Współczesne systemy komputerowe

dr inż. Sławomir Koczubiej

Politechnika Świętokrzyska
Wydział Zarządzania i Modelowania Komputerowego
Katedra Technologii Informatycznych

(10 października 2023)

Kontakt

Budynek C, pokój 3.26
sk@tu.kielce.pl

Materiały do pobrania, aktualności, terminy zaliczeń

http://staff.tu.kielce.pl/sk

Organizacja wykładów

- Wykłady są nieobowiązkowe (zgodnie z postanowieniami regulaminu), ale...
- Na wykłady czasem warto zajrzeć.

Warunki zaliczenia wykładu

Egzamin zaliczeniowy po zakończeniu wykładów.

Organizacja laboratoriów

- Zajęcia laboratoryjne są obowiązkowe.
- Dopuszcza się jedną nieobecność.
- Większa liczba nieobecności powoduje zmniejszenie oceny do niedostatecznej włącznie (3 lub więcej nieobecności).
- W przypadku usprawiedliwionej nieobecności zajęcia można odrobić z inną grupą (jeśli istnieje taka możliwość).

Warunki zaliczenia laboratoriów

Wykonanie ćwiczeń i zaliczenie sprawdzianów kontrolnych.

Treść wykładów

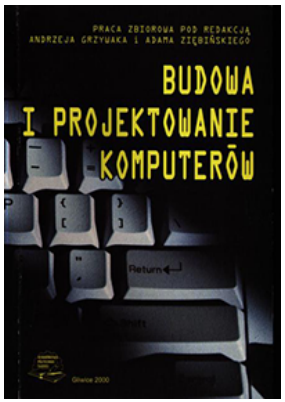
- Wstęp. Budowa i architektura komputera. Architektura i organizacja pamięci.
- Dane i ich reprezentacja. Model programowy i struktura użytkowa komputera.
- Zasoby komputera. Współczesna architektura komputera.
- System operacyjny, definicja, zadania, klasyfikacja.
- Budowa systemu operacyjnego, procesy. Systemy i typy plików. Operacje na plikach.
- Wirtualizacja. Cechy wybranych współczesnych systemów operacyjnych.

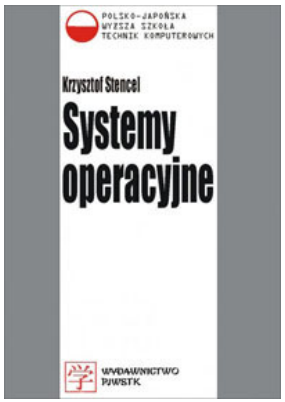
Treść laboratoriów

- Wirtualizacja. Instalacja systemu GNU/Linux.
- Wstępna konfiguracja systemu operacyjnego.
- Pliki, katalogi, prawa dostępu, wyszukiwanie plików.
- Instalacja oprogramowania. Archiwizacja.
- Zarządzanie użytkownikami i zasobami dyskowymi.
- Start systemu operacyjnego. Zarządzanie procesami i usługami.
- Monitorowanie systemu operacyjnego.

Literatura

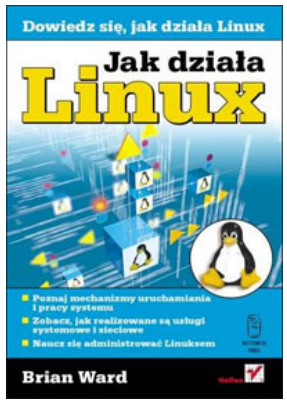
- Grzywak A. *Budowa i projektowanie komputerów.* Wydawnictwo Politechniki Śląskiej, Gliwice 2000.
- Stallings W. *Organizacja i architektura systemu komputerowego.* WNT, Warszawa 2004.
- Biernat J. *Architektura komputerów.* Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2005.
- Stencel K. *Systemy operacyjne.* Wydawnictwo PJWSTK, Warszawa 2004.
- Internet

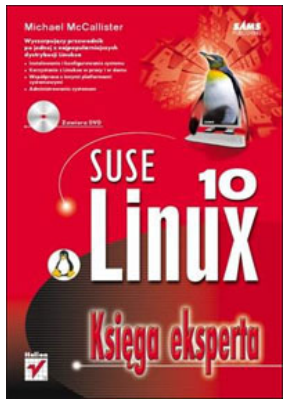


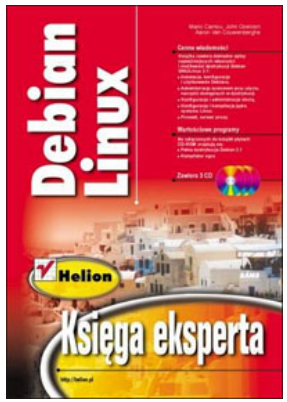


Literatura

- Negus C. *Linux. Biblia. Ubuntu, Fedora, Debian i 15 innych dystrybucji.* Wydawnictwo Helion, Gliwice 2011.
- Ward B. *Jak działa Linux.* Wydawnictwo Helion, Gliwice 2005.
- McCallister M. *SUSE Linux 10. Księga eksperta.* Wydawnictwo Helion, Gliwice 2006.
- Camou M., Goerzen J., Van Couwenberghe A. *Debian Linux. Księga eksperta.* Wydawnictwo Helion, Gliwice 2001.
- man

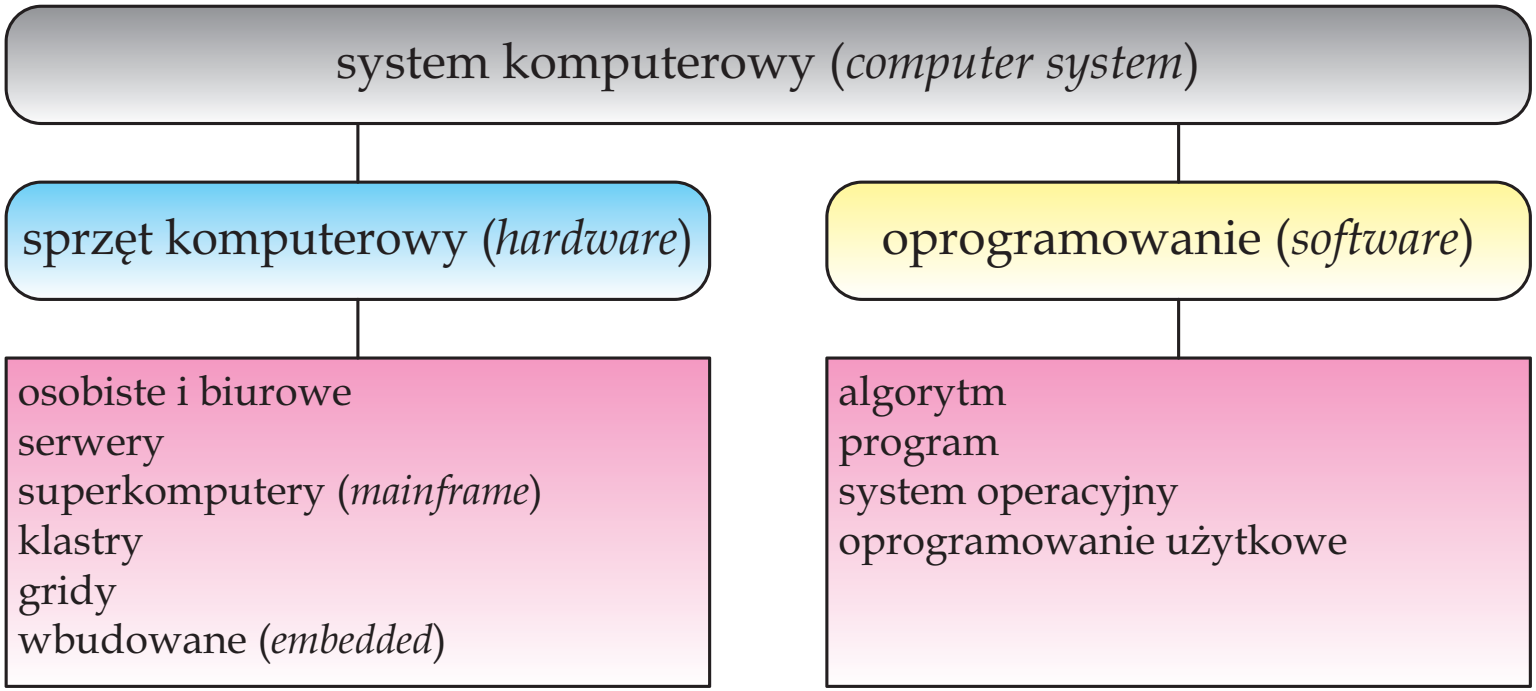






System komputerowy

Układ dwóch składowych: **sprzętu komputerowego** oraz **oprogramowania**. System komputerowy coraz częściej działa w sieci komputerowej. Można mówić o różnych poziomach takiego systemu: sprzęt komputerowy, system operacyjny (oprogramowanie systemowe), oprogramowanie użytkowe (aplikacje).



Sprzęt komputerowy – materialna część systemu komputerowego. Ogólnie hardware'em nazywa się sprzęt komputerowy jako taki i odróżnia się go od software'u – czyli oprogramowania.

Podział ten jest nieostry, gdyż współcześnie wiele elementów sprzętu komputerowego posiada *wszyte* weń na stałe oprogramowanie, stanowiące jego integralną część, bez którego elementy te nie mogłyby funkcjonować. Np. większość drukarek komputerowych posiada w swojej pamięci zestaw komend, przy pomocy których realizuje proces drukowania i których odpowiednik znajduje się w pamięci komputera stanowiąc programowy sterownik tego urządzenia. Wiele urządzeń – typu karty graficzne, płyty główne posiada własne oprogramowanie nazywane **BIOS**-em. W stosunku do oprogramowania niektórych urządzeń używa się słowa **firmware**.

Oprogramowanie – całość informacji w postaci zestawu instrukcji, zaimplementowanych interfejsów i zintegrowanych danych przeznaczonych dla komputera do realizacji wyznaczonych celów. Celem oprogramowania jest przetwarzanie danych w określonym przez twórcę zakresie.

Oprogramowanie tworzą programiści w procesie programowania. Oprogramowanie jako przejaw twórczości jest chronione prawem autorskim.

Oprogramowanie pisane jest zazwyczaj przy użyciu różnych języków programowania z wykorzystaniem algorytmów.

Często składowe systemu komputerowego (sprzęt i oprogramowanie) przedstawia się w postaci kilku warstw:

- komputer,
- oprogramowanie systemowe,
- oprogramowanie narzędziowe,
- oprogramowanie użytkowe,
- użytkownicy.

Komputer – zapewnia podstawowe możliwości obliczeniowe (procesor, pamięć, urządzenia wejścia/wyjścia), czyli są to podstawowe zasoby systemu komputerowego.

Oprogramowanie systemowe – kontroluje i koordynuje działanie zasobów sprzętowych przez zastosowanie różnych programów użytkowych dla różnych użytkowników.

Oprogramowanie narzędziowe – dogodne interfejsy użytkowe wspomagające zarządzanie zasobami sprzętowymi oraz usprawniające, modyfikujące oprogramowanie systemowe.

Oprogramowanie użytkowe – określają sposoby użycia zasobów systemowych do rozwiązywania problemów obliczeniowych zadanych przez użytkownika (kompilatory, systemy baz danych, gry, oprogramowanie biurowe), tworzone przez programistów.

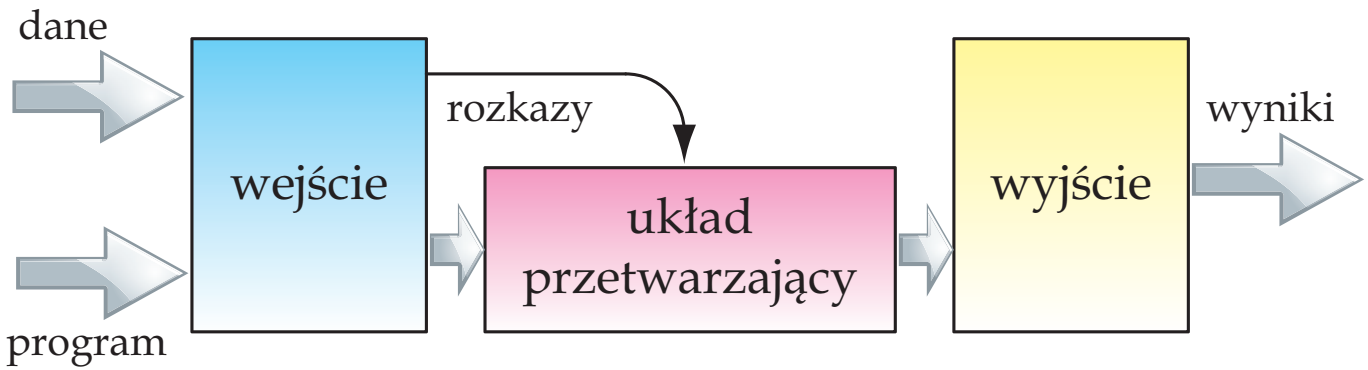
Użytkownicy – ludzie, maszyny, inne komputery, mający bezpośredni kontakt z oprogramowaniem użytkowym.

Komputer

Urządzenie do przetwarzanie danych, umożliwiające wprowadzanie, przechowywanie i wyprowadzanie danych. Charakteryzuje je zdolność wykonywania wielokrotnie, automatycznie powtarzanych obliczeń, według algorytmicznego wzorca zwanego programem.

Granica jest tu umowna, ponieważ taką definicję komputera spełniają też kalkulatory programowalne (naukowe, inżynierskie), jednak kalkulatory służą tylko do obliczeń numerycznych, podczas gdy nazwa komputer najczęściej dotyczy **urządzeń wielofunkcyjnych**.

Jakkolwiek istnieją mechaniczne urządzenia liczące, które potrafią realizować całkiem złożone programy, zazwyczaj nie zalicza się ich do komputerów. Warto jednak pamiętać, że prawzorem komputera jest abstrakcyjny model zwany **maszyną Turinga**, a pierwsze urządzenia ułatwiające obliczenia były znane w starożytności, np. **abakus** z 440 bc.



Architektura komputera

Sposób organizacji elementów tworzących komputer. Pojęcie to używane jest dosyć luźno. Może ono dzielić systemy komputerowe ze względu na wiele czynników, zazwyczaj jednak pod pojęciem architektury komputera rozumie się organizację połączeń pomiędzy pamięcią, procesorem i urządzeniami wejścia-wyjścia.

Innym, stosowanym potocznie znaczeniem terminu *architektura komputera* jest typ procesora wraz z zestawem jego instrukcji. Właściwszym określeniem w tym przypadku jest **model programowy procesora** (*ISA*, *Instruction Set Architecture*).

Model programowy procesora to określenie dotyczące organizacji, funkcjonalności i zasad działania procesora, widoczne z punktu widzenia programisty jako dostępne mechanizmy programowania.

Na model programowy procesora składają się m.in.:

- lista rozkazów procesora,
- obsługiwane typy danych,
- dostępne tryby adresowania,
- zestaw rejestrów dostępnych dla programisty,
- zasady obsługi wyjątków i przerwań.

Procesory posiadające ten sam model programowy są ze sobą **kompatybilne**, co oznacza, że mogą wykonywać te same programy i generować te same rezultaty. Przykładami modeli programowych:

- **IA-32**: i386, Pentium, K6, Athlon (RISC),
- **SPARC**: UltraSPARC, SPARC64,
- **AMD64**: Athlon 64 i wzwyż, Pentium 4 Prescott i wzwyż.

Taksonomie architektur

Taksonomia Flynna to klasyfikacja architektur komputerowych, zaproponowana w latach sześćdziesiątych XX wieku przez Michaela Flynna, opierająca się na liczbie przetwarzanych strumieni danych i strumieni rozkazów (ma raczej znaczenie historyczne).

	1 strumień danych	N strumieni danych
1 strumień instrukcji	SISD	SIMD
N strumieni instrukcji	MISD	MIMD

W taksonomii Flynna wyróżnia się cztery grupy:

- **SISD** (*Single Instruction, Single Data*) – przetwarzany jest jeden strumień danych przez jeden wykonywany program – komputery skalarne (sekwencyjne), jest to najbardziej rozpowszechniona architektura.
- **SIMD** (*Single Instruction, Multiple Data*) – przetwarzanych jest wiele strumieni danych przez jeden wykonywany program – komputery wektorowe.
- **MISD** (*Multiple Instruction, Single Data*) – wiele równolegle wykonywanych programów przetwarza jednocześnie jeden wspólny strumień danych; trudno wskazać wzorcowego reprezentanta tego typu, może być stosowana w systemach wykorzystujące redundancję (wielokrotne wykonywanie tych samych obliczeń) do minimalizacji błędów; można przyjąć, że założenia MISD w pewnym sensie realizują maszyny potokowe, np. procesory graficzne.
- **MIMD** (*Multiple Instruction, Multiple Data*) – równolegle wykonywanych jest wiele programów, z których każdy przetwarza własne strumienie danych, przykładem mogą być komputery wieloprocesorowe, a także klastry i gridy.

Tabelę Flynna można rozszerzyć o dodatkowy wiersz i kolumnę, odpowiadające zerowej liczbie strumieni instrukcji i danych.

Urządzenie bez strumieni danych nie jest komputerem – w tej części tabeli można by umieścić niektóre automaty.

Znacznie bardziej interesujący jest wiersz odpowiadający architekturom bez strumieni instrukcji. Same dane mogą nieść informacje o potrzebnym przetwarzaniu. Są to tzw. **komputery sterowane przepływem danych** (*dataflow*).

Innym rodzajem klasyfikacji jest **Taksonomia Skillicorna**, zaproponowana ok. 1988 roku. Zakłada ona, że każda architektura stanowi połączenie pewnej liczby abstrakcyjnych składników. W efekcie, taksonomia syntetyzuje architekturę, zamiast ją klasyfikować.

Klasyfikacja Skillicorna posługuje się elementami architektury:

- procesory instrukcji (sterujące) – **IP** (*Instruction Processor*),
- procesory danych – **DP** (*Data Processor*),
- hierarchia pamięci instrukcji – **IM** (*Instruction Memory Hierarchy*),
- hierarchia pamięci danych – **DM** (*Data Memory Hierarchy*),
- układy łączące powyższe elementy.

Zadania **procesora instrukcji**:

- wyznaczenie adresu następnej instrukcji do wykonania,
- sterowanie adresacją pamięci instrukcji,
- odczyt i dekodowanie instrukcji,
- sterowanie procesorem danych – wysłanie mu instrukcji do wykonania,
- wyznaczanie adresów operandów,
- odbieranie informacji stanu od procesorów danych.

Zadania **procesora danych**:

- odbiór od układu sterującego instrukcji i adresów operandów,
- odczyt operandów z pamięci danych,
- wykonanie instrukcji operandach,
- wyznaczenie informacji stanu dla układu sterującego,
- zapamietanie wyników obliczeń w pamięci danych.

Hierarchiczna pamięć instrukcji jest zbudowana podobnie jak **hierarchiczna** pamięć danych.

Hierarchiczna pamięć instrukcji lub danych zawiera:

- pamięć podręczną instrukcji/danych,
- pamięć operacyjną,
- pamięć dodatkową (np. dyskową).

Pamięć operacyjna oraz pamięć dodatkowa może być wspólna lub oddzielna dla danych i instrukcji. Tego problemu klasyfikacja Skillicorna nie rozstrzyga.

Układy łączące obejmują następujące typy:

- połączenie 1 do 1 – pojedyncze połączenie, (1 - 1),
- połączenie n do n – *n* połączeń pojedynczych równoległych (n - n),
- połączenie 1 do n – rozesłanie informacji do *n* odbiorców (1 - n),
- połączenie n na n – przełącznik krzyżowy łączący *n* wejść z *n* wyjściami (n$\times$n).

Same procesory danych nie zawierają żadnych elementów pamiętających.

W modelach architektur przyjmuje się, że liczba hierarchii pamięci jest równa liczbie procesorów danego typu. Oznacza to, że model architektury ze wspólną hierarchią pamięci dla kilku procesorów jest przedstawiany jako model z kilkoma hierarchiami pamięci i możliwością dostępu każdego procesora do każdej hierarchii pamięci.

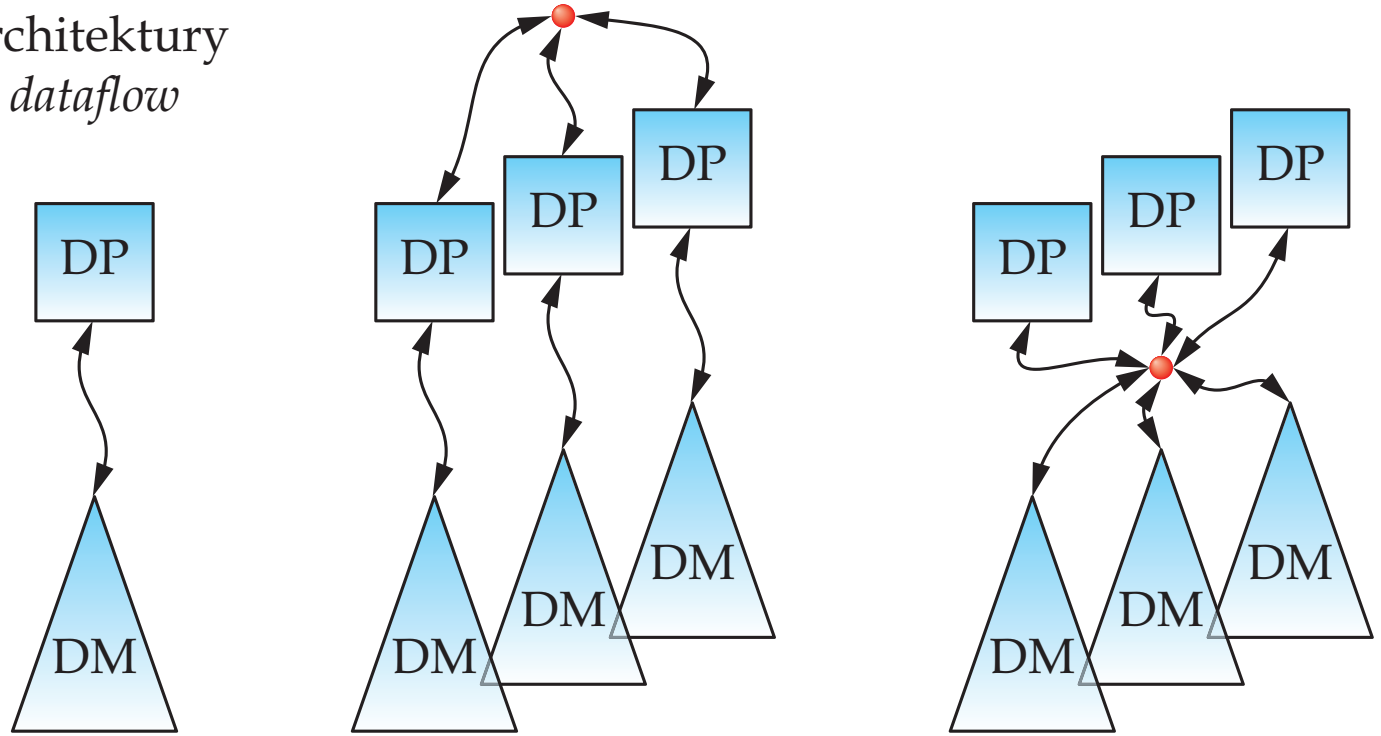
Komputer musi zawierać przynajmniej jeden procesor danych. Dozwolone są połączenia pomiędzy procesorami i hierarchiami pamięci tego samego rodzaju (kodu albo danych) oraz połączenia pomiędzy procesorami (tego samego lub różnych typów).

Używając taksonomii Skillcorna można zbudować około 30 różnych modeli architektur. Sześć z tych modeli ma sensowne znaczenie:

- uniprocesor dataflow (w taksonomii Flynna będzie to model bez strumienia instrukcji i jednym strumieniem danych, NISD),
- wieloprocesor dataflow (w taksonomii Flynna będzie to model bez strumienia instrukcji i N strumieniami danych, NIMD),
- uniprocesor von Neumanna (SISD),
- procesor wektorowy (SIMD),
- wieloprocesor słabo sprzężony (MIMD),
- wieloprocrsor silnie sprzężony (MIMD).

Sprzężenie pomiędzy procesorami tego samego typu ma sens tylko dla procesorów danych. Sprzężenie słabe będzie realizowane przez kanał komunikacyjny a silne przez wzajemny dostęp do hierarchii pamięci (praktycznie wspólna hierarchia pamięci).

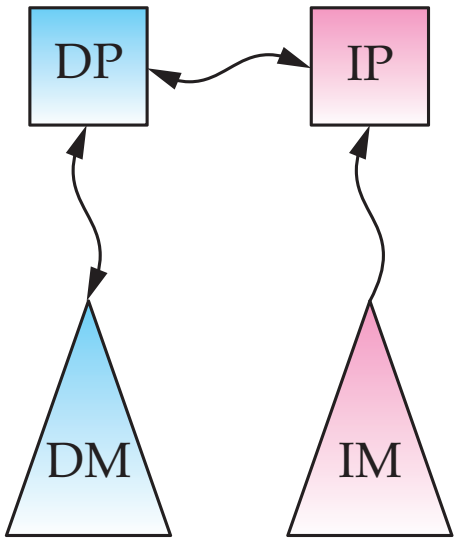
architektury
dataflow



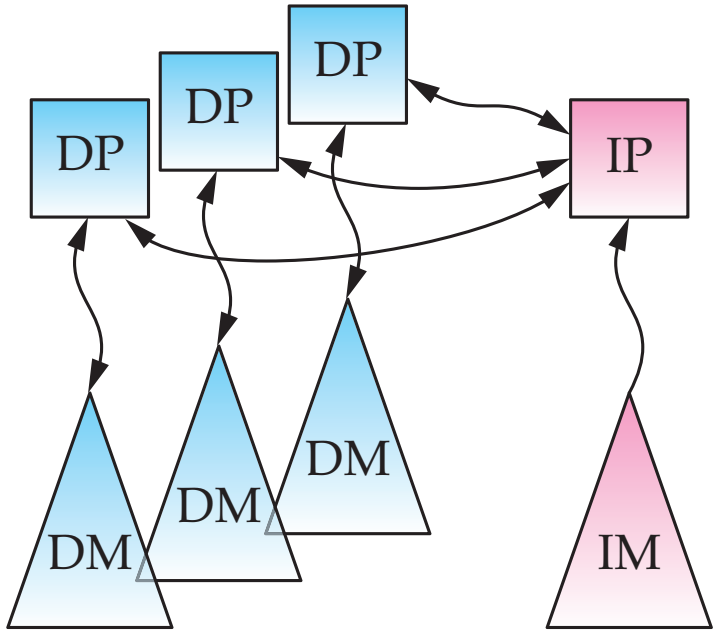
uniprocesor

wieloprocesor
słabo sprzężony

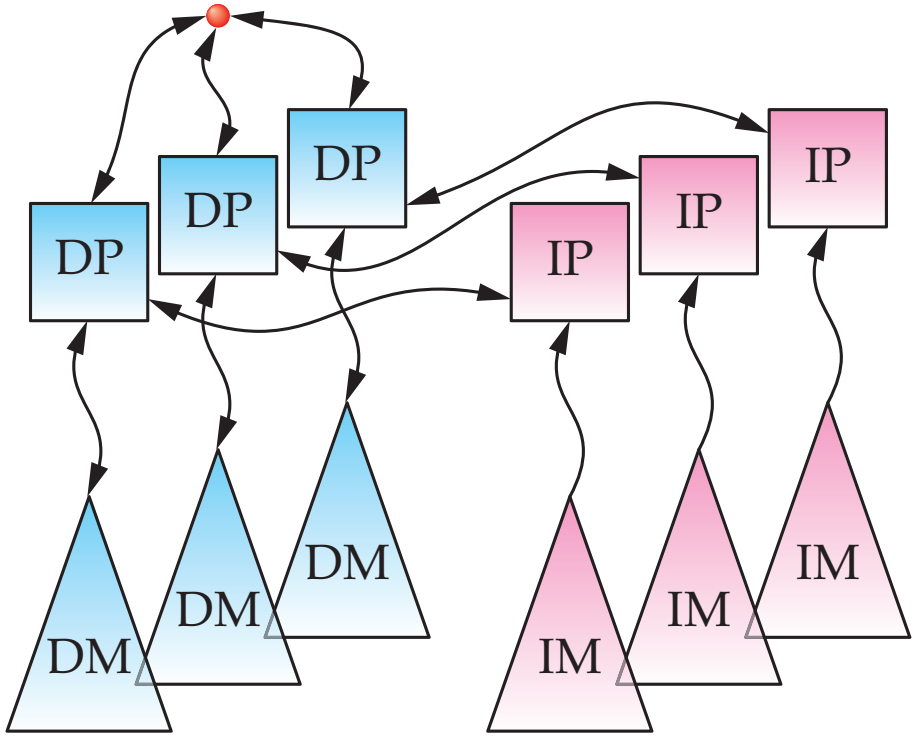
wieloprocesor
silnie sprzężony



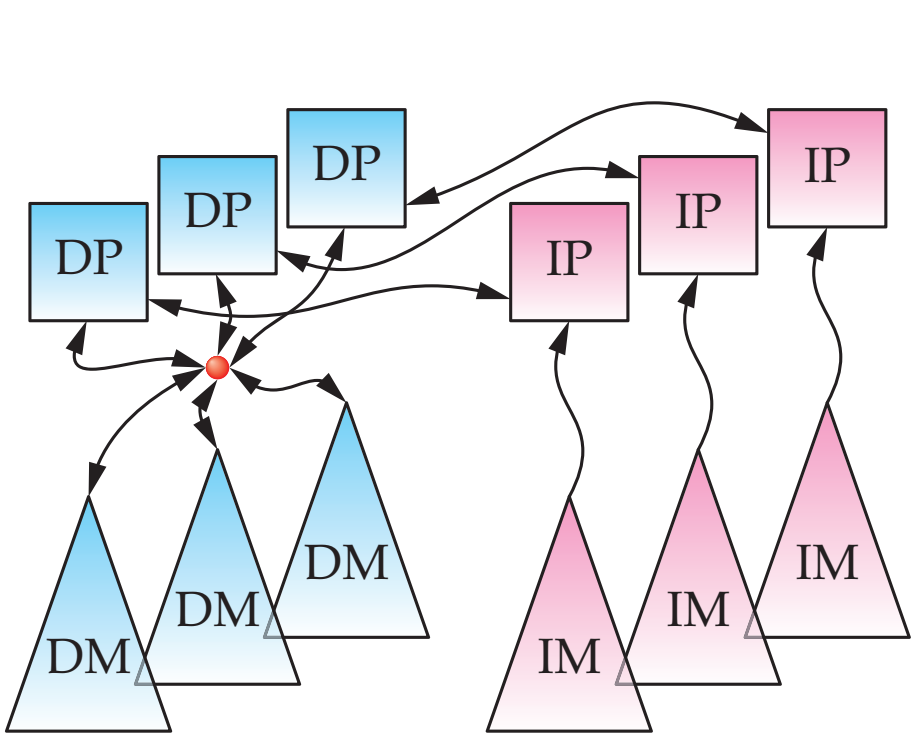
uniprocesor
von Neumanna



procesor
wektorowy



wieloprocesor
słabo sprzężony



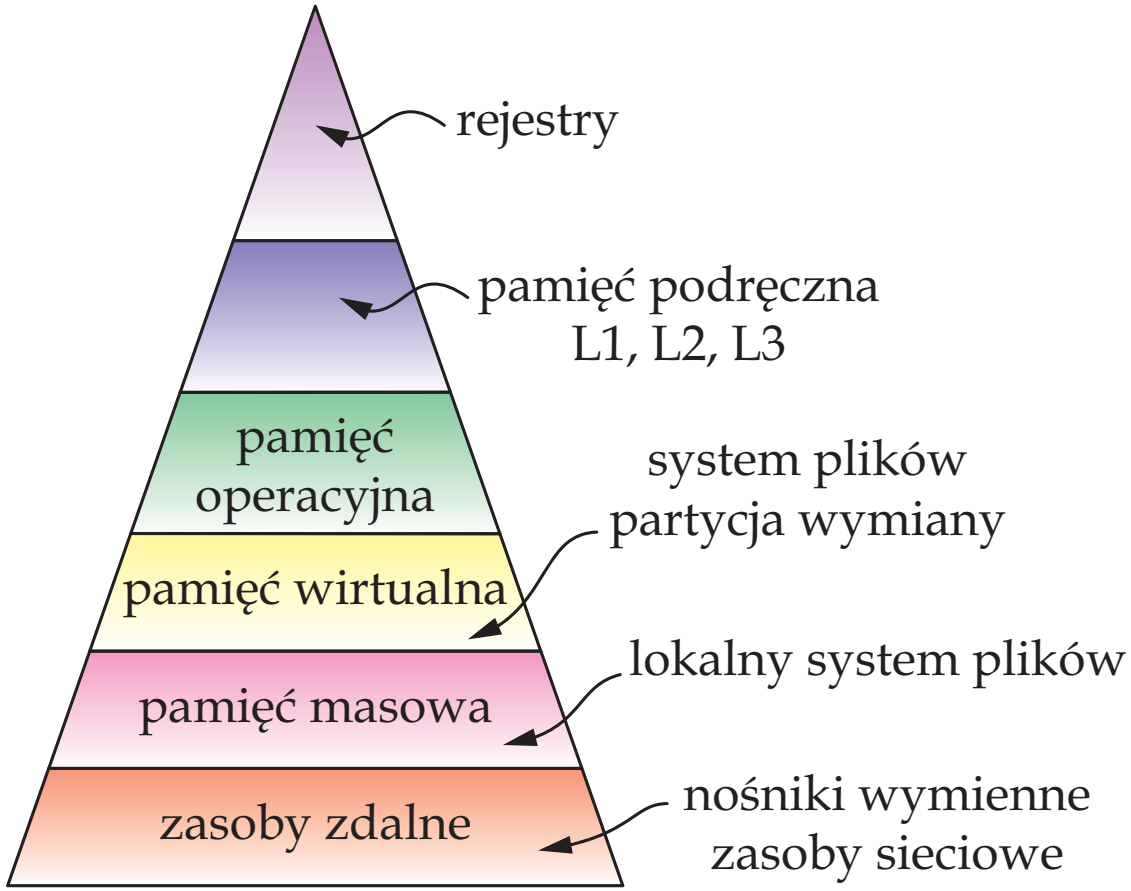
wieloprocesor
silnie sprzężony

Hierarchia pamięci

W taksonomia Skillicorna występuje pojęcie hierarchii pamięci (a nie pamięć). Słowo *hierarchia* dobrze oddaje budowę pamięci współczesnego komputera, w którym znajduje się kilka bloków funkcjonalnych służących do przechowywania programów i danych.

