

Principy počítačů a operačních systémů

Zvyšování výkonnosti procesorů

Zimní semestr 2007/2008

Zvyšování výkonnosti procesorů

Mikroarchitektura

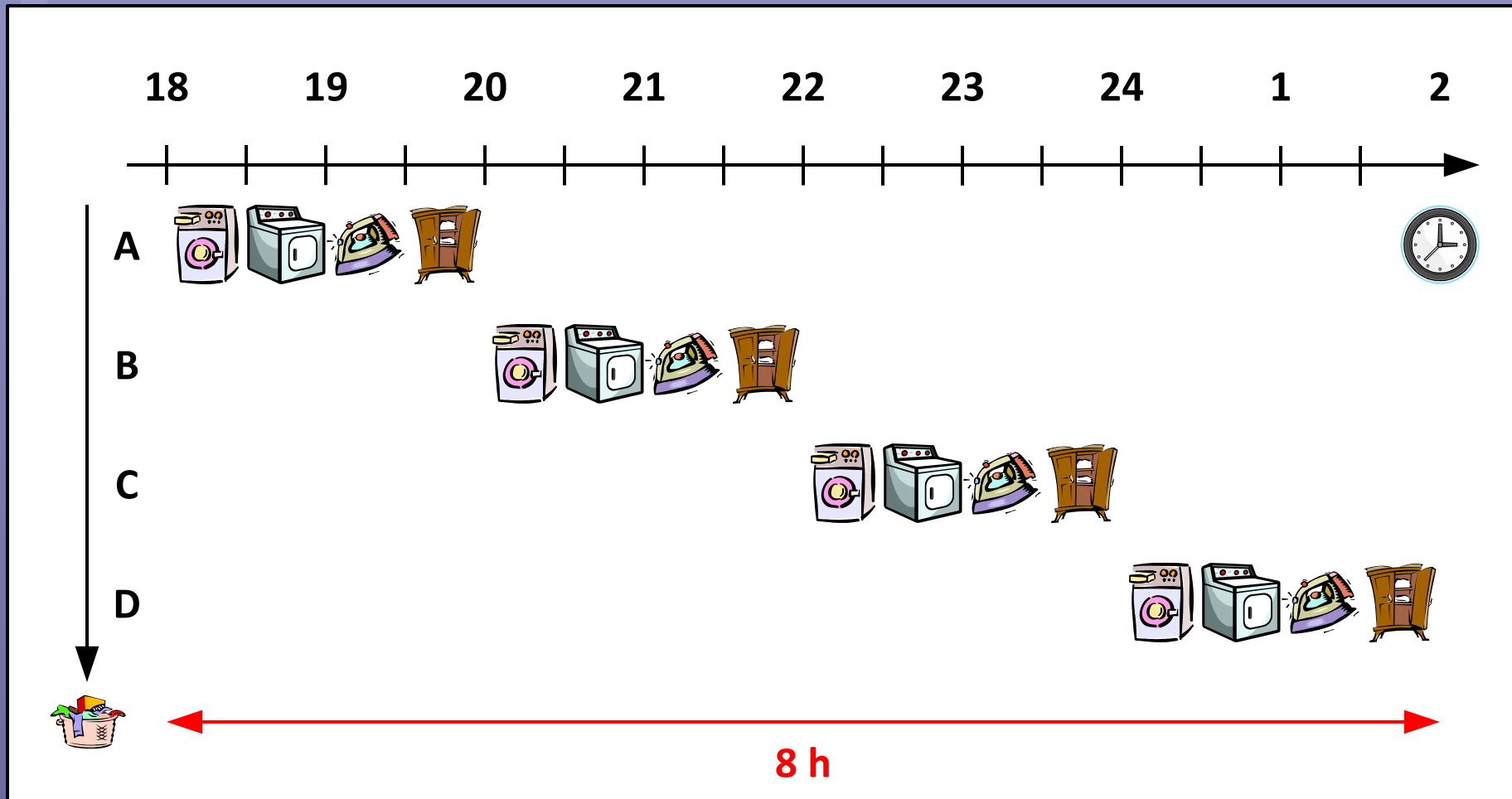
- Proudové zpracování
 - ♦ pipelining, deep pipelining
- Současné vykonávání více instrukcí
 - ♦ static/dynamic multiple issue
- Spekulativní provádění instrukcí
 - ♦ reakce na problémy s výše uvedeným

Instruction set architecture

- usnadnění realizace výkonné mikroarchitektury

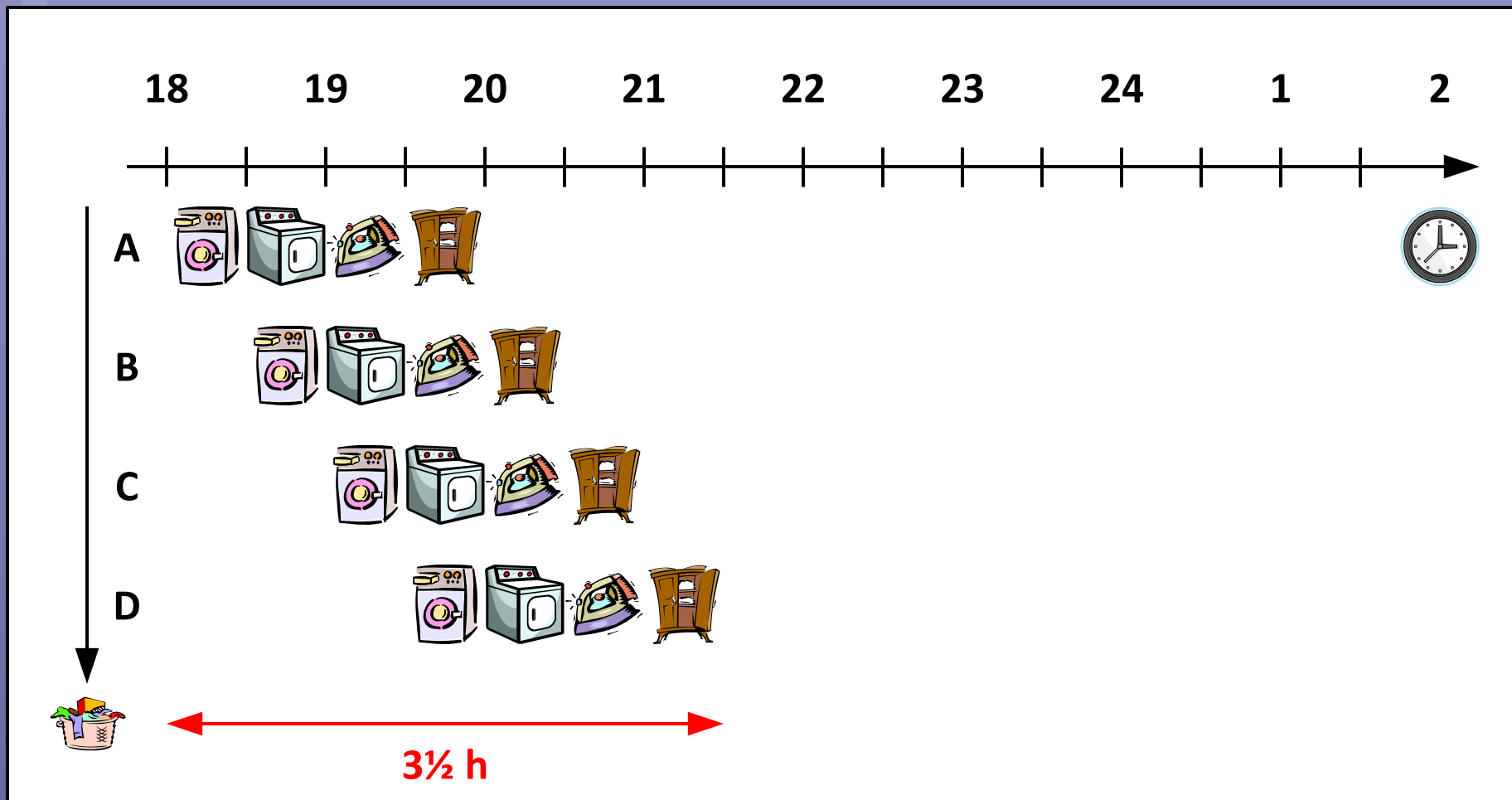
Proudové zpracování (1)

Motivace: neefektivní využití prostředků



Proudové zpracování (2)

Řešení: minimalizace prostojů



Proudové zpracování (3)

Zpracování instrukcí procesorem (MIPS)

- načtení instrukce z paměti
- přečtení registrů a dekódování instrukce
- vykonání operace nebo výpočet adresy
- přístup k operandům v paměti
- zápis výsledku do registru

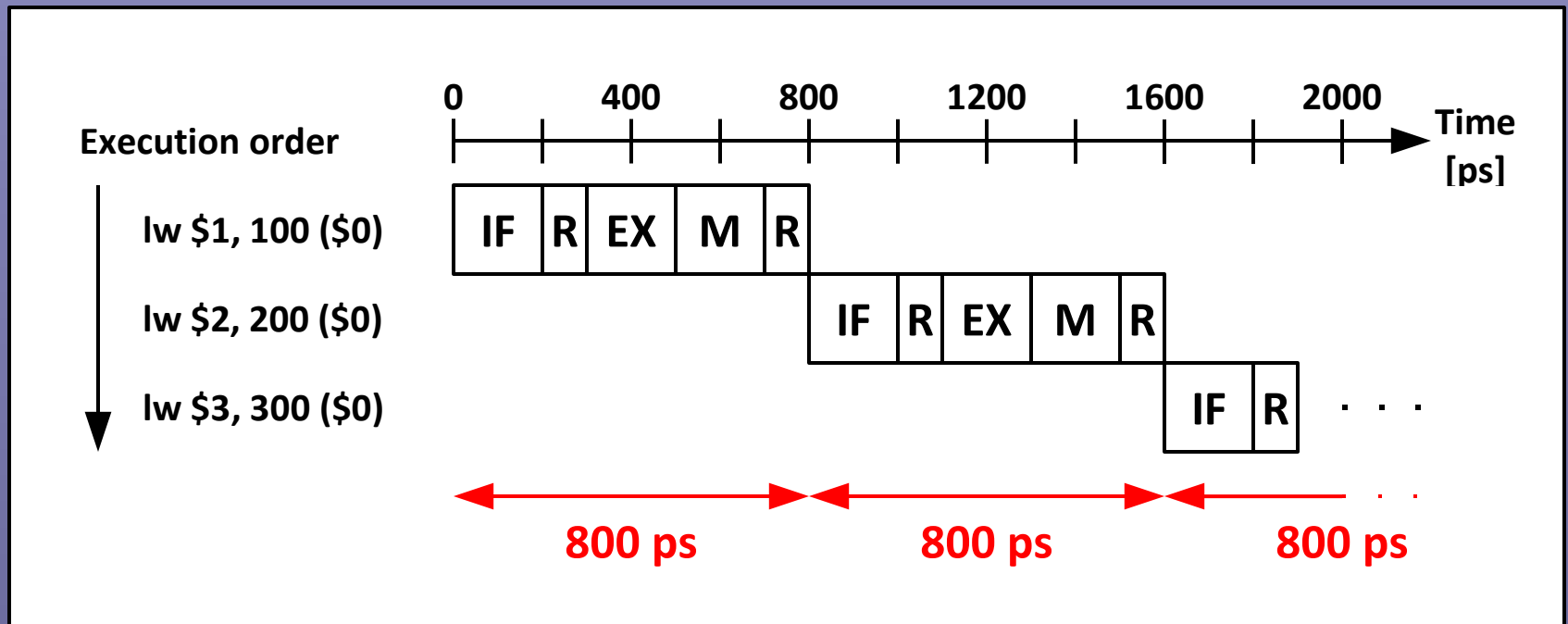
Jednocyklové vs. proudové zpracování

- výkon závisí na nejpomalejší operaci

Proudové zpracování (4)

