Architektura počítačů
Zvyšování výkonnosti

http://d3s.mff.cuni.cz/teaching/computer_architecture/

Lubomír Bulej

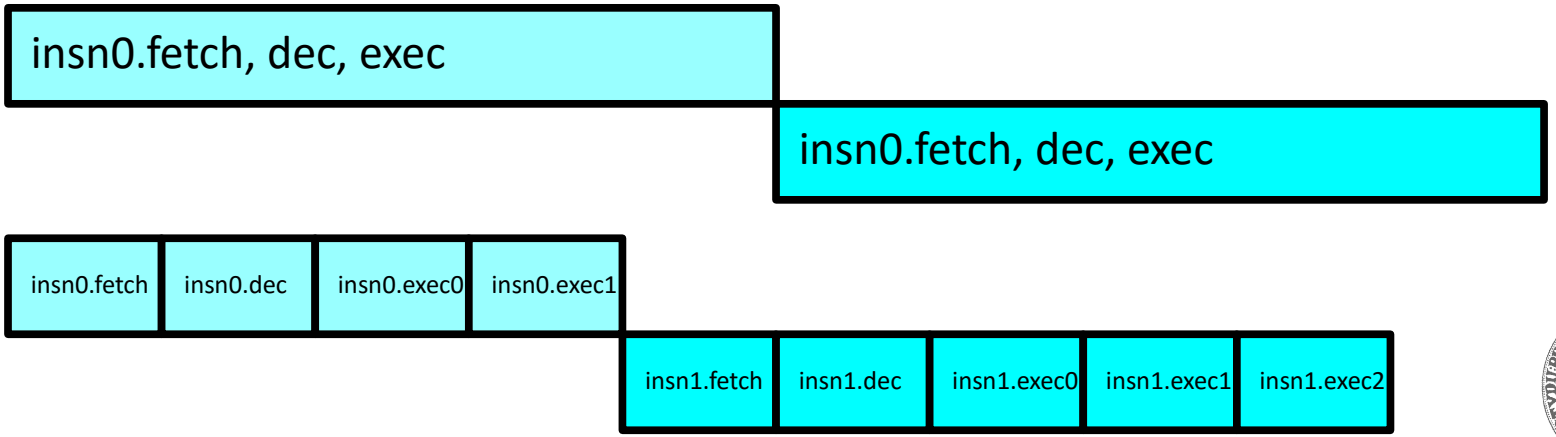
bulej@d3s.mff.cuni.cz

CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

Faktory omezující výkon procesoru

- **Délka strojového cyklu**
  - Omezení délkou nejsložitější fáze nejsložitější instrukce
  - **Zrychlení:** Přechod od jednocyklové datové cesty k vícecyklové datové cestě
    - Jednoduché instrukce lze provést rychleji než složitější instrukce

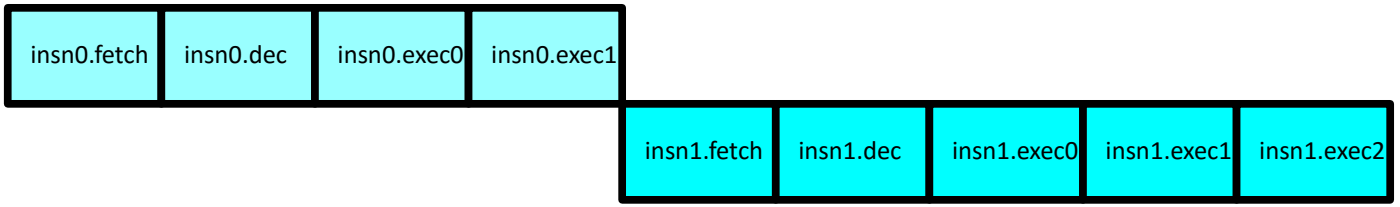


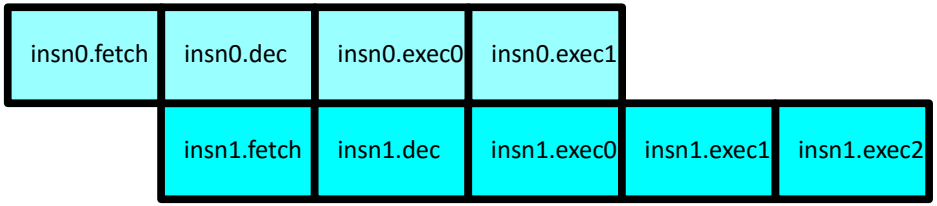
Faktory omezující výkon procesoru (2)

- ***Clock per instruction* (CPI)**
  - Omezení počtem zpracovávaných instrukcí v jednom okamžiku
    - I vícecyklová datová cesta v každý okamžik zpracovává pouze jednu instrukci
  - Latence vs. propustnost
    - **Latence** jedné instrukce je odvozena od délky strojového cyklu (tuto délku nemůžeme zkracovat neomezeně)
    - **Propustnost** posloupnosti instrukcí (celého programu) zvýšíme paralelním zpracováním více instrukcí současně

Zřetězené zpracování instrukcí

- ***Pipelining***
  - Při zahájení 2. kroku instrukce zahájí datová cesta současně 1. krok následující instrukce
  - Paralelismus na úrovni instrukčních kroků (zachování sekvenčního modelu)
  - Latence (doba zpracování) jednotlivých instrukcí se nemění, zvyšuje se propustnost





Výkon zřetězeného procesoru

- **Hrubý odhad**
  - Zpracování *n* instrukcí, délka cyklu *t*, *k* kroků na instrukci

$$T = n \cdot (k \cdot t)$$

  - Zřetězené zpracování s *k*-stupňovou pipeline
    - První instrukce opustí pipeline po *k* taktech, každá další po 1 taktu

$$T_p = k \cdot t + (n-1) \cdot t$$

  - Zrychlení

$$Speedup = \frac{T}{T_p} = \frac{n \cdot (k \cdot t)}{k \cdot t + (n-1) \cdot t} = \frac{n \cdot k}{k + (n-1)}$$