Idealny komputer powinien mieć jak największą i jak najszybszą pamięć. Pojemność pamięci wpływa na jej fizyczne rozmiary, a te – na czas dostępu. Nie można więc zbudować dowolnie dużej i jednocześnie szybkiej pamięci.

Problem ten rozwiązuje się przez wyodrębnienie wielu warstw o zróżnicowanej pojemności i szybkości, tworzących razem hierarchię pamięci. Kolejne warstwy w miarę oddalania się od procesora mają coraz większe pojemności i coraz dłuższe czasy dostępu.


rejestry
pamięć podręczna
L1, L2, L3
pamięć
operacyjna
system plików
partycja wymiany
pamięć wirtualna
lokalny system plików
pamięć masowa
nośniki wymienne
zasoby sieciowe
zasoby zdalne

Hierarchia pamięci współczesnego komputera, z punktu widzenia konstrukcji komputera, składa się z czterech warstw.

Rejestry fizycznie znajdują się wewnątrz procesora, dzięki czemu dostęp do nich jest bardzo szybki.

Pamięć podręczna (kieszeń, *cache*), wprowadzone po raz pierwszy około 1968 roku, zapewniają buforowanie danych pomiędzy procesorem i pamięcią operacyjną w celu przyspieszenie dostępu do pamięci.

Warstwa pamięci **wirtualnej**, powstała również około 1968 roku, zapewnia rozszerzenie pamięci operacyjnej.

Z punktu widzenia użytkownika do hierarchii pamięci należy zaliczyć wszelkie zasoby służące przechowywaniu danych. Logiczne staje się więc uzupełnienie rysunku o **lokalny system plików** komputera oraz o **zasoby zdalne**, w postaci nośników wymiennych i serwerów sieciowych.

Mechanizmy sterujące przemieszczaniem danych pomiędzy poszczególnymi warstwami są różne.

O umieszczeniu danych w rejestrach decyduje **programista** piszący program w asemblerze lub **kompilator** języka wysokiego poziomu.

Styk warstwy pamięci podręcznej i pamięci operacyjnej jest sterowany na **poziomie sprzętu**.

Stykiem pamięci operacyjnej i wirtualnej steruje **system operacyjny** przy użyciu jednostki zarządzania pamięcią.

O umieszczeniu danych w pamięci wirtualnej decyduje **użytkownik** – otwierając plik danych lub uruchamiając program.

Przemieszczaniem danych pomiędzy lokalnym systemem plików i nośnikami wymiennymi lub zasobami sieciowymi steruje **użytkownik**.

Tabela przedstawia orientacyjne parametry poszczególnych warstw hierarchii pamięci. Należy zwrócić uwagę na dużą różnicę czasów dostępu pamięci podręcznej i pamięci operacyjnej lub wirtualnej – czas podany dla pamięci dotyczy pojedynczego, losowego dostępu do pamięci dynamicznej typu DDR.

warstwa, pojemność, czas dostępu		
rejestry	< 1 kB	< 1 ns
pamięć podręczna L1	$\leqslant 128$ kB	≈ 1 ns
pamięć podręczna L2	$\leqslant 12$ MB	$1 \ldots 2$ ns
pamięć podręczna L3	$\leqslant 256$ MB	$2 \ldots 5$ ns
pamięć operacyjna	$\leqslant 64$ GB	$10 \ldots 50$ ns
pamięć wirtualna, system plików	$\geqslant 128$ GB	$\leqslant 10$ ms
nośniki wymienne, sieć komputerowa	∞	s, min.

Maszyna von Neumanna

Maszyna von Neumanna – model architektury komputera, opracowany przez Johna von Neumanna, Johna W. Mauchly'ego oraz Johna P. Eckerta w 1945 roku. Jej cechą charakterystyczną jest taki sam sposób przechowywania instrukcji i danych w hierarchii pamięci. Model maszyny von Neumanna wprowadza specyficzny mechanizm dostępu do pamięci – poprzez adres.

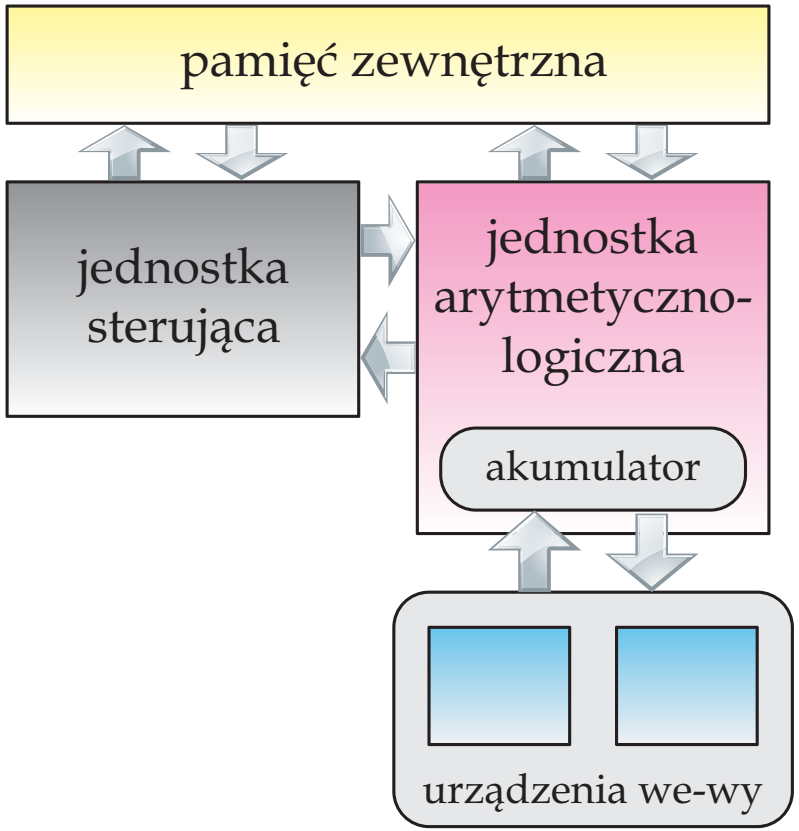
Maszyna von Neumanna następujące cechy:

- wykorzystuje model obliczeń zaproponowany przez Turinga, wykonuje obliczenia zgodnie z programem,
- program jest przechowywany w pamięci razem z danymi,
- pamięć składa się z pewnej liczby ponumerowanych komórek,
- dostęp do pamięci następuje poprzez podanie numeru komórki, czyli **adresu**,
- adres jest przechowywany i inkrementowany w specjalnym rejestrze procesora, zwanym **licznikiem instrukcji** (PC, *Program Counter*).

W architekturze tej komputer składa się z czterech głównych komponentów:

- pamięci komputerowej przechowującej dane programu oraz instrukcje programu, każda komórka pamięci ma unikatowy adres,
- jednostki sterującej odpowiedzialnej za pobieranie danych i instrukcji, pamięci oraz ich sekwencyjne przetwarzanie,
- jednostki arytmetyczno-logicznej odpowiedzialnej za wykonywanie podstawowych operacji arytmetycznych,
- rządzeń wejścia-wyjścia służących do interakcji z operatorem.

Jednostka sterująca wraz z jednostką arytmetyczno-logiczną tworzą procesor.


pamięć zewnętrzna
jednostka sterująca
jednostka arytmetyczno-logiczna
akumulator
urządzenia we-wy

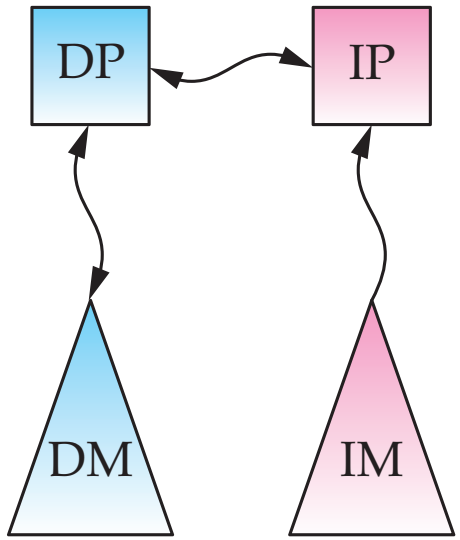
Dwa warianty architektury von Neumanna różnią, się sposobem przechowywania instrukcji i danych:

- architektura **Harvard** – oddzielne hierarchie pamięci danych i rozkazów,
- architektura **Princeton** – wspólna hierarchia pamięci danych i rozkazów.

Architektura Harvard jest niekiedy uważana za architekturę nie spełniającą postulatów von Neumanna wobec faktu oddzielnego przechowywania instrukcji i danych.

Cechy charakterystyczne architektury Harvard:

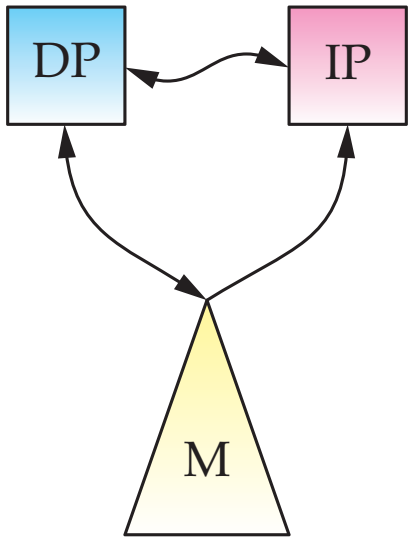
- oddzielne hierarchie pamięci danych i instrukcji,
- wysoka wydajność dzięki możliwości równoczesnego pobierania instrukcji i wykonywania operacji na hierarchii pamięci,
- brak możliwości programowania – nie ma możliwości zapisu instrukcji do pamięci instrukcji,
- komputer jest dostarczany ze stałym programem,
- jest wykorzystywana w procesorach sygnałowych oraz mikrokomputerach jednoukładowych (zastosowania wbudowane).



Uwaga! Rysunek posługuje się symbolami zapożyczonymi z taksonomii Skillicorna w sposób sprzeczny z zasadami budowy modeli wprowadzonymi przez tę taksonomię.

Cechy charakterystyczne architektury Princeton:

- *wzorcowa* realizacja maszyny von Neumanna ze wspólną hierarchią pamięci instrukcji i danych,
- nie można równocześnie pobierać danych i rozkazów (*von Neumann bottleneck*),
- nieograniczone możliwości modyfikacji programu,
- obiekt zapisany jako dana może być pobrany jako instrukcja,
- wykorzystywana w komputerach uniwersalnych.

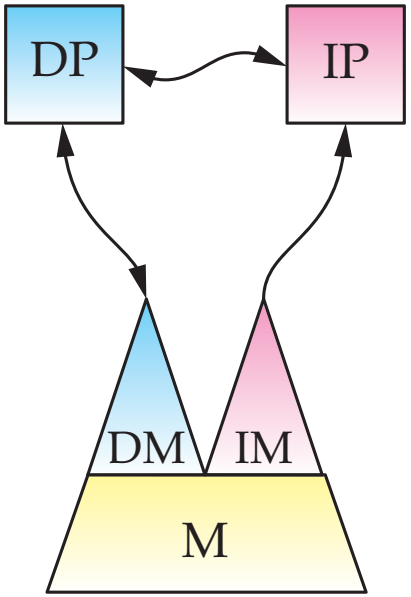


Uwaga! Rysunek posługuje się symbolami zapożyczonymi z taksonomii Skillicorna w sposób sprzeczny z zasadami budowy modeli wprowadzonymi przez tę taksonomię.

Zmodyfikowana architektura harwardzka – architektura mieszana, łączy w sobie cechy architektury harwardzkiej i architektury von Neumanna. Oddzielone zostały częściowo hierarchie pamięci na dane i instrukcje.

Cechy charakterystyczne architektury Harvard-Princeton:

- realizacja maszyny von Neumanna z oddzielonymi *górnymi* warstwami hierarchii pamięci i wspólnymi *dolnymi*,
- przynajmniej jeden poziom pamięci podręcznej jest oddzielny dla procesorów instrukcji i danych,
- szybkie działanie dzięki równoległości dostępów jak w architekturze Harvard,
- możliwość programowania niezbędna w komputerach uniwersalnych, jak w architekturze Princeton,
- program użytkowy nie ma pełnej kontroli nad położeniem obiektów w hierarchii pamięci, brak możliwości modyfikacji,
- w/w kontrolę może mieć wyróżniony program (proces) – system operacyjny, większość współczesnych komputerów uniwersalnych, w tym komputery PC ma architekturę Harvard-Princeton.



Uwaga! Rysunek posługuje się symbolami zapożyczonymi z taksonomii Skillicorna w sposób sprzeczny z zasadami budowy modeli wprowadzonymi przez tę taksonomię.

Dla dowolnego systemu liczenia istnieje zbiór cyfr, z których tworzone są liczby.

Systemy liczbowe dzieli się na:

- pozycyjne,
- niepozycyjne.

Systemy niepozycyjne

W systemach **niepozycyjnych** poszczególne cyfry zachowują swą wartość liczbową bez względu na miejsce jakie zajmują w liczbie (np. system rzymski – siedem znaków: I, V, X, L, C, D, M).

XII = 10 (X) + 1 (I) + 1 (I) = 12
IX = 10 (X) - 1 (I) = 9
MCDX = 1000 (M) + 500 (D) - 100 (C) + 10 (X) = 1410

Systemy pozycyjne

W systemach **pozycyjnych** wartość liczbowa cyfry zależy od umiejscowienia (pozycji) w liczbie (np. system dziesiętny, dwójkowy).

Liczba różnych cyfr systemu nazywa się jego **podstawą** P.

Wartość liczbowa cyfry określona jest przez **wagę**. Waga na pozycji n równa jest podstawie P podniesionej do potęgi n.

Sposób zapisu liczby L w dowolnym systemie pozycyjnym jest bardzo prosty:

$$L = a_{n-1} \cdot P^{n-1} + a_{n-2} \cdot P^{n-2} + \ldots + a_0 \cdot P^0 = \sum_{i=0}^{n-1} a_i \cdot P^i.$$

Dla systemu **dziesiętnego** (decymalnego):

$$P = 10, \quad a_n = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\},$$

$$142 = 1 \cdot 10^2 + 4 \cdot 10^1 + 2 \cdot 10^0 = 1 \cdot 100 + 4 \cdot 10 + 2 \cdot 1.$$

Dla systemu **dwójkowego** (binarnego):

$$P = 2, \quad a_n = \{0, 1\},$$

$$10010_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 1 \cdot 16 + 0 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = 18.$$

Dla systemu **ósemkowego** (oktalnego):

$$P = 8, \quad a_n = \{0, 1, 2, 3, 4, 5, 6, 7\},$$

$$144_8 = 1 \cdot 8^2 + 4 \cdot 8^1 + 4 \cdot 8^0 = 1 \cdot 64 + 4 \cdot 8 + 4 \cdot 1 = 100.$$

Dla systemu **szesnastkowego** (heksadecymalnego):

$$P = 16, \quad a_n = \{0, 1, 3, 4, 5, 6, 7, 8, 9, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}, \mathrm{f}\},$$

$$3e8_{16} = 3 \cdot 16^2 + 14 \cdot 16^1 + 8 \cdot 16^0 = 3 \cdot 256 + 14 \cdot 16 + 8 \cdot 1 = 1000.$$

Konwersja liczb naturalnych

$$190 = 1 \cdot 10^2 + 9 \cdot 10^1 + 0 \cdot 10^0,$$

$$190 = ?_2$$

dzielenie, wynik		
$190 / 2$	$= 95$	reszta $= 0$
$95 / 2$	$= 47$	reszta $= 1$
$47 / 2$	$= 23$	reszta $= 1$
$23 / 2$	$= 11$	reszta $= 1$
$11 / 2$	$= 5$	reszta $= 1$
$5 / 2$	$= 2$	reszta $= 1$
$2 / 2$	$= 1$	reszta $= 0$
$1 / 2$	$= 0$	reszta $= 1$

$$190 = 10111110_2.$$

$$10111110_2 = 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 =$$

$$= 1 \cdot 128 + 0 \cdot 64 + 1 \cdot 32 + 1 \cdot 16 + 1 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 0 \cdot 1.$$

Używając jednego bajtu, można zapisać liczę z zakresu $0 \ldots 255$, czyli $00000000_2 \ldots 11111111_2$ i $00_{16} \ldots ff_{16}$.

Działania arytmetyczne na liczbach w systemie dwójkowym i szesnastkowym są odpowiednikiem działań w systemie dziesiętnym i opierają się na elementarnych działaniach, np.:

- $1_2 + 0_2 = 1_2$,
- $1_2 + 1_2 = 10_2$,
- $1_2 \cdot 0_2 = 0_2$,
- $1_2 \cdot 1_2 = 1_2$,
- $10_2 - 1_2 = 1_2$.

Przykłady:

$$\begin{array}{r} \ 1\ 1\ 1\ 1\ 1\ 1 \\ \ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\ +\quad\ \ 1\ 0\ 0\ 1\ 1 \\ \hline 1\ 0\ 0\ 1\ 0\ 0\ 1\ 0 \end{array}$$

$$\begin{array}{r} \ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\ -\quad\ \ 1\ 0\ 0\ 1\ 1 \\ \hline 1\ 1\ 0\ 1\ 1\ 0\ 0 \end{array}$$

$$\begin{array}{r} \ \ 1\ 1\ 0\ 1 \\ \times\quad\ \ 1\ 0\ 1\ 1 \\ \hline \ \ 1\ 1\ 0\ 1 \\ \ 1\ 1\ 0\ 1 \\ \ 0\ 0\ 0\ 0 \\ +\ 1\ 1\ 0\ 1 \\ \hline 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1 \end{array}$$

$$\begin{array}{r|l} \ 1\ 1\ 0 & \\ \hline \ 1\ 1\ 0\ 1 & / \ 1\ 0 \\ -\ 1\ 0 & \\ \hline \ 0\ 1\ 0\ 1 & \\ -\quad 1\ 0 & \\ \hline \ 0\ 0\ 0\ 1 & \\ -\quad\ \ 1\ 0 & \\ \hline \ 0\ 0\ 0\ 1 & \end{array}$$

Jednostki informacji

Najmniejszą jednostką informacji potrzebna do określenia, który z dwóch równie prawdopodobnych stanów przyjął układ jest **bit**, **b** (*BInary digiT*). Bit odpowiada informacji Tak – Nie, Prawda – Fałsz, 1 – 0.

Jest to również najmniejsza jednostka informacji używana w odniesieniu do sprzętu komputerowego. Bit przyjmuje wartości wtedy **0** lub **1**.

Wyższe jednostki to informacji:

- półbajt (*nybble*) – 4 bity,
- **oktet** (*octet*) – 8 bitów,
- **bajt** (*byte*) **B** – pierwotnie liczba bitów przetwarzana jednocześnie przez komputer lub adresowana przez procesor, obecnie używany wyłącznie do oznaczania 8 bitów (czyli oktetu).

Często używa się też:

- **słowo** (*word*) to zwykle 16 bitów (2 bajty) – jednostka informacji, na której operuje komputer, mogą zdarzać się słowa o innej długości, np.: 4, 8, 16 bajtów,
- **słowo procesora** – jednostka informacji o długości naturalnej dla procesora,
- **słowo pamięci** – jednostka informacji możliwa do przesłania w jednym cyklu transmisji do/z pamięci, zwykle 64 b, czasem 128 b.

Do określenia większej liczby bajtów stosuje się często (**niepoprawnie**) przedrostki **kilo**, **mega**, **giga** itd. Są to przedrostki dziesiętne układu SI, będące wielokrotnościami liczby 10 ($10^{3 \cdot n}$) a nie liczby 2:

- **kilobajt** (*kilobyte*), **kB** – $10^3 = 1000$ bajtów,
- **megabajt** (*megabyte*), **MB** – $10^6 = 1000^2 = 1$ milion bajtów,
- **gigabajt** (*gigabyte*), **GB** – $10^9 = 1000^3 = 1$ miliard bajtów,
- **terabajt** (*terabyte*), **TB** – $10^{12} = 1000^4 = 1$ bilion bajtów.

W celu odróżnienia przedrostków o mnożniku 1000 od przedrostków o mnożniku 1024, pojawiła się propozycja ujednoznacznienia, opracowana przez IEC, polegająca na dodawaniu litery **i** po symbolu przedrostka dwójkowego oraz **bi** po jego nazwie.

Nowe przedrostki nazywane zostały przedrostkami **dwójkowymi** (binarnymi). Jednak ta propozycja rozwiązania problemu niejednoznaczności przedrostków nie została przyjęta przez wszystkie środowiska. Przedrostki dwójkowe są wielokrotnościami liczby 2 ($2^{10 \cdot n}$):

- **kibibajt** (*kibibyte*), **KiB** – $2^{10} = 1024$ bajty,
- **mebibajt** (*mebibyte*), **MiB** – $2^{20} = 1024^2 = 1$ milion 48 tysięcy 576 bajtów,
- **gibibajt** (*gibibyte*), **GiB** – $2^{30} = 1024^3 = 1$ miliard 73 miliony 741 tysięcy 824 bajtów,
- **tebibajt** (*tebibyte*), **TiB** – $2^{40} = 1024^4 = 1$ bilion 99 miliardów 511 milionów 627 tysięcy 776 bajtów.

Współczesne komputery są używane do przetwarzania danych różnych typów – liczbowych, logicznych, tekstowych, a także obrazów i dźwięków. Działają w oparciu o system binarny.

- Wartości logiczne (prawda/fałsz).
- Znaki pisarskie.
- Liczby
  - całkowite
    - nieujemne
    - ze znakiem
  - niecałkowite
    - stałopozycyjne
    - zmiennopozycyjne
- Dźwięki, sygnały jednowymiarowe.
- Obrazy rastrowe.

Wszystkie dane, na których operuje komputer, są zapisane w postaci tbciągów cyfr binarnych – bitów, interpretowanych najczęściej jako liczby binarne.

Wszelkie dane o charakterze innym niż liczby, muszą być zapisane (zakodowane) w postaci liczb lub grup liczb.

Komputer przetwarza wyłącznie grupy bitów, tworząc liczby binarne, których długość wynosi $8 \cdot 2^n$ bitów (np. 8, 16, 32, 64).

Dane alfanumeryczne

Dane alfanumeryczne, tekstowe – mają postać znaków pisarskich – liter, cyfr, znaków przestankowych i innych symboli. W komputerze są one reprezentowane przez liczby, określające pozycję danego symbolu w tablicy kodowej.

We współczesnych komputerach używa się kilku standardów kodowania znaków pisarskich:

- ASCII,
- EBCDIC,
- UNICODE.

ASCII

ASCII – (*American Standard Code for Information Interchange*), 7-bitowy kod przyporządkowujący liczby z zakresu 0 . . . 127 literom (alfabetu angielskiego), cyfrom, znakom przestankowym i innym symbolom oraz poleceniom sterującym. Został opracowany dla urządzeń dalekopisowych.

Na przykład litera **a** jest kodowana liczbą 97, a znak spacji jest kodowany liczbą 32.

Litery, cyfry oraz inne znaki drukowane tworzą zbiór znaków ASCII. Jest to 95 znaków o kodach 32 . . . 126. Pozostałe 33 kody (0 . . . 31 i 127) to tzw. kody sterujące służące do sterowania urządzeniem odbierającym komunikat, np. drukarką lub terminalem.

- Kody sterujące zajmują pozycje: $0 \ldots 31$, w tym:
  - **CR** - powrót na początek wiersza: kod 13,
  - **LF** – przejście do następnego wiersza: kod 10,
  - inne ważne: **HT** (tabulacja pozioma), **FF** (rozpoczęcie nowej strony), **BEL** (sygnał dźwiękowy), **VT** (tabulacja pionowa), **BSP** (cofnięcie o jeden znak).
- **Spacja**: kod 32 ($0x20$).
- **Cyfry** $0 \ldots 9$: kody $48 \ldots 57$ ($0x30 \ldots 0x39$).
- **Litery** w kolejności alfabetycznej:
  - wielkie: kody $65 \ldots 90$ ($0x41 \ldots 0x5a$),
  - małe: kody $97 \ldots 122$ ($0x61 \ldots 0x7a$).
- **Kasowanie znaku**: kod 127 ($0x7f$).

Ponieważ kod ASCII jest 7-bitowy, a większość komputerów operuje na 8-bitowych bajtach, dodatkowy bit został wykorzystany na powiększenie zbioru kodowanych znaków do 256 symboli.

Powstało wiele różnych rozszerzeń ASCII wykorzystujących ósmy bit. W kodach tych pierwsze 128 pozycji jest identyczne, jak w kodzie ASCII, a następne 128 pozycji zawiera znaki dodatkowe, np. litery akcentowane, rozszerzony zestaw symboli matematycznych, litery alfabetów narodowych (słowiańskie, skandynawskie, cyrylica, etc.).

Istnieje wiele rozszerzeń tej rodziny, używanych w różnych częściach świata. W Polsce najpowszechniej używa się kodów **ISO8859-2** oraz Microsoft **CP1250**, nazywanych stronami kodowymi.

BIN, DEC, HEX, znak				BIN, DEC, HEX, znak			
00000000	0	00	NUL (*Null*)	01000001	65	41	A
00000001	1	01	SOH (*Start Of Heading*)	01000010	66	42	B
00000010	2	02	STX (*Start of Text*)	01000011	67	43	C
00000011	3	03	ETX (*End of Text*)	01000100	68	44	D
...				...			
00110000	48	30	0	01110111	119	77	w
00110001	49	31	1	01111000	120	78	x
00110010	50	32	2	01111001	121	79	y
00110011	51	33	3	011 1010	122	7A	z
...				...			
00111101	61	3D	=	01111100	124	7C	|
00111110	62	3E	>	01111101	125	7D	}
00111111	63	3F	?	01111110	126	7E	~
01000000	64	40	@	01111111	127	7F	DEL (*Delete*)
...				...			

Unicode

Unicode – uniwersalny komputerowy zestaw znaków mający w zamierzeniu obejmować wszystkie znakówi pisarskie zapisu fonetycznego (głoskowego) używanych na całym świecie. Liczba pozycji kodowych jest praktycznie nieograniczona, obecnie jest zdefiniowanych kilkadziesiąt tysięcy znaków.

Definiują go dwa standardy – Unicode oraz **ISO10646**. Znaki obu standardów są identyczne. Standardy te różnią się w drobnych kwestiach, m.in. Unicode określa sposób składu.

Unicode jest rozwijany jest przez konsorcjum, w którego skład wchodzą firmy komputerowe, producenci oprogramowania, instytuty naukowe, agencje międzynarodowe oraz grupy zainteresowanych użytkowników. Konsorcjum współpracuje z organizacją ISO.

Standard Unicode obejmuje przydział przestrzeni numeracyjnej poszczególnym grupom znaków, nie obejmuje zaś sposobów bajtowego kodowania znaków.

Jest kilka metod kodowania, oznaczanych skrótowcami **UCS** (*Universal Character Set*) i **UTF** (*Unicode Transformation Format*). Do najważniejszych należą:

- UTF-32/UCS-4,
- UTF-16,
- UTF-8.

Kody pierwszych 256 znaków Unicode pokrywają się z kodami ISO Latin 1 (czyli ISO8859-1), przez co kody pierwszych 128 znaków pokrywają się z kodami ASCII.

Należy jednak pamiętać, że jest to zbieżność wyłącznie numerów przyporządkowanych konkretnym znakom, wartości bajtów użytych do ich zapisania mogą się różnić od tych, które uzyska się stosując Latin 1 lub ASCII.

UTF-8 – zmiennej długości system kodowania znaków dla Unicode. Może reprezentować każdy znak w systemie Unicode. Kodowanie zostało zaprojektowane dla zapewnienia zgodności z ASCII.

Zalety kodowania UTF-8:

- każdy tekst w ASCII jest tekstem w UTF-8,
- żaden znak spoza ASCII nie zawiera bajtu z ASCII,
- typowy tekst ISO Latin-x rozrasta się w bardzo niewielkim stopniu po przekonwertowaniu do UTF-8,
- znaki o kodzie różnym od 0 nie zawierają bajtu 0, co pozwala stosować UTF-8 w ciągach zakończonych zerem,
- o każdym bajcie wiadomo, czy jest początkiem znaku, czy też leży w jego środku,
- nie zawiera bajtów 0$x$ff i 0$x$fe, więc łatwo można go odróżnić od tekstu UTF-16.
- brak problemów z uporządkowaniem bajtów (*endianness*),
- jest domyślnym kodowaniem w ML (również w jego aplikacjach: HTML, SVG, XSL, CML, MathML).

Kody rodziny **EBCDIC** są używane w systemach firmy IBM. Bazują one na binarnym kodowaniu liczb dziesiętnych reprezentujących pozycje kodowe znaków.

Dźwięk i obraz

Dźwięk jest zapamiętywany w postaci wartości napięcia reprezentującego chwilowe ciśnienie akustyczne. Wartość ta jest kwantowana z użyciem rozpiętości zwykle od 8 bitów do 32 bitów z częstotliwością zależną od potrzeb, zwykle od 8 kHz do 48 kHz (próbkowanie).

Obraz rastrowy jest zapisany w postaci prostokątnej macierzy punktów (**pikseli**). Każdemu pikselowi odpowiada jeden kolor, zapisany w postaci składowych – jasności świateł (barw) podstawowych. Wartości jasności zapisane są w postaci liczb.

Dane logiczne

Najprostszy typ danych stanowią dane **logiczne**. Mogą one przyjmować dwie wartości. Bajtowe adresowanie danych używane w komputerach oraz fakt, że wiele komputerów traktuje jako podstawowy format danych słowo 32-bitowe powodują, że dane logiczne są zwykle zapisywane w postaci bajtów lub słów, pomimo, że do ich zapisu wystarczyłby pojedynczy bit.

Należy zwrócić uwagę na reprezentację wartości *prawda* i *fałsz* w różnych językach programowania. Wartość *fałsz* jest zwykle reprezentowana przez słowo o wszystkich bitach równych 0.

Wartość *prawda* może być reprezentowana przez liczbę całkowitą równą 1, liczbę całkowitą o dowolnej wartości różnej od zera (również ujemną), słowo o wszystkich bitach równych 1.

Liczby całkowite nieujemne

W dalszej będziemy posługiwali się założeniem, że dane reprezentowane są przez słowo komputera, w którym poszczególne bity zostały ponumerowane od prawej do lewej strony.

NBC

Kod **NBC** (*Natural Binary Code*, NKB, naturalny kod binarny) jest w zasadzie tym samym co pozycyjny system dwójkowy. Numer bitu jest równy wykładnikowi jego wagi binarnej.

$$L_{NCB} = \sum_{i=0}^{n-1} b_i \cdot 2^i.$$

Wartości wag w kodzie NBC (dla bajtu)

waga	$2^7 = 128$	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
cyfra	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

BCD

Zapis **BCD** (*Binary-Coded Decimal*, system dziesiętny zakodowany dwójkowo) polega na oddzielnym zakodowaniu w postaci binarnej (w kodzie NBC) każdej cyfry zapisu dziesiętnego, w postaci czterech bitów (tetrady). Zapis ten jest bardzo rzadko stosowany, głównie w mikrokontrolerach. Dozwolone wartości tetrady wynoszą $0 \ldots 9$. Można wyróżnić wersję *niespakowaną* kodu – jedna cyfra w bajcie lub *spakowaną* – dwie cyfry w bajcie.

Kody cyfr w kodzie BCD

kod	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
cyfra	0	1	2	3	4	5	6	7	8	9

Istnieją też inne warianty kodowania BCD.

Liczby całkowite ze znakiem

SM

Zapis **SM** (*Signed Magnitude*, ZM, znak-moduł) wydaje się być najbardziej intuicyjnym – jeden bit jest interpretowany jako znak liczby, pozostałe bity – jako wartość bezwzględna w kodzie NBC. Jest on jednak niewygodny dla jednostek arytmetycznych i współczesne komputery nie obsługują takich danych.

$$L_{SM} = -1^{b_{n-1}} \sum_{i=0}^{n-1} b_i \cdot 2^i.$$

Wartości wag w kodzie SM

waga	-1	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
cyfra	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

1C

System **1C** (*One's Complement*, U1, uzupełnień do 1) jest podobny do NBC, z tą różnicą, że najbardziej znaczący bit ma wagę ujemną powiększoną o jeden. Jest jeszcze czasem używany. Negacja liczby w tym systemie polega na negacji bitowej. Zero posiada dwie reprezentacje.

$$L_{1C} = -b_{n-1} \cdot (2^{n-1} - 1) + \sum_{i=0}^{n-1} b_i \cdot 2^i.$$

Wartości wag w kodzie 1C

waga	$-(2^8-1) = -127$	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
cyfra	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

2C

Zapis **2C** (*Two's Complement*, U2, uzupełnień do 2) jest najczęściej stosowanym zapisem liczb całkowitych. Jest on podobny do NKB, z tą różnicą, że najbardziej znaczący bit ma wagę ujemną. Typ `int` jest we współczesnych komputerach implementowany jako zapis 2C. Negacja liczby w tym systemie polega na negacji bitowej i inkrementacji.

$$L_{2C} = -b_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i.$$

Wartości wag w kodzie 2C

waga	$-(2^8) = -128$	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
cyfra	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

Biased

Zapis **biased** (spolaryzowany) umożliwia reprezentację liczb ze znakiem jako liczb bez znaku, z odpowiednim przesunięciem wartości 0. Liczby ujemne zapisywane są jako bliskie zeru, a zero jako wartość w połowie zakresu reprezentacji NBC. Zapis powstaje przez dodanie do zapisywanej wartości stałej – **podkładu**, a następnie zapisanie tak uzyskanej liczby w kodzie NBC. Jako wartość podkładu przyjmuje się zwykle wartość leżącą tuż poniżej połowy zakresu w NBC.

$$L_B = -BIAS + \sum_{i=0}^{n-1} b_i \cdot 2^i, \;\; BIAS = 2^{n-1} - 1.$$

Kody całkowitoliczbowe charakteryzują się zestawem własności, które decydują o wygodzie ich stosowania. Istotne własności to m.in.:

- reprezentacja zera - wpływa na łatwość wykrywania wartości 0 przy użyciu prostego układu logicznego,
- reprezentacja znaku – wpływa na łatwość rozróżnienia liczb ujemnych od dodatnich,
- operacja, jaką należy wykonać w celu zmiany znaku liczby – może to być prosta operacja logiczna, prosta lub złożona operacja arytmetyczna,
- łatwość wykonywania operacji arytmetycznych – dodawanie i odejmowanie w U2 jest realizowane tak samo, jak w NKB; mnożenie i dzielenie w U2 jest niewygodne, za to liczby zapisane w kodzie ZM można mnożyć i dzielić niemal tak samo, jak w NKB.

