

Informatyka 1

Politechnika Białostocka - Wydział Elektryczny
Elektrotechnika, semestr II, studia stacjonarne I stopnia
Rok akademicki 2017/2018

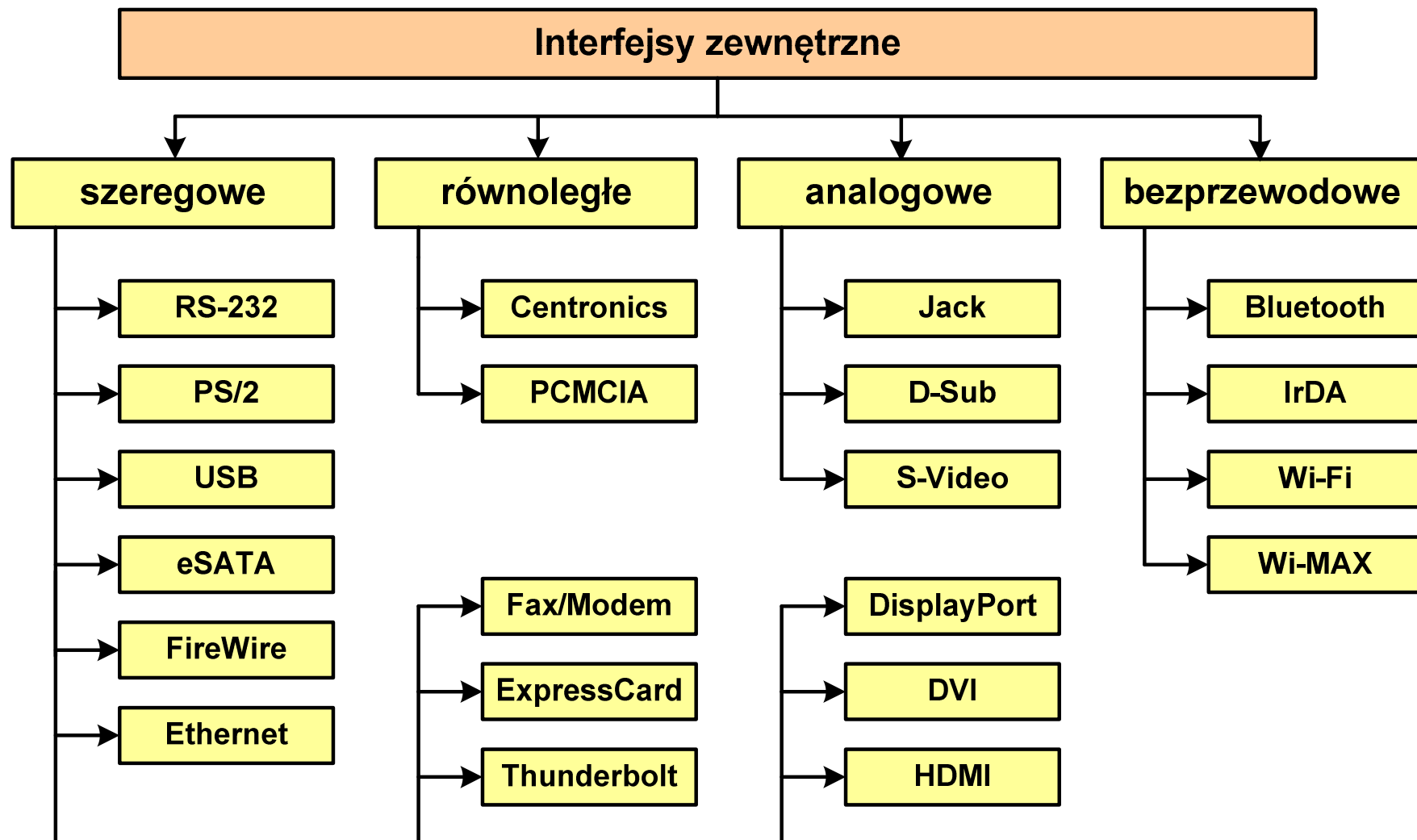
Wykład nr 6 (28.05.2018)

dr inż. Jarosław Forenc

Plan wykładu nr 6

- Budowa komputera
 - interfejsy zewnętrzne
- Struktura i funkcjonowanie komputera
 - procesor, rozkazy, przerwania
 - struktura połączeń, magistrala, DMA
 - pamięć komputerowa, hierarchia pamięci, pamięć podręczna
- Algorytmy komputerowe
 - definicje, sposoby opisu
 - rekurencja
 - złożoność obliczeniowa
 - algorytmy sortowania (proste wstawianie, proste wybieranie, bąbelkowe, quick-sort)

Interfejsy sprzętowe komputera



RS-232

(zewnętrzny, szeregowy)

- **RS-232** (Recommended Standard 232)
- 1962 rok
- magistrala przeznaczona do szeregowej transmisji danych
- najbardziej popularna wersja standardu: RS-232C
- przepustowość: do 115,2 kbit/s
- długość magistrali: do ok. 15 m
- w architekturze PC przewidziano obecność do 4 portów COM (COM1-COM4)
- zastosowania: mysz komputerowa, modemy, telefony komórkowe, łączenie dwóch komputerów kablem, starsze drukarki, tunery satelitarne, programowanie układów logicznych
- obecnie zastąpiona przez USB

RS-232

(zewnątrzny, szeregowy)



DE-9 (gniazdo męskie)



DB-25 (gniazdo żeńskie)



DE-9 (wtyk żeński)

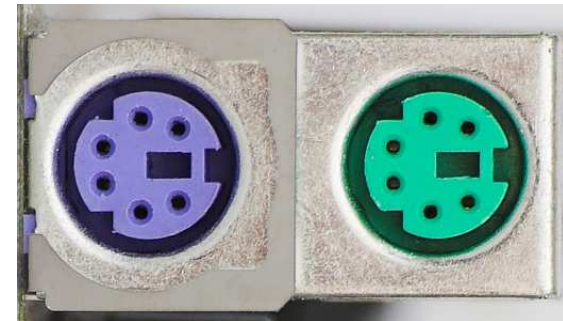


DB-25 (wtyk męski)

PS/2

(zewnątrzny, szeregowy)

- złącze używane do podłączenia klawiatury i myszy komputerowej
- IBM, 1987 rok
- zastąpiło złącze szeregowe myszy DE-9 i złącze klawiatury DIN
- przepustowość: 40 kB/s
- długość: 1,8 m
- zastąpione przez USB
- klawiatura - kolor fioletowy
- mysz - kolor zielony



6-pin Mini-DIN connector

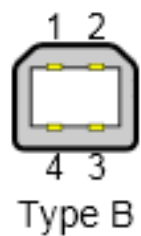
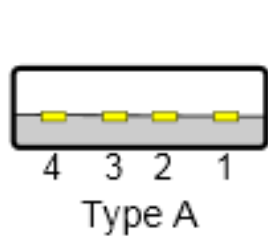
USB (zewnątrzny, szeregowy)

- **USB** (Universal Serial Bus)
- port komunikacyjny zastępujący stare porty szeregowy i równoległy
- zastosowanie: kamery i aparaty cyfrowe, telefony komórkowe, dyski, modemy, skanery, myszki, klawiatury, pen-drive'y, ...
- w systemie Windows obsługa USB od Windows 95 OSR2

Wersja	Przepustowość	Rok	Zasilanie	Przewód
USB 1.1 (Low Speed)	do 1,5 Mbit/s	1998	5 V, 500 mA	3 m
USB 1.1 (Full Speed)	do 12 Mbit/s	1998	5 V, 500 mA	5 m
USB 2.0 (Hi-Speed)	do 480 Mbit/s	2000	5 V, 500 mA	5 m
USB 3.0 (SuperSpeed)	do 4,8 Gbit/s	2008	5 V, 900 mA	3 m
USB 3.1 (SuperSpeed+)	do ok.10 Gbit/s	2014	5 V, 2 A	1 m

USB

(zewnątrzny, szeregowy)



Type A



Type B



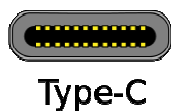
Mini-A

Mini-B



Micro-A

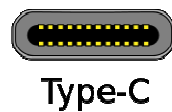
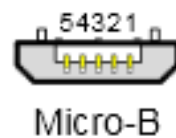
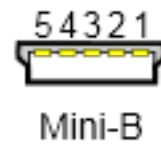
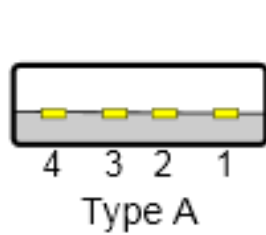
Micro-B



Type-C

USB

(zewnątrzny, szeregowy)



Type-B
SuperSpeed



Micro-B
SuperSpeed



Type-C

eSATA

(zewnątrzny, szeregowy)

- **eSATA** (external SATA) - 2004 rok
- zewnętrzny port SATA 3 Gbit/s przeznaczony do podłączania pamięci masowych zewnętrznych
- maksymalne przepustowości: 150 MB/s, 300 MB/s
- maksymalna długość kabla: 2 m



FireWire

(zewnątrzny, szeregowy)

- standard złącza szeregowego umożliwiający szybką komunikację i synchroniczne usługi w czasie rzeczywistym
- 1995 rok, dokument IEEE 1394
- przepustowość: 400/800/1600/3200 Mbit/s
- długość kabla: do 4,5 m
- złącze: IEEE-1394 (4, 6 lub 9 pinów)
- zastosowania: kamery i aparaty cyfrowe, skanery, drukarki



9-pin, 6-pin connectors



6-pin IEEE-1394 ports



4-pin connectors

Ethernet

(zewnątrzny, szeregowy)

- **BNC (Bayonet Neill-Concelman)** - złącze stosowane do łączenia sieci komputerowych zbudowanych z kabli koncentrycznych
- występuje w wersji 50 i 75-omowej



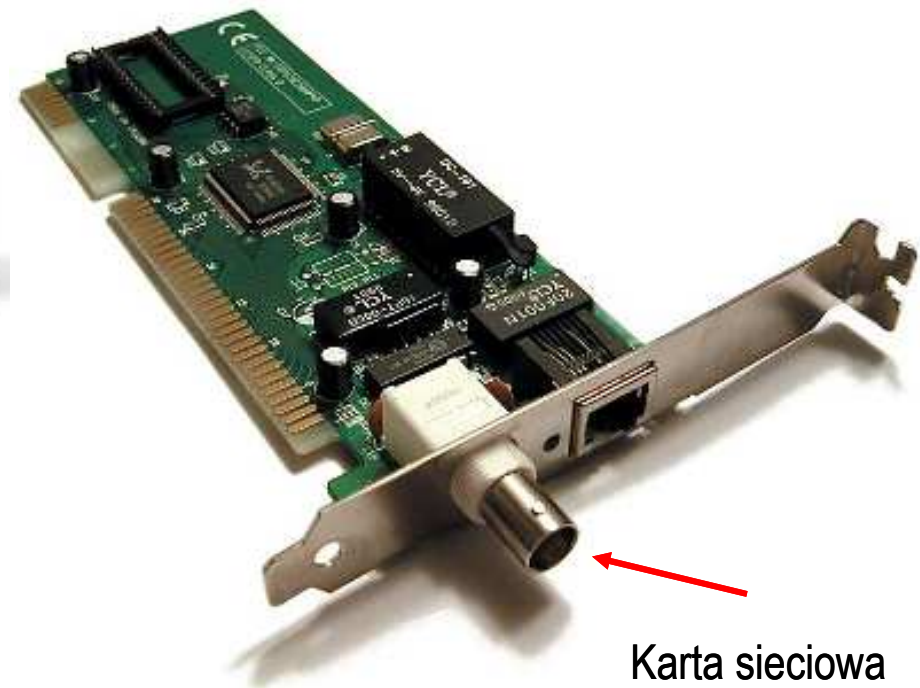
Złącze BNC



Trójnik



Terminator



Karta sieciowa
ze złączem BNC

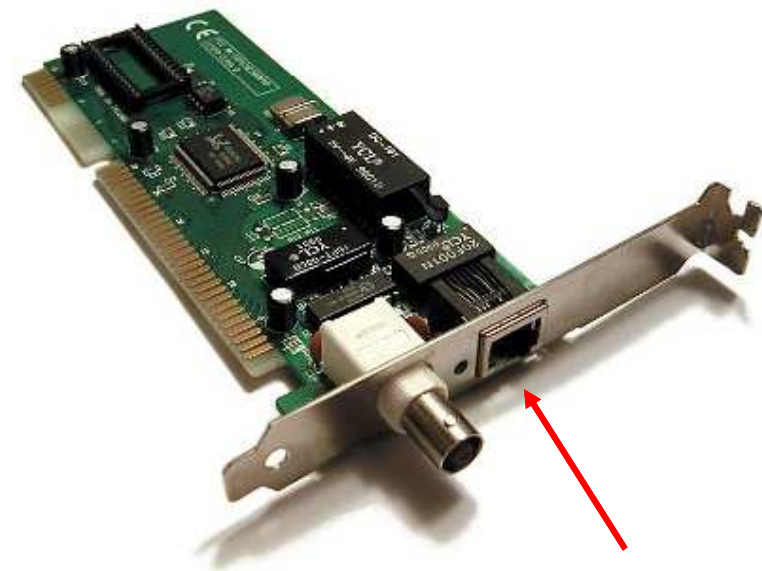
Ethernet

(zewnątrzny, szeregowy)

- **8P8C (8 Position 8 Contact)** - ośmiostykowe złącze wykorzystywane w sprzęcie komputerowym i telekomunikacyjnym
- nazywane RJ-45



Złącze 8P8C
na płycie głównej



Karta sieciowa
ze złączem 8P8C

Fax/Modem (RJ-11) (zewnętrzny, szeregowy)

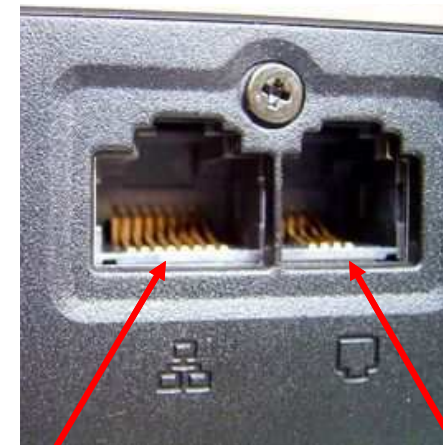
- **RJ-11 (Registered Jack - Type 11)** - złącze stosowane do podłączania sprzętu telekomunikacyjnego (linii telefonicznej)
- **6P2C (6 Position 2 Contact)** - sześciokrotny wtyk telefoniczny z dwoma stykami stosowany do zakończenia przewodów łączących sprzęt telekomunikacyjny