  - Zrychlení pro *n* >> *k*

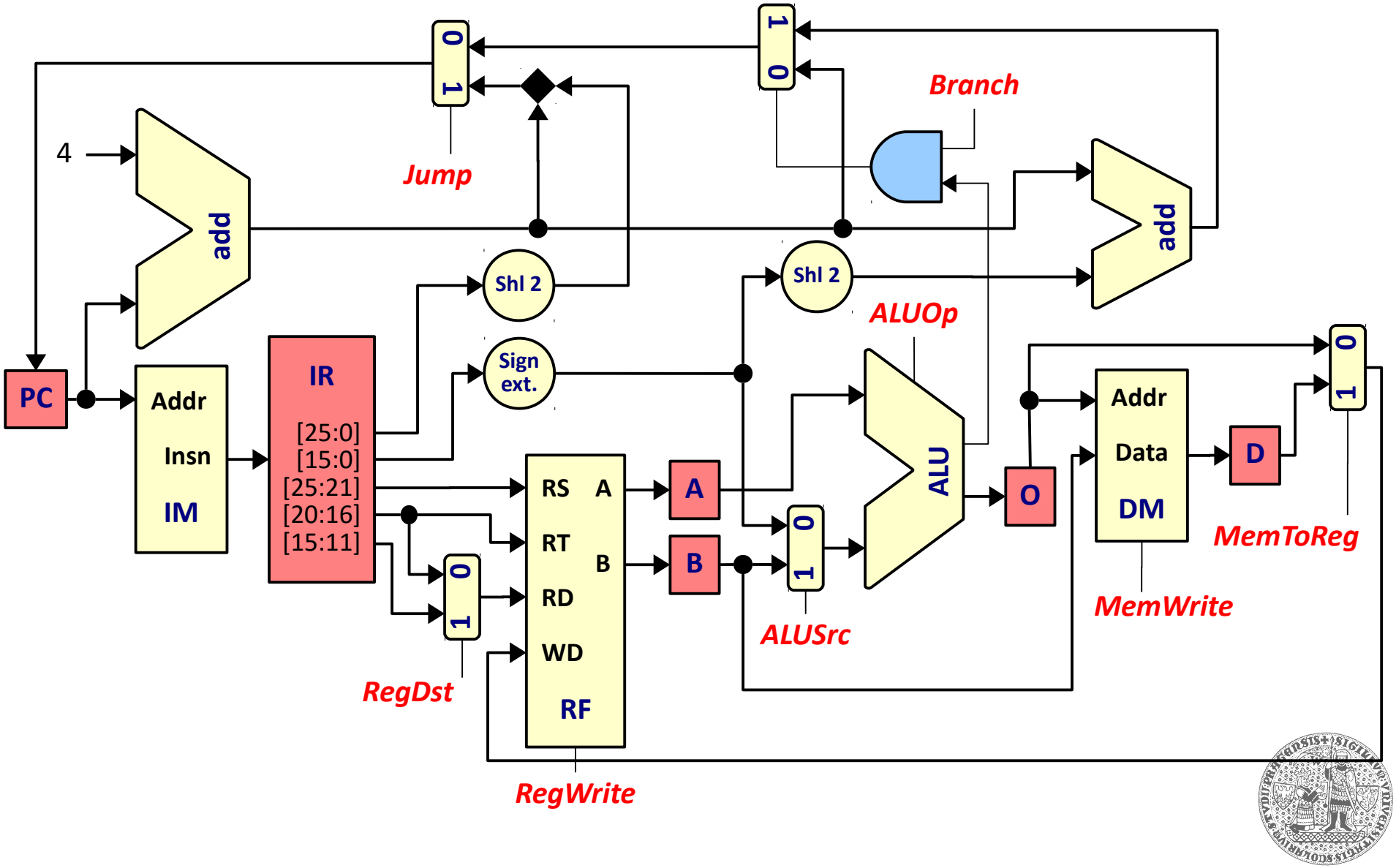
$$k + (n-1) \approx n$$

$$Speedup \rightarrow k$$

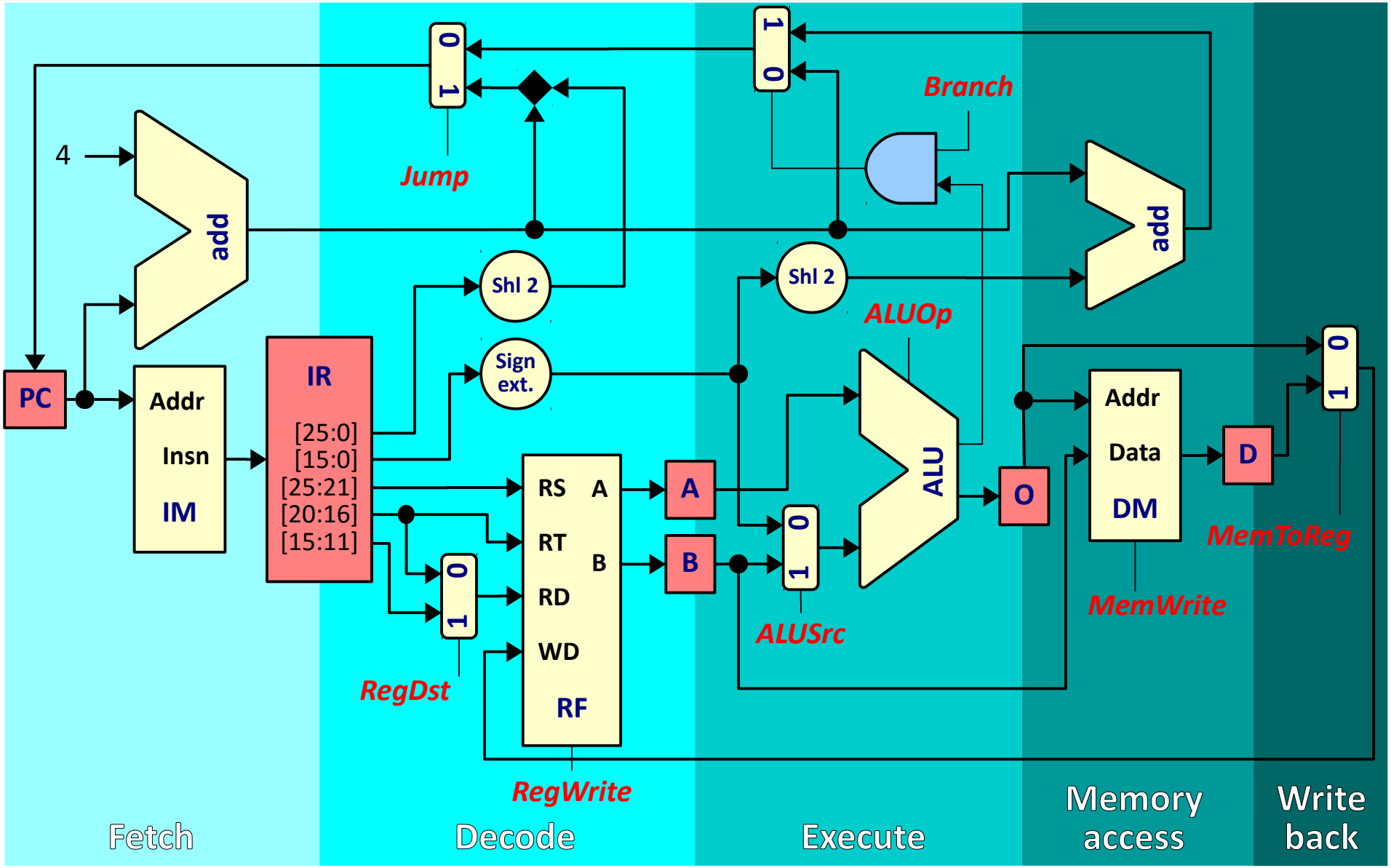
Datová cesta pro zřetězené zpracování

- **Základní myšlenka**
  - Základem jednocyklová datová cesta
    - Oddělená paměť pro instrukce a data
    - Pomocné sčítačky (ALU se nesdílí)
