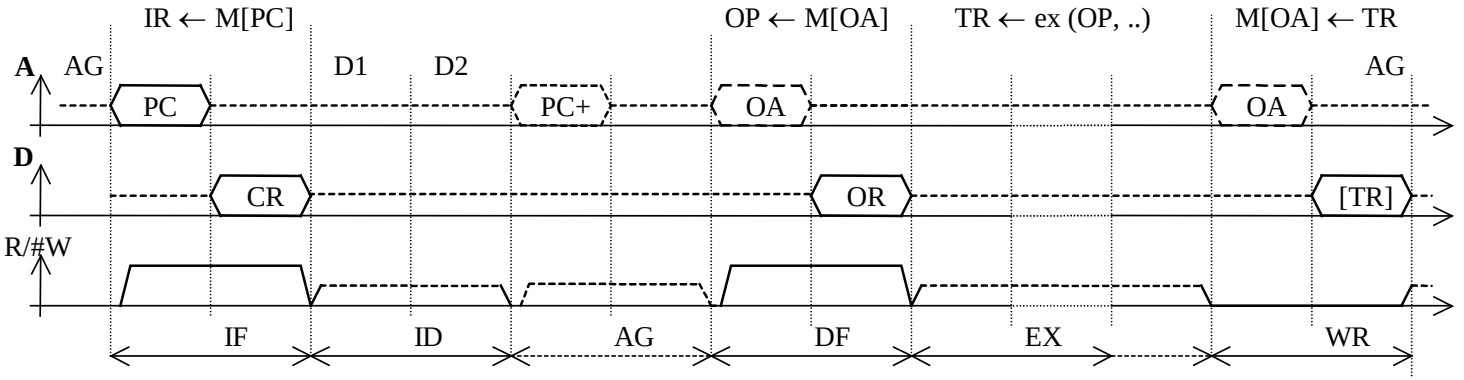
Cykl rozkazowy (CISC)



(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

Koncepcja przetwarzania potokowego

√ etapy przetwarzania wymagają *specjalizowanego sprzętu*

√ *przepływ danych* między układami funkcjonalnymi jest *jednokierunkowy*

- każdy etap jest (może być) wykonywany niezależnie od innych
- możliwe jest *jednoczesne wykonanie różnych etapów* różnych instrukcji
  → konieczna separacja układów wykonujacych różne etapy
  → wynik etapu musi być tymczasowo zapamiętany
- kolejny etap przetwarzania instrukcji może być wykonany dopiero po udostępnieniu (zwolnieniu) potrzebnego układu funkcjonalnego
  → tempo przetwarzania ogranicza czas wykonania *najdłuższego etapu*
  → korzystna jest eliminacja lub dekompozycja czasochłonnych etapów
- *narzut separacji etapów* ogranicza szybkość przetwarzania

! *Powielenie* układów funkcjonalnych *umożliwia* jednoczesne wykonanie tych samych etapów kilku instrukcji lecz tego *nie zapewnia*.

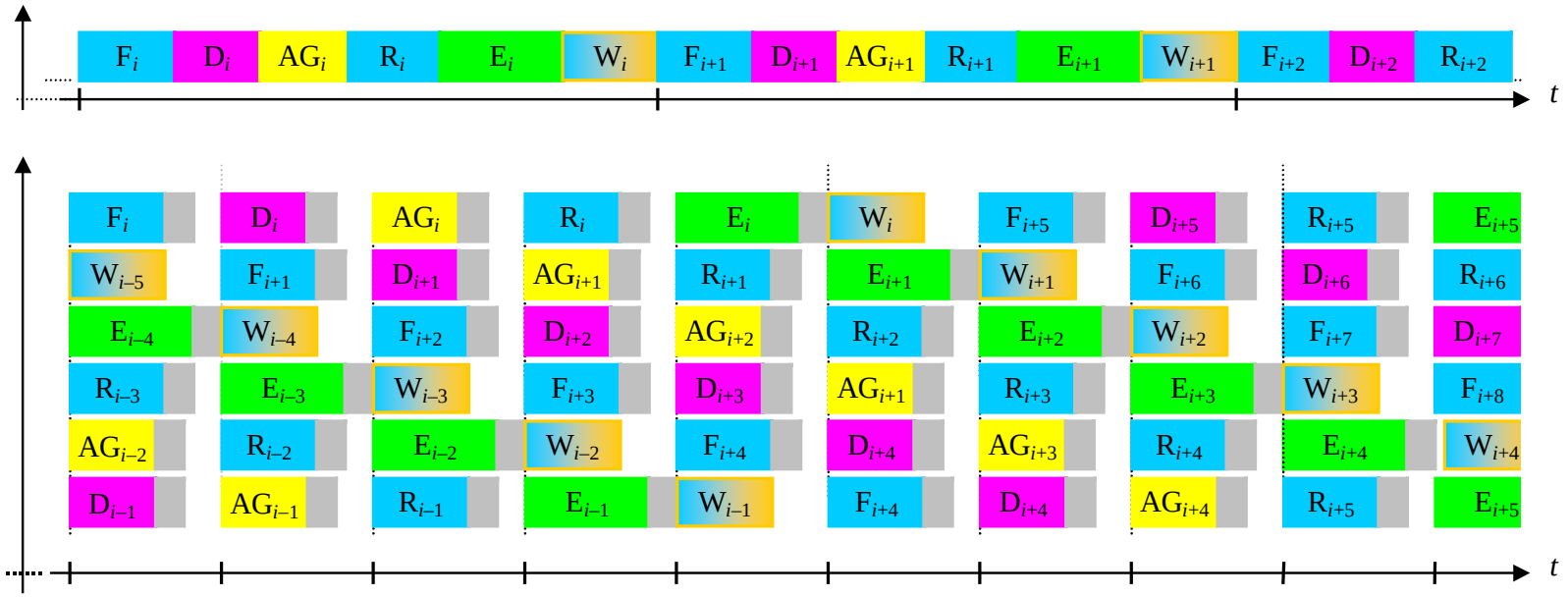
! Wzrost *kwantyzacji cyklu* pociąga za sobą *wzrost narzutu separacji etapów*.

Czas wykonania podstawowych etapów przetwarzania

Najistotniejsze czynniki oddziałujące na *czas wykonania* etapu:

- pobranie kodu rozkazu (IF)
  → rozmiar kodu rozkazu i wpasowanie kodu w pamięci (ang. *alignment*)
  → czas dostępu do pamięci (odczyt słowa lub jego części)
- dekodowanie (ID)
  → architektura listy rozkazów (złożoność i różnorodność działań)
  → struktura kodu rozkazu (regularność i jednorodność)
- wytworzenie adresu operandu w pamięci (AG)
  → tryb adresowania i czas konwersji adresu (system wielozadaniowy)
- pobranie operandu z pamięci (DR)
  → rozmiar operandu
  → czas dostępu do pamięci (odczyt)
- wykonanie (EX)
  → złożoność wykonywanych działań
  → dostępność argumentów
- zapis wyniku do rejestru (PA) [lub pamięci (WR)]
  → [czas dostępu do pamięci (zapis słowa lub jego części)]

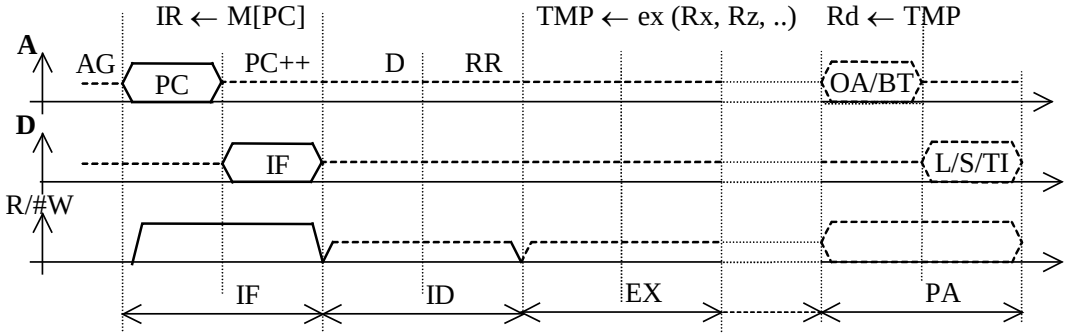
Potok CISC - architektura R/M



Wnioski:

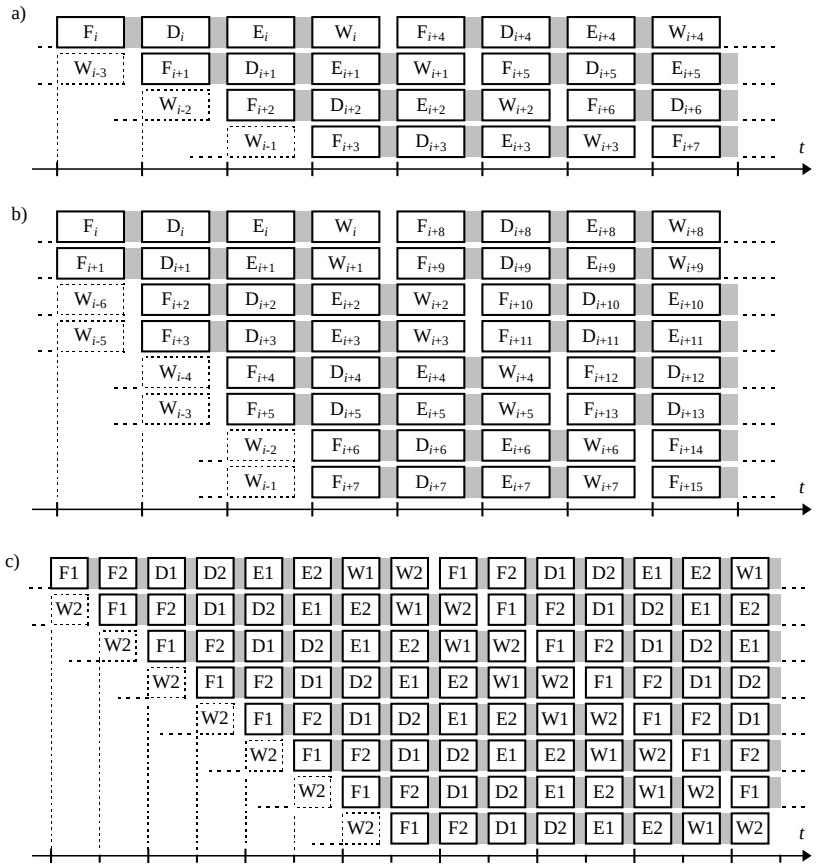
- w każdym cyklu mogą być potrzebne 3 dostępy do pamięci
- użycie rozdzielonego bufora cache na osobne bufory kodu i danych redukuje konflikt dostępu do pamięci,
- umieszczenie danych w rejestrach redukuje liczbę dostępów do pamięci i powoduje skrócenie czasu dostępu do danych

Cykl rozkazowy (RISC)



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

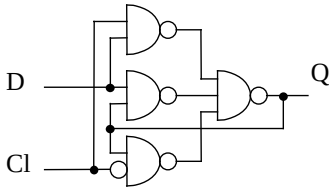
Współbieżne przetwarzanie instrukcji - architektura RISC



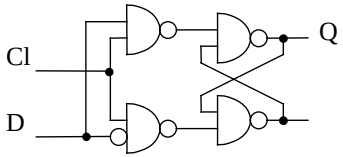
Przetwarzanie: a) potokowe, b) superskalarne i c) superpotokowe

Separacja etapów – zegar i taktowanie zatrzasków

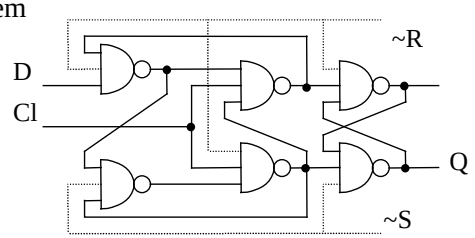
a) zatrzask Earle'a



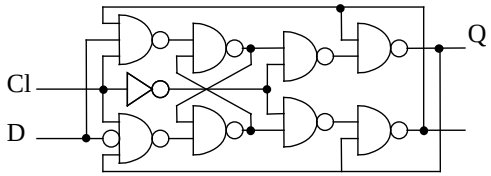
b) zatrzask D

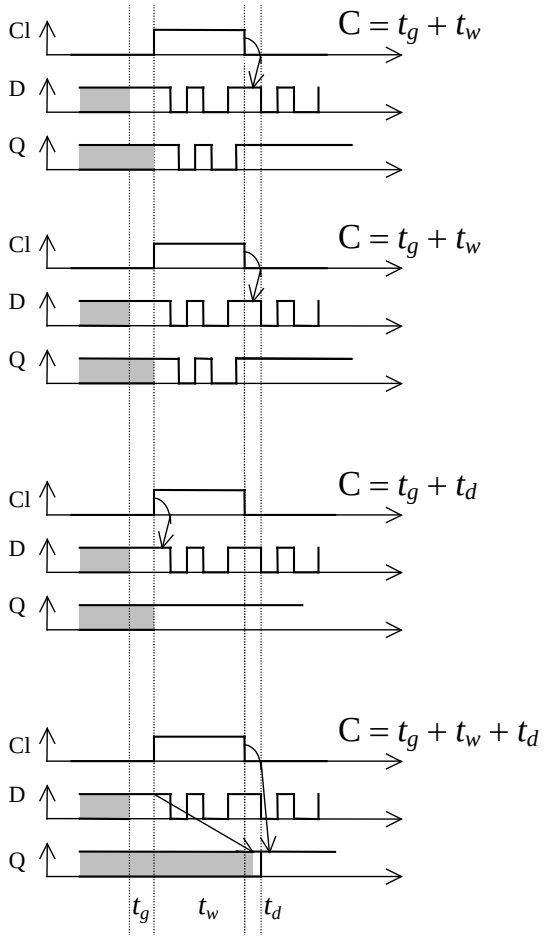


c) wyzwalany zboczem



d) D master-slave





Czas wykonania programu

Czas wykonania programu przez pojedynczy procesor $t_P = T_c \sum_{i=1}^{i=N} I_i \overline{C_i}$ zależy od