Jednocyklová datová cesta

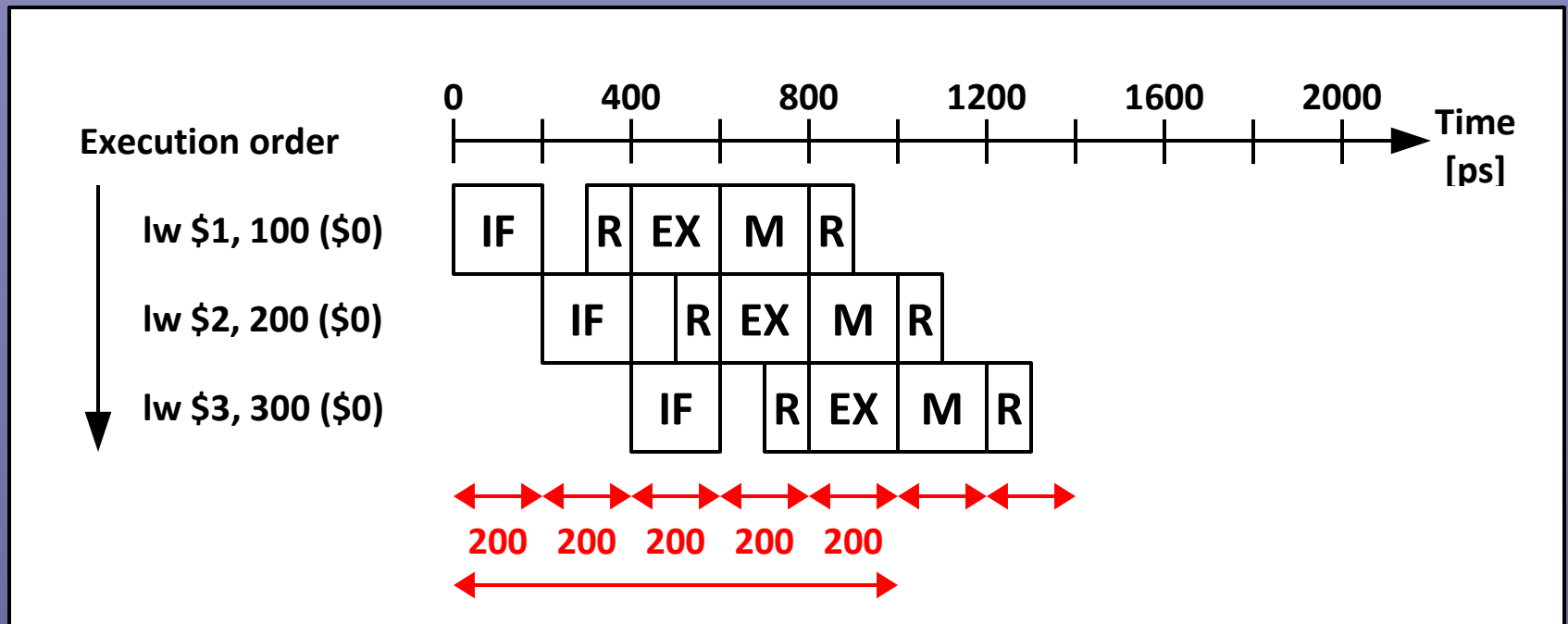
- instrukce *load word* (nejdelší)
 - ♦ práce s registry 100ps, ostatní 200ps



Proudové zpracování (5)

Proudová datová cesta

- instrukce *load word* (nejdelší)
 - ♦ jednotlivé kroky 200ps



Proudové zpracování (6)

Zrychlení při proudovém zpracování

- zpracování n instrukcí v pipeline s k kroky
- hodinový cyklus s periodou t_{clk}

- zrychlení proti sekvenčnímu zpracování

$$S = \frac{T_s}{T_p} = \frac{n \cdot k \cdot t_{clk}}{k \cdot t_{clk} + (n-1) \cdot t_{clk}} = \frac{n \cdot k}{k + (n-1)}$$

- pro $n \gg k$

$$k + (n-1) \approx n, S \rightarrow k$$

Proudové zpracování (7)

Návrh instrukční sady (MIPS)

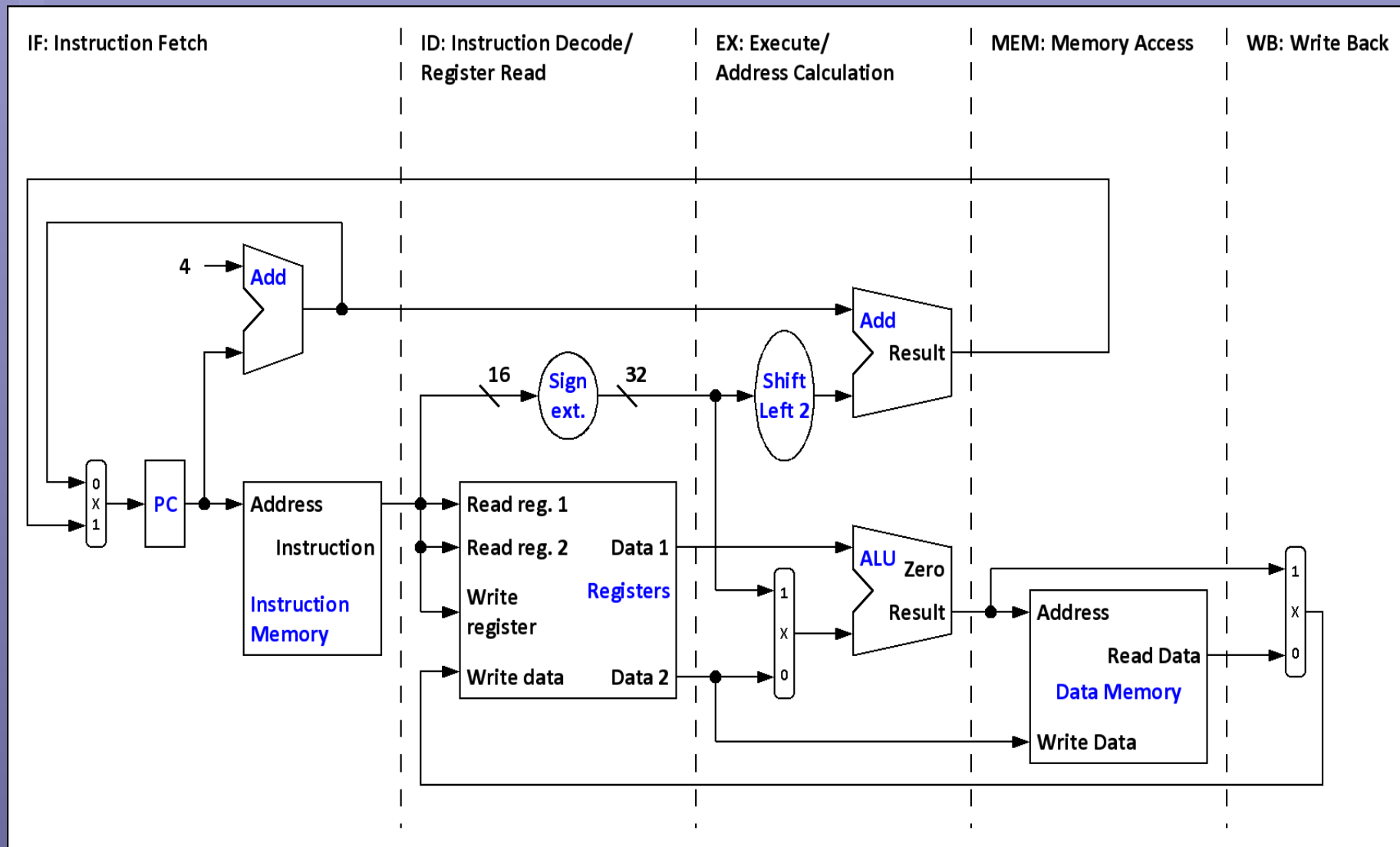
- instrukční kód stejné délky
 - ♦ jednoduché načítání instrukcí do pipeline
- omezený počet formátů instrukčního kódu
 - ♦ čtení registrů probíhá zároveň s dekodováním instrukce
- paměťové operandy pouze v load/store operacích
 - ♦ výpočet adresy ve fázi vykonání instrukce
 - ♦ přístup do paměti v následujícím kroku
- operandy na zarovnaných adresách
 - ♦ přístup do paměti vyžaduje pouze 1 stupeň pipeline

Proudové zpracování (8)

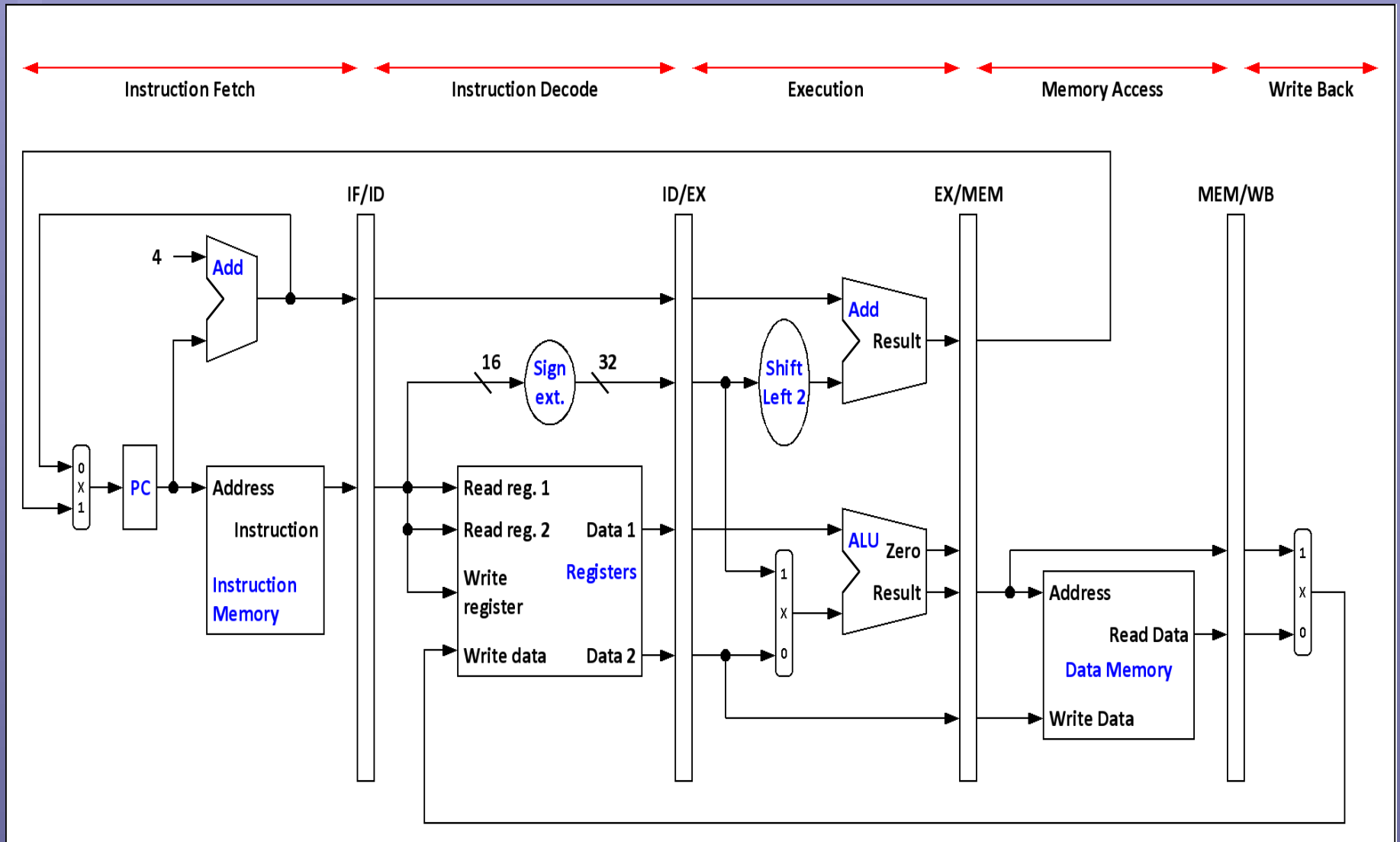
Konstrukce datové cesty

- zpracování rozděleno k stupňů (stages)
 - ♦ nejpomalejší stupeň určuje rychlost pipeline
 - ♦ obdoba nejdelší instrukce v jednocyklové datové cestě
- registry pro předávání dat mezi stupni
 - ♦ obdoba pásu na výrobní lince
 - ♦ obdoba pomocných registrů ve víceciklové datové cestě

