

Informatyka 1

Politechnika Białostocka - Wydział Elektryczny
Elektrotechnika, semestr II, studia niestacjonarne I stopnia
Rok akademicki 2018/2019

Wykład nr 5 (05.04.2019)

dr inż. Jarosław Forenc

Plan wykładu nr 5

- Język C
 - pętla for, operatory ++ i --
- Klasyfikacja systemów komputerowych (Flynna)
- Architektura von Neumanna i architektura harwardzka
- Budowa komputera
 - jednostka centralna
 - płyta główna
 - procesor (mikroarchitektury)

Język C - 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

Język C - 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);

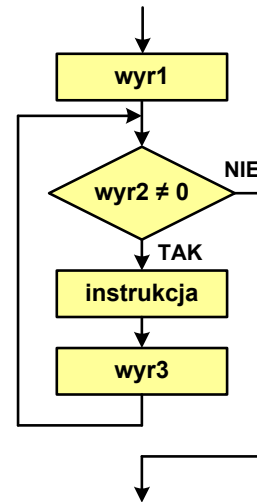
    return 0;
}
```

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



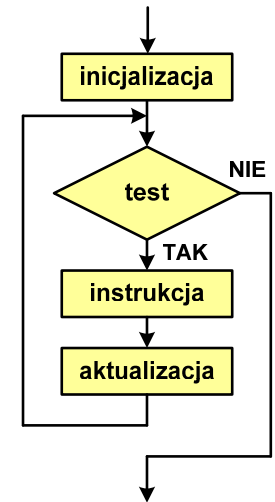
Język C - pętla for

- Najczęściej stosowana postać pętli **for**

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

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

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



Język C - pętla for (wyświetlenie tekstu)

```
#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 (suma liczb: 1 + 2 + ... + N)

```
#include <stdio.h>  
#define N 1234  
  
int main(void)  
{  
    int i, suma=0;  
  
    for (i=1; i<=N; i++)  
        suma = suma + i;  
  
    printf("Suma %d liczb to %d\n", N, suma);  
  
    return 0;  
}
```

```
Suma 1234 liczb to 761995
```

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);  
}
```

□ `continue` przerywa bieżącą iterację i przechodzi do obliczania `wyr3`

□ `break` przerywa wykonywanie pętli

1 3 5

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!

error C2143: syntax error : missing ';' before ')'

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 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 - 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**

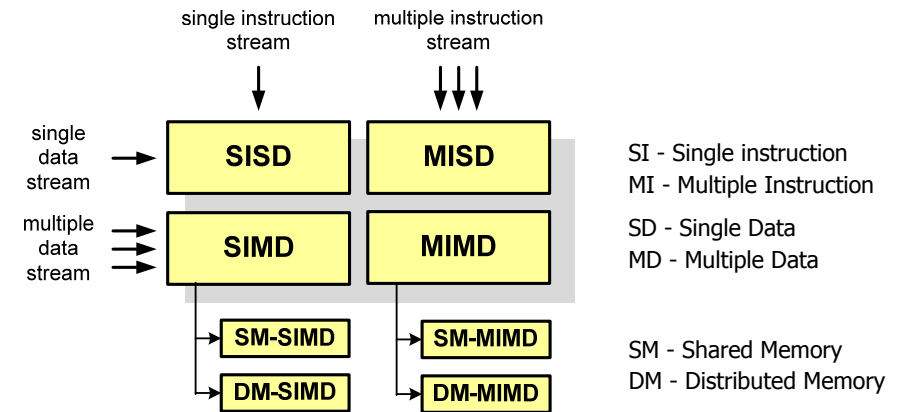
Język C - priorytet operatorów ++ i --

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

Klasyfikacja systemów komputerowych

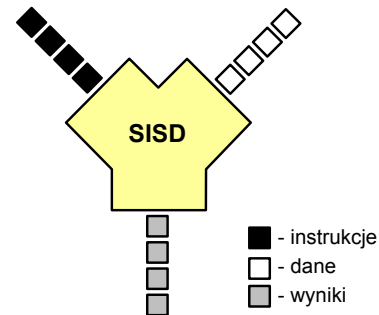
- **Taksonomia Flynna** - pierwsza, najbardziej ogólna klasyfikacja architektur komputerowych (1972):
 - Flynn M.J.: „Some Computer Organizations and Their Effectiveness”, IEEE Transactions on Computers, Vol. C-21, No 9, 1972.