  - Prvky vícecyklové datové cesty
    - Zpracování instrukce v oddělených krocích
    - Registry pro mezivýsledky jednotlivých kroků (výstupy z instrukční a datové paměti, registrového pole a ALU)

Datová cesta pro zřetězené zpracování (2)



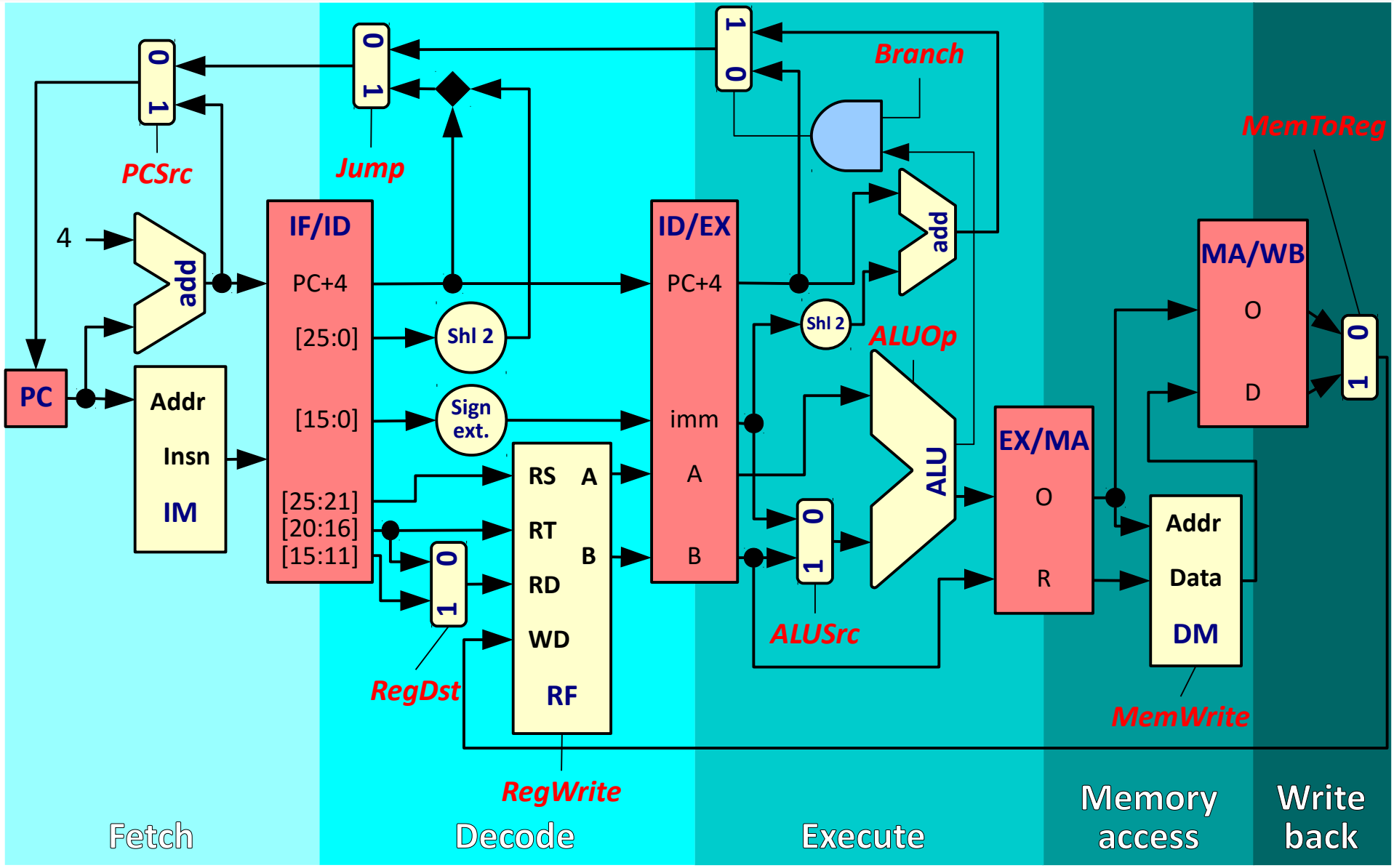
Datová cesta pro zřetězené zpracování (3)