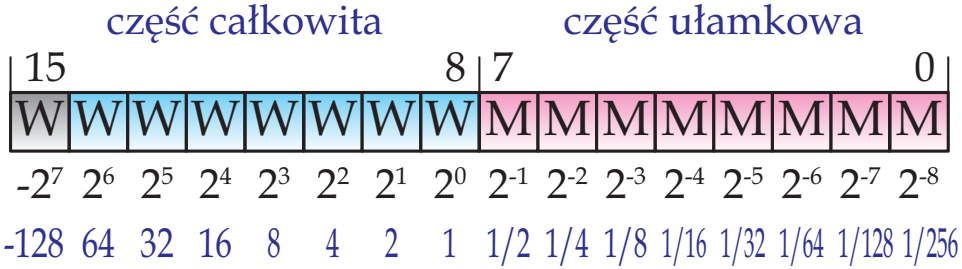
Liczby niecałkowite

System stałopozycyjny

Do zapisywania liczb ułamkowych i mieszanych można użyć zapisu **stałopozycyjnego**. W zapisie tym liczba jest reprezentowana przez słowo binarne, w którym pewne, z góry określone liczby bitów reprezentują część całkowitą i część ułamkową liczby.

Odpowiada to interpretacji zapisu całkowitoliczbowego, pomnożonej przez wartość będącą ujemną potęgą liczby 2.

Do zapisu liczb bez znaku używa się jako bazowej postaci NKB, a do zapisu liczb ze znakiem kodu U2 (zwykle).



Komputery zazwyczaj nie obsługują w szczególny sposób zapisów stałopozycyjnych. Podstawowe operacje są wykonywane tak samo, jak na liczbach całkowitych, odmienna jest jedynie interpretacja zapisu, za którą jest odpowiedzialny wyłącznie programista.

Zapis zmiennopozycyjny dla systemu dziesiętnego

Zapis **zmiennopozycyjny** umożliwia zapisywanie liczb całkowitych i ułamkowych o bardzo dużym zakresie dynamiki wartości bezwzględnych.

Każda liczba może być zapisana na kilka sposobów, różniących się położeniem przecinka oddzielającego część całkowitą od ułamkowej i wartością wykładnika:

$$-1,2345 \cdot 10^5 \qquad -0,12345 \cdot 10^6 \qquad -12,345 \cdot 10^4.$$

Wartość liczby zmiennoprzecinkowej jest obliczana według wzoru:

$$L = Z \cdot M \cdot B^W,$$

gdzie:

- Z (*sign*) – znak liczby, 1 lub -1,
- M (*mantissa*) – mantysa, liczba ułamkowa,
- B (*base*) – podstawa systemu liczbowego,
- W (*exponent*) – wykładnik (cecha), liczba całkowita.

Postacią **znormalizowaną** nazwiemy liczbę, która ma część całkowitą części znaczącej wyrażoną przez pojedynczą cyfrę różną od zera.

Aby zapisać (przechować) liczbę, musimy zapisać jej znak, część znaczącą oraz wykładnik, który jest liczbą całkowitą ze znakiem. Podstawa systemu jest ustalona i jej zapis jest zbędny.

$$-1,2345e5$$

W postaci znormalizowanej nie da się zapisać zera, ponieważ zero nie ma żadnej cyfry znaczącej różnej od 0.

Binarny zapis zmiennopozycyjny

Obecnie niemal wszystkie komputery posługują się binarnym zapisem zmiennopozycyjnym zgodnym ze standardem IEEE754.

Standard zakłada, że, o ile tylko jest to możliwe, liczby zapisuje się w postaci znormalizowanej. **Bazą** systemu jest liczba 2, **wykładnik** określa potęgę liczby 2.

Ponieważ w systemie binarnym jedyną cyfrą różną od zera jest jedynka, każda liczba w postaci znormalizowanej ma część całkowitą równą 1, nie ma więc potrzeby zapisywania jej, zapisuje się tylko część ułamkową.

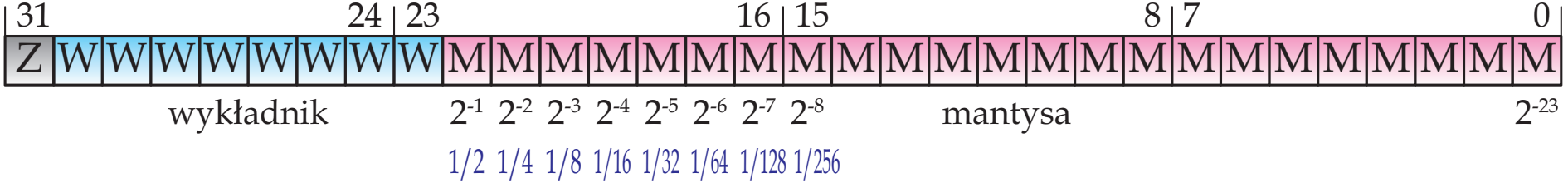
Wykładnik jest liczbą całkowitą ze znakiem. W IEEE754 wykładnik jest zapisywany w kodzie **spolaryzowanym** (biased), w którym wartość podkładu jest określona wzorcem bitowym $01\ldots11$, o liczbie bitów równej szerokości pola bitowego wykładnika. Dwie wartości pola wykładnika są zarezerwowane i oznaczają, że zapis nie reprezentuje postaci znormalizowanej.

Bity wykładnika o wzorcu $00\ldots00$ oznaczają zapis zdenormalizowany. Wartość wykładnika jest w tym przypadku taka sama, jak przy zapisie znormalizowanym z wzorcem wykładnika $00\ldots01$, a część całkowita części znaczącej ma wartość 0 (a nie 1 jak w postaci znormalizowanej).

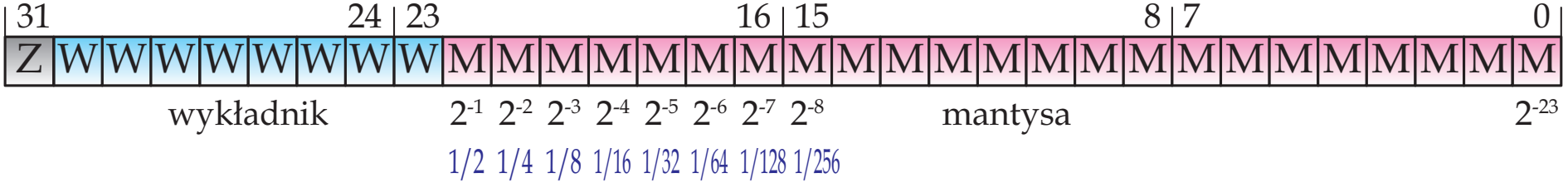
Pole wykładnika o wzorcu $11\ldots11$ oznacza nie-liczby: wartości **nieskończone** i wartości **błędne**.

Liczba pojedynczej precyzji zajmuje 4 bajty (32 bity).

Mantysa ma 23 bity, wykładnik ma 8 bitów, znak 1 bit, a nadmiar wynosi 127.



Mantysa jest **znormalizowana**, tj. należy do przedziału $[1, 2)$ (przedział prawostronnie otwarty). Jeżeli M jest stałe, a W zmienia się, wówczas przesunięciu ulega przecinek – stąd właśnie pochodzi nazwa tej reprezentacji.



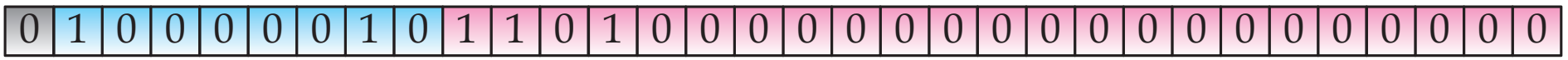
Procedura zapisu liczby L:

- określenie znaku, bit 31 $= 1$ jeżeli L jest ujemna, 0 jeśli dodatnia,
- znalezienie największej liczby postaci 2^W, mniejszej niż L,
- zapisanie wykładnika z nadmiarem $W + 127$,
- dzielenie liczby $L/2^W$ (wynik 1.xxxx),
- odjęcie 1 i szukanie mantysy,
- zaznaczenie bitu jako 1 jeśli po odjęciu $1/2$ (bit $+ 1$) mamy wartość nieujemną, jeśli wartość jest ujemna, bit oznaczany jest jako 0, i ignorowana operacja odejmowania,
- procedura jest powtarzana aż w wyniku odejmowania otrzymamy 0 lub dojdzie do bitu nr 0.

Przykład dla liczby 14,5:

- liczba jest dodatnia $\Rightarrow$ bit 31 $= 0$,
- największa liczba 2^W mniejsza niż 14,5 to $3^3 = 8 \Rightarrow W = 3$,
- zapisanie wykładnika $127 + W = 130 \Rightarrow 10000010$,
- dzielenie $14,5/2^3 = 1,8125$,
- odejmowanie $1,8125 - 1 = 0,8125$,
  - $0,8125 - 1/2 = 0.3125 \Rightarrow$ bit 23 $= 1$,
  - $0,3125 - 1/4 = 0.0625 \Rightarrow$ bit 22 $= 1$,
  - $0,0625 - 1/8 = -0.0625 \Rightarrow$ bit 21 $= 0$ (ignorowanie operacji),
  - $0,0625 - 1/16 = 0 \Rightarrow$ bit 20 $= 1$,
- pozostałe bity mantysy $= 0$.

14,5



Niektóre wartości specjalne

wartość,	wykładnik,	mantysa,	uwagi
NaN	11...11	xx...xx	symbol nie reprezentuje wartości liczbowej, powstaje zazwyczaj w wyniku niedozwolonej operacji (np. pierwiastkowanie liczby ujemnej)
INF	11...11	00...00	bit znaku decyduje o znaku wartości, rozróżnia się $+\infty$ i $-\infty$
0	00...00	00...00	bit znaku decyduje o znaku wartości, jest wynikiem operacji w przypadku wystąpienia nadmiaru (przepełnienia), przy dzieleniu przez 0, itp., może być dodatnia lub ujemna

Standard IEEE754, definiuje dwie klasy liczb:

- pojedynczej precyzji (`single`),
- podwójnej precyzji (`double`).

Podstawowym formatem jest format typu `double` – 64-bitowy (podwójnej precyzji).

Format 32-bitowy jest używany w zastosowaniach, gdzie wymagana precyzja jest niewielka, ma on tylko 23 bity znaczące (typ `single`).

Liczba **pojedynczej** precyzji: mantysa 23 bity, wykładnik 8 bitów, znak 1 bit, nadmiar $2^8 - 1 = 127$.

Liczba **podwójnej** precyzji: mantysa 52 bity, wykładnik 11 bitów, znak 1 bit, nadmiar $2^{11} - 1 = 1023$.

Format 80-bitowy (`extended`) był używany w starszych jednostkach zmiennopozycyjnych procesorów rodziny x86. Obecnie wychodzi z użycia.

Format 128-bitowy jest formatem *przyszłościowym* dla liczb o dużej precyzji.

Precyzja – liczba miejsc po przecinku jest określona przez mantysę.

Pojedyncza precyzja
Mantysa ma 23 bity, $1/2^{23} \approx 1,2 \cdot 10^{-7} \Rightarrow$ 7 cyfr po przecinku.

Podwójna precyzja
Mantysa ma 52 bity, $1/2^{52} \approx 2,2 \cdot 10^{-16} \Rightarrow$ 15 - 16 cyfr po przecinku.

Rozszerzona precyzja
Mantysa ma 64 bity, $1/2^{64} \approx 5,4 \cdot 10^{-20} \Rightarrow$ 19 cyfr po przecinku.

Operacje arytmetyczne

Mamy dwie liczby zmiennoprzecinkowe: $L_1 = M_1 \cdot B^{W_1}$ i $L_2 = M_2 \cdot B^{W_2}$, przy czym $L_1 > L_2$, wówczas:

- dodawanie
$$L_1 + L_2 = \left(M_1 + M_2 \cdot B^{W_2 - W_1}\right) \cdot B^{E_1},$$
- odejmowanie
$$L_1 - L_2 = \left(M_1 - M_2 \cdot B^{W_2 - W_1}\right) \cdot B^{E_1}.$$

Mamy dwie liczby zmiennoprzecinkowe: $L_1 = Z_1 \cdot M_1 \cdot B_1^W$ i $L_2 = Z_2 \cdot M_2 \cdot B_2^W$, wówczas:

- mnożenie
$$L_1 \cdot L_2 = (Z_1 \cdot Z_2) \cdot (M_1 \cdot M_2) \cdot B^{W_1 + W_2},$$
- dzielenie
$$L_1 / L_2 = (Z_1 \cdot Z_2) \cdot (M_1 / M_2) \cdot B^{W_1 - W_2}.$$

Jeśli liczby mają różne wykładniki, to podczas dodawania mantysa liczby o mniejszym wykładniku musi zostać **zdenormalizowana** – we wzorze jest to przemnożenie M_2 przez czynnik $B^{W_2-W_1}$.

W szczególnym przypadku, jeśli $W_1 - W_2$ jest większe niż liczba cyfr mantysy, to po denormalizacji mantysa będzie miała wartość 0, a liczba o mniejszym wykładniku nie wpłynie na wynik dodawania bądź odejmowania.

Liczby o skończonej reprezentacji dziesiętnej mogą mieć nieskończoną reprezentację binarną (np.: 0,1; 0,3).

Reprezentacja zmiennopozycyjna jest reprezentacją przybliżoną, a wyniki operacji są w rzeczywistości przybliżeniami. Oznacza to, że w praktyce nie można stosować relacji równości w odniesieniu do liczb zmiennopozycyjnych ($|a - b| < \epsilon$).

Arytmetyka liczb zmiennopozycyjnych **nie jest łączna**, dla x i y może zachodzić:

$$(x + y) + z \neq x + (y + z),$$
$$(x \cdot y) \cdot z \neq x \cdot (y \cdot z),$$

nie jest też rozdzielna, czyli:

$$x \cdot (y + z) \neq (x \cdot y) + (x \cdot z).$$

Innymi słowy, kolejność wykonywania operacji arytmetycznych ma znaczenie i wpływa na końcowy wynik.

Konwencje adresowania danych wielobajtowych

Podstawowa adresowalna komórka pamięci ma rozmiar jednego bajtu. Dane o rozmiarach przekraczających jeden bajt są przechowywane w kilku kolejnych komórkach, pod kilkoma kolejnymi adresami.

Fizyczna organizacja pamięci współczesnych komputerów jest nieco odmienna od organizacji logicznej. Bajty są pogrupowane w słowa pamięci, których długość jest najczęściej dwukrotnie większa od długości słów danych przetwarzanych przez procesor.

Taka konstrukcja pamięci umożliwia zwiększenie jej wydajności poprzez transmitowanie większej porcji danych podczas pojedynczego dostępu.

Współczesne procesory 64-bitowe współpracują z pamięcią o szerokości 128 bitów.

Istnieją dwa sposoby zapisu danej wielobajtowej w pamięci, zwane konwencjami adresowania danych. Nazwy konwencji pochodzą z powieści *Podróże Guliwera*, w której odnosiły się one do dwóch społeczności, różniących się zasadami jedzenia gotowanych jaj.

- Little-Endian – najmniej znaczący bajt pod najmniejszym adresem.
- Big-Endian – najbardziej znaczący bajt pod najmniejszym adresem.

Słowo 32-bitowe

A (MSB)	B	C	D (LSB)

Zapis Little-Endian

ADDR + 0	D (LSB)
ADDR + 1	C
ADDR + 2	B
ADDR + 3	A (MSB)

Zapis Big-Endian

ADDR + 0	A (MSB)
ADDR + 1	B
ADDR + 2	C
ADDR + 3	D (LSB)

W konwencji **Little-Endian** adres bajtu odzwierciedla wagę bajtu w liczbie. Jest to konwencja naturalna dla komputera, dla człowieka wydaje się dziwna – jesteśmy przyzwyczajeni do zapisywania liczb począwszy od najbardziej znaczących cyfr.

Istotną cechą tej konwencji jest to, że dana całkowitoliczbowa w długim formacie (np. 64 bity), przy dostępach do jej mniej znaczącej części o mniejszej długości (np. dwóch lub czterech bajtów) adres danej będzie w każdym przypadku ten sam.

Jest to wygodne przy częstym rzutowaniu typów całkowitoliczbowych, co ma miejsce w programach pisanych w języku C.

Zapis **Big-Endian**, naturalny dla człowieka, jest zwykle mniej wygodny dla komputera. Dostęp do danej całkowitoliczbowej w pamięci wymaga zmiany wartości adresu w zależności od długości danej.

Jeżeli liczba 32-bitowa zostanie zapisana pod adresem `A`, to jej wartość w postaci jednobajtowej ma adres `A + 3`.

Istotną zaletą konwencji Big-Endian jest możliwość szybkiego porównywania łańcuchów tekstowych przy użyciu instrukcji operujących na liczbach całkowitych o długości 32 lub 64-bitów. Wektor znaków (bajtów) interpretowany jako liczba ma w najbardziej znaczącym bajcie pierwszy bajt łańcucha.

Zamiast więc prowadzić operację porównywania łańcuchów znak po znaku, można ją wykonać posługując się grupami znaków o długości słowa procesora.

Wyrównanie danych

Dane powinny być umieszczone w pamięci w taki sposób, aby dostęp do danej, której rozmiar nie przekracza długości słowa pamięci następował w pojedynczym cyklu dostępu do pamięci.

W nowszych architekturach komputerów rozmieszczenie danych gwarantujące dostęp w jednym cyklu transmisji jest obligatoryjne. Próba dostępu do danej położonej inaczej generuje błąd.

Dostęp w jednym cyklu przesłania można zagwarantować, umieszczając każdą daną skalarną pod adresem podzielnym przez jej długość. Dane 32-bitowe powinny być położone pod adresami podzielnymi przez 4, a 16-bitowe pod adresami parzystymi. Takie umieszczenie danych w pamięci nazywa się **wyrównaniem naturalnym**.

Nawet jeśli model programowy procesora nie narzuca takiego wymagania, wyrównanie naturalne jest zwykle wymuszane przez kompilatory (np. x86).

Wymagania języków wysokiego poziomu

Głównym celem projektantów pierwszych komputerów było uzyskanie urządzenia, które byłoby w stanie działać bezawaryjnie wystarczająco długo, aby wykonać potrzebne obliczenia.

Po upowszechnieniu komputerów ustabilizowały się metody ich programowania, obecnie powszechnie używa się języków wysokiego poziomu – najczęściej **proceduralnych** i **obiektowych**.

Obecnie komputery są budowane z myślą o programowaniu ich w takich właśnie językach. Struktura logiczna komputerów jest zaprojektowana tak, aby mogły one łatwo i wydajnie wykonywać programy powstałe przez translację zapisu algorytmów w językach wysokiego poziomu.

Języki obiektowe od strony implementacji nie różnią się znacząco od języków proceduralnych. Możny przyjąć, że komputer, który daje się efektywnie programować w języku **C** będzie podobnie skutecznie wykonywał programy napisane w innych podobnych językach proceduralnych i obiektowych.

Przykładowy program w języku **C** zawiera podstawowe obiekty i mechanizmy występujące w typowych programach.

```
int x, y;
char *p;

int add(int a, int b)
{
  int r;
  r = a + b;
  return r;
}

void clean(void)
{
  free(p);
}

int main(void)
{
  p = malloc(256);
  x = add(2, 3);
  clean();
}
```

Główna procedura programu wywołuje inne **procedury** lub **funkcje**, przekazując do nich **argumenty** i odbierając wartości.

W programie występują dane o różnym zasięgu i czasie życia: **zmienne zewnętrzne x**, **y** i **p**, **zmienne lokalne** i **argumenty procedur** oraz zmienna wskazywana przez zmienną wskaźnikową **p**.

Wykonanie programu napisanego w języku wysokiego poziomu wymaga umieszczenia w pamięci komputera wszystkich występujących w programie **obiektów** oraz realizacji mechanizmów związanych z **wywoływaniem procedur**.

Poszczególne rodzaje obiektów – **instrukcje** i **dane** będą odwzorowane w odrębne sekcje, czyli **klasy** pamięci.

Wywoływanie procedur wymaga zrealizowania mechanizmów **przekazywania argumentów**, przekazywania i zwracania **sterowania** oraz zwracania **wartości funkcji**.

Podczas wykonania programu w pamięci znajdą się obiekty czterech podstawowych rodzajów, odwzorowane w różne klasy pamięci.

- Kod – będzie zawierała instrukcje tworzące program.
- Klasa danych statycznych – zawiera dane zadeklarowane na poziomie zewnętrznym:
  - stałe,
  - zmienne zainicjowane,
  - zmienne niezainicjowane.
- Klasa danych automatycznych – zawiera argumenty wywołania i zmienne lokalne procedur.
- Klasa danych kontrolowanych – zawiera dane dynamiczne jawnie tworzone i usuwane przez programistę.

Sekcja **kodu** nie zmienia swoich rozmiarów ani zawartości przez cały czas wykonywania programu. Jest ona tworzona przed rozpoczęciem wykonywania programu, a niszczona po jego zakończeniu (jest statyczna).

Oprócz samych instrukcji sekcja może zawierać stałe niezbędne do działania programu, w tym m.in. **adresy ciągów instrukcji** odpowiedzialnych za poszczególne ścieżki instrukcji złożonych typu switch oraz **stałe** (literały) występujące w programie, które nie mogą być zapisane bezpośrednio jako argumenty natychmiastowe.

Sekcja **danych statycznych** zawiera dane deklarowane w języku programowania na poziomie zewnętrznym oraz dane deklarowane lokalnie wewnątrz procedur ze słowem kluczowym `static`.

Sekcja danych statycznych, ma stały rozmiar i istnieje przez cały czas wykonania programu. W sekcji tej mogą jednak występować obiekty o różnych dozwolonych rodzajach dostępu i odmiennych sposobach inicjowania.

Sekcja **danych dynamicznych** automatycznych służy do przechowywania **argumentów procedur** i ich **zmiennych lokalnych**. Są one powoływane do życia w miarę zagłębiania wywołań procedur i usuwane podczas kończenia wykonania procedur. Kolejność usuwania jest zawsze odwrotna do kolejności tworzenia. Dane te tworzą strukturę danych zwaną **stosem** (*stack*).

Dane dynamiczne kontrolowane są alokowane i dealokowane **jawnie przez programistę**. Przed rozpoczęciem dealokacji obszar tych danych zachowuje się podobnie do stosu, jednak kolejność dealokacji może być dowolna, a nowe obiekty mogą być alokowane w miejscu zwolnionym przez poprzednie. Dane dynamiczne kontrolowane tworzą strukturę danych zwaną **stertą** (*heap*).

Kolejność rozmieszczenia sekcji w przestrzeni adresowej oraz wartości ich adresów początkowych zależy od projektanta systemu operacyjnego (projektanta).

Adresy bliskie zera często pozostają wolne, a dostęp do nich nie jest dozwolony. Dzięki temu próby odwołań przy użyciu niezainicjowanych zmiennych wskaźnikowych lub wskaźników o wartości NULL (zwykle równej 0) są wychwytywane i sygnalizowane jako błędy wykonania.

Z cech maszyny von Neumanna wynika obecność w procesorze rejestru, służącego do wskazywania kolejnych wykonywanych instrukcji. Ponieważ zawartość tego rejestru jest inkrementowana po pobraniu każdej instrukcji, rejestr ten nosi nazwę **licznika instrukcji**.

Podczas wykonywania instrukcji rejestr PC wskazuje następną instrukcję po aktualnie wykonywanej. Taka wartość PC jest określana jako nextPC (PC następnej instrukcji).

Podczas wykonywania instrukcji skoku do rejestru licznika instrukcji jest ładowana nowa wartość.

Procesor przystosowany do wykonywania programu napisanego w języku wysokiego poziomu musi umożliwiać łatwą implementację **przekazywania sterowania** pomiędzy procedurami. W tym celu potrzebne są dwie instrukcje: **skoku ze śladem** i **powrotu według śladu**.

Instrukcja **skoku ze śladem** wykonuje skok po uprzednim zapamiętaniu wartości rejestru PC. Podczas wykonywania instrukcji skoku ze śladem PC wskazuje następną instrukcję po instrukcji skoku, i taka właśnie wartość PC podlega zapamiętaniu.

Instrukcja skoku ze śladem służy do przekazania sterowania do procedury (wywołania procedury).

Instrukcja **powrotu według śladu** jest instrukcją skoku, której adres docelowy jest uprzednio zapamiętanym adresem śladu wywołania, czyli adresem wskazującym następną instrukcję po ostatnio wykonanej instrukcji skoku ze śladem.

Z opisu mechanizmów wywołania i powrotu z procedur wynika, że stos oprócz argumentów i zmiennych lokalnych procedur powinien przechowywać również **ślady powrotów** i ewentualnie **inne informacje** wymagające przechowania podczas wywoływania procedur.

Najprostszy model procesora zawiera minimum mechanizmów niezbędnych do wykonania zaprezentowanego wcześniej programu w języku C.

Modelowy procesor jest wyposażony w **rejestr licznika instrukcji** oraz rejestr przechowujący wynik obliczeń, służący jednocześnie jako rejestr argumentu źródłowego. Rejestr taki jest nazywany **akumulatorem**.

Model procesora obsługuje stos i realizuje podstawowe operacje na **stosie**, jednak implementacja stosu pozostaje nieokreślona.

Procesor wykonuje dostępy do danych wyspecyfikowanych przez podanie nazw lub wartości. Oczywiście ten aspekt modelu jest wysoce nierealistyczny.

instrukcja, działanie, uwagi		
`LOAD d`	`A = d`	załadowanie danej do akumulatora
`STORE d`	`d = A`	składowanie danej z akumulatora do pamięci
`ADD d`	`A += d`	dodanie danej do akumulatora
`PUSH d`		umieszczenie danej na stosie
`CALL a`	`PUSH PC; PC = a`	skok i zapamiętanie śladu na sosie
`RETURN`	`POP PC`	powrót i odtworzenie śladu ze stosu
`CREATE d`		alokacja zmiennej na stosie
`DESTROY d`		dealokacja danej ze stosu

```
// x = add(2, 3)

    push    2
    push    3
    call    add

    destroy
    destroy
    store   x
```

Zgodnie z typową konwencją języka C argumenty procedury są umieszczane na stosie w kolejności od ostatniego do pierwszego. Służy do tego instrukcja PUSH.

Po przygotowaniu argumentów następuje skok ze śladem, rozpoczyna wykonywanie się procedury, zapamiętany ślad wskazuje na instrukcje następna po instrukcji skoku.

```
// int r;
    create    r

// r = a + b;
    load      a
    add       b
    store     r

// return r;
    load      r

    destroy   r
    return
```

Alokacja zmiennej lokalnej – CREATE, załadowanie danej a i dodanie do niej b i zapamiętanie na stosie.

Ostatnie dwie instrukcje stanowią epilog procedury. Służą one do dealokacji zmiennej lokalnej i powrotu do procedury wywołującej.

```
// x = add(2, 3)

    push    2
    push    3
    call    add

    destroy
    destroy
    store   x
```

Procedura wywołująca usuwa ze stosu niepotrzebne już argumenty, instrukcja DESTROY.

Wartość funkcji dostępna w rejestrze zostaje przesłana do zmiennej x.

Stos jest najczęściej implementowany w pamięci komputera. W tym celu do procesora wprowadza się **rejestr wskaźnika stosu**, który zawiera adres wierzchołka stosu.

Najczęściej implementowany rodzaj stosu nazywa się **stosem pełnym schodzącym**. Przy takiej implementacji SP zawiera adres danej ostatnio umieszczonej na stosie, a stos rośnie w kierunku malejących adresów.

Każda procedura do swojego wykonania potrzebuje tylko niewielkiego fragmentu stosu, złożonego z argumentów wywołania, śladu powrotu i zmiennych lokalnych.

Fragment stosu używany przez procedurę nazywa się **ramką stosu** procedury. Część ramki, która jest obecna na stosie w chwili przekazania sterowania do procedury, nazywa się **rekordem aktywacji**.

W każdej chwili podczas wykonania procedury można wyznaczyć odległość pomiędzy wierzchołkiem stosu i każdym obiektem ramki stosu. Dostęp do obiektów ramki stosu może być realizowany poprzez adresy, stanowiące sumę wskaźnika stosu i stałej, określającej odległość obiektu od wierzchołka stosu. Adresy takie mogą zastąpić nazwy.

Do adresowania ramki stosu można posłużyć się **rejestrem wskaźnika ramki**. Zawartość wskaźnika ramki, w przeciwieństwie do zawartości wskaźnika stosu, nie zmienia się podczas wykonania ciała procedury.

Wskaźnik ramki wskazuje ramkę bieżącej procedury. Oznacza to, że każda procedura ustanawia własną wartość wskaźnika ramki.

Każda procedura spodziewa się również, że jej wartość wskaźnika ramki nie zostanie zmieniona. Oznacza to, że procedura musi przed powrotem odtworzyć wskaźnik ramki procedura wołającej.

Po wprowadzeniu wskaźnika ramki możemy wskazać odwzorowanie modelu procesora w rzeczywisty model programowy procesorów rodziny x86, używanych w komputerach PC.

Procesory x86 posiadają 8 rejestrów uniwersalnych (w tym główny akumulator **EAX**, wskaźnik stosu **ESP** i wskaźnik ramki **EBP**) oraz rejestr licznika instrukcji oznaczony symbolem **EIP**.

Większość instrukcji występuje w postaci dwuargumentowej, a w zapisie symbolicznym instrukcji jako pierwszy podaje się argument docelowy, a następnie argumenty źródłowe.

W instrukcjach arytmetycznych i logicznych argument docelowy jest jednocześnie pierwszym argumentem źródłowym.

Procesory x86 umożliwiają adresowanie pamięci sumą zawartości rejestru i stałej (przemieszczenia). Taki tryb adresowania jest wygodny do adresowania obiektów w ramce stosu. Możliwe jest adresowanie zarówno względem wskaźnika stosu, jak i względem wskaźnika ramki.

instrukcja,	instrukcja x86,	działanie,	uwagi
	`MOV a, s`	`d = s`	przesłanie danych
`LOAD d`	`MOV EAX, d`	`EAX = d`	załadowanie danej do akumulatora
`STORE d`	`MOV d, EAX`	`d = EAX`	załadowanie danej z akumulatora
`ADD d`	`ADD EAX, d`	`EAX += d`	dodanie danej do akumulatora
`PUSH d`	`PUSH d`	`*--ESP = d`	umieszczenie danej na stosie
`POP d`	`POP d`	`d = *ESP++`	zdjęcie danej ze stosu
`CALL l`	`CALL l`	`*--ESP = EIP; EIP = l`	wywołanie procedury
`RETURN`	`RET`	`EIP = *--ESP`	powrót z procedury
`CREATE d`	`SUB ESP, s`	`ESP -= s`	alokacja danych na stosie
`DESTROY d`	`ADD ESP, s`	`ESP += s`	dealokacja danych ze stosu

```
// x = add(2, 3)

    push    2
    push    3
    call    add

    add     esp, 8
    mov     [x], eax
```

Sekwencja wywołania procedury jest taka sama.

Po powrocie z procedury następuje usunięcie argumentów. Polega ono na przesunięciu wskaźnika stosu orozmiar argumentów – w tym przypadku o 8 (dwa argumenty po 32 bity).

Wartość funkcji zostaje zapamiętana w zmiennej statycznej x. Zapis [x] oznacza komórkę pamięci o adresie **x**. Ponieważ x jest zmienną statyczną, jej adres jest stałą. która może zostać zapisana symbolicznie.

```
    push        ebp
    mov         ebp, esp

// int r;
    sub         esp, 4

// r = a + b;
    mov         eax, [ebp + 8]
    add         eax, [ebp + 12]
    mov         [ebp], eax

//return r;
    mov         eax, [ebp - 4]

    mov         esp, ebp
    pop         ebp
    ret
```

W wersji korzystającej ze wskaźnika ramki na początku następuje zapamiętanie wskaźnika ramki procedury wołającej i ustanowienie własnej wartości wskaźnika ramki. Kolejna instrukcja służy do alokacji zmiennej lokalnej.

Alokacja i dealokacja zmiennej `r` polega na wykonaniu operacji na wskaźniku stosu. Dostępy do obiektów lokalnych są realizowane przy użyciu adresowania z użyciem ramki.

Począwszy od modelu procesora 80186 dwie instrukcje (`MOV ESP, EBP; POP EBP`) mogą być zastąpione pojedynczą, bezargumentową instrukcją `LEAVE`.

Składniki modelu programowego

Użytkowy model programowy to zbiór zasobów logicznych komputera widzianych przez programistę lub kompilator. Model programowy nie ma jednoznacznego i bezpośredniego związku z implementacją, czyli budową procesora i komputera.

Istnieje wiele rodzin komputerów czy procesorów o wspólnym modelu programowym i wielu odmiennych implementacjach. Przykładem może to być rodzina x86 (zapoczątkowana w 1976 roku).

- Rejestry – liczba i funkcjonalność rejestrów procesora.
- Tryby adresowania – sposoby specyfikacji argumentów operacji.
- Model operacji warunkowych – sposób realizacji konstrukcji warunkowych.
- Lista instrukcji – zestaw operacji możliwych do wykonania przez procesor.

Rejestry

Rejestry procesora mogą pełnić różne role w programach.

Akumulator może być użyty jako argument źródła i równocześnie przeznaczenia dla operacji arytmetycznej lub logicznej.

Rejestr służący do uzyskania adresu danej umieszczonej w pamięci nazywa się ogólnie **rejestrem adresowym**.

Jeśli rejestr może być użyty w trybie adresowania rejestrowym pośrednim, jest on nazywany **rejestrem bazowym**.

Rejestr może również służyć do odliczania iteracji pętli. Rejestr przewidziany do takiego zastosowania nazywa się **licznikiem pętli**.

Istnieje wiele różnych koncepcji organizacji zestawu rejestrów, począwszy od architektur bezrejestrowych a skończywszy na architekturach z bardzo dużym zestawem rejestrów.

- Brak rejestrów.
- Minimalny zestaw rejestrów.
- Mały zestaw rejestrów specjalizowanych.
- Mały zestaw rejestrów uniwersalnych.
- Duży zestaw rejestrów uniwersalnych.
- Zestaw rejestrów jako bufor ramki.
- Stosowy zestaw rejestrów.

