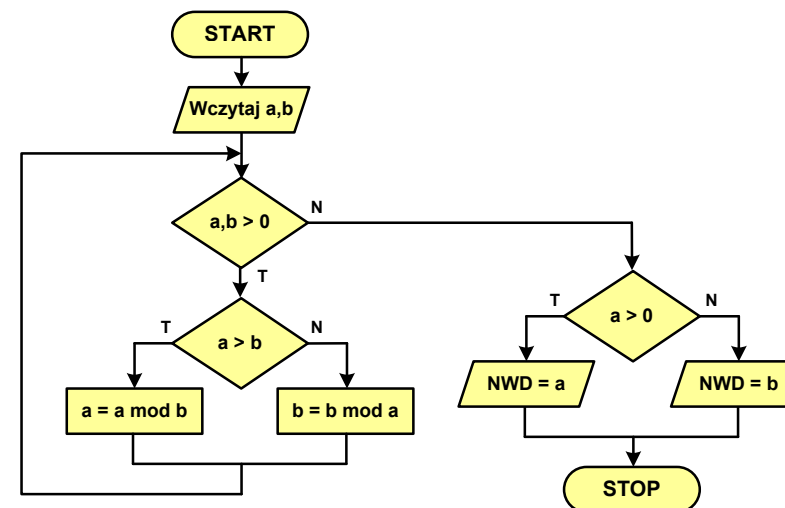
$$NWD(1675,3752) = ?$$

Algorytm Euklidesa - przykład

a	b	Dzielenie większej liczby przez mniejszą	Zamiana
1675	3752	$b/a = 3752/1675 = 2$ reszta 402	$b = 402$
1675	402	$a/b = 1675/402 = 4$ reszta 67	$a = 67$
67	402	$b/a = 402/67 = 6$ reszta 0	$b = 0$
67	0	KONIEC	

$$NWD(1675,3752) = 67$$

Algorytm Euklidesa - schemat blokowy



Algorytm Euklidesa - pseudokod

```

NWD(a,b)
while a>0 i b>0
do if a>b
    then a ← a mod b
    else b ← b mod a
if a>0
then return a
else return b
    
```

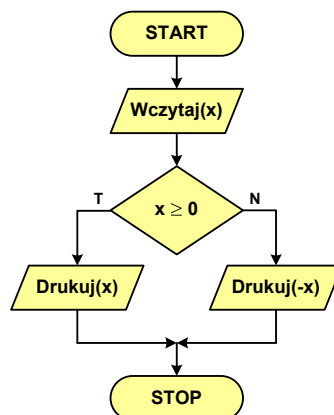
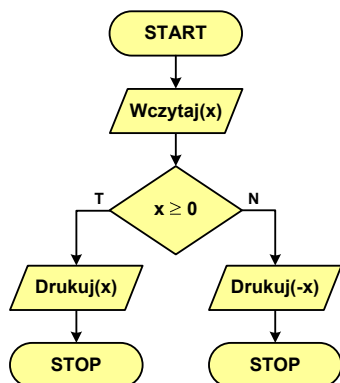
Algorytm Euklidesa - język programowania (C)

```

#include <stdio.h>
int main(void)
{
    int a = 1675, b = 3752, NWD;
    while (a>0 && b>0)
        if (a>b)
            a = a % b;
        else
            b = b % a;
    if (a>0)
        NWD = a;
    else
        NWD = b;
    printf("NWD = %d\n", NWD);
}
    
```

Wartość bezwzględna liczby - schemat blokowy

$$|x| = \begin{cases} x & \text{dla } x \geq 0 \\ -x & \text{dla } x < 0 \end{cases}$$



Równanie kwadratowe - schemat blokowy

$$ax^2 + bx + c = 0$$

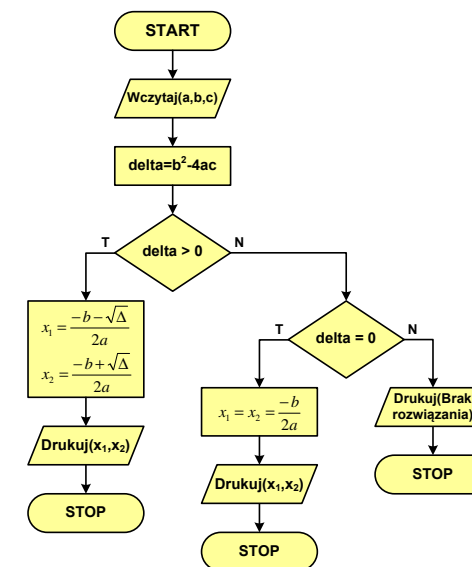
$$\Delta = b^2 - 4ac$$

$$\Delta > 0:$$

$$x_1 = \frac{-b - \sqrt{\Delta}}{2a}, \quad x_2 = \frac{-b + \sqrt{\Delta}}{2a}$$

$$\Delta = 0:$$

$$x_1 = x_2 = \frac{-b}{2a}$$



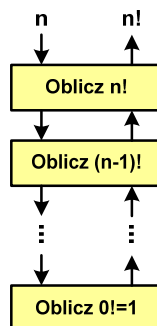
Rekurencja

- **Rekurencja** lub **rekursja** - jest to odwoływanie się funkcji lub definicji do samej siebie
- Rozwiązanie danego problemu wyraża się za pomocą rozwiązań tego samego problemu, ale dla danych o mniejszych rozmiarach
- W matematyce mechanizm rekurencji stosowany jest do definiowania lub opisywania algorytmów