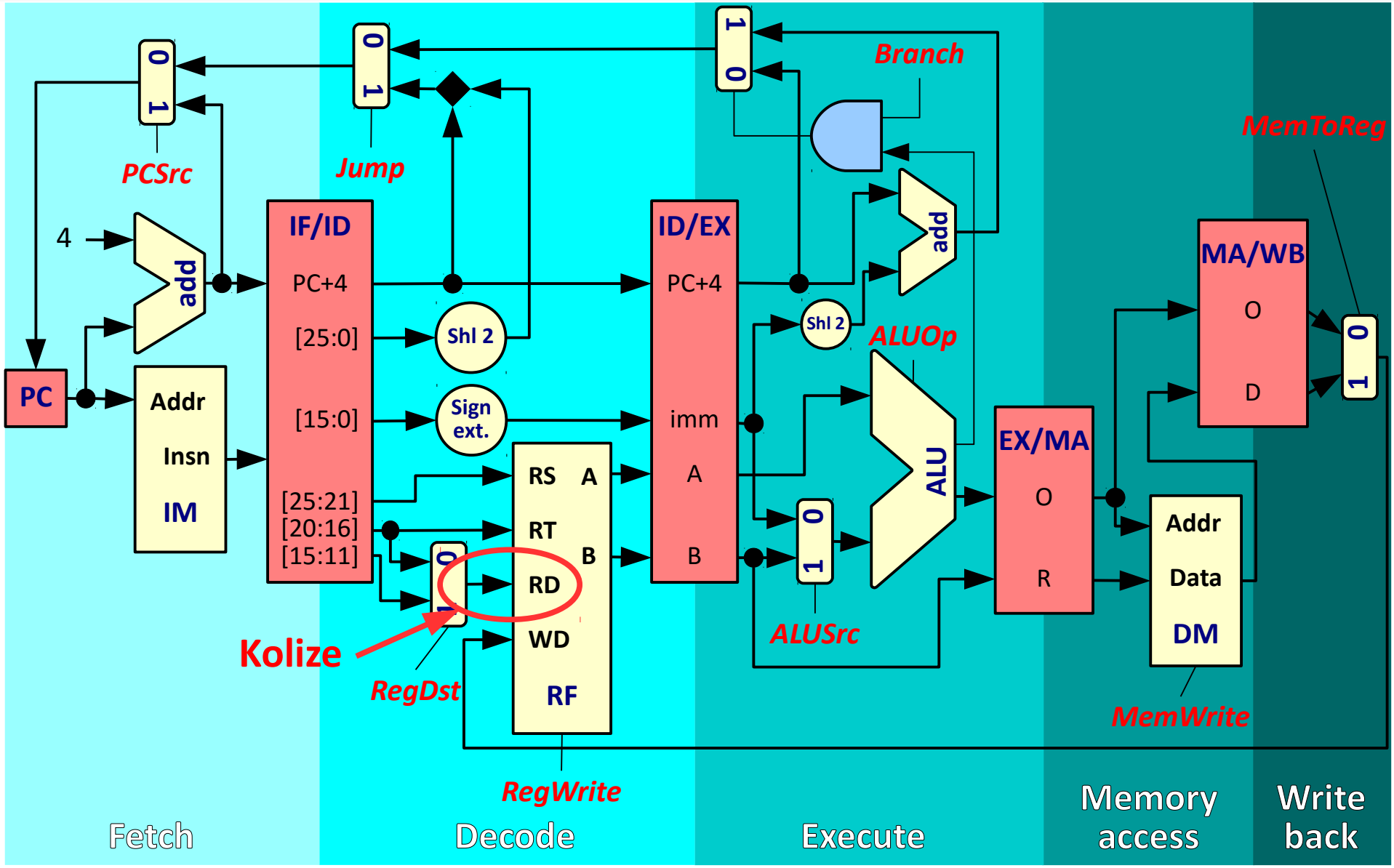
Datová cesta pro zřetězené zpracování (4)

- **Datová cesta rozdělena do *k* úseků (stupňů, *stages*)**
  - V každém úseku jiná instrukce
    - Nejpomalejší úsek určuje rychlost pipeline
    - Mezi jednotlivými úseky registry (latches)
      - Stav instrukce, operandy, řídící signály
      - Instrukce uvnitř datové cesty jsou v různých stavech
  - **Ideální stav:** CPI = 1
    - V každém hodinovém cyklu opustí pipeline jedna instrukce
      - Latence instrukcí zvyšuje režii, ne propustnost
    - Reálně: CPI > 1 (zpoždění pipeline)

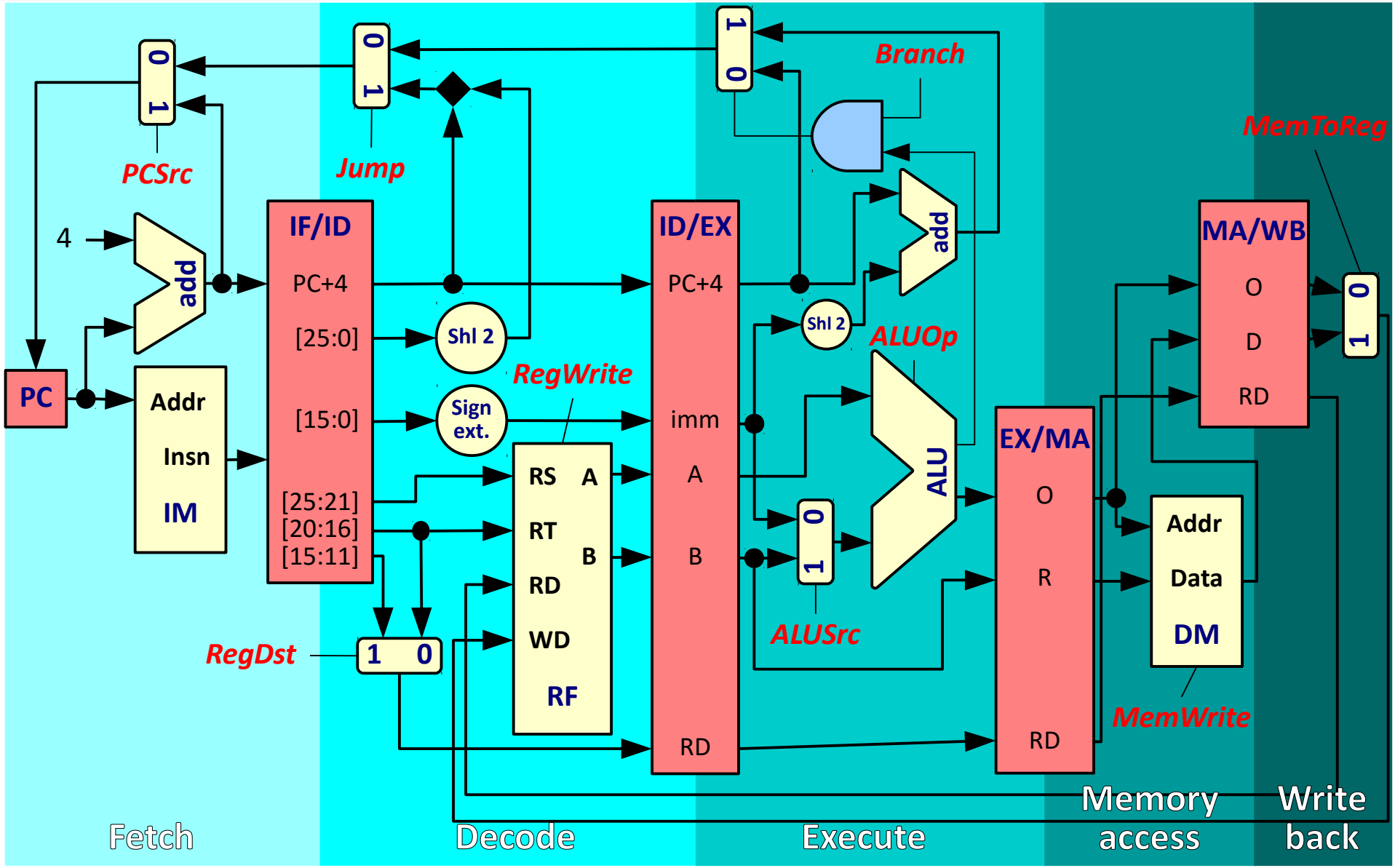
Datová cesta pro zřetězené zpracování (5)