Architektury bezrejestrowe, określane również mianem *pamięć-pamięć*, nie przechowują danych w rejestrach procesora. Procesor jest wyposażony w kilka rejestrów, które służą wyłącznie do przechowywania adresów.

Architektury z **minimalnym zestawem rejestrów** posiadają jeden lub dwa akumulatory i jeden lub dwa rejestry adresowe. Niemal wszystkie instrukcje wymagają odwołania do pamięci, ale jeden z argumentów instrukcji zwykle znajduje się w rejestrze.

Generowanie kodu dla takich procesorów jest proste, ale uzyskany program wykonuje się stosunkowo wolno wskutek dużej liczby odwołań do pamięci.

Architektury tego typu nie są stosowane w komputerach uniwersalnych.

W architekturach z **małym zestawem rejestrów specjalizowanych** (np. x86 w trybie 16-bitowym) mamy do czynienia z zestawem kilku rejestrów, niekiedy częściowo uniwersalnych, które są argumentami domyślnymi wielu instrukcji.

Ponieważ kompilator używa czasem instrukcji, których argumenty muszą być umieszczone w konkretnych rejestrach, rejestry nie mogą być użyte do przechowywania danych programu (argumentów i zmiennych lokalnych procedur).

Służą one jedynie do przechowywania tymczasowych wyników obliczeń oraz jako argumenty instrukcji, z którymi są domyślnie związane.

Pomimo, że pojemność zestawu rejestrów jest znacząca, kompilator nie jest w stanie efektywnie korzystać z rejestrów.

W architekturach z **małym zestawem rejestrów uniwersalnych** mamy do czynienia z zestawem około 8 rejestrów, z których przynajmniej 5 daje się użyć w dowolnym charakterze – jako akumulatory lub rejestry adresowe.

Przykładem jest tu 32-bitowa wersja x86, zapoczątkowana przez model 80386 w roku 1985. Rejestry odziedziczone po wersji 16-bitowej zostały rozszerzone do 32 bitów. Wprowadzono również nowy zestaw trybów adresowania, umożliwiający wykorzystanie do adresowania wszystkich rejestrów.

Wprowadzenie nowych instrukcji mnożenia umożliwiło zmniejszenie *przywiązania* rejestrów do instrukcji. W ten sposób kompilator może używać rejestrów znacznie elastyczniej niż we wcześniejszych procesorach tej rodziny.

W architekturach z **dużym zestawem rejestrów** (np. AMD64) mamy do czynienia z wieloma rejestrami, które mogą pełnić dowolną rolę.

Umożliwia to kompilatorom używanie rejestrów do przekazywania argumentów wywołania i zmiennych lokalnych procedury.

Odwołania do pamięci zostają skupione w prologu i epilogu procedury, gdzie zachodzi przeładowanie ramki stosu pomiędzy rejestrami i stosem w pamięci.

W niektórych architekturach bardzo duży zestaw rejestrów jest używany do zbuforowania wewnątrz procesora wierzchołka stosu umieszczonego w pamięci (IA-64). W ten sposób uzyskuje się znaczną redukcję liczby odwołań do pamięci, również w prologu i epilogu procedury.

Ponieważ zestaw rejestrów mieści kilka ramek stosu, odwołania do pamięci zachodzą przy przepełnieniu lub niedopełnieniu stosu w rejestrach, co ma miejsce raz na kilka poziomów wywołań procedur.

W architekturach ze **stosowym zestawem rejestrów**, są one zorganizowane w postaci niewielkiego stosu. Ponieważ operacje są domyślnie wykonywane na wierzchołku stosu, procesory o takich zestawach rejestrów mają zwykle instrukcje bezargumentowe lub jednoargumentowe.

Tryby adresowania

Tryb adresowania oznacza sposób specyfikacji (wyznaczania adresu) argumentu operacji (operandu) i wyników operacji.

- Nie odnoszące sie do pamięci:
  - adresowanie natychmiastowe,
  - adresowanie rejestrowe bezpośrednie.
- Odnoszące się do pamięci:
  - adresowanie pamięciowe bezpośrednie,
  - adresowanie rejestrowe pośrednie,
  - adresowanie pamięciowe pośrednie,
  - adresowanie indeksowe (z przesunięciem),
  - adresowanie bazowe.

W adresowaniu **natychmiastowym** argument jest pobierany bezpośrednio z rozkazu (np. `MOV AX, 5`).

W adresowaniu **rejestrowym bezpośrednim** operandy znajdują się w rejestrach (np. `MOV AX, BX`).

W przypadku adresowania **pamięciowego bezpośredniego** adres operandu znajduje się w instrukcji (np. `MOV AX, [40]`).

W trybie adresowania **rejestrowego pośredniego** argument znajduje się w pamięci, a adres argumentu lub jego składnik znajduje się w rejestrze (np. `MOV AX, [CX]`).

W trybie adresowania **pamięciowego pośredniego** operand zawarty jest w pamięci pod adresem, który jest w pamięci (np. `MOV AX, [[40]]`), raczej rzadko spotykane.

Adresowanie **indeksowe** jest rodzajem adresowania pośredniego, powstaje przez zsumowanie adresu efektywnego z wartością rejestru, opcjonalnie pomnożoną przez stałą (będącą potęgą liczby dwa).

Adresowanie z **bazą w liczniku instrukcji** umożliwia adresowanie danych względem adresu bieżącej instrukcji, jest wygodne do adresowania tablic adresów.

Adresowanie z **automodyfikacją bazy** jest rodzajem adresowania pośredniego, w którym wartość rejestru bazowego jest modyfikowana o długość przesłanej danej przed lub po wykonaniu przesłania danej.

Niejawnie korzystają z nich operacje stosowe (PUSH – predekrementacja, POP – postinkrementacja).

W celu umożliwienia efektywnej implementacji języków wysokiego poziomu procesor powinien posiadać trzy tryby adresowania.

Tryb natychmiastowy służy do ładowania do rejestrów stałych, w tym również adresów danych statycznych.

Tryb rejestrowy bezpośredni umożliwia użycie zawartości rejestru jako argumentu operacji.

Do adresowania danych w pamięci wystarczy jeden z trybów rejestrowych pośrednich. W ten sposób uzyskujemy możliwość adresowania zmiennym adresem, wyliczonym uprzednio w rejestrze.

Adresowanie danych stałym adresem może być potraktowane jako szczególny przypadek adresowania ze zmiennym adresem. Najwygodniejszym trybem adresowania pamięci jest tryb rejestrowy pośredni z przemieszczeniem.

Modele operacji warunkowych

Współczesne procesory implementują jeden z trzech sposobów realizacji operacji warunkowych:

- Model ze znacznikami.
- Model bez znaczników.
- Model z predyktami.

Znacznik to jednobitowy rejestr atrybutu ostatnio wykonanej operacji.

Operacja warunkowa jest realizowana dwufazowo (dwie instrukcje), obie fazy mogą być rozsunięte w czasie.

W pierwszej fazie w wyniku wykonania operacji arytmetycznej lub logicznej zostają ustawione znaczniki (np. CMP – porównania a dokładniej odejmowanie argumentów).

W drugiej fazie następuje wykonanie operacji zależne od stanu znaczników.

Instrukcja warunkowa specyfikuje warunek wykonania. Jeśli warunek (wyrażenie logiczne na znacznikach) jest spełniony, instrukcja wykonuje zadaną operację (np. **skok** lub **przesłanie**). W przeciwnym razie instrukcja wykonuje się jako pusta.

W niektórych architekturach większość instrukcji jest wykonywanych jako warunkowe.

W architekturze x86 znaczniki umieszczone są w rejestrze stanu procesora **EFLAGS**:

- OF – nadmiar (bit 0),
- SF – znak (bit 2),
- ZF – zero (bit 4),
- AF – przeniesienie pomocnicze (bit 6),
- PF – parzystość (bit 7),
- CF – przeniesienie (bit 11).

Symboliczne warunki dostępne w x86. Stosowane w nazwach warunków skróty **A**, **B**, **G** i **L** (*above*, *below*, *greater*, *less*) oznaczają odpowiednio relacje większości i mniejszości dla liczb bez znaku (A, B) i ze znakiem (G, L).

flaga, ∼flaga, sprawdzany warunek		
O	NO	`OF = 1`, `OF = 0`
C, B, NAE	NC, NB, AE	`CF = 1`, `CF = 0`
Z, E	NZ, NE	`ZF = 1`, `ZF = 0`
BE, NA	NBE, A	`(CF or ZF) = 1`, `(CF or ZF) = 0`
S	NS	`SF = 1`, `SF = 0`
P, PE	NP, PO	`PF = 1`, `PF = 0`
L, NGE	NL, GE	`(SF xor OF) = 1`, `(SF xor OF) = 0`
LE, NG	NLE, G	`((SF xor OF) or ZF) = 1`, `((SF xor OF) or ZF) = 0`

Przykłady instrukcji skoku warunkowego: `JB` – mniejsze (liczby bez znaku), `JNLE` – nie mniejsze i nie równe (liczby ze znakiem) i przypisania warunkowego: `SETB`, `SETNLE`.

W modelu operacji warunkowych **bez znaczników** operacja warunkowa jest realizowana przez pojedynczą instrukcję. Instrukcja ta ewaluuje wartość logiczną relacji, a następnie wykonuje jakąś czynność, jeśli relacja jest spełniona.

Taki model operacji warunkowych jest prosty i wydajny w implementacji. jest on stosowany w prostych architekturach RISC.

Predykaty to uogólnione znaczniki, mogą przechowywać wartość logiczną dowolnej wcześniej obliczonej relacji. Duża liczba predykatów pozwala na równoczesne przechowywanie wartości wielu relacji.

Model z predykatami jest spotykany w nowych architekturach o dużej równoległości wykonania instrukcji, dużych zestawach rejestrów. Jedyną architekturą komercyjną implementującą ten model jest IA-64.

Podejście CISC i RISC

Wspomniano, że model programowy procesora jest kompozycją czterech składników: zestawu rejestrów, zestawu trybów adresowania, modelu operacji warunkowych i listy instrukcji.

Istnieją trzy główne podejścia do kompozycji modelu programowego:

- CISC (*Complex Instrucion Set Computer*),
- RISC (*Reduced Instrucion Set Computer*),
- VLIW.

W tym podziale chodzi o złożoność instrukcji a nie o ich liczbę.

Podejścia CISC i RISC różnią się złożonością poszczególnych instrukcji. Istnieją procesory RISC wykonujące zaledwie około 30 instrukcji, ale również takie, które wykonują ponad 200 różnych instrukcji. Typowe procesory CISC wykonują kilkadziesiąt instrukcji.

Podejście **CISC** jest podejściem klasycznym (zapoczątkowane w latach 60 i 80 XX w.) do budowy modelu programowego. Instrukcje języka wysokiego poziomu powinny mieć proste odwzorowanie w instrukcjach procesora, a dane lokalne są przechowywane na stosie w pamięci.

W CISC rejestry służą do adresowania pamięci i przechowywania danych tymczasowych, a tryby adresowania odzwierciedlają mechanizmy stosowane w językach wysokiego poziomu (tablice, struktury).

Ponieważ rejestry służą do przechowywania wyników pośrednich, a nie danych programu, mogą one być zamazywane podczas kolejnych faz obliczeń.

Z tego powodu dominującą grupę instrukcji stanowią instrukcje dwuargumentowe, w których wynik zastępuj jeden z argumentów źródłowych.

Złożoność poszczególnych instrukcji CISC powoduje, że jednostka wykonawcza jest skomplikowana, a wykonanie instrukcji – wielofazowe.

Podejście **RISC** powstało stosunkowo niedawno. Stanowiło ono sposób na budowę szybkiego procesora o prostej strukturze.

Praktycznie wszystkie architektury opracowane po 1985 roku są architekturami klasy RISC.

W podejściu RISC zakłada się, że skalarne obiekty lokalne są przechowywane w rejestrach procesora. Tym samym odwołania do pamięci ograniczają się do dostępów do danych strukturalnych oraz przeładowywania ramki stosu podczas przekazywania sterowania pomiędzy procedurami.

Duży zestaw rejestrów mieści co najmniej dane jednej procedury. Aby nie niszczyć danych podczas operacji, instrukcje specyfikują oddzielnie argumenty źródłowe i przeznaczenia. Ograniczona liczba odwołań do pamięci umożliwia redukcję liczby i złożoności dostępnych trybów adresowania.

Prosta budowa procesora wynikająca z prostoty instrukcji umożliwia uzyskanie wysokiej wydajności dzięki wysokiej częstotliwości pracy. Duża wydajność procesora pozwala z kolei na syntezę dowolnie złożonych operacji i trybów adresowania na drodze programowej.

Ponieważ dane są przechowywane w rejestrach – instrukcje operujące na danych korzystają wyłącznie z rejestrów i stałych natychmiastowych. Dzięki temu długość obrazu binarnego instrukcji jest ograniczona.

Trochę więcej o x86

Architektura x86 (IA-32) jest dość nietypową, lecz najbardziej obecnie popularną architekturą CISC. Stało się tak dzięki zastosowaniu przez firmę IBM w 1980 roku procesorów tej rodziny w komputerach IBM PC.

Architektura x86 przechodziła szereg ewolucji i obecnie została ulepszona i rozszerzona do 64 bitów przez firmę AMD (AMD64).

Dalsze informacje i prezentowane przykłady będą dotyczyły tzw. płaskiego modelu pamięci, z liniowym adresowaniem i 32-bitowym adresem.

Operacje na danych w x86, podobnie jak w większości innych architektur CISC, mogą być prowadzone na słowach o długościach 8, 16 lub 32 bitów.

Posiada **mały zestaw ośmiu rejestrów uniwersalnych** (EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI), rejestr stanu (EFLAGS), licznik instrukcji (EIP).

Operacje warunkowe wykorzystują **model ze znacznikami** (6 znaczników). Wszystkie znaczniki są ustawiane przez dwuargumentowe instrukcje arytmetyczne i logiczne.

Znaczniki nie są zmieniane przez instrukcje przesłań.

Poza **jednostką stałopozycyjną**, procesory rodziny x86 posiadają **jednostkę zmiennopozycyjną** x87, wprowadzoną jeszcze w latach 70 XX w. i zbudowaną na jej bazie jednostkę wektorową MMX/3DNow! oraz nowszą jednostkę zmiennopozycyjną i wektorową **SSE**.

x87 powoli wychodzi z użycia, procesory są w nią wyposażone dla zachowania zgodności binarnej oprogramowania ze starszymi modelami.

Nowe oprogramowanie korzysta z jednostki SSE, a sama jednostka jest intensywnie rozwijana poprzez rozszerzanie listy instrukcji i obsługę nowych formatów danych.

Kolejne wersje jednostki są oznaczane jako SSE2, SSE3, SSSE3 i SSE4.

W trybie 32-bitowym **adres jest złożony z czterech składników**:
$R_b + R_i \cdot S + d$:

- R_b – dowolny rejestr bazowy,
- R_i – dowolny indeksowy, dowolny oprócz ESP,
- S – skala: 1, 2, 4 lub 8,
- d – przemieszczenie lub adres absolutny.

Każdy ze składników jest opcjonalny, jeden musi wystąpić (jeśli tylko d, to jest to adres absolutny.

prefix	kod operacji	ModRM	SIB	przemieszczenie	stała
0 . . . 4 B	0 . . . 3 B	brak, 1 B	brak, 1 B	0, 1, 2, 4 B	0, 1, 2, 4 B

Szczególną cechą zapisu binarnego instrukcji w x86 jest obecność **prefiksów** – jednobajtowych kodów operacyjnych nie reprezentujących żadnych instrukcji, lecz modyfikujących sposób dekodowania przez procesor następującej po prefiksach instrukcji.

Proste tryby adresowania, w tym rejestrowe pośrednie, są specyfikowane przez bajt ModRM.

Tryby indeksowe wymagają następnego bajtu – SIB.

Pełny opis wszystkich instrukcji można znaleźć w dokumentach *IA-32 Intel Architecture Software Developer's Manual* lub *AMD64 Architecture Programmer's Manual*.

Wybrane grupy instrukcji:

- przesłania,
- instrukcje arytmetyczne i logiczne,
- przesunięcia i rotacje bitowe,
- mnożenie i dzielenie,
- skoki,
- instrukcje wspomagające.

Instrukcje przesłań:

- `MOV` – przesłanie,
- `XCHG` – wymiana (przesłanie dwukierunkowe),
- `MOVSX` – przesłanie dalej z rozszerzeniem bitu znaku (np. `MOVSX EAX, BL`, dana 8-bitowa z BL jest rozszerzana do 32 bitów w EAX),
- `MOVSX` – przesłanie dalej z rozszerzeniem zerami (np. `MOVZX EAX, AL`, dana 8-bitowa z AL jest rozszerzana zerami do 32 bitów w EAX).

Instrukcje arytmetyczne i logiczne:

- INC, DEC – inkrementacja, dekrementacja,
- NOT, NEG – negacja bitów, zmiana znaku,
- ADD, ADC – dodawanie zwykłe i z przeniesieniem wchodzącym,
- SUB, SBB – odejmowanie zwykłe i z pożyczką wchodzącą,
- CMP – porównanie (odejmowanie bez zapisu wyników, ustawienie znaczników),
- AND, OR, XOR – iloczyn, suma i różnica symetryczna (bitowo),
- TEST – iloczyn logiczny (bez zapisu wynikow, ustawienie znaczników).

Instrukcje przesunięć i rotacji:

- SHL, SHR – przesunięcie w lewo i prawo z dopełnieniem zerami,
- SAR – przesunięcie *arytmetyczne* w prawo z kopiowaniem bitu znaku (realizacja dzielenia liczb ze znakiem przez potęgi liczby 2,
- ROL, ROR – rotacja w lewo i prawo,
- RCL, RCR – rotacja w lewo i prawo poprzez bity przeniesienia,
- SHLC, SHRD – przesunięcie dwóch słów w lewo i prawo z zapisem bardziej/mniej znaczącego słowa wyniku.

Instrukcje mnożenia:

- Postaci:
  - jednoargumentowa – argumenty domyślne, wynik dwa razy dłuższy od argumentów,
  - dwuargumentowa – rejestr, rejestr lub pamięć, wynik o długości argumentów,
  - trzyargumentowa – rejestr, rejestr lub pamięć, stała, wynik o długości argumentów,
- Formy:
  - `MUL` – bez znaku,
  - `IMUL` – ze znakiem.

Instrukcje dzielenia:

- Jedna postać – jednoargumentowa, argumenty domyślne, dzielna dwa razy dłuższa od dzielnika,
- Dwie formy: `DIV` – bez znaku i `IDIV` – ze znakiem.

Skoki:

- `JMP`, `CALL`, `Jxx` – warunkowe i bezwarunkowe,
- `RET`, `RET n` – powroty z procedur, i ze zdjęciem argumentów,
- `CALL`, `JMP` – bezwarunkowe ze zmiennym adresem docelowym.

Instrukcje wspomagające:

- `ENTER 0, n` – prolog procedury (`PUSH EBP; MOV EBP, ESP; SUB ESP, n`),
- `LEAVE` – epilog procedury (`MOV ESP, EBP; POP EBP`),
- `BOUND r, m` – sprawdzenie zakresu zmiennej.

Architektura MIPS32

- Klasyczna prosta architektura RISC.
- Instrukcje o stałej długości 32-bitów:
  - format R – instrukcje arytmetyczna i logiczne z argumentami w rejestrze, skok i skok ze śladem z adresem w rejestrze,
  - format I – instrukcje arytmetyczne i logiczne z argumentem natychmiastowym, instrukcje z wymiany z pamięcią, skoki warunkowe,
  - format J – skoki i skoki ze śladem z długim adresem natychmiastowym.
- 32 rejestry stałopozycyjne.
- 2 rejestry jednostki mnożąco-dzielącej (HI, LO).
- Licznik instrukcji PC.
- 32 rejestry zmiennopozycyjne.
- brak sprzętowej realizacji stosu (operacje syntezowane).

Architektura ARM

- Architektura RISC, 32 bity.
- Wprowadzona jako 16-bitowa.
- Bardzo rozpowszechniona w zastosowaniach wbudowanych (telefony komórkowe, tablety, osprzęt sieciowy).
- 16 rejestrów, dodatkowe rejestry systemowe.
- Model operacji warunkowych ze znacznikami.
- warunkowe wykonanie niemal wszystkich instrukcji.
- Dostępne operacje stosowe.
- Współczesne procesowy ARM często mają drugi, alternatywny zestaw instrukcji 16-bitowych (Thumb) z ograniczonym dostępem do części rejestrów.

Procesor jednocyklowy

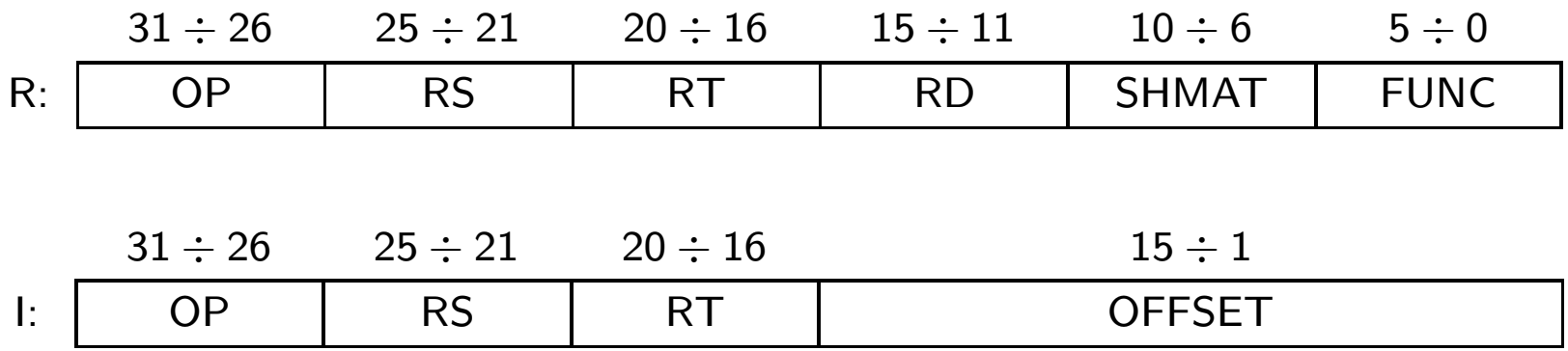
Procesor o prostym modelu jednocyklowym, można zbudować w postaci układu sekwencyjnego. Zmienia on swój stan tylko jeden raz, na końcu instrukcji.

Wykonanie instrukcji odbywać się będzie w układzie kombinacyjnym.

Założenia:

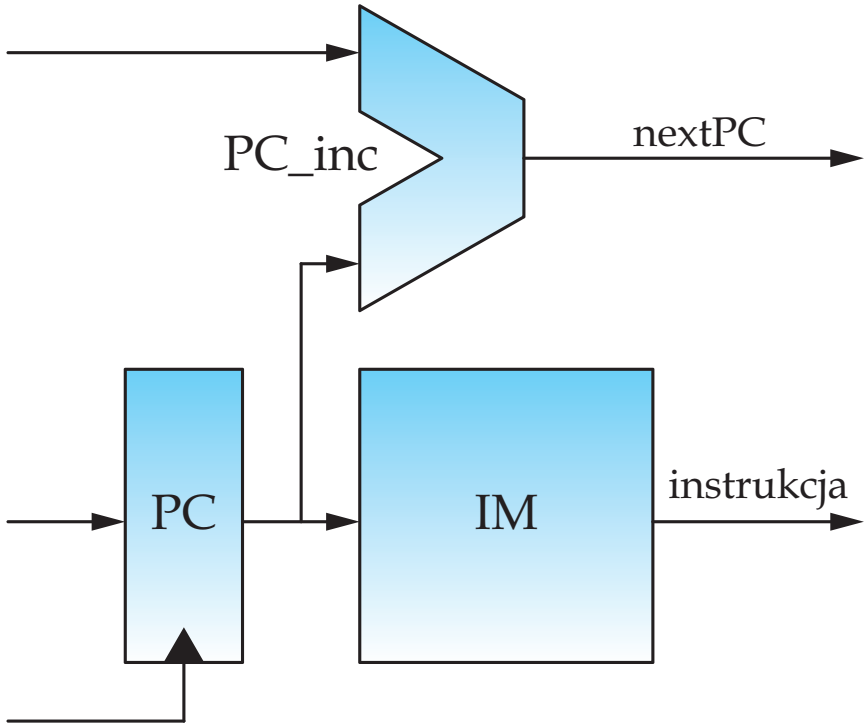
- uproszczony model RISC,
- architektura typu Harvard,
- instrukcje o równej długości,
- bajtowe adresowanie pamięci,
- wymuszone wyrównanie naturalne danych i instrukcji.

Modelowy procesor ma możliwość wykonywania instrukcji w dwóch formatach – R oraz I. W obu formatach występują pola głównego kodu operacyjnego i dwa numery rejestrów.



- OP – główny kod operacyjny.
- RS – rejestr źródłowy.
- RT – rejestr źródłowy lub przeznaczenia.
- RD – rejestr przeznaczenia.
- SHAMT – wartość przesunięcia bitowego dla instrukcji przesunięć.
- FUNC – rozszeżenie kodu operacyjnego.
- OFFSET – stała, argument operacji, przemieszczenie pamięci lub przemieszczenie skoku.

Pobranie instrukcji

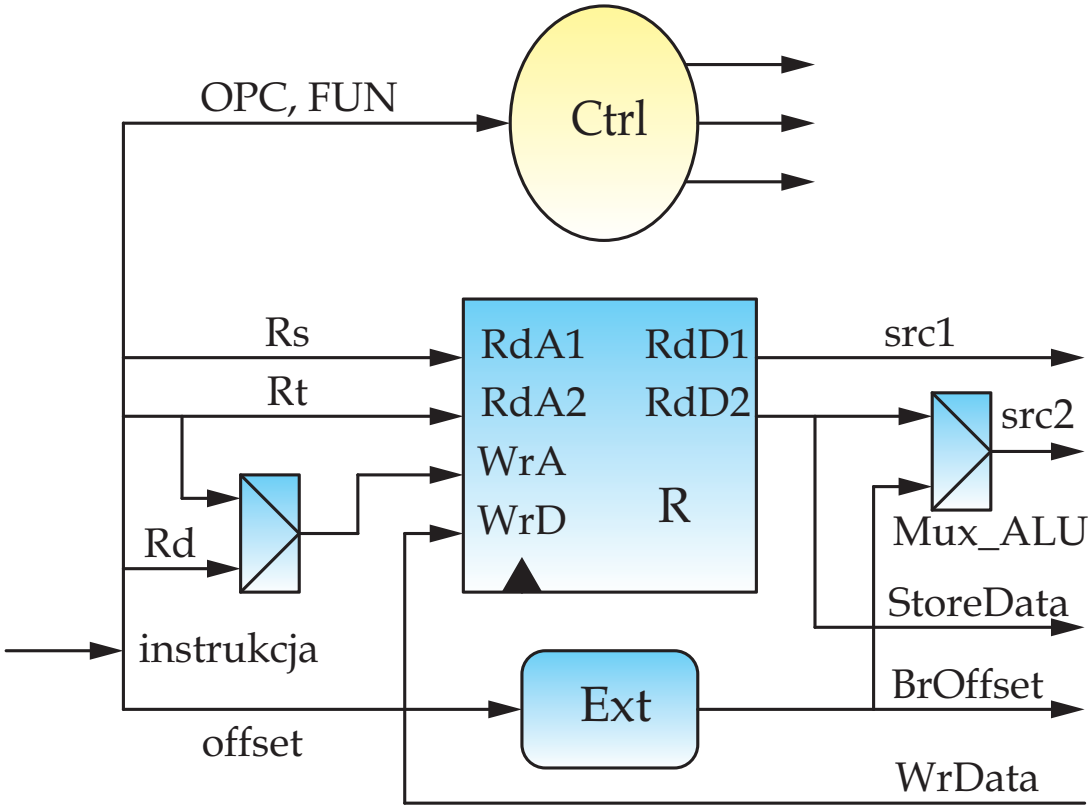


- PC – licznik instrukcji.
- Inc PC – inkrementer licznika instrukcji.
- IM – pamięć instrukcji.

- Licznik instrukcji jest zrealizowany jako rejestr typu D.
- Zawartość licznika instrukcji jest podawana na:
  - wejście adresowe pamięci programu IM,
  - wejście inkrementera Inc PC generującego adres następnej instrukcji.
- Na wyjściu pamięci pojawia się binarny obraz instrukcji.
- Instrukcja zawiera:
  - kod operacyjny instrukcji,
  - numery rejestrów,
  - stałą natychmiastową, używaną jako argument operacji lub przemieszczenie adresu,
  - wartość przesunięcia bitowego.

Układ sterujący

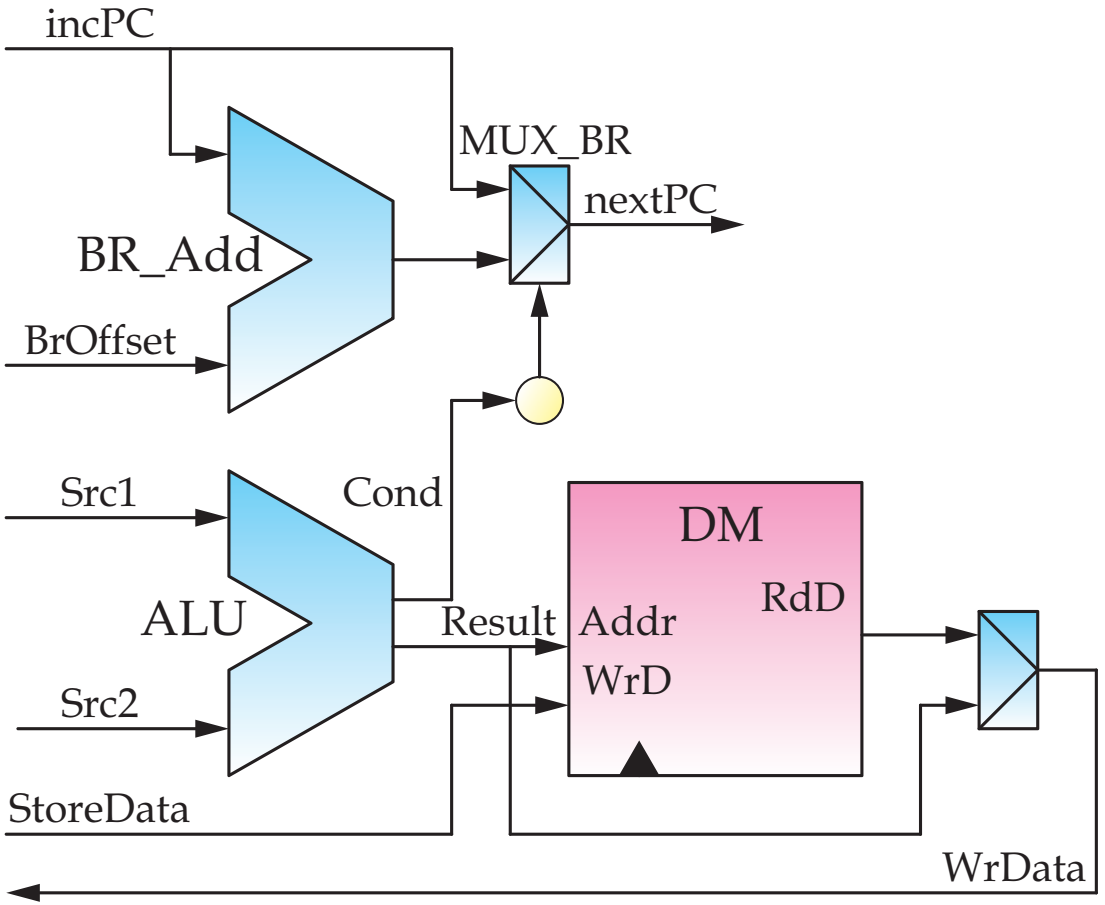
Układ sterujący jest wielowyjściowym układem kombinacyjnym. Generuje sygnały sterujące wszystkimi częściami procesora.



- OPC, FUN, Rt, Rd, offset – pola obrazu binarnego instrukcji.
- RdA1, RdA2 – adresy odczytu zestawu rejestrów.
- WrA – numer zapisywanego rejestru.
- WrD – dane zapisywane do rejestru.
- RdD1, RdD2 – dane odczytywane z rejestrów.
- Mux_ALU – multiplekser wyboru drugiego argumentu źródłowego.
- BrOffset – przemieszczenie skoku.
- StoreData – dane zapisywane do pamięci.

- Na wejście układu sterującego jest podawana część słowa instrukcji:
  - główny kod operacyjny,
  - pole FUN (tylko dla instrukcji w formacie R).
- Układ sterujący generuje sygnały sterujące pozostałymi częściami procesora, w tym między innym:
  - sygnały sterujące multiplekserami,
  - sygnały zezwolenia na odczyt i zapis pamięci danych,
  - sygnały zezwolenia na zapis rejestru docelowego,
  - kod operacji dla jednostki arytmetyczno logicznej,
  - sygnał zezwolenia na skok warunkowy,
  - sygnał sterujący pracą układu rozszerzenia danej natychmiastowej.

ALU, układ skoków, pamięć danych



- Br_Add – sumator adresów docelowych skoków.
- MUX_BR – multiplekser skoków warunkowych.
- ALU – jednostka arytmetyczno-logiczna.
- Result – wynik operacji.
- Cond – warunek skoku.
- DM – pamięć danych.
- WrData – dane zapisywane do rejestru.

- Jednostka arytmetyczno-logiczna wykonuje na argumentach wejściowych operację określoną przez wykonywaną instrukcję.
- sumator skoków generuje potencjalny adres docelowy skoku warunkowego.
- Multiplekser skoków względnych wybiera adres następnej instrukcji.
- Moduł wymianu danych z pamięcią jest bezczynny podczas instrukcji innych niż ładowanie i składowanie danych.
- Wartość z wyjścia ALU jest używana jako adres odczytu lub zapisu pamięci.
- Multiplekser wyboru służy do wybrania wartości, która ma być zapisana do rejestru (ALU lub pamięć). Po przepropagowaniu sygnałów przez wszystkie układy procesora następuje zakończenie wykonywania instrukcji (sterowane zboczem sygnału zegarowego):
  - zapis do PC wartości z wyjścia multipleksera skoków,
  - zapis wyniku operacji lub danej odczytanej z pamięci do rejestru docelowego, zapis danej odczytanej z rejestru do pamięci.