- Opiera się na liczbie przetwarzanych strumieni rozkazów i strumieni danych:
 - **strumień rozkazów** (Instruction Stream) - odpowiednik licznika rozkazów; system złożony z n procesorów posiada n liczników rozkazów, a więc n strumieni rozkazów
 - **strumień danych** (Data Stream) - zbiór operandów, np. system rejestrujący temperaturę mierzoną przez n czujników posiada n strumieni danych

Taksonomia Flynna



SISD (Single Instruction, Single Data)

- Jeden wykonywany program przetwarza jeden strumień danych
- Klasyczne komputery zbudowane według **architektury von Neumanna**
- Zawierają:
 - jeden procesor
 - jeden blok pamięci operacyjnej zawierający wykonywany program.

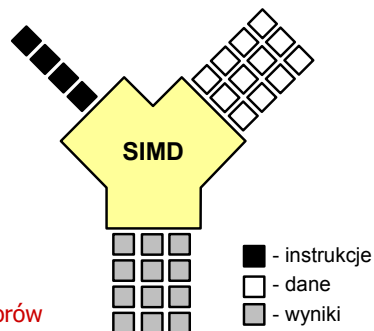


SISD (Single Instruction, Single Data)



SIMD (Single Instruction, Multiple Data)

- Jeden wykonywany program przetwarza wiele strumieni danych
- Te same operacje wykonywane są na różnych danych
- Podział:
 - SM-SIMD (Shared Memory SIMD):
 - komputery wektorowe
 - rozszerzenia strumieniowe procesorów (MMX, 3DNow!, SSE, SSE2, SSE3, AVX, ...)
 - DM-SIMD (Distributed Memory SIMD):
 - tablice procesorów
 - procesory kart graficznych (GPGPU)



SM-SIMD - Komputery wektorowe