Datová cesta pro zřetězené zpracování (6)



Datová cesta pro zřetězené zpracování (7)



Terminologie

- **Skalární pipeline (*scalar pipeline*)**
  - V každém úseku se zpracovává pouze 1 instrukce
- **Superskalární pipeline (*superscalar pipeline*)**
  - V některých úsecích (ne nutně všech) lze zpracovávat více instrukcí současně (ne nutně všechny kombinace)
    - Více ALU, složitější řízení
    - Více pipeline vedle sebe, které sdílejí některé zdroje
      - Pipeline U a V u původního Pentia

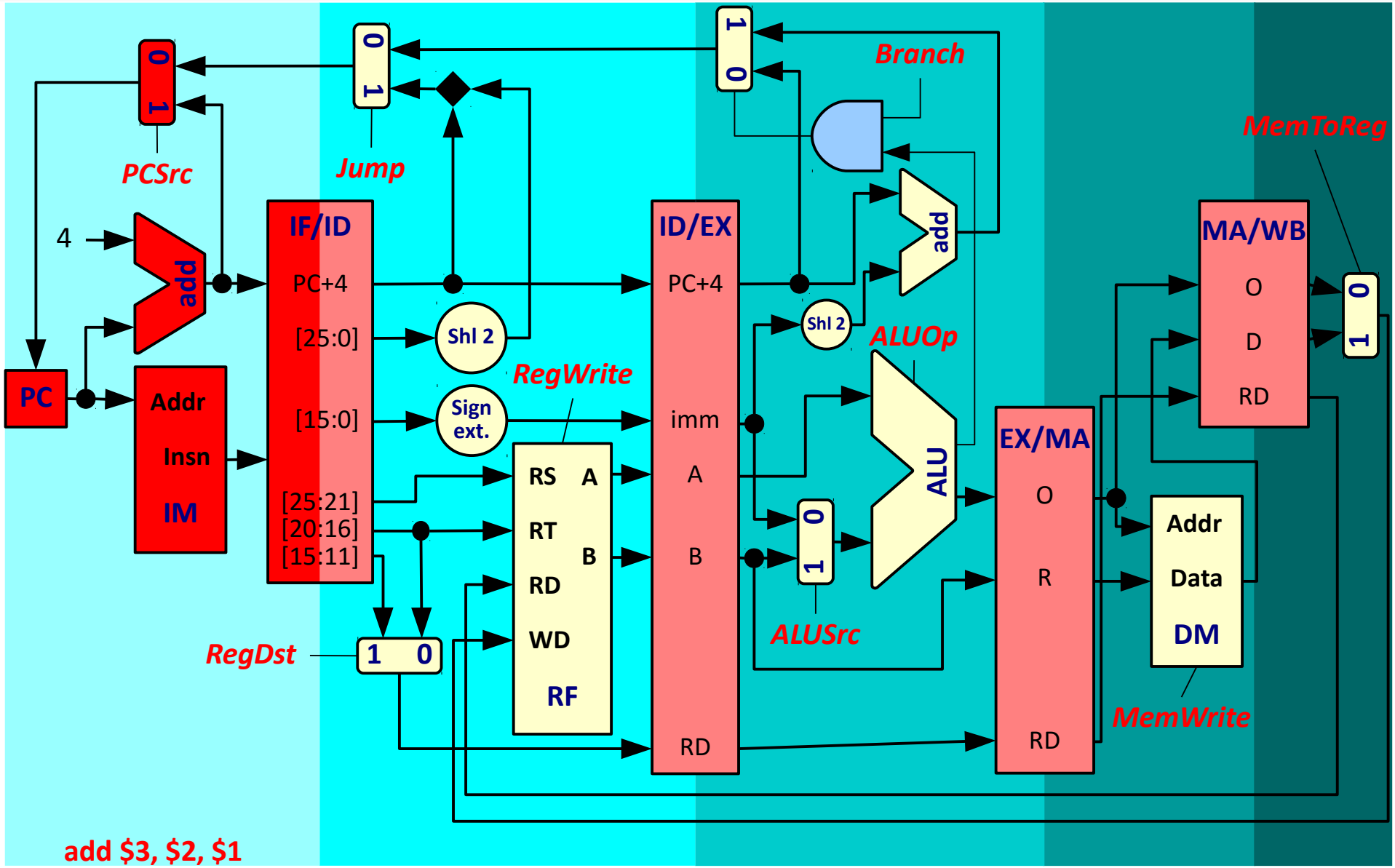
Terminologie (2)

- **Zpracování podle pořadí (*in-order pipeline*)**
  - Instrukce vstupují do pipeline podle pořadí v paměti
- **Zpracování mimo pořadí (*out-of-order pipeline*)**
  - Pipeline může prohazovat pořadí provádění instrukcí
  - Typické pro superskalární pipeline
    - Cílem je vždy využít všechny ALU, které jsou k dispozici
    - Předdekódování instrukcí pro zjištění jejich typu