Ochrona zasobów

Współczesne komputery pracują z wieloprocesowymi i wieloużytkownikowymi systemami operacyjnymi. Procesy nie mogą sobie nawzajem przeszkadzać ani się szpiegować.

Konieczne jest wykorzystywanie **mechanizmów zabezpieczających** przed błędami lub świadomymi akcjami programów w celu zagwarantowania poprawnej pracy pozostałych procesów.

Zasoby podlegające ochronie:

- **procesor** – proces nie może zmonopolizować czasu procesora,
- **pamięć** – proces może mieć dostęp tylko do przydzielonej mu pamięci, nie może się odwoływać do pamięci w nieprawidłowy sposób,
- **kłady wejścia-wyjścia** – procesy nie powinny przeszkadzać sobie w dostępie do urządzeń zewnętrznych, w praktyce uniemożliwia się bezpośrednie odwołania do urządzeń.

Realizacja mechanizmów ochrony wymaga wyodrębnienia **poziomów zaufania**. Procesy użytkowe pracują na **poziomie użytkownika**, na którym podlegają one zasadom określonym przez system operacyjny.

System operacyjny pracuje na **poziomie systemowym** (pełny dostęp do wszystkich zasobów), na których istnieje możliwość określenia zasad ochrony obowiązujących procesy użytkowe.

Zakłada się, że oprogramowanie działające na poziomie systemowe jest **zaufane**, czyli wolne od błędów.

W x86 istnieją cztery poziomy zaufania: 0 – **jądro systemu operacyjnego**; 1, 2 – dodatkowe poziomy systemowe; 3 – **poziom użytkownika**.

Informacja o bieżącym poziomie zaufania jest przechowywana w niewidocznym programowo rejestrze deskryptora segmentu kodu **CPL** (*Current Privilege Level*).

Każda próba naruszenia zasad ochrony jest wykrywana i systemowi operacyjnemu jest zgłaszany **wyjątek**, a system operacyjny ma możliwość zakończenia wykonywania procesu naruszającego zasady ochrony i usunięcia go.

Zarządzanie pamięcią

Funkcje systemu zarządzania pamięcią:

- **sprzętowa relokacja** – zorganizowanie przestrzeni adresowej dla wszystkich procesów,
- **ochrona** – zabezpieczenie przed odczytem lub modyfikacją kodu i danych nie należących do danego procesu,
- **dynamiczna alokacja i dealokacja** – powiększanie i zmniejszane rozmiaru przestrzeni adresowej procesu w czasie pracy,
- **wirtualizacja** – uniezależnienie dostępności pamięci od fizycznej konfiguracji komputera i bieżącego stopnia wykorzystania pamięci.

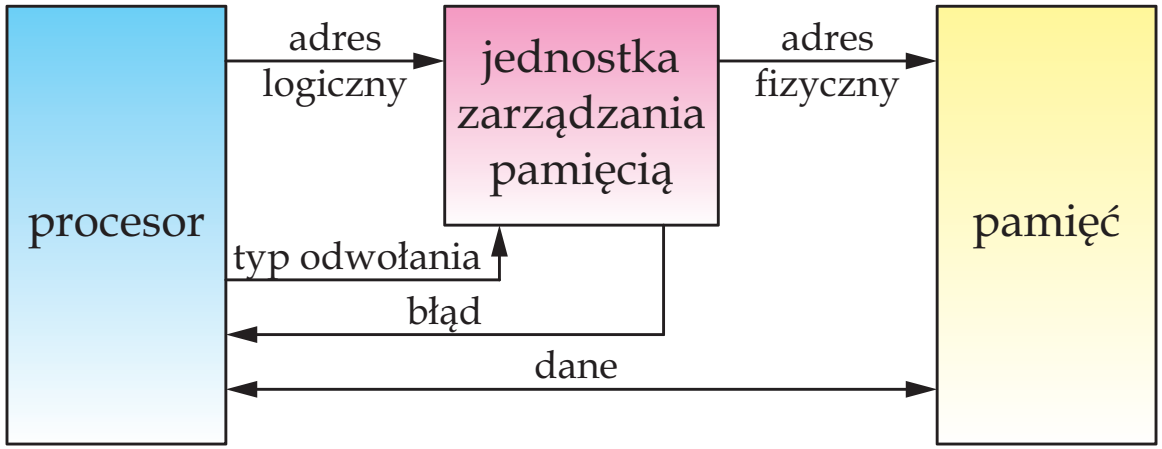
Zarządzanie pamięcią wymaga umieszczenia pomiędzy jednostką wykonawczą procesora i pamięcią dodatkowego bloku funkcjonalnego – **jednostki zarządzania pamięcią**.

Adres wytwarzany przez jednostkę wykonawczą zgodnie z użytkowym modelem programowym procesora jest nazywany **adresem logicznym** lub **wirtualnym**.

Adres ten trafia do jednostki zarządzania pamięcią sprawdza poprawność odwołania i generuje adres podawany na wejście pamięci fizycznej, zwany **adresem fizycznym**.

Techniki relokacji:

- prosta relokacja,
- segmentacja,
- stronicowanie.


procesor
adres
logiczny
jednostka
zarządzania
pamięcią
adres
fizyczny
pamięć
typ odwołania
błąd
dane

Prosta relokacja

Jednostka zarządzania pamięcią zawiera dwa rejestry: **fizyczny adres bazowy** obszaru pamięci przypisany procesowi i rejestr zawierający **adres fizyczny końca obszaru** przydzielonego procesowi.

Adres logiczny jest sumowany z zawartością rejestru bazowego, uzyskany w ten sposób adres fizyczny jest porównywany z zawartością rejestru limitu.

Każdy proces ma własne wartości rejestrów bazy i limitu. Podczas przełączania procesów następuje ładowanie nowych wartości do rejestrów.

Alokacja pamięci jest realizowana przez rozszerzenie obszaru zajmowanego przez proces. Obszar może być rozszerzony, jeśli obszar położony bezpośrednio za nim nie jest zajęty przez inny proces.

W przypadku zajętości system operacyjny przemieszcza procesy w pamięci w celu uzyskania wolnego obszaru pamięci o odpowiedniej długości.

Może to powodować powstawanie małych pustych obszarów pamięci – **fragmentacja zewnętrzna**.

Segmentacja

Segmentacja jest uogólnieniem metody relokacji. Przestrzeń adresowa procesu jest podzielona w sposób wynikający z jej logicznej struktury: kod, dane statyczne, stos, sterta.

Obszary te (**segmenty**) mogą mieć różne długości, segmentacja jest zwykle widoczna na poziomie modelu programowego użytkownika.

Logiczna przestrzeń adresowa jest **dwuwymiarowa**, adres logiczny składa się z dwóch części: identyfikatora segmentu i adresu wewnątrzsegmentowego (*offset*).

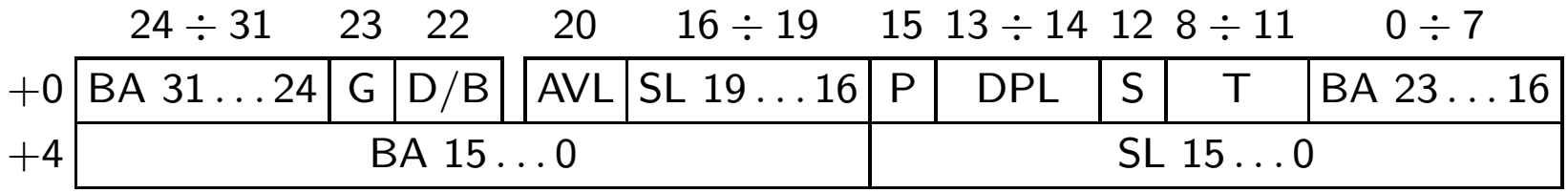
Jednostka transluje adresy w przestrzeni dwuwymiarowej do adresów w przestrzeni jednowymiarowej – **liniowej** (adres liniowy).

Jednostka segmentacji otrzymuje dodatkowo informacje o **typie segmentu** i **poziomie zaufania** dostępu, rozmiarze segmentu, adresie bazowym (**deskryptor**). Może zgłaszać błąd dostępu do pamięci i uniemożliwić odwołanie do pamięci.

W przypadku braku błędów jednostka segmentacji generuje adres liniowy odwołania poprzez zsumowanie adresu bazowego z adresem wewnątrzsegmentowym.

Alokacja pamięci może następować na dwa sposoby: rozszerzenie istniejącego segmentu sterty i alokacja nowego segmentu sterty. Rozszerzenie istniejącego segmentu zwykle wymaga przemieszczania tego lub innych segmentów.

Taki sposób adresowania jest możliwy w modelu x86.



- BA – Base address.
- SL – Segment limit.
- G – granularność rozmiaru segmentu (1B/4kB).
- D/B – domyślny rozmiar segmentu (16b/32b).
- AVL – pole dostępne dla oprogramowania.
- P – znacznik ważności deskryptora.
- DPL – poziom zaufania deskryptora.
- S – deskryptor systemowy (specjalny).
- T – typ despryptora (np. segment kodu, segment danych).

Stronicowanie

Stronicowanie polega na podziale logicznej i fizycznej przestrzeni adresowej na bloki o długości wyrażonej potęgą liczby 2, naturalnie wyrównane. Podział taki nie ma związku z logiczna strukturą przestrzeni adresowej.

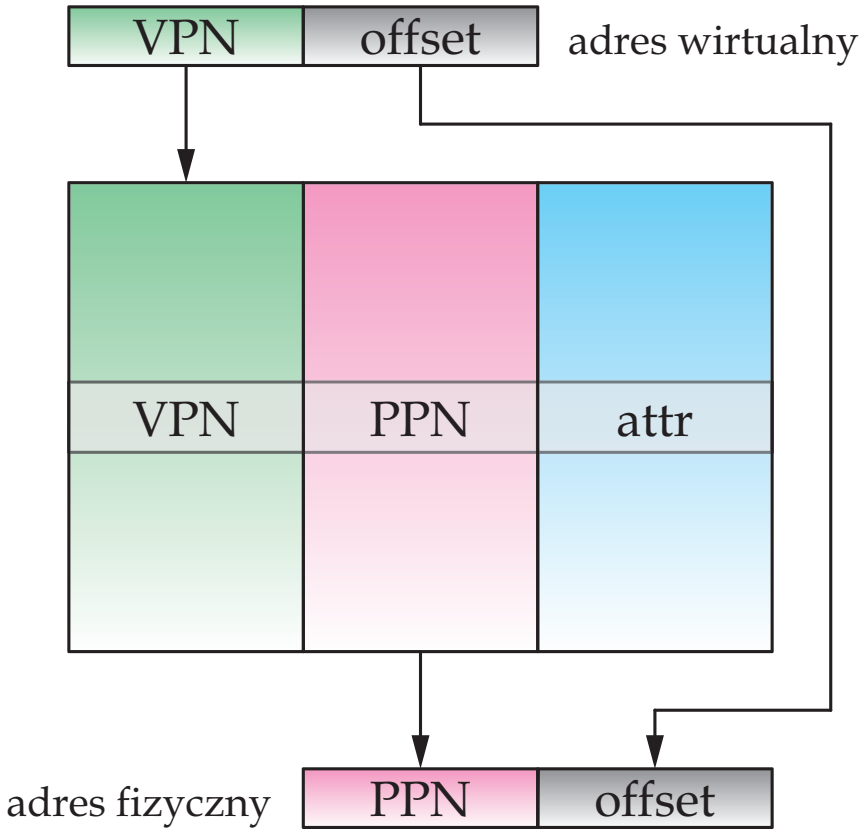
Bloki przestrzeni logicznej nazywane są **stronami logicznymi** lub **wirtualnymi**. Bloki przestrzeni fizycznej nazywane są **stronami fizycznymi** lub **ramkami stron**.

Obecnie strona może mieć kilka rozmiarów, stronicowanie nie jest widoczne w modelu programowym.

Typowy rozmiar strony wynosi od 4 do 8 kB (czasem 16 kB). Translacja adresu polega na przyporządkowaniu stronie logicznej strony wirtualnej. Adres logiczny jest dzielony na dwa pola: **numer strony wirtualnej** (VPN) i **adres wewnątrzstronicowy** (offset). Podczsa translacko VPN zostaje zastąpiony przez numer strony fizycznej (PPN).

Podstawową strukturą sprzętową jednostki stronicowania jest bufor translacji (forma pamięci cache). Przechowuje pewna liczbę ostatnio używanych deskryptorów stron.

Adres fizyczny powstaje poprzez konkatenację fizycznego numeru strony i adresu wewnątrzstronicowego.


VPN
offset
adres wirtualny
VPN
PPN
attr
adres fizyczny
PPN
offset

Duża liczba stron umożliwia określenie zbioru roboczego, a strony chwilowo nieużywane mogą zostać przeniesione do pamięci masowej.

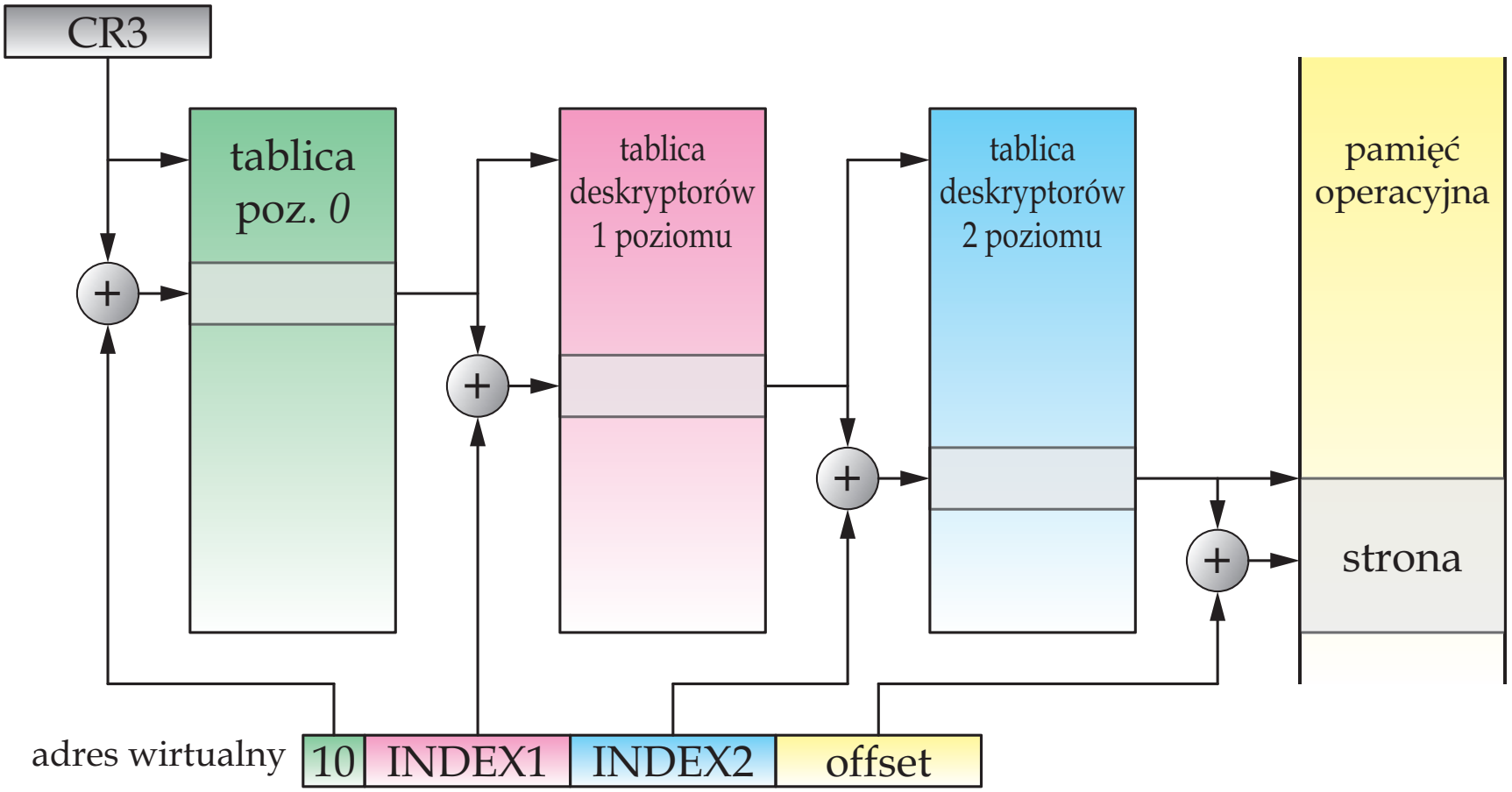
Stały rozmiar strony eliminuje fragmentacje pamięci operacyjnej.

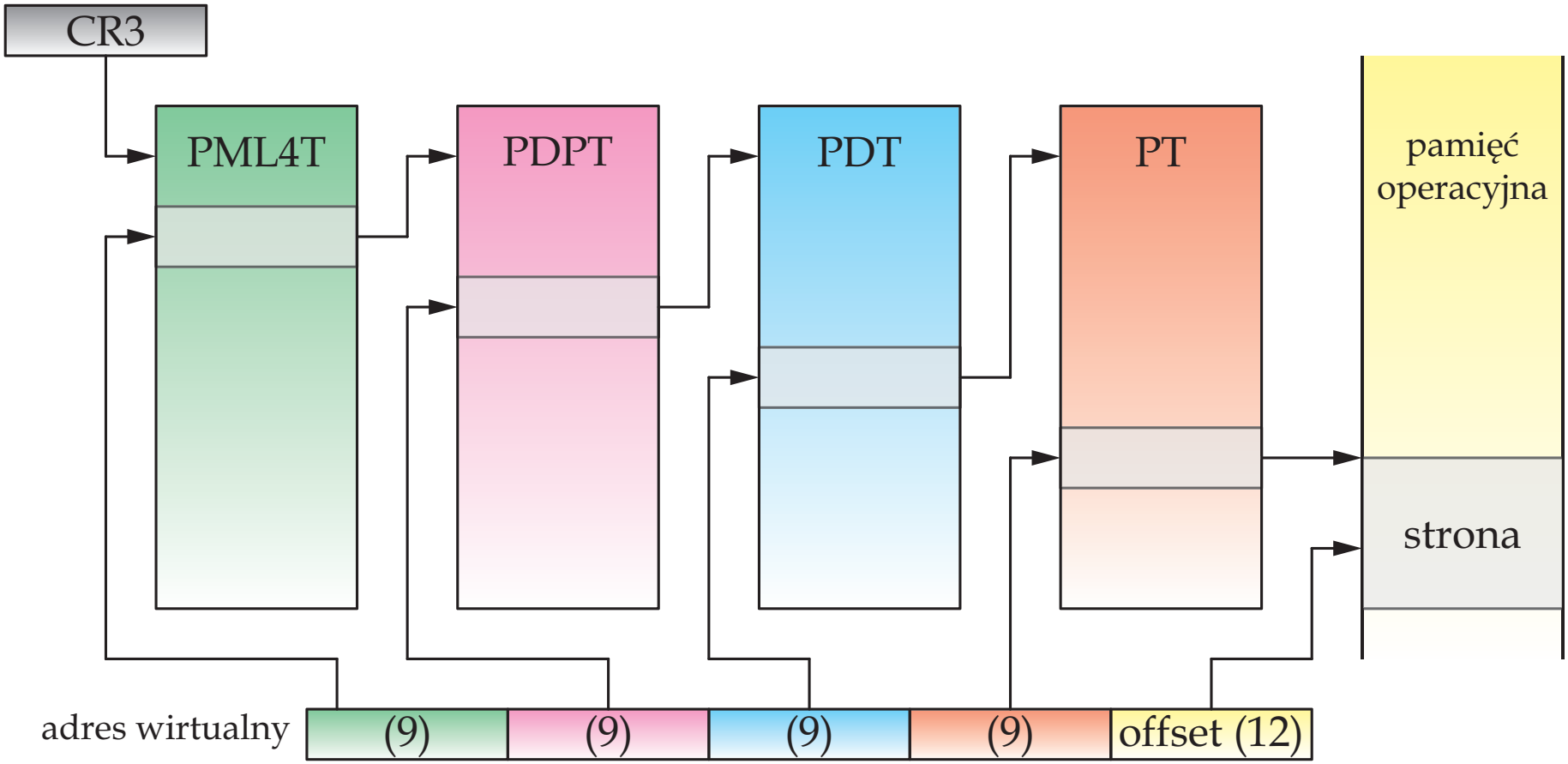
Stronicowanie zostało opracowane z myślą o implementacji systemu pamięci wirtualnej.

Obecnie przestrzeń adresowa procesu dzieli się przynajmniej na dwie części: użytkownika (pierwszego poziomu) i systemową (drugiego poziomu).

Część systemowa jest wspólna dla wszystkich procesów. Każdy proces ma własną tablicę pierwszego poziomu.

Obecnie stosuje się adresowanie o trzech (x86) lub nawet czterech poziomach (AMD64).


CR3
tablica
poz. 0
tablica
deskryptorów
1 poziomu
tablica
deskryptorów
2 poziomu
pamięć
operacyjna
strona
adres wirtualny
10
INDEX1
INDEX2
offset

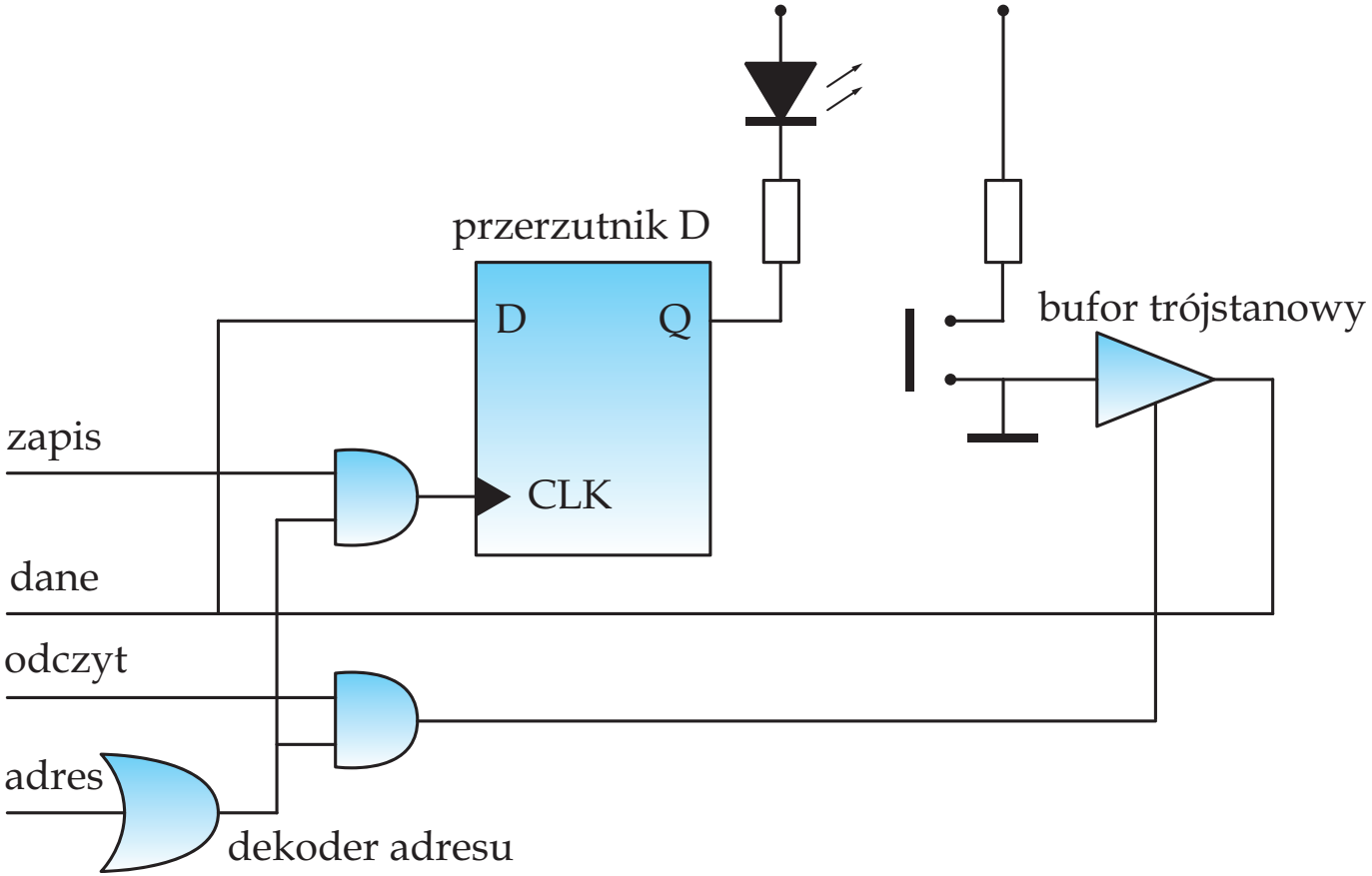

CR3
PML4T
PDPT
PDT
PT
pamięć operacyjna
strona
adres wirtualny
(9)
(9)
(9)
(9)
offset (12)

Wejście-wyjście

Port wejściowy lub **wyjściowy** jest w modelu programowym widziany tak samo lub podobnie, jak lokacja pamięci. W większości architektur porty są odwzorowane w przestrzeni adresowej pamięci.

W niektórych architekturach (x86), porty urządzeń zewnętrznych są dostępne w oddzielnej przestrzeni adresowej, na której operują specjalne instrukcje procesora – **instrukcje wejścia-wyjścia**.

W porównaniu ze współpracą procesora z pamięcią transmisja danych do i z urządzeń zewnętrznych jest o tyle bardziej skomplikowana, że nie może zachodzić w dowolnym momencie, lecz jedynie wtedy, kiedy urządzenie jest gotowe do przyjęcia danej lub do udostępnienia danej komputerowi.


przerzutnik D
D
Q
CLK
bufor trójstanowy
zapis
dane
odczyt
adres
dekoder adresu

Transmisja danych wymaga **synchronizacji**, czyli uwzględnienia stanu gotowości urządzenia.

Wciśnięto klawisz na klawiaturze i można odczytać kod klawisza, drukarka przyjęła znak do wydrukowania i jest gotowa na przyjęcie następnego.

Istnieją trzy metody synchronizacji, o zróżnicowanych kosztach implementacji i wydajności:

- aktywne oczekiwanie (*polling*),
- przerwania,
- bezpośredni dostęp do pamięci DMA (*Direct Memory Access*).

Metoda aktywnego oczekiwania nie wymaga żadnych nakładów sprzętowych.

Cała synchronizacja transmisji jest osiągana na drodze programowej, poprzez **testowanie** stanu sygnałów gotowości i **ustawianie** sygnałów sterujących transmisją danych.

Takie postępowanie angażuje czas procesora, rozwiązanie stosowane wyłącznie dla prostych systemów jednoprocesowych (proste sterowniki urządzeń).

Obsługa z użyciem przerwań korzysta z mechanizmu sytuacji wyjątkowych – zdarzeń obsługiwanych przez system operacyjny. Przejście urządzenia w stan gotowości powoduje zgłoszenie przerwania i system operacyjny odnotowuje gotowość urządzenia.

Proces zgłasza **żądanie transmisji** systemowi operacyjnemu i jeśli urządzenie jest gotowe dana jest przesyłana, jeśli urządzenie nie jest gotowe proces jest usypiany.

Procedura obsługi przerwania powoduje wznowienie wykonania procesu.

Główną zaletą obsługi przy użyciu przerwań jest możliwość zajęcia się przez procesor wykonywaniem innych czynności podczas oczekiwania na **gotowość urządzeń**, co jest niezbędne w systemach wieloprocesowych.

Występuje znaczna komplikacja oprogramowania obsługującym przerwania w systemie operacyjnym. System operacyjny wykonuje dużo operacji na procesach.

Jeśli urządzenie transmituje dane szybko, a co za tym idzie – często zgłasza przerwania, może się okazać, że procesor traci dużo czasu na **przełączanie kontekstu** wynikające z obsługi przerwań oraz na czynności systemowe wynikające ze zmian stanu urządzeń, w tym m.in. zmiany stanu procesów oczekujących na te urządzenia.

Obsługa z użyciem przerwań może się w tym przypadku stać bardziej czasochłonna, niż aktywne oczekiwanie.

Najbardziej efektywnym sposobem obsługi urządzeń zewnętrznych jest **bezpośredni dostęp do pamięci**. Wymaga on sprzętowej realizacji specjalnego modułu – sterownika bezpośredniego dostępu do pamięci.

Transmisja danych pomiędzy urządzeniem i pamięcią operacyjną następuje **bez użycia procesora**.

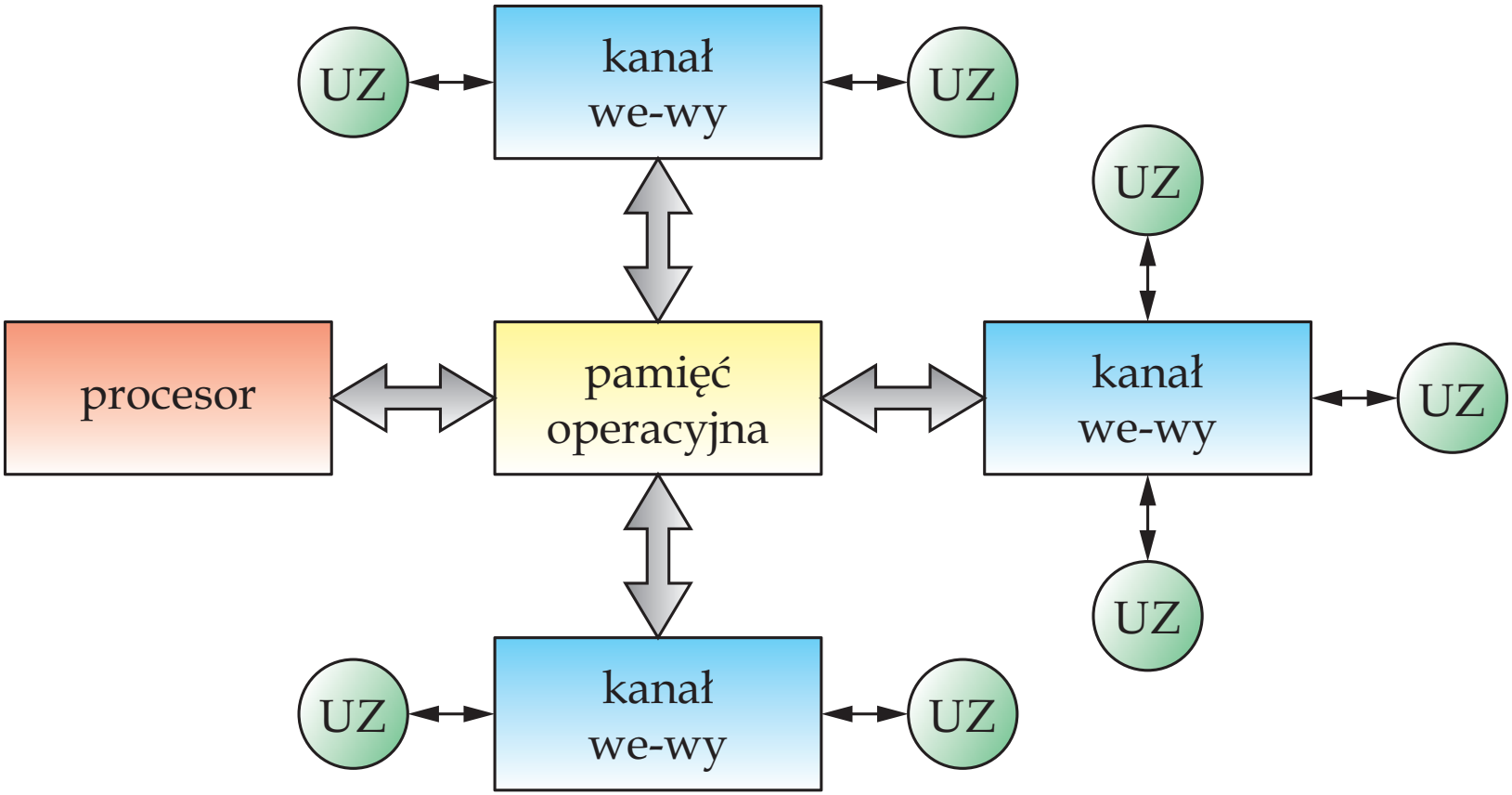
Przy korzystaniu z bezpośredniego dostępu do pamięci przerwanie jest zgłaszane do procesora po przetransmitowaniu całego bloku danych. Narzut czasowy na obsługę programową transmisji jest więc bardzo mały.

Jest to preferowana metoda współpracy z szybkimi urządzeniami zewnętrznymi, stosowana szeroko również w komputerach osobistych, sterowniki są zwykle wbudowane do konkretnego urządzenia – *bus matering*.

Typowe struktury komputerów od ich początków do współczesności:

- Architektura pamięciowo-centryczna.
- Architektura szynowa.
- Architektura wieloszynowa.
- Współczesna architektura z połączeniami punkt-punkt.

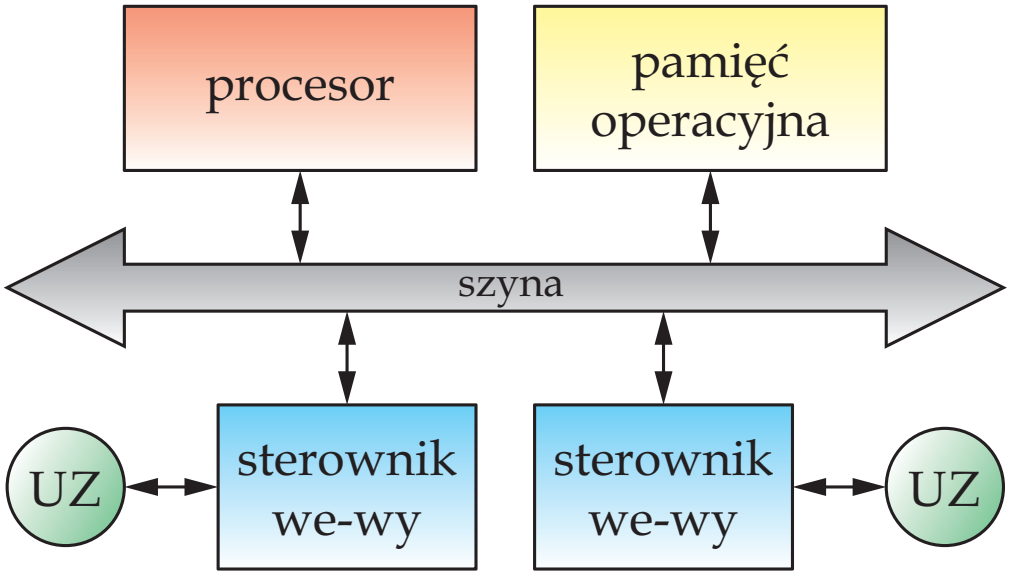
Architektura pamięciowo-centryczna



Architektura charakterystyczna dla pierwszych budowanych komputerów. Centralnym elementem struktury jest pamięć, wyposażona w kilka portów, umożliwiających połączenie jej z kilkoma urządzeniami.