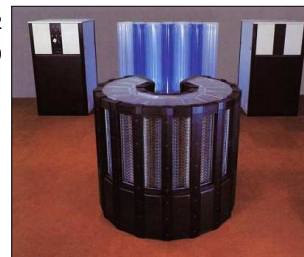
CDC
Cyber 205
(1981)



Cray-1
(1976)



Cray-2
(1985)



Hitachi
S3600
(1994)

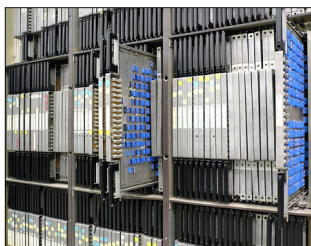


DM-SIMD - Tablice procesorów

Illiack IV
(1976)



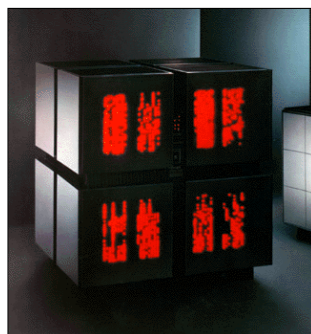
Illiack IV
(1976)



MasPar
MP-1/MP-2
(1990)



Thinking
Machines
CM-2
(1987)



DM-SIMD - Procesory graficzne (GPU)

GeForce
GTX Titan X



Tesla
V100



DGX-1
Volta

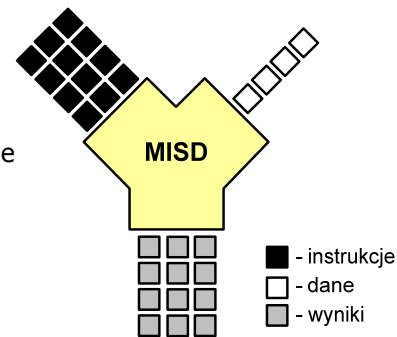


Tesla
D870



MISD (Multiple Instruction, Single Data)

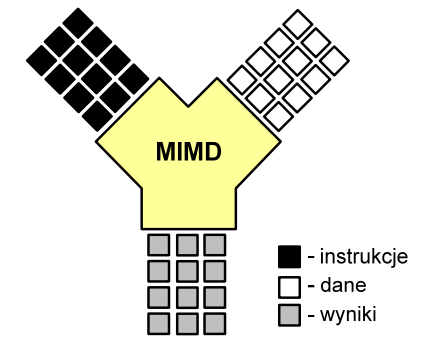
- Wiele równoległe wykonywanych programów przetwarza jednocześnie jeden wspólny strumień danych
- Systemy tego typu nie są spotykane



MIMD (Multiple Instruction, Multiple Data)

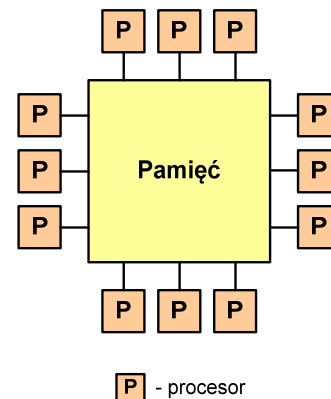
- Równoległe wykonywanych jest wiele programów, z których każdy przetwarza własne strumienie danych

- Podział:
 - SM-MIMD (Shared Memory):
 - wieloprocesory
 - DM-MIMD (Distributed Memory):
 - wielokomputery
 - klastry
 - gridy



SM-MIMD - Wieloprocesory

- Systemy z niezbyt dużą liczbą działających niezależnie procesorów
- Każdy procesor ma dostęp do wspólnej przestrzeni adresowej pamięci