Wtyk RJ-11



Gniazdo RJ-11



RJ-45

RJ-11

Thunderbolt (zewnątrzny, szeregowy)

- interfejs do podłączania urządzeń zewnętrznych
- w założeniu ma zastąpić USB, FireWire, HDMI
- opracowanie - 2009 rok, pierwsze urządzenia - 2011 rok
- Intel, Apple Inc.
- przepustowość: 10 Gbit/s (Thunderbolt 1), 20 Gbit/s (Thunderbolt 2)



Złącze Thunderbolt w laptopie



Wtyczka
Thunderbolt

DisplayPort (zewnątrzny, szeregowy)

- **DisplayPort** - uniwersalny interfejs cyfrowy do przesyłania dźwięku i obrazu z prędkością 1,62 lub 2,7 Gb/s
- opracowany w 2006 roku
- dwukierunkowa wymiana informacji
- możliwa ochrona sygnału technologią DRM



Wtyk i gniazdo DisplayPort



Gniazdo DisplayPort

DVI

(zewnątrzny, szeregowy)

- **DVI (Digital Visual Interface)** - standard złącza pomiędzy kartą graficzną a monitorem komputera
- wersje:
 - **DVI-I** - przesyła dane cyfrowe i analogowe
 - **DVI-D** - przesyła dane cyfrowe
 - **DVI-A** - przesyła dane analogowe



DVI-I (Single Link)



DVI-I (Dual Link)



DVI-D (Single Link)



DVI-D (Dual Link)



DVI-A

HDMI

(zewnętrzny, szeregowy)

- **HDMI (High Definition Multimedia Interface)** - interfejs do przesyłania cyfrowe, nieskompresowanego sygnału audio i wideo
- wrzesień 2003 r.
- wersje:
 - 1.0, 1.1, 1.2, 1.3, 1.4
 - 2.0, 2.0a, 2.0b (4096x2160p60)
 - 2.1 (2017 r., 48 Gb/s, 7680x4320p120)



IEEE 1284 (zewnątrzny, równoległy)

- port równoległy wykorzystywany do podłączenia urządzeń peryferyjnych (drukarki, skanery, plotery)
- nazywany **portem równoległym** lub **LPT** (Line Print Terminal)
- standard IEEE 1284 został opracowany w 1994 roku
- zapewnia kompatybilność z używanym w latach 70-tych jednokierunkowym portem **Centronics**
 - LPT1, I/O Port 0x378, IRQ7 + LPT2, I/O Port 0x278, IRQ5
- protokoły transmisji danych (wybrane):
 - **SPP** (Standard Parallel Port) - tryb kompatybilności z Centronics, możliwość transmisji dwukierunkowej, transfer do 150 kb/s, obsługa za pomocą przerwań
 - **EPP** (Enhanced Parallel Port) - sprzętowo ustalone parametry transmisji (automatycznie), brak kanału DMA
 - **ECP** (Extended Capability Port) - używa DMA, transfer do 2 Mb/s

IEEE 1284

(zewnątrzny, równoległy)



Port równoległy w laptopie



DB-25



Port równoległy
na płycie głównej

PCMCIA (zewnątrzny, równoległy)

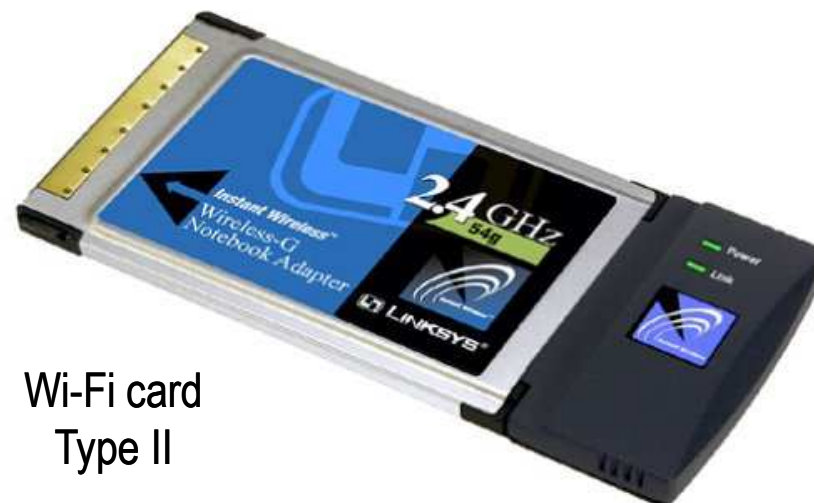
- Personal Computer Memory Card International Association
- 1991 - standard interfejsu wejścia-wyjścia dla kart pamięci
- w kolejnych latach przekształcony w karty rozszerzeń, pełniące funkcje modemu, faksmodemu, karty sieciowej, Wi-Fi
- ustandaryzowane wymiary: 85,6 × 54 mm
- podział ze względu na wielkość:
 - **typ I** - grubość 3,3 mm; karty pamięci SRAM lub Flash
 - **typ II** - grubość 5,0 mm; karty rozszerzeń (modem, karta sieciowa)
 - **typ III** - grubość 10,5 mm; karty rozszerzeń (dysk twardy)
- podział ze względu na interfejs:
 - **PC Card 16** - interfejs magistrali ISA 16bit, zasilanie 5 V
 - **CardBus** - interfejs magistrali PCI 32bit, zasilanie 3-3,3 V

PCMCIA

(zewnątrzny, równoległy)



USB card
Type II



Wi-Fi card
Type II

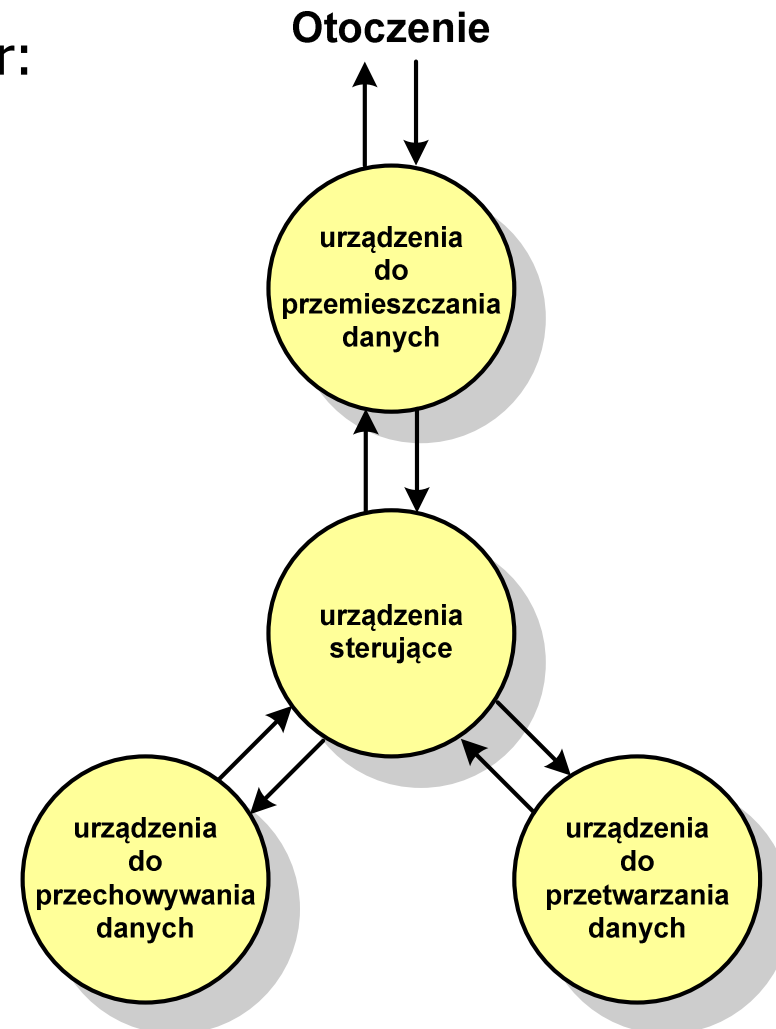


gniazda
PCMCIA



Funkcjonowanie komputera

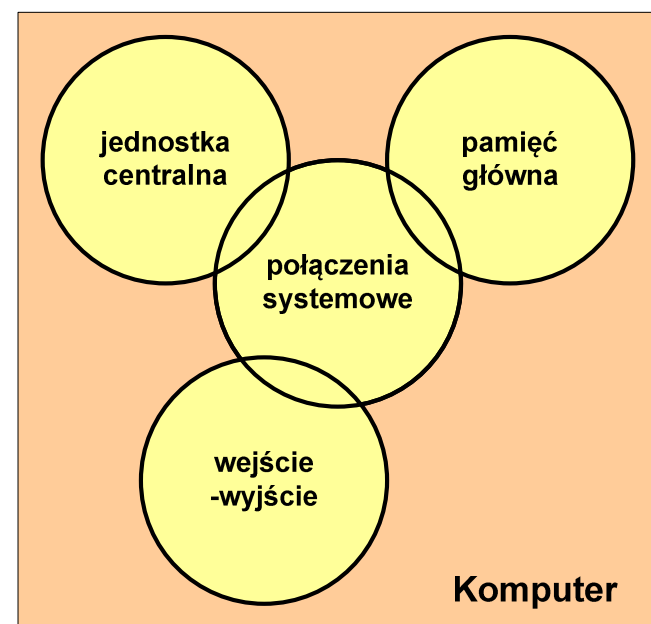
- Funkcje realizowane przez komputer:
 - przetwarzanie danych
 - przechowywanie danych (krótkotrwałe lub długotrwałe)
 - przenoszenie danych (pomiędzy komputerem a światem zewnętrznym)
 - sterowanie (powyższymi funkcjami)



Ogólna struktura systemu komputerowego

■ Komputer tworzą cztery główne składniki:

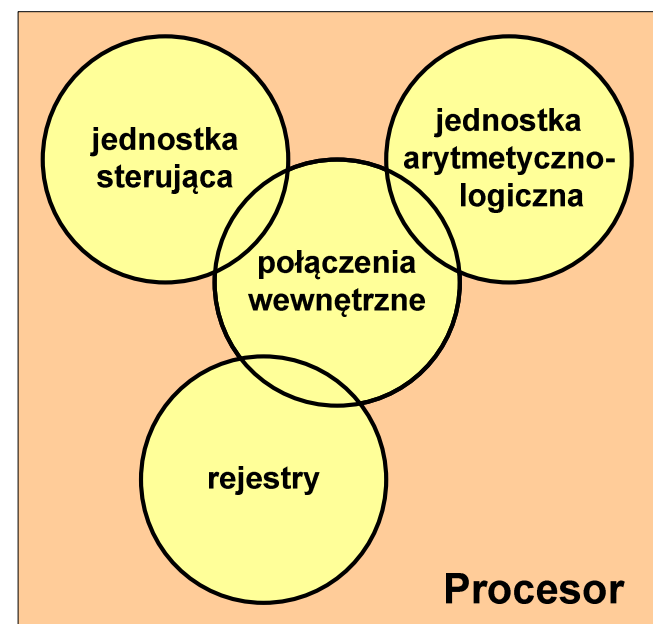
- **procesor** (jednostka centralna, CPU)
- steruje działaniem komputera
i realizuje przetwarzanie danych
- **pamięć główna** - przechowuje dane
- **wejście-wyjście** - przenosi dane
między komputerem a jego
otoczeniem zewnętrznym
- **połączenia systemu** - mechanizmy
zapewniające komunikację między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



Ogólna struktura procesora

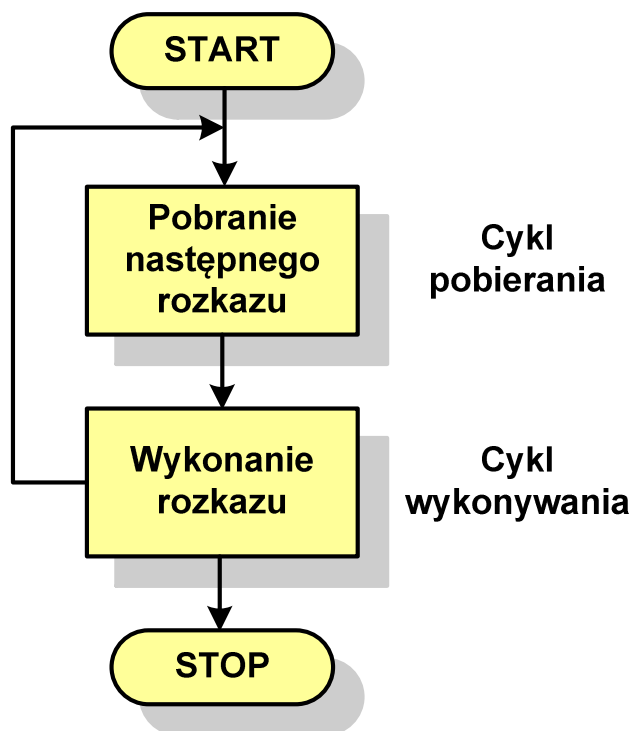
■ Główne składniki strukturalne procesora to:

- **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

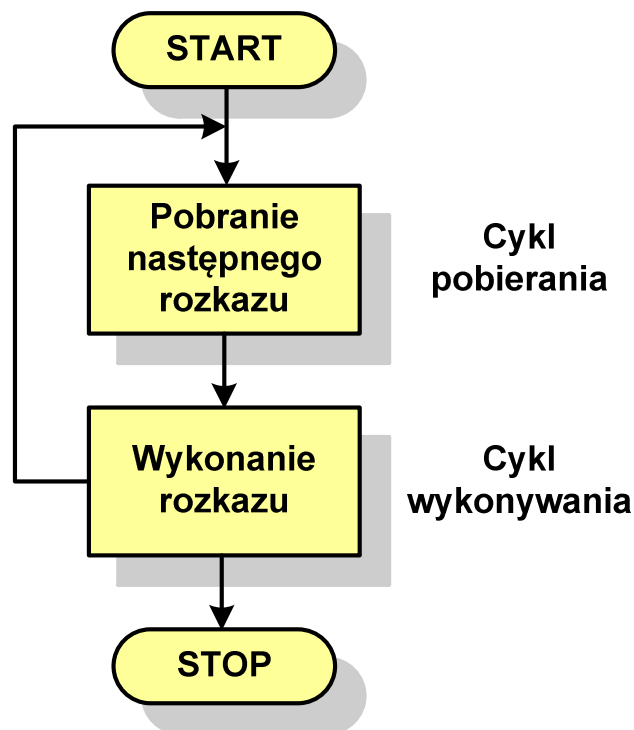
- Podstawowe zadanie komputera to wykonywanie **programu**
- Program składa się z **rozkazów** przechowywanych w pamięci
- Rozkazy są przetwarzane w dwu krokach:



- 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

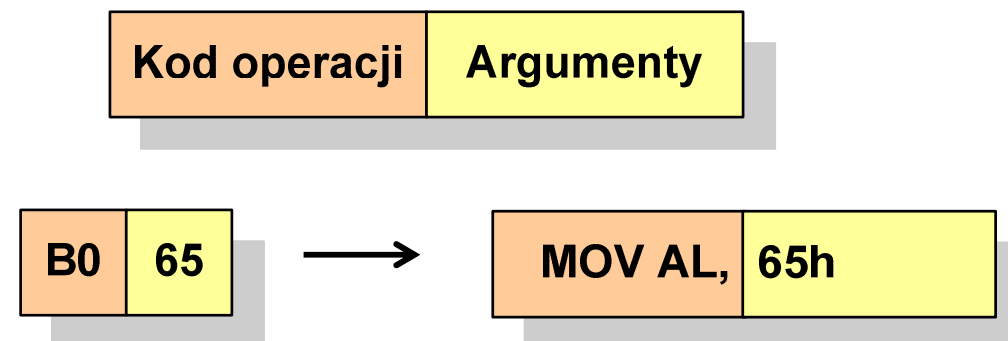
- Podstawowe zadanie komputera to wykonywanie **programu**
- Program składa się z **rozkazów** przechowywanych w pamięci
- Rozkazy są przetwarzane w dwu krokach:



- 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

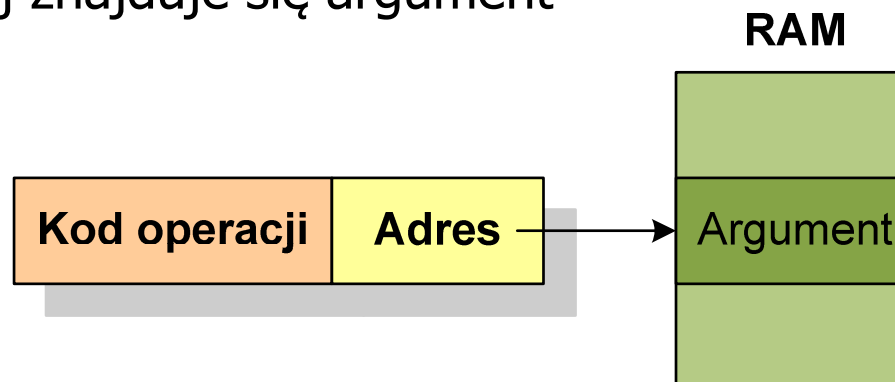


Działanie komputera

- **Tryb adresowania** - sposób określania miejsca przechowywania argumentów rozkazu (operandów)
- **Adresowanie natychmiastowe** - argument znajduje się w kodzie rozkazu

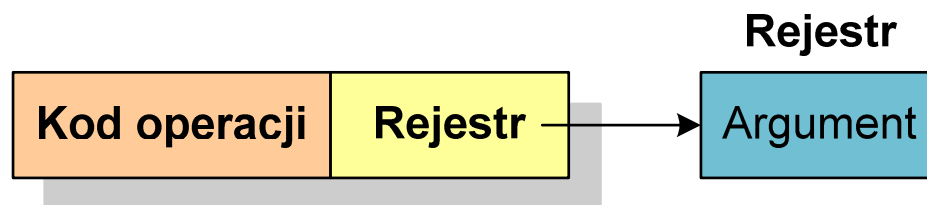


- **Adresowanie bezpośrednio** - kod rozkazu zawiera adres komórki pamięci, w której znajduje się argument

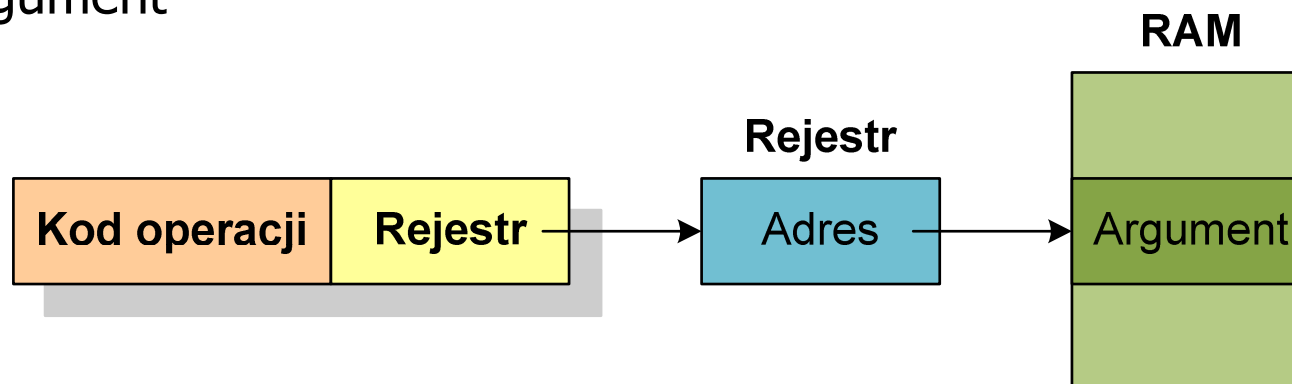


Działanie komputera

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

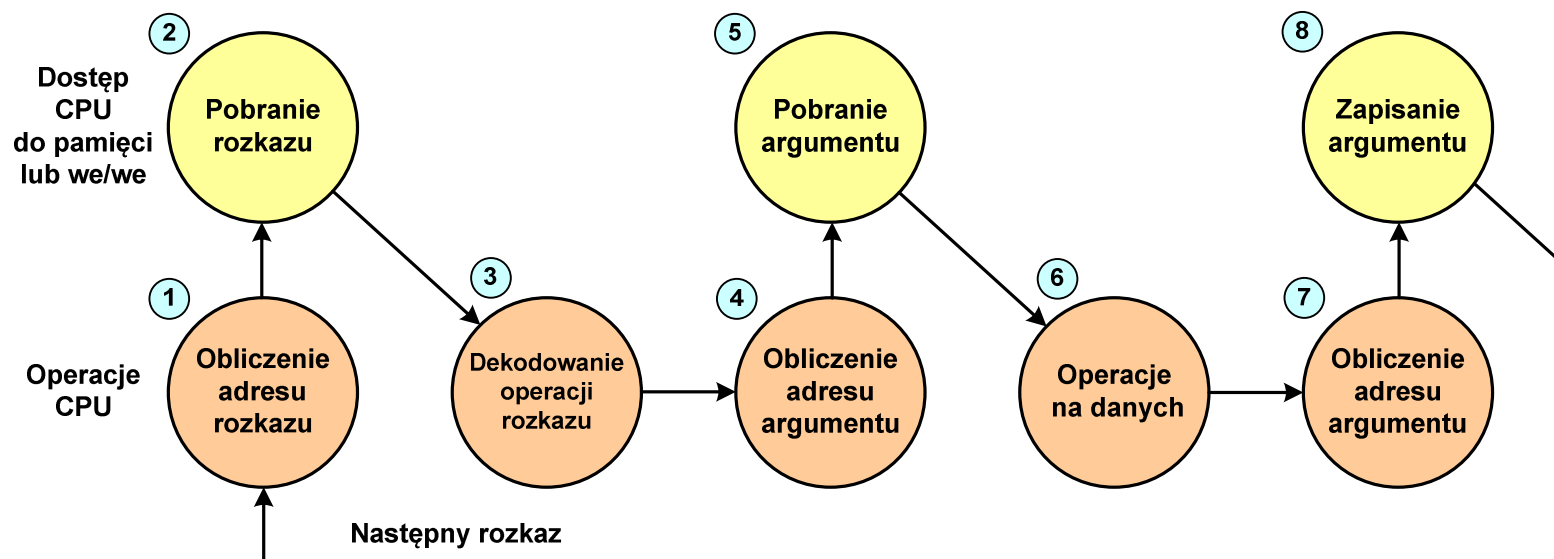


- **Adresowanie pośrednie** - kod rozkazu zawiera oznaczenie rejestru, w którym znajduje się adres komórki pamięci, w której znajduje się argument



Działanie komputera

- Graf stanów cyklu wykonania rozkazu ma następującą postać:



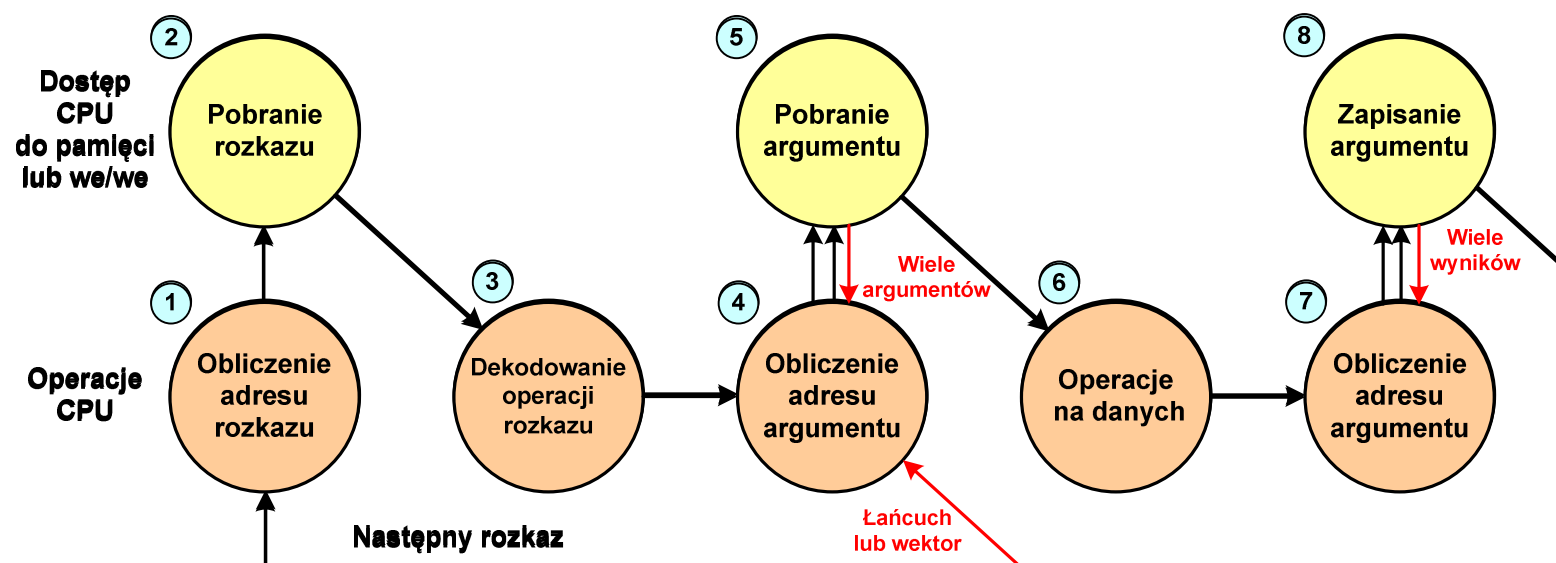
(3) - analiza rozkazu w celu określenia rodzaju operacji, która ma być wykonana oraz w celu określenia argumentu (jednego lub kilku)

(8) - zapisanie wyniku w pamięci lub skierowanie go do we-wy

- Nie wszystkie stany z powyższego schematu muszą występować

Działanie komputera

- Graf stanów cyklu wykonania rozkazu ma następującą postać:



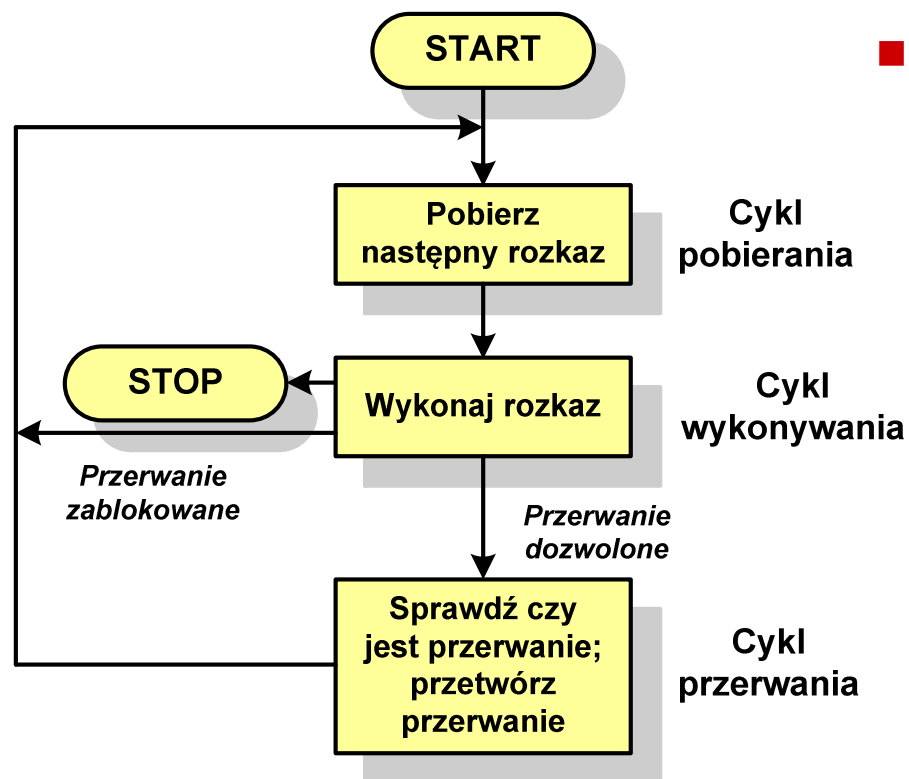
- Mogą wystąpić sytuacje, w których jeden rozkaz może określać operacje na wektorze liczb lub na szeregu znaków, co wymaga powtarzania operacji pobrania i/lub przechowywania.