- Silnia:

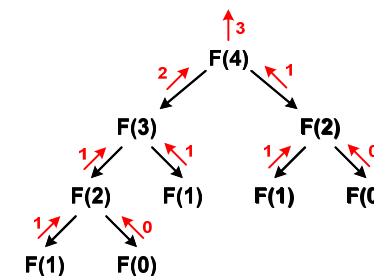
$$n! = \begin{cases} 1 & \text{dla } n = 0 \\ n(n-1)! & \text{dla } n \geq 1 \end{cases}$$

```
int silnia(int n)
{
    return n==0 ? 1 : n*silnia(n-1);
}
```



Rekurencja - ciąg Fibonacciego

$$F_n = \begin{cases} 0 & \text{dla } n = 0 \\ 1 & \text{dla } n = 1 \\ F_{n-1} + F_{n-2} & \text{dla } n > 1 \end{cases}$$



```
int F(int n)
{
    if (n==0) return 0;
    if (n==1) return 1;
    return F(n-1) + F(n-2);
}
```

Złożoność obliczeniowa

- W celu rozwiązania danego problemu obliczeniowego szukamy algorytmu najbardziej **efektywnego** czyli:
 - najszybszego (najkrótszy czas otrzymania wyniku)
 - o możliwie małym zapotrzebowaniu na pamięć
- **Problem:** Jak ocenić, który z dwóch różnych algorytmów rozwiązujących to samo zadanie jest efektywniejszy?
- Do oceny efektywności służy **złożoność obliczeniowa algorytmu** (**koszt algorytmu**) czyli ilość zasobów potrzebnych do jego działania (czas, pamięć)
- Miarą złożoności **czasowej** jest liczba podstawowych (dominujących) operacji (porównanie, podstawienie, operacja arytmetyczna) - pozostałe operacje są pomijane
- Miarą złożoności **pamięciowej** jest liczba wykorzystanych komórek pamięci (bajty lub liczba zmiennych określonego typu)

Złożoność obliczeniowa

- Złożoność obliczeniowa algorytmu jest **funkcją** opisującą zależność między **liczbą danych** a **liczbą operacji** wykonywanych przez ten algorytm
- W praktyce stosuje się oszacowanie powyższej funkcji - są to tzw. notacje (klasy złożoności):
 - O (duże O) - najbardziej popularna
 - Ω (omega)
 - Θ (theta)

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.

$$f(n) = n^2 + 2n \rightarrow O(n^2) \quad f(n) = n^2 + n - 5 \rightarrow O(n^2)$$
- W powyższych przykładach dla dużego n wpływ składnika liniowego i stałego na wartość funkcji jest nieistotny w porównaniu ze składnikiem głównym 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 \leq lub \geq

Przykład:

- Tablica nieposortowana:

6	4	5	2	3	1
---	---	---	---	---	---
- Tablica posortowana zgodnie z relacją \leq (od najmniejszej do największej liczby):

1	2	3	4	5	6
---	---	---	---	---	---
- Tablica posortowana zgodnie z relacją \geq (od największej do najmniejszej liczby):

6	5	4	3	2	1
---	---	---	---	---	---

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):

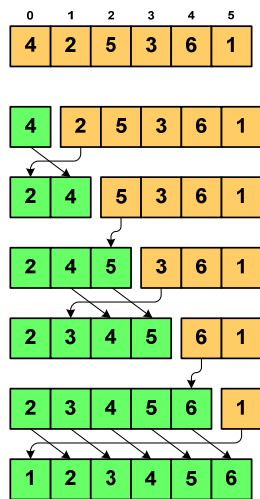
Jan	Piotr	Ela	Magda	Ola	Paweł
Kamiński	Kowalski	Król	Mazur	Nowak	Wójcik
20	25	22	15	18	23

- Tablica posortowana (klucz - wiek):

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

Proste wstawianie (insertion sort)

Przykład:



Program w języku C:

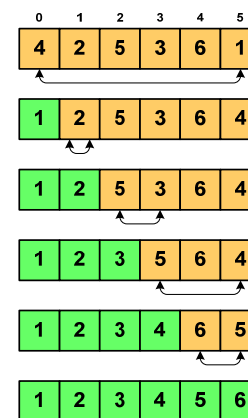
```
int main(void)
{
    int tab[N], i, j, tmp;
    // ...
    for (i=1; i<N; i++)
    {
        j=i;
        tmp=tab[i];
        while (tab[j-1]>tmp && j>0)
        {
            tab[j]=tab[j-1];
            j--;
        }
        tab[j]=tmp;
    }
}
```

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)
 - kubelkowe / przez zliczanie (bucket sort)

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;
    }
}
```

Koniec wykładu nr 7

Dziękuję za uwagę!