Komputer jako automat (FSM)

Architektura komputera – zbiór rozkazów R i pamięć $\boldsymbol{M}$: $\mathbf{A}=(\boldsymbol{M},\mathsf{R})$

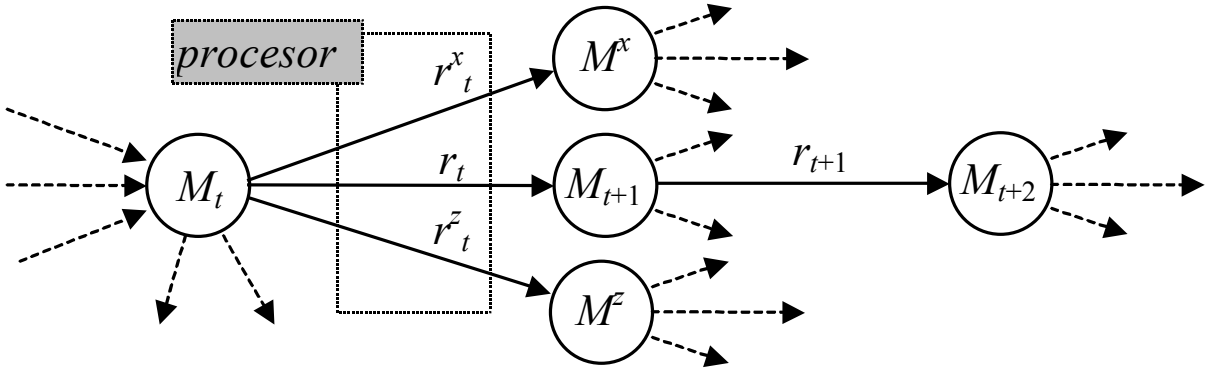
Stan procesora – zawartość *rejestrów* procesora $\boldsymbol{G}$.

Stan komputera – superpozycja *stanu procesora* $\boldsymbol{G}$ oraz *stanu pamięci* $\boldsymbol{S}$

$$M_t=G_t \mid S_t=\{g_j(t); j\in\mathbb{N}\} \mid \{s_i(t);\ i\in\mathbb{Z}\}\in\boldsymbol{M}$$

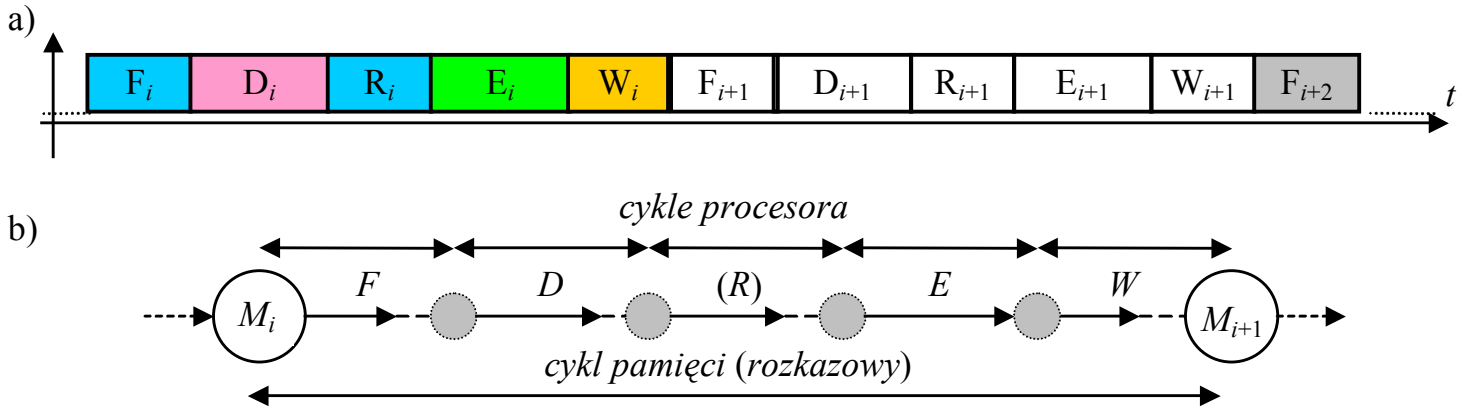
Rozkaz – funkcja $r_x\in\mathsf{R}$, która odniesiona do *stanu początkowego* (*komputera*) wytwarza *stan wyjściowy komputera*

$$r_x:\boldsymbol{M}\to\boldsymbol{M},\quad r_x\in\mathsf{R},\quad M_{t+1}=r_x(M_t)\in\boldsymbol{M}$$



Cykl pamięci i cykle procesora

Cykl – czas upływający pomiędzy kolejnymi zmianami stanu



Cykle (etapy) wykonania rozkazu a) diagram czasowy, b) graf automatu.

Fazy (etapy) wykonania rozkazu:

- *F* (*fetch*) – pobranie kodu rozkazu z pamięci
- *D* (*decode*) – dekodowanie w celu wytworzenia sygnałów sterujących
- *R* (*read*) – pobranie operandu z pamięci (opcjonalne)
- *E* (*execute*) –wytworzenie wyniku (zmienny czas)
- *W* (*write*) – zwrotny zapis wyniku wykonania (do pamięci lub rejestru)

Nawigacja - gdzie są potrzebne słowa?

tryb *control flow* – polecenie (rozkaz) czeka na dane

- najpierw musi być pobrane słowo polecenia
  - musi być ustalona lokacja pierwszego pobieranego słowa
- z interpretacji słowa wynika zapotrzebowanie na argumenty
- kolejne słowo polecenia musi być wskazane:
  - w treści polecenia bieżącego
  - automatycznie → *licznik rozkazów* (*program counter*)

typowe realizacje procesorów

tryb *data flow* – dane czekają na polecenie

- osobne strumienie danych i poleceń
- zestawy poleceń tworzą sekwencję opisaną przez algorytm
  - możliwe przetwarzanie współbieżne
- zestawy danych są (częściowo) uporządkowane
  - możliwe jednoczesne przetwarzanie danych przez kilka rozkazów

! *brak komercyjnych realizacji procesorów dla trybu dataflow*

powszechne realizacje jednostek wykonawczych

Sterowanie wykonaniem programu – udany kompromis

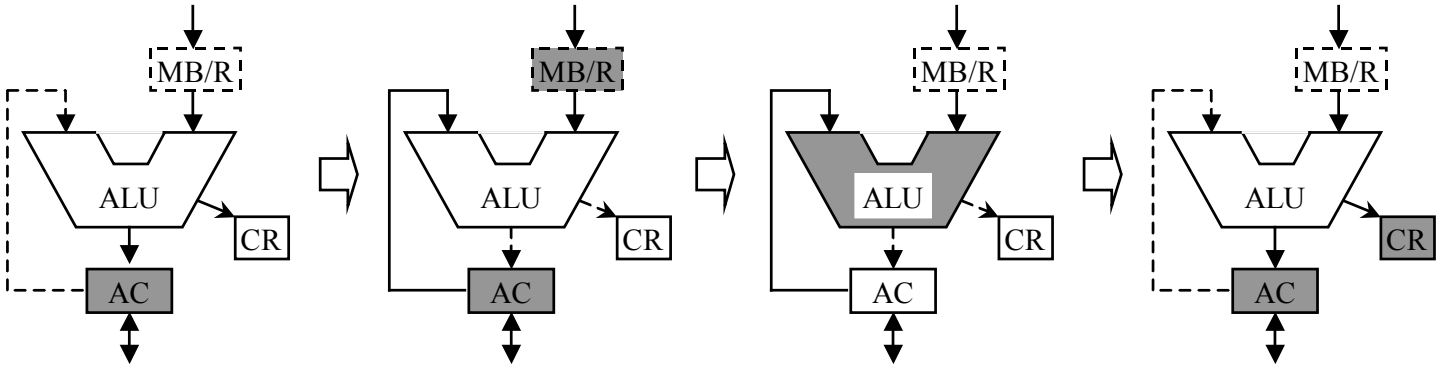
Adres (lokacja) kolejnego słowa polecenia

- wskazana automatycznie (*licznik rozkazów*) – powiązanie *sekwencyjne*
  - wyklucza sterowanie – kolejne polecenie nie zależy od wyniku poleceń wcześniejszych
  - oszczędne – w kodzie polecenia nie jest potrzebny adres kolejnego polecenia
- wskazana w treści polecenia bieżącego – powiązanie *łańcuchowe*
  - rozwiązanie elastyczne, sterowanie dowolne
  - niepotrzebna rozbudowa kodu większości poleceń

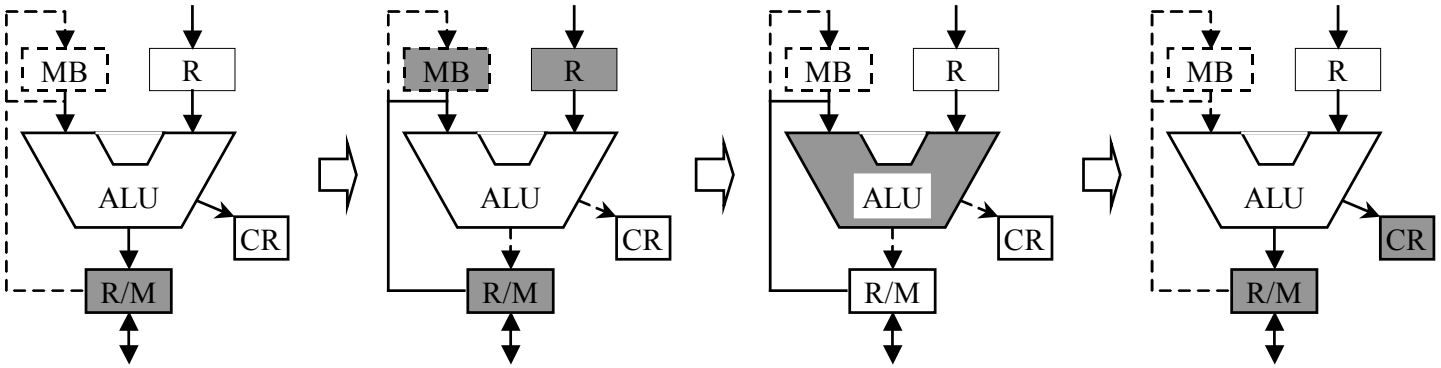
Kompromis

- zasada – powiązanie *sekwencyjne* $\rightarrow$ *licznik rozkazów*
- wyjątek – powiązanie *łańcuchowe* $\rightarrow$ możliwość sterowania
  - specjalne rozkazy skoku (*jump*)/rozgałęzienia (*branch*) $\rightarrow$ wymuszanie stanu *licznika rozkazów*