Urządzeniami tymi są: procesor (lub procesory) i tzw. kanały wejścia-wyjścia, czyli specjalizowane procesory transmitujące dane pomiędzy urządzeniami zewnętrznymi i pamięcią komputera.

Architektura szynowa



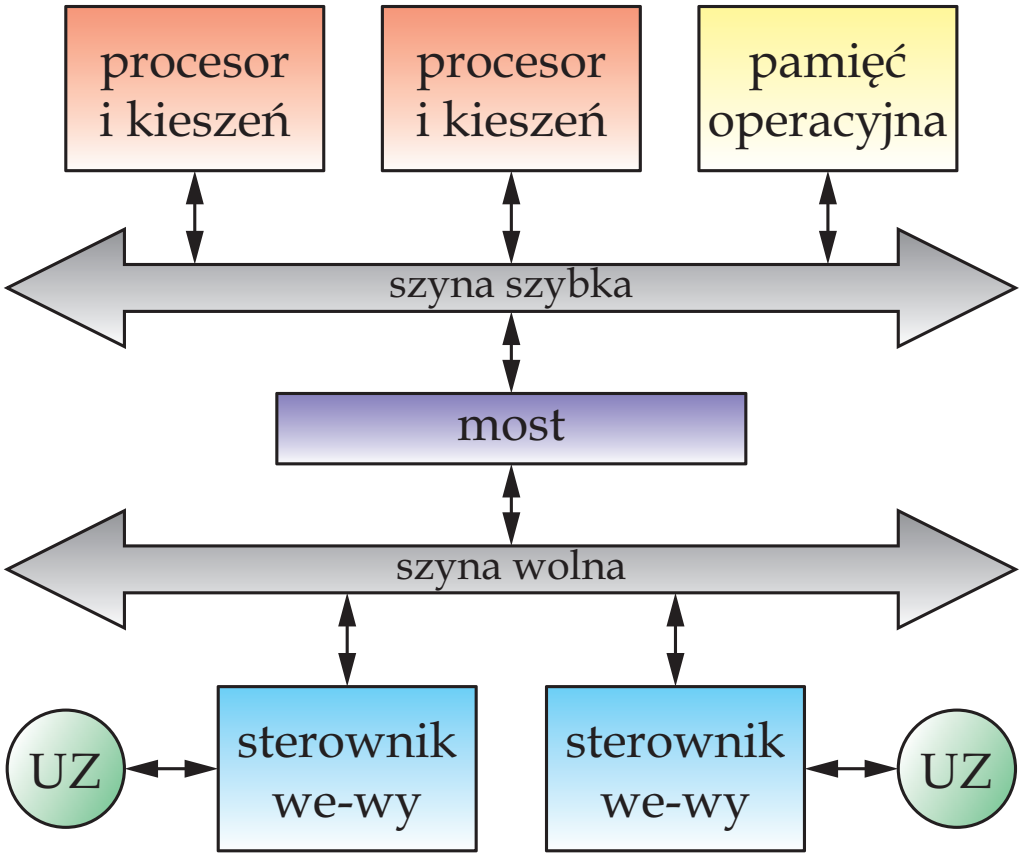
Struktura szynowa, wprowadzona w tzw. minikomputerach. Podstawowym elementem komputera jest tu szyna, czyli zespół połączeń pomiędzy modułami.

Komputer składa się z pewnej liczby modułów dołączonych do szyny – jednego lub kilku procesorów, bloków pamięci i sterowników wejścia-wyjścia. Pamięci i sterowniki urządzeń są widziane przez procesor w podobny sposób – jako lokacje w przestrzeni adresowej.

Architektura łatwa do rekonfiguracji i rozbudowy, stosunkowo tania, sterowniki wejścia-wyjścia widziane przez procesor tak samo jak pamięć.

Długość i struktura połączeń ograniczają szybkość transmisji (zjawiska falowe, rozproszona indukcyjność i pojemność), charakteryzuje się też dysproporcją wydajności powiększoną przez wolna transmisję danych na szynie.

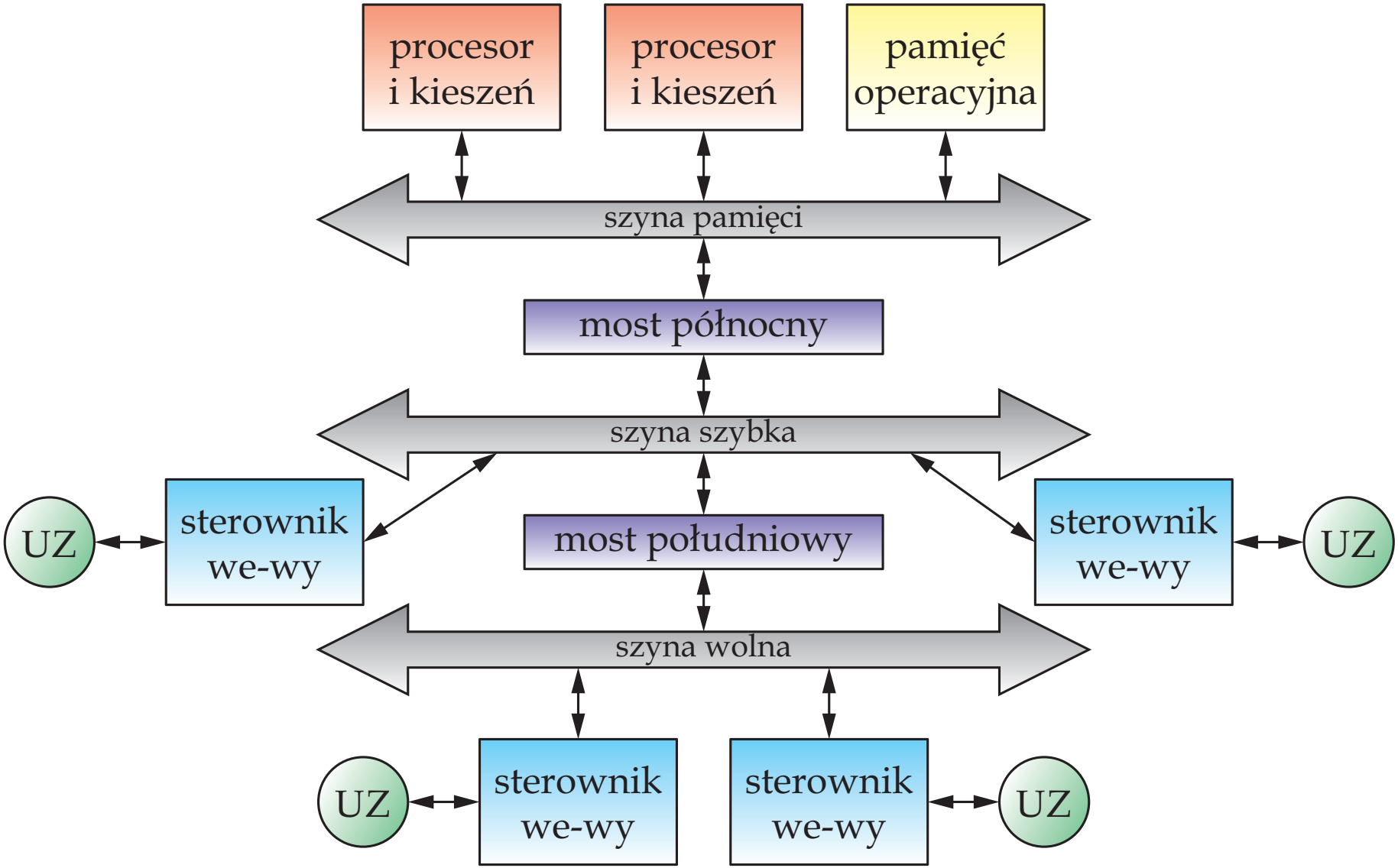
Architektura dwuszynowa



Architektura dwuszynowa zawiera dwie szyny: szybką, łączącą procesor lub procesory z pamięcią oraz wolną, do której podłączone są sterowniki urządzeń zewnętrznych. Obie szyny łączy układ mostu.

Logicznie obie szyny są widziane jak jedna szyna. Problemem jest to, ze niektóre urządzenia zewnętrzne wymagają bardzo szybkiej transmisji.

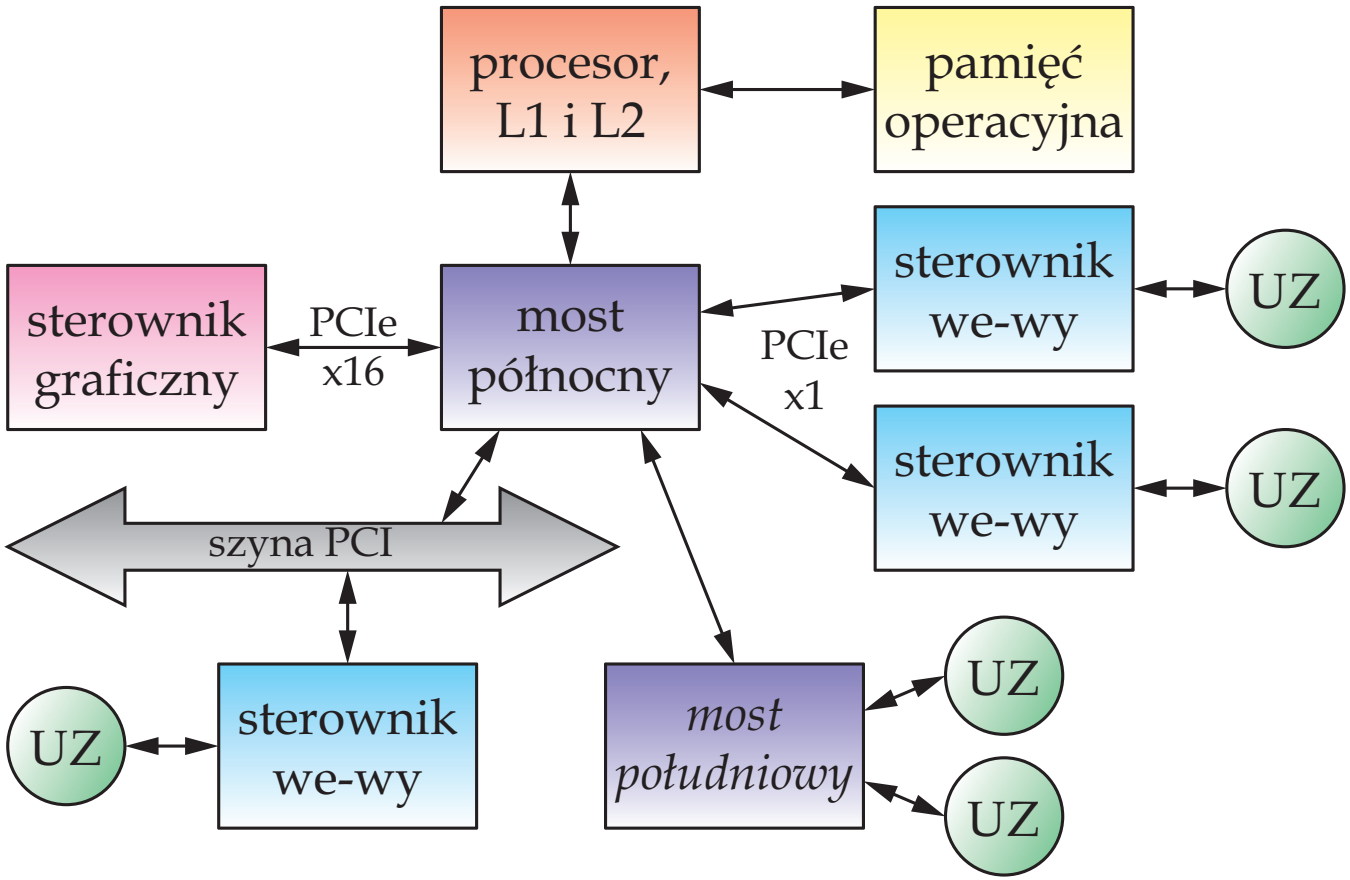
Architektura trójszynowa



W architekturze trójszynowej istnieją dwie szyny urządzeń zewnętrznych – szybka i wolna oraz dodatkowo szyna procesora i pamięci. Występują dwa mosty, zwane od swego położenia na schemacie blokowym odpowiednio *północnym* i *południowym*.

W praktyce most południowy zawiera sterowniki niektórych urządzeń a sterownik pamięci jest umieszczony w moście północnym. Problemem jest zbyt mało wydajna szyna szybka dla podsystemu graficznego.

Architektura z połączeniami punkt-punkt



We współczesnych komputerach **sterownik pamięci** umieszczony jest w **procesorze**. **Most północny** jest wyposażony w indywidualne łącza dla sterowników urządzeń zewnętrznych, zrealizowane w standardzie **PCI express**.

Most południowy jest zintegrowanym **sterownikiem urządzeń zewnętrznych**. Szyna **PCI** została zachowana w celu umożliwienia podłączenia starszych sterowników urządzeń. Jest ona przeznaczona do usunięcia.

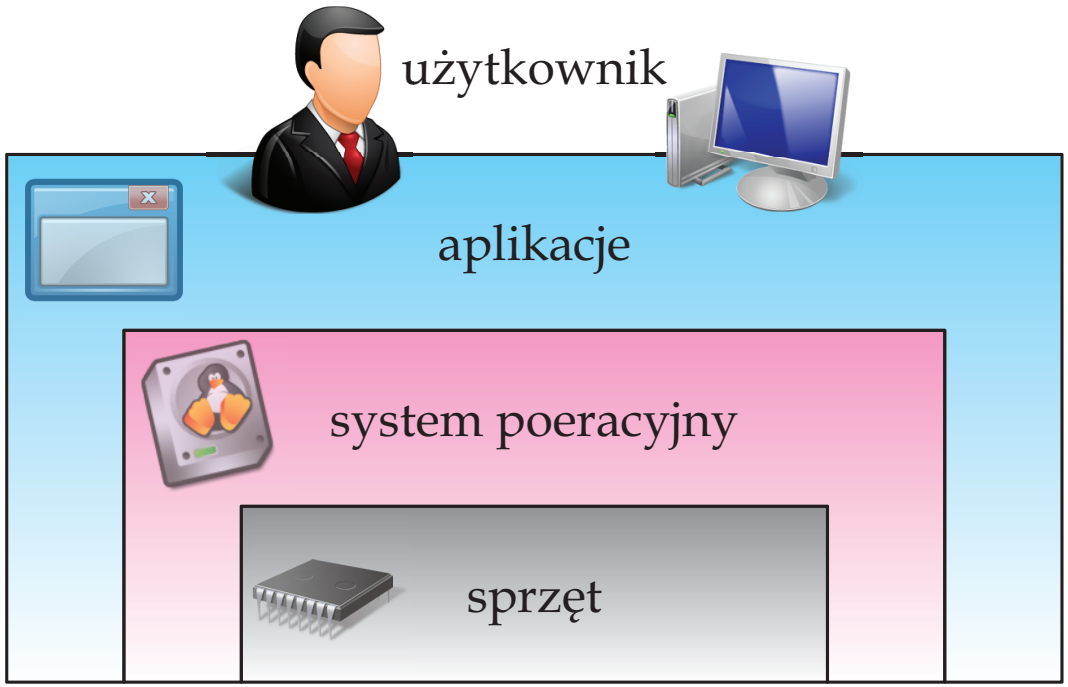
Pojęcie **systemu operacyjnego** jest trudne do zdefiniowania w zwartej formie. Krótki opis jest zbyt ogólny, żeby określić rolę i sposób działania tego specyficznego oprogramowania. Podobnie, trudne jest określenie elementów składowych systemu operacyjnego, czyli jednoznaczne oddzielenie oprogramowania systemowego od aplikacyjnego.

System operacyjny

System operacyjny jest warstwą oprogramowania operującą bezpośrednio na sprzęcie, której celem jest zarządzanie zasobami systemu komputerowego i stworzenie użytkownikowi środowiska łatwiejszego do zrozumienia i wykorzystania.

Należy zwrócić uwagę na dwie zasadnicze kwestie:

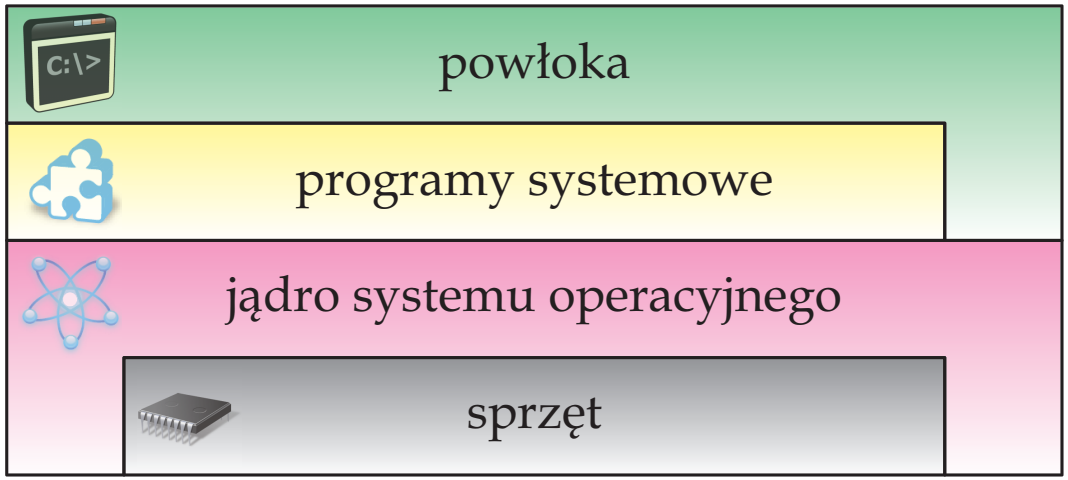
- zarządzanie zasobami systemu komputerowego,
- stworzenie środowiska dla użytkownika.



System operacyjny pośredniczy pomiędzy **użytkownikiem** a **sprzętem**, dostarczając **wygodnego środowiska** do **wykonywania programów**. Użytkownik końcowy korzysta z programów (aplikacji), na potrzeby których przydzielane są zasoby systemu komputerowego.

Przydziałem tym zarządza system operacyjny, dzięki czemu można uzyskać stosunkowo duży stopień niezależności programów od konkretnego sprzętu oraz odpowiedni poziom bezpieczeństwa i sprawności działania.

Nie ma precyzyjnego określenia, które składniki wchodzą w skład systemu operacyjnego jako jego części. Najczęściej akceptuje się definicję *marketingową*, zgodnie z którą to wszystko, co producent udostępnia w ramach zbioru oprogramowania nazywanego systemem operacyjnym, stanowi jego cześć (podejście takie było przyczyną wielu problemów prawnych firmy Microsoft).



W ogólnym przypadku w strukturze systemu operacyjnego wyróżnia się **jądro** oraz **programy systemowe**, które dostarczane są razem z systemem operacyjnym, ale nie stanowią integralnej części jądra. Jądro jest zbiorem **modułów**, które ukrywają szczegóły sprzętowej realizacji systemu komputerowego, udostępniając pewien zestaw **usług**, wykorzystywanych między innymi do implementacji programów systemowych. W dalszej części system operacyjny będzie rozumiany głównie jako jądro, i inne elementy oprogramowania integralnie związane z funkcjonowaniem jądra.

Z punktu widzenia kontaktu z użytkownikiem istotny jest **interpreter poleceń**, który może być częścią jądra lub programem systemowym (np. w systemach UNIX/Linux). Interpreter wykonuje pewne polecenia **wewnętrznie**, tzn. moduł lub program interpretera dostarcza implementacji tych poleceń. Jeśli interpreter nie może wykonać wewnętrznie jakiegoś polecenia, uruchamia odpowiedni program (polecenie **zewnętrzne**), jako odrębny proces.

Zadania systemu operacyjnego

- Definicja interfejsu użytkownika.
- Udostępnianie systemu plików.
- Udostępnianie środowiska do wykonywania programów użytkownika:
  - mechanizm ładowania i uruchamiania programów,
  - mechanizmy synchronizacji i komunikacji procesów.
- Sterowanie urządzeniami wejścia-wyjścia.
- Obsługa podstawowej klasy błędów.

Definiowanie **interfejsu użytkownika** – system operacyjny dostarcza użytkownikom zbiór poleceń lub system okienkowy wraz z odpowiednim menu, który umożliwia interakcję z systemem komputerowym.

Udostępnianie systemu plików – system operacyjny organizuje i ułatwia dostęp do informacji np. w postaci hierarchicznego systemu plików.

Udostępnianie środowisko do wykonywania programów: system operacyjny dostarcza struktur danych do organizacji wykonywania programu oraz zachowywania i odtwarzania stanu przetwarzania (procesy i przełączanie kontekstu). System operacyjny udostępnia też programistom mechanizmy komunikacji pomiędzy procesami (kolejki komunikatów, strumienie, pamięć współdzielona) oraz mechanizmy synchronizacji procesów (semafory, muteksy).

Sterowanie urządzeniami wejścia-wyjścia – odpowiednie moduły sterujące, integrowane z systemem operacyjnym, inicjalizują pracę urządzeń zewnętrznych oraz pośredniczą w efektywnym przekazywaniu danych pomiędzy jednostką centralną a tymi urządzeniami.

Obsługa podstawowej klasy błędów – system operacyjny reaguje na błędy użytkowników (niedostępność zasobów, brak prawa dostępu), programistów (błąd dzielenia przez 0, naruszenie ochrony pamięci, nieprawidłowy kod rozkazu) oraz systemu (błąd braku strony pamięci, błąd magistrali).

Efektywność zarządzania zasobami oraz wygodny interfejs dla użytkownika są dwoma ogólnymi, niezależnymi celami projektowymi systemów operacyjnych. Pierwszy z tych celów był kluczowy w rozwoju rodziny systemów uniksowych. Dopiero w późniejszych etapach ich rozwoju pojawił się intuicyjny okienkowy interfejs użytkownika.

Systemy rodziny MS Windows zorientowane były natomiast przede wszystkim na interfejs użytkownika, na bazie którego w późniejszych etapach rozwoju powstawał *pełnowartościowy* system operacyjny, uwzględniający szerzej rozumiane zarządzanie zasobami.

Zarządzanie zasobami systemu operacyjnego

- Przydział zasobów.
- Planowanie dostępu do zasobów.
- Ochrona i autoryzacja dostępu do zasobów.
- Odzyskiwanie zasobów.
- Rozliczanie – gromadzenie danych o wykorzystaniu zasobów.

Na nasze potrzeby, **zasób** będzie rozumiany intuicyjnie jako **element systemu komputerowego** istotny, czy wręcz kluczowy dla realizacji przetwarzania. Funkcja zarządzania zasobami nie jest bezpośrednio wykorzystywana przez użytkownika (czasami nie jest przez niego w ogóle dostrzegana). Jej celem jest optymalizacja wykorzystania zasobów przez użytkowników.

Przydział zasobów – realizacja żądań dostępu do zasobów w taki sposób, że zasoby używane są zgodnie z intencją użytkowników (np. zagwarantowanie wyłącznego dostępu drukarki).

Planowanie dostępu do zasobów – strategia przydziału zasobów gwarantująca bezpieczeństwo, żywotność, brak zakleszczenia, sprawiedliwość oraz optymalność ich wykorzystania. Warto zwrócić uwagę na odróżnienie planowania od samego przydziału – przydział oznacza powiązanie zasobu z realizowanym zadaniem, podczas gdy planowanie wiąże się z odejmowaniem decyzji odnośnie wyboru zdania, któremu zasób będzie przydzielony.

Ochrona i autoryzacja dostępu do zasobów – dopuszczanie możliwości użytkowania zasobu tylko przez osoby uprawnione i w zakresie przydzielonych im uprawnień.

Odzyskiwanie zasobów – dołączanie zwolnionych zasobów do zbioru zasobów wolnych po zakończeniu ich użytkowania.

Rozliczanie – rejestrowanie i udostępnianie informacji o wykorzystaniu zasobów w celach kontrolnych i rozrachunkowych.

Zasoby zarządzane przez system operacyjny

Typowymi zasobami zarządzanymi przez system operacyjny są: procesor, pamięć, urządzenia wejścia-wyjścia (w tym system plików, stanowiący tzw. wirtualne wejście-wyjście). Zależnie od zasobu zarządzanie charakteryzuje się pewną specyfiką.

- Procesor – przydział czasu procesora.
- Pamięć:
  - alokacja przestrzeni adresowej dla procesów,
  - ochrona i transformacja adresów.
- Urządzenia wejścia-wyjścia:
  - udostępnianie i sterowanie urządzeniami pamięci masowej,
  - alokacja przestrzeni dyskowej,
  - udostępnianie i sterownie drukarkami, skanerami itp.
- Informacja (system plików):
  - organizacja i udostępnianie informacji,
  - ochrona i autoryzacja dostępu do informacji.

Procesor jest zasobem współdzielonym przez wiele procesów, w związku z czym zadaniem systemu operacyjnego jest przydział kwantu czasu procesora i wywłaszczanie zadania, które:

- wykorzystało już swój czas,
- nie może kontynuować przetwarzania ze względu na brak innych zasobów (brak gotowości urządzeń wejścia-wyjścia),
- ma zbyt niski priorytet.

Pamięć jest zasobem, który przydzielany jest na wyłączność danego przetwarzania. Zadaniem systemu jest zatem utrzymywanie informacji o zajętości przestrzeni adresowej, znajdowanie i przydzielanie odpowiednich fragmentów wolnej pamięci na żądanie aplikacji użytkownika lub innych modułów systemu operacyjnego oraz reagowanie na naruszenie ochrony pamięci.

System operacyjny pośredniczy również w transformacji adresów wirtualnych na fizyczne w systemach z segmentacją lub stronicowaniem przez organizację tablicy segmentów lub stron oraz obsługę błędów strony.

Urządzenia zewnętrzne są stosunkowo wolne, w związku z czym, w celu poprawy efektywności, zarządzanie tymi urządzeniami wymaga odpowiedniej organizacji systemu przerwań oraz buforowania danych, ewentualnie spoolingu.

Osobnym zagadnieniem jest zarządzanie urządzeniami pamięci masowej, zwłaszcza odpowiednia organizacja przestrzeni dyskowej oraz optymalizacja ruchu głowic dyskowych.

Informacja jest z punktu widzenia użytkownika najważniejszym zasobem, gdyż jej przetwarzanie jest celem systemu komputerowego. System operacyjny odpowiada za organizację gromadzenia i udostępniania informacji, jej ochronę przed nieuprawnioną ingerencją, spójność w przypadku awarii itp.

Klasyfikacja systemów operacyjnych (sposób przetwarzania)

- Systemy przetwarzania bezpośredniego (*on-line processing systems*) – systemy interakcyjne:
  - występuje bezpośrednia interakcja pomiędzy użytkownikiem a systemem,
  - wykonywanie zadania użytkownika rozpoczyna się zaraz po przedłożeniu.
- Systemy przetwarzania pośredniego (*off-line processing systems*) – systemy wsadowe:
  - występuje znacząca zwłoka czasowa między przedłożeniem a rozpoczęciem wykonywania zadania,
  - niemożliwa jest ingerencja użytkownika w wykonywanie zadania.

W przypadku **systemu przetwarzania bezpośredniego** użytkownik wprowadza zadanie do systemu i oczekuje na wyniki. W trakcie przetwarzania jest zatem możliwa interakcja pomiędzy użytkownikiem a systemem (aplikacją). Użytkownik może być na przykład poproszony o wprowadzenie jakiś danych na terminalu, wybranie czegoś z menu itp.

W przypadku **przetwarzania pośredniego** zadanie jest realizowane w czasie wybranym przez system. Po przedłożeniu zadania ingerencja użytkownika jest niemożliwa. Wszystkie dane muszą być zatem dostępne w momencie przedkładania zadania, a jakikolwiek błąd programowy lub niekompletność danych oznacza konieczność przedłożenia i wykonania zadania ponownie.

Klasyfikacja systemów operacyjnych (liczba wykonywanych programów)

- Systemy jednozadaniowe – niedopuszczalne jest rozpoczęcie wykonywania następnego zadania użytkownika przed zakończeniem poprzedniego.
- Systemy wielozadaniowe – dopuszczalne jest istnienie jednocześnie wielu zadań (procesów), którym zgodnie z pewną strategią przydzielany jest procesor.

Systemy jednoprogramowe, zwane też **jednozadaniowymi**, umożliwiają uruchomienie jednego zadania użytkownika, które ewentualnie może być wykonywane współbieżnie z pewnymi procedurami systemu operacyjnego.

Systemy wieloprogramowe (**wielozadaniowe**) dostarczają mechanizm przełączania kontekstu, umożliwiając w ten sposób zachowanie stanu wykonywania określonego programu (stanu procesu), a następnie odtworzenie stanu wykonywania innego programu (w szczególności innego wykonywania tego samego programu). Przełączenie kontekstu jest skutkiem zwolnienia procesora, które z kolei następuje w wyniku:

- żądania przydziału dodatkowego zasobu,
- zainicjowania operacji wejścia-wyjścia,
- przekroczenia ustalonego limitu czasu (kwantu czasu),
- uzyskania gotowości przez inne zadanie (proces) o wyższym priorytecie.

Klasyfikacja systemów operacyjnych (liczba użytkowników)

- Systemy dla jednego użytkownika – zasoby przeznaczone są dla jednego użytkownika, nie ma mechanizmów autoryzacji, a mechanizmy ochrony informacji są ograniczone.
- Systemy wielodostępne – wielu użytkowników może korzystać z zasobów systemu komputerowego, a system operacyjny gwarantuje ich ochronę przed nieupoważnioną ingerencją.

W systemach dla **jednego użytkownika** nie ma problemu autoryzacji, czyli konieczności identyfikowania zleceniodawcy poszczególnych zadań. Mechanizmy ochrony są ograniczone w tym sensie, że nie ma potrzeby ochrony zasobów jednego użytkownika przed drugim użytkownikiem tego samego systemu operacyjnego, ale w czasie powszechności sieci rozległych istnieje jednak problem ochrony zasobów przed ingerencją z zewnątrz.

System operacyjny w przypadku **wielodostępu** musi zagwarantować, że jeden użytkownik nie jest w stanie zakłócić pracy innych użytkowników. Jest to problem właściwego udostępniania zasobów oraz dostępności mechanizmów ochrony prywatnych zasobów jednego użytkownika przed ingerencją innego.

Inne rodzaje systemów operacyjnych

- Systemy czasu rzeczywistego (*real-time systems*) – zorientowane na przetwarzanie z uwzględnieniem czasu zakończenie zadania, tzw. linii krytycznej (*deadline*).
- Systemy sieciowe i rozproszone (*network and distributed systems*) – umożliwiają zarządzanie zbiorem rozproszonych jednostek przetwarzających, czyli zbiorem jednostek (komputerów), które są zintegrowane siecią komputerową i nie współdzielą fizycznie zasobów.
- Systemy operacyjne komputerów naręcznych – tworzone dla rozwiązań typu PDA, czy telefonów komórkowych, podlegają istotnym ograniczeniom zasobowym.

W **systemach czasu rzeczywistego** priorytetem jest minimalizacja czasu odpowiedzi (reakcji) lub czasu realizacji zadania, gdyż po przekroczeniu pewnego czasu wartość wyników albo jest znacznie mniejsza (przewidywanie kursów akcji na giełdzie) albo są one całkowicie bezużyteczne (prognozowanie pogody).

Szczególnym przypadkiem, gdzie czas jest krytyczny, są wszelkiego rodzaju systemy sterowania w czasie rzeczywistym (w komputerach pokładowych samochodów lub samolotów, urządzeniach medycznych). Systemy operacyjne czasu rzeczywistego są więc budowane pod kątem szybkości reakcji na zdarzenie zewnętrzne. Ich zadaniem jest minimalizować czas oczekiwania na zasoby dla czasowo krytycznych zadań, dlatego unika się w ich przypadku rozwiązań, które zmniejszają przewidywalność tego czasu (np. pamięci wirtualnej).

Rozproszone systemy operacyjne zapewniają, że system komputerowy, złożony z autonomicznych jednostek przetwarzających połączonych siecią komputerową, postrzegany jest jako całość. Zasoby tego systemu udostępniane są w jednolity sposób niezależnie od ich fizycznej lokalizacji – niezależnie od tego, czy są to zasoby lokalne danej jednostki, czy zasoby związane integralnie z jednostką zdalną.

Cecha ta odróżnia systemy rozproszone od systemów sieciowych, które również umożliwiają dostęp do zdalnych zasobów, ale nie ukrywają faktu fizycznego rozproszenia tych zasobów. Inaczej mówiąc, w systemie sieciowym odróżnia się dostęp lokalny i dostęp zdalny do zasobów.

Rozwiązania dla **systemów naręcznych** nie muszą tworzyć środowiska dla zaawansowanego przetwarzania wielozadaniowego, ale ze względu na niewielkie rozmiary urządzeń podlegają dość rygorystycznym ograniczeniom zasobowym.

W przypadku tego typu rozwiązań, jak również rozwiązań dla innych urządzeń mobilnych, dość istotnym zasobem, którym należy odpowiednio zarządzać jest energia.

Budowa i zasada działania systemu operacyjnego

System operacyjny jest **programem**, jednak jego działanie jest dość specyficzne, gdyż musi on **nadzorować** (monitorować) **pracę komputera** nawet wówczas, gdy wykonywany jest jakiś program aplikacyjny.

System operacyjny musi **reagować** na **błędy** w programach aplikacyjnych, **porządkować** system komputerowy po awariach, z kolei błędy w kodzie jądra systemu operacyjnego mogą zdestabilizować funkcjonowanie całego systemu komputerowego.

Działanie współczesnych systemów operacyjnych jest rezultatem ewolucji w architekturze sprzętowo-programowej, w której potrzeby w zakresie implementacji pewnych mechanizmów systemu operacyjnego wymuszały wprowadzanie stosownych rozwiązań na poziomie architektury komputera (procesora, jednostki zarządzania pamięcią, układu bezpośredniego dostępu do pamięci, układów wejścia-wyjścia).