- Komunikacja procesorów poprzez uzgodniony obszar wspólnej pamięci
- Do SM-MIMD należą komputery z **procesorami wielordzeniowymi**
- Podział:
 - UMA (Uniform Memory Access)
 - NUMA (NonUniform Memory Access)
 - COMA (Cache Only Memory Architecture)



SM-MIMD - Wieloprocesory

Cray YM-P
(1988)



Cray CS6400
(1993)

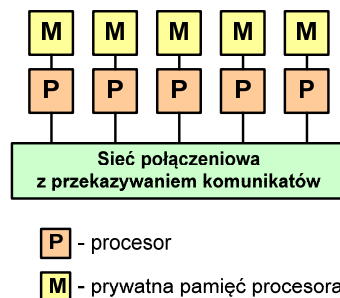


Cray J90
(1994)



DM-MIMD - Wielokomputery

- Każdy procesor wyposażony jest we własną pamięć operacyjną, niedostępną dla innych procesorów
- Komunikacja między procesorami odbywa się za pomocą sieci poprzez przesyłanie komunikatów
- Biblioteki komunikacyjne:
 - MPI (Message Passing Interface)
 - PVM (Parallel Virtual Machine)



DM-MIMD - Wielokomputery

Cray T3E
(1995)



Thinking
Machines
CM-5
(1991)

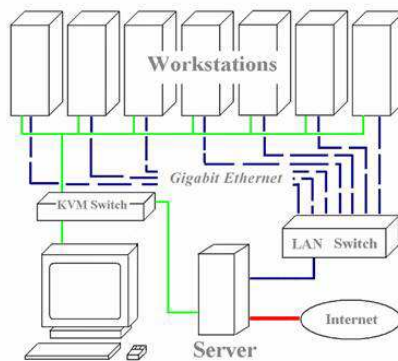
nCube 2s
(1993)



Meiko
CS-2
(1993)

DM-MIMD - Klastry

- **Klaster** (cluster):
 - równoległy lub rozproszony system składający się z komputerów
 - komputery połączone są siecią
 - używany jest jako pojedynczy, zintegrowany zespół obliczeniowy
- **Węzeł** (node) - pojedynczy komputer przyłączony do klastra i wykonujący zadania obliczeniowe



źródło:
http://leda.elfak.ni.ac.rs/projects/SeeGrid/see_grid.htm

KVM - Keyboard, Video, Mouse

DM-MIMD - Klastry

- Klastry Beowulf budowane były ze zwykłych komputerów PC



Odin II Beowulf Cluster Layout, University of Chicago, USA

DM-MIMD - Klastry

- Klastry Beowulf budowane były ze zwykłych komputerów PC



NASA 128-processor Beowulf cluster: A cluster built from 64 ordinary PC's

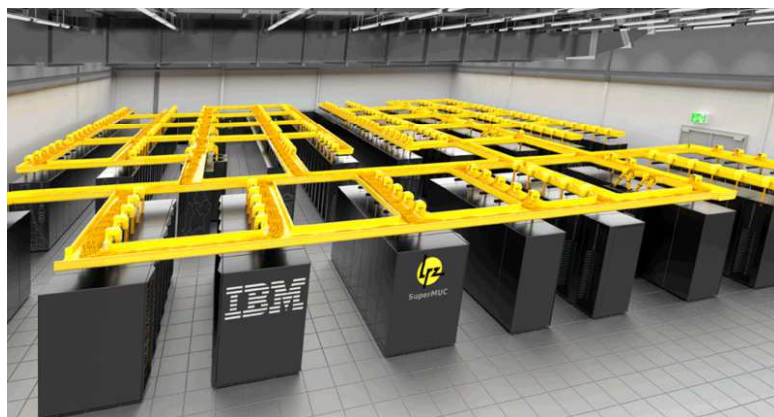
DM-MIMD - Klastry



Early Aspen Systems Beowulf Cluster With RAID

DM-MIMD - Klastry

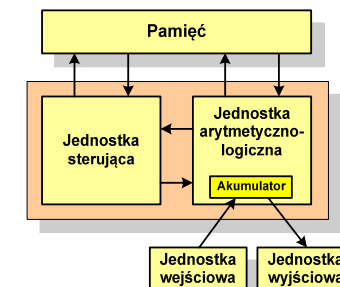
- Obecnie klastry też są bardzo popularnym typem systemów



SuperMUC-NG, Leibniz Rechenzentrum, Germany

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

- Architektura komputera, w której pamięć danych jest oddzielona od pamięci instrukcji
- Nazwa architektury pochodzi komputera Harward Mark I:
 - zaprojektowany przez Howarda Aikena
 - pamięć instrukcji - taśma dziurkowana, pamięć danych - elektromechaniczne liczniki

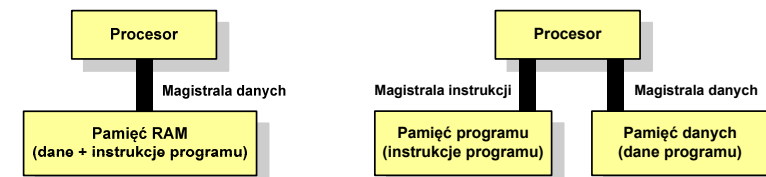


Architektura harwardzka

- Pamięci danych i instrukcji mogą różnić się:
 - technologią wykonania
 - strukturą adresowania
 - długością słowa
- Przykład:
 - ATmega16 - 16 kB Flash, 1 kB SRAM, 512 B EEPROM