Architektura jednostki wykonawczej procesora

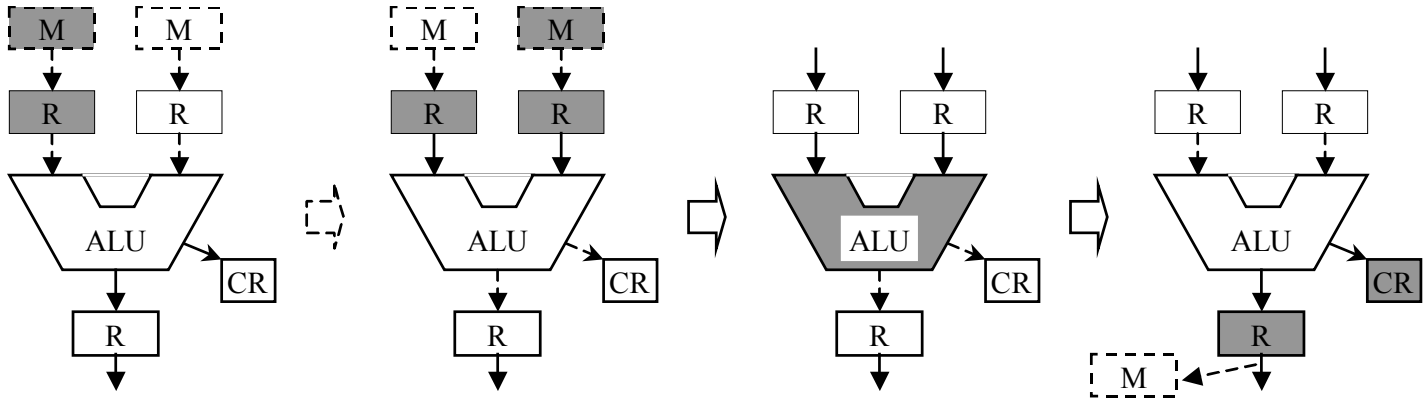


organizacja akumulatorowa i rozszerzona (AC → R)

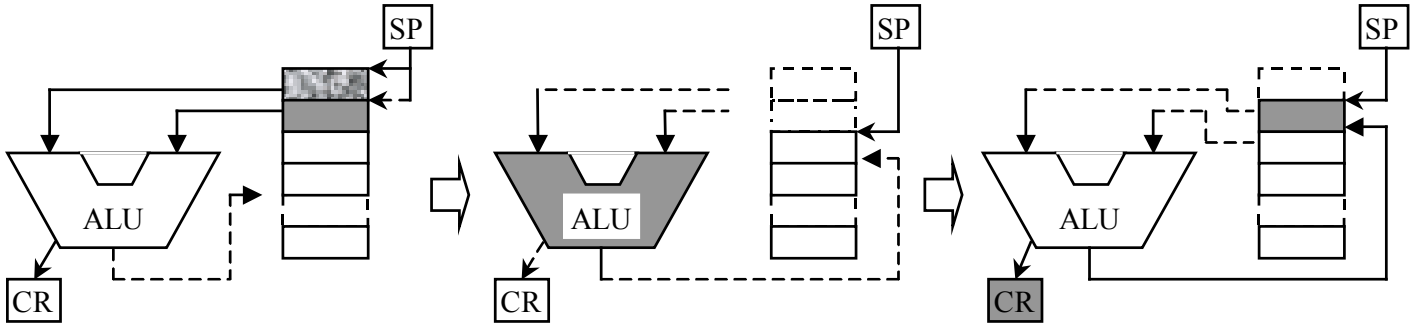


organizacja akumulatorowa uogólniona (AC → M/R)

Rejestrowa i stosowa architektura jednostki wykonawczej procesora



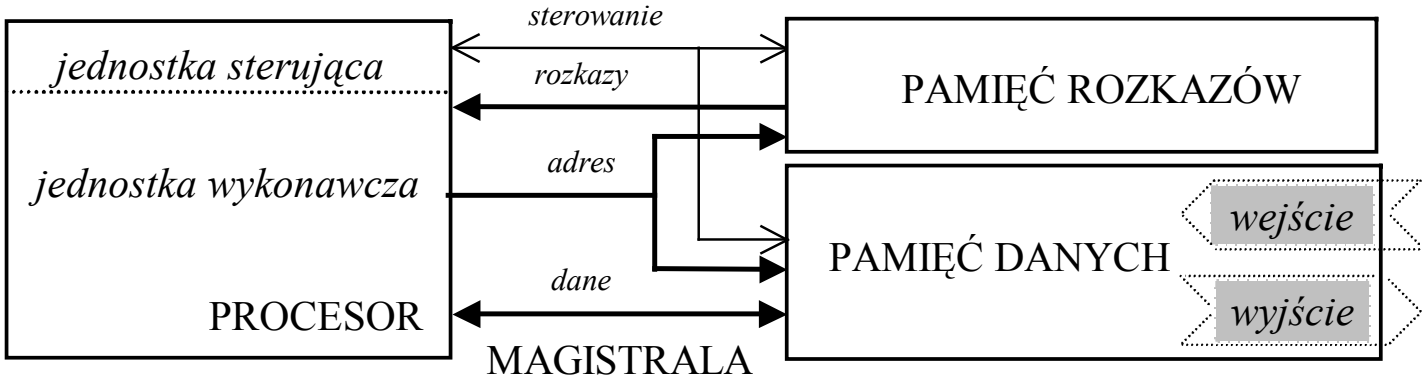
organizacja rejestrowa (*load–store*) i uniwersalna (R → R/M)



organizacja stosowa

Architektura harwardzka

(*H.Aiken*, 1949)



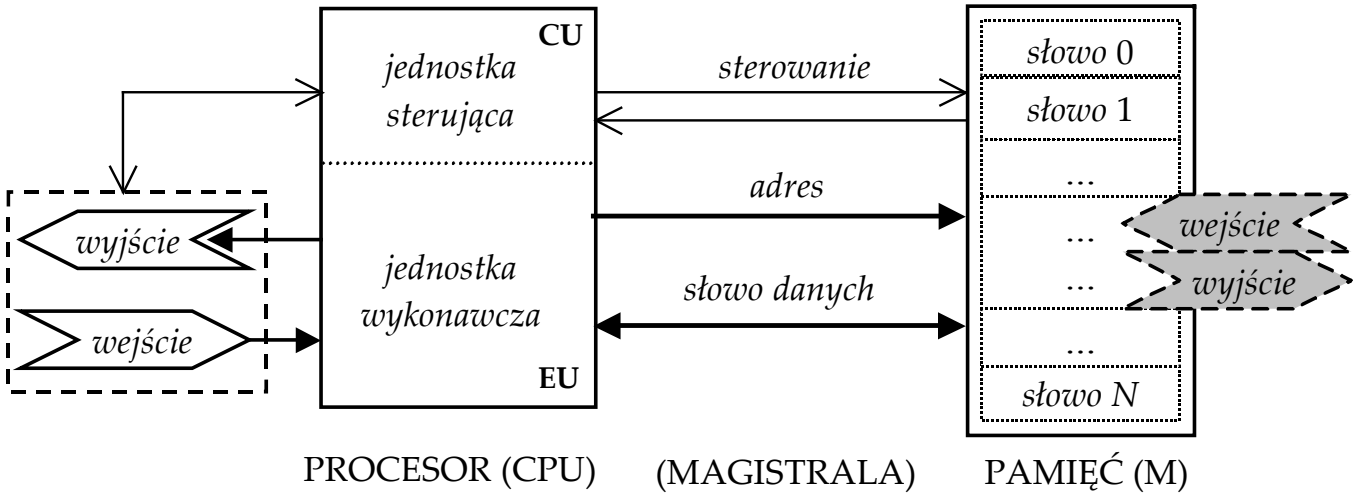
- osobne bloki pamięci danych i pamięci kodu (rozkazów)

zalety	wady
brak konfliktu dostępu do pamięci	oznaczanie danych
ochrona kodu programu	podwójne magistrale
łatwe buforowanie pamięci	mała elastyczność odwzorowania

Ewolucja koncepcji komputera z programem przechowywanym

memory-mapped I/O

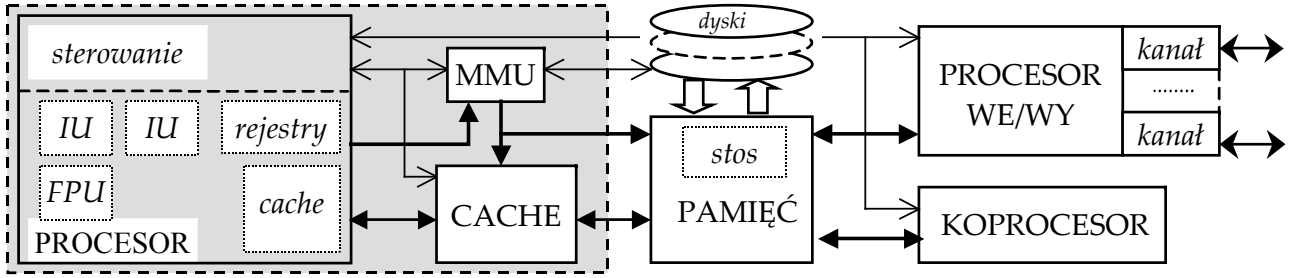
- zintegrowana jednostka centralna CPU
- jednolite adresowanie pamięci oraz we/wy
- uproszczony przepływ sterowania, adresów i danych