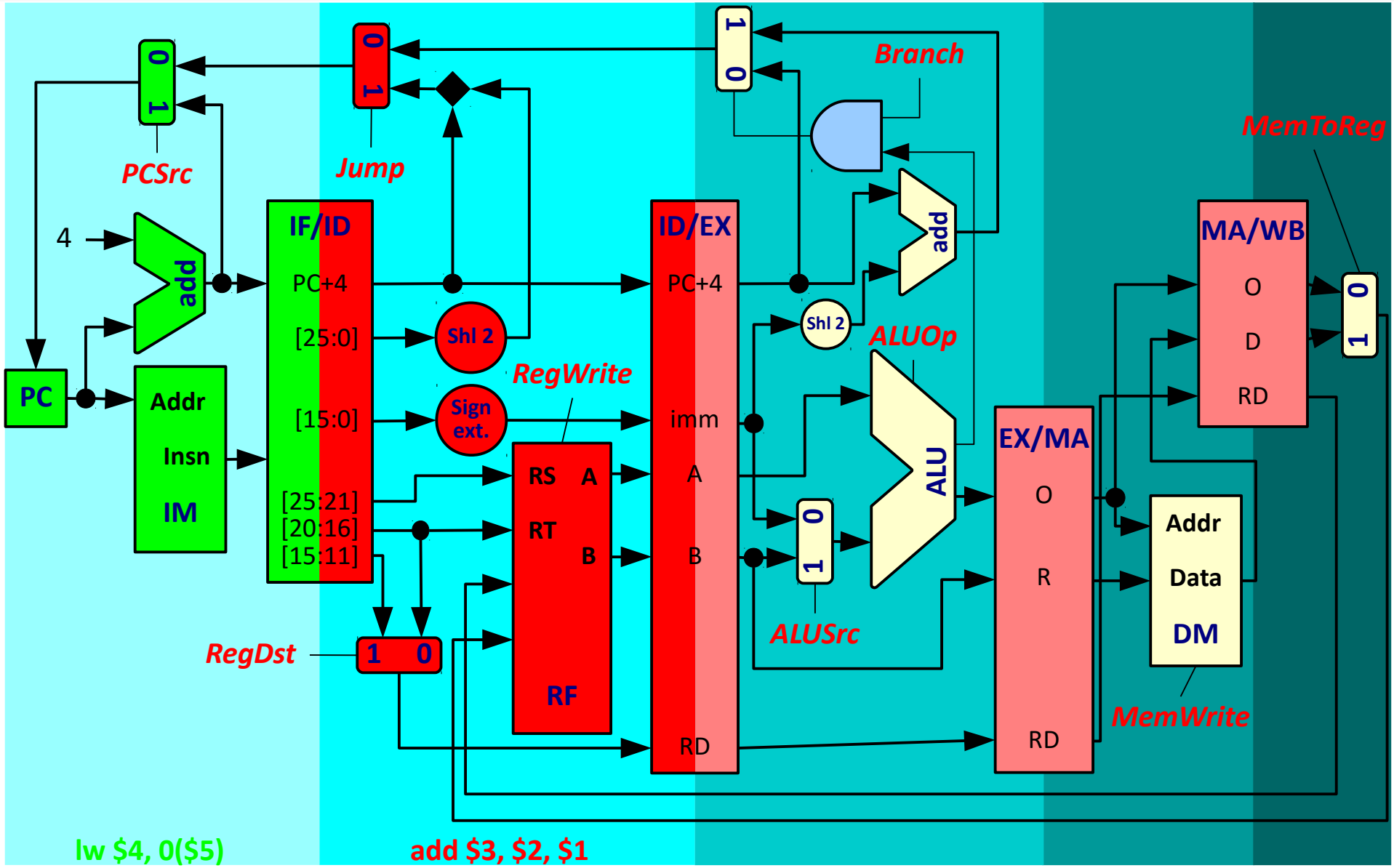
Terminologie (3)

- **Délka/hloubka pipeline, počet stupňů pipeline (*pipeline depth*)**
  - Počet úseků (*stages*) pipeline
  - U jednodušších RISCových procesorů bývá počet úseků odvozen od logických kroků instrukce
  - U složitějších (superskalárních, out-of-order) procesorů trend používat více stupňů pipeline
    - Obvykle „lehce“ více než 10 stupňů
      - 14-19 u Haswell/Broadwell/Skylake/Kaby Lake
    - Mikroarchitektura *NetBurst* (Pentium 4)
      - *Hyper Pipelined Technology*
      - 20 stupňů od *Willamette*, 31 stupňů od *Prescott*