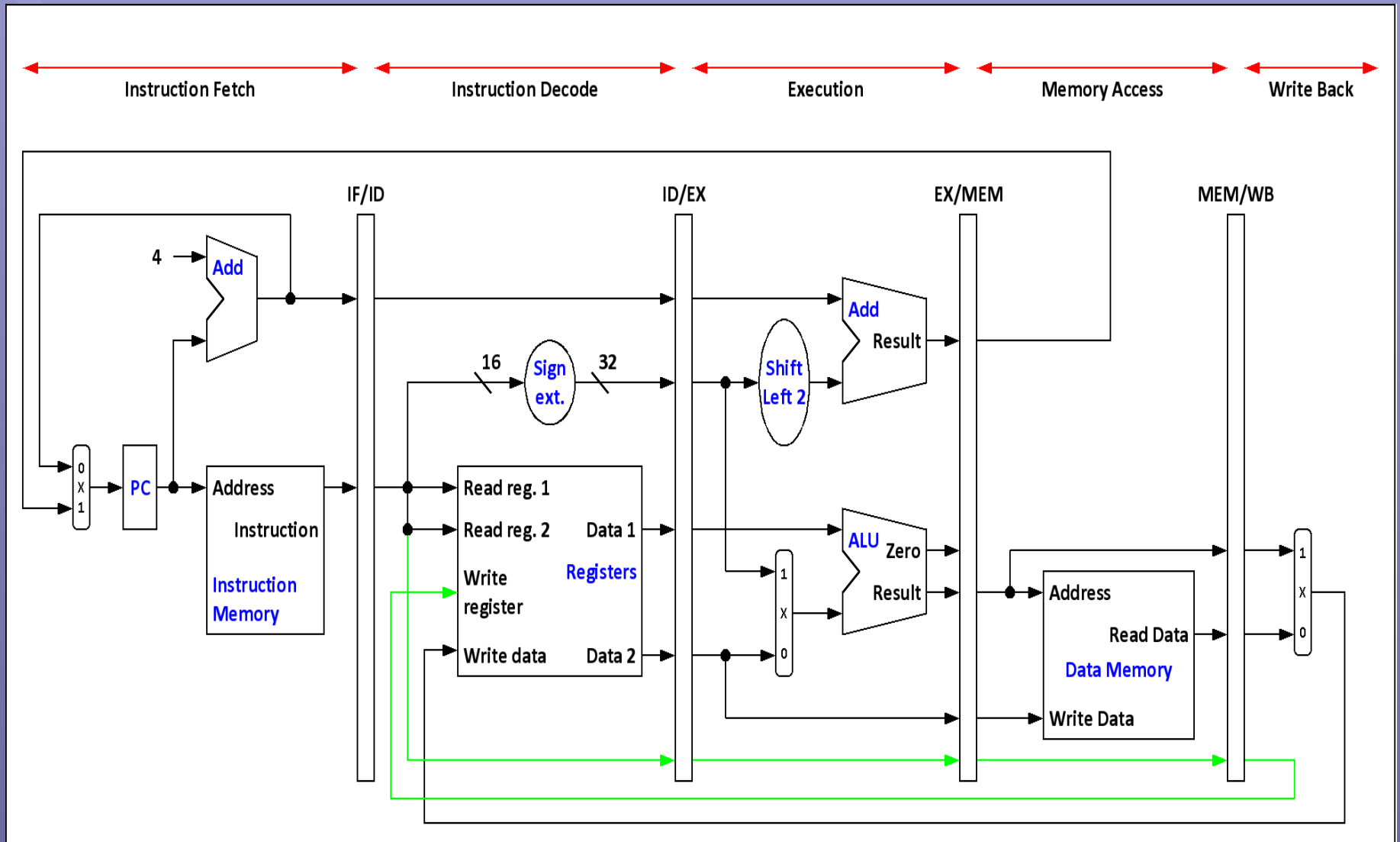
MIPS: Jednocyklová datová cesta



MIPS: Proudová datová cesta (1)



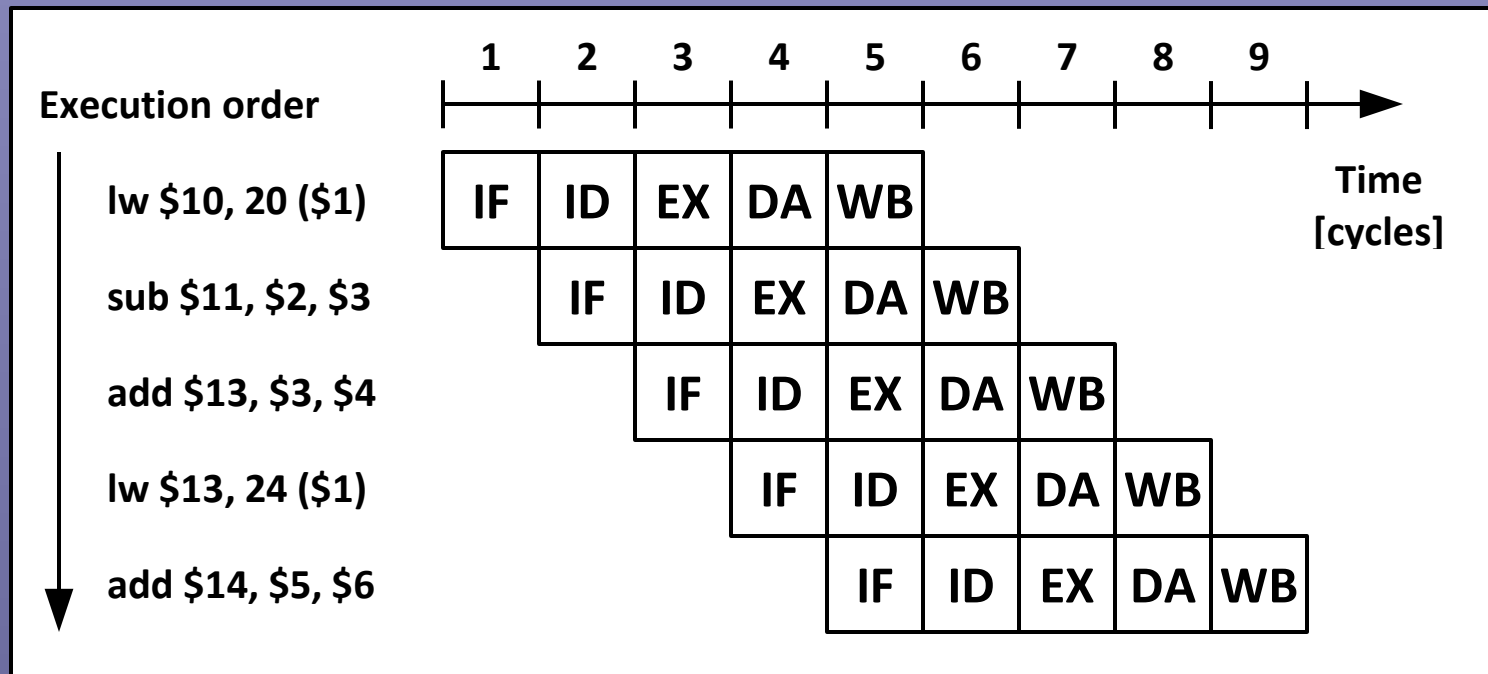
MIPS: Proudová datová cesta (2)



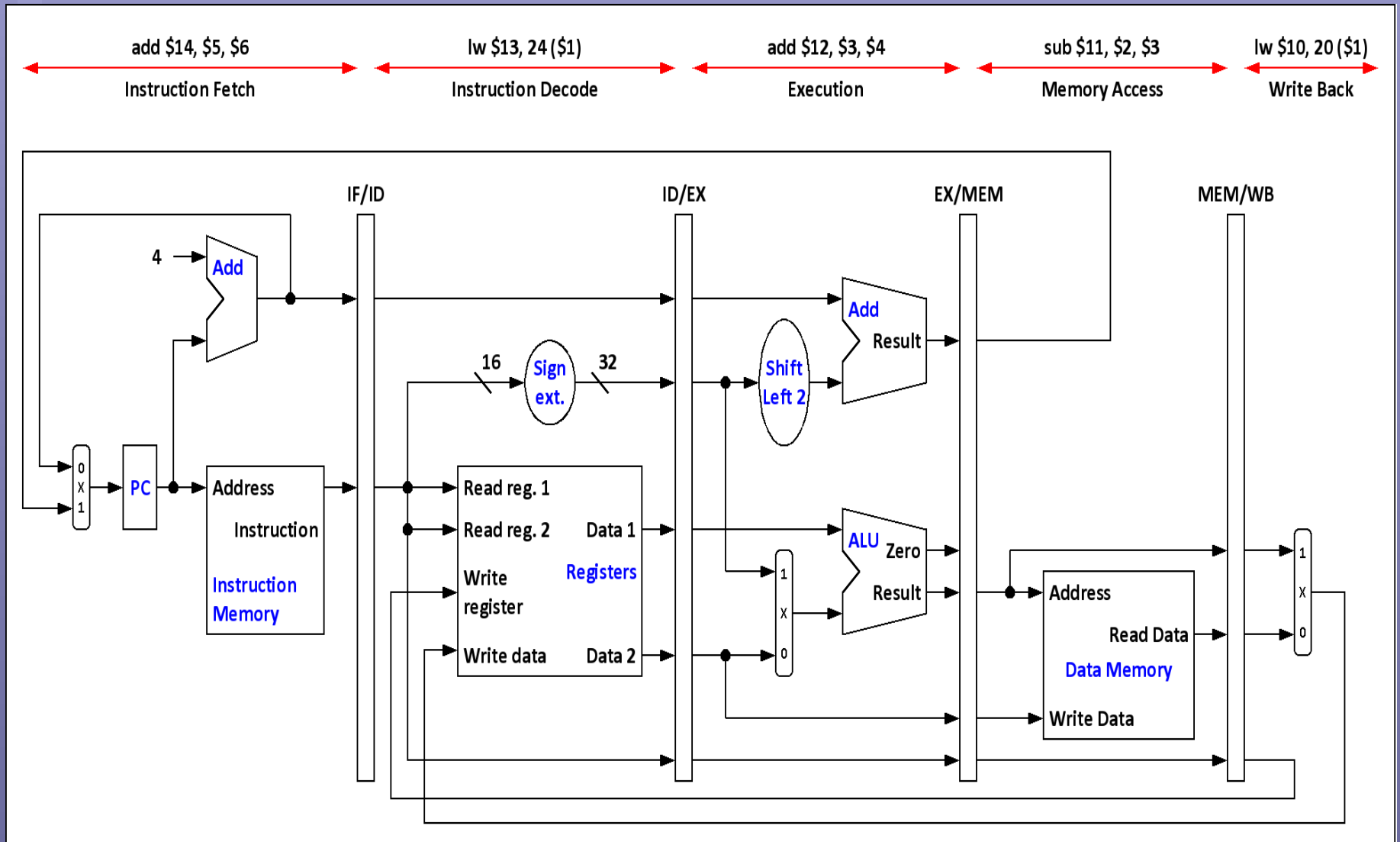
Proudové zpracování (9)

Reprezentace pipeline

- relativní čas (v hodnových cyklech)
- všechny fáze trvají 1 cyklus



MIPS: Proudová datová cesta (3)