Działania procesora, w zasadzie ogranicza się do **wykonywania rozkazów** programów (systemu operacyjnego też), które powtarzają się cyklicznie. W związku z czym określa się je jako cykle rozkazowe. Realizacja cyklu rozkazowego wymaga na ogół kilku interakcji procesora z pamięcią. Każdą taką interakcję określa się mianem cyklu maszynowego.

W każdym **cyklu rozkazowym** występuje cykl maszynowy pobrania kodu rozkazu. W zależności od trybu dostępności operandów mogą też wystąpić cykle pobrania operandu z pamięci (albo rejestrów wejścia-wyjścia) lub składowania operandu w pamięci (albo rejestrach wejścia-wyjścia). Każdy cykl maszynowy oznacza zatem zapis lub odczyt pamięci, przy czym cykl pobrania kodu rozkazu oznacza zawsze odczyt.

W czasie wykonywania rozkazu mogły nastąpić pewne **zdarzenia zewnętrzne** w stosunku do procesora, nie związane z bieżącym cyklem rozkazowym, ale wymagające od procesora jakiejś reakcji. Konieczność reakcji zgłaszana jest poprzez sygnał na odpowiedniej linii wejściowej procesora. Ostatnia faza cyklu rozkazowego polega zatem na sprawdzeniu, czy wystąpiło takie zgłoszenie.

Jeśli nie było zgłoszenia, rozpoczyna się następny cykl maszynowy. Jeśli jednak było zgłoszenie – nazywane **przerwaniem**, następuje ciąg działań, zmierzających do zidentyfikowania źródła przerwania, a następnie przekazania sterowania do stosownej **procedury obsługi**.

Procedury obsługi przerwań są częścią programu jądra systemu operacyjnego.

Przerwania można podzielić na:

- **zewnętrzne** – pochodzące z układów na zewnątrz procesora, czyli od urządzeń wejścia-wyjścia, czasomierzy, układu bezpośredniego dostępu do pamięci itp.,
- **wewnętrzne** (diagnostyczne) – zgłaszane przez procesor, będące następstwem wykrycia jakiegoś stanu wyjątkowego,
- **programowe** – wynikające z wykonania specjalnej instrukcji procesora, umożliwiające programom użytkownika dostęp do wybranych funkcji jądra systemu operacyjnego.

Przerwania od urządzeń zewnętrznych zgłaszane są po zakończeniu operacji wejścia-wyjścia i przekazywane na specjalne wejście procesora najczęściej przez sterownik przerwań. Tą samą ścieżką zgłaszane są również przerwania od układów ściśle współpracujących z procesorem – czasomierzy, układów bezpośredniego dostępu do pamięci itp.

Są to typowe przerwania, gdyż ich źródło jest poza procesorem i jest od niego niezależne.

W przeciwieństwie do przerwań zewnętrznych, przerwania programowe są wynikiem wykonania specjalnej instrukcji procesora.

Przerwania diagnostyczne są z kolei generowane wewnętrznie przez procesor w sytuacji zajścia określonego stanu. Są zatem pośrednim skutkiem wykonania określonego ciągu rozkazów prowadzących do osiągnięcia tego stanu. Tego typu przerwania można traktować jak pułapki lub wyjątki.

System przerwań umożliwia **niesekwencyjne** (współbieżne) wykonywanie programów. Zmiana sekwencji wykonywania instrukcji polega na tym, że w reakcji na przerwanie następuje **zapamiętanie bieżącego stanu** przetwarzania (najważniejszych rejestrów procesora), **przekazanie sterowania** do ustalonej procedury obsługi i rozpoczęcie wykonywania instrukcji tej procedury.

W szczególności może to prowadzić do **przełączenia kontekstu**, czyli przekazania sterowania po zakończeniu procedury obsługi przerwania do innego przetwarzania, niż to które zostało przerwane.

Sposób **przechowywania informacji** jest specyficzny dla każdego rodzaju pamięci (dysk CD, dysk twardy), dlatego też systemy operacyjne oferują **specjalne struktury** danych nazywane **plikami**.

Pliki pozwalają uzyskać **dostęp** do **informacji** w ten sam sposób, niezależnie do tego gdzie ta informacja jest zapisana. Aby ułatwić zarządzanie plikami systemy operacyjne utrzymują informacje o nich w katalogach, które z kolei są organizowane w struktury katalogów.

Struktury katalogów wraz z plikami tworzą **system plików**. System plików jest najbardziej dostrzegalna dla użytkownika częścią systemu operacyjnego. Często systemy operacyjne obsługują więcej niż jeden system plików.

Definicja pliku, typy plików

Plik jest abstrakcyjnym typem danych dostarczanym przez system operacyjny w celu umożliwienia spójnej metody dostępu do informacji umiejscowionych w pamięciach różnego typu. Pliki posiadają szereg cech, z których najważniejsza, dla użytkownika jest jednoznacznie je identyfikująca **nazwa**.

Format pliku może być **swobodny** lub **ściśle określony**. Do plików o swobodnym formacie należą pliki **tekstowe**, w których dane są przechowywane w postaci ciągów znaków zakończonych znakiem (-kami) końca wiersza. Pliki o określonym formacie zawierają najczęściej **dane numeryczne binarne** lub **rekordy danych**.

Ogólnie, pliki mogą przechowywać programy, zarówno w postaci źródłowej, jak i wynikowej oraz dane. Informacje w plikach umieszczane są **porcjami**, które nazywamy rekordami **logicznymi**. Rozmiar takiego rekordu jest zmienny, w szczególności może on wynosić jeden bajt. Ponieważ rekordy logiczne są rozmieszczane w **blokach alokacji**, a rzadko się zdarza aby rozmiar rekordu logicznego był wielokrotnością rozmiaru bloku alokacji, jest to przyczyna powstawania fragmentacji.

Typ pliku jest cechą pliku, która określa jego **strukturę**. Systemy operacyjne mogą, ale nie muszą interpretować typy plików.

Przykładem systemów dysponujących taką cechą są systemy rodziny Windows, w których użytkownik może skojarzyć określoną akcję z konkretnym typem pliku.

Inne podejście może zakładać, że system operacyjny w ogóle nie będzie interpretował typów plików. Takie podejście zastosowano w systemie UNIX. Zawartość plików w tym systemie jest traktowana jako ciąg bajtów, a ich interpretacja jest pozostawiana aplikacjom użytkownika.

Operacje na plikach

Ponieważ plik jest abstrakcyjną strukturą danych, to system operacyjny musi dostarczyć definicji **operacji**, które mogą być na niej wykonywane. Istnieje pięć takich operacji, które są uznawane za podstawowe: **tworzenie pliku**, **zapis pliku**, **odczyt pliku**, **ustawienie wskaźnika pliku**, **usuniecie pliku**. Wszystkie te operacje wiążą się z przeszukaniem katalogu i znalezieniem pozycji odpowiadającej plikowi, na którym operacja ma zostać przeprowadzona (w przypadku tworzenia pliku musi to być pozycja pusta).

Aby wyeliminować przeszukiwanie katalogu z tych operacji wprowadza się dodatkową operacje nazywaną **otwieraniem pliku**. Jeśli plik jest otwierany, to system operacyjny tworzy w pamięci operacyjnej kopie jego wpisu w katalogu. Operacją uzupełniającą do otwierania jest **zamykanie pliku**, które powoduje zapisanie wpisu z pamięci operacyjnej na dysk, jeśli te wersje się różnią i usunięcie wpisu z pamięci. Systemy wielozadaniowe dostarczają także operacji **blokowania** (*lock*) części lub całości plików, celem ochrony ich przed współbieżną **modyfikacją**.

Semantyka spójności

Semantyka spójności nazywana również semantyką spoistości lub semantyką integralności określa **reguły dostępu** do plików **współdzielonych** w systemach wielozadaniowych, dzięki którym zmiany w takich plikach mogą być dokonywane w sposób bezpieczny.

Semantyka ta określa również zakres widoczności zmian dokonywanych w pliku przez jego pozostałych użytkowników.

W semantyce spójności systemów UNIX/Linux, zasób jakim jest plik ma tylko jeden obraz, widoczny dla każdego z użytkowników. Może to powodować opóźnienia w realizacji operacji wykonywanych na pliku, ze względu na konieczność modyfikacji na zasadach wyłączności.

Semantyka systemu UNIX/Linux definiuje reguły:

- jeśli kilku użytkowników ma otwarty plik współdzielony i jeden z nich dokona modyfikacji jego zawartości, to jej skutki nie są widoczne natychmiast dla pozostałych,
- zmiany w pliku są widoczne tylko dla tych użytkowników, którzy zamknęli plik i ponownie go otworzyli.

Te reguły wymagają, aby każdy z użytkowników pliku otrzymywał jego kopie, unikalny obraz tego pliku dostępny dla każdego z nich osobno.

Katalogi logiczne

W systemach plików są definiowane **katalogi logiczne**. Katalog logiczny jest plikiem, którego zawartość stanowią wykazy informacji o innych plikach, czyli tak zwane metadane plików. Do tych metadanych mogą zaliczać się:

- unikatowa nazwa pliku,
- informacja o typie pliku (np. trzyliterowe rozszerzenie nazwy),
- wskaźnik pliku,
- rozmiar pliku (w bajtach lub jednostkach alokacji),
- lokalizacja pliku, czyli np. numer pierwszego bloku alokacji przedzielonego plikowi,
- informacje o ochronie, czyli o trybie dostępu,
- licznik użycia, określający ilu użytkowników korzysta w danej chwili z pliku,
- identyfikator właściciela lub twórcy pliku,
- czasy utworzenia, ostatniej modyfikacji lub ostatniego dostępu do pliku.

Katalogi tworzą strukturę katalogów. Taka struktura może obejmować nie tylko jedno urządzenie fizyczne, ale również kilka takich urządzeń (np. struktura katalogów systemu UNIX/Linux). Dzięki temu system operacyjny ukrywa przed użytkownikiem faktyczną lokalizację plików.

Kontrola dostępu

Z każdym plikiem lub katalogiem można związać **wykaz dostępów** (*access control list*), czyli opis praw jakie przysługują poszczególnym użytkownikom do tego pliku (czy użytkownik może dokonywać zapisu, odczytu itd) po ich identyfikacji przez system operacyjny.

Jeśli użytkownik próbuje wykonać jakąś operację, to system operacyjny odnajduje jego identyfikator w wykazie dostępów i sprawdza jakie prawa mu przysługują. Jeśli w wykazie znajduje się operacja, którą użytkownik chce wykonać, to jest ona wykonywana, w przeciwnym razie system odmawia dostępu użytkownikowi. Wadą tego rozwiązania może okazać się wielkość takiego wykazu.

W **dostępie grupowym** z każdym plikiem lub katalogiem związanych jest dziewięć bitów. Każda trójka bitów opisuje prawa dostępu do pliku pozwalające na **odczyt**, **zapis** i **wykonanie** (rwx w notacji uniksowej). Również znaczenie każdej z tych trójek jest inne.

Pierwsza opisuje prawa **właściciela pliku** (*user*), druga prawa **grupy** użytkowników do której należy właściciel pliku (*group*), a trzecia określa prawa dostępu dla **pozostałych** użytkowników w systemie (*others*).

Grupy użytkowników są zdefiniowane w specjalnie chronionym pliku systemowym. Dzięki dostępowi grupowemu możliwe jest współdzielenie plików. Również ilość informacji związanych z ochroną plików, które trzeba zapamiętać jest mała w stosunku do wykazu dostępów.

Emulacja i wirtualizacja

Emulacja (pojecie węższe niż symulacja – czyli imitowanie przez program komputerowy zachowania określonego modelu abstrakcyjnego) polega na symulowaniu przez program komputerowy uruchomiony w systemie komputerowym o danej strukturze pełnego zachowania innego systemu komputerowego, najczęściej o odmiennej konstrukcji. Program symulujący określany jest mianem emulatora.

Istnieją dwa rodzaje emulatorów:

- emulatory pełne – działają zgodnie ze specyfikacja określonego fizycznego komputera (qemu, Bochs, uae, Frodo),
- emulatory `API` – emulują jedynie działanie interfejsu użytkownika określonego systemu operacyjnego i nazywane są często warstwami kompatybilności (wine, DosBox, dosemu).

Warto zauważyć, że emulatory API nie są całkowicie zgodne z podaną wyżej definicją emulatora – symulują działanie tylko jednej warstw systemu operacyjnego. W związku z tym niektórzy są zdania, że nie należny nazywać ich emulatorami. Szczególnie podkreślają to twórcy **wine**, którego nazwa jest rozwijana rekurencyjnie do *Wine Is Not an Emulator*.

Emulatory pełne symulują działanie platformy sprzętowej do poziomu kodu maszynowego. Istnieją dwie możliwości realizacji tak dokładnego odwzorowania. Pierwsza polega na interpretacji programu przygotowanego dla danej platformy przez emulator. Druga oparta jest na dynamicznej rekompilacji, tzn. tłumaczeniu określonego bloku kodu zamiast pojedynczej instrukcji.

Emulatory tworzone są z trzech powodów:

- potrzeba użycia określonej platformy sprzętowej, która nie jest dostępna,
- potrzeba uruchomienia oprogramowania przeznaczonego na starsze platformy sprzętowe,
- stworzenie platformy testowej dla oprogramowania.

Podczas gdy emulacja oznacza głównie symulacje określonej platformy sprzętowej na innej platformie, to **wirtualizacja** najczęściej oznacza w tym kontekście **stworzenie kopii** takiej samej platformy na jakiej przeprowadzana jest wirtualizacja.

Kopia ta, nazywana **maszyną wirtualną**, nie jest dokładna z racji tego, ze nie może ona używać w całości zasobów należących do maszyny fizycznej. Niemniej jednak jest ona na tyle wierna, że pozwala uruchomić system operacyjny i oprogramowanie użytkowe.

Wirtualizacja jest techniką zarządzania zasobami, która umożliwia ich współdzielenie przez procesy lub nawet systemy komputerowe za pomocą:

- podziału,
- agregacji.

Kryteria jakie powinna spełniać poprawnie działająca maszyna wirtualna oraz architektura komputera, aby możliwe było dla niej opracowanie maszyny wirtualnej są następujące.

- **Odpowiedniość** – program działający na maszynie wirtualnej musi zachowywać się dokładnie tak samo jakby był wykonywany na maszynie fizycznej.
- **Kontrola zasobów** – maszyna wirtualna musi w pełni kontrolować wszystkie zasoby podlegające wirtualizacji.
- **Wydajność** – większość instrukcji musi być wykonywana na maszynie fizycznej, bez udziału maszyny wirtualnej.

W praktyce warunki te nie muszą (i zazwyczaj nie są) dokładnie wypełnione. Powoduje to ograniczenia w działaniu maszyn wirtualnych oraz niższą wydajność.

Zastosowania wirtualizacji:

- zwiększenie niezawodności serwerów,
- prosta migracja maszyn wirtualnych,
- możliwość jednoczesnej pracy wielu wersji systemu operacyjnego,
- wirtualny hosting,
- testowanie systemów operacyjnych, aplikacji i oprogramowania sieciowego.

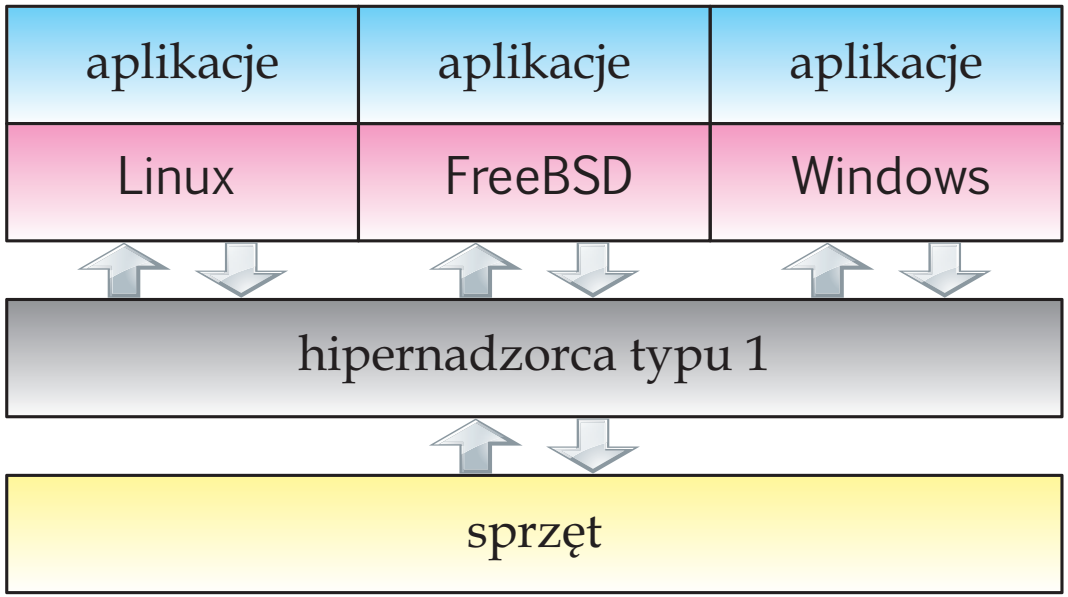
Hipernadzorcy i parawirtualizacja

Możemy wyróżnić trzy grupy rozkazów procesora: **rozkazy zwykłe**, które mogą być wykonywane we wszystkich trybach pracy procesora, **rozkazy uprzywilejowane**, których próba wykonania w trybie użytkownika powoduje wygenerowanie wyjątku oraz **rozkazy wrażliwe**, które powinny być wykonywane tylko w trybie jądra.

Realizacje wirtualizacji z użyciem **hipernadzorcy typu 1** jest możliwa jeśli zbiór rozkazów wrażliwych jest podzbiorem zbioru instrukcji uprzywilejowanych.
W przypadku architektur PC taką wirtualizacje umożliwiają procesory firmy AMD z technologią SVM (*Secure Virtual Machine*), przemianowaną później na **AMD-V** oraz firmy Intel z technologią **VT** (*Virtualization Technology*), które umożliwiają hipernadzorcy określenie, które rozkazy powinny generować wyjątki przy próbie wykonania ich w trybie użytkownika, a które nie.

Działanie wirtualizacji opartej o hipernadzorcę typu 1 można opisać następująco:

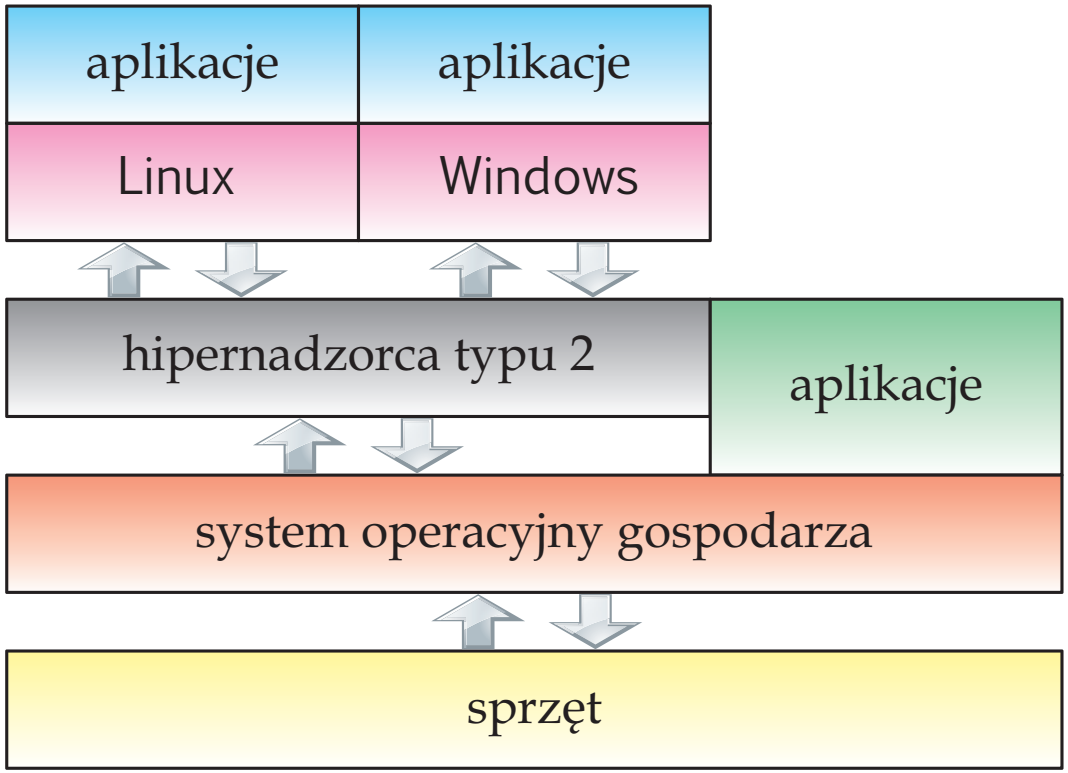
- w trybie jądra procesora działa tylko hipernadzorca nazywany również monitorem maszyn wirtualnych,
- hipernadzorca kontroluje pracę wszystkich pracujących maszyn wirtualnych,
- system operacyjny maszyny wirtualnej, nazywany systemem operacyjnym – gościem działa w trybie użytkownika maszyny rzeczywistej oraz w trybie jadra maszyny wirtualnej,
- procesy użytkownika działają w trybie użytkownika zarówno maszyny wirtualnej, jaki i rzeczywistej,
- rozkazy nieuprzywilejowane są wykonywane przez system operacyjny – gościa i procesy użytkownika bezpośrednio,
- jeśli pojawi się wyjątek spowodowany przez próbę wykonania rozkazu uprzywilejowanego, to hipernadzorca sprawdza, kto ją próbował wykonać; jeśli był to proces użytkownika, to uruchamiana jest obsługa wyjątku w systemie – gościu, jeśli był to system - gość, to hipernadzorca wykonuje ten rozkaz.


aplikacje
aplikacje
aplikacje
Linux
FreeBSD
Windows
hipernadzorca typu 1
sprzęt

Wirtualizacja z uzyciem **hipernadzorcy typu 2** nie wymaga wsparcia ze strony procesora w postaci takich technologi jak Intel VT lub AMD-V. Przykładami takich hipernadzorców jest **VirtualBox**.

Działanie wirtualizacji z użyciem hipernadzorcy typu 2 można opisać następująco:

- hipernadzorca jest oprogramowaniem działajacym w trybie użytkownika pod kontrolną systemu operacyjnego nazywanego systemem operacyjnym – gospodarzem,
- z punktu widzenia użytkownika hipernadzorca zachowuje się jak emulator maszyny rzeczywistej na której jest uruchomiony; pozwala zainstalować system operacyjny (nazywany systemem operacyjnym – gościem) i uruchamiać procesy użytkownika,
- problem rozkazów wrażliwych hipernadzorca rozwiązuje stosując translację binarną, która polega na znalezieniu bloków podstawowych, czyli ciągów rozkazów zakończonych dowolnym rozkazem zmieniającym przepływ sterowania i zastąpieniu wszystkich instrukcji wrażliwych znajdujących się w takich blokach wywołaniami procedur hipernadzorcy,
- w celu zwiększenia wydajności stosowana jest translacja z wyprzedzeniem oraz pamięci podręczne.



Wbrew intuicji wirtualizacja z użyciem hipernadzorcy typu 2 może być wydajniejsza niż wirtualizacja z uzyciem hipernadzorcy typu 1.

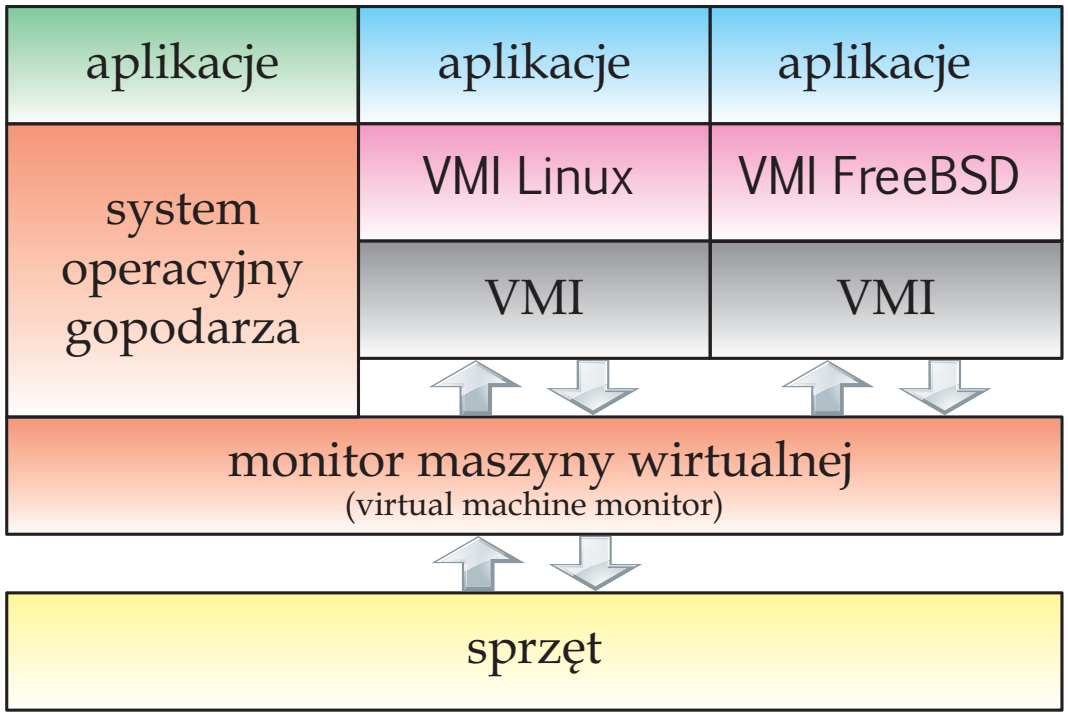
Parawirtualizacja jest wydajniejsza niż opisane wcześniej sposoby wirtualizacji, ale wymaga specjalnie przygotowanej wersji systemu operacyjnego – gościa. Przykładem hipernadzorcy umożliwiającego parawirtualizacje jest **Xen**.

Działanie parawirtualizacji można scharakteryzować następująco:

- hipernadzorca działa w trybie użytkownika maszyny rzeczywistej, tak jak hipernadzorca typu 2,
- hipernadzorca posiada własne `API`,
- kod źródłowy systemu operacyjnego – gościa posiada specjalnie spreparowany kod źródłowy, w którym każde odwołanie do instrukcji wrażliwej zastąpiono wywołaniem podprogramu wchodzącego w skład `API` hipernadzorcy,
- wykonywalna wersja systemu – gościa jest uruchamiana w trybie użytkownika maszyny rzeczywistej; nie wykonuje ona żadnych (lub prawie żadnych) rozkazów wrażliwych, jedynie korzysta z `API` hipernadzorcy.

Parawirtualizacja możne być stosowana na jednym komputerze, obok wyżej wymienionych sposobów wirtualizacji. Hipernadzorców, którzy umożliwiają parawirtualizacje możne być wielu i każdy z nich posiada swoje `API`.

Aby umożliwić systemowi – gościowi współpracę z nimi wszystkimi bez konieczności wykonywania dodatkowych modyfikacji wprowadzono warstwę pośrednia nazywaną biblioteką **VMI** (*Virtual Machine Interface*).


aplikacje
aplikacje
aplikacje
system operacyjny gopodarza
VMI Linux
VMI FreeBSD
VMI
VMI
monitor maszyny wirtualnej
(virtual machine monitor)
sprzęt

Wirtualizacja pamięci i urządzeń wejścia-wyjścia

Oprócz **procesora** również pozostałe zasoby maszyny rzeczywistej muszą podlegać wirtualizacji. Najważniejszym z nich jest **pamięć operacyjna**. We współczesnych systemach najczęściej stosowanym sposobem zarządzania RAM jest stronicowanie na zadanie, co utrudnia gospodarowanie pamięcią na rzecz maszyn wirtualnych.

Kłopotliwy scenariusz powstaje wtedy, gdy co najmniej dwie maszyny wirtualne próbują odwzorować swoje strony na te same ramki. Rozwiązaniem problemu jest utrzymywanie przez hipernadzorcę dla każdej maszyny wirtualnej specjalnej tablicy stron, która stanowi kolejną warstwę w procesie translacji numeru strony (zamienia adres ramki wirtualnej maszyny na adres ramki maszyny rzeczywistej).

Inny problem polega na tym, że system operacyjny – gość może zmienić zawartość tablicy bez używania rozkazów wrażliwych, stosując jedynie zwykły zapis do pamięci. Ta modyfikacja powinna być wykryta przez hipernadzorcę, żeby mógł uaktualnić tablice dla maszyny wirtualnej. W przypadku wirtualizacji hipernadzorca wykrywa, które strony zawierają tablice strony systemu – gościa i oznacza je jako tylko do odczytu. Każda próba modyfikacji zawartości tych stron generuje wyjątek, który obsługuje hipernadzorca.

W przypadku parawirtualizacji system – gość powiadamia hipernadzorce o zmianie zawartości swojej tablicy stron.

Hipernadzorca musi także przechwytywać polecenia **sterowników urządzeń** systemu operacyjnego – gościa. Konieczność ta wynika z potrzeby ochrony sprzętu oraz niemożliwości przydzielenia niektórych zasobów (np. dysku) w całości pojedynczej maszynie wirtualnej.

Zaletą takiego rozwiązania jest również udogodnienie polegające na tym, że oprogramowanie obsługujące sprzęt starego lub innego typu można wykonać z użyciem urządzeń nowego typu. Konwersją poleceń miedzy oboma typami urządzeń zajmie się ponownie hipernadzorca.

Problem wirtualizacji obsługi **wejścia-wyjścia** można również rozwiązać przeznaczając jedną z maszyn wirtualnych do tego zadania i przekierowując do niej zadania wykonania operacji pochodzące z innych maszyn. To rozwiązanie jest szczególnie wygodne w przypadku parawirtualizacji i stosuje je Xen. Maszyna wirtualna, która zajmuje się wykonywaniem operacji I/O nazywa się w tej platformie **domena 0**.

Takie podejście do realizacji operacji wejścia-wyjścia korzystne jest również w wirtualizacji z użyciem hipernadzorcy typu 1, bo nie musi on zawierać sterownika do danego urządzenia. Hipernadzorca typu 2 może polegać na tym, że sterownik określonego urządzenia posiada system operacyjny – gospodarz.

Cechy systemu

System operacyjny **GNU/Linux** (krótko **Linux**) jak wszystkie systemy operacyjne ma do spełnienia dwa podstawowe cele:

- zapewnienie wygodnej pracy użytkownikom,
- efektywne zarządzanie zasobami systemu komputerowego.

Osiągnięcie powyższych celów wiąże się z realizacją określonych zadań obejmujących:

- zarządzanie zadaniami,
- zarządzanie pamięcią operacyjną i pomocniczą,
- zarządzanie systemem wejścia-wyjścia,
- zarządzanie plikami,
- pracę sieciową,
- ochronę zasobów,
- komunikację z użytkownikami.