Rozszerzenia architektury klasycznej

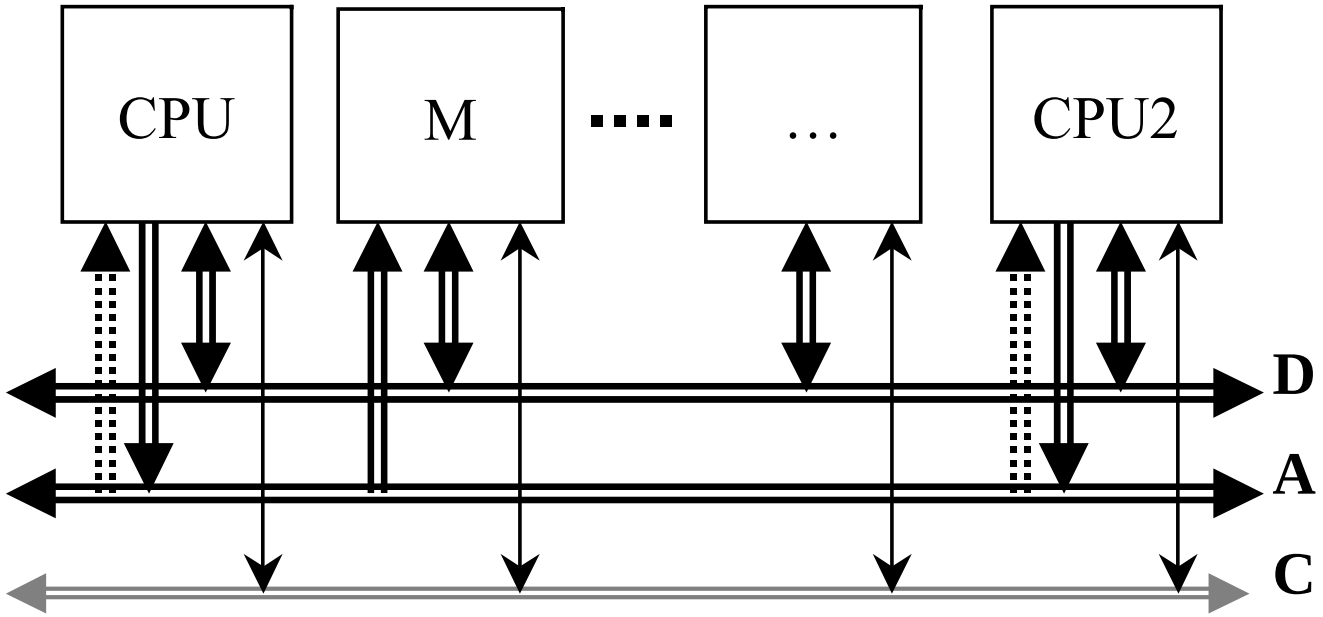
niesprzeczne z modelem von Neumanna (*zachowana zasada działania*)

- rozbudowa systemu pamięci
- rozbudowa jednostki wykonawczej



- rejestry robocze
- pamięć wirtualna i układ zarządzania MMU (ang. *memory management unit*)
- pamięć masowa
- pamięć podręczna zewnętrzna (CACHE) i wewnętrzna (*cache*)
- kilka jednostek wykonawczych
- wspomaganie specyficznych działań – koprocesory

Architektura magistralowa



- magistrala adresowa
- magistrala danych
- magistrala sterowania
- standard przyłączenia
- niezbędny protokół dostępu
- konieczny arbitraż

Architektura klasyczna CISC

CISC – Complex Instruction Set Computer

Lista rozkazów

- rozkazy realizujące działania proste i skomplikowane
- rozbudowane sposoby (tryby) adresowania
- argumenty umieszczone zwykle w pamięci
- stałe w dodatkowych słowach kodu
- niejednolita struktura – różna liczba słów kodu

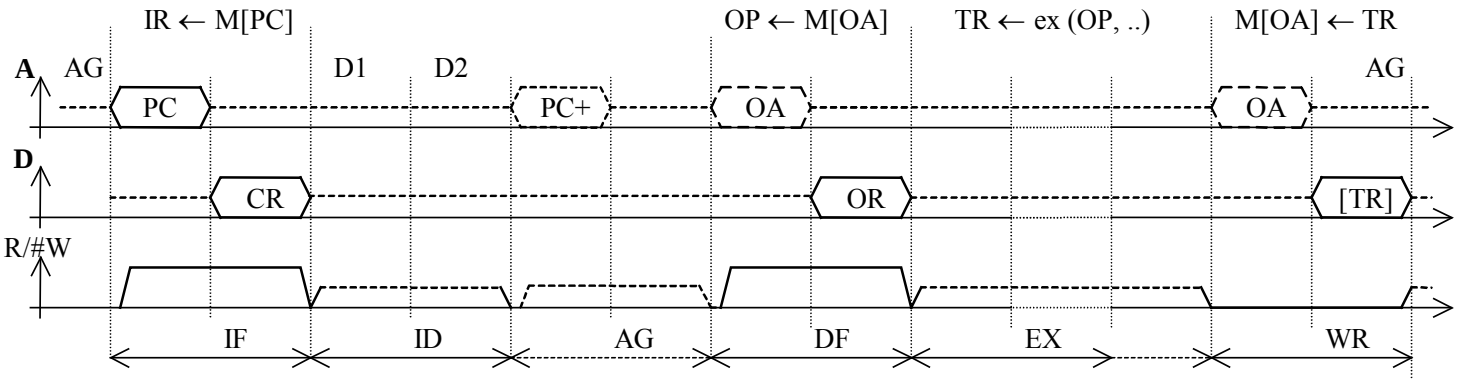
Organizacja – rozwiązania intuicyjne

- akumulator lub niewiele rejestrów uniwersalnych
- większość argumentów w pamięci, rejestry specjalizowane
- trudne buforowanie i dekodowanie rozkazów (zmienny rozmiar)

Skutki

- zmienny czas wykonania tych samych etapów przetwarzania
- bariera przepustowości pamięci

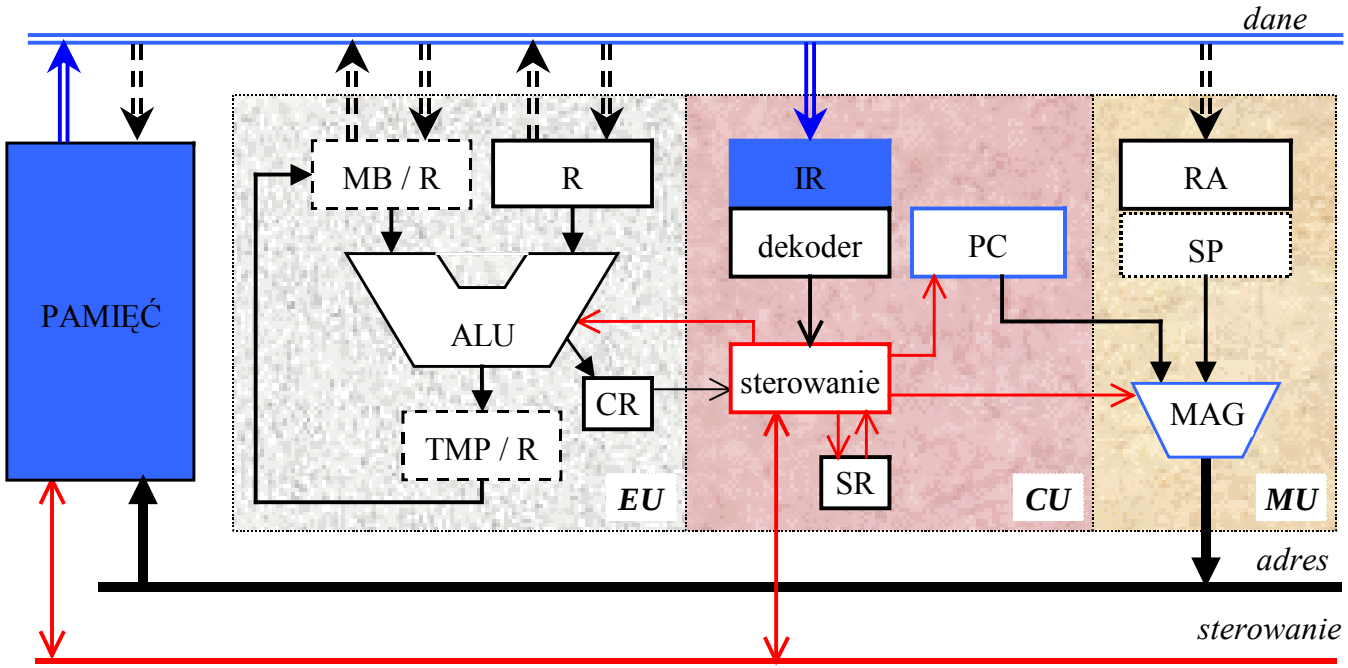
Cykl rozkazowy (CISC)



(architektura R/M)