- Procesor może w tym samym czasie czytać instrukcje oraz uzyskiwać dostęp do danych

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

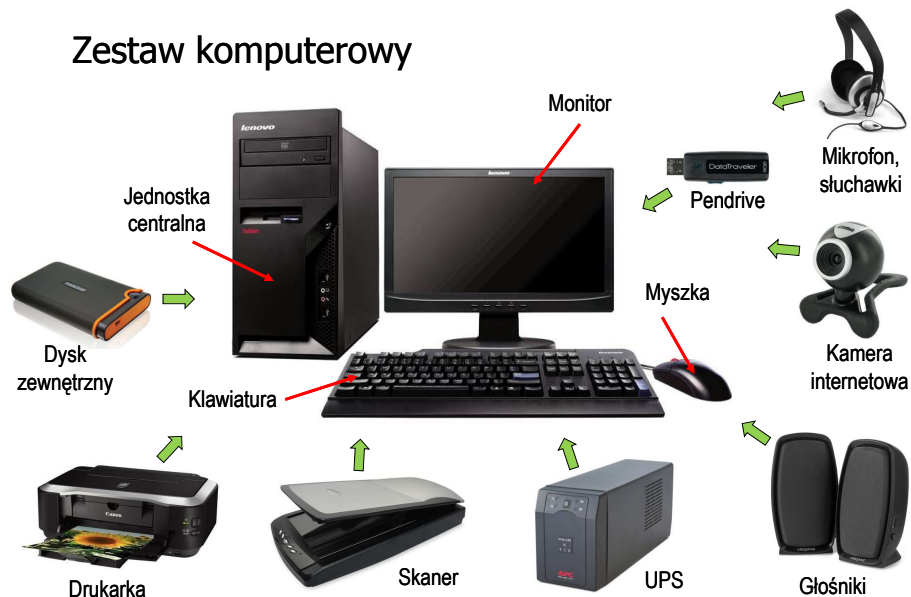


Architektura von Neumanna

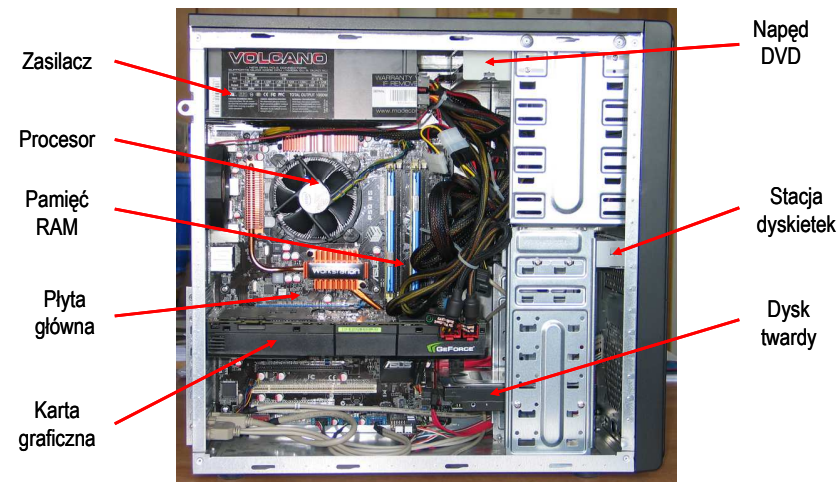
Architektura harwardzka

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

Zestaw komputerowy



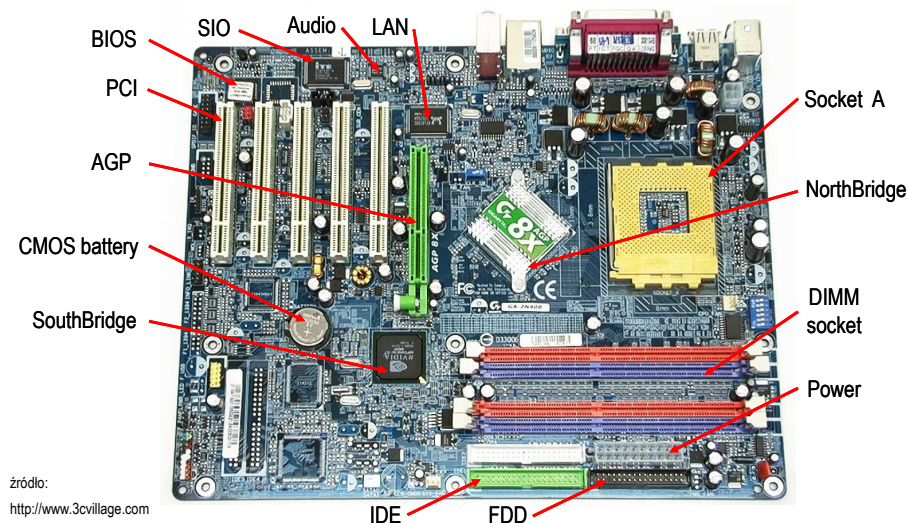
Jednostka centralna



Płyta główna (motherboard) - przykłady

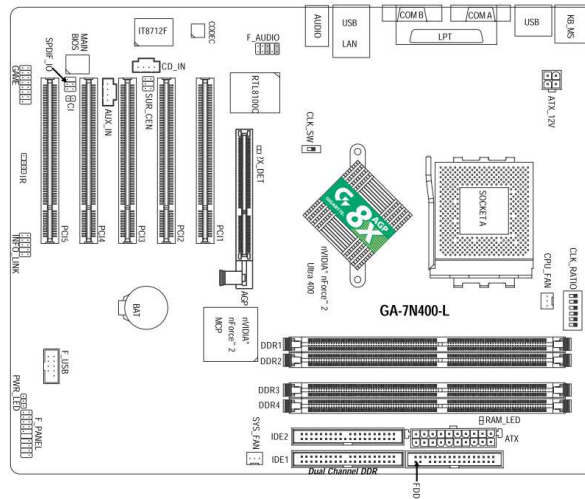
Model	Gigabyte GA-7N400-L	Gigabyte GA-X58A-UD5	Gigabyte G1-Assassin 2
Rok	2003	2009	2011
Gniazdo	Socket A	Socket 1366	Socket 2011
Procesor	AMD Athlon, Athlon XP	Intel Core i7	Intel Core i7
Northbridge	nVIDIA nForce 2 Ultra 400	Intel X58 Express Chipset	Intel X79
Southbridge	nVIDIA nForce 2 MCP	Intel ICH10R	
Pamięć	4 x 184-pin DDR DIMM sockets, max. 3 GB	6 x 1.5V DDR3 DIMM sockets, max. 24 GB	4 x 1.5V DDR3 DIMM sockets, max. 32 GB
Format	ATX	ATX	ATX
Inne	AGP, 5 x PCI, 2 x IDE, FDD, LPT, 2 x COM, 6 x USB, IrDA, RJ45, 2 x PS/2	4 x PCIe x16, 2 x PCIe x1, PCI, 8 x SATA II 3 Gb/s, 2 x SATA II 6 Gb/s, 2 x eSATA, IDE, FDD, 2 x RJ45, 10 x USB 2.0, 2 x USB 3.0, 2 x PS/2	3 x PCIe x16, 2 x PCIe x1, PCI, 4 x SATA II 3 Gb/s, 4 x SATA III 6 Gb/s, 2 x eSATA, RJ45, 9 x USB 2.0, 3 x USB 3.0, PS/2