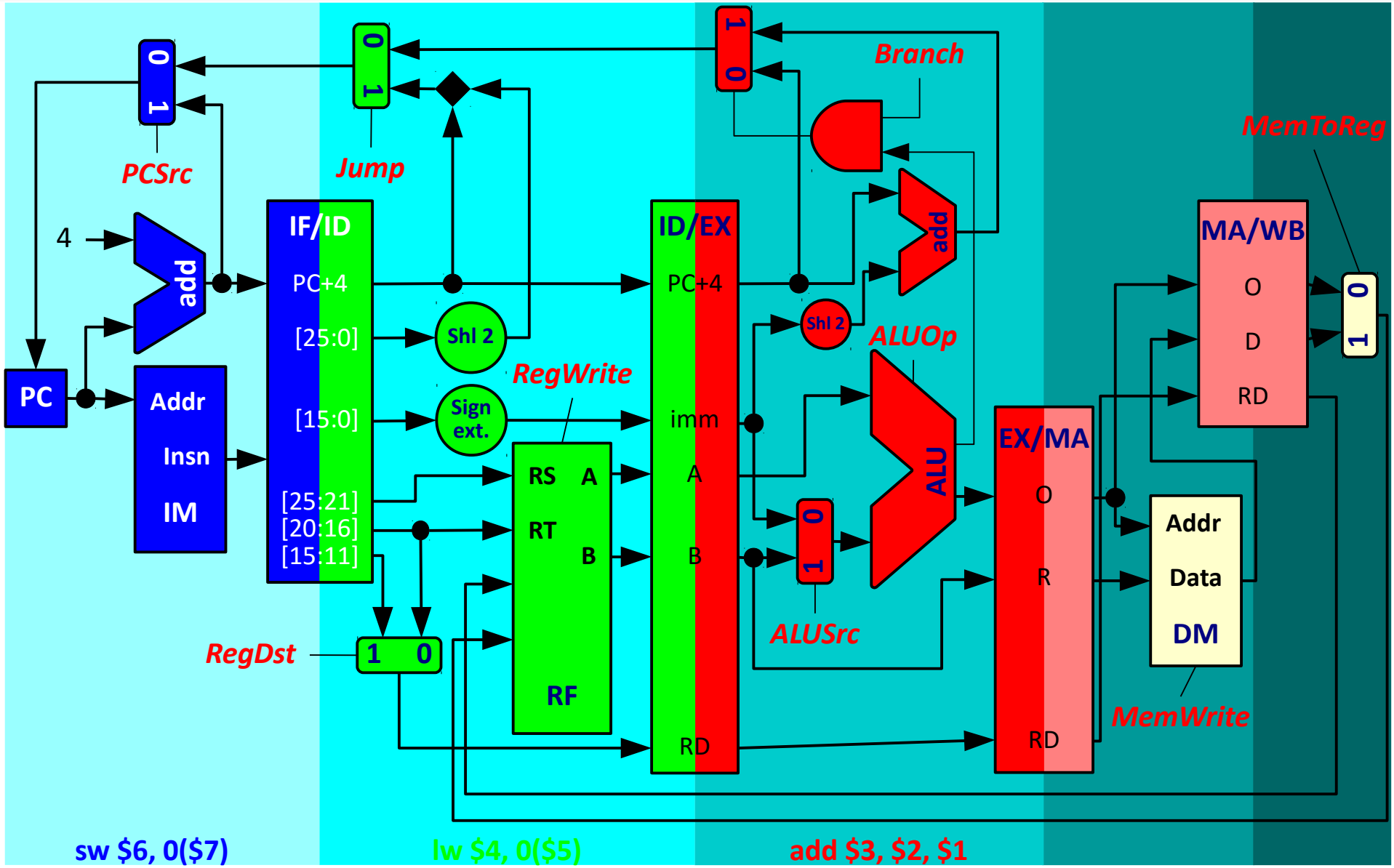
Zpracování 3 instrukcí, krok 1



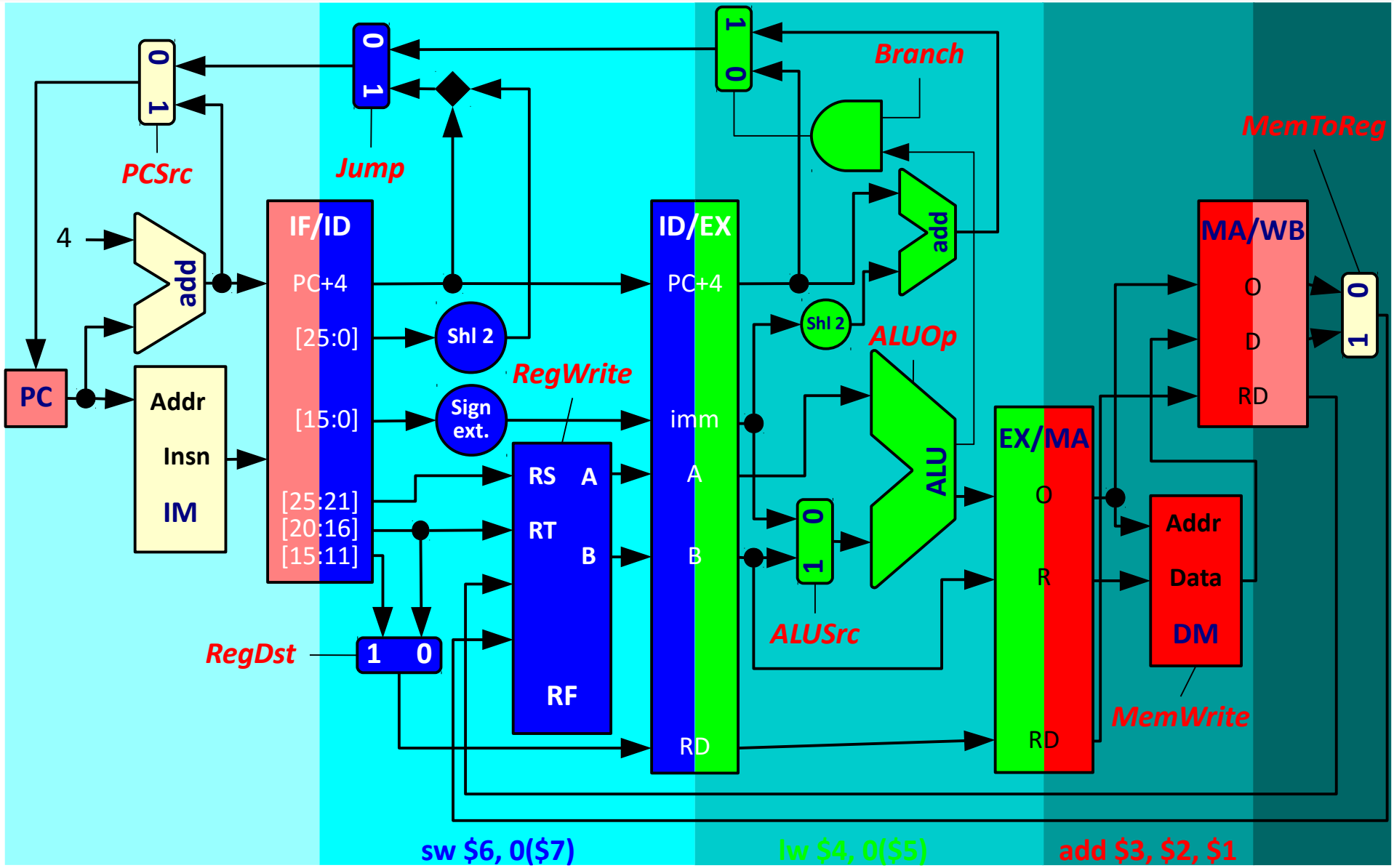
Zpracování 3 instrukcí, krok 2