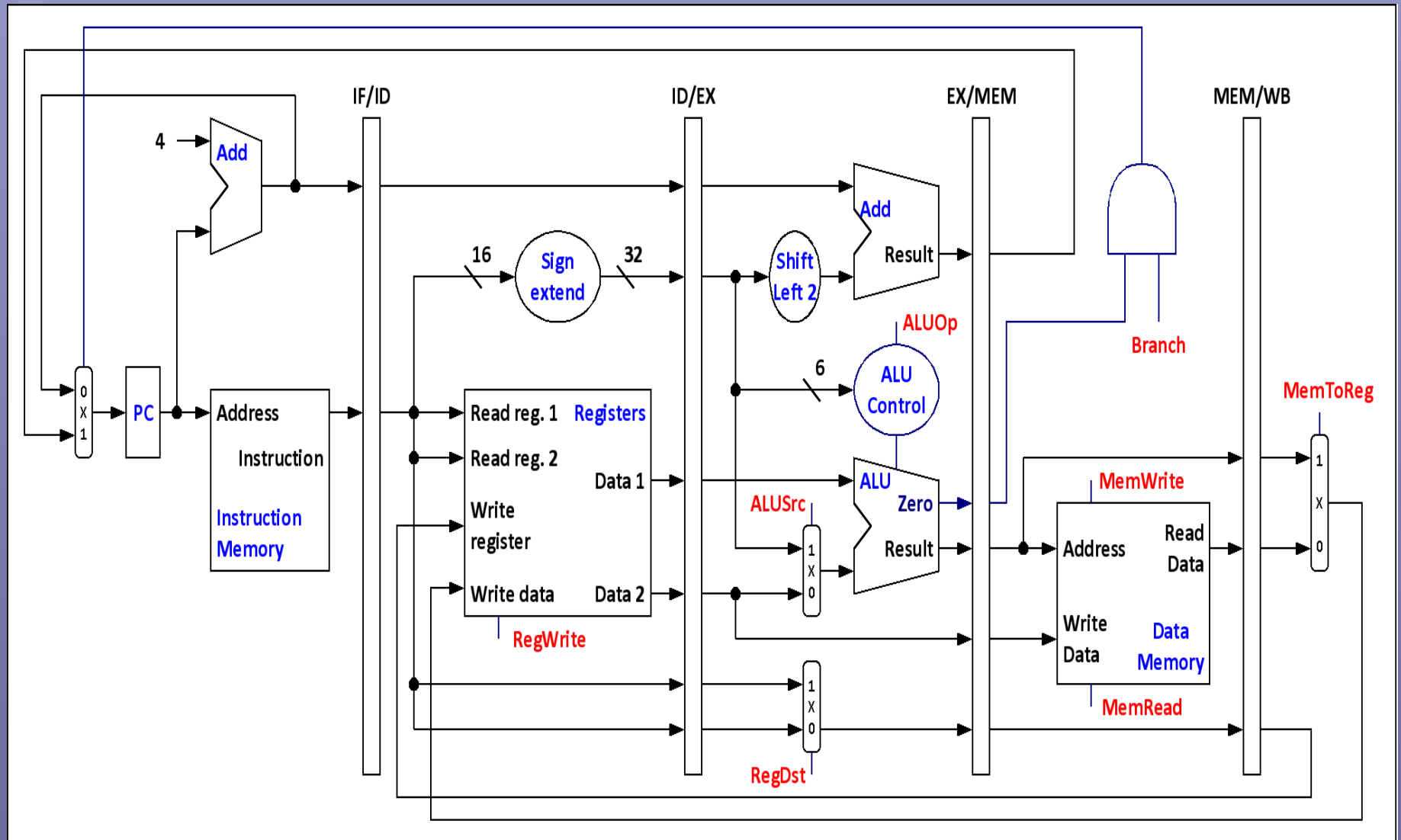


Proudové zpracování (10)

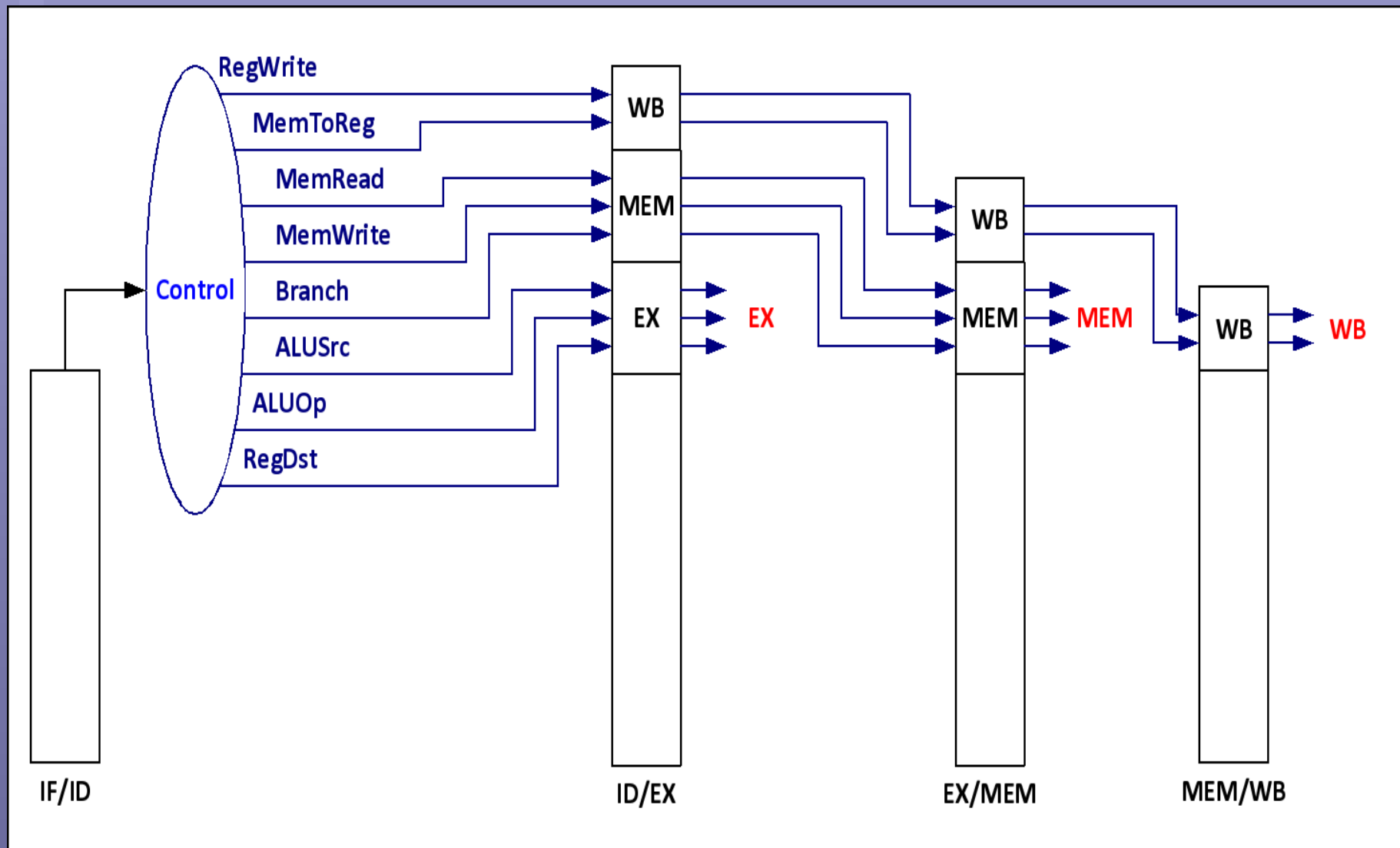
Řízení proudové datové cesty

- různé instrukce v různých fázích
 - ♦ řídicí signály pro jednotlivé stupně pipeline
- původní jednocyklový řadič
 - ♦ kombinační obvod dekodující operační kód
 - ♦ signály zapsány do pipeline registru při načtení instrukce

MIPS: Proudová datová cesta (4)



MIPS: Řízení proudové datové cesty



Proudové zpracování (10)

Problémy při proudovém zpracování

- strukturální hazard
 - ♦ hardware nepodporuje danou kombinaci instrukcí
 - ♦ současný přístup k prostředku z více stupňů pipeline
- datový hazard
 - ♦ instrukce nemá k dispozici data pro vykonání
 - ♦ čeká se na výsledek předchozí instrukce
- řídicí hazard
 - ♦ nutno učinit rozhodnutí před vykonáním instrukce
 - ♦ načtení následující instrukce po instrukci skoku

Datový hazard v proudovém zpracování (1)

Závislosti mezi operandy instrukcí

- operand je výsledek předchozí operace
- operand je obsah paměti čtený předchozí instrukcí
- zjišťování závislostí
 - ♦ diagram pipeline + hrany označující závislost
 - ♦ závislosti jsou hrany “dopředu v čase”

Ošetření datového hazardu

- dodání mezivýsledku (forwarding/bypassing)
- pozdržení instrukce v pipeline (pipeline stall)

Datový hazard v proudovém zpracování (2)

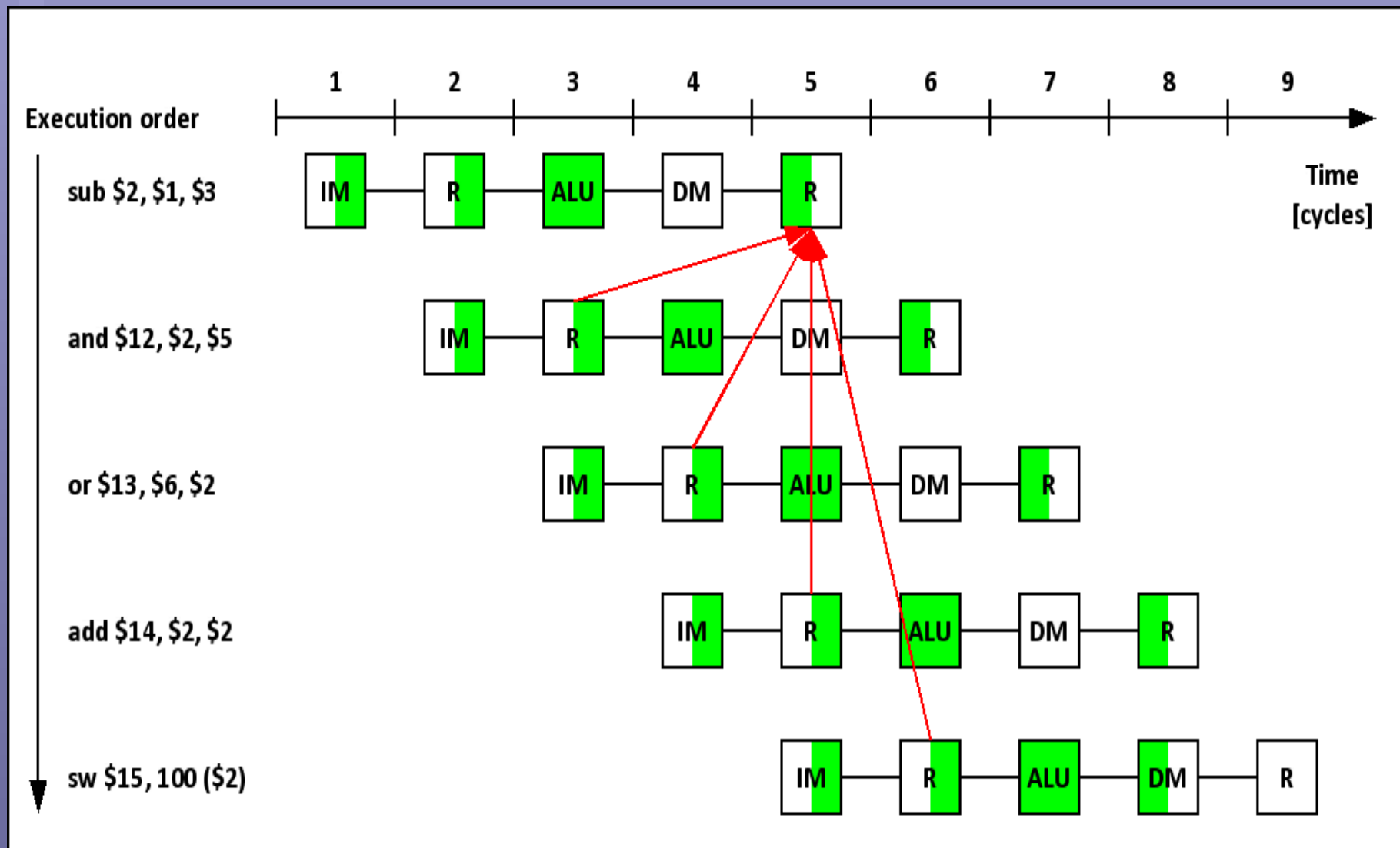
Forwarding/bypassing

- poskytnutí mezivýsledku následující instrukci
- nelze forwardovat zpátky v čase