Gigabyte GA-7N400-L



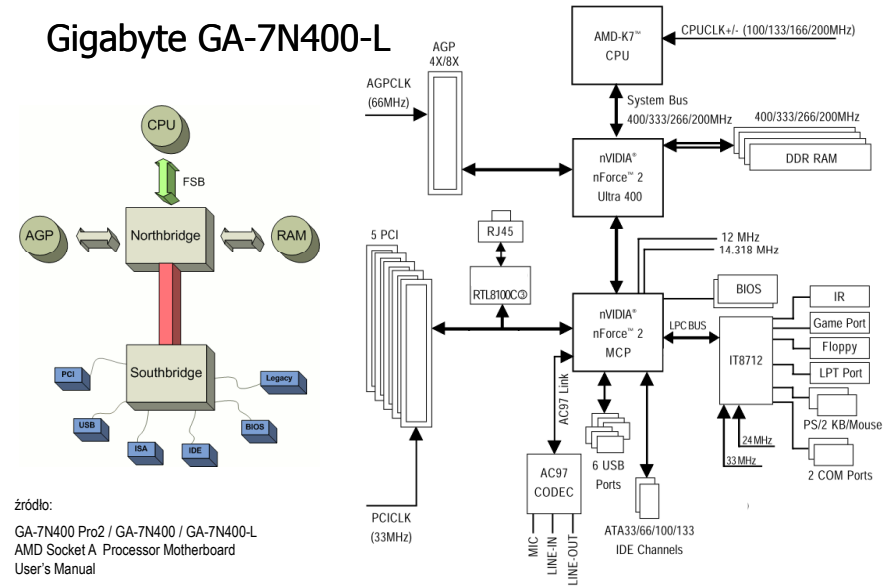
źródło:
<http://www.3cvillage.com>

Gigabyte GA-7N400-L



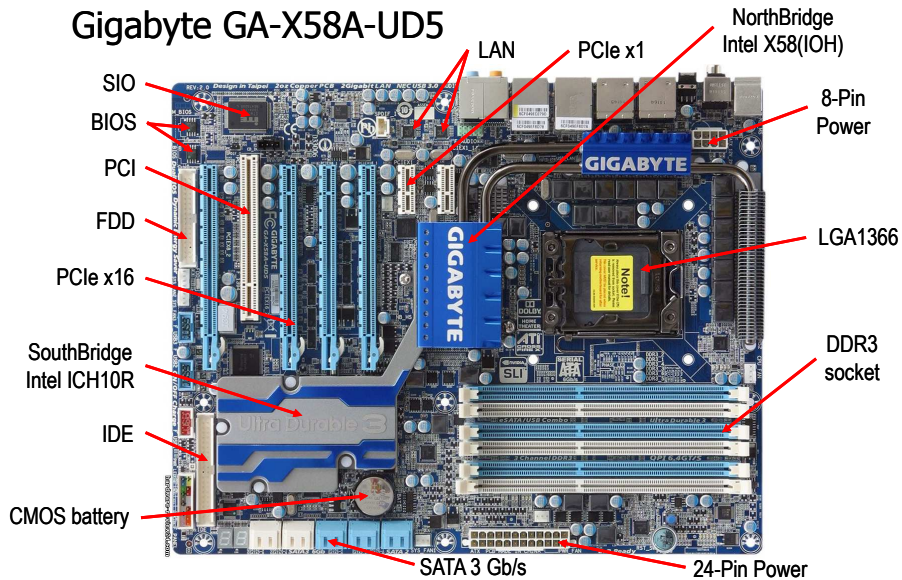
źródło:
GA-7N400 Pro2 / GA-7N400 / GA-7N400-L
AMD Socket A
Processor Motherboard
User's Manual

Gigabyte GA-7N400-L

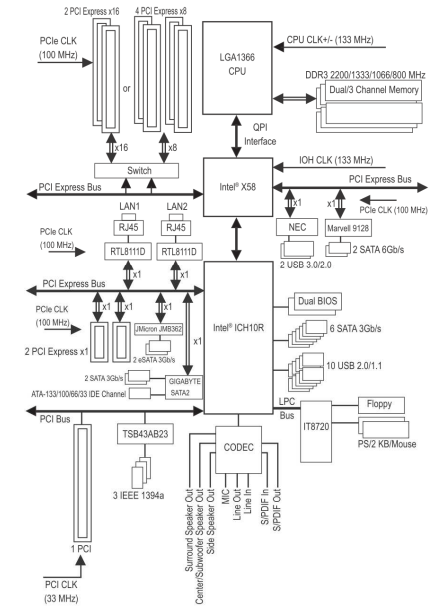


źródło:
GA-7N400 Pro2 / GA-7N400 / GA-7N400-L
AMD Socket A Processor Motherboard
User's Manual

Gigabyte GA-X58A-UD5

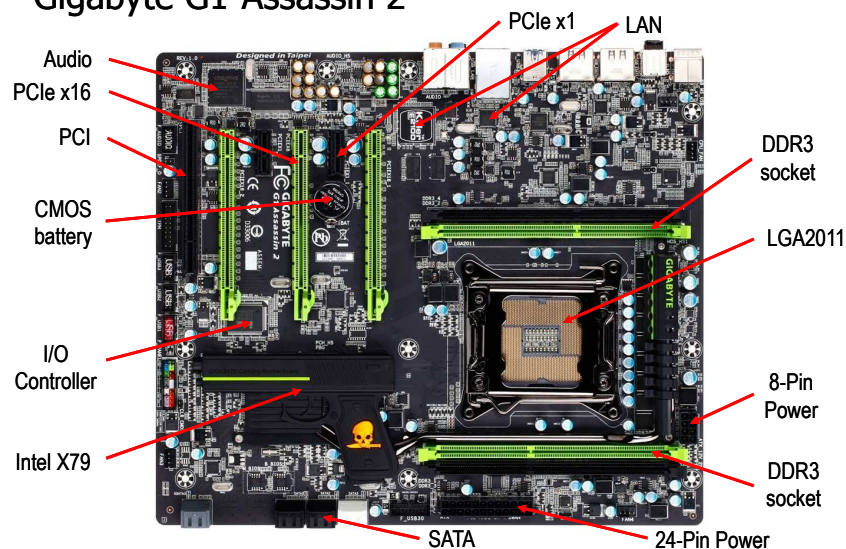


Gigabyte GA-X58A-UD5

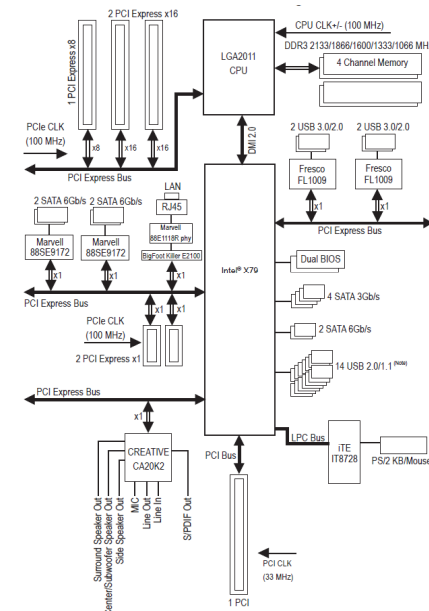


źródło:
GA-X58A-UD5
LGA1366 socket motherboard for Intel® Core™ i7 processor family
User's Manual

Gigabyte G1-Assassin 2



Gigabyte G1-Assassin 2



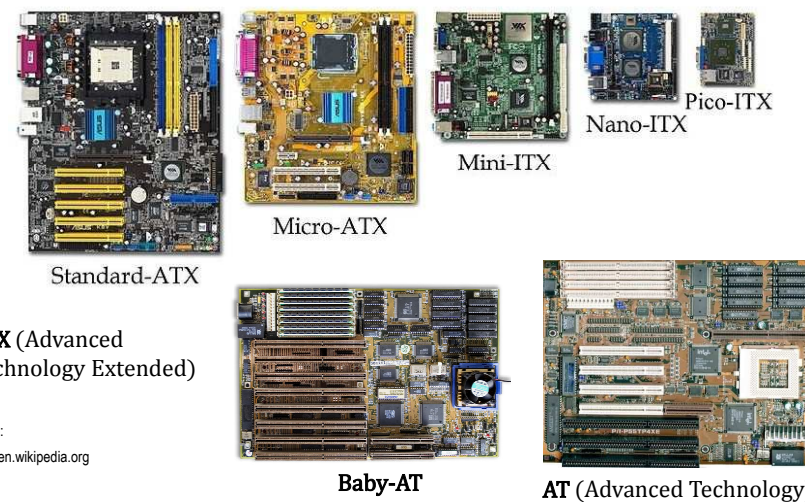
źródło:
Gigabyte G1.Assassin 2, User's Manual, Rev. 1001

Płyty główne - standardy

Standard	Rok	Wymiary
AT	1984 (IBM)	12 × 11–13 in 305 × 279–330 mm
Baby-AT	1985 (IBM)	8.5 × 10–13 in 216 × 254–330 mm
ATX	1996 (Intel)	12 × 9.6 in 305 × 244 mm
Micro-ATX	1996	9.6 × 9.6 in 244 × 244 mm
Mini-ITX	2001 (VIA)	6.7 × 6.7 in 170 × 170 mm max.