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 przerwań wymaga dodania cyklu przerwania do cyklu rozkazu:

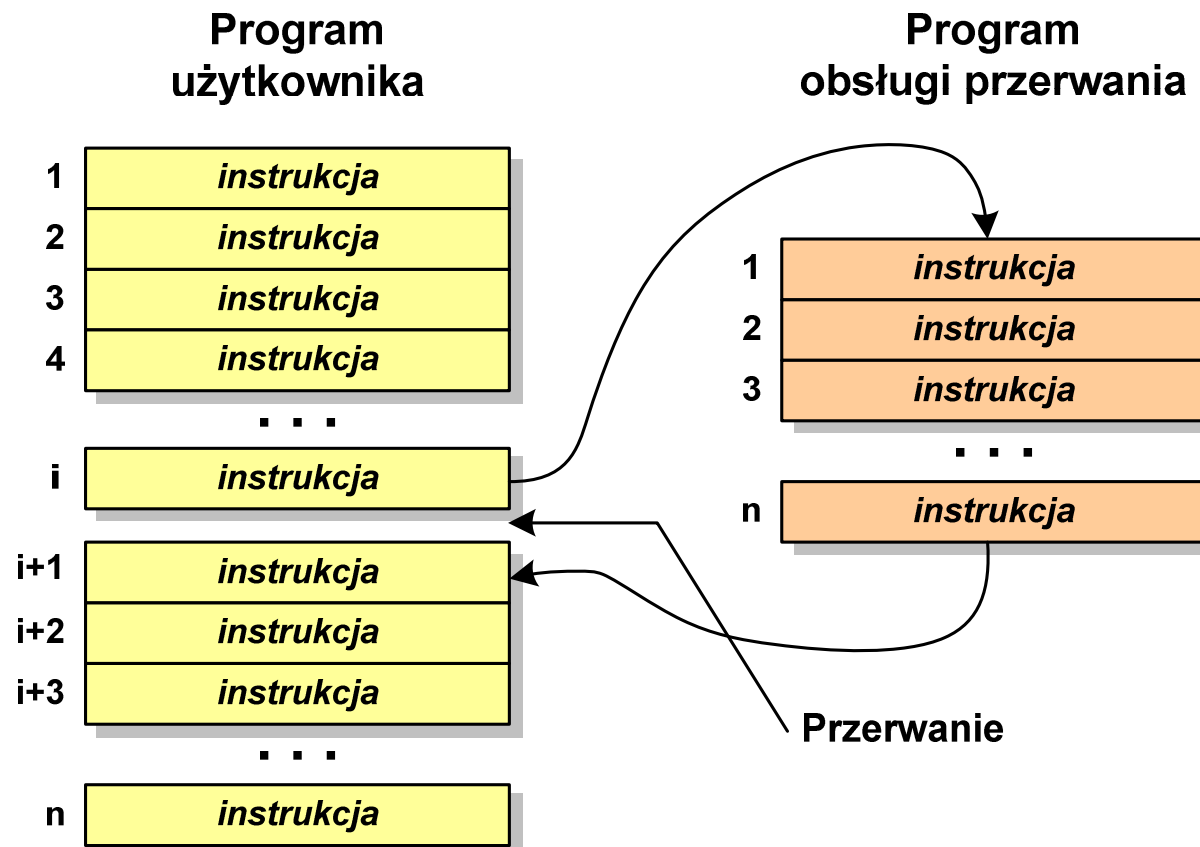


- Po sygnale przerwania procesor:

- zawiesza wykonanie bieżącego programu i zachowuje jego kontekst
- ustawia licznik programu na początkowy adres programu obsługi przerwania
- wykonuje program obsługi przerwania
- wznowia wykonywanie programu użytkowego.

Działanie komputera - przerwania

- Jak działa przerwanie?



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

Program w asemblerze

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

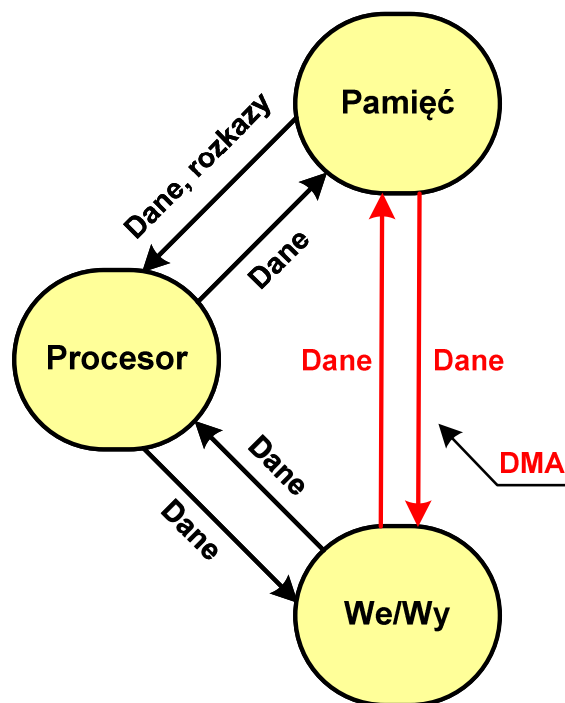
    handler:
        pusha
        push ds
        pop ds
        popa
        iret

    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 - DMA

- Struktura połączeń musi umożliwiać przesyłanie następujących danych:

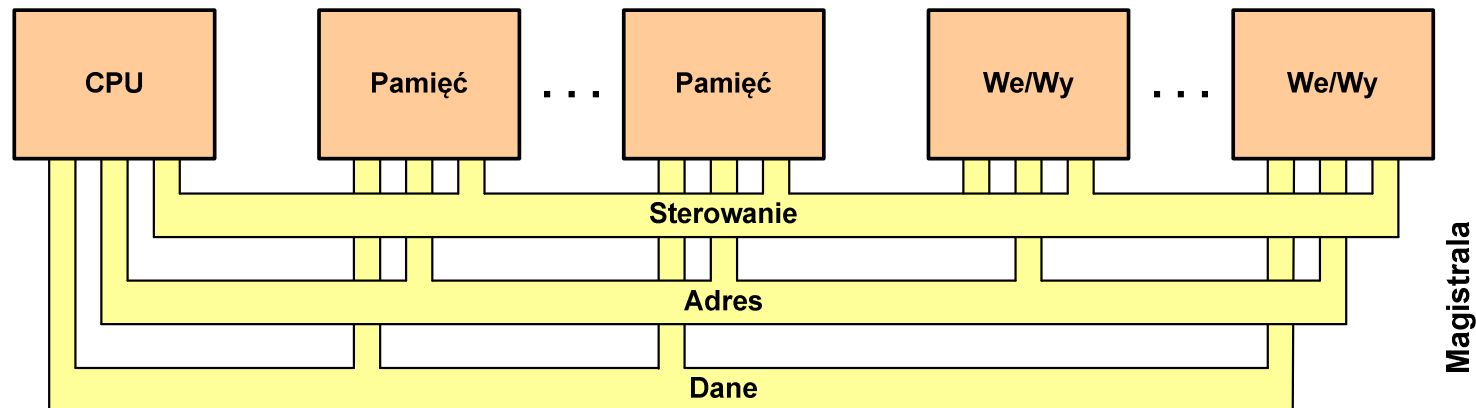


- DMA - bezpośredni dostęp do pamięci:

- najczęściej procesor bezpośrednio odczytuje dane z pamięci i zapisuje oraz komunikuje się z urządzeniami we-wy
- w pewnych przypadkach pożądane jest umożliwienie bezpośredniej wymiany danych między we-wy a pamięcią
- podczas takiego przesyłania moduł we-wy odczytuje lub zapisuje rozkazy w pamięci, uwalniając procesor od odpowiedzialności za tę wymianę
- powyższe operacje nazywane są bezpośrednim dostępem do pamięci (ang. DMA - Direct Memory Access)

Działanie komputera - magistrala

- Najczęściej stosowana struktura połączeń to **magistrala**
- Magistrala składa się z wielu linii komunikacyjnych, którym przypisane jest określone znaczenie i określona funkcja
- Linie dzielą się na trzy grupy: linie danych, adresów i sterowania



Działanie komputera - magistrala

■ Linie danych:

- przenoszą dane między modułami systemu
- wszystkie linie danych nazywane są **szyną danych**
- liczba linii określa szerokość szyny danych (8, 16, 32, 64 bity)

■ Linie adresowe:

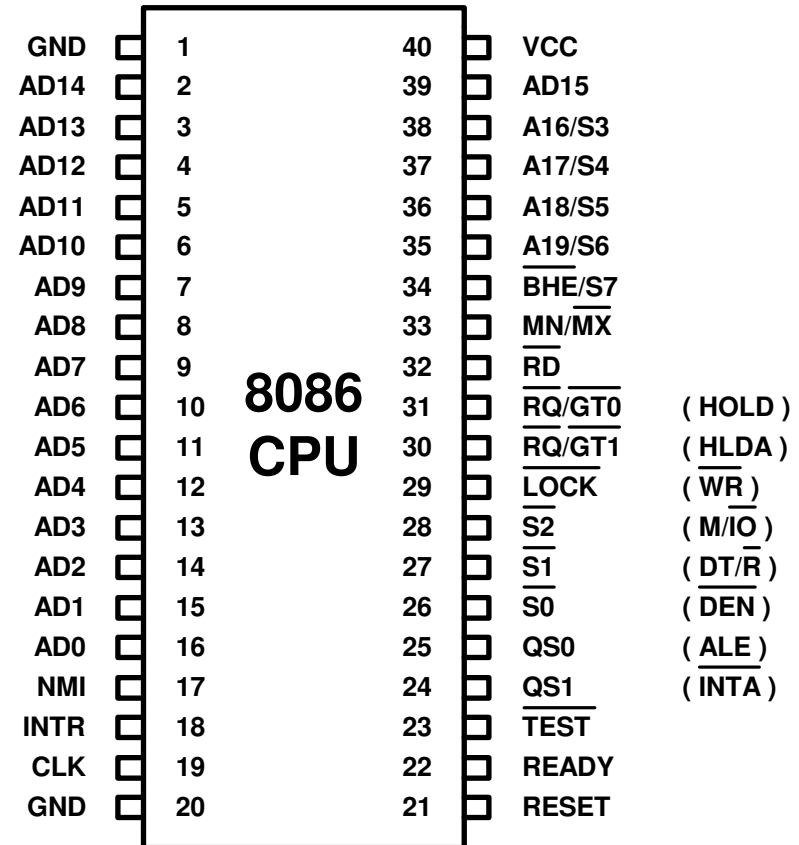
- służą do określania źródła i miejsca przeznaczenia danych przesyłanych magistralą
- liczba linii adresowych (szerokość szyny adresowej) określa maksymalną możliwą pojemność pamięci systemu

■ Linie sterowania:

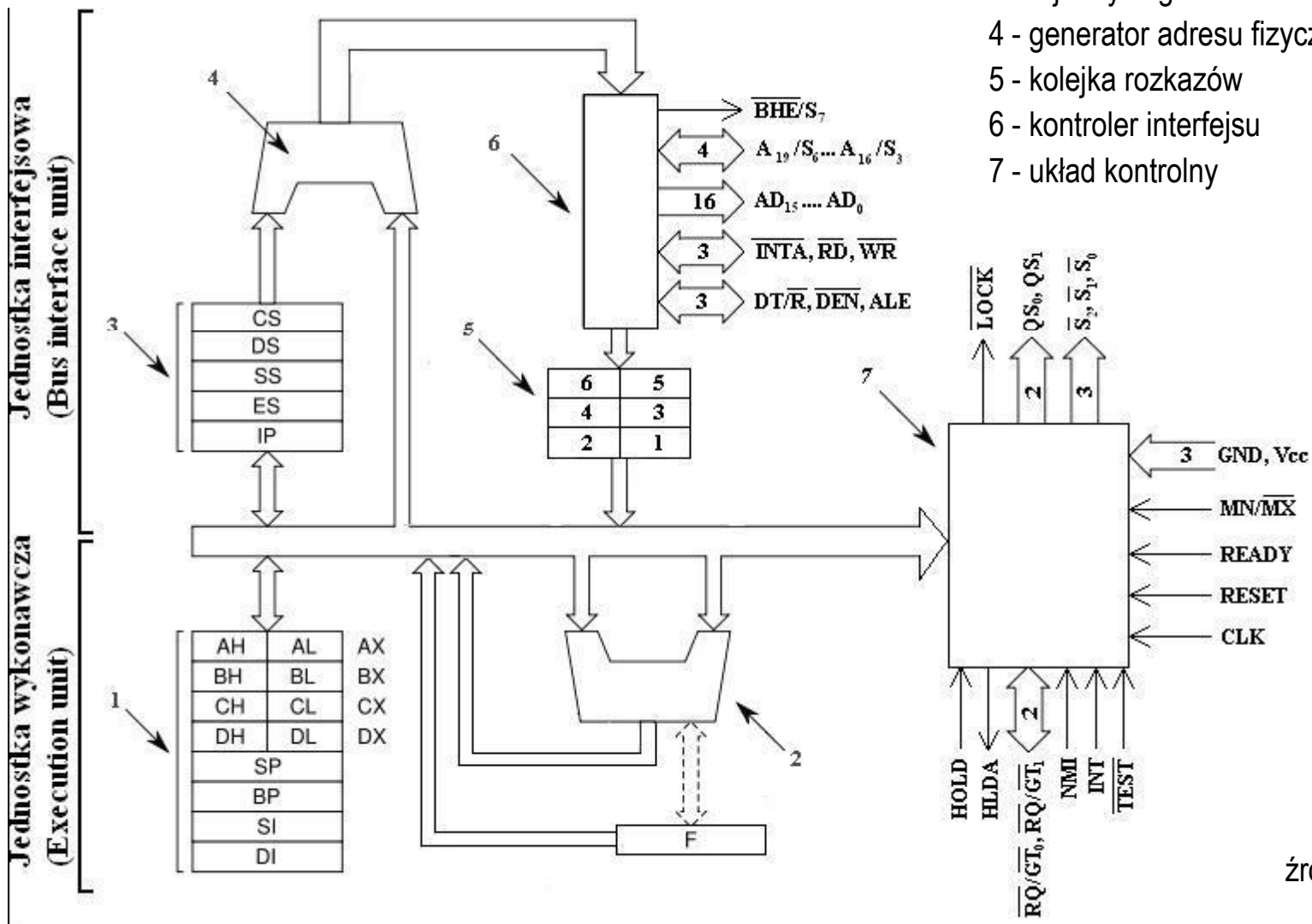
- służą do sterowania dostępem do linii danych i linii adresowych