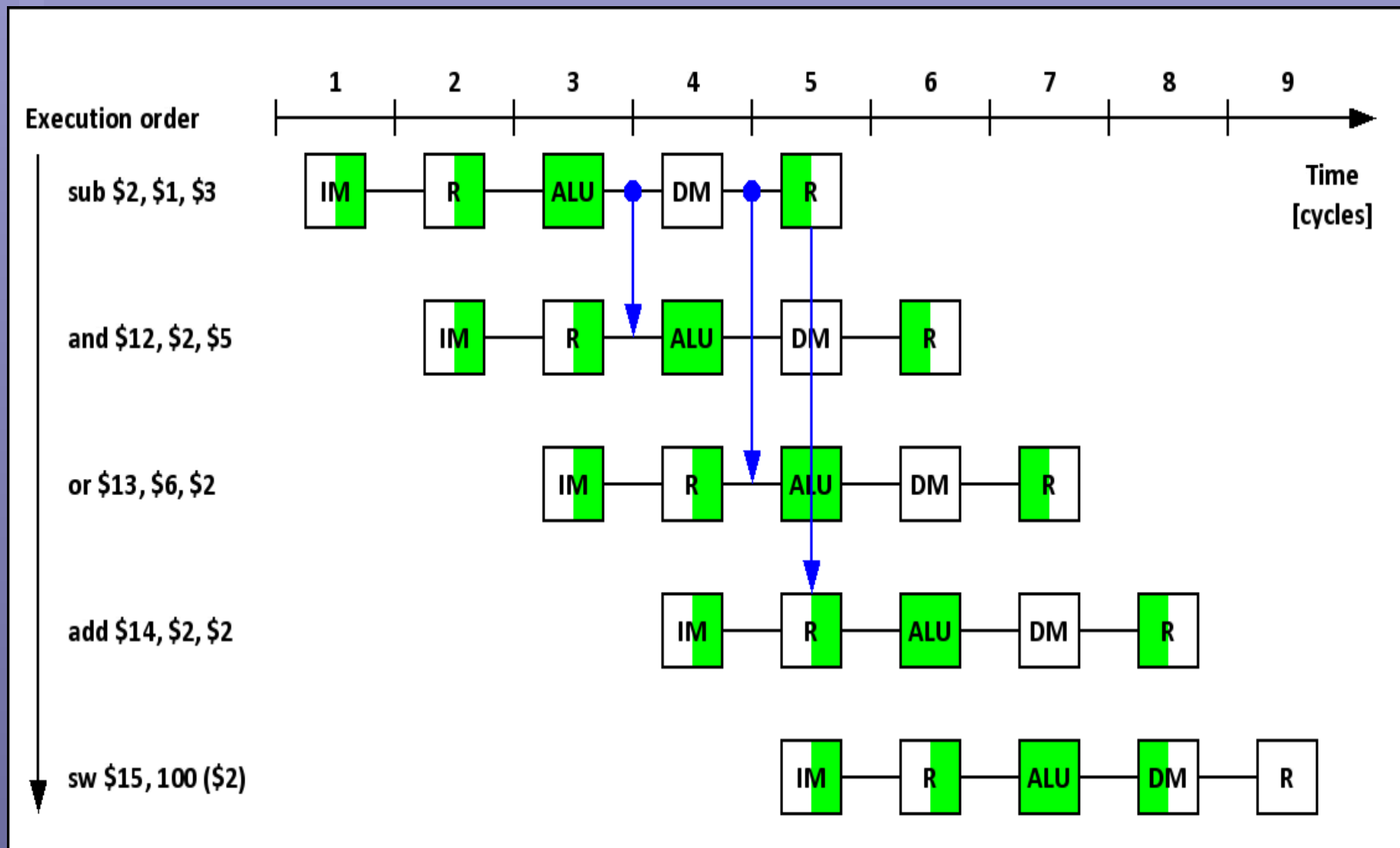
Forwarding unit

- zdrojový operand vykonávané instrukce je cílový operand výsledku dřívější instrukce
 - ♦ $EX/MEM.RegisterRd = ID/EX.RegisterRs$
 - ♦ $EX/MEM.RegisterRd = ID/EX.RegisterRt$
 - ♦ $MEM/WB.RegisterRd = ID/EX.RegisterRs$
 - ♦ $MEM/WB.RegisterRd = ID/EX.RegisterRt$

Forwarding v proudovém zpracování



Forwarding v proudovém zpracování



Datový hazard v proudovém zpracování (2)

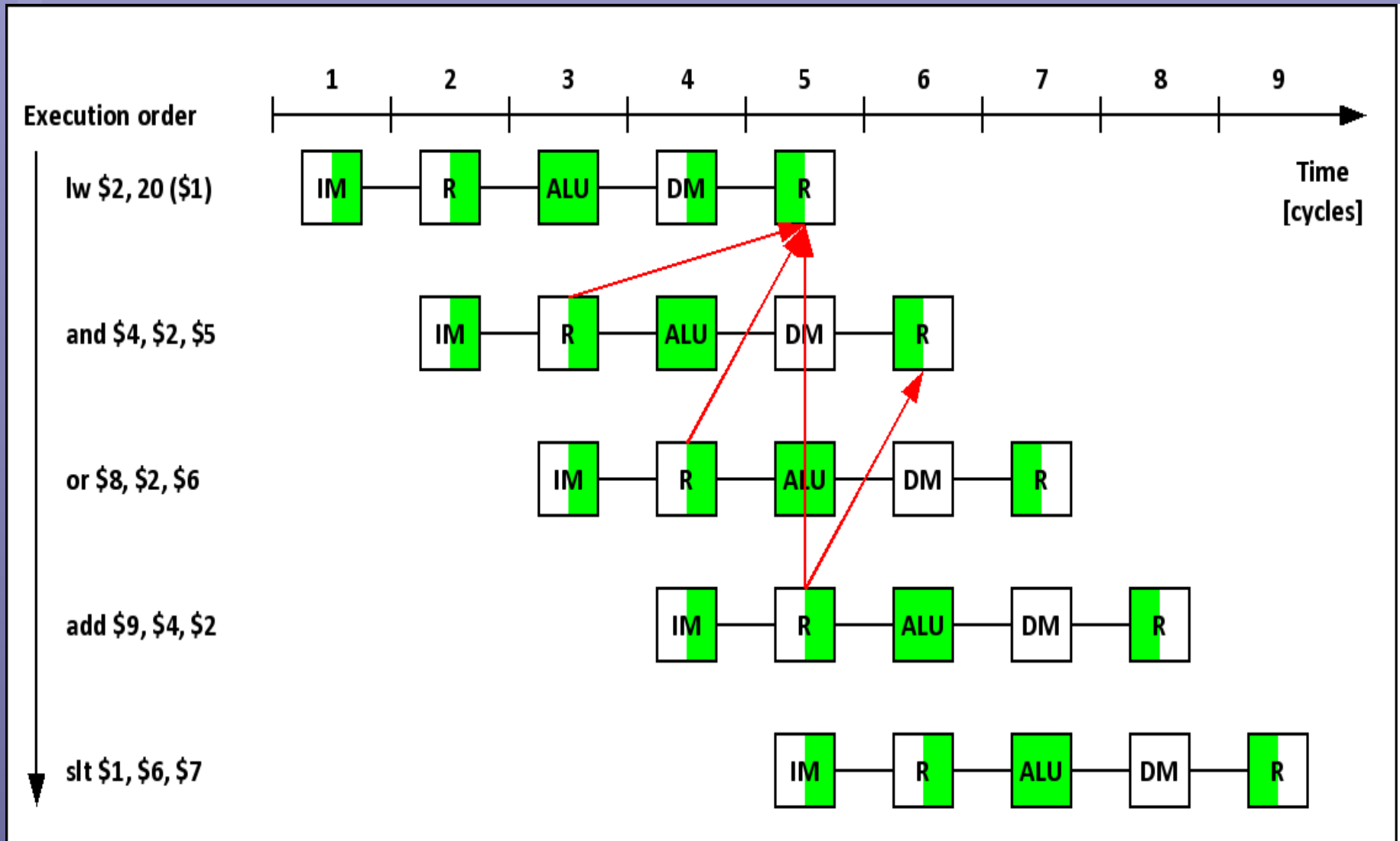
Zpoždění instrukce v pipeline

- použití operandu bezprostředně po načtení
 - ♦ load-use dependency

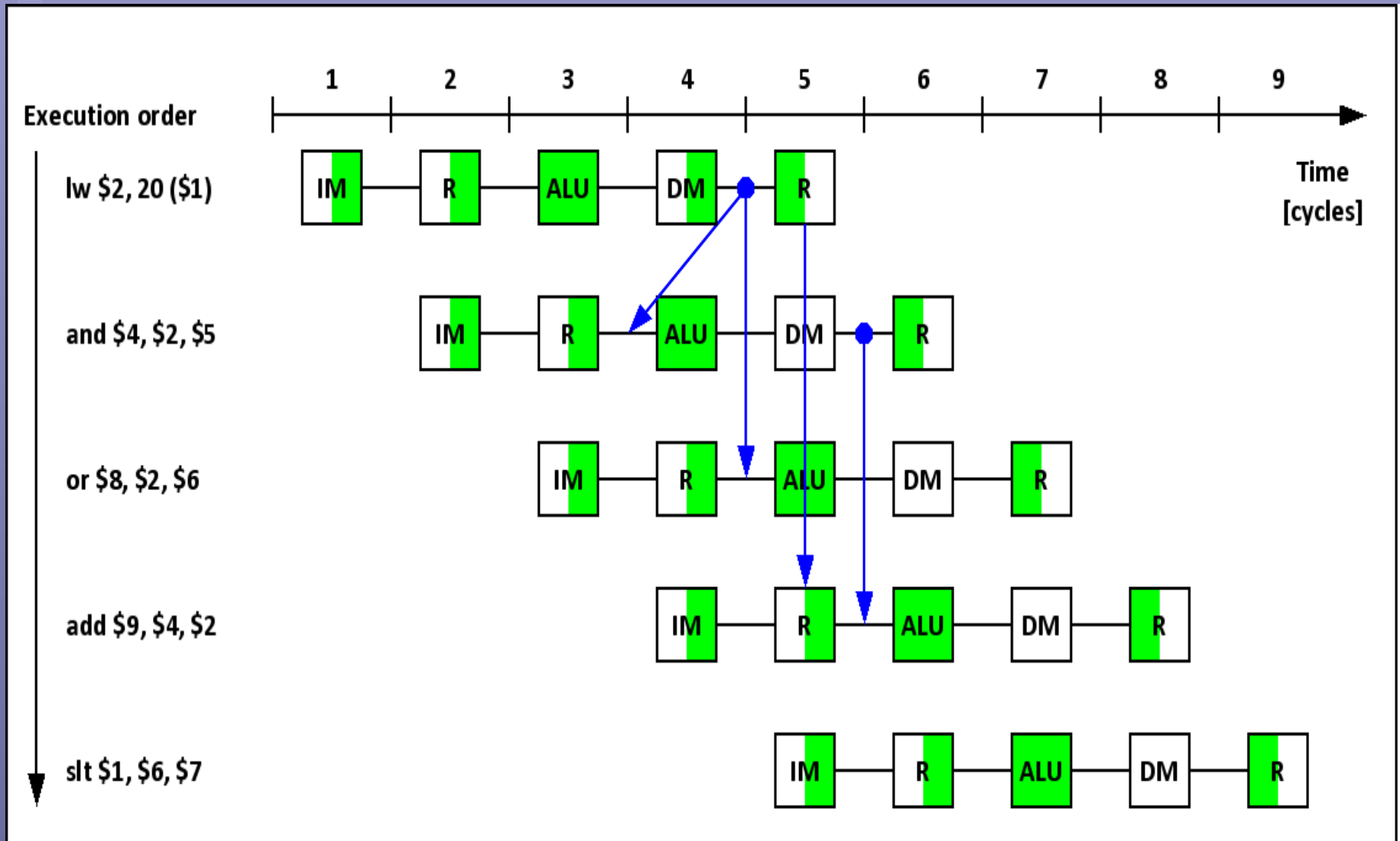
Hazard detection unit

- zdrojový operand dekódované instrukce je cílový operand dřívější instrukce čtení z paměti
 - ♦ $ID/EX.MemRead$ and
($ID/EX.RegisterRt = IF/ID.RegisterRs$ or
 $ID/EX.RegisterRt = IF/ID.RegisterRt$)