- czasu cyklu procesora T_c
- liczby wykonań instrukcji *i*-go typu I_i
- średniej liczby cykli $\overline{C_i}$ potrzebnych do zakończenia instrukcji *i*-go typu od chwili zakończenia instrukcji poprzedniej (wyprowadzenie wyniku)

Czas cyklu procesora zależy od

- konstrukcji układów (organizacji komputera) i technologii wykonania,
- złożoności instrukcji (architektury listy rozkazów)
  - CISC – duża złożoność, RISC – instrukcje proste $\Rightarrow T_{c\text{-}RISC} < T_{c\text{-}CISC}$

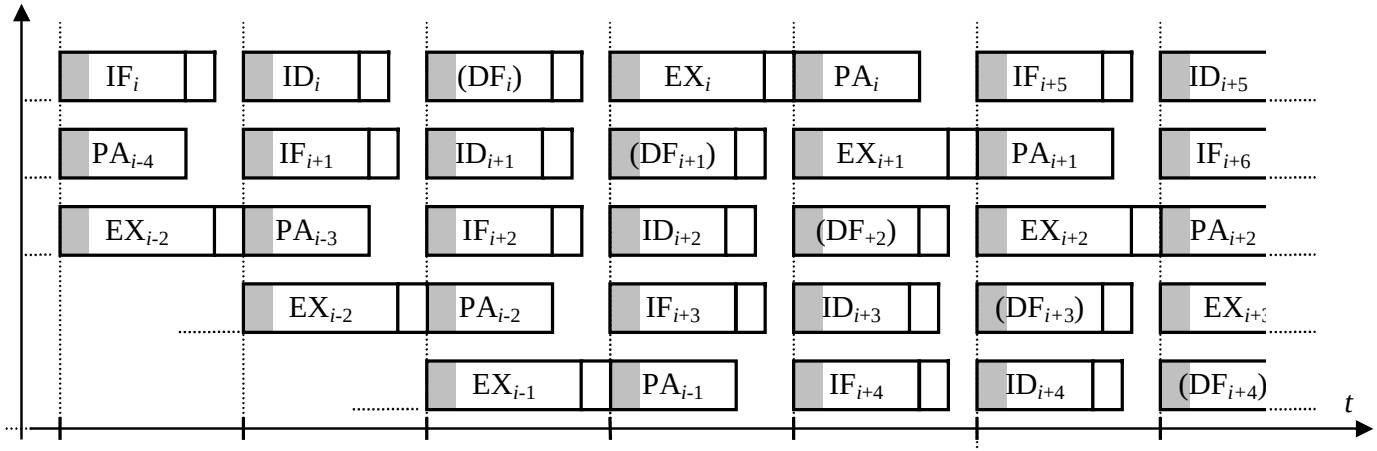
Liczba wykonań instrukcji *i*-go typu zależy od

- użytego algorytmu
- architektury listy rozkazów ($I_{i\text{-}RISC} > I_{i\text{-}CISC}$)

Średnia liczba cykli $\overline{C_i}$ zależy od organizacji przetwarzania

- w potoku zwykłym $\rightarrow 1+\varepsilon$ ($\varepsilon_{\text{RISC}} < \varepsilon_{\text{CISC}}$)
- w sekwencji $\rightarrow k$>>1 (IF,ID,AG,DF,EX+,WR/PA)

Cykl procesora potokowego (RISC/CISC)



długość cyklu – czynniki technologiczne

$$T_{CL} = C + (1+k)(1+\rho(n))\,T\,n^{-1} \quad [\text{ns}]$$

C – narzut zegara (ang. *clock overhead*) – separacja etapów,

k – narzut asynchronizmu zegara (ang. *clock skew*),

$\rho(n)$ – współczynnik wydłużenia cyklu na skutek kwantyzacji instrukcji,

n – głębokość potoku (liczba etapów cyklu rozkazowego),

T – nominalny czas wykonania n etapowej instrukcji prostej.

Przepustowość potoku

Przepustowość (ang. *throughput*) – liczba instrukcji wykonanych w jednostce czasu:

$$G(n) = P(n)/T_{CL}(n) \quad \text{[MIPS]}$$

$$G(n) = p[(1-b)+rd+bn]^{-1}[C+(1+k)(1+\rho(n))Tn^{-1}]^{-1} \text{ [MIPS]}$$

Problem:

$$\max : G(n) = P(n)T_{CL}^{-1}(n),$$

$$\sum_{i=1}^{n} t_i = T, \quad \max_{1\le i\le n} t_i \le \Delta t = (1+\rho(n))Tn^{-1}$$

Hipoteza:

Przepustowość jest największa, gdy minimalne są straty kwantyzacji instrukcji

$$\rho(n) = \min T^{-1}\sum_{i=1}^{n}(\Delta t - t_i), \quad \sum_{i=1}^{n} t_i = T,$$

Rozwiązanie przybliżone ($d\rho(n)/dn \to 0,\ \rho(n)=\rho$)

$$\frac{d}{dn}G(n) = 0 \Rightarrow G(n_0 = \sqrt{(1+k)(1+\rho)TC^{-1}}\sqrt{b^{-1}(1-b+rd)}) = G_{\max}$$

Przepływ danych w potoku

Ścieżki przepływu danych:

- w każdym cyklu
  → zwiększany licznik rozkazów (w etapie IF lub wcześniej)
  → konieczne pobranie następnej instrukcji
  → potrzebne pobranie nowych danych
- transfery danych mogą wystąpić *jeden za drugim* (ang. *back-to-back*),
  → rozdzielenie buforów odczytu (ang. *load*) i zapisu (ang. *store*) danych
- dane aktualne mogą być potrzebne do wykonania etapu kolejnej instrukcji
  → rejestry zatrzaskowe (ang. *latch*)
    - następny stan licznika rozkazów PC
    - wynik działania jednostki ALU

Potencjalne żądania *jednoczesnego dostępu*

- do pamięci – pamięć wielodrożna, rozdzielone bufory pamięci podręcznej
- do pliku rejestrów – dostęp wielodrożny, buforowanie zapisów

Tryb przetwarzania

Wykonanie każdego etapu potoku – kojarzenie ortogonalnych zasobów:

- danych, które stanowią argumenty działań
- układów lub jednostek wykonawczych, które je przetwarzają.

Tryby kojarzenia zasobów

- (ang. *control flow*) – inicjowane przez jednostkę wykonawczą szukanie danych
  → konieczne w etapie pobierania kodu IF (*przetwarzanie sterowane napływem rozkazów*)
- (ang. *data flow*) – inicjowane przez dane angażowanie jednostki wykonawczej
  → możliwe w etapie wykonania EX, wymaga:
    - powielenia układów wykonawczych i kontroli ich dostępności
    - identyfikacji napływających danych
    - kontroli kolejności rozsyłania wyników
  → możliwe w etapie dekodowania ID, wymaga:
    - powielenia dekodera
    - dostępności serii (grup) instrukcji

Przyczyny przestojów (blokady) potoku

blokada potoku (ang. *interlock*) – skutek konfliktów przetwarzania

konflikt danych (ang. *data hazard*)

- niedostępność operandu, będącego wynikiem instrukcji nieukończonej

konflikt strukturalny (*zasobu*) (ang. *resource hazard*)

- kilka instrukcji wymaga jednocześnie dostępu do unikatowego zasobu
- instrukcja wymaga kilku etapów EX (ang. *run-on effect*)

konflikt sterowania (ang. *control hazard*)

- wznowienie potoku po pobraniu instrukcji docelowej (ang. *branch target*)
- opóźnienie użycia warunku

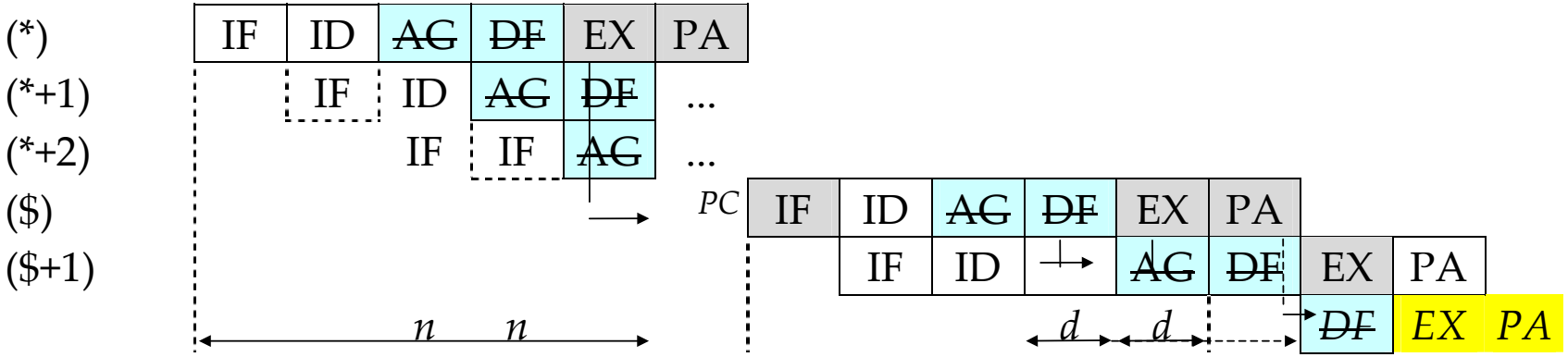
→ rozgałęzienie warunkowe niewykonane (ang. *branch not taken*) – instrukcja następna w sekwencji (ang. *in-line instruction*) jest dekodowana dopiero w cyklu następującym po wytworzeniu warunku

→ rozgałęzienie warunkowe wykonane (ang. *branch taken*) – dekodowanie instrukcji docelowej jest możliwe po jej pobraniu i wytworzeniu warunku

→ rozgałęzienie bezwarunkowe (ang. *branch unconditional*) – dekodowanie instrukcji docelowej jest możliwe dopiero po jej pobraniu

Wydajność potoku (CISC)

- *konflikt sterowania* – możliwa zmiana ścieżki przetwarzania przy rozgałęzieniu
- *konflikt danych* – użycie tej samej danej przez 2 kolejne instrukcje
- *konflikt strukturalny* – niemożność jednoczesnego użycia jednego zasobu



wydajność (ang. *performance*) – przeciętna liczba instrukcji zakończonych w cyklu

$$P = p[(1-b-r)+rd+bn)]^{-1} \quad \text{[IPC]},$$

p – nominalna wydajność potoku [*IPC, instructions per cycle*],
n – głębokość potoku (liczba etapów cyklu rozkazowego),
b – częstość nieprzewidzianych rozgałęzień,
d – liczba cykli opóźnienia z powodu konfliktu danych (d=1 lub 2),
r – częstość konfliktów danych ($r \ll b$).

Analiza konfliktów – potok statyczny (CISC)

(składnia: *mnemonik arg-akumulacyjny, arg-swobodny*)

kolejne etapy (architektura R/M): IF–ID–AG–DF–EX–WR

- dekodowanie (ID), wykonanie (EX) i zapis (WR) w kolejności napływu instrukcji; jeśli instrukcja zakończy się przed poprzednią, to zapis (WR) jest opóźniony
- zasoby są unikatowe i nie można jednocześnie wykonać 2 ścieżek rozgałęzienia
- instrukcje wytwarzają kody warunkowe pod koniec etapu wykonania (EX)
- instrukcja docelowa skoku ($) jest pobierana w cyklu DF rozkazu rozgałęzienia

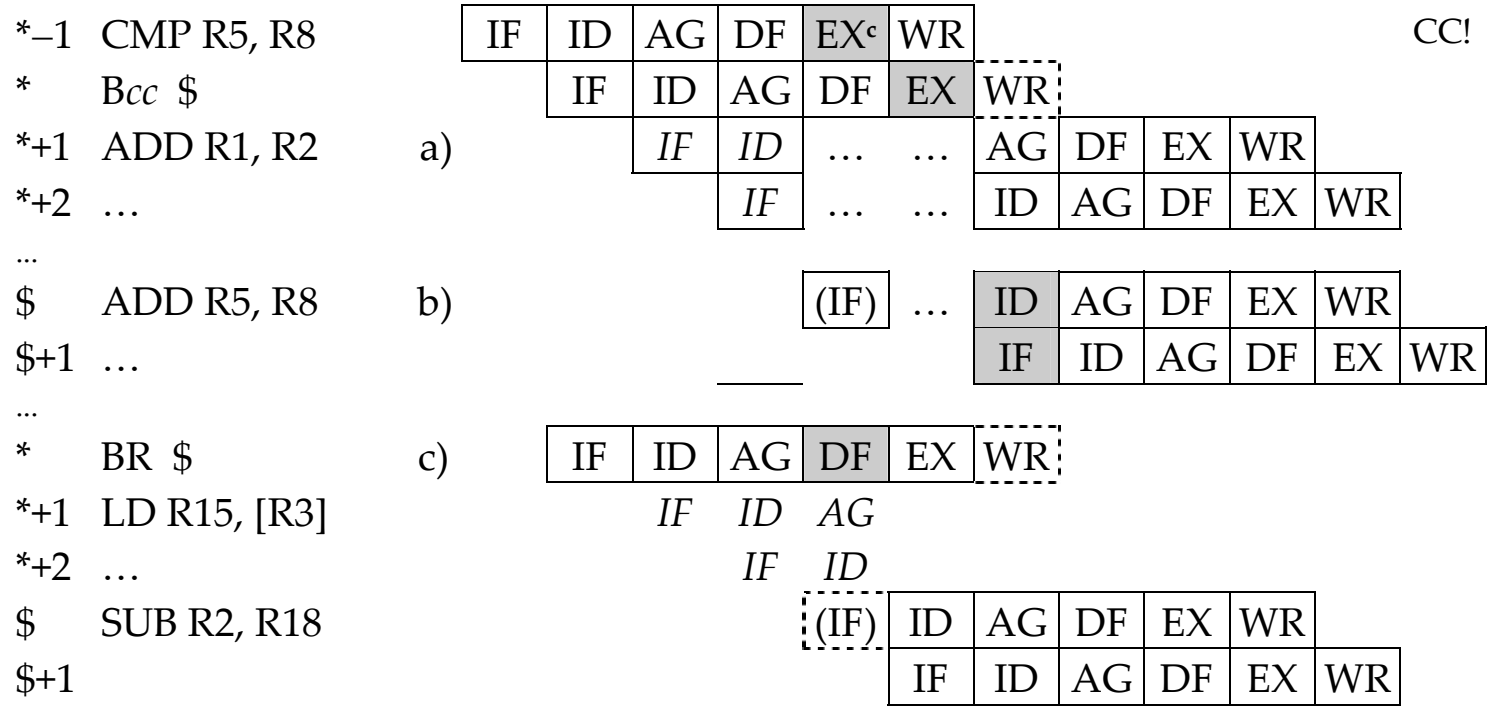
Konflikt danych

*	ADD R1, [R2]		IF	ID	AG	DF	EX	WR					
*+1	ADD R4, [R1]	a)		IF	ID	…	…	…	AG	DF	EX	WR	
*+2	…				IF	…	…	…	ID	AG	DF	EX	WR

$	ADD R1, R3		IF	ID	AG	DF	EX	WR					
$+1	ST [R4], R1	b)		IF	ID	AG	…	…	DS	~~*EX*~~	~~*WR*~~		
$+2	LD R1, [R5]	c)			IF	ID	…	…	AG	…	DF	~~*EX*~~	~~*WR*~~

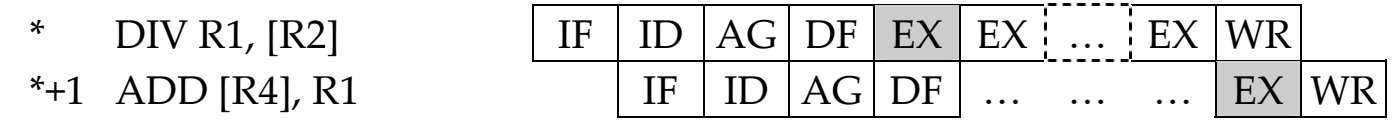
Konflikt: a) adresów $\Delta t_{da} = t_{WR} - t_{AG} = 3$, b) argumentów (RAW) $\Delta t_{dd} = t_{WR} - t_{DF} = 2$, c) kumulacja konfliktów (RAW–WAR) – alternatywny rejestr (R1) eliminuje opóźnienie instrukcji (*+2)

Konflikt sterowania – bez prognozy rozgałęzienia ($t_{CC}=t_{EX}$)



Rozgałęzienia: a) warunkowe niewykonane, $\Delta t_{bnt}=t_{EX}-t_{IF}-1=3$, b) warunkowe wykonane $\Delta t_{bt}=t_{EX}-t_{IF}=4$, c) bezwarunkowe, $\Delta t_{bu}=t_{DF}-t_{IF}=3$

Konflikt strukturalny – wydłużone wykonanie (*run-on hazard*)



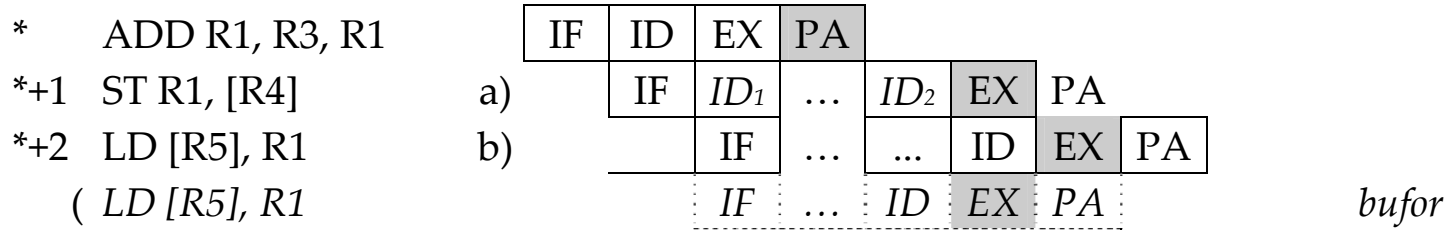
Analiza konfliktów – potok statyczny (RISC)

(składnia: *mnemonik arg*1, *arg*2, *wynik*)

kolejne etapy (architektura L/S): IF–ID–EX–PA

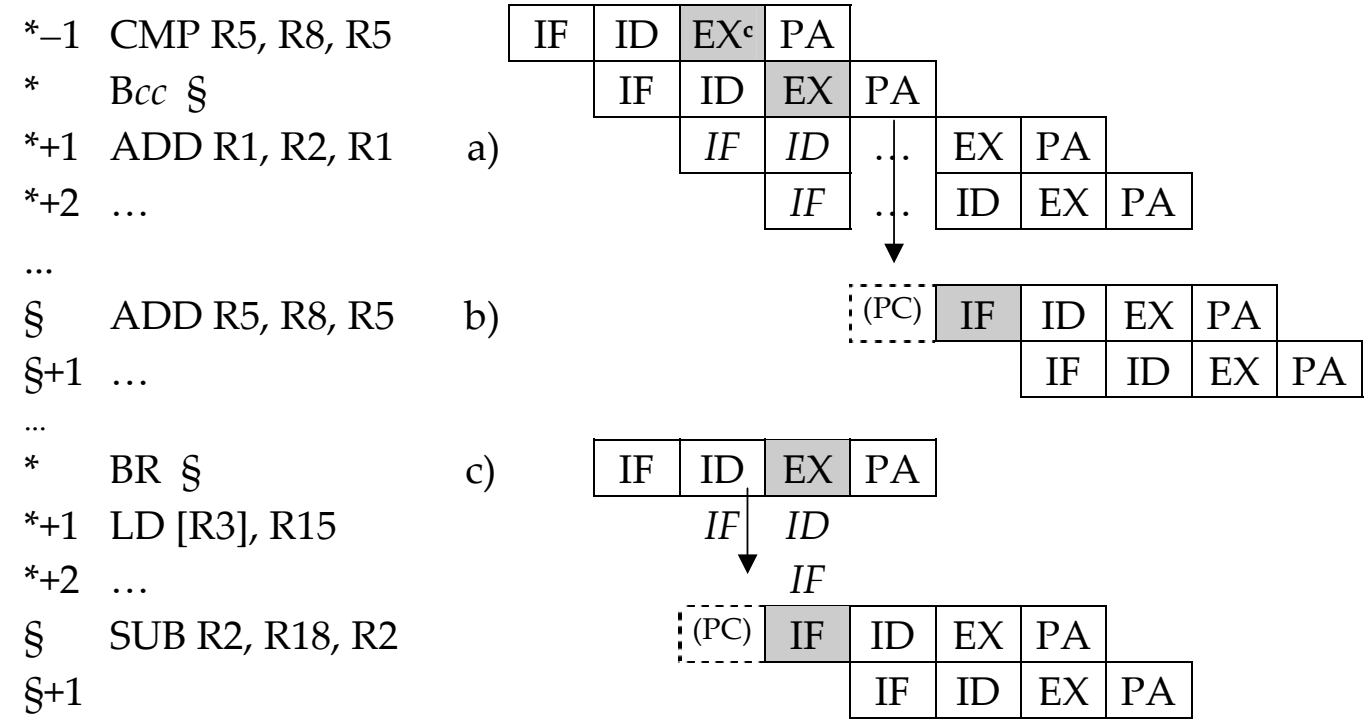
- dekodowanie (ID), wykonanie (EX) i przechowanie (PA) w kolejności napływu instrukcji; jeśli instrukcja zakończy się przed poprzednią, to cykl (PA) jest opóźniony
- zasoby są unikatowe i nie można jednocześnie wykonać 2 ścieżek rozgałęzienia
- instrukcje wytwarzają kody warunkowe CC pod koniec etapu wykonania (EX)
- instrukcja docelowa skoku (§) jest pobierana w cyklu PA rozkazu rozgałęzienia
- dostęp do pamięci (load / store) realizowany w etapie PA

Konflikt danych



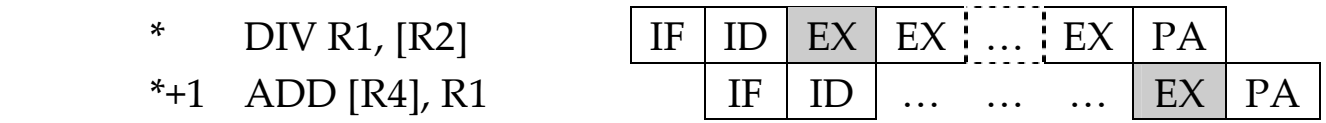
Konflikt argumentów: a) odczyt po zapisie (RAW) $\Delta t_{dd} = t_{PA} - t_{ID} = 2$,
b) kumulacja konfliktów (RAW–WAR) – bufor wyniku eliminuje opóźnienie instrukcji (*+2)

Konflikt sterowania – bez prognozy rozgałęzienia (założenia optymistyczne)



Rozgałęzienia: a) warunkowe niewykonane, $\Delta t_{bnt} = t_{EX} - t_{ID} = 1$, b) warunkowe wykonane $\Delta t_{bt} = t_{EX} - t_{IF} = 2$,
c) bezwarunkowe, $\Delta t_{bu} = t_{ID} - t_{IF} = 1$

Konflikt strukturalny – wydłużone wykonanie (*run-on*)



Straty na skutek konfliktu sterowania

Konflikt sterowania (t_{IF} – koniec IF, t_{DF} – koniec DF, t_{CC} – ustalenie warunku):

- rozgałęzienie bezwarunkowe

$$\Delta t_{bu} = t_{ID} - t_{IF}$$

- rozgałęzienie warunkowe wykonane ($t_{ID} \le t_{CC} \le t_{EX}$)

$$\Delta t_{bt} = t_{EX} - t_{IF}$$

- rozgałęzienie warunkowe niewykonane ($t_{ID} \le t_{CC} \le t_{EX}$)

$$\Delta t_{bnt} = t_{EX} - t_{ID}$$

- przekazanie sterowania do nieznanego adresu (w rejestrze lub na stosie)

$$\Delta t_{u} = t_{EX} - t_{IF}$$

Straty na skutek konfliktu danych

Konflikt danych (CISC) (t_{AG} – koniec AG, t_{DF} – koniec DF, t_{WR} – koniec PA):

- zależność adresowa – obliczenie adresu wymaga operandu, którego wartość wytwarza poprzednia instrukcja w etapie WR, co powoduje opóźnienie

$$\Delta t_{da} = t_{PA} - t_{AG}$$

- zależność danych – wykonanie instrukcji wymaga operandu wytwarzanego przez poprzednią instrukcję, co powoduje opóźnienie

$$\Delta t_{dd} = t_{PA} - t_{DF}$$

Konflikt danych (RISC) (t_{DF} – koniec DF, t_{WR} – koniec PA):

- zależność adresowa – obliczenie adresu wymaga operandu, którego wartość wytwarza poprzednia instrukcja w etapie PA, co powoduje opóźnienie

$$\Delta t_{da} = t_{PA} - t_{ID}$$

- zależność danych – wykonanie instrukcji wymaga operandu wytwarzanego przez poprzednią instrukcję, co powoduje opóźnienie

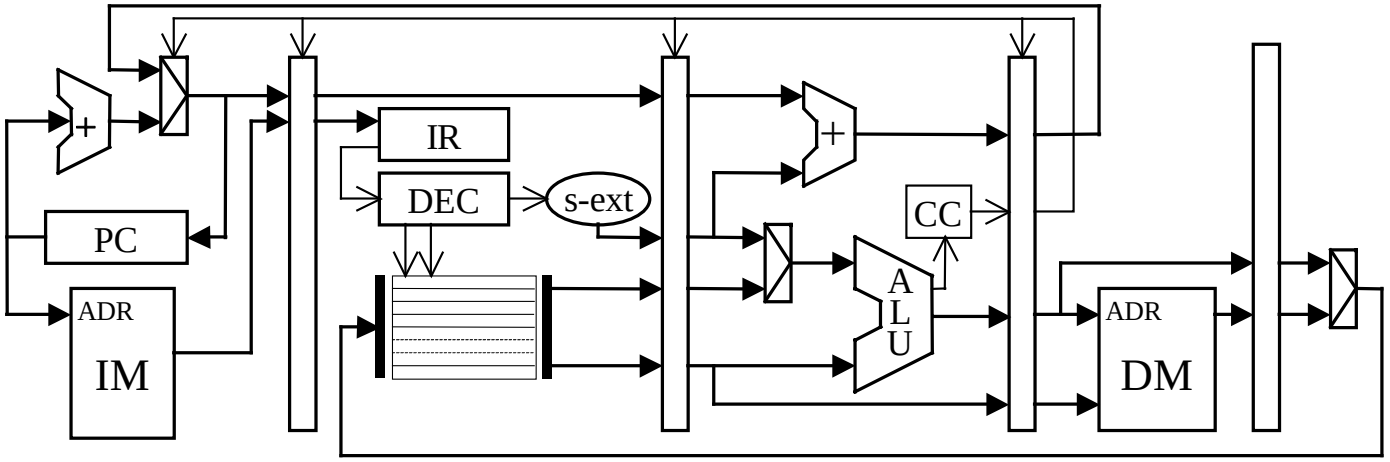
$$\Delta t_{dd} = t_{PA} - t_{ID}$$

Organizacja procesora potokowego

- potok płytki (ang. *shallow pipeline*), jeden etap = kilka działań atomowych,
- potok głęboki (ang. *deep pipeline*), jeden etap = jedno działanie atomowe

potok statyczny (ang. *static pipeline*) – schemat podstawowy

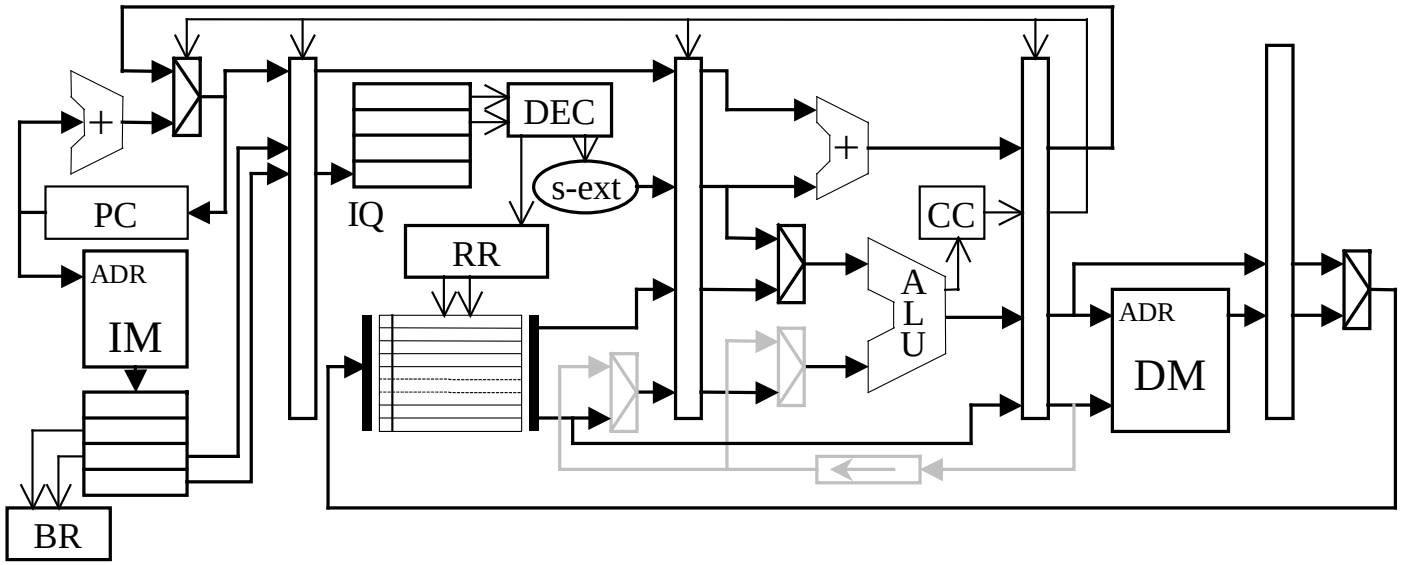
- nie można przeskoczyć żadnego etapu
- konieczne nieaktywne oczekiwanie na dokończenie etapu poprzedniego
- każdy konflikt powoduje przestój (ang. *stall*)



Przepływ danych w potoku statycznym

Przepływ danych w prostym potoku dynamicznym (I rodzaju)

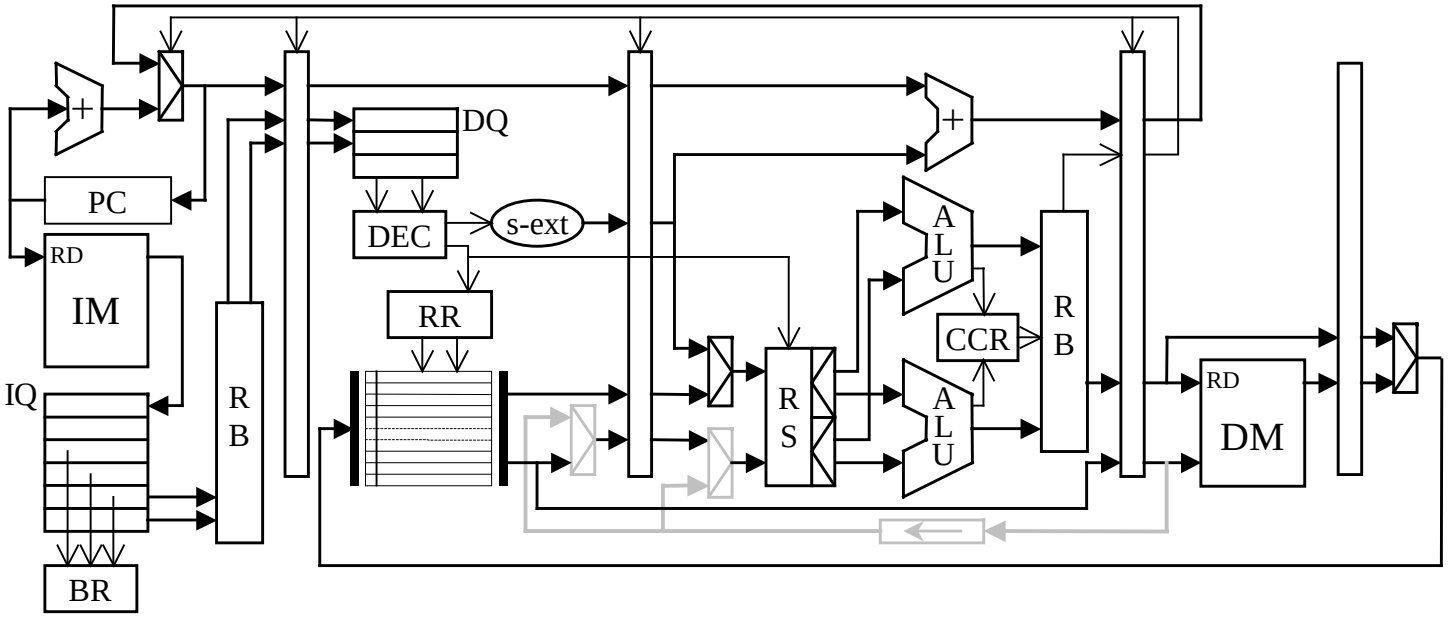
- ominięcie (ang. *bypassing*) etapów nie związanych bezpośrednio z wykonaniem współbieżne z zapisem bezpośrednie przekazanie wyniku (ang. *forwarding*)
- kolejność dekodowania i zapisów – zgodna z kolejnością pobierania
- wytworzenie i translacja adresu instrukcji *load* / *store* muszą być zakończone zanim którakolwiek z instrukcji następnych wykona zapis.



Skróty na ścieżkach przepływu danych w potoku dynamicznym I rodzaju

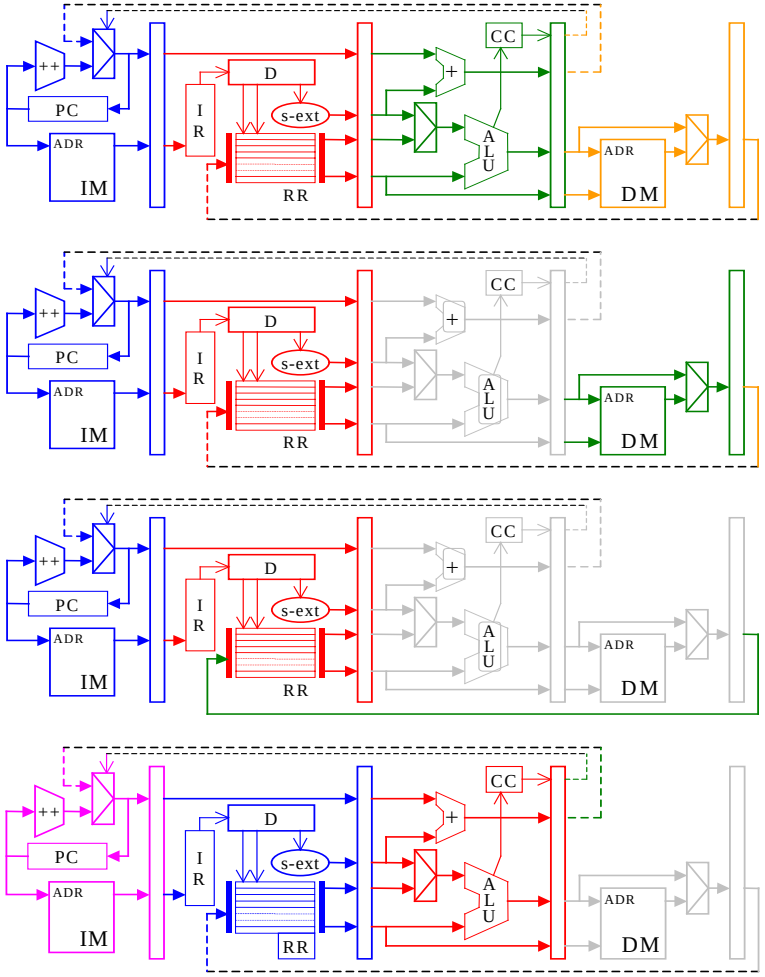
Przepływ danych w potoku dynamicznym II i III rodzaju

- potok II rodzaju – zachowana tylko kolejność dekodowania
- potok III rodzaju – dekodowanie w takiej kolejności, aby instrukcja dekodowana była niezależna od instrukcji, których wykonanie trwa

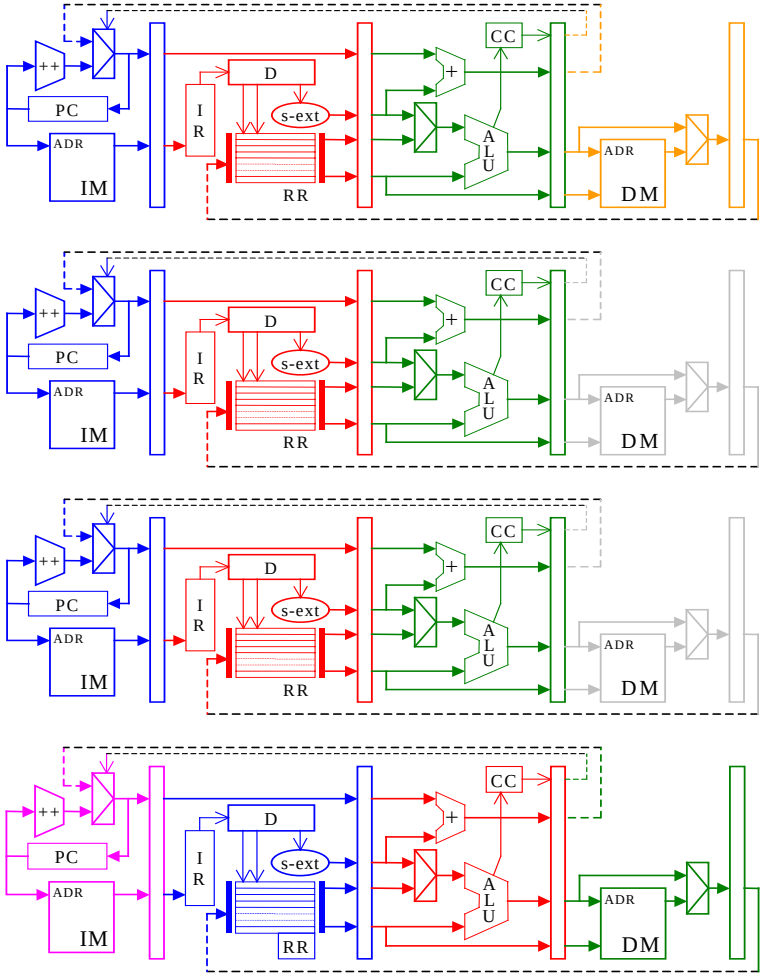


Przepływ danych w potoku dynamicznym (RR – przemianowanie rejestrów, RS – stacja rezerwacyjna, RB – bufory przestawień, IQ – kolejka rozkazów)

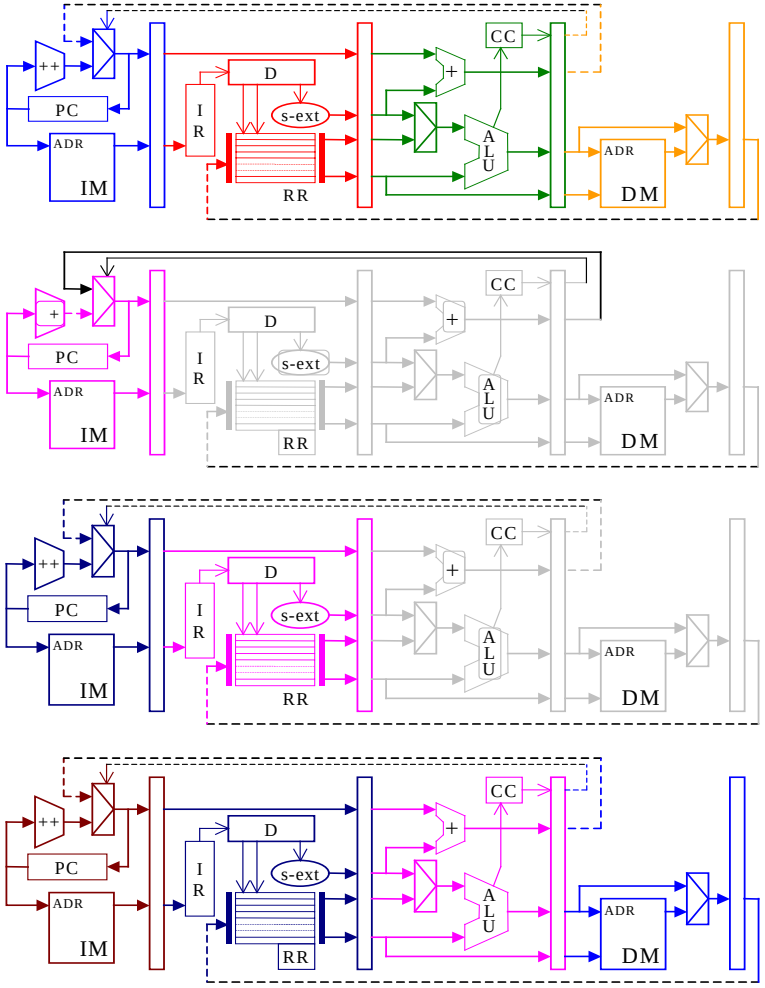
Konflikt danych (potok statyczny)



Konflikt dostępu (potok statyczny)



Konflikt sterowania (potok statyczny)



Konflikt sterowania

spekulacyjne wykonanie (ang. *speculative execution*) na podstawie *prognozy*

→ uprzedzające przełączenie strumienia rozkazów zanim rozkaz zostanie zdekodowany → bufor kolejki rozkazów

prognoza rozgałęzienia – cel: minimalizacja czasu przestoju potoku

- statyczna – oparta na częstości wykonania rozgałęzienia danego typu,
- dynamiczna – oparta na historii przetwarzania

warunki skutecznego wpływu na przepustowość potoku,

- wcześniejsze rozpoznanie rozkazów rozgałęzień
- użycie buforów rozkazu docelowego skoku i kolejnych rozkazów

metody minimalizacji blokad wskutek rozgałęzień warunkowych

programowe

- przyśpieszenie wytworzenia warunku przez zmianę sekwencji rozkazów

sprzętowe

- zapamiętanie warunku w specjalnym rejestrze
- skok opóźniony – instrukcja po rozkazie skoku jest zawsze wykonywana

Statyczna prognoza rozgałęzień

Rozpoznanie typu rozgałęzienia

Zasięg przemieszczenia (adres względny) zapisany w kodzie w systemie U2

- skok w przód – kod adresu 0x…xx (przemieszczenie dodatnie)
- skok w tył – kod adresu 1x…xx (przemieszczenie ujemne)

Ocena średniej straty czasu zależnie od decyzji (*wykonaj* – *pomiń*):

p – prawdopodobieństwo wykonania skoku

N – przestój wskutek błędnej prognozy ($N > 4$)

t – przestój wskutek potrzeby przeładowania kolejki rozkazów ($t > 2$)

$$\mathrm{E}(wykonaj) = p\ t + (1–p)\ N$$

$$\mathrm{E}(pomiń) = (1–p) + p\ N$$

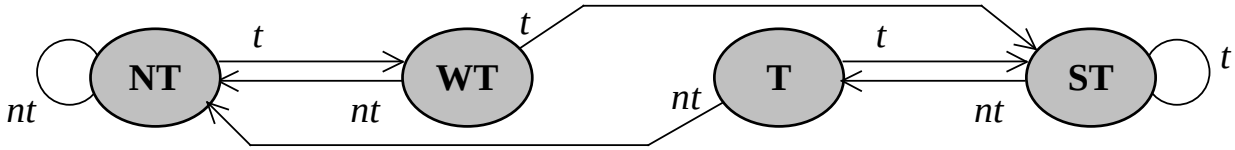
Przesłanka statystyczna – *reguła 90/50* (statystyczna)

- około 90% skoków warunkowych w tył jest wykonywanych
  → **decyzja**: wykonaj rozgałęzienie (skok)
- równe szanse wykonania i niewykonania skoku do przodu
  → **decyzja**: nie wykonuj rozgałęzienia (wykonaj kolejną instrukcję)

Dynamiczna prognoza rozgałęzień

Prognoza dynamiczna – ocena szansy wykonania na podstawie historii

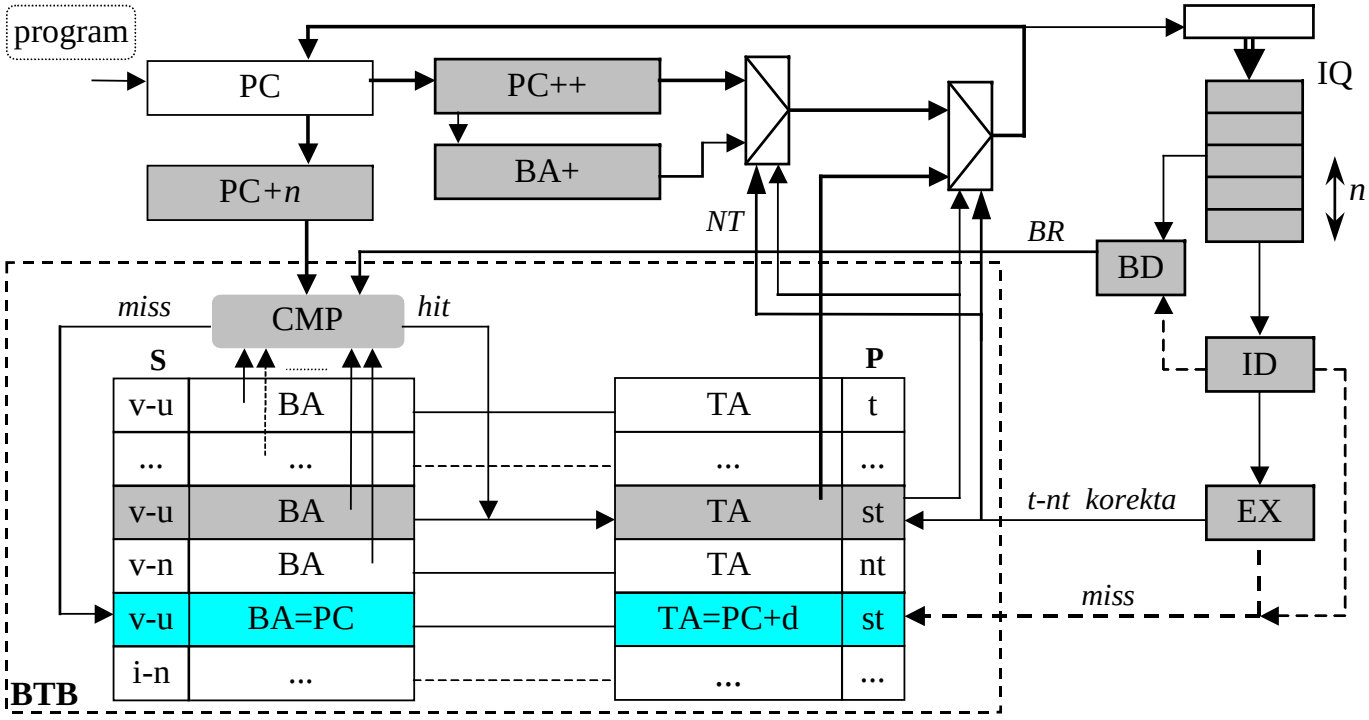
- czterostanowy układ prognozy
  - prognoza trafna podnosi a nietrafna obniża poziom szansy wykonania rozgałęzienia w kolejnym rozpoznaniu adresu docelowego skoku



ST (*strongly taken*), T (*taken*), WT (*weakly taken*), NT (*not taken*)
t – skok wykonany, *nt* – skok niewykonany

- wzorce sekwencji wykonań
  - jednostopniowe – pamięć poprzedniego kroku
  - wielostopniowe – pamięć kilku poprzednich kroków, liczba możliwych schematów 2^{k-1}, nie wszystkie używane, najczęściej pamięć 2 kroków

Bufor prognozy rozgałęzień



Prognoza dynamiczna: BA, BA+–adres instrukcji rozgałęzienia i instrukcji następnej, S–status linii (v/i–ważny /nieważny, u/n–bieżący/dawny), TA–adres docelowy, PC–licznik rozkazów, PC++–inkrementer PC, BD–detektor rozgałęzienia, IQ–kolejka rozkazów, ID–dekoder

Prognoza rozgałęzień łączna

W buforze odnotowane są tylko skoki wykonane

Prognoza dynamiczna – model jednostopniowy

- ST, T – **decyzja**: wykonaj rozgałęzienie, → weryfikacja
- NT, WT – **decyzja**: nie wykonuj rozgałęzienia, → weryfikacja

Prognoza dynamiczna ma priorytet – prognoza statyczna jest podejmowana wtedy, gdy w buforze prognozy rozgałęzienie nie jest odnotowane

Przy braku prognozy statycznej rozgałęzienie napotkane po raz pierwszy (nie odnotowane w buforze BTB) *nie jest wykonane*

W razie przepełnienia bufora BTB:

wymiana linii – strategia FIFO lub LRU

Konflikt danych

dynamiczny, zależny od programu, wymaga przewidywania zagrożeń.

Sposoby łagodzenia lub eliminacji zależą od typu zależności.

- RAR (ang. *read after read*) –odczyt po odczycie
  → występuje w potoku superskalarnym – dwuportowy odczyt rejestrów
- RAW (ang. *read after write*) – odczyt po zapisie
  → skrót (ang. *bypass*) na ścieżkach przepływu danych (ang. *data paths*)
- WAR (ang. *write after read*) – zapis po odczycie
  → przemianowanie rejestrów (ang. *register rename, aliasing*)
- WAW (ang. *write after write*) – zapis po zapisie.
  → przemianowanie rejestrów (ang. *register rename, aliasing*)

zapis jest niszczący → kolejnemu zapisowi do rejestru jest równoważny
zapis do dowolnego innego rejestru,
którego zawartość jest w danej chwili nieistotna.

Przemianowanie rejestrów

Instrukcja	Przemianowanie	Działanie	Status (po zakończeniu)							
			R0	R1	R2	R3	R4	R5	R6	R7
mov bx, ax	ax→R0, bx→R1	R1←R0	1	1	–	–	–	–	–	–
add ax, cx	cx→R2, ax→R3	R3←R2+R0	0	–	1	1	–	–	–	–
mov cx, [bx]	cx→R4, bx…	R4←[R1]	–	1	0	–	1	–	–	–
xor cx, ax	cx→R5, ax…	R5←R4⊕R3	–	–	–	1	0	1	–	–
sub ax, dx	ax→R7, dx→R6	R7←R3–R6	–	–	–	0	–	–	1	1
inc bx	bx→R0,	R0←R1+1	1	0	–	–	–	–	–	–
mov dx, [bx]	dx→R2, bx…	R2←[R0]	1	–	1	–	–	–	0	–

Przemianowanie rejestrów (jednocześnie mogą być wykonane 3 instrukcje „–”– stan zależny od zakończenia wcześniejszych instrukcji, 1 – rejestr zajęty, 0 – rejestr zwolniony, „…” – utrzymanie wcześniejszego przemianowania)

Przemianowanie rejestrów (?)

Instrukcja	Przemianowanie		Działanie	Status (po zakończeniu)							
				ax		bx		cx		dx	
				R0	R1	R2	R3	R4	R5	R6	R7
mov bx, ax	ax (R0)	bx (R2)	R2 ← R0	1	0	1	0	–	–	–	–
add ax, cx	ax (R0 → R1)	cx (R4)	R1 ← R0 + R4	0	1	1	0	1	0	–	–
mov cx, [bx]	cx (R4)	bx (R2)	R4 ← [R2]	0	1	1	0	1	0	–	–
xor cx, ax	cx (R4 → R5)	ax (R1)	R5 ← R4 ⊕ R1	0	1	1	0	0	1	–	–
sub ax, dx	ax (R1 → R0)	dx (R6)	R0 ← R1 – R6	1	0	1	0	0	1	1	–
inc bx	bx (R2 → R2)		R2 ← R2++	1	0	1	0	0	1	1	–
mov dx, [bx]	dx (R7)	bx (R2)	R7 ← [R2]	1	0	1	0	0	1	0	1

Przemianowanie rejestrów (jednocześnie mogą być wykonane 3 instrukcje „–” – stan zależny od zakończenia wcześniejszych instrukcji, 1 – rejestr zajęty, 0 – rejestr zwolniony, „…” – utrzymanie wcześniejszego przemianowania)

Konflikt strukturalny i jego rozwiązania

- konflikt dostępu do pamięci podręcznej, spowodowany żądaniem dostępu do 2 linii jednocześnie, pochodzącym z jednej lub z różnych faz potoku
→ buforowanie linii *cache* lub rozdzielenie na osobne pamięci kodu i danych

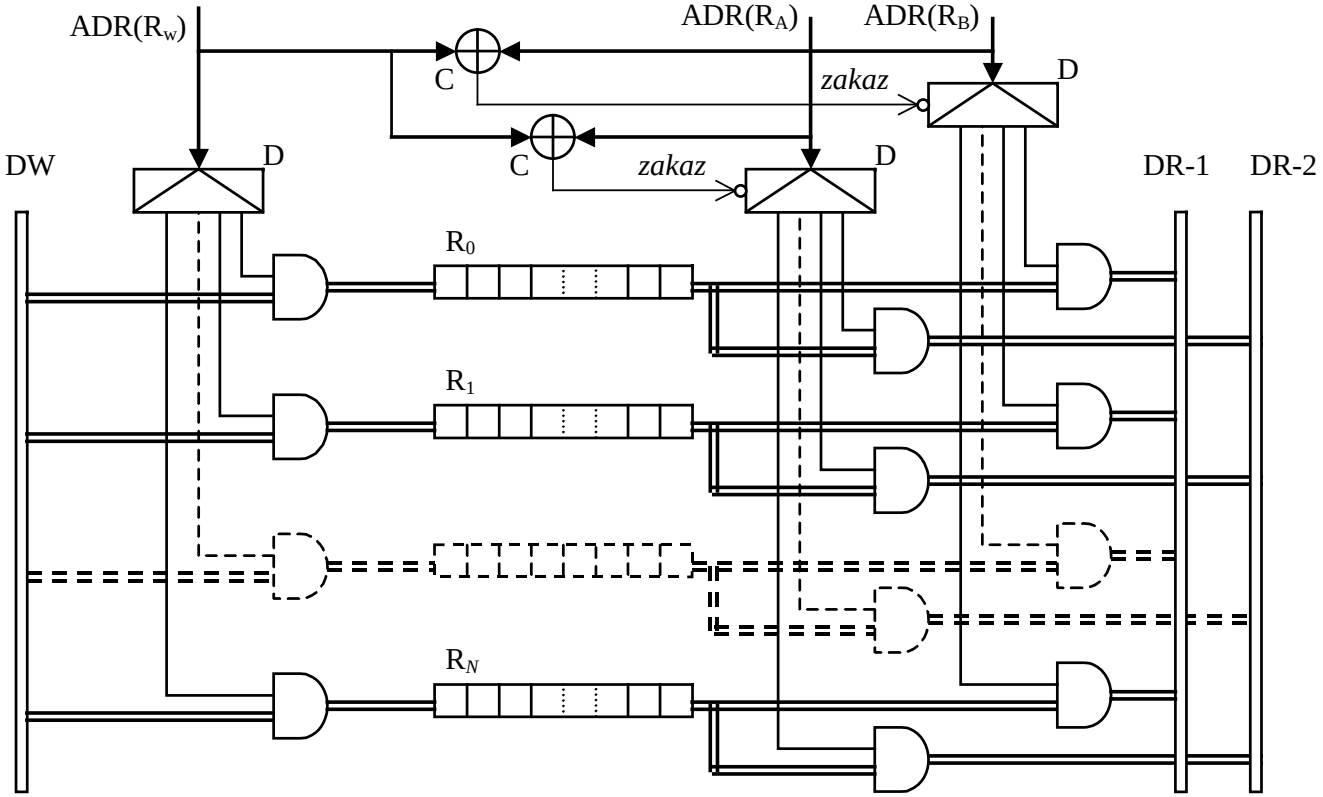
- konflikt wydłużonego dostępu do pamięci wskutek chybienia
→ antycypowanie pobieranie linii, użycie bufora zapisu

- konflikt dostępu do pliku rejestrowego, spowodowany jednoczesnymi żądaniami użycia rejestrów pliku
→ równoległe magistrale odczytu i osobna magistrala zapisu z arbitrażem

- konflikt przedłużonego wykonania instrukcji złożonej (*run-on effect*).
→ niekolejne wykonanie instrukcji, optymalizacja układów

- ma charakter statyczny i może być rozwiązany przez rozbudowę układów

Wielomagistralowy dostęp do pliku rejestrowego



ADR(RX) – indeks rejestru RX, D – dekodery, DR# – magistrale odczytu,
C – komparatory, DW – magistrala zapisu, (priorytet zapisu)

Niekolejne wykonanie rozkazów

Wykonanie instrukcji w kolejności zapisanej w programie (*in-order execution*):

- instrukcja czasochłonna (*long-running*) powoduje wstrzymanie (*stall*) potoku
- konflikt danych RAW powoduje wstrzymanie (*stall*) potoku

Zmiana kolejności wykonania rozkazów:

- w rytmie napływu rozkazów (*control flow*)
  - *stacja rezerwacyjna* (*reservation station*) – ogólny bufor instrukcji przekazanych do wykonania i ich statusu
  - status każdej instrukcji jest na bieżąco aktualizowany
  - sprawdzanie dostępności argumentów
  - przekazywanie instrukcji do dostępnej jednostki wykonawczej
- w rytmie gotowości danych (*data flow*).
  - każda jednostka wykonawcza ma własny bufor instrukcji i danych
  - daną niedostępną wskazuje jej adres (nr jednostki ją wytwarzającej)
  - dana chwilowo niedostępna może być pobrana bezpośrednio z wyjścia jednostki wykonawczej przez wewnętrzną magistralę
  - może istnieć wiele kopii tej samej danej
  - status każdej instrukcji jest na bieżąco aktualizowany

Zmiana kolejności wykonania rozkazów

niekolejne wykonanie instrukcji
→ wymaga analizy instrukcji w etapie dekodowania lub wykonania

skuteczne, gdy
→ procesor używa kilku niezależnych jednostek wykonawczych
→ jednostki wykonawcze działają potokowo

dekoder → układ *wydawania instrukcji* (*instruction issue*) z kolejki rozkazów

- rozpoznawanie rozkazów jednocześnie w kilku lokacjach kolejki
  → współbieżny napływ rozkazów do jednostek
- analiza szans wykonania instrukcji (*scoreboarding*)
  → rozsyłanie instrukcji do jednostek wykonawczych
  → ewentualne przemianowanie rejestrów
- instrukcja przesłana do bufora *stacji rezerwacyjnej* (*reservation station*)
  → czeka na wykonanie, dopóki nie są dostępne jej argumenty
  → konflikty danych rozstrzygane we wstępnej fazie wykonania

Podstawowy schemat wykonania rozkazów w potoku

potok statyczny:

- wykonywanie instrukcji w kolejności zapisanej w programie
  – IF – pobranie (kodu) instrukcji zgodnie z kolejnością w algorytmie
  – ID – dekodowanie kolejnych instrukcji (*in-order instruction issue*) w celu identyfikacji argumentów i wytworzenia sterowania dla jednostki wykonawczej,
  – (DR) – dostarczenie argumentów, poprzedzone obliczeniem adresu (AG)
  – EX – wytworzenie wyniku działania kolejnej instrukcji
  – PA – wyprowadzenie wyniku z potoku (zapis w miejscu docelowym)
- instrukcja czasochłonna (*long-running*), wymusza zatrzymuje (*stall*) potoku
- konflikt danych wstrzymuje wysłanie kolejnej instrukcji do fazy EX
- rozgałęzienie wymusza wstrzymanie potoku do identyfikacji stanu warunku

potok dynamiczny:

- IF – pobranie grupy instrukcji z kolejki i wstępna analiza (prognoza rozgałęzień)
- IS – wydanie kilku kolejnych instrukcji do bufora we EX
- EX – wytworzenie wyników działania instrukcji w buforze wy
- PA – wyprowadzenie wyniku z potoku (zapis w miejscu docelowym)

Niekolejne wykonanie rozkazów

Skuteczne sposoby łagodzenia konfliktów
– dla instrukcji rozgałęzienia: **prognoza**
– dla pozostałych instrukcji: **niekolejne wykonanie**

Wykonanie instrukcji w kolejności innej niż w zapisie algorytmu możliwe gdy:

- nie występuje konflikt danych
  – wykrycie możliwe podczas dostarczania operandów,
- nie występuje konflikt strukturalny
  – wykrycie możliwe podczas dekodowania.

Chwilowe pominięcie wykonania instrukcji, której argument jest niedostępny *nie musi wstrzymywać* wykonania instrukcji *kolejnych*.

Rozwiązania:
– statyczne: konflikt danych rozwiązywany po zdekodowaniu
– dynamiczne: zastąpienie dekodera przez dyspozytorem (ang. *dispatcher*), czyli układ *wydawania instrukcji* (ang. *instruction issue*).

Dyspozytor może analizować wiele instrukcji jednocześnie.

Dyspozytor rozkazów

- realizuje funkcje dekodera w dwóch etapach:
  – rozpoznanie działania i wykrycie konfliktu strukturalnego
  – dostarczenie operandów
  – opcjonalnie: wykrycie konfliktu danych
- możliwe jednoczesne rozpoznawanie kilku instrukcji

data flow – wykonanie rozkazów w rytmie gotowości danych – schematy:
– IS – *control flow*, EX – *data flow* (RISC, CISC)
– IS + EX – *data flow* (RISC)

Pierwsza metoda (w rytm dostępności jednostek wykonawczych: IS – *control flow*):
- notowanie stanu instrukcji w *stacji rezerwacyjnej* (ang. *reservation station*)
- sprawdzenie dostępności argumentów (ang. *scoreboarding*)
- rozsyłanie do jednostek wykonawczych jeśli dostępne są argumenty (ang. *issue*).

Druga metoda (w rytm dostępności argumentów IS – *data flow*):
- dostarczenie argumentów (ang. *issue*) do *stacji rezerwacyjnej* (ang. *reservation station*)
- notowanie stanu instrukcji w *stacji rezerwacyjnej* (ang. *reservation station*)
- ocena możliwości wykonania – sprawdzenie dostępności jednostki wykonawczej

Dekodowanie zgodne z napływem instrukcji, wykonanie dowolne

rozsyłanie instrukcji i sterowanie współbieżnością w etapie dekodowania.

– implementacja: CDC 6600 (1970),
– blokada rozsyłania jeśli występuje konflikt danych,
– blokada rozsyłania jeśli bufor zajęty przez instrukcje bieżące (ang. *pending*)

Stacja rezerwacyjna (bufor wejściowy) jest pojedyncza i zawiera:

- tablicę stanu operacyjnego instrukcji bieżących
- tablicę notowań dla rejestrów.

Chwilowy brak dostępu do rejestru nie blokuje wpisu do bufora rezerwacji

Tablica notowań (ang. *scoreboard*) zawiera dla każdego rejestru etykiety (*tag*):

- identyfikującą jednostkę dokonującą zapisu do tego rejestru
- identyfikującą jednostkę dokonującą odczytu z tego rejestru
- określającą priorytet (ang. *precedence*) zapisu/odczytu (kolejność dostępu).

Problem: możliwe konflikty WAR oraz WAW
– rozwiązanie: przemianowanie rejestrów (ang. *register rename*)

Tablica notowań dla rejestrów

Etykieta ***dostępu*** identyfikuje jednostkę funkcjonalną, która czeka na dostęp ($U_{\#}$) w celu wykonania odczytu lub zapisu, etykieta **0:** stan rejestru stabilny, rejestr dostępny dla każdego działania

Etykieta ***pierwszeństwa*** (*precedence*) wskazuje priorytet zapisu (W) lub odczytu (R).

register	R0	R1	R2	R3	R4		R31
access label (read)	0	…	U_a	0 /U_p	…		0
access label (write)	U_z	…	U_b	…	0 /U_q		0
precedence	W		R	~~(r)~~	~~(w)~~		

Tablica notowań (*scoreboard*) dla rejestrów ($U_{\#}$ - jednostka wykonawcza, W – zapis, R – odczyt).

Jeśli argument instrukcji jest wynikiem innej instrukcji aktualnie wykonywanej, to wpis „$U_{\#}$" do tablicy notowań jest zbędny, ale *pozostaje etykieta R* (wynik instrukcji jest rozsyłany (*broadcast*) wewnętrzną magistralą i może być z niej pobrany jednocześnie z zapisem do rejestru (skrót na ścieżce danych)).

By uniknąć blokady strukturalnej, należy umożliwić dostęp jednoczesny do kilku rejestrów, co wymaga wieloportowego układu dostępu.

Przykład
Jest 8 rejestrów i 3 jednostki funkcjonalne (mnożąca – MUL, dzieląca – DIV i dodająco-odejmująca – A/S).
Stan notowań rejestrów podczas wykonania DIV w sekwencji:

```
DIV R3, R1, R2          ; R3=R1/R2  (jednostka DIV)
MUL R5, R3, R4          ; R5=R3*R4 (RAW) (jednostka MUL)
ADD R4, R6, R7          ; R4=R6+R7 (WAR) (jednostka A/S)
```

jest następujący

	R0	R1	R2	R3	R4	R5	R6	R7
read access	0	0	0	0*	MUL	0	A/S	A/S
write access	0	0	0	DIV	A/S	MUL	0	0
precedence	—	—	—	W	R	W	R	R

Stan jednostek funkcjonalnych :

DIV: OP1=R1, OP2=R2, DEST=R3
MUL: OP1=DIV, OP2=R4, DEST=R5 (*R3 będzie dostarczony skrótem z magistrali)
A/S: OP1=R6, OP2=R7, DEST=R4 (czeka na zakończenie MUL)

Ulepszenie – jeśli w stacji rezerwacyjnej przechowuje się kopię danych z rejestru zamiast jego numeru, zbędne jest markowanie oczekiwania na odczyt i notowanie priorytetu odczyt/zapis.
uproszczona tablica:

	R0	R1	R2	R3	R4	R5	R6	R7
source	–	–	–	DIV	A/S	MUL	–	–

Opis wykonania sekwencji instrukcji

Kompletny opis stanu wykonania obejmuje oprócz notowań rejestrów, tablicę stanu instrukcji w toku (*pending*) i tablicę stanu jednostek funkcjonalnych.

Opis dla struktury potoku: IF–IS–DF–EX–PA i bufora 7-stanowiskowego:
Przykładowy status stacji rezerwacyjnej (Qz – źródło argumentu: Rz – rejestr, (Rx) – kopia rejestru w buforze, XX – jednostka dostarczająca argument, Az – dostepność rejestru: N – czekanie na wynik lub kopia rejestru w buforze, T – rejestr wolny dla odczytu).
Kursywą zaznaczono instrukcje oczekujące.

Instrukcja	Etap			
(*akcja Rd, Rx, Ry*)	IS	DF	EX	PA
add R1, R5, R3	√	√	√	√
ld R2, [R3]	√	√	√	
add R4, R2, R7	√			
mul R5, R1, R3	√	√		
st [R6], R4				
sub R1, R2, R6	√			↑
mul R6, R3, R4				↓
and R4, R3, R7				

Jednostka		Rejestry			Źródło		Dostępny	
nazwa	zajęta	Rd	Rx	Ry	Qx	Qy	Ax	Ay
LS	Tak	R2	R3		(R3)		(t)	
IU1	Tak	R4	R2	R7	LS	R7	N	T
MP	Tak	R5	R1	R3	(R1)	R3	N	T
LS	*Nie*	*[…]*	*R4*	*R6*				
IU2	Tak	R1	R2	R6	LS	(R6)	N	T
MP	*Nie*	*R6*	*R3*	*R4*				
IU1	*Nie*	*R4*	*R3*	*R7*				

rejestr	R0	R1	R2	R3	R4	R5	R6	R7
źródło	—	IU2	LS	—	IU1	MUL	—	—

(+1 cykl) – zakończona instrukcja ld R2, [R3]

Instrukcja	Etap			
(*akcja Rd, Rx, Ry*)	IS	DF	EX	PA
add R1, R5, R3	√	√	√	√
ld R2, [R3]	√	√	√	√
add R4, R2, R7	√	√		
mul R5, R1, R3	√	√	√	
st [R6], R4	√			
sub R1, R2, R6	√	√		
mul R6, R3, R4				↑
and R4, R3, R7				↑
add R1, R2, R5				

Jednostka		Rejestry			Źródło		Dostępny	
nazwa	zajęta	Rd	Rx	Ry	Qx	Qy	Ax	Ay
—	—				—		—	—
IU1	Tak	R4	R2	R7	(R2)	R7	N	T
MP	Tak	R5	R1	R3	R1	(R3)	N	N
LS	Tak	[…]	R4	R6	IU1	R6	N	T
IU2	Tak	R1	R2	R6	(R2)	R6	N	T
MP	*Nie*	*R6*	*R3*	*R4*				
IU2	*Nie*	*R4*	*R3*	*R7*				
IU1	*Nie*	*R1*	*R2*	*R5*				

rejestr	R0	R1	R2	R3	R4	R5	R6	R7
źródło	—	IU2	—	—	IU1	MUL	—	—

(+3 cykle) – zakończone instrukcje mul R5, R1, R3 (EX1+EX2), add R4, R2, R7 oraz sub R1, R2, R6

Instrukcja	Etap			
(*akcja Rd, Rx, Ry*)	IS	DF	EX	PA
add R1, R5, R3	√	√	√	√
ld R2, [R3]	√	√	√	√
add R4, R2, R7	√	√	√	√
mul R5, R1, R3	√	√	√	√
st [R6], R4	√	√	√	
sub R1, R2, R6	√	√	√	√
mul R6, R3, R4	√	√		
and R4, R3, R6	√	√		
add R1, R2, R5	√			
				↑

Jednostka		Rejestry			Źródło		Dostępny	
nazwa	zajęta	Rd	Rx	Ry	Qx	Qy	Ax	Ay
—	—							
LS	Tak	[…]	R4	R6	(R4)	(R6)	N	N
MP	Tak	R6	R3	R4	R3	R4	T	T
IU2	Tak	R4	R3	R6	R3	MP	T	T
IU1	Tak	R1	R2	R5	R2	R5	T	T

rejestr	R0	R1	R2	R3	R4	R5	R6	R7
źródło	—	IU1	—	—	IU2	—	MUL	—

Algorytm Tomasulo (1)

Szeregowanie działań w rytm dostępności danych (R. Tomasulo, '67) – IBM360/91

Zasada algorytmu Tomasulo – dostarczanie *danych* do stacji rezerwacyjnych
Wszelkie transfery danych następują przez wspólną magistralę
– możliwe wiele kopii aktualnej danej w różnych buforach,
– możliwe różne edycje jednej zmiennej (stare i nowe).
– zapis do rejestru docelowego możliwy dopiero, gdy wszystkie potrzebne kopie dostarczono do stacji rezerwacyjnych.
– jeśli źródło danych jest chwilowo niedostępne, do bufora zostaje wysłany adres, a bufor przechodzi w tryb podglądania magistrali.

Pozostałe istotne różnice tego schematu w porównaniu z poprzednim to:

- stacja rezerwacyjna osobna dla każdej jednostki funkcjonalnej
- możliwość dynamicznego anulowania zapisu do rejestru docelowego, jeśli kopie są rozesłane do buforów i nie będą potrzebne przed kolejnym zapisem.

Algorytm Tomasulo (2)

– nie ma potrzeby dostępu do scentralizowanego pliku rejestrowego
– wyeliminowane konflikty WAR (zapis do rejestru możliwy gdy wszystkie kopie jego starej zawartości są już w buforach stacji rezerwacyjnych SR) oraz WAW. (wiele kopii danych w buforach stacji rezerwacyjnych jest koncepcyjnie tożsame z przemianowaniem rejestrów – w danej chwili dostępnych może być wiele edycji zmiennej, z których część nigdy nie była zapisana w rejestrze docelowym.

Do synchronizacji wykonania rozkazów wystarczy rozróżnienie trzech etapów:

- wydanie instrukcji (*instruction issue, IS*) – przekazanie instrukcji do stacji
- wykonanie (*execute, EX*) – skompletowanie danych i wytworzenie wyniku
- ekspedycja wyniku (*put away, PA*) – wysłanie wyniku do rejestru.

– wydanie instrukcji tylko wtedy, gdy we właściwej SR jest miejsce; w przeciwnym razie wstrzymane wydawanie wszystkich instrukcji.
– wydawanie jest wstrzymane także wtedy, gdy między instrukcją kierowaną do SR a instrukcją już tam umieszczoną jest konflikt RAW; nie blokuje to wysłania kolejnych instrukcji które nie są w zależności RAW z instrukcją wstrzymaną lub obecną w stacji rezerwacyjnej.

Algorytm Tomasulo (3)

– wykonanie instrukcji podjęte tuż po skompletowaniu danych
– w SR obecne są tylko instrukcje już wydane (*IS*), których wykonanie (*EX*) nie zostało jeszcze zakończone (jeśli każdy bufor mieściłby tylko jedną instrukcję, to nie wystąpiłby efekt szeregowania w rytm gotowości danych i algorytm działałby tak, jak prosta implementacja tablicy notowań).
– jednostka pobierania danych z pamięci (*load*) może działać tak jak inne jednostki wykonawcze i posiadać własną stację rezerwacyjną, jeśli przesłania danych odbywają się wspólną magistralą.
– jednostka zapisu do pamięci (*store*) musi działać autonomicznie, a aby uniknąć konfliktów WAR i WAW powinna być wyposażona w bufor zapisu.
– ograniczenie szybkości algorytmu Tomasulo: przepustowość wspólnej magistrali.

Algorytm Tomasulo jest skuteczny w realizacji iterowanych pętli (równoważne dynamicznemu sprzętowemu rozwijaniu pętli (ang. *dynamic loop unrolling*) o ile wyniki i argumenty mają różne adresy

Algorytm Tomasulo – przykład

Instrukcja	Etap		
(*akcja Rd, Rx, Ry*)	IS	EX	PA
ld R2, [R4]	√	√	
mul R5, R1, R3	√		
add R4, R2, R4	√		
sub R1, R2, R6	√		
div R6, R3, R4	√		
and R0, R3, R1	√		
add R1, R2, R5			

Jednostka						
bufor	zajęta	działanie	Vx	Vy	Qx	Qy
LD1	T	ld	(R4)	—		—
MD1	T	mul	(R1)	(R3)		
IU1	T	add		(R4)	LD1	
IU2	T	sub		(R6)	LD1	
MD2	T	div	(R3)			IU1
IU3	T	and	(R3)	(R1)		
IU#						

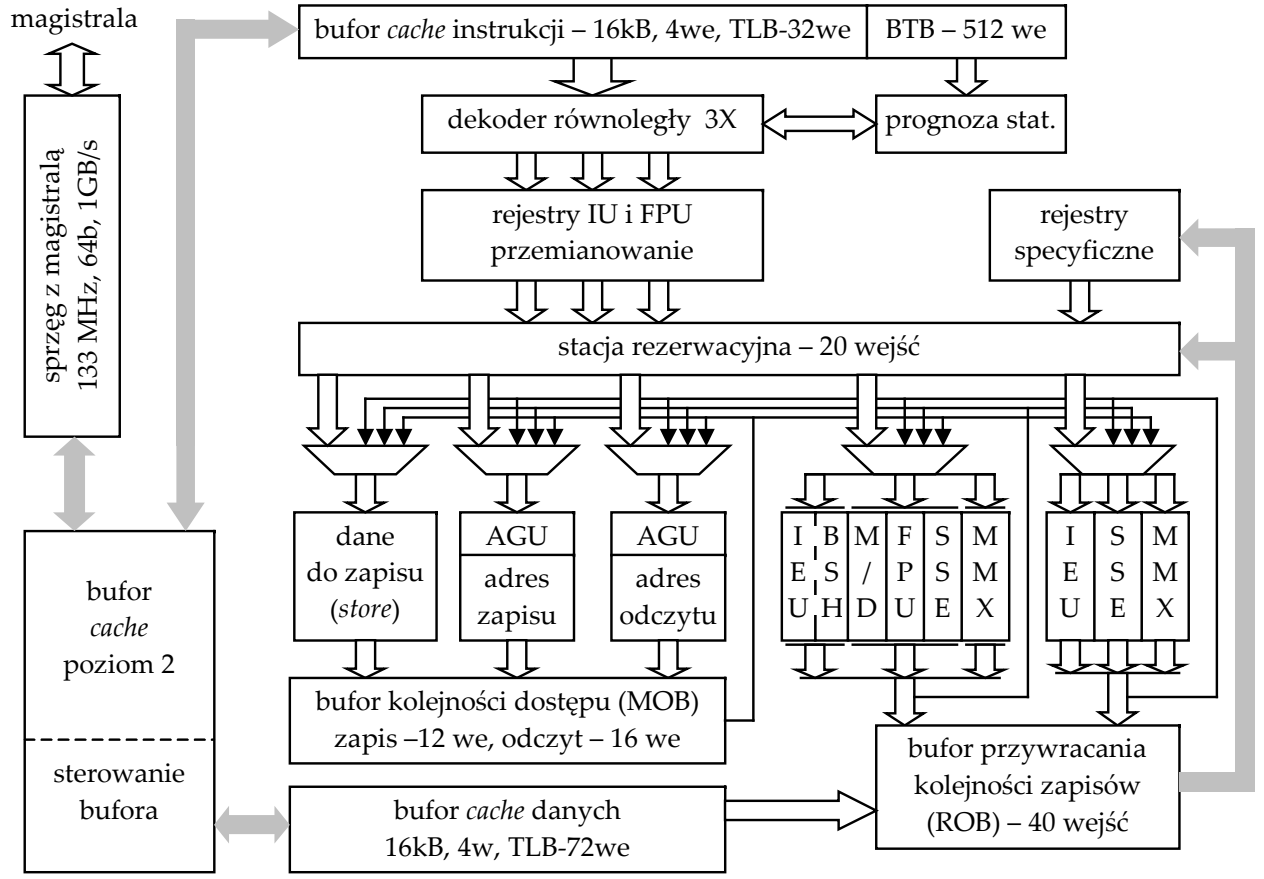
(+1 cykl) – zakończone instrukcje add R4, R2, R4 oraz sub R1, R2, R6

Instrukcja	Etap		
(*akcja Rd, Rx, Ry*)	IS	EX	PA
mul R5, R1, R3	√	√	
add R4, R2, R4	√	√	√
sub R1, R2, R6	√	√	√
div R6, R3, R4	√	√	
and R0, R3, R1	√	√	
add R1, R2, R5	√		
sub R3, R7, R1	√		

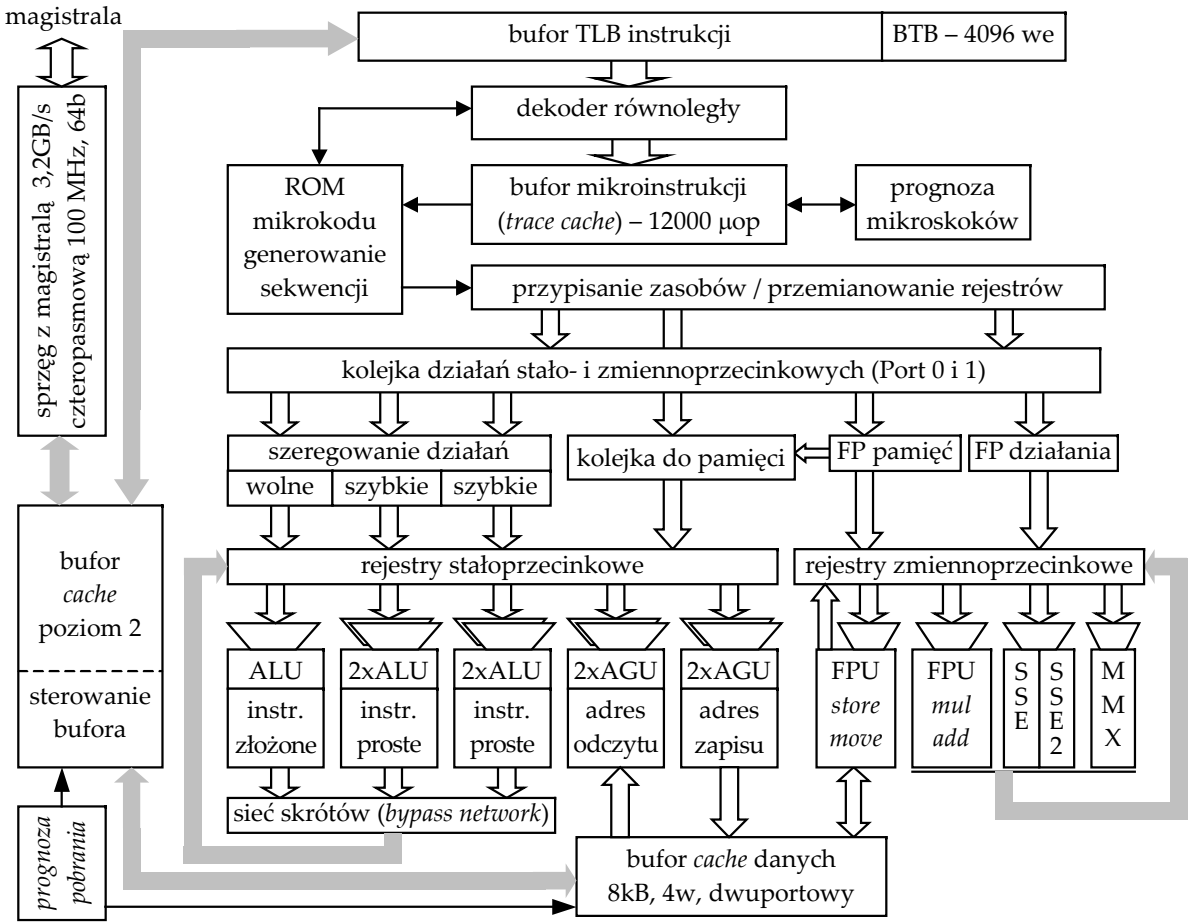
Jednostka						
bufor	zajęta	działanie	Vx	Vy	Qx	Qy
MD1	T	mul	(R1)	(R3)		
MD2	T	div	(R3)	(IU1)		
IU3	T	and	(R3)	(R1)		
IU1	T	add	(R2)			MD1
IU2	T	sub	(R7)			IU1

Algorytm Tomasulo (Vz – bufor zawierający argument lub rejestr, który go zawiera,
Qz – dostępność danych wytworzonych przez bieżące instrukcje, pierwszy na liście argument docelowy).
Jednocześnie mogą być wykonywane 3 instrukcje proste (w IU1–3) i 2 złożone (w MD1 lub MD2)

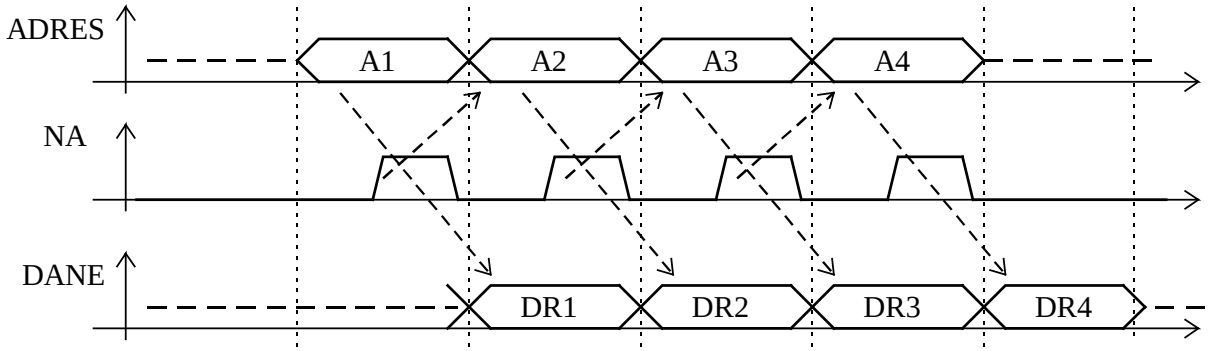
Data flow w architekturze IA-32 (Pentium 3)



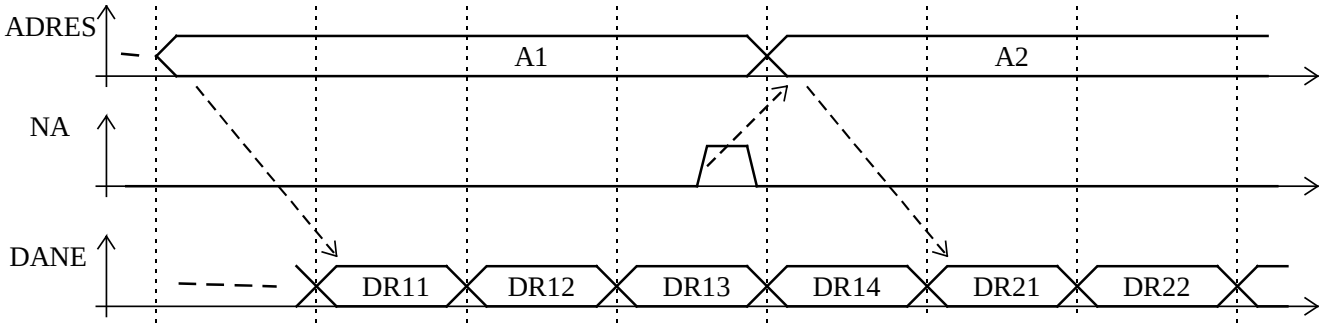
Data flow w architekturze IA-32 (Pentium 4)



Potokowy dostęp do pamięci



Potokowy dostęp do pamięci



Potokowy transfer blokowy

*Procedura doboru ziarnistości (struktury) potoku

Rozwiązanie przybliżone

- na podstawie planowanej organizacji maszyny i statystyk częstości wykonania rozkazów oceń współczynnik b, gdy ich brak, przyjmij b=0,2
- jeśli brak innych przesłanek, przyjmij k=0,05 (w praktyce k<0,1)
- na podstawie przyjętej metody taktowania i parametrów zatrzasków oceń C
- oblicz $T = t_1 + t_2 + \ldots + t_n$
- oblicz pierwsze przybliżenie n_0 jako (ρ=0, $r \ll b$)

$$n_0 = \left| \sqrt{b^{-1}(1+k)TC^{-1}} \right|,$$

któremu odpowiada długość cyklu $\Delta t_0 \cong T n_0^{-1}$

Procedura

- sprawdź, czy istnieje podział $\{t_i < \Delta t_0\}$,
  - jeśli nie, to ustal $\Delta t_0 = \max t_i$
  - jeśli tak, to oblicz $G(n_0)$, zwiększ n_0 i powtórz działanie
- przeanalizuj podziały na dłuższe etapy Δt, takie że $t_i + t_{i+1} + \ldots + t_s < \Delta t$
- wybierz Δt, które minimalizuje efekt kwantyzacji i narzut zegara.

*Dobór struktury i głębokości potoku – potok statyczny

- (IF) pobranie kodu – 23 ns, w tym:
  - (PC) wytworzenie w rejestrze PC rzeczywistego adresu instrukcji – 4 ns
  - (CA) dostęp do katalogu pamięci podręcznej – 6 ns
  - (CR) dostęp do danych w pamięci podręcznej – 10 ns
  - (CF) pobranie z pamięci podręcznej do rejestru rozkazów IR – 3 ns
- (ID) dekodowanie – 12 ns, zawierające 2 nierozdzielne etapy:
  - (D1) identyfikacja typu instrukcji i (D2) pobranie operandów adresowych
- (AG) wytworzenie adresu – 12 ns, obejmujące:
  - (RA) obliczenie (wytworzenie) adresu rzeczywistego – 9 ns
  - (AW) zapis do rejestru adresowego pamięci – 3 ns
- (DF) pobranie danych – 19 ns, w tym:
  - (CA) dostęp do katalogu pamięci podręcznej – 6 ns
  - (CR) dostęp do danych w pamięci podręcznej – 10 ns
  - (CF) pobranie z pamięci podręcznej do rejestru – 3 ns
- (EX) wykonanie – 25 ns, obejmujące wraz z zapisem:
  - (E1) dostęp do rejestrów – 6 ns
  - (E2) wykonanie działania w ALU – 8 ns
  - (E3) wytworzenie wyniku – 7 ns
- (WR) zapis wyniku do rejestru i aktualizacja PC – 3 ns.

*Dobór struktury i głębokości potoku – potok statyczny (c.d.)

T=90 ns $\rightarrow$ gdy b=0,2, k=0,05 oraz C=4 ns, mamy n_0=9,721111

Gdy $n \geq 9$, warunek $t_i < \Delta t \leq 10$ ns nie jest spełniony dla niepodzielnej fazy ID
$\rightarrow$ najkrótszy cykl musi trwać $\Delta t \geq 12$ ns

Podziały nominalnego cyklu instrukcji T=90 ns (b=0,2)

$T_c = (1{,}05^*\Delta t + 4)$[ns], $G = \{T_c^*(1+(n-1)b)\}^{-1}$[MIPS].

Δt	Etapy ($t_i \leq \Delta t$)	T_c	n	G	nT_c
~~10 ns~~	~~(4-6) 10-3 (12+) 9 (3-6) 10 (3-6) 8 (7-3)~~	~~14,50 ns~~	~~11~~	~~24,63~~	~~145,0 ns~~
12 ns	(4-6)-10-3-12-9-(3-6)-10-(3-6)-8-(7-3)	16,60 ns	10	21,51	166,0 ns
13 ns	(4-6)-(10-3)-12-9-(3-6)-10-(3-6)-8-(7-3)	17,65 ns	9	21,79	158,5 ns
14 ns	(4-6)-(10-3)-12-9-(3-6)-(10-3)-(6-8)-(7-3)	18,70 ns	8	22,28	149,6 ns
20 ns	(4-6-10)-(3-12)-(9-3-6)-(10-3-6)-(8-7-3)	25,00 ns	5	22,22	125,0 ns
24 ns	(4-6-10-3)-(12-9)-(3-6-10-3)-(6-8-7-3)	29,20 ns	4	21,40	116,8 ns

- mniej etapów – mniej układów separujących
- krótszy cykl – mniejsze straty czasu wywołane zakłóceniami potoku
  – oszczędność czasu przy wykonaniu długich instrukcji