Intel 8086

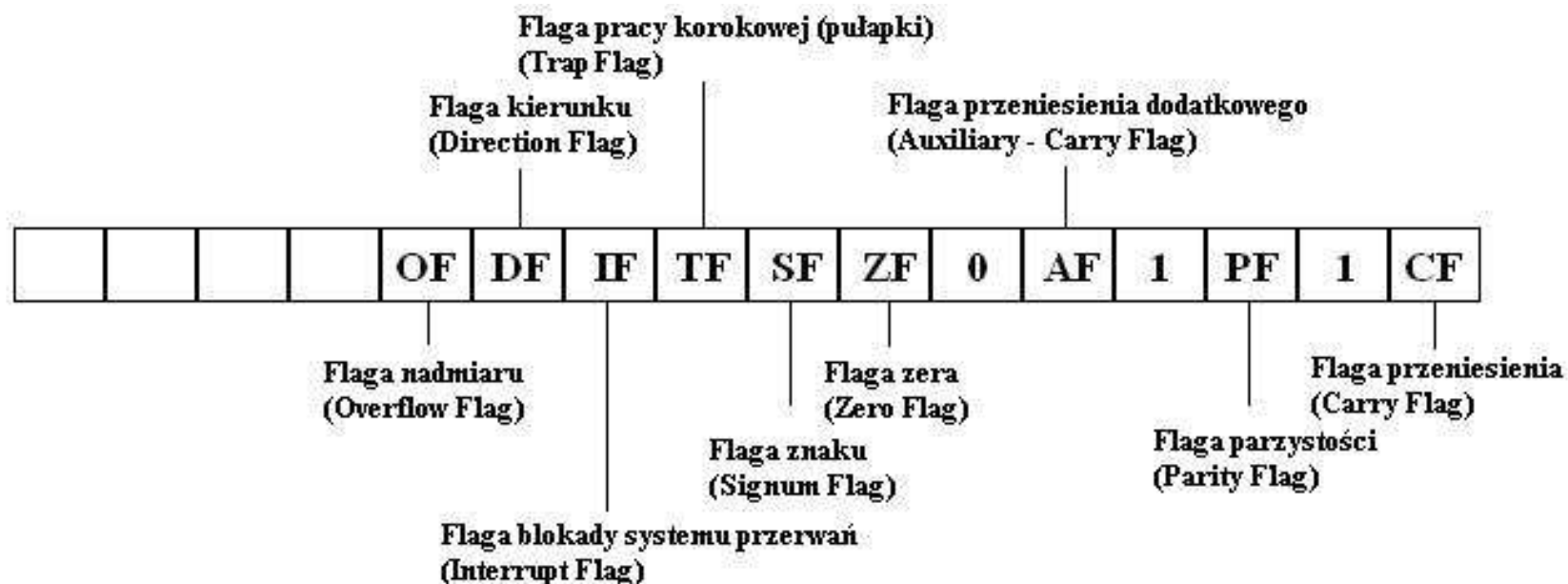
- 1978 rok
- Procesor 16-bitowy
- 16-bitowa magistrala danych
- 20-bitowa magistrala adresowa
- Adresowanie do 1 MB pamięci
- Częstotliwość: 10 MHz
- Multipleksowane magistrale:
danych i adresowa
- Litografia: 3 μm



Intel 8086



Intel 8086 - Rejestr flag



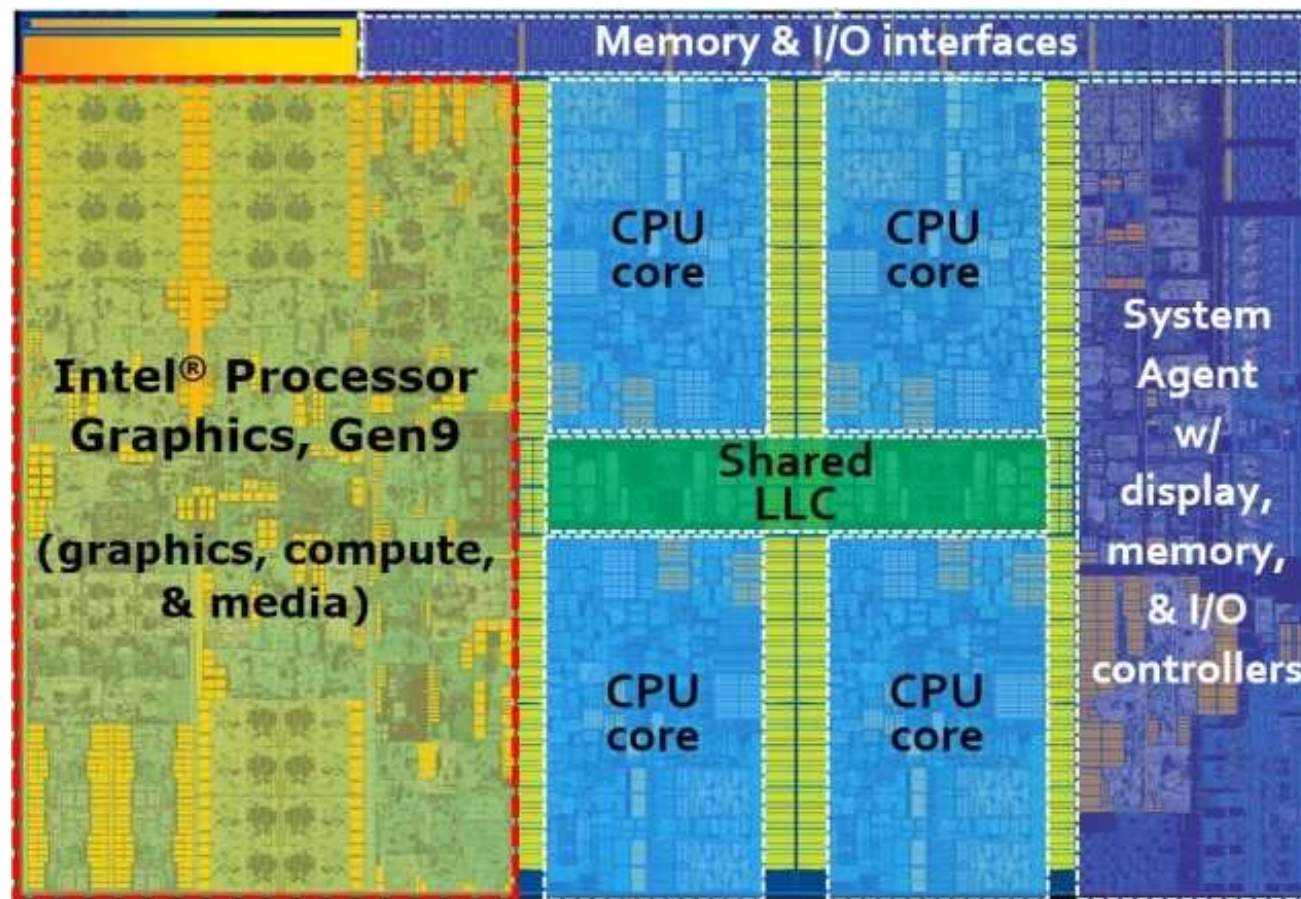
źródło: wikipedia

Intel Core i7-6700K

- III kwartał 2015 rok
- Procesor 64-bitowy
- Liczba rdzeni/wątków: 4/8
- Wielkość pamięci: 64 GB (DDR4/DDR3L)
- Częstotliwość: 4 GHz
- Układ graficzny:
Intel HD Graphics 530
- Litografia: 14 nm
- Nazwa kodowa: Skylake

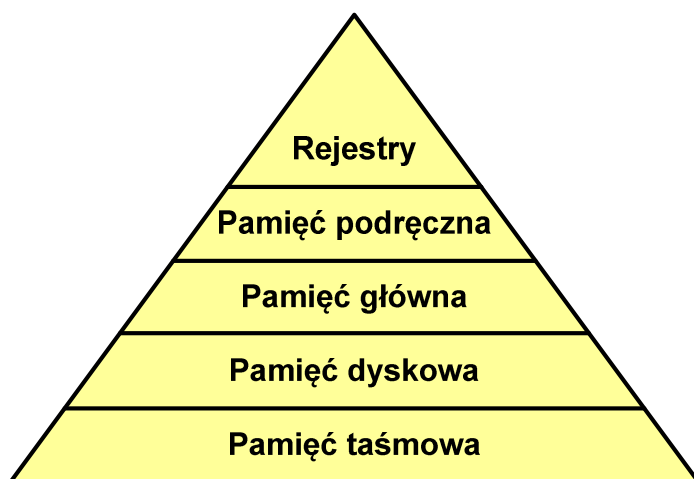


Intel Core i7-6700K - struktura



Systemy pamięci komputerowych

- Ze względu na **położenie** pamięci w stosunku do komputera wyróżniamy pamięć:
 - procesora (rejstry)
 - wewnętrzną (pamięć główna)
 - zewnętrzną (pamięć pomocnicza - pamięci dyskowe)
- W systemach komputerowych nie stosuje się jednego typu pamięci, ale **hierarchię pamięci**



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

Półprzewodnikowa pamięć główna

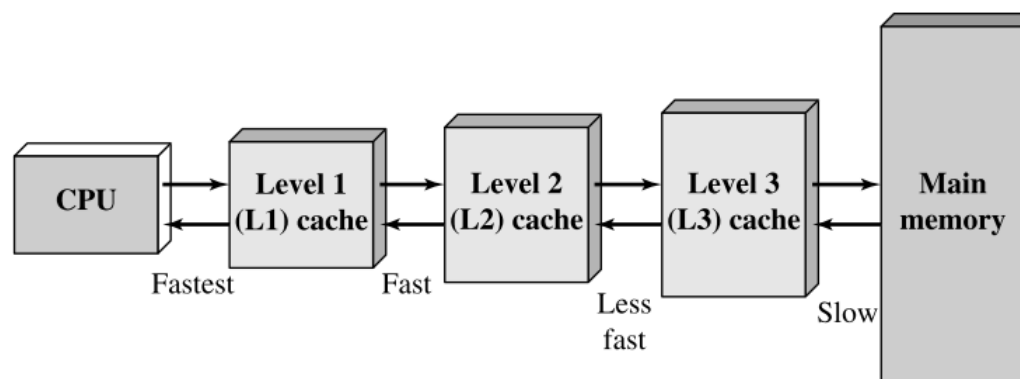
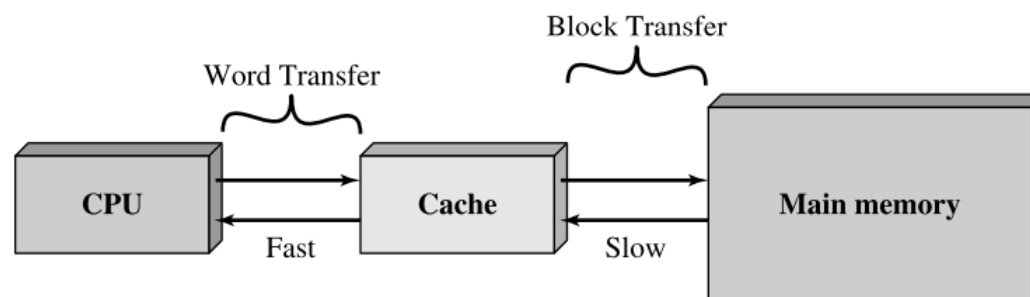
- **RAM** (Random Access Memory) - pamięć o dostępie swobodnym
 - odczyt i zapis następuje za pomocą sygnałów elektrycznych
 - pamięć ulotna - po odłączeniu zasilania dane są tracone
 - **DRAM** - pamięć dynamiczna:
 - przechowuje dane podobnie jak kondensator ładunek elektryczny
 - wymaga operacji odświeżania
 - 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

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

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

Algorytm - definicje

Definicja 1

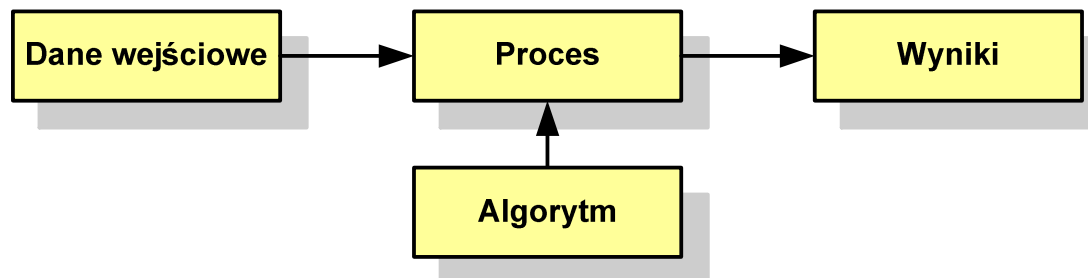
- Skończony, uporządkowany ciąg jasno zdefiniowanych czynności, koniecznych do wykonania pewnego zadania

Definicja 2

- Metoda rozwiązania zadania

Definicja 3

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



Algorytmy

- Słowo „**algorytm**” pochodzi od nazwiska Muhammada ibn-Musy **al-Chuwarizmiego** (po łacinie pisanego jako **Algorismus**), matematyka perskiego z IX wieku
- Badaniem algorytmów zajmuje się **algorytmika**
- Algorytm może zostać **zaimplementowany** w postaci **programu komputerowego**
- Przetłumaczenie algorytmu na wybrany język programowania nazywane jest też **kodowaniem** algorytmu
- Ten sam algorytm może być zaimplementowany (zakodowany) w różny sposób przy użyciu różnych języków programowania.

Podstawowe cechy algorytmu

- Posiada dane wejściowe (w ilości większej lub równej zero) pochodzące z dobrze zdefiniowanego zbioru
- Zwraca wynik
- Jest precyzyjnie zdefiniowany (każdy krok algorytmu musi być jednoznacznie określony)
- Poprawność (dla każdego z założonego dopuszczalnego zestawu danych wejściowych)
- Kończy działanie po skończonej liczbie kroków (powinna istnieć poprawnie działająca reguła stopu algorytmu)
- Efektywność (jak najkrótszy czas wykonania i jak najmniejsze zapotrzebowanie na pamięć).