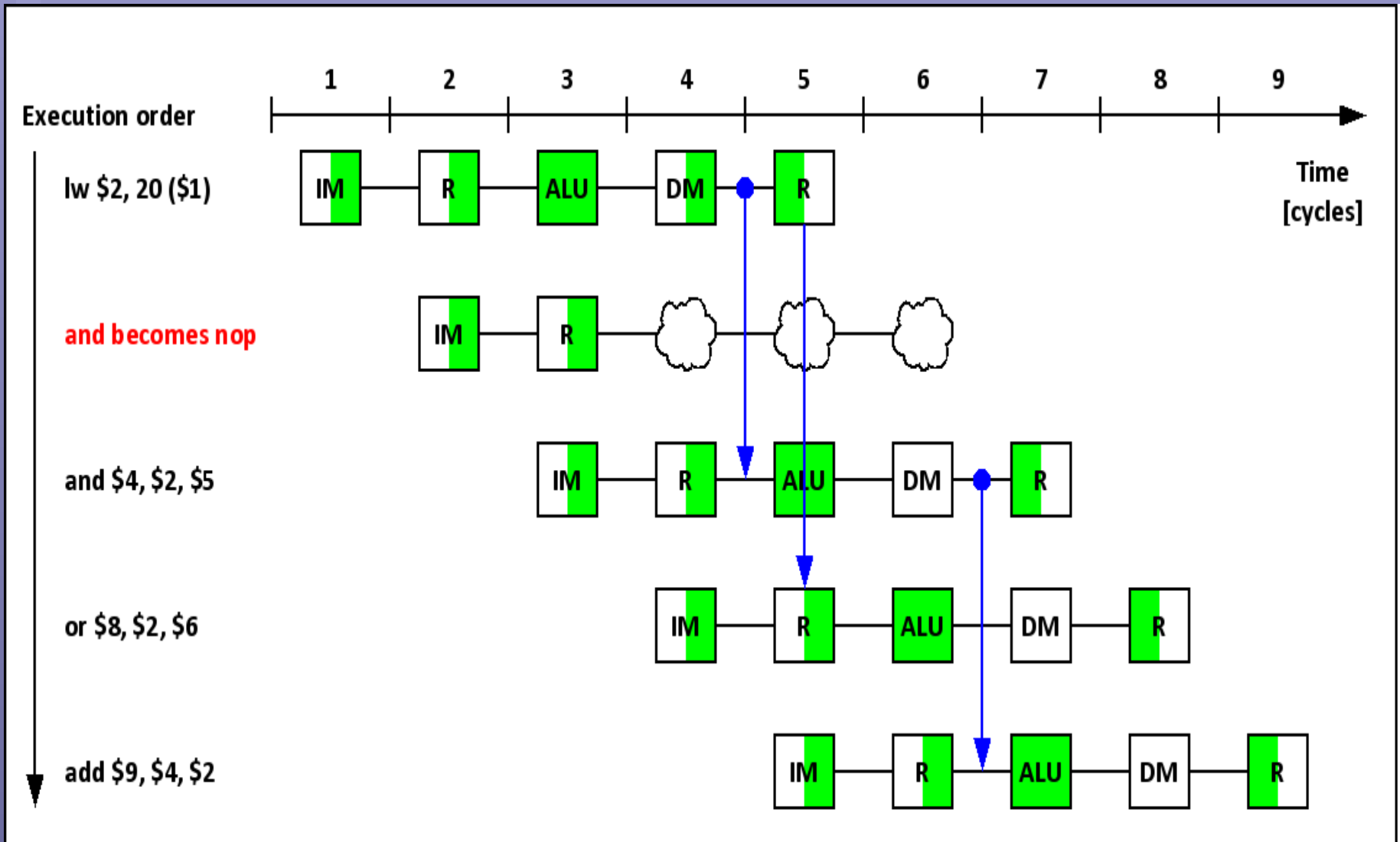
Zpoždění v proudovém zpracování (1)



Zpoždění v proudovém zpracování (1)



Zpoždění v proudovém zpracování (2)



Řídící hazard v proudovém zpracování (1)

Rozhodnutí o adrese další instrukce

- v pipeline jsou rozpracované instrukce
 - ♦ rozhodnutí závisí na výsledku
- hlavní typy řídicích hazardů
 - ♦ větvení
 - ♦ výjimky

Ošetření řídicích hazardů

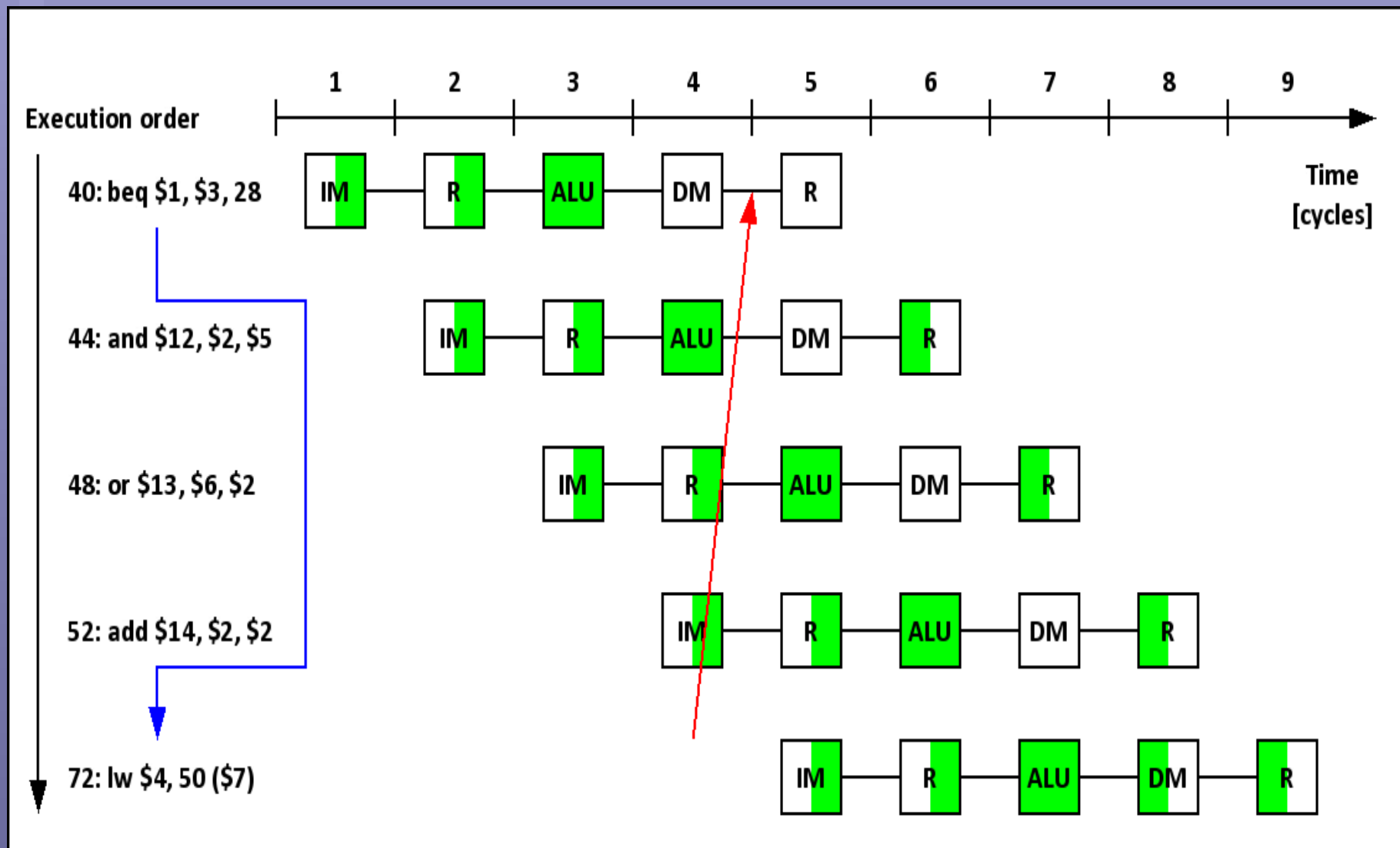
- forwarding nelze (adresa je, neví se zda použít)
- snaha o minimalizaci prodlev v pipeline

Řídící hazard v proudovém zpracování (2)

Řídící hazard při větvení

- snaha o udržení plné pipeline
 - ♦ problém: odkud číst další instrukce?
- zpozdit pipeline do dokončení skoku
 - ♦ horší situace nastat nemůže → prostor pro zlepšení
- předpokládat že skok se neprovede
 - ♦ pokud se skok provedl, vyprázdnit pipeline
 - ♦ redukce zpoždění při provedeném skoku
- dynamická predikce skoků
 - ♦ predikce na základě chování v minulosti

Řídící hazard při větvení (1)



Řídící hazard při větvení (2)

Statická predikce skoků

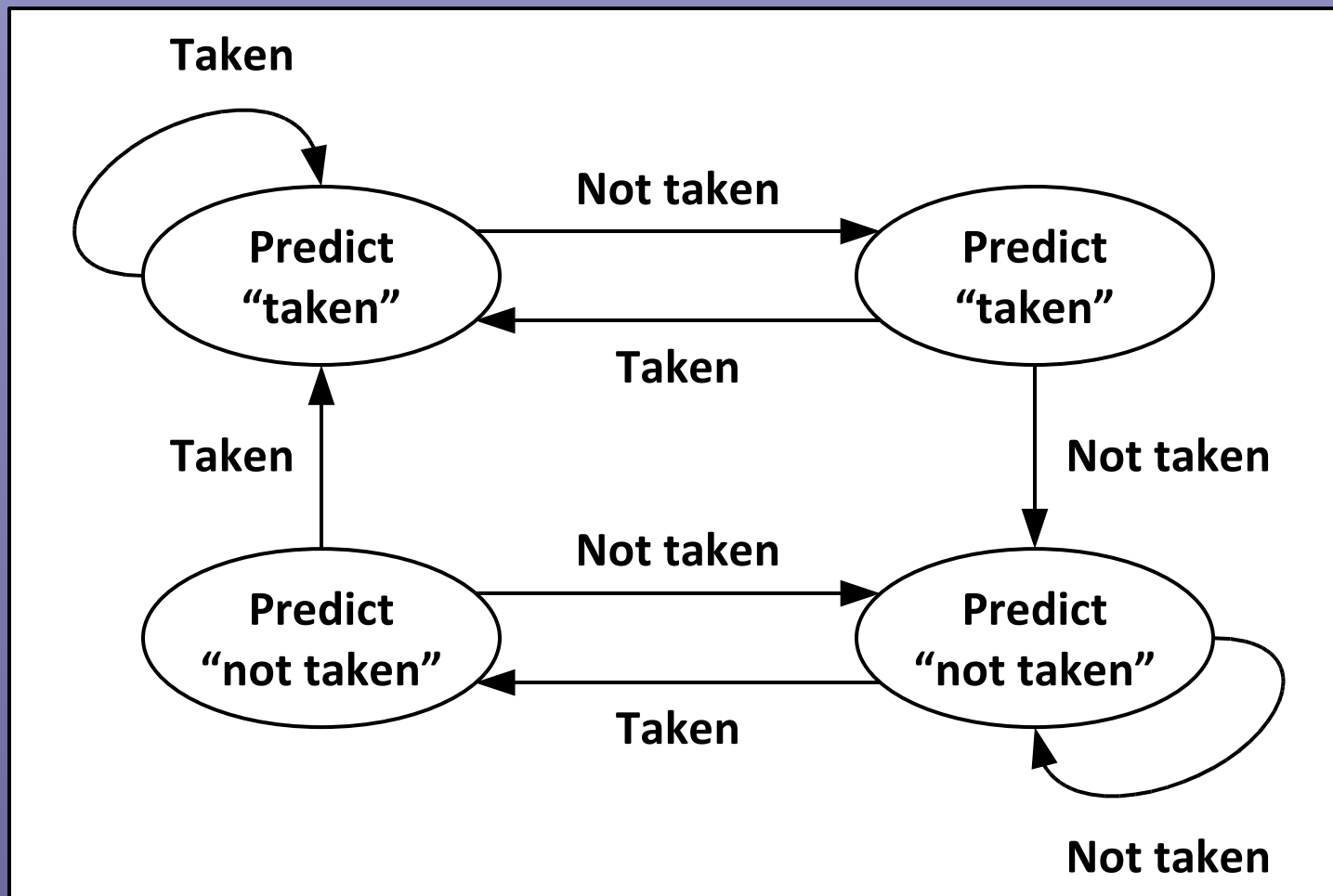
- predikce nezávisí na historii
- předpoklad určen HW, nebo bitem v instrukci