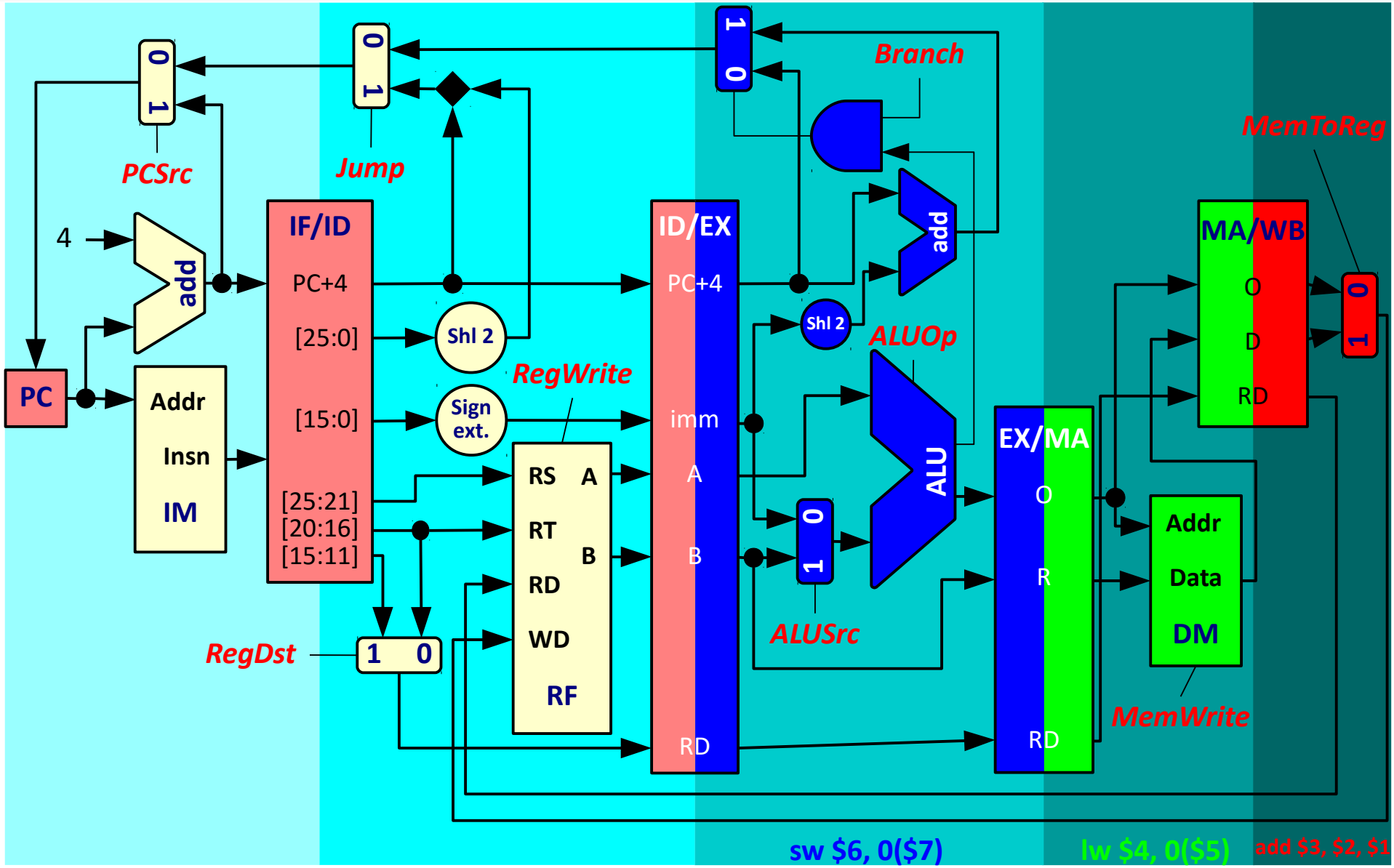
Zpracování 3 instrukcí, krok 3



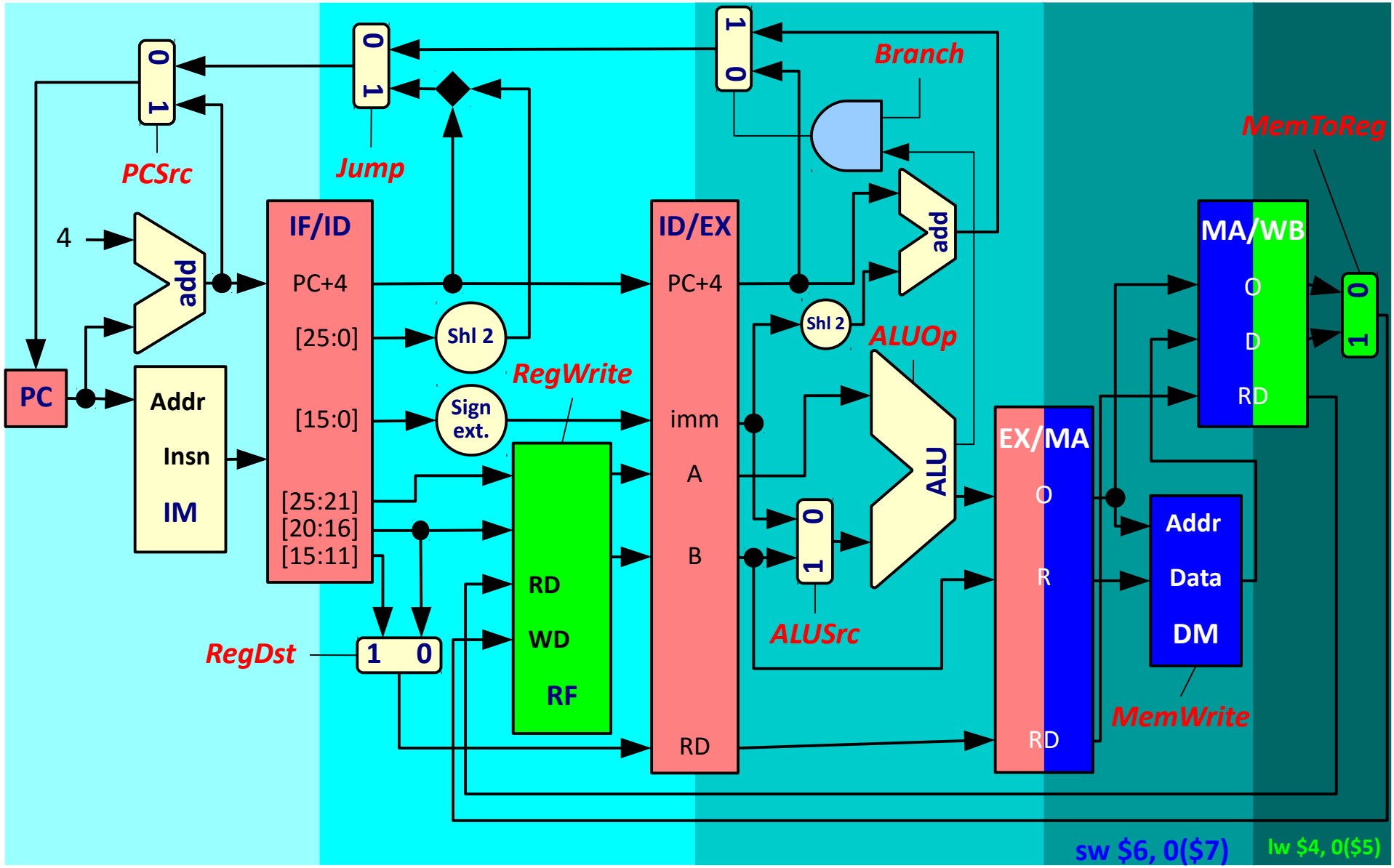
Zpracování 3 instrukcí, krok 4