Sposoby opisu algorytmów

1. Opis słowny w języku naturalnym lub w postaci listy kroków (opis w punktach)
2. Schemat blokowy
3. Pseudokod (nieformalna odmiana języka programowania)
4. Wybrany język programowania

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

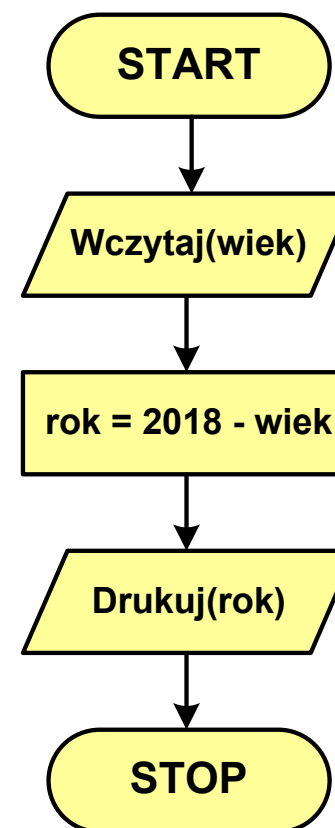
Opis algorytmu: Ziemniaki obrać i pokroić w plasterki. Kiełbasę pokroić w plasterki. Ziemniaki wrzucić na gorącą oliwę na patelni i przyrumienić z obu stron. Kiełbasę wrzucić na gorącą oliwę na patelni i przyrumienić z obu stron. Ubić jajka i dodać do połączonych ziemniaków i kiełbasy. Dodać sól i pieprz.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

Lista kroków

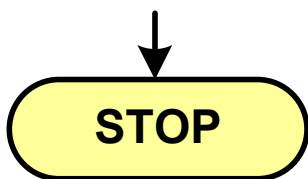
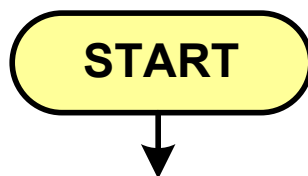
- Uporządkowany opis wszystkich czynności, jakie należy wykonać podczas realizacji algorytmu
- **Krok** jest to pojedyncza czynność realizowana w algorytmie
- Kroki w algorytmie są numerowane, operacje wykonywane są zgodnie z rosnącą numeracją kroków
- Jedynym odstępstwem od powyższej reguły są operacje 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

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**)

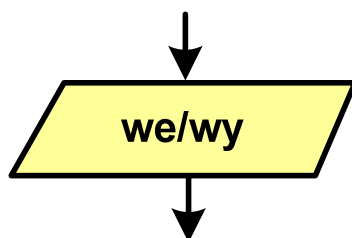


Schemat blokowy - symbole graficzne

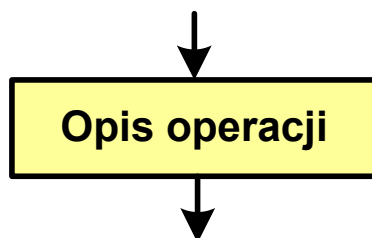


- **blok startowy**, początek algorytmu
 - wskazuje miejsce rozpoczęcia algorytmu
 - ma jedno wyjście
 - może występować tylko jeden raz
-
- **blok końcowy**, koniec algorytmu
 - wskazuje miejsce zakończenia algorytmu
 - ma jedno wejście
 - musi występować przynajmniej jeden raz

Schemat blokowy - symbole graficzne

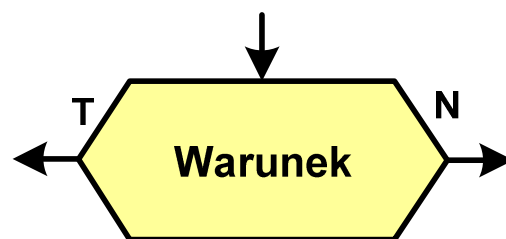
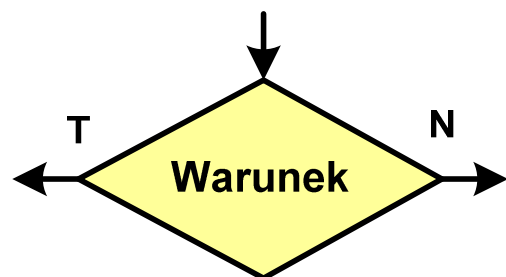


- **blok wejścia-wyjścia**
- poprzez ten blok wprowadzane są (czytane) dane wejściowe i wyprowadzane (zapisywane) wyniki
- ma jedno wejście i jedno wyjście

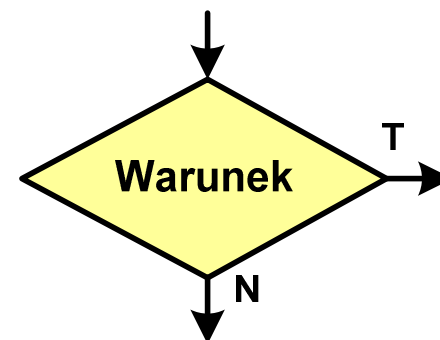
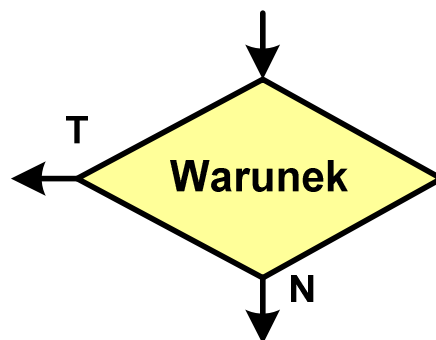


- **blok wykonawczy**, blok funkcyjny, opis procesu
- zawiera jedno lub kilka poleceń (elementarnych instrukcji) wykonywanych w podanej kolejności
- instrukcją może być np. operacja arytmetyczna, podstawienie
- ma jedno wejście i jedno wyjście

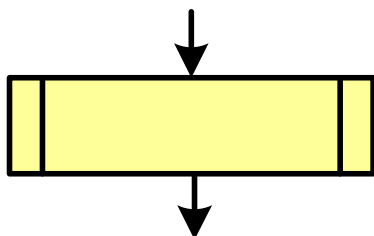
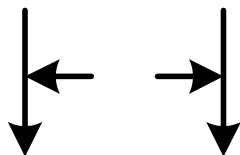
Schemat blokowy - symbole graficzne



- **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

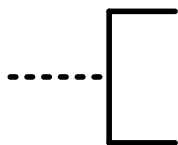


Schemat blokowy - symbole graficzne

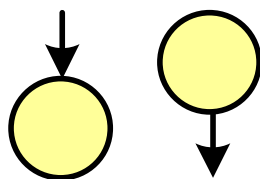


- **linia przepływu**, połączenie, linia
- występuje w postaci linii zakończonej strzałką
- określa kierunek przemieszczania się po schemacie
- łączy inne bloki występujące na schemacie
- linie pochodzące z różnych części algorytmu mogą zbiegać się w jednym miejscu
- **podprogram**
- wywołanie wcześniej zdefiniowanego fragmentu algorytmu (podprogramu)
- ma jedno wejście i jedno wyjście

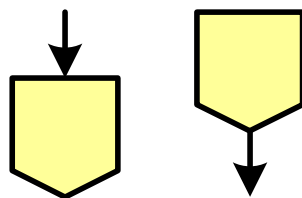
Schemat blokowy - symbole graficzne



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



- **łącznik stronicowy** (wewnętrzny)
- połączenie dwóch odrębnych części schematu znajdujących się na tej samej stronie
- łączniki opisywane są etykietami



- **łącznik międzystronicowy** (zewnątrzny)
- połączenie dwóch odrębnych części schematu znajdujących się na różnych stronach
- łączniki opisywane są etykietami

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)

Największy wspólny dzielnik - algorytm Euklidesa

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

$$\text{NWD}(1675, 3752) = ?$$

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	

$$\text{NWD}(1675, 3752) = 67$$

Algorytm Euklidesa - lista kroków

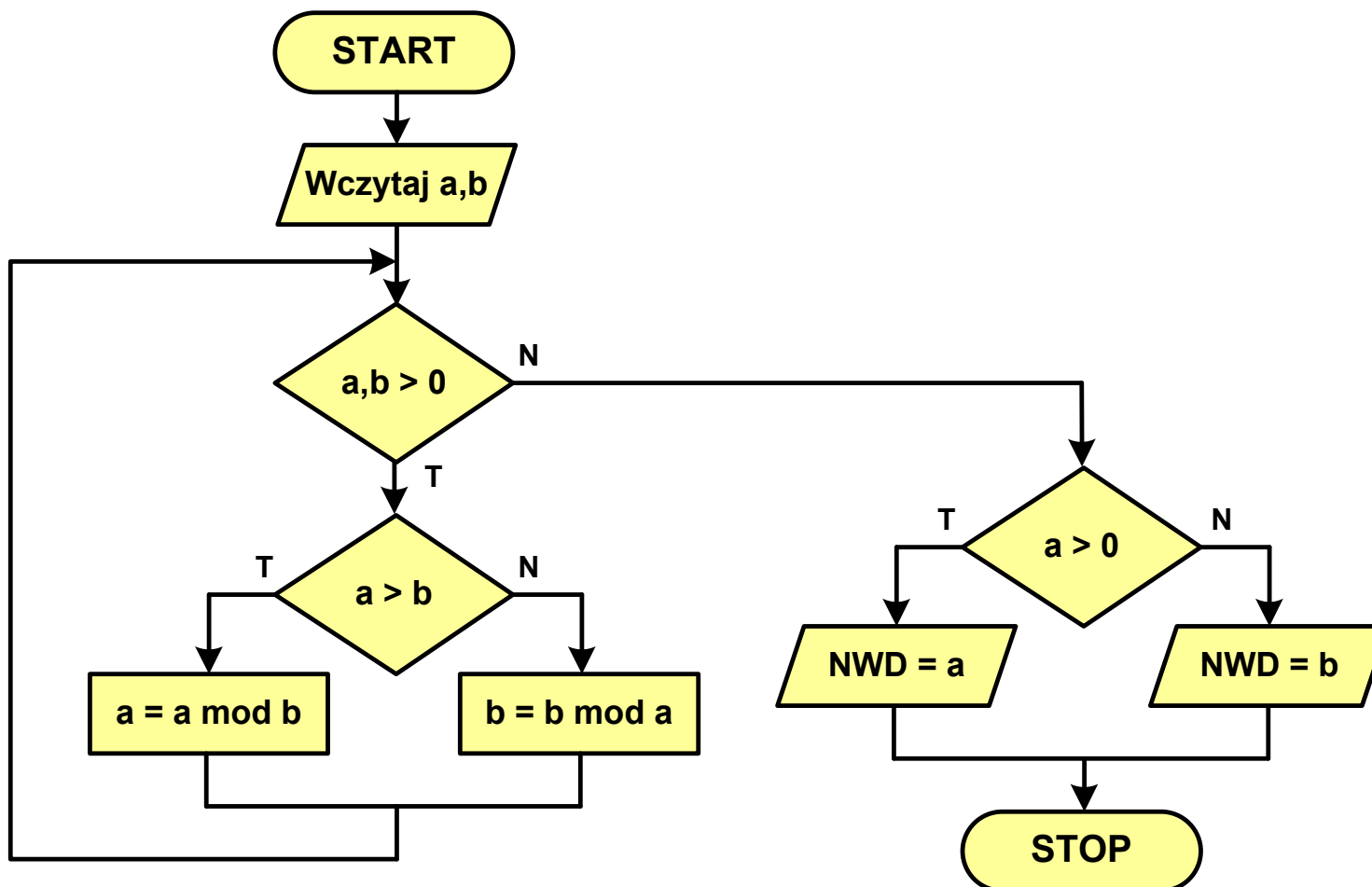
Dane wejściowe: niezerowe liczby naturalne a i b

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 następnie przejdź do **kroku 4**
3. Jeśli a jest większe od b , to weź za a resztę z dzielenia a przez b , w przeciwnym razie 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 $NWD(a,b)$

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;

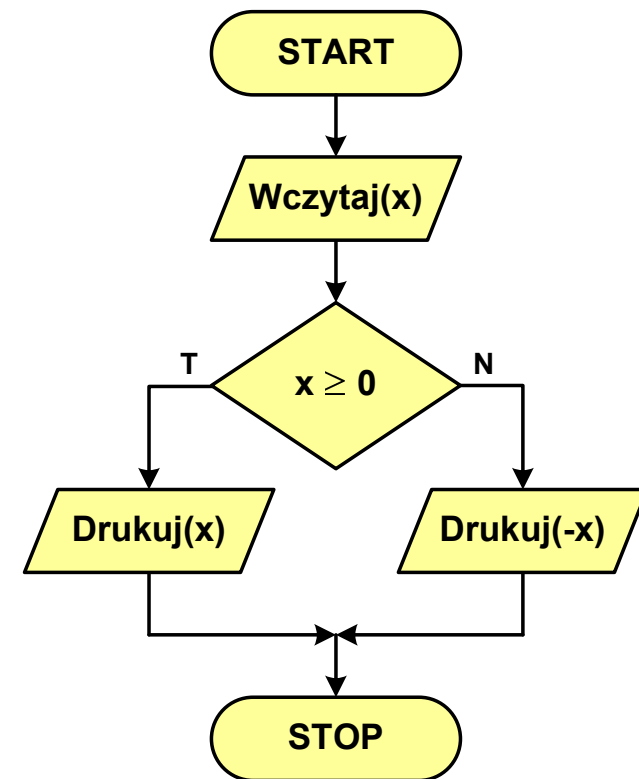
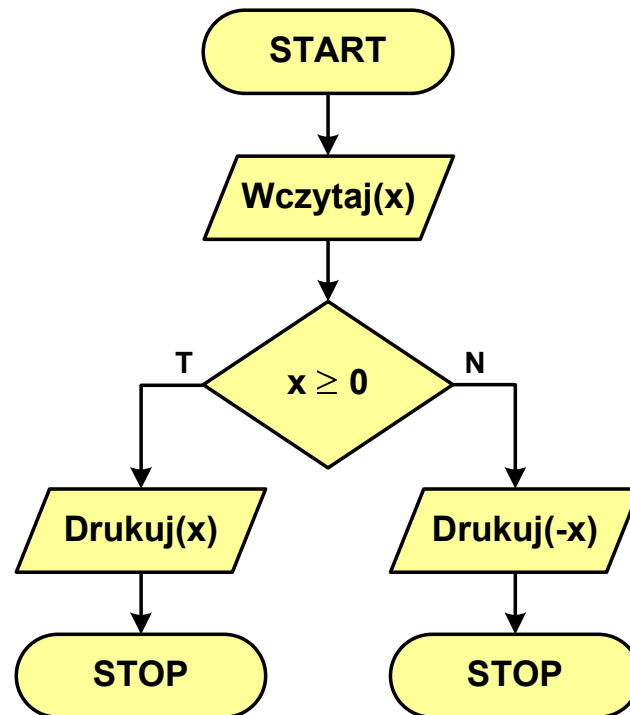
    while (a>0 && b>0)
        if (a>b)
            a = a % b;
        else
            b = b % a;

    if (a>0)
        NWD = a;
    else
        NWD = b;