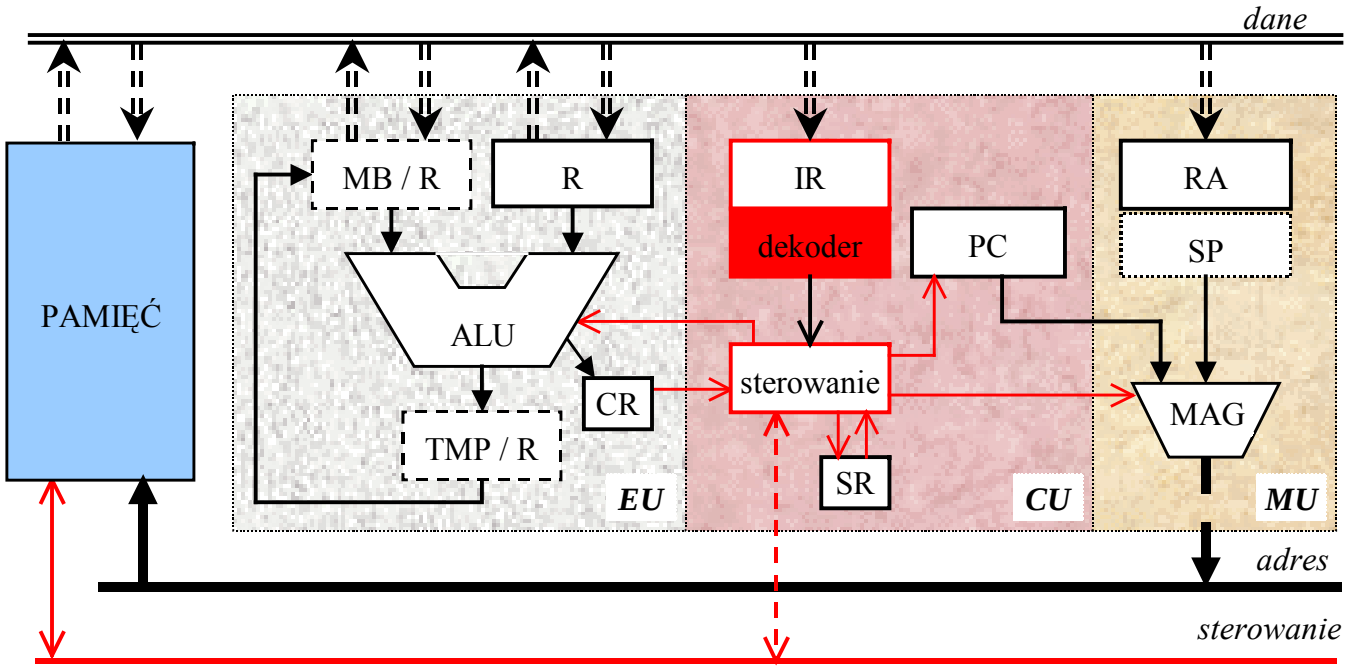
- pobranie kodu rozkazu z pamięci do rejestru rozkazów (ang. *instruction fetch*, IF)
- dekodowanie zawartości rejestru rozkazów (ang. *instruction decode*, ID)
- wytworzenie adresu operandu (ang. *address generation,* AG)
- pobranie operandów z pamięci (ang. *data fetch*, DF)
  - pobranie adresu skoku (ang. *target instruction address*, TA)
- wykonanie (ang. *execute*, EX)
- zapis wyniku (ang. *write*, WR) do pamięci (M[OA]) lub do rejestru (ang. *put away*)

Organizacja procesora sekwencyjnego (F)



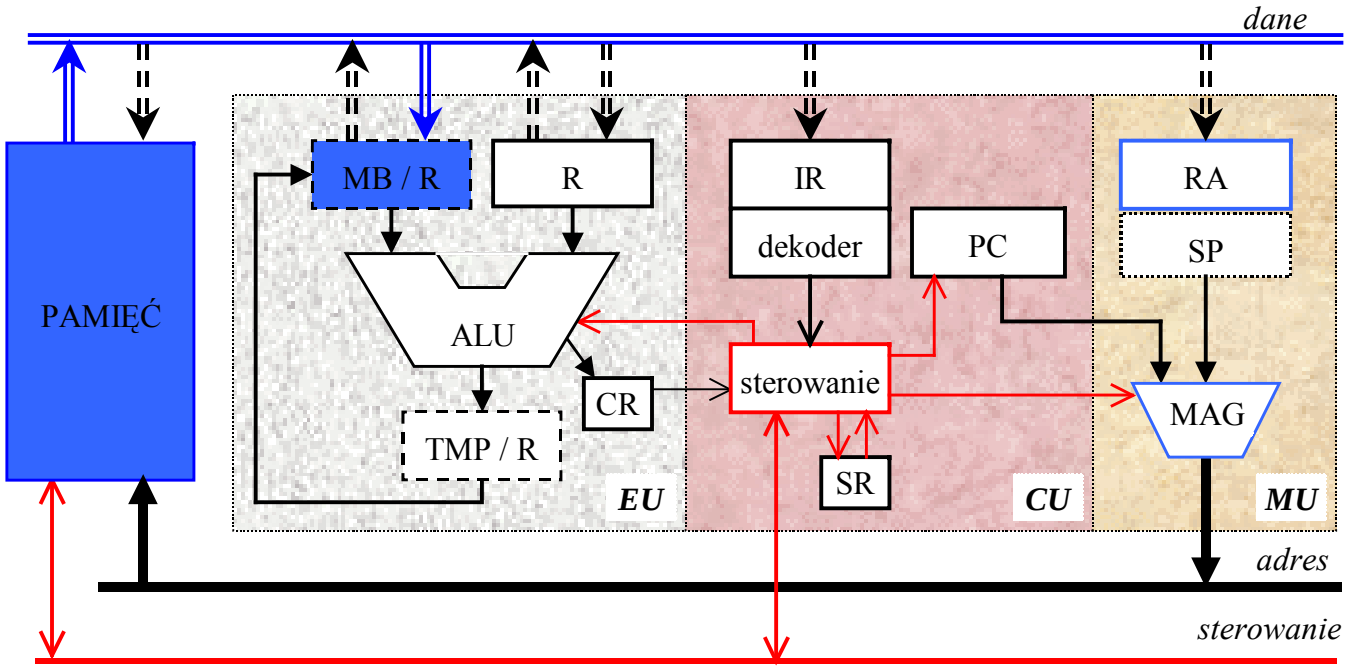
- *F* (pobranie kodu): adres (PC) → pamięć → rejestr rozkazów IR

Organizacja procesora sekwencyjnego (D)



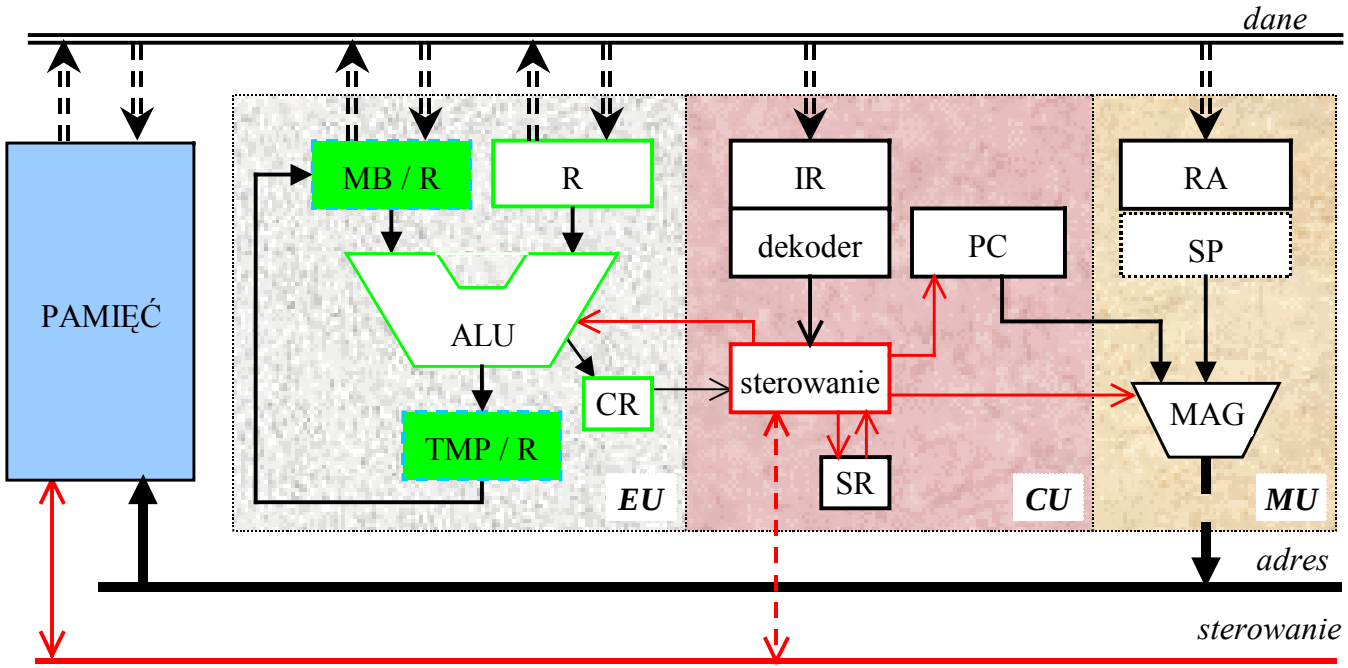
- *D* (dekodowanie) : rejestr rozkazów IR → sterowanie (CR,SR)

Organizacja procesora sekwencyjnego (R)



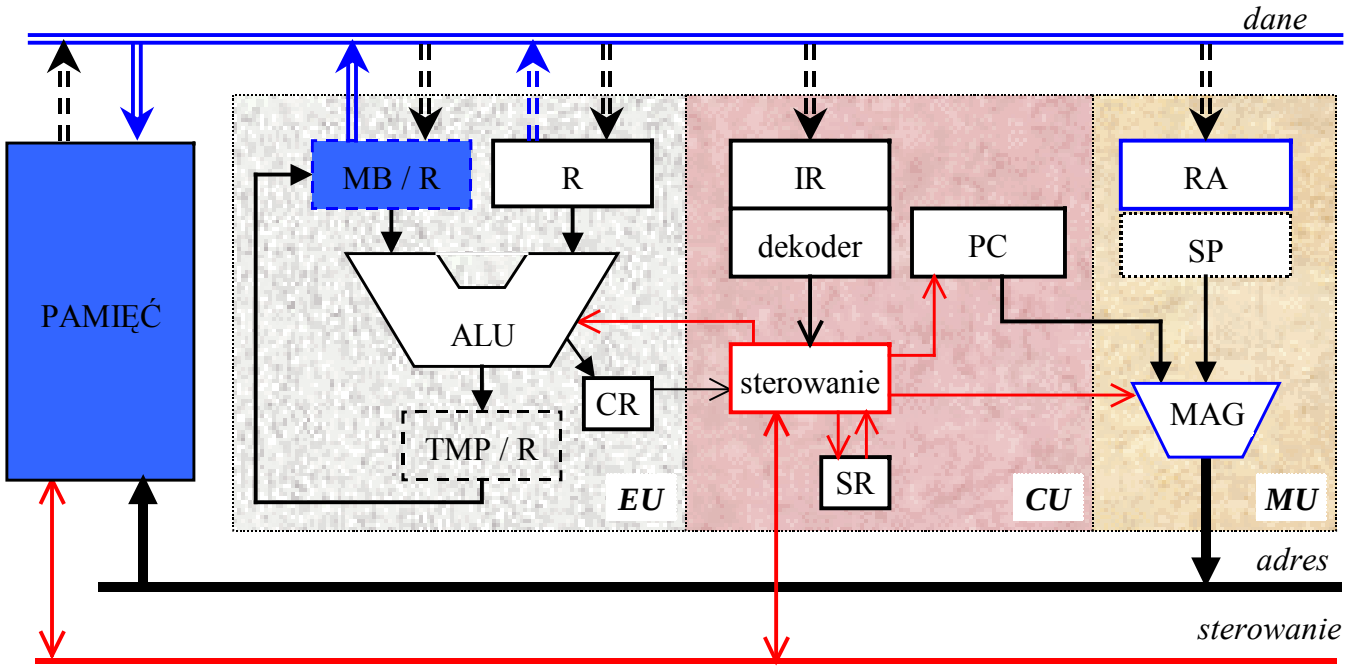
- *R* (odczyt danej): adres (RA) → pamięć → bufor (rejestr) MB

Organizacja procesora sekwencyjnego (E)



- *E* (wytworzenie wyniku): ALU (MB/R, R) → bufor TMP/ rejestr R

Organizacja procesora sekwencyjnego (W)



- *W* (zapis wyniku): bufor MB → pamięć (adres (RA))

Koncepcja przetwarzania potokowego

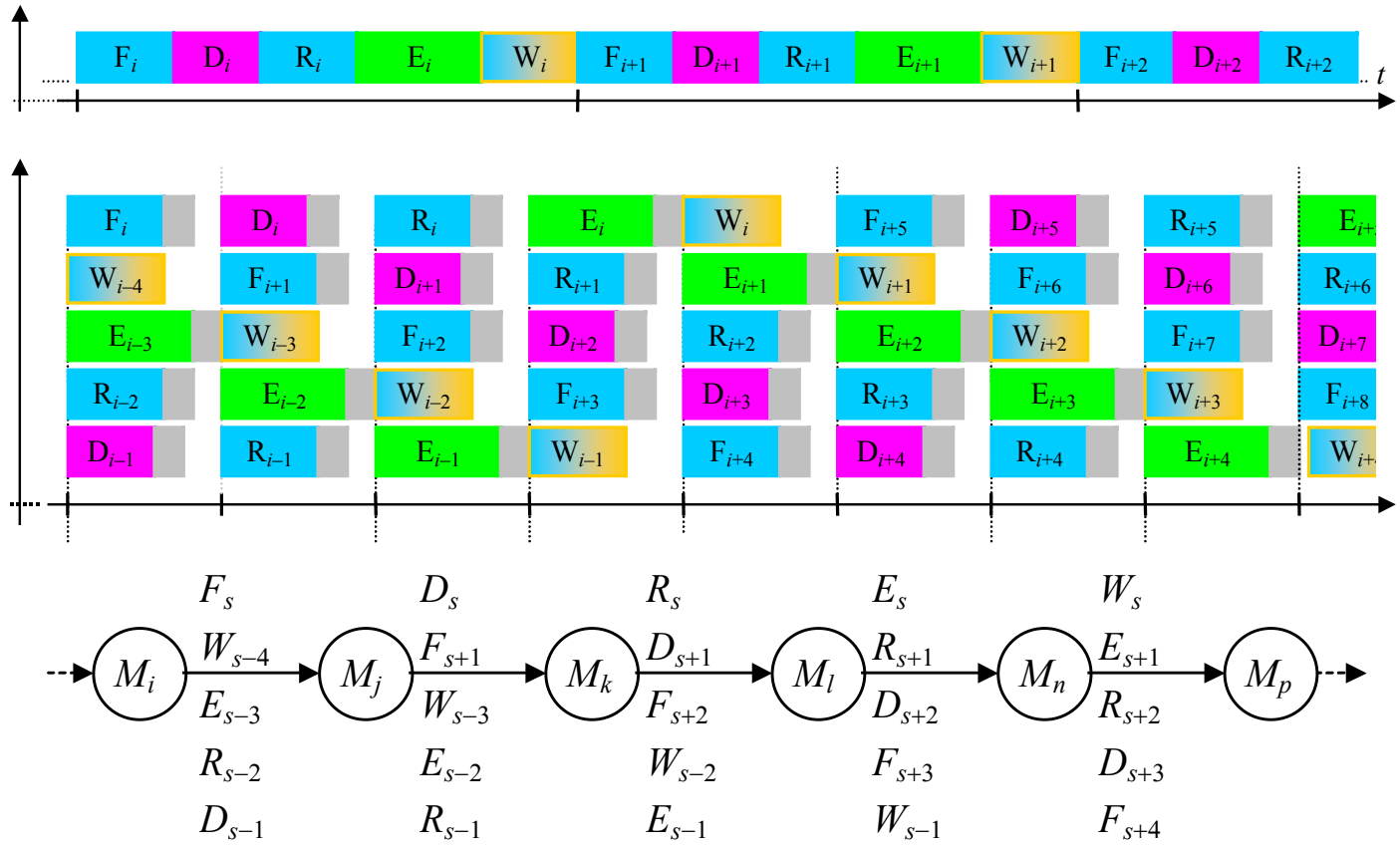
Każdy etap cyklu rozkazowego wykonuje specjalizowany układ funkcjonalny:

- pobranie rozkazu **z pamięci** (ang. *fetch*) – wskaźnik rozkazów i układ odczytu
  - rozmiar kodu rozkazu
  - czas dostępu do pamięci (odczyt słowa lub jego części)
- **dekodowanie** (ang. *decode*) – dekoder rozkazu
  - architektura listy rozkazów (złożoność i różnorodność działań)
  - struktura kodu rozkazu (niejednorodność)
- pobranie operandu **z pamięci** (ang. *data read*) – układ adresowania i odczytu
  - tryb adresowania i czas dostępu do pamięci (odczyt)
- **wykonanie** (ang. *execute*) – układ/układy wykonawcze (ALU/FPU/...)
  - złożoność wykonywanych działań
- zapis wyniku do **rejestru** (ang. *put away*) [lub **do pamięci** (ang. *data write*)]
  [czas dostępu do pamięci (zapis słowa lub jego części)]

Przepływ danych między układami funkcjonalnymi jest **jednokierunkowy**

- możliwe *jednoczesne wykonanie różnych etapów*
- szybkość wykonania ogranicza *najdłuższy etap* i *narzut separacji etapów*

Procesor potokowy



Cykle procesora potokowego (w uproszczeniu)

Analiza wykonania rozkazów w rozwiązaniach klasycznych (CISC)

analiza:
rozbudowana lista rozkazów i nieregularna struktura kodu
→ zmienny czas pobrania kodu i dekodowania rozkazu
→ skomplikowany dekoder (układ sekwencyjny)
→ skomplikowane układy wykonawcze, zmienny czas wykonania działań
większość operandów w pamięci
→ częste konflikty dostępu podczas wykonania etapów F, R i W
→ długi czas wykonania etapów F, R i W

wnioski:
→ uprościć listę rozkazów
→ ograniczyć repertuar trybów adresowania danych
→ ujednolicić strukturę słowa kodu
→ zwiększyć liczbę rejestrów procesora

potencjalna korzyść – łatwiejsza implementacja przetwarzania potokowego

Ewolucja CISC → RISC

oczekiwane efekty:

uproszczenie listy rozkazów oraz ujednolicenie struktury słowa kodu:

→ stały czas pobrania kodu i dekodowania rozkazu

→ prosty dekoder kombinacyjny

→ proste układy wykonawcze, krótki czas wykonania podstawowych działań

zwiększenie liczby rejestrów procesora – przechowanie wyników cząstkowych:

→ mniejsza intensywność komunikacji z pamięcią etapy R i W (load/store)

→ rzadkie konflikty dostępu podczas wykonania etapów F, R, W

ograniczenie repertuaru trybów adresowania danych –

→ prostszy układ obliczania adresu

problem: kodowanie argumentów stałych:

→ większość możliwych stałych jest bardzo rzadko używana

→ stałe często używane mają małe wartości

wniosek: skrócone kodowanie stałych często używanych,
rzadko używane można tworzyć proceduralnie

Koncepcja RISC

RISC – Reduced Instruction Set Computer

!***redukcja*** *dotyczy* ***różnorodności*** *a nie liczby instrukcji* (*racjonalizacja*)!

Lista rozkazów

- rozkazy proste
- proste tryby adresowania
- specjalne rozkazy komunikacji z pamięcią
- ograniczony zakres i krótkie kody stałych
- jednolita struktura kodu

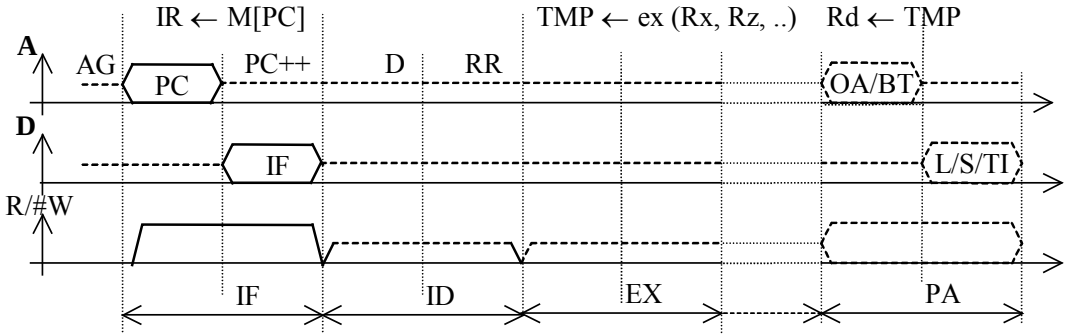
Organizacja (zasada lokalności, buforowanie informacji)

- dużo rejestrów uniwersalnych
- buforowanie informacji (kolejka rozkazów, cache)

Korzyści

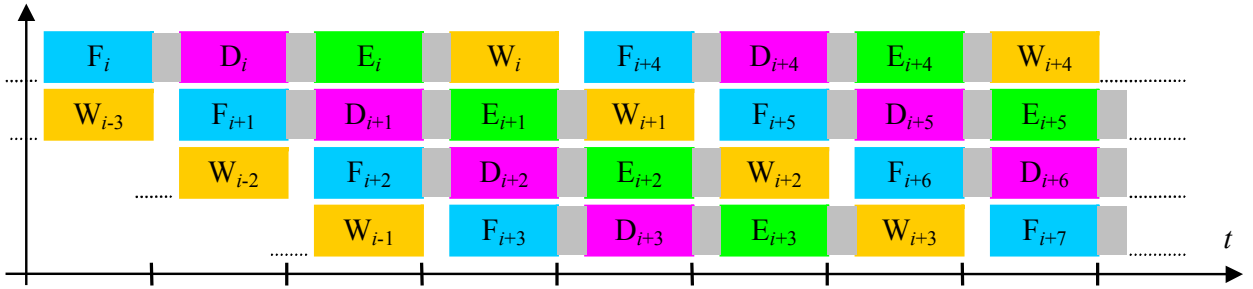
- prawie stały czas wykonania tych samych etapów przetwarzania
- podobny czas wykonania różnych etapów przetwarzania
- łatwa implementacja potoku

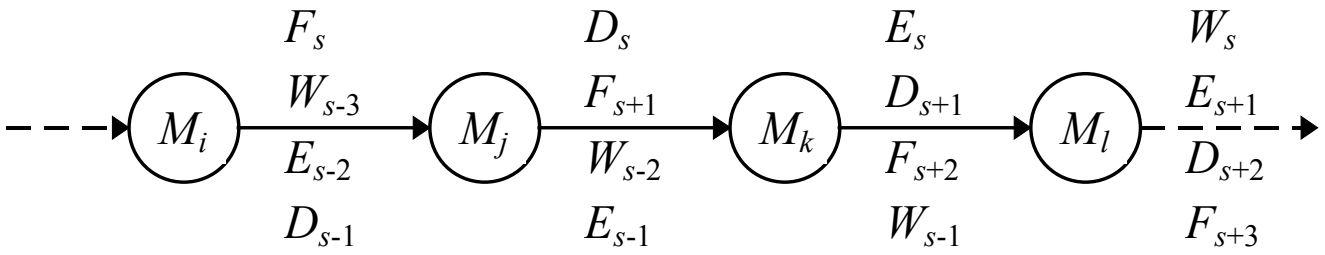
Cykl rozkazowy (RISC)



- pobranie kodu rozkazu z pamięci do rejestru rozkazów (*instruction fetch*, IF)
- dekodowanie zawartości rejestru rozkazów (*instruction decode*, ID)
  - dostarczenie danych z rejestrów
- wykonanie działania arytmetycznego lub logicznego (*execute*, EX) lub
  - L/S – wytworzenie adresu danej (*operand address*, OA)
  - BR – wytworzenie adresu docelowego skoku (*branch target address*, BT)
- umieszczenie wyniku w rejestrze albo w buforze (*put away*, PA) lub
  - L/S – pobranie z pamięci danej (*load*) lub zapis danej do pamięci (*store*)
  - BR – pobranie instrukcji docelowej skoku (*target instruction fetch*, TI)

Procesor potokowy RISC





Cykle procesora potokowego RISC

Potokowe wykonanie rozkazów w architekturze RISC

Uproszczona lista rozkazów oraz stały rozmiar i struktura słowa kodu,
→ prosty dekoder kombinacyjny
→ proste układy wykonawcze, krótki czas wykonania działań

Większość operandów w rejestrach procesora
→ wyeliminowanie etapu R i krótszy czas etapu W (load/store)
→ konflikty dostępu tylko podczas wykonania rozkazów load/store (E/W)

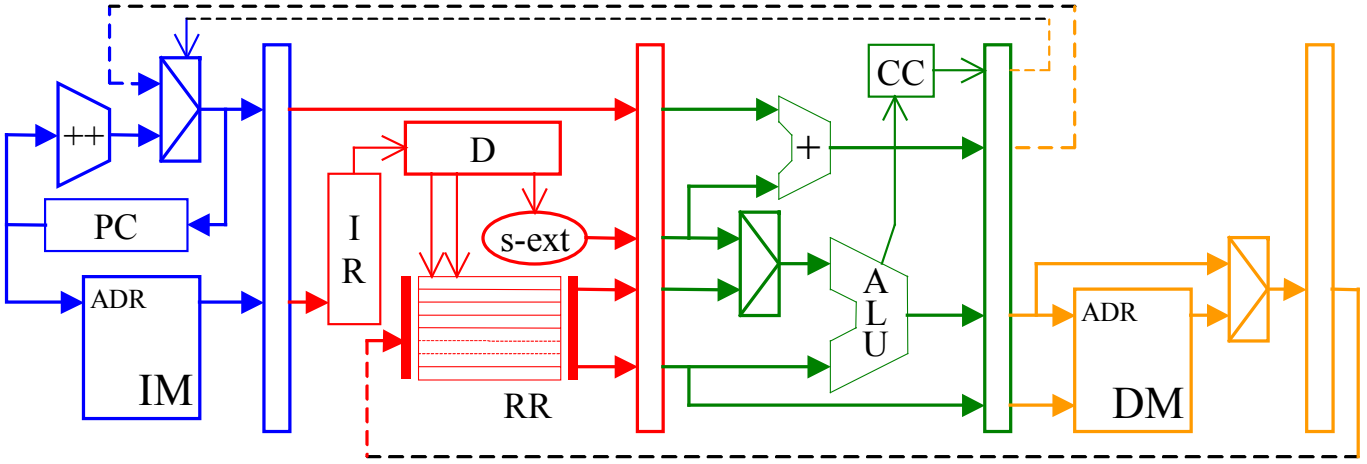
Niezbędne skrócenie czasu pobierania kodu z pamięci
→ buforowanie pamięci

Nieuniknione przestoje (ang. *buble*) wskutek konfliktów (ang. *hazards*)
→ spekulacyjne wykonanie niektórych rozkazów
→ zmiana kolejności przetwarzania rozkazów

Przypuszczenie
podział etapów F, D, E, W na podetapy → wzrost przepustowości …

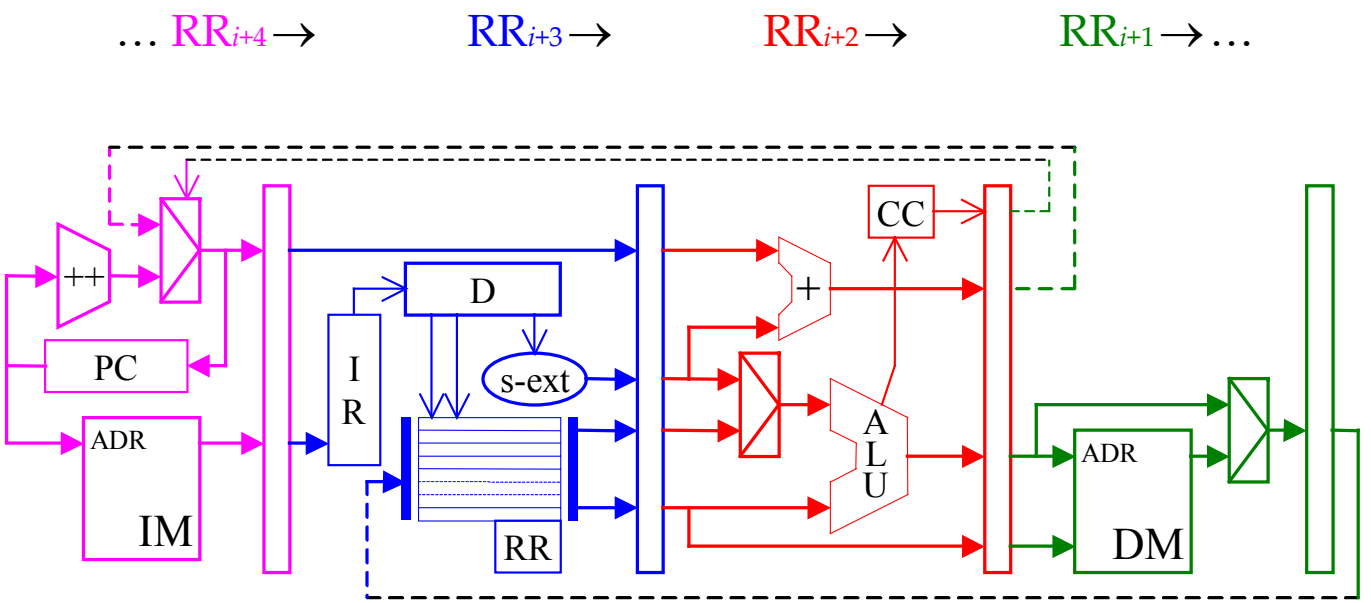
Organizacja procesora potokowego (rejestrowa)

… RR_{i+3} → RR_{i+2} → RR_{i+1} → RR_i → …



- *F*: RR_{i+3} (IM – bufor pamięci podręcznej kodu, PC – licznik rozkazów)
- *D*: RR_{i+2} (IR – rejestr rozkazów, D – dekoder, RR – plik rejestrów)
- *E*: RR_{i+1} (ALU – jednostka arytmetyczno-logiczna, CC – kod warunków)
- *W*: RR_i (DM – bufor pamięci podręcznej danych (*load*/*store*))

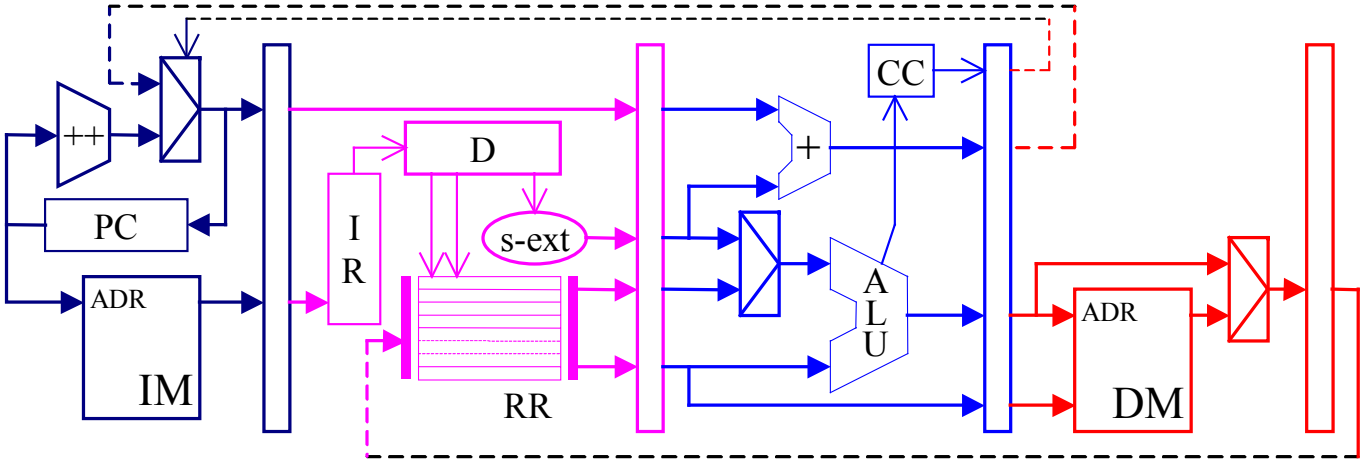
Organizacja procesora potokowego (rejestrowa)



- *F*: RR$_{i+4}$ (IM – bufor pamięci podręcznej kodu, PC – licznik rozkazów)
- *D*: RR$_{i+3}$ (IR – rejestr rozkazów, D – dekoder, RR – plik rejestrów)
- *E*: RR$_{i+2}$ (ALU – jednostka arytmetyczno-logiczna, CC – kod warunków)
- *W*: RR$_{i+1}$ (DM – bufor pamięci podręcznej danych (*load*/*store*))

Organizacja procesora potokowego (rejestrowa)

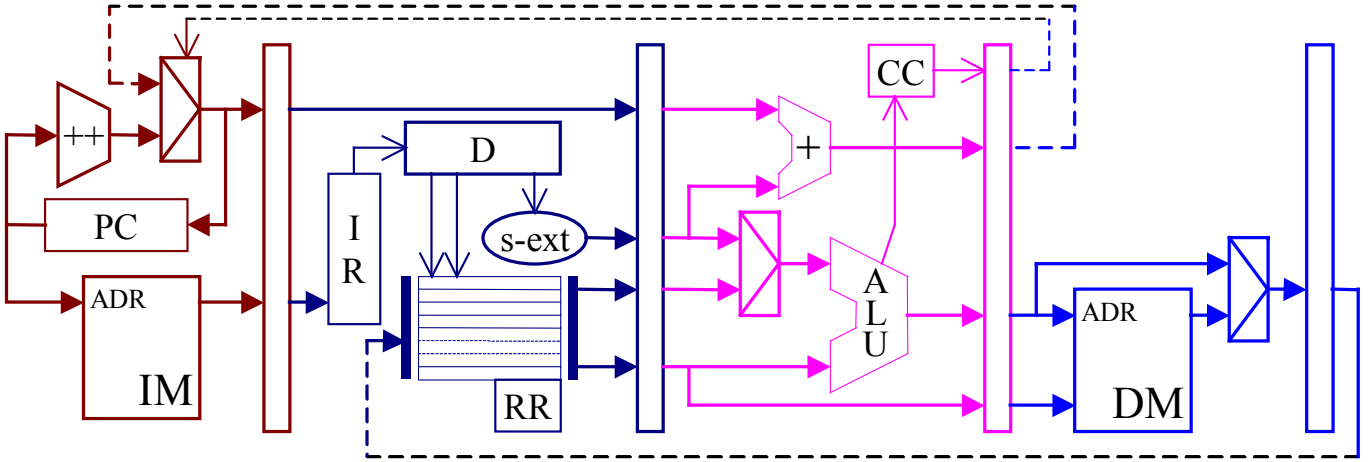
… RR_{i+5} → RR_{i+4} → RR_{i+3} → RR_{i+2} → …



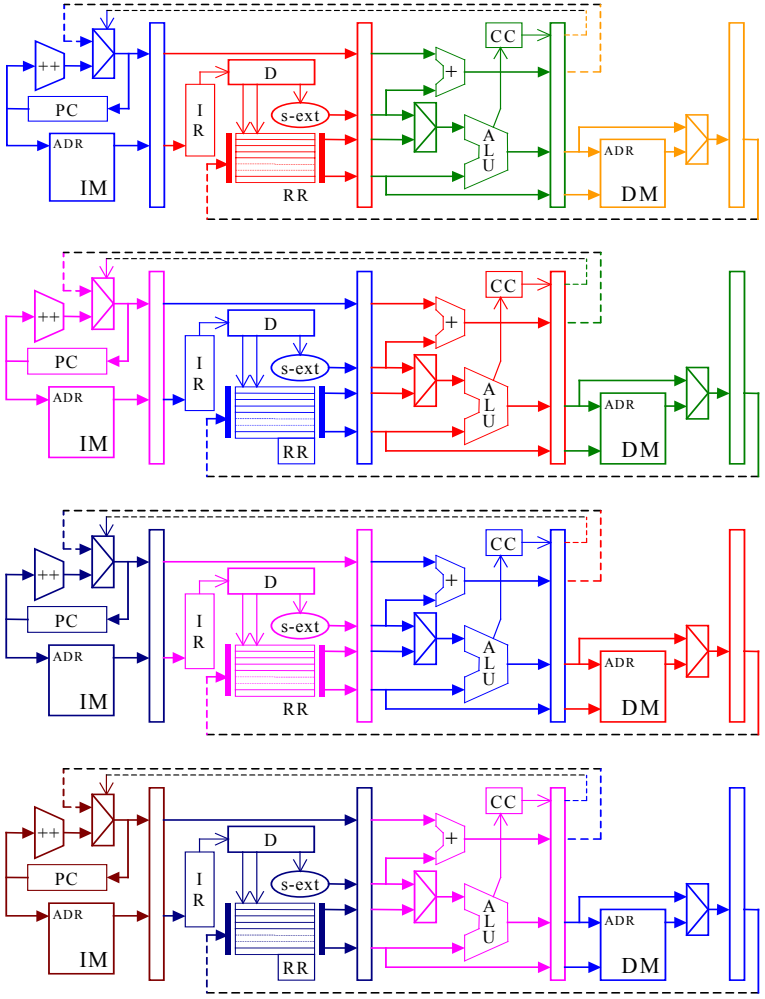
- *F*: RR_{i+5} (IM – bufor pamięci podręcznej kodu, PC – licznik rozkazów)
- *D*: RR_{i+4} (IR – rejestr rozkazów, D – dekoder, RR – plik rejestrów)
- *E*: RR_{i+3} (ALU – jednostka arytmetyczno-logiczna, CC – kod warunków)
- *W*: RR_{i+2} (DM – bufor pamięci podręcznej danych (*load*/*store*))

Organizacja procesora potokowego (rejestrowa)

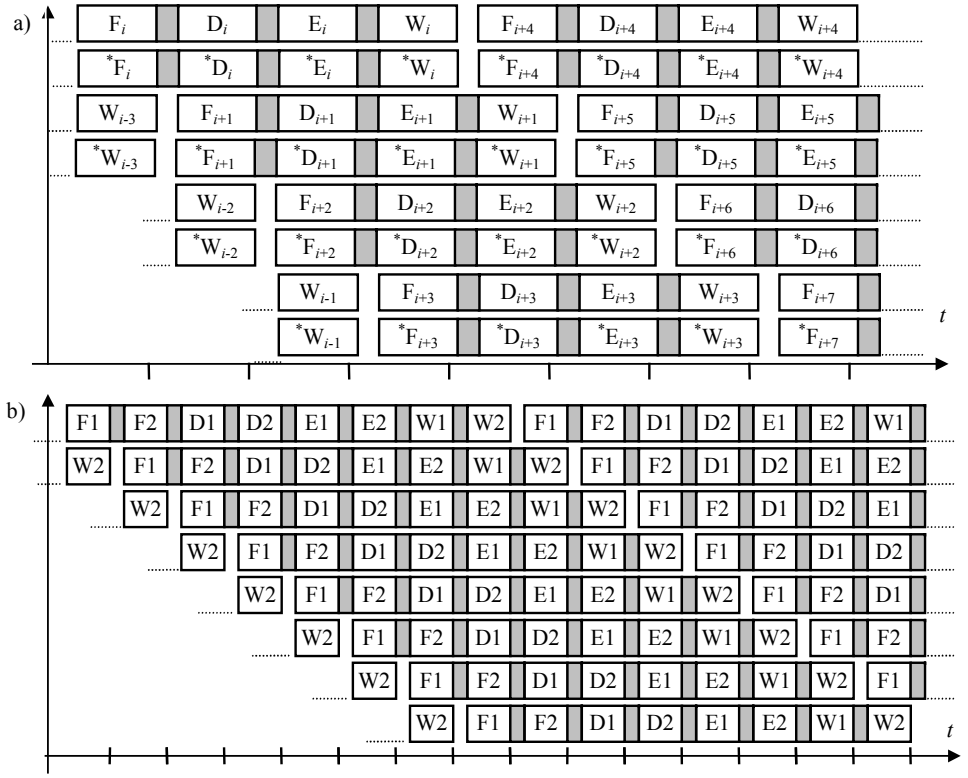
… RR_{i+5} → RR_{i+5} → RR_{i+4} → RR_{i+3} → …



- *F*: RR_{i+6} (IM – bufor pamięci podręcznej kodu, PC – licznik rozkazów)
- *D*: RR_{i+5} (IR – rejestr rozkazów, D – dekoder, RR – plik rejestrów)
- *E*: RR_{i+4} (ALU – jednostka arytmetyczno-logiczna, CC – kod warunków)
- *W*: RR_{i+3} (DM – bufor pamięci podręcznej danych (*load*/*store*))


++
PC
ADR
IM
I
R
D
s-ext
RR
+
A
L
U
CC
ADR
DM

Przetwarzanie superskalarne i superpotokowe



Schematy przetwarzania: a) superskalarne, b) superpotokowe

Inne koncepcje przyśpieszenia przetwarzania programu

Specjalizowane jednostki wykonawcze

- jednostka stałoprzecinkowa IU
- jednostka zmiennoprzecinkowa FPU
- jednostka rozgałęzień BPU
- procesory graficzne

Powielenie i zróżnicowanie jednostek wykonawczych

- przetwarzanie w trybie data flow
- przygotowanie danych i działań – stacje rezerwacyjne
- równoległe wykonywanie działań

Przyśpieszanie pamięci

- lokalne bufory pamięci
- pamięć wieloportowa

Przyśpieszenia przetwarzania wielu programów

współbieżność (ang. *concurrent processing*)

- jednoczesne (etapowo naprzemienne) wykonywanie wielu procesów
- synchronizacja i komunikacja

równoległość (ang. *massive parallelism*)

- równoczesne (w tym samym czasie) wykonywanie niektórych części procesu
- prawo Amdahl'a i prawo Gustaffsona – ograniczenie przyśpieszenia

przetwarzanie wielowątkowe (ang. *multithreading*)

- multiprocesory
- procesory wielordzeniowe

przetwarzanie rozproszone (ang. *distributed processing*)

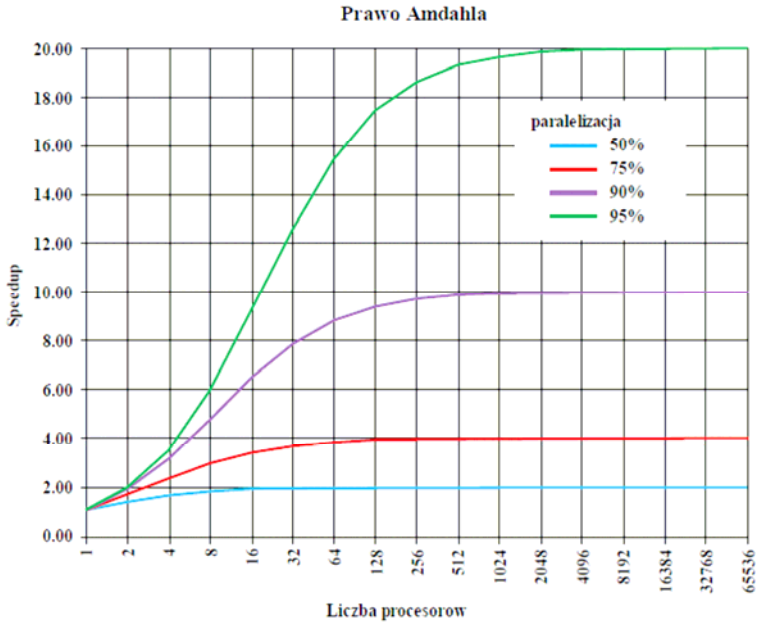
- klastry komputerów

Prawo Amdahla (1967)

Przyśpieszenie przetwarzania (ang. speed-up) jest ograniczone przez część algorytmu, której nie można wykonać równolegle z innymi jego częściami

$$S_{latency} = \frac{T(1-p)+Tp}{T(1-p)+Tpn^{-1}} = \frac{1}{1-p+\frac{p}{n}}$$

1–p – udział części sekwencyjnej, n – liczba procesorów/zasobów zrównoleglonych



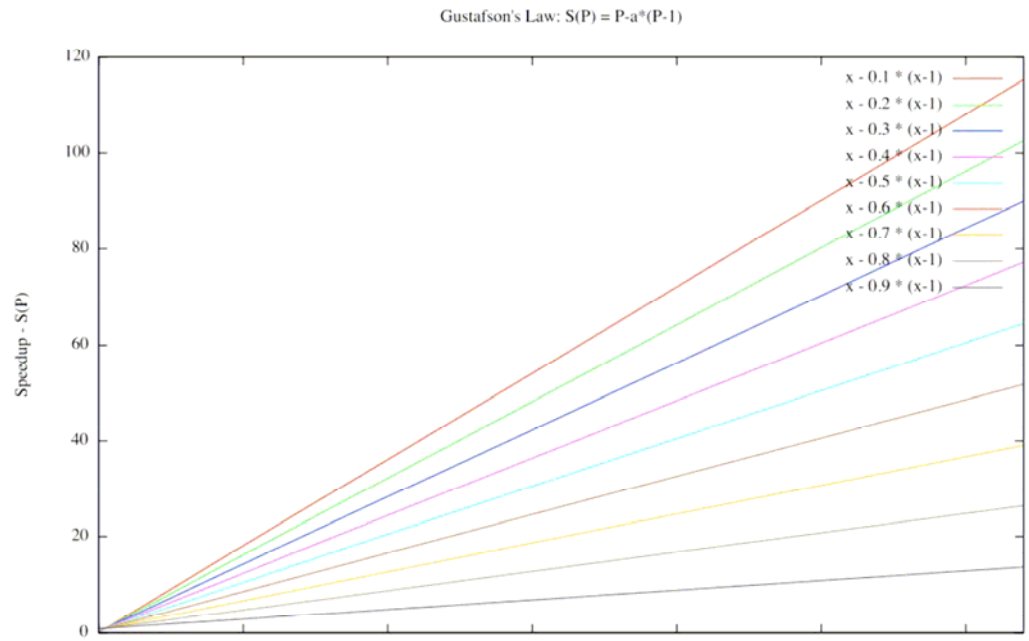
Prawo Gustafsona-Barsisa (1988)

Nakład pracy (ang. *workload*) po ulepszeniu w skali n mamy $W(n)=(1-p)W+npW$

Przyśpieszenie wykonania (*skrócenie zwłoki w wykonaniu*) zadania w ustalonym czasie wykonania (*at fixed execution time*), spodziewane po ulepszeniu zasobów w skali n.

$$S(n) = TW(n)/TW = W(n)/W = 1 - p + pn$$

- n – przyśpieszenie (*speed-up in latency*) wykonania części zadania z użyciem ulepszonych zasobów;
- p – udział w nakładzie czasu pracy całego zadania (przed ulepszeniem) części ulepszanej;



Pzetwarzanie równoległe

Taksonomia Flynn'a (*SISD*/*SIMD*/*MISD*/*MIMD*)
(*S – Single*, *M – Multiple*, *I – Instruction*, *D – Data*)

procesor równoległy ang. *parallel processor*

- organizacja przetwarzania umożliwiająca
  jednoczesne (współbieżne) wykonanie wielu programów

multiprocesor ang. *massively parallel processor, MPP*

- wykonanie w tym samym czasie różnych (równocześnie)
  instrukcji różnych programów
  wraz z organizacją przetwarzania umożliwiającą dystrybucję zadań
  pomiędzy procesorami
- pamięć operacyjna wspólna – możliwa komunikacja przez pamięć
- pamięć operacyjna rozdzielona – możliwa komunikacja przez kanał

procesor wektorowy

- równoczesne wykonywanie prostych działań, lokalna pamięć