Dynamická predikce skoků

- predikce na základě historie
- branch prediction buffer/history table
 - ♦ predikce stejného chování jako v minulosti
 - ♦ cykly většinou skáčí na začátek → 2 špatné predikce
- 2-bitový prediktor
 - ♦ musí se 2x splést než změní “názor”

Řídící hazard při větvení (3)

2-bitový prediktor skoků (4 stavy)



Řídící hazard při větvení (4)

Redukce zpoždění při chybné predikci

- zrychlené vyhodnocení skoku (MIPS)
 - ♦ přesun výpočtu adresy skoku a vyhodnocení podmínky ze stupňů EX a MEM do stupně ID
 - ♦ podmínkou je test na rovnost (jednodchá realizace), ale vyžaduje forwardování mezivýsledků
 - ♦ zpoždění 1 cyklus při skoku
- zpožděný skok
 - ♦ vždy se vykoná 1 instrukce po instrukci skoku
- branch target buffer
 - ♦ informace o cílové adrese specifické instrukce

Řídící hazard v proudovém zpracování (3)

Řídící hazard při výjimce

- jako větvení: změna adresy pro čtení instrukcí
- důraz na zachování správného stavu procesoru
 - ♦ výsledky se nezapisují aby bylo možné lépe identifikovat příčinu výjimky
 - ♦ pokud to dává smysl, instrukce se provede znovu
- více instrukcí v pipeline → více zdrojů výjimek
 - ♦ přiřazení výjimky ke správné instrukci
 - ♦ prioritizace, zachycení všech výjimek v cyklu

Zvyšování instrukčního paralelismu

Prodloužení pipeline

- jednodušší stupně → kratší cyklus, *deep pipeline*

Zpracování více instrukcí v každé fázi pipeline

- replikace prostředků, *multiple issue pipeline*

Hlavní problémy *multiple issue pipeline*

- plnění instrukčních slotů (issue slots)
 - ♦ ne všechny instrukce lze provádět paralelně
- ošetření datových a řídicích hazardů

Hlavní problémy multiple issue pipeline (1)

Plnění instrukčních slotů

- kolik/které instrukce lze vykonat současně?

Static multiple issue

- instrukce plánuje především/výhradně překladač

Dynamic multiple issue

- instrukce ke zpracování vybírá procesor
- překladač dodá “rozumné” implicitní pořadí

Hlavní problémy multiple issue pipeline (2)

Ošetření datových a řídících hazardů

- datové závislosti, větvení, ...

Static multiple issue

- většina/všechny důsledky hazardů řeší překladač

Dynamic multiple issue

- procesor se snaží omezit negativní vliv hazardů
- překladač dodá “rozumný” výchozí kód
 - ♦ ne všechny hazardy je možné odstranit při překladu

Spekulativní provádění instrukcí

Odhadnutí vlastností/výsledku instrukce

- umožňuje zahájit zpracování závislých instrukcí
- nutno zajistit vždy korektní výsledek

Spekulace při překladu

- speciální kód pro opravu chybných spekulací

Spekulace v procesoru

- spekulativní výsledky kumulovány v procesoru

Problém s výjimkami

- nevyvolávat dokud jsou spekulativní

Static multiple issue

Issue packet

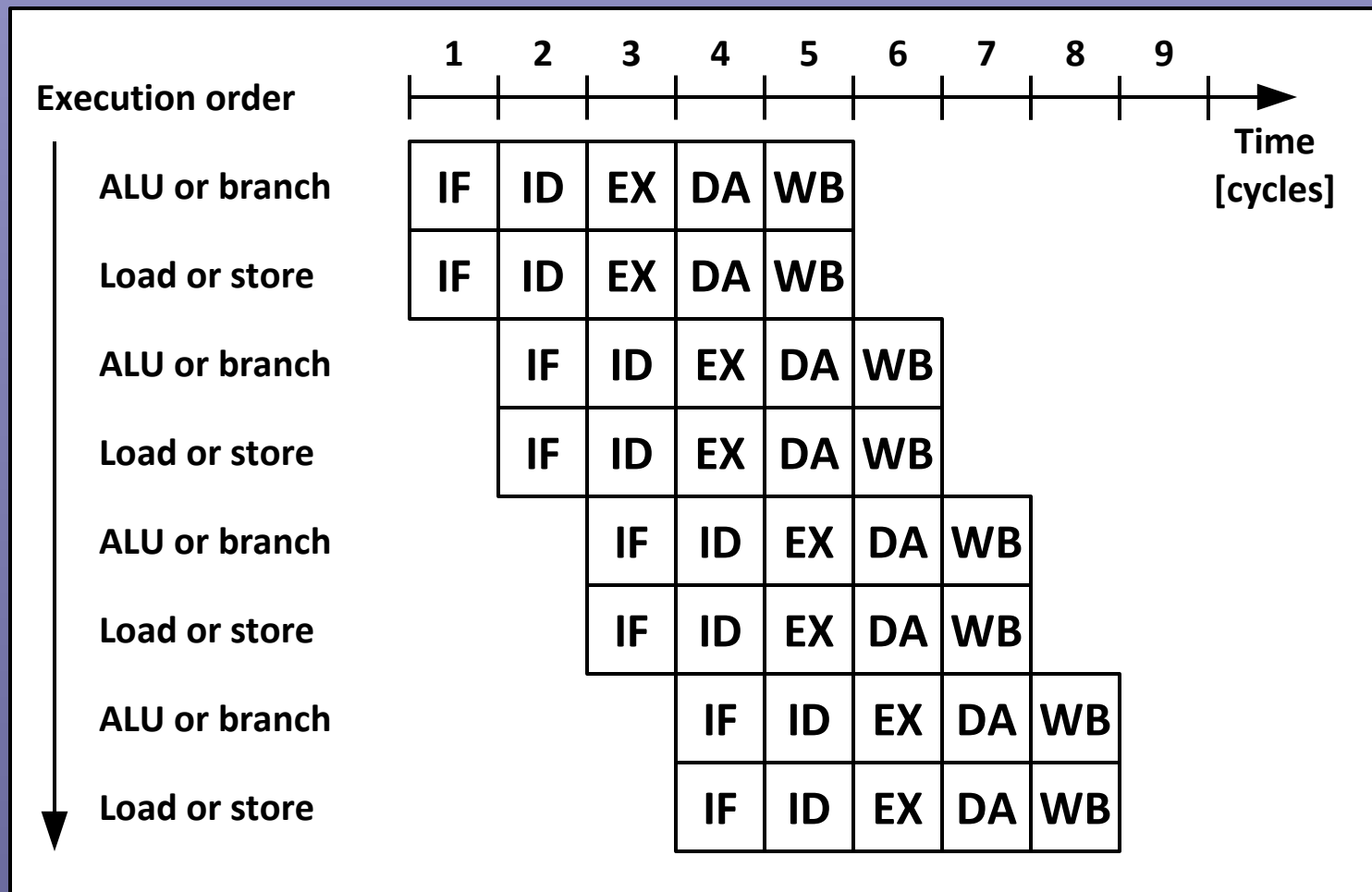
- sada instrukcí, které se mají provést zároveň
- sloty v issue paketu nejsou ortogonální
 - ♦ Very Long Instruction Word (VLIW)
 - ♦ Explicit Parallel Instruction Computer (EPIC)

Primární role překladače

- statická predikce skoků
- omezení/odstranění datových hazardů

Příklad: multiple issue MIPS ISA (1)

Vykonávání instrukcí v pipeline



Příklad: multiple issue MIPS ISA (2)

Změny oproti single issue

- načítání 64-bit instrukcí, zarovnání na 8 bajtů
 - ♦ nevyužitý slot obsahuje “nop”
- registrové pole pro přístup z obou instrukcí
- samostatná sčítačka pro výpočet adresy v paměti

Nevýhody

- vyšší latence při použití výsledků
 - ♦ složitější plánování instrukcí překladačem
- prostoje v důsledku hazardů jsou “dražší”

Příklad: architektura IA-64 (1)

Hlavní rysy IA-64

- mnoho registrů
 - ♦ 128 GP, 128 FP, 8 branch, 64 condition
 - ♦ registrová okna s podporou přetečení do paměti
- instruction bundle
 - ♦ svazek instrukcí vykonávaný současně
 - ♦ pevný formát, explicitní závislosti
- podpora spekulace a eliminace větvení
 - ♦ lepší využití instrukčního paralelismu dražší

Příklad: architektura IA-64 (2)

Zajímavé vlastnosti

- instruction group
 - ♦ skupina instrukcí bez datových závislostí
 - ♦ skupiny odděleny speciálním indikátorem **stop**
- struktura instrukčního svazku
 - ♦ 5 bitů template (použité výkonné jednotky)
 - ♦ 3 x 41 bitů instrukce
- predikace instrukcí (predication)
 - ♦ většina instrukcí může záviset na podmínkovém registru
 - ♦ 6 bitů výběr 1 z 64 podmínkových registrů

Dynamic multiple issue (1)

Superskalární procesory

- procesor vybírá instrukce ke zpracování
- uspořádání instrukcí ovlivňuje pouze výkon

In-order execution

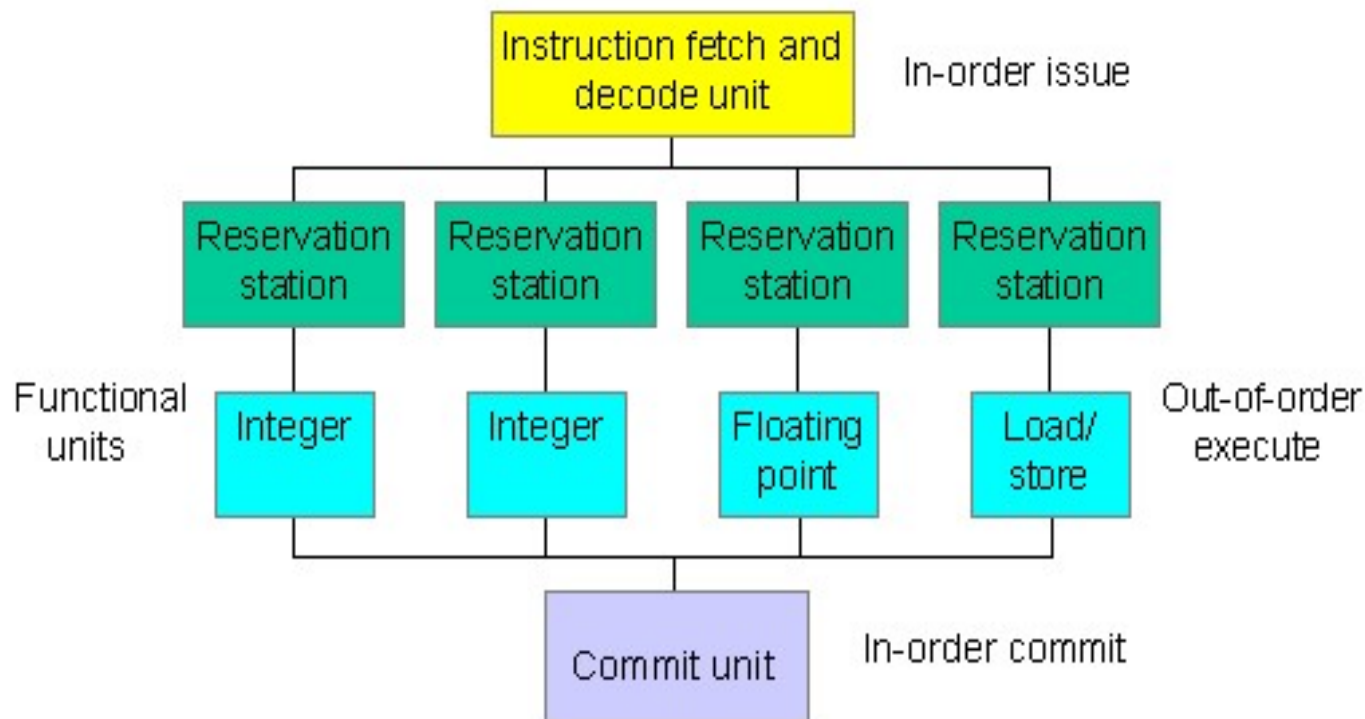
- překladač omezuje datové závislosti

Out-of-order execution

- dynamické plánování instrukcí v pipeline
 - ♦ cílem je eliminovat hazardy a prostoje
- překladač se snaží usnadnit plánování