Nano-ITX	2003 (VIA)	4.7 × 4.7 in 120 × 120 mm
Pico-ITX	2007 (VIA)	100 × 72 mm max.

źródło: <http://en.wikipedia.org>

Płyty główne - standardy



ATX (Advanced Technology Extended)

źródło:
<http://en.wikipedia.org>

Procesory Intel - mikroarchitektury

■ Mikroarchitektura - organizacja procesora

Proces	Mikroarchitektura	Nazwa kodowa	Data	Procesory
45 nm	Nehalem	Nehalem	2008-11-17	Bloomfield, Lynnfield, Clarkfield
32 nm		Westmere	2010-01-04	Westmere-EX, -EP, Gulftown, Clarkdale
22 nm	Sandy Bridge	Sandy Bridge	2011-01-09	Sandy Bridge-EP, -E, -M, Sandy Bridge
	Ivy Bridge	Ivy Bridge	2012-04-29	Ivy Bridge-EX, -EP, -E, -M, Ivy Bridge
14 nm	Haswell	Haswell	2013-06-02	Haswell-EX, -EP, -E, -DT, -MB, -LP
		Broadwell	2014-09-05	Broadwell-EX, -EP, -E
	Skylake	Skylake	2015-08-05	Skylake-EX, -EP
		Kaby Lake	2016-10	KabyLake-X
		Coffee Lake	2017-10-05	CoffeLake-S
		Whiskey Lake Amber Lake	2018-08-28	Whiskey Lake-U AmberLake-Y
		Cascade Lake	2019-04-02	Cascade Lake-X
Cannonlake	2018-05	Cannon Lake-U		
10 nm	Icelake	Icelake	2019 / 2020	
Tigerlake				

Procesory Intel - mikroarchitektury

■ Mikroarchitektura - organizacja procesora

Proces	Mikroarchitektura	Nazwa kodowa	Nazwa marketingowa
45 nm	Nehalem	Nehalem	Core i3, i5, i7, Pentium, Celeron, Xeon
32 nm		Westmere	
22 nm	Sandy Bridge	Sandy Bridge	Core i3, i5, i7 (2 gen.), Pentium, Celeron, Xeon
	Ivy Bridge	Ivy Bridge	Core i3, i5, i7 (3 gen.), Pentium, Celeron, Xeon
14 nm	Haswell	Haswell	Core i3, i5, i7 (4 gen.), Pentium, Celeron, Xeon
		Broadwell	Core i3, i5, i7 (5 gen.), Core M, Pentium, Celeron, Xeon
	Skylake	Skylake	Core i3, i5, i7 (6 gen.), Core M, Pentium, Celeron, Xeon
		Kaby Lake	Core i3, i5, i7 (7/8 gen.), Celeron, Pentium, Xeon
		Coffee Lake	Core i3, i5, i7, i9 (8/9 gen.), Celeron, Pentium Gold
		Whiskey Lake Amber Lake	Core i3, i5, i7 (8 gen.)
		Cascade Lake	Core i7, i9, Xeon
Cannonlake			
10 nm	Icelake	Icelake	
Tigerlake			

Koniec wykładu nr 5

Dziękuję za uwagę!
(następny wykład: 12.04.2019)