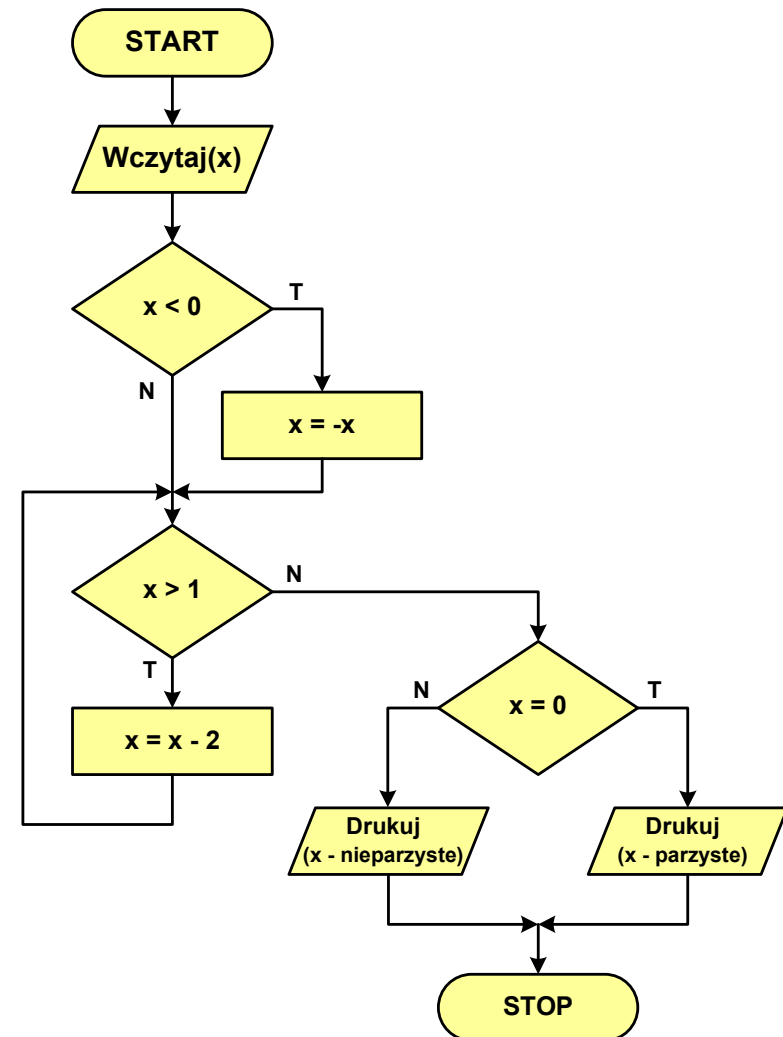
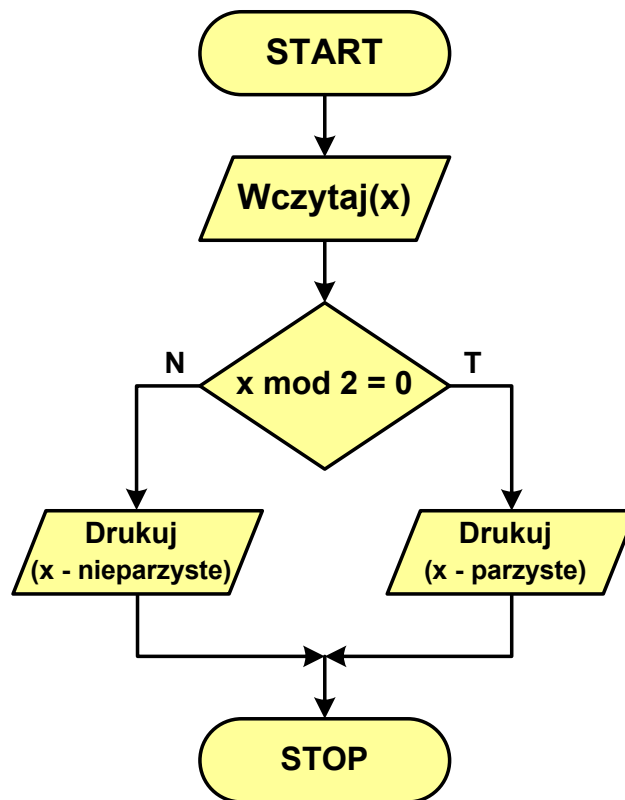
    printf("NWD = %d\n", NWD);
}
```

Wartość bezwzględna liczby - schemat blokowy

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



Parzystość liczby - schemat blokowy



Równanie kwadratowe - schemat blokowy

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

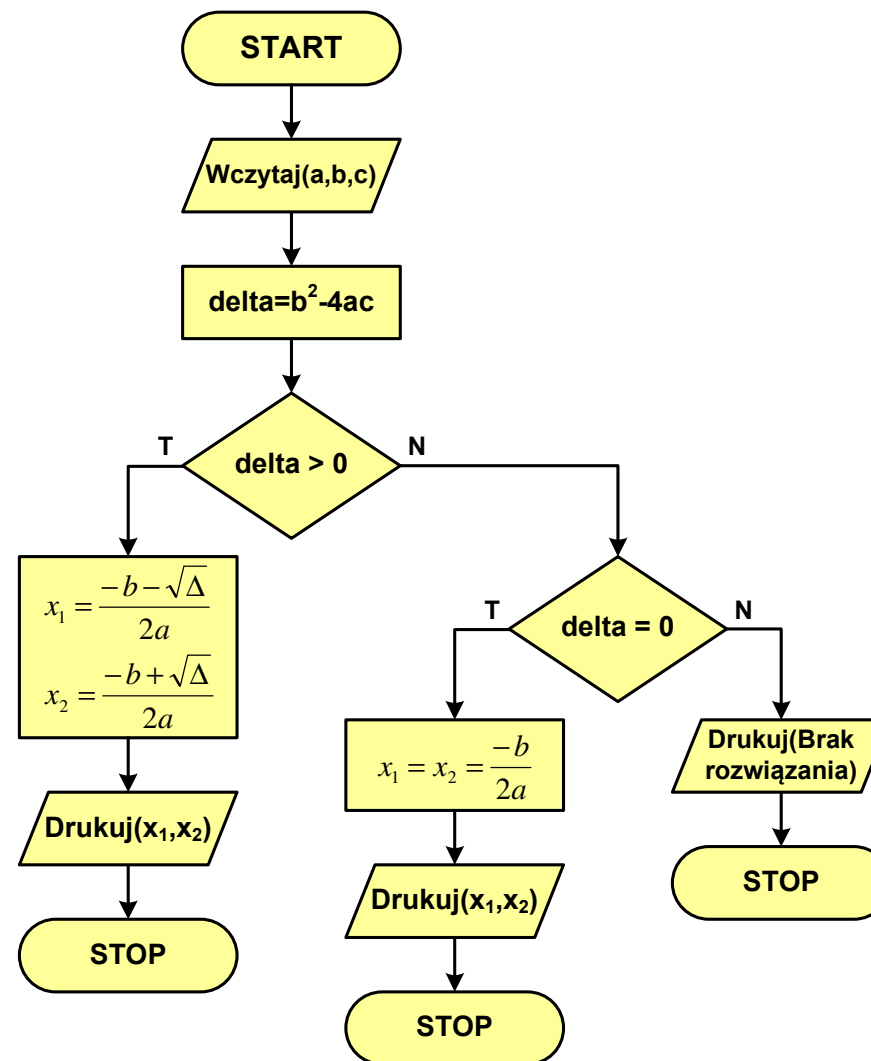
$$\Delta = b^2 - 4ac$$

$$\Delta > 0:$$

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

$$\Delta = 0:$$

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



Silnia - schemat blokowy

$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$$

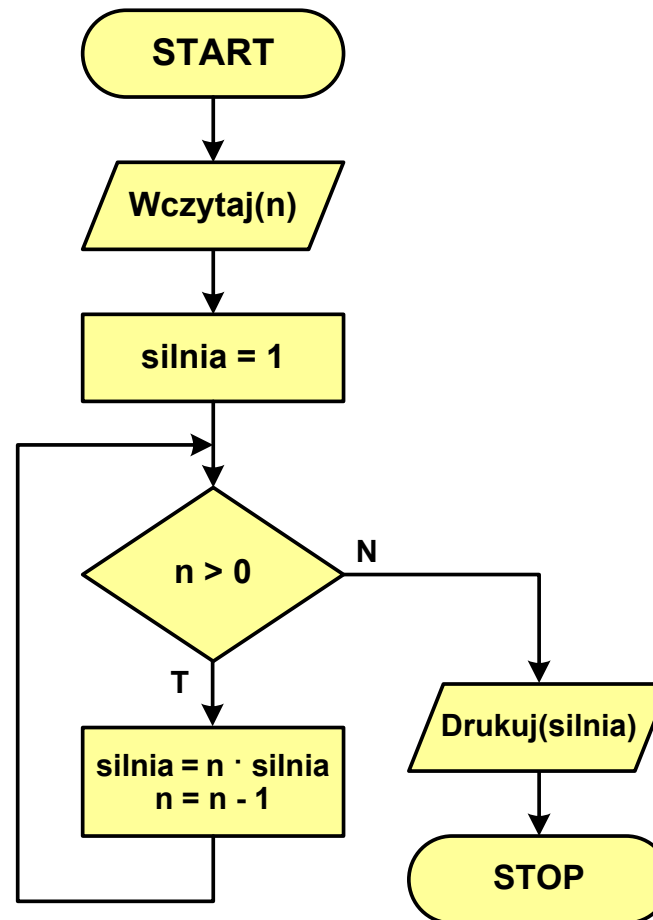
$$0! = 1$$

$$1! = 1$$

$$2! = 1 \cdot 2$$

$$3! = 1 \cdot 2 \cdot 3$$

...



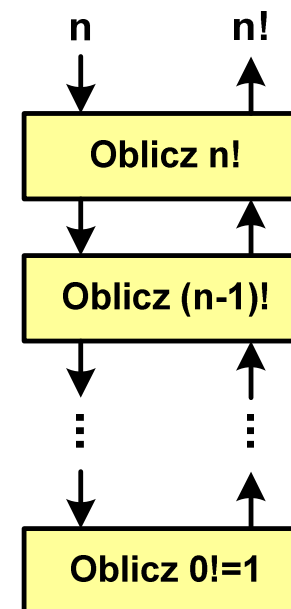
Rekurencja

- **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:

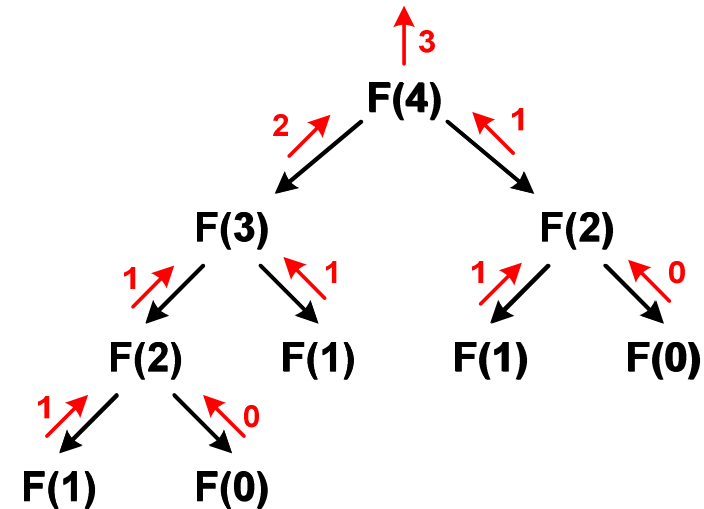
$$n! = \begin{cases} 1 & \text{dla } n = 0 \\ n(n-1)! & \text{dla } n \geq 1 \end{cases}$$

```
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;
    else
        if (n==1) return 1;
        else
            return F(n-1) + F(n-2);
}
```

Rekurencja - algorytm Euklidesa

$$NWD(a,b) = \begin{cases} a & \text{dla } b = 0 \\ NWD(b, a \bmod b) & \text{dla } b \geq 1 \end{cases}$$