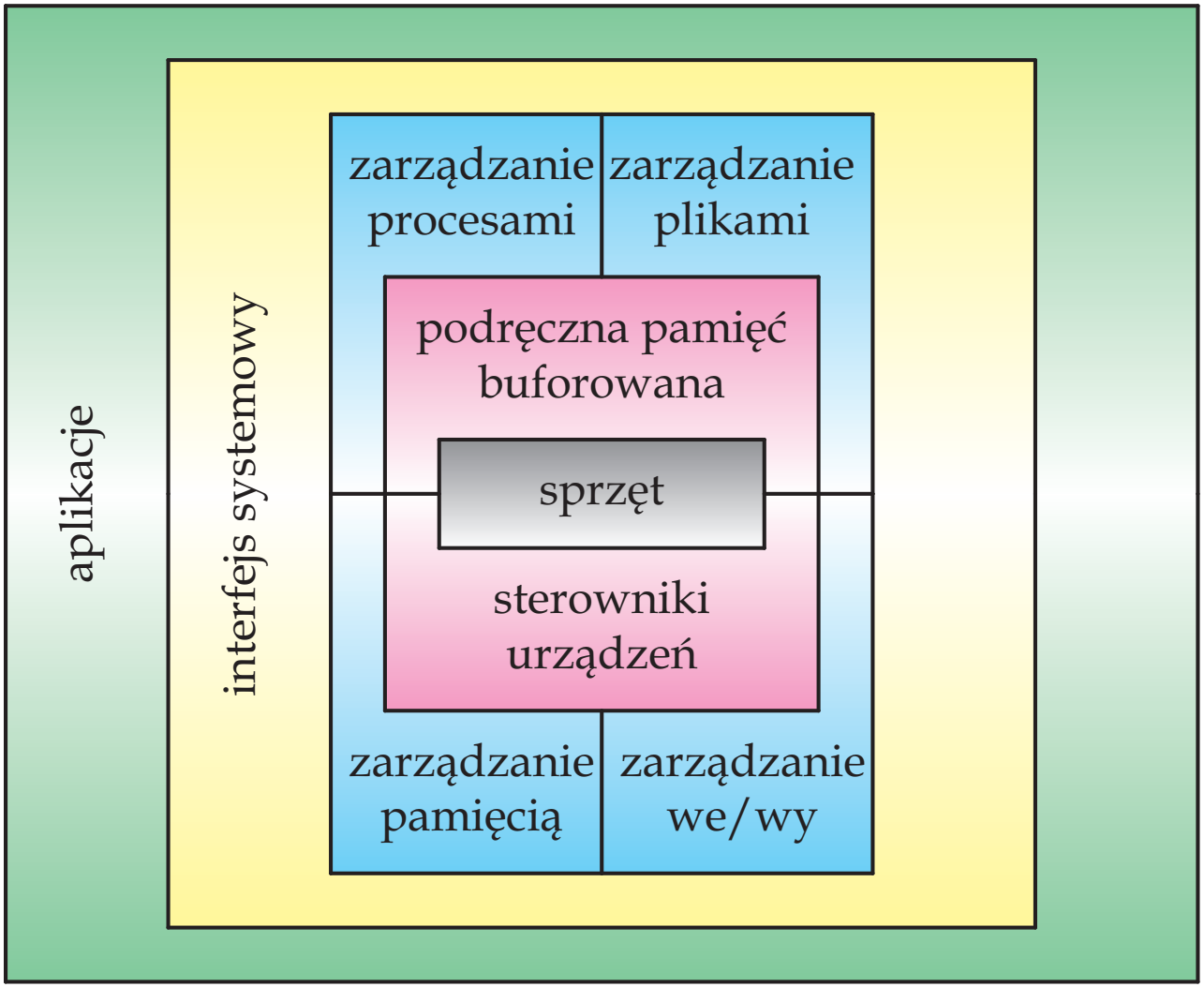
Krótka charakterystyka:

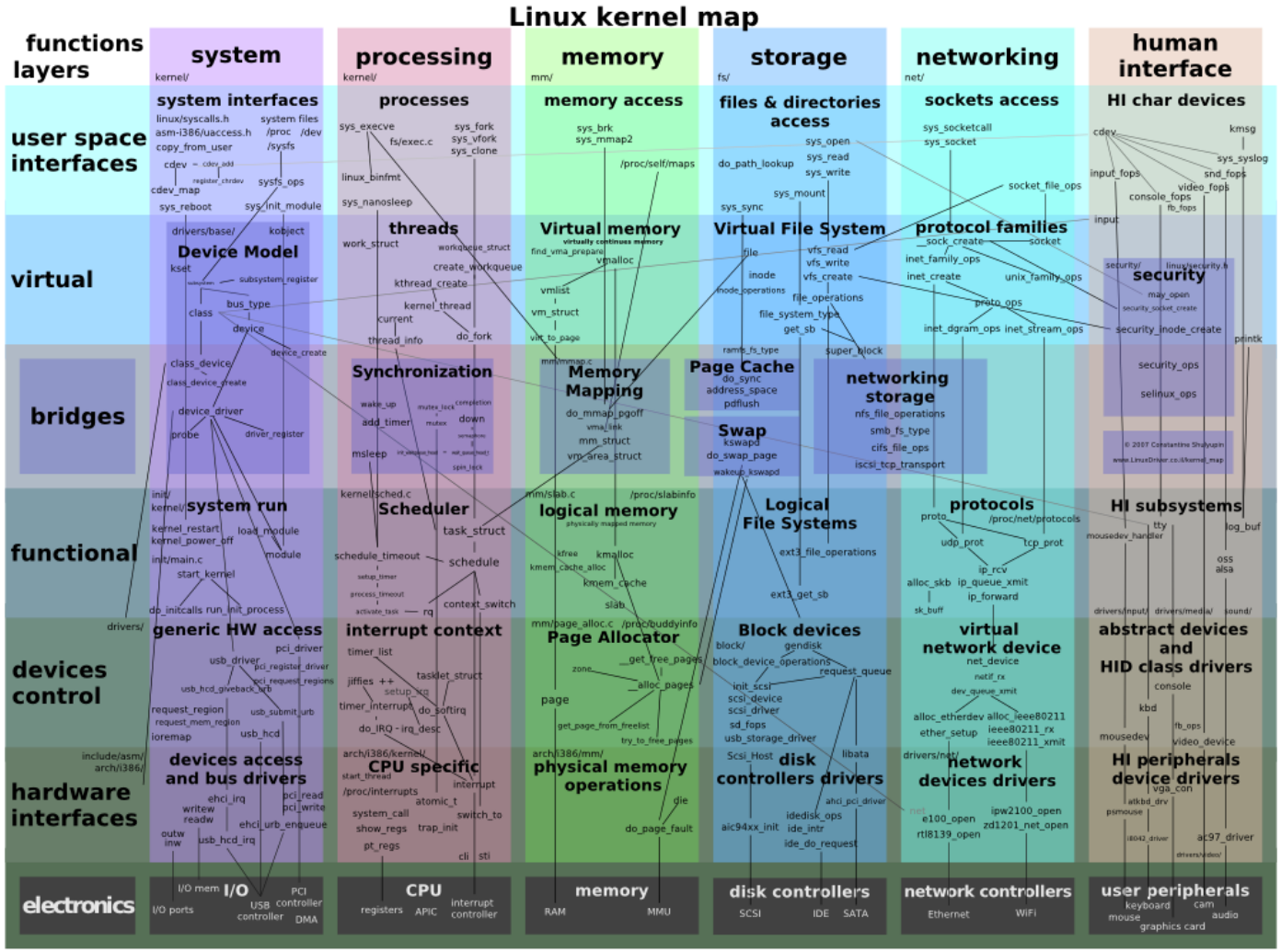
- efektywność i stabilność systemu,
- powszechna dostępność bez jakichkolwiek opłat licencyjnych,
- bogaty zestaw oprogramowania umożliwiający szeroki zakres zastosowań,
- możliwość pracy na wielu platformach sprzętowych przy stosunkowo niewielkich wymaganiach,
- możliwość łatwej współpracy z innymi popularnymi systemami operacyjnymi,
- bogata dokumentacja w wersji elektronicznej,
- dostępność kodu źródłowego.

Najważniejsze cechy systemu:

- wielodostęp,
- wielozadaniowość, czyli praca z podziałem czasu procesora pomiędzy wiele zadań,
- wieloprzetwarzanie, czyli praca wieloprocesorowa,
- możliwość uruchamiania zadań w łagodnym czasie rzeczywistym,
- obsługa różnych typów systemów plików,
- obsługa różnych protokołów sieciowych,
- obsługa różnych formatów plików wykonywalnych,
- wykorzystanie współdzielonych bibliotek.

Jądro Linux


aplikacje
interfejs systemowy
zarządzanie procesami
zarządzanie plikami
podręczna pamięć buforowana
sprzęt
sterowniki urządzeń
zarządzanie pamięcią
zarządzanie we/wy


Linux kernel map
functions
layers
system
kernel/
processing
kernel/
memory
mm/
storage
fs/
networking
net/
human interface
user space interfaces
system interfaces
processes
memory access
files & directories access
sockets access
HI char devices
virtual
Device Model
threads
Virtual memory
Virtual File System
protocol families
security
bridges
Synchronization
Memory Mapping
Page Cache
networking storage
Swap
functional
system run
Scheduler
logical memory
Logical File Systems
protocols
HI subsystems
devices control
generic HW access
interrupt context
Page Allocator
Block devices
virtual network device
abstract devices and HID class drivers
hardware interfaces
devices access and bus drivers
CPU specific
physical memory operations
disk controllers drivers
network devices drivers
HI peripherals device drivers
electronics
I/O
CPU
memory
disk controllers
network controllers
user peripherals

Jądro Linux jest w dużym stopniu zgodne ze standardami **POSIX** (*Portable Operating System Interface for UNIX* – standaryzacja różnych odmian systemu operacyjnego UNIX, obejmująca: interfejs programistyczny, interfejs użytkownika i właściwości powłoki systemowej), obsługuje wielozadaniowość, wielowątkowość, wielobieżność, pamięć wirtualną, biblioteki współdzielone, ładowanie na żądanie, współdzielony kod wykonywalny, dobre zarządzanie pamięcią i obsługę sieci TCP/IP.

Jest jądrem **monolitycznym** (makrokernel) z ładowalnymi **modułami**. Sterowniki urządzeń i rozszerzenia jądra zwykle pracują w trybie z pełnym dostępem do sprzętu (nieliczne działają w trybie użytkownika). W przeciwieństwie do typowych jąder monolitycznych, sterowniki urządzeń są zwykle kompilowane jako moduły, które można załadować i wyładować na działającym systemie. Podobnie, sterowniki mogą być wywłaszczone w określonych warunkach.

Jako **makrokernel** zajmuje się nie tylko komunikacją pomiędzy poszczególnymi procesami, ale także zarządzaniem pamięcią, obsługą urządzeń czy obsługą systemów plików.

Dla porównania, w architekturze mikrokernela, jądro zajmuje się tylko zapewnieniem komunikacji pomiędzy procesami, natomiast obsługą poszczególnych urządzeń, czy systemów plików, zajmują się osobne wykonywalne programy.

Modułowa budowa Linuksa oznacza, że części kodu mogą zostać od niego oddzielone i dzięki temu zyskuje największe zalety mikrokerneli, czyli:

- obsługę poszczególnych urządzeń czy protokołów można rozwijać niezależnie od samego jądra,
- podział prac nad jądrem na grupy zajmujące się poszczególnymi częściami znacznie ułatwia i przyspiesza rozwój,
- możliwość zmniejszenia wielkości jądra, poprzez wykasowanie z pamięci operacyjnej nieużywanych modułów,
- możliwość tworzenia zamkniętych, binarnych sterowników.

Ponadto struktura makrokernela pozwala na wbudowanie modułów w strukturę samego jądra, co przyspiesza działanie, ponieważ nie ma konieczności wielokrotnego powtarzania przełączania procesów, przy komunikacji pomiędzy poszczególnymi modułami.

Moduły, odpowiadają **sterownikom** w terminologii DOS/Windows. Mogą one obsługiwać nie tylko urządzenia, ale dodatkowo realizować:

- obsługę systemów plików,
- bezpośrednią komunikację z urządzeniami,
- obsługę protokołów (np. protokołów sieciowych),
- dodatkowe funkcjonalności rozszerzające możliwości jądra.

zarządzanie modułami	
`lsmod`	wyświetlanie załadowanych modułów
`modinfo`	wyświetlenie informacji o module
`insmod`	załadowanie modułu
`modprobe`	załadowanie modułu z zależnościami (preferowane)
`rmmod`	usunięcie modułu

Drzewo katalogów

W ystemie GNU/Linux podstawowa struktura katalogów jest dość ściśle określona według **FHS** (*Filesystem Hierarchy Standard*). W katalogu głównym oznaczanym przez / (*root*), jest tylko kilka katalogów i nie powinno umieszczać się w nim żadnych dodatkowych plików czy dodatkowych katalogów.

Nie spowoduje to nieprawidłowego działania systemu, ale jest to ogólnie przyjęty i dość restrykcyjnie przestrzegany standard, dzięki czemu system katalogów jest przejrzysty.

zawartość katalogów	
/bin	binarne (wykonywalne) pliki najbardziej podstawowych narzędzi systemowych (bash, cat, cp, dd, gzip, mount, rm, vi)
/boot	pliki niezbędne do uruchomienia systemu (kernel, initrd, pliki bootloadera) (backup_mbr, vmlinuz, initrd, menu.lst)
/dev	pliki urządzeń (nie są faktycznie plikami na dysku), za ich pośrednictwem system komunikuje się z urządzeniami (console, tty*, ttyS*, lp*, fd*, hd*, sd*, random, null)
/etc	pliki konfiguracyjne, ustawienia systemowe (inittab, xorg.conf, fstab, profile, passwd, shadow, group, hosts)
/home	katalogi domowe użytkowników (.profile, baskrc, .bash_profile)

zawartość katalogów	
/lib	systemowe biblioteki współdzielone
/media	katalog montowania nośników wymiennych (pendrive, CD/DVD-ROM)
/mnt	katalog montowania dysków (np. w Ubuntu, dyski są montowane w /media)
/proc	wirtualny katalog, zawierający dane o aktualnie uruchomionych procesach (1/*, cpuinfo, dma, interrupts, ioports, filesystems, modules, mounts, net/*, partitions, bus/pci/*, bus/usb/*, sys/*)
/root	katalog domowy użytkownika *root*
/sbin	pliki wykonywalne poleceń, które mogą być wykonywane tylko przez administratora (fdisk, fsck*, init, ip, mkfs*, modprobe, mount, rmmod, shutdown, umount)
/sys	wirtualne pliki systemu (sprzęt)
/tmp	pliki tymczasowe
/usr	dodatkowe programy, mogą być uruchamiane przez zwykłego użytkownika
/var	pliki systemowe, których zawartość często się zmienia (logi, poczta, cron, często pliki serwerów usług)

Partycje

W systemach operacyjnych GNU/Linux, należy przyporządkować konkretne partycje katalogom lub podkatalogom znajdującym się w strukturze katalogów FHS. System potrzebuje co najmniej jednej partycji, która wtedy będzie automatycznie montowana do katalogu głównego *root* /.

Zalecane jest, aby dodatkowo tworzyć partycję wymiany **swap**. Służy ona do tymczasowego przechowywania danych w sytuacji, gdy ich ilość przekracza zasoby wolnej pamięci RAM lub gdy z różnych powodów korzystniej jest przechowywać je (lub ich część) na dysku twardym (np. hibernacja laptopa).

System może pracować bez partycji wymiany.

W systemach produkcyjnych należy stosować więcej niż jedną partycję. Pozwala to zwiększyć bezpieczeństwo danych w razie poważnej awarii dysku czy systemu, chociaż to ostatnie jest bardzo mało prawdopodobne.

Można też katalogi domowe użytkowników i katalogi z oprogramowaniem (`/home`, `/opt`, `/usr`) współdzielić lub delegować na inny komputer lub macierz dyskową.

Nie można delegować katalogów potrzebnych do działania systemu, muszą się znaleźć na lokalnych dyskach: `/bin`, `/boot`, `/dev`, `/etc`.

Katalogi a partycje – przykład

```
/ (root)
|-/bin
|-/boot
|-/etc
|-/home
|-/lib
|-/usr
|   |-/bin
|   |-/lib
|   |-/share
|   +-/X11R6
|-/tmp
+-/var
```

Katalogi /bin, /dev, /etc, /lib i /sbin muszą być na głównej partycji montowanej do /.

Montowanie i odmontowanie

Partycje i systemy plików montowane automatycznie podczas startu systemu i montowane przez zwykłych użytkowników muszą być skonfigurowane w pliku `/etc/fstab`:

```
/dev/disk/by-id/ata-...-part1 swap   swap  defaults        0 0
/dev/disk/by-id/ata-...-part2 /      ext3  acl,user_xattr  1 1
proc        /proc             proc       defaults        0 0
sysfs       /sys              sysfs      noauto          0 0
debugfs     /sys/kernel/debug debugfs    noauto          0 0
usbfs       /proc/bus/usb     usbfs      noauto          0 0
devpts      /dev/pts          devpts     mode=0620,gid=5 0 0
/dev/sda7   /export/data1     vfat       defaults        1 2
/dev/sda8   /export/data2     ext2       defaults        1 2
/dev/sda9   /export/data3     reiserfs   defaults        1 2
```

Zwykle jako punkt montowania dysków twardych używa się podkatalogów w katalogu `/mnt`, a nośników wymiennych: podkatalogów w katalogu `/media`.

Przykłady polecenia mount:

```
$ mount -t vfat /dev/hdb1 /mnt/old_c
$ mount -t iso9660 -o loop /dev/cdrom /mnt/cdrom
$ mount /dev/fd0 /mnt/floppy
```

Przykłady polecenia umount:

```
$ umount /mnt/old_c
$ umount /mnt/cdrom
$ umount /mnt/floppy
```

Najpopularniejszymi systemami plików używanymi w Linuksie są: **Ext2**, **Ext3**, **Ext4**, **ReiserFS**, **XFS**. Każdy system plików ma swoje mocniejsze i słabsze strony, przy czym, to nie są kolosalne różnice.

ReiserFS oraz XFS mają nieco większą wydajność ale małą odporność na zaniki napięcia, dlatego w systemach bez zasilacza awaryjnego, potrafiącego poprawnie zamknąć system w razie zaniku napięcia, lepiej stosować Ext4 lub Ext3.

Na partycjach szyfrowanych należy używać systemów bez księgowania, np. Ext2.

Organizacja danych, systemu plików Ext

Dyski twarde i dyskietki posiadają blokową strukturę danych, dane są w nich przechowywane postaci bloków, a blok musi być w całości zapisywany i odczytywany. W przypadku dysków wielkość pojedynczego bloku danych jest wielokrotnością rozmiaru pojedynczego sektora (najczęściej 512 bajtów).

Bloki tworzą strukturę liniową:

Systemy operacyjne łączą bloki w **grupy** (klastry). Klaster składa się z kilku bloków. Dyskowy system plików część pamięci wykorzystuje na tzw. sektor rozruchowy (*boot sector*), który nie stanowi zasadniczej części systemu plików i jest używany do rozruchu systemu.

Systemy plików przechowuje listy wszystkich klastrów i informacje do których plików one należą w *tablicach alokacji*. Przykład alokacji w systemie plików Ext2.

Grupa fizyczna bloków zawiera:

- redundantną kopię bloku identyfikacyjnego (superblok),
- bloki z deskryptorami wszystkich grup logicznych systemu plików,
- blok z mapą zajętości bloków danych,
- blok z mapą zajętości tablicy i-węzłów,
- bloki z tablicą i-węzłów,
- bloki danych – określonej grupy logicznej.

Przydzielając miejsce dla pliku, system Ext wybiera najpierw grupę bloków. System usiłuje przydzielić blokom danych tę grupę bloków, do której przydzielono i-węzeł pliku.

Do przydzielania i-węzłów plików nie będących katalogami system wybiera tę grupę bloków, w której mieści się macierzysty katalog pliku. Pliki katalogowe nie są przechowywane razem, lecz są rozproszone w dostępnych grupach bloków.

Te zasady przyjęto po to, aby powiązane ze sobą informacje trzymać w tej samej grupie bloków, a także by rozłożyć obciążenie dysku między jego grupy bloków w celu zmniejszenia fragmentacji obszarów dyskowych.

Podczas inicjalizowania systemu, do pamięci wczytywane są bloki z deskryptorami grup logicznych z pierwszej grupy fizycznych bloków dyskowych, gdy nie występują sytuacje wyjątkowe system nie korzysta z bloków z deskryptorami i z bloku identyfikacyjnego z pozostałych grup fizycznych.

Prawa dostępu do plików i katalogów

Z każdym plikiem i katalogiem związane są prawa dostępu. Mają na celu umożliwić określenie uprawnień odczytu, edycji i uruchamiania dla użytkowników i grup użytkowników.

Kilka przykładów (po `ls -l`):

```
-rw-r--r--  1 user  users   /home/user/file
-rwxr-xr-x  1 root  root    /bin/bash
-rwsr-xr-x  1 root  shadow  /usr/bin/passwd
drwxrwxrwt 38 root  root    /tmp
drwxrws---  2 user  audio   /home/user/directory
```

Prawa dostępu (pierwsza kolumna):

`[-dlsfcb] rwx rwx rwx` *owner group*

Prawa dostępu przydzielane są dla kategorii:

- **u**ser – użytkownik, właściciel pliku,
- **g**roup – grupa, właściciel pliku,
- **o**ther – pozostali użytkownicy.

Dla każdej z tych kategorii możliwe są trzy prawa dostępu, opisane literowo (rwx) lub liczbowo trzema bitami:

- **r**ead – odczyt pliku, listowanie katalogu: wartość: $100_{(2)} = 4_{(8)}$,
- **w**rite – zapis, zmiana pliku, tworzenie i kasowanie plików w katalogu: wartość: $010_{(2)} = 2_{(8)}$,
- e**x**ecute – wykonywanie pliku, zmiana katalogu: wartość: $001_{(2)} = 1_{(8)}$.

Wartości liczbowe praw dostępu:

- `rwx` – 421, $4 + 2 + 1 = 7$,
- `r-x` – 4-1, $4 + 1 = 5$,
- `rw-` – 42-, $4 + 2 = 6$,
- `r-` – 4-, 4.

Typy plików:

- `-` – plik,
- `d` – katalog,
- `l` – dowiązanie symboliczne,
- `s` – gniazdo,
- `f` – kolejka FIFO,
- `c` – urządzenie znakowe,
- `b` – urządzenie blokowe.

Domyślne prawa dostępu dla nowo tworzonego pliku wynoszą 666, a katalogu 777. Można zmienić ustawienia domyślne za pomocą polecenia umask (prawa zdefiniowane w umask są odejmowane od domyślnych).

zmiana domyślnych praw dostępu

domyślne	rwx	rwx	rwx		rw-	rw-	rw-	
	7	7	7		6	6	6	
umask	–	-w-	-w-		–	-w-	-w-	
	0	2	2		0	2	2	
wynik	rwx	r-x	r-x		rw-	r–	r–	
	7	5	5		6	4	4	

Oprócz podstawowego zestawu uprawnień (`rwxrwxrwx`) do każdego pliku i katalogu przyporządkowane są jeszcze 3 bity określające specjalne prawa dostępu:

- **SUID**, setuid, SetUserID,
- **SGID**, setgid, SetGroupID,
- **Sticky**.

bity specjalne, pliki, katalogi

sticky	—	użytkownik może usunąć pliki których jest właścicielem lub gdy jest właścicielem katalogu w którym znajdują się pliki (/tmp)
SGID	program uruchamia się z identyfikatorem grupy właściciela	pliki tworzone w katalogu mają identyfikator grupy właściciela katalogu
SUID	program uruchamia się z identyfikatorem użytkownika właściciela	—

Narzędzia do pracy z plikami

Należy pamiętać, że:

- Linux rozróżnia wielkość znaków,
- nazwa pliku może mieć do 255 znaków, może zawierać znaki mające specjalne znaczenie (_, %), ale niektóre ($, ;, [SPACE]) muszą być poprzedzone znakiem backslash \,
- można używać znaków narodowych (zależne od konfiguracji lokalizacji).

zarządzanie plikami	
`cd`	zmiana katalogu
`ls`	listowanie zawartości katalogu
`rmdir`	kasowanie katalogu
`pwd`	wyświetlenie bierzącego katalogu
`touch`	zmiana daty modyfikacji i dostępu do pliku lub utworzenie nowego pliku
`mv`	przenoszenie plików
`cp`	kopiowanie plików
`rm`	kasowanie plików
`ln`	tworzenie linków

zmiana uprawnień plików	
chmod	zmiana uprawnień pliku
chown	zmiana właściciela pliku
umask	zmiana domyślnych praw dostępu

wyświetlanie plików	
`tail`	wyświetlania końcowych linii pliku
`head`	wyświetlania początkowych linii pliku
`cat`	wyświetlanie zawartości pliku
`grep`	przeszukiwanie zawartości pliku
`egrep`	przeszukiwanie zawartości pliku
`less`	stronicowanie (często używane z `cat`)

szukanie plików	
`find`	wyszukiwanie plików według kryteriów
`locate`	wyszukiwanie plików uwzględniając nazwę
`whereis`	wyszukiwanie plikó binarnych, manulai i źródeł
`which`	wyszukiwanie w ścieżkach przeszukiwania

Użytkownicy

Każdy użytkownik w systemie ma nazwę i swój unikalny numer **UID** (*User ID*), użytkownik *root* ma UID = 0.

Użytkownicy są łączeni w grupy. Grupy mają nazwy i unikalne numery **GID** (*Group ID*), grupa *root* ma GID = 0.

Dane o użytkownikach są przechowane w pliku `/etc/passwd`:

`User Name:Password:UID:GID:Comment:Home Directory:Login Shell`

```
# /etc/passwd
at:x:25:25:Batch jobs daemon:/var/spool/atjobs:/bin/bash
root:x:0:0:root:/root:/bin/bash
sshd:x:71:65:SSH daemon:/var/lib/sshd:/bin/false
wwwrun:x:30:8:WWW daemon apache:/var/lib/wwwrun:/bin/false
user:x:1000:100::/home/user:/bin/bash
```

Pierwotnie zaszyfrowane hasło użytkowników było w miejscu x, obecnie hasła znajdują się w pliku /etc/shadow:

User Name:Encrypted Password:Last Change:Next Possible Change:Next Obligatory Change:Warning:Limit:Lock

```
# /etc/shadow
at:*:15776:0:99999:7:::
root:$2y...woS:15776::::::
wwwrun:*:15385::::::
user:$2y...0nS:15776:0:99999:7:::
```

Hasła szyfruje się zwykle algorytmami **DES** lub **MD5**.

Znak * lub ! w polu hasła oznacza, że użytkownik nie może się zalogować (nieważne hasło).

Kolejne pola określają daty ważności i zmiany hasła. Na przykład 3 pole to liczba dni od 1970 roku, kiedy hasło zostało zmienione po raz ostatni.

Ustawienie jej na 0 wymusza zmianę hasła przy następnym logowaniu.

Poza domyślną grupą podaną w `/etc/passwd` użytkownik może należeć do dowolnej liczby innych grup.

Informacje o grupach zawarte są w pliku `/etc/group`:

```
# /etc/group
dialout:x:16:user
video:x:33:user
wheel:x:10:
www:x:8:
users:x:100:
```

zarządzanie użytkownikami i grupami	
`id`	ID użytkownika i grup użytkownika
`whoami`	wyświetla nazwę aktualnego użytkownika
`groups`	grupy użytkownika
`finger`	informacje o użytkowniku
`useradd`	dodanie użytkownika
`userdel`	usunięcie użytkownika
`usermod`	modyfikacja konta użytkownika
`passwd`	zmiana hasła
`groupadd`	dodanie grupy
`groupdel`	usunięcie grupy
`groupmod`	modyfikacja grup użytkownika
`gpasswd`	zmiana hasła grupy
`su`	zmiana użytkownika
`newgrp`	zmiana efektywnej grupy

Program sudo

sudo umożliwia wykonywanie poleceń z uprawnieniami innego użytkownika, najczęściej *root*, którego hasła nie trzeba znać.

Program sudo pozwala na:

- można przydzielać i zabierać uprawnienia,
- można określać prawa tylko do niektórych poleceń,
- można określać porę wykonywania poleceń,
- wykonywane polecenia są logowane do `/var/log/auth.log`.

Plik konfiguracyjny `/etc/sudoers` nie może być edytowany dowolnym edytorem tylko poleceniem `visudo`, które sprawdza jego składnię:

```
user host (users) = command1, command2, !command3, ...
```

W pliku konfiguracyjnym musi znajdować się linia: `root ALL=(ALL) ALL`

Przykładowy plik konfiguracyjny:

```
# /etc/sudoers
User_Alias ADMIN = user, dummy
Cmnd_Alias PRINT    = /usr/bin/lpc, /usr/bin/lprm
Cmnd_Alias SHUTDOWN = /sbin/shutdown

root ALL=(ALL) ALL
ADMIN ALL = PRINT, SHUTDOWN
user  ALL = NOPASSWD: /usr/bin/passwd [A-z]*, !/usr/bin/passwd root
user  ALL = (root) NOPASSWD: /sbin/fdisk
dummy sun = (operator) /bin/ls, (root) /bin/kill
```

Krótko o powłokach

Do komunikacji użytkownika z jądrem systemu operacyjnego służy **powłoka** systemu (*shell*), w linuksie jest dostępnych kilka powłok:

- The **C** shell (`/bin/csh`, często link do `/bin/tcsh`),
- The **TC** shell (`/bin/tcsh`),
- The **Korn** shell (`/bin/ksh`),
- The **Bourne** shell (`/bin/sh`, często link do `/bin/bash`),
- The **Bourne again** shell (`/bin/bash`).

Powłoka uruchomiana zaraz po zalogowaniu użytkownika nosi nazwę **login shell**.

Różne powłoki można uruchamiać w trakcie pracy z systemem.

Pliki konfiguracyjne powłoki login shell (**bash**):

- `/etc/profile` – plik dla całego systemu,
- `~/.profile` – dodatkowa konfiguracja dla każdego użytkownika,
- `/etc/bash.bashrc` – konfiguracja powłoki dla całego systemu (konfiguracja polecenia (aliasy systemowe),
- `~/.bashrc` – konfiguracja użytkownika.

W przypadku powłok uruchamianych z linii poleceń (**non-login shell**) konfiguracja jest pobierana z plików:

- `/etc/bash.bashrc`,
- `~/.bashrc`.

Klawisza `[TAB]` można używać do dokańczania poleceń lub nazw plików i katalogów.

W pliku `~/.bash_history` jest zapamiętana historia wpisywanych poleceń, można ją wyświetlić poleceniem `history`.

Powłoki pozwalają na definiowanie zmiennych, są one potrzebne do poprawnego działania samej powłoki jak również innych programów.

Zdefiniowane zmienne można wyświetlić poleceniem env.

ważniejsze zmienne	
PATH	ścieżka przeszukiwania poleceń
HOME	katalog domowy użytkownika
USER	nazwa aktualnego użytkownika
HOSTNAME	nazwa hosta (komputera)
SHELL	powłoka systemu

Pojedyncze zmienne można wyświetlić poleceniem echo.

Powłoka pozwala na definiowanie skrótów poleceniem alias.

```
$ alias dir='ls -l'
$ unalias dir
```

Strumienie i przekierowanie

Strumienie są kanałami, przez które aplikacja kontaktuje się z użytkownikiem i środowiskiem w jakim pracuje:

- **standard input** (`stdin`, `0`) – standardowe wejście, zwykle klawiatura,
- **standard output** (`stdout`, `1`) – standardowe wyjście, zwykle monitor,
- **standard error** (`stderr`, `2`) – kanał błędów, zwykle monitor.

Każdy strumień może być przekierowany:

przekierowania	
<	przekierowanie standardowego wejścia
>, 1>	przekierowanie standardowego wyjścia (nadpisanie)
», 1»	przekierowanie standardowego wyjścia (dołączanie)
2>, 2»	przekierowanie błędów

Przykłady:

```
$ ls /opt /dummy > file 2> /dev/null
$ ls /opt /dummy >> file 2> /dev/null
$ echo "Hello!" > greetings
```

Standardowe wyjście polecenia może być wejściem innego polecenia, służy do tego symbol | (*pipe*). Ponadto, wyjście z wielu poleceń można łączyć za pomocą nawiasów:

```
$ ls -l / | wc -l
$ ls -l /etc | less
$ (id; ls -l /var) > outputd
```

Każde polecenie zwraca kod mówiący o stanie po jego wykonaniu. Wartość 0 oznacza sukces, wartości większe oznaczają błąd. Zwracana wartość jest przechowywany w zmiennej ?.

```
$ ls /
$ echo $?
0
```

Zwracana wartość może być użyta do kontrolowania wykonania kolejnych poleceń.

łączenie poleceń	
command1 && *command2*	polecenie *command2* zostanie uruchomione jeśli polecenie *command1* wykona się bez błędów
command1 || *command2*	polecenie *command2* zostanie uruchomione jeśli polecenie *command1* wykona się z błędami

Bardzo krótko o skryptach powłoki

Skrypty powłoki są plikami tekstowymi zawierającymi polecenia do wykonania.

Pliki powinny mieć prawo do wykonywania `x`, można je nadać wydając polecenie `chmod u+x` *script*.

Skrypt można także uruchomić w innej (kolejnej) powłoce `sh` *script* (ten sposób nie wymaga prawa do wykonywania).

Skrypty mogą (powinny) być umieszczane w katalogu, który będzie przeszukiwany. Zwyczajowo jest to katalog `~/bin`. Aby dopisać katalog do `PATH`, należy dodać `export PATH=$PATH:~/bin` do pliku `~/.bashrc`.

Zmienne, czytanie wejścia, podstawienia:

```
#!/bin/bash

STR="Hello!"
read NAME

echo "$STR $NAME, today is `date +%d-%m-%Y`"
echo "Number of parameters: $#"
echo "Script name: $0"
echo "First parameter: $1"
```

Podstawienia, operacje arytmetyczne:

```
#!/bin/bash

A=5
B=$A+3       # 5+3
let C=$A+3   # 8
((D=$A+3))   # 8
E=$(($A+3)) # 8

declare -i F
declare -i G

F=5
G=F+3
```

Instrukcja warunkowa:

```
#!/bin/bash

if [ $# -lt 2 ]
then
  echo "Need more than 2 parameters"
fi

if [[ $# < 2 ]]; then ...; fi

if grep -q expr file
then
  echo "File contains expr"
else
  echo "File does not contain expr"
fi
```

Pętle:

```
#!/bin/bash

INT=0
while [ $INT -lt 5 ]
do
  INT=$((INT+1))
  echo "i=$INT"
done

for I in aaa "bbb ccc" "eee" "fff"
do
  echo $I
done
```

```
#!/bin/bash

for I in $(ls)
do
  echo "File: $I"
done

for I in `seq 50`
do
  kill $((I+300))
done
```

Przypomnienie: adresy IP i maski podsieci

Każdy komputer podłączony di sieci Internet posiada adres IP (dla protokołu IPv4 w formacie aaa.bbb.ccc.ddd).

Adresy w Internecie przydziela organizacja o nazwie IANA (*Internet Assigned Numbers Authority*).

Dla uzyskania komunikacji **TCP/IP** (na poziomie lokalnym), komputer musi mieć co najmniej skonfigurowany **adres IP** i **maskę sieci**.

adresy					
Adres IP	192.168.3.77	11000000	10101000	00000011	01001101
Maska	255.255.255.0	11111111	11111111	11111111	00000000
Sieć	192.168.3.0	11000000	10101000	00000011	00000000
Broadcast	192.168.3.255	11000000	10101000	00000011	11111111

Standardy numeracji definiują pule prywatnych adresów IP, wykorzystuje się je często w sieciach lokalnych:

- 10.0.0.0 - 10.255.255.255 – jedna klasa **A**,
- 172.16.0.0 - 172.31.255.255 – 16 klas **B**,
- 192.168.0.0 - 192.168.255.255 – 256 klas **C**.

Jądro udostępnia interfejs **loopback** (pętla zwrotna, pseudosieć). Interfejs ten dostaje adres IP `127.0.0.1`.

Komputer łącząc się z tym adresem, połączy się sam ze sobą. Interfejs umożliwia lokalne uruchamianie usług, które wymagają sieci.

Karty sieciowe są oznaczane symbolami **eth*X*** (`eth0`, `eth1`, `eth2`, ...).

Przykład konfiguracji i włączenia interfejsu sieciowego:

```
$ ip addr add 192.168.0.2/24 brd + dev eth0
$ ip link set eth0 up
$ ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
    state UP qlen 1000
    link/ether 08:00:27:d0:ba:59 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.2/24 brd 192.168.0.255 scope global eth0
    inet6 fe80::a00:27ff:fed0:ba59/64 scope link
       valid_lft forever preferred_lft forever
$ ip route add default via 192.168.0.1
$ ip route show
default via 192.168.0.1 dev eth1
192.168.0.0 dev eth1 proto kernel scope link src 192.168.0.2
127.0.0.0/8 dev lo scope link
```

konfiguracja i testowanie sieci	
`ip addr`	ustawienia adresów IP
`ip link`	konfiguracja kart sieciowych
`ip route`	ustawienia tablicy rutingu
`arp`	zarządzanie tablicą ARP
`ping`	wysyłanie pakietów ICMP
`traceroute`	wyświetlanie marszruty pakietów
`host`	informacje DNS
`nslookup`	odpytywanie serwerów DNS
`dig`	odpytywanie DNS
`ifup`	włączenie karty sieciowej
`ifdown`	wyłączenie karty sieciowej
`/etc/sysdonfig/network/*`	konfiguracja kart sieciowych
`/etc/host.conf`	konfiguracja nazwy komputerów
`/etc/reslov.conf`	adresy serwerów DNS