Dynamic multiple issue (2)

Základní uspořádání



Dynamic multiple issue (3)

Datové závislosti při přerovnání

- True data dependency – RAW (Read after Write)
 - ♦ výstup instrukce je použit jako vstup následující
- Output dependency – WAW (Write after Write)
 - ♦ dvě instrukce zapisují na stejné místo
- Anti-dependency – WAR (Write after Read)
 - ♦ zatímco jedna instrukce zpracovává data, další instrukce tato data změní
- WAW a WAR lze vyřešit přeznačením

Dynamic multiple issue (4)

Eliminate WAW

MOVE r3, r7

ADD r3, r4, r5

MOVE r1, r3

...

MOVE r3, r7

ADD r8, r4, r5

MOVE r1, r8

...

Dynamic multiple issue (5)

Příklad: out-of-order execution

01	LOAD R2,A
02	ADD R1,R2,R3
03	BPOS R1,LAB1 (Taken)
04	LOAD R4,B
05	BNEG R4,LAB2
06	LAB1: LOAD R4,C
07	ADD R5,R4,R3
08	LAB2: SUB R5,R7,R0
09	BPOS R5,LAB3 (NOT Taken)
10	ADD R5,R0,R3

01	LOAD R2,A
06	LAB1: LOAD R4,C
08	LAB2: SUB R5,R7,R0
10	ADD R5,R0,R3
02	ADD R1,R2,R3
07	ADD R5,R4,R3
09	BPOS R5,LAB3 (NOT Taken)
03	BPOS R1,LAB1 (Taken)

Dynamic multiple issue (6)

Výjimky

- při výskytu výjimky musí procesor „zastavit činnost“ na **tomto** místě
- následující instrukce nesmí ovlivnit stav stroje
- nesmí existovat nezpracované instrukce před **touto** instrukcí
- všechny výjimky způsobené předchozími instrukcemi jsou vyřízeny
- precizní přerušení/výjimka
 - ♦ vždy spojeno se správnou instrukcí

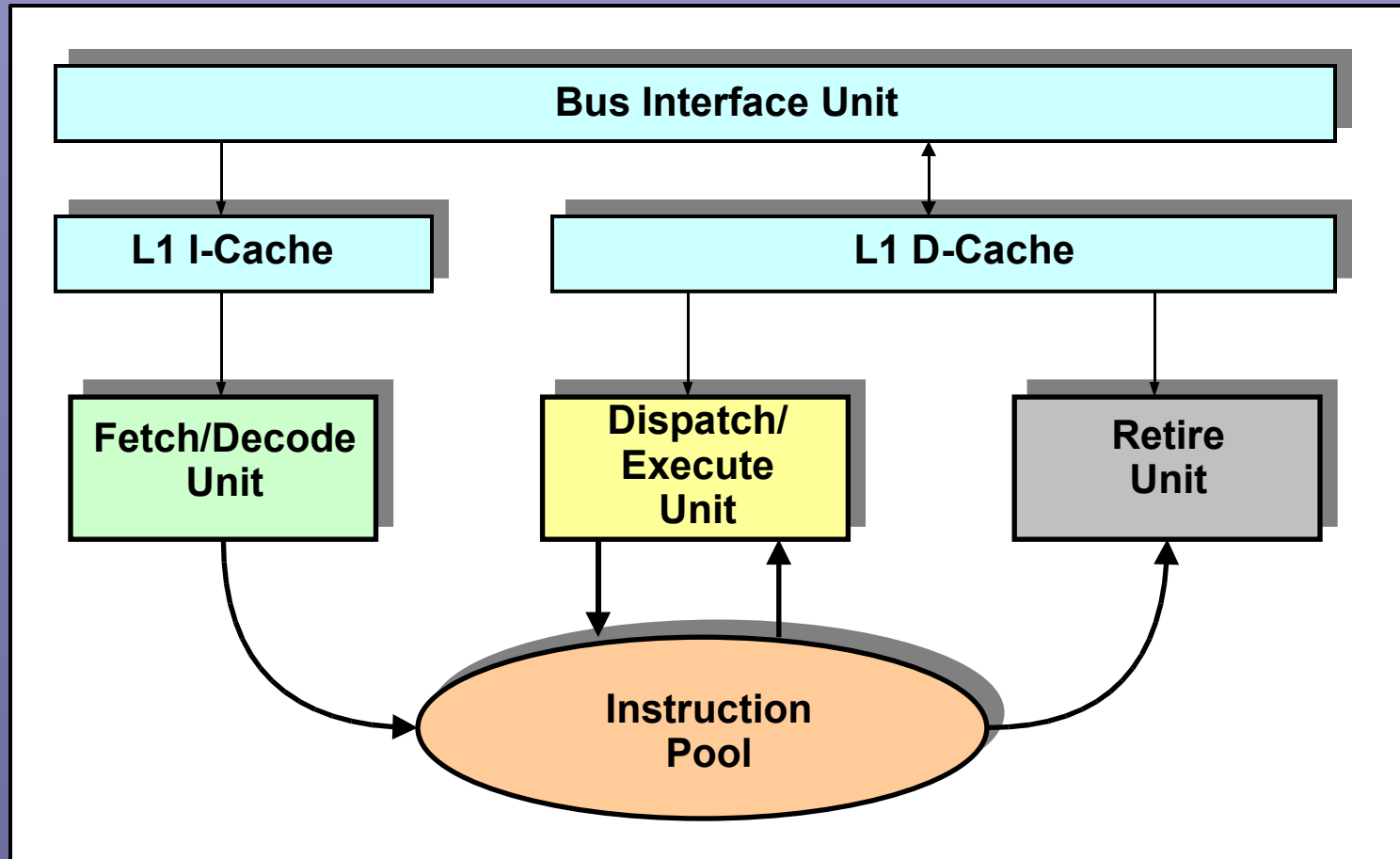
Příklad: architektura IA-32 (1)

Intel Pentium Pro až Pentium 4

- instrukční sada CISC interpretována mikrooperacemi na jádře post-RISC
- instrukce rozkládány na mikroinstrukce
- pipeline provádí mikroinstrukce
- superskalární, spekulativní provádění instrukcí

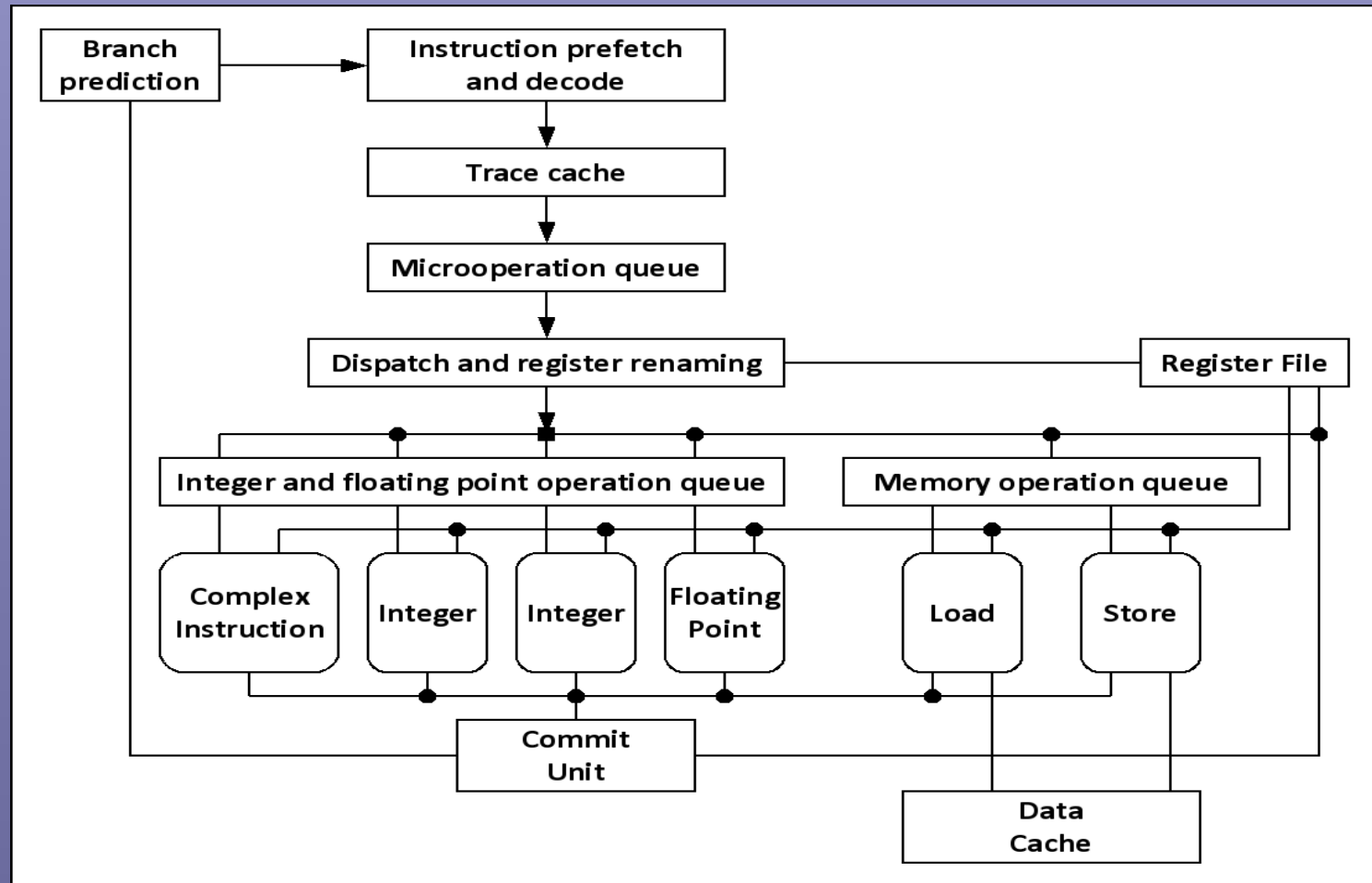
Příklad: architektura IA-32 (2)

Základní struktura



Příklad: architektura IA-32 (3)

Intel Pentium 4



Příklad: architektura IA-32 (4)

Intel Pentium 4 vs. Pentium III

- dvojnásobná délka pipeline (cca 20 vs 10 stupňů)
- více funkčních jednotek (7 vs 5)
- podpora více rozpracovaných operací (126 vs 40)
- trace cache (urychluje dekodování instrukce)
- lepší prediktor skoků (4K položek vs 512)
- vylepšený paměťový subsystém

Literatura

D. A. Patterson, J. L. Hennessy

- Computer Organization and Design