```
int NWD(int a, int b)
{
    if (b==0)
        return a;
    else
        return NWD(b, a % b);
}
```

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)**
- Złożoność obliczeniowa algorytmu to ilość zasobów potrzebnych do jego działania (czas, pamięć)

Złożoność obliczeniowa

Złożoność czasowa

- Czas wykonania algorytmu wyrażony w jednostkach czasu, liczbie cykli procesora, **liczbie wykonywanych operacji**
- Jej miarą jest zazwyczaj liczba podstawowych operacji (dominujących) - pozostałe operacje są pomijane
- Podstawowe operacje - porównanie, podstawienie, operacja arytmetyczna

Złożoność pamięciowa

- Jest miarą wykorzystania pamięci (liczba komórek pamięci)
- Wyrażana jest w liczbie bajtów lub liczbie zmiennych określonego typu

Złożoność obliczeniowa

- Jeśli wykonanie algorytmu zależne jest od zestawu danych wejściowych, to wyróżnia się:

Złożoność optymistyczna

- Odpowiada danym najbardziej sprzyjającym dla algorytmu

Złożoność średnia

- Złożoność uśredniona po wszystkich możliwych zestawach danych, występująca dla „typowych” (losowych) danych wejściowych

Złożoność pesymistyczna

- Odpowiada danym najbardziej niesprzyjającym dla algorytmu
- Przykład: poszukiwanie określonej wartości w N-elementowej tablicy liczb

Złożoność obliczeniowa

- Złożoność obliczeniowa algorytmu jest **funkcją** rozmiaru danych, na których pracuje ten algorytm
- Złożoność obliczeniowa wyznaczana jest poprzez zliczanie operacji
- W praktyce stosuje się oszacowanie powyższej funkcji - są to tzw. notacje (klasy złożoności):
 - O (duże O)
 - Ω (omega)
 - Θ (theta)

Notacja O („duże O ”)

- Wyraża złożoność matematyczną algorytmu
- Po literze O występuje wyrażenie w nawiasach zawierające literę n , która oznacza liczbę elementów, na której działa algorytm
- Do wyznaczenia złożoności bierze się pod uwagę liczbę wykonywanych w algorytmie podstawowych operacji

Przykład:

- $O(n)$ - złożoność algorytmu jest prostą funkcją liczby elementów (jeśli sumowanie 10.000 elementów zajmuje 5 s, to sumowanie 20.000 elementów zajmie 10 s)
- $O(n^2)$ - czas konieczny do wykonania algorytmu rośnie wraz z kwadratem liczby elementów (przy podwojeniu liczby elementów ich obsługa będzie trwała cztery razy dłużej)

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

Sortowanie

Po co stosować sortowanie?

- Posortowane elementy można szybciej zlokalizować
- Posortowane elementy można przedstawić w czytelniejszy sposób

Klasyfikacje algorytmów sortowania

- **Złożoność obliczeniowa algorytmu** - zależność liczby wykonywanych operacji od liczebności sortowanego zbioru n
- **Złożoność pamięciowa** - wielkość zasobów zajmowanych przez algorytm (sortowanie **w miejscu** - wielkość zbioru danych podczas sortowania nie zmienia się lub jest tylko nieco większa)
- **Sortowanie wewnętrzne** (odbywa się w pamięci komputera) i **zewnętrzne** (nie jest możliwe jednoczesne umieszczenie wszystkich elementów zbioru sortowanego w pamięci komputera)

Klasyfikacje algorytmów sortowania

- Algorytm jest **stabilny**, jeśli podczas sortowania zachowuje kolejność występowania elementów o tym samym kluczu

Przykład:

- Tablica nieposortowana (imię, nazwisko, wiek):

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

- Tablica posortowana algorytmem stabilnym (klucz - wiek):

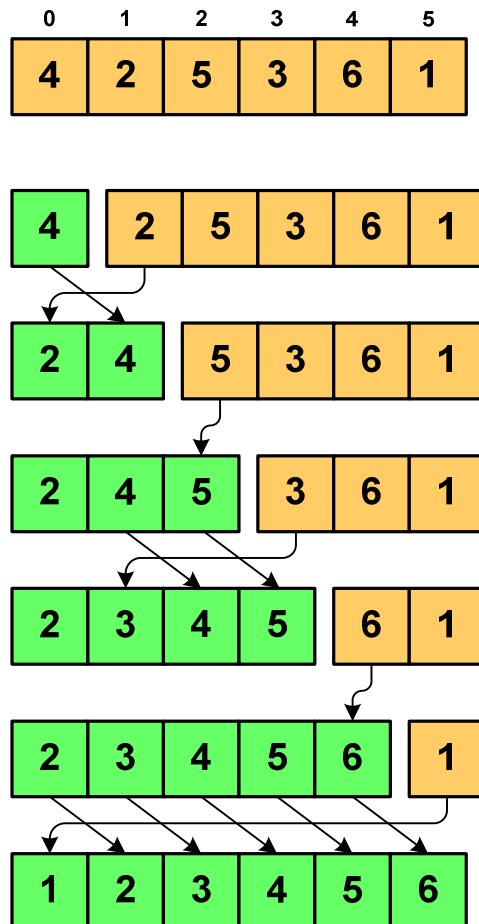
Magda	Ola	Ela	Piotr	Jan	Paweł
Mazur	Nowak	Król	Kowalski	Kamiński	Wójcik
15	18	18	20	20	23

- Tablica posortowana algorytmem niestabilnym (klucz - wiek):

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

Proste wstawianie (insertion sort)

Przykład:



Program w języku C:

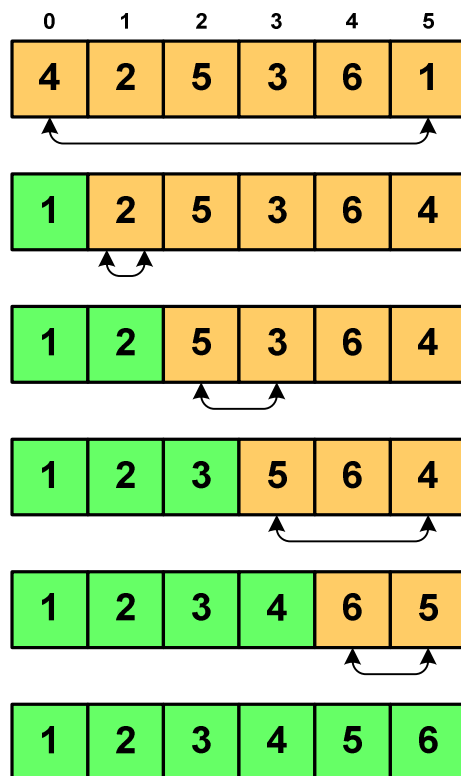
```
int main(void)
{
    int tab[N], i, j, tmp;
    // ...
    for (i=1; i<N; i++)
    {
        j=i;
        tmp=tab[i];
        while (tab[j-1]>tmp && j>0)
        {
            tab[j]=tab[j-1];
            j--;
        }
        tab[j]=tmp;
    }
}
```

Proste wstawianie (insertion sort)

- Złożoność algorytmu: $O(n^2)$
 - + wydajny dla danych wstępnie posortowanych
 - + wydajny dla zbiorów o niewielkiej liczebności
 - + małe zasoby zajmowane podczas pracy (sortowanie w miejscu)
 - + stabilny
 - + prosty w implementacji
- mała efektywność dla normalnej i dużej ilości danych.

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;
    }
}
```


Proste wybieranie (selection sort)

- Złożoność algorytmu: $O(n^2)$
 - + szybki w sortowaniu niewielkich tablic
 - + małe zasoby zajmowane podczas pracy (sortowanie w miejscu)
 - + prosty w implementacji
- liczba porównań elementów jest niezależna od początkowego rozmieszczenia elementów w tablicy
- w algorytmie może zdarzyć się, że wykonywana jest zamiana tego samego elementu ze sobą.

Bąbelkowe (bubble sort)

- **Sortowanie bąbelkowe** (ang. bubble sort), nazywane jest także:
 - sortowaniem pęcherzykowym
 - sortowaniem przez prostą zamianę (ang. straight exchange)
- Metoda ta polega na porównywaniu dwóch kolejnych elementów i zamianie ich kolejności jeśli jest to konieczne
- Nazwa metody wzięła się stąd, że kolejne porównania powodują „wypychanie” kolejnego największego elementu na koniec („wypłynięcie największego bąbelka”)

Bąbelkowe (bubble sort)

- Złożoność algorytmu: $O(n^2)$
 - + prosta realizacja
 - + wysoka efektywność użycia pamięci (sortowanie w miejscu)
 - + stabilny
 - mała efektywność.

Sortowanie szybkie (Quick-Sort) - faza dzielenia

- Tablica jest dzielona na dwie części wokół pewnego elementu x (nazywanego elementem centralnym)
- Jako element centralny x najczęściej wybierany jest element środkowy (choć może to być także element losowy)
- Przeglądamy tablicę od lewej strony, aż znajdziemy element $a_i \geq x$, a następnie przeglądamy tablicę od prawej strony, aż znajdziemy element $a_j \leq x$
- Zamieniamy elementy a_i i a_j miejscami i kontynuujemy proces przeglądania i zamiany, aż nastąpi spotkanie w środku tablicy
- W ten sposób otrzymujemy tablicę podzieloną na lewą część z wartościami mniejszymi lub równymi x i na prawą część z wartościami większymi lub równymi x

Sortowanie szybkie (Quick-Sort) - faza sortowania

- Zawiera dwa rekurencyjne wywołania tej samej funkcji sortowania: dla lewej i dla prawej części posortowanej tablicy
- Rekurencja zatrzymuje się, gdy wielkość tablicy wynosi 1

Przykład:

- Sortujemy 6-elementową tablicę **tab**:

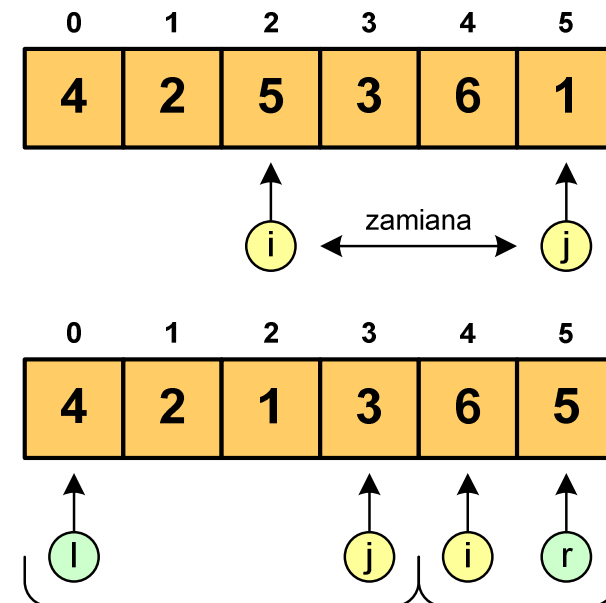
	0	1	2	3	4	5
tab	4	2	5	3	6	1

- Wywołanie funkcji **QS()** ma postać:

QS (tab, 0, 5) ;

Sortowanie szybkie (Quick-Sort) - $QS(tab, 0, 5)$

- Element środkowy: $(0+5)/2 = 2$, $x = tab[2] = 5$
- Od lewej szukamy $tab[i] \geq x$,
a od prawej szukamy $tab[j] \leq x$,
zamieniamy elementy miejscami
- Poszukiwania kończymy,
gdy indeksy i, j mijają się



- Wywołujemy rekurencyjnie funkcję $QS()$ dla elementów z zakresów $[l, j]$ i $[i, r]$:

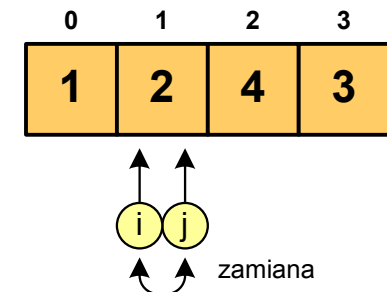
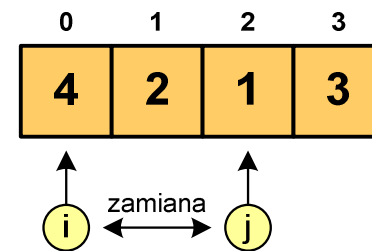
$QS(tab, 0, 3);$

$QS(tab, 4, 5);$

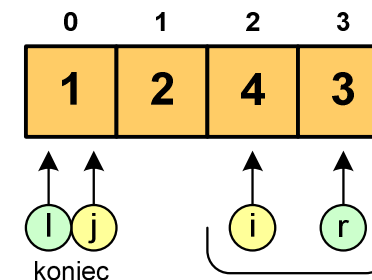
Sortowanie szybkie (Quick-Sort) - QS(tab,0,3)

- Element środkowy: $(0+3)/2 = 1$, $x = \text{tab}[1] = 2$

- Od lewej szukamy $\text{tab}[i] \geq x$,
a od prawej szukamy $\text{tab}[j] \leq x$,
zamieniamy elementy miejscami



- Poszukiwania kończymy,
gdy indeksy i, j mijają się



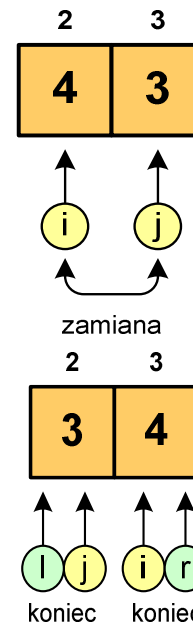
- Wywołanie QS() tylko dla elementów z zakresu [2,3], gdyż po lewej stronie rozmiar tablicy do posortowania wynosi 1:

QS (tab, 2, 3) ;

Sortowanie szybkie (Quick-Sort) - QS(tab,2,3)

- Element środkowy: $(2+3)/2 = 2$, $x = \text{tab}[2] = 4$

- Od lewej szukamy $\text{tab}[i] \geq x$,
a od prawej szukamy $\text{tab}[j] \leq x$,
zamieniamy elementy miejscami



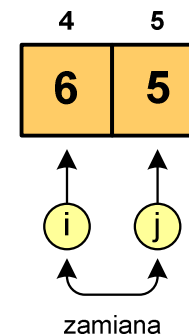
- Poszukiwania kończymy,
gdy indeksy i, j mijają się

- Rozmiar obu tablic do posortowania wynosi 1 więc nie ma nowych wywołań funkcji QS()

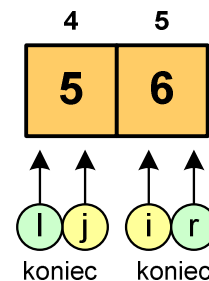
Sortowanie szybkie (Quick-Sort) - QS(tab,4,5)

- Element środkowy: $(4+5)/2 = 4$, $x = \text{tab}[4] = 6$

- Od lewej szukamy $\text{tab}[i] \geq x$,
a od prawej szukamy $\text{tab}[j] \leq x$,
zamieniamy elementy miejscami



- Poszukiwania kończymy,
gdy indeksy i, j mijają się



- Rozmiar obu tablic do posortowania wynosi 1 więc nie ma nowych wywołań funkcji QS()

Sortowanie szybkie (Quick-Sort)

Funkcja w języku C:

```
void QuickSort(int tab[], int l, int r)
{
    int i,j,x,y;

    i=l;
    j=r;
    x=tab[(l+r)/2];
    do
    {
        while (tab[i]<x) i++;
        while (x<tab[j]) j--;
        if (i<=j)
        {
            y=tab[i];
            tab[i]=tab[j];
            tab[j]=y;
            i++; j--;
        }
    } while (i<=j);
    if (l<j) QuickSort(tab,l,j);
    if (i<r) QuickSort(tab,i,r);
}
```

Koniec wykładu nr 5

Dziękuję za uwagę!
(następny wykład: 11.06.2018)

Zaliczenie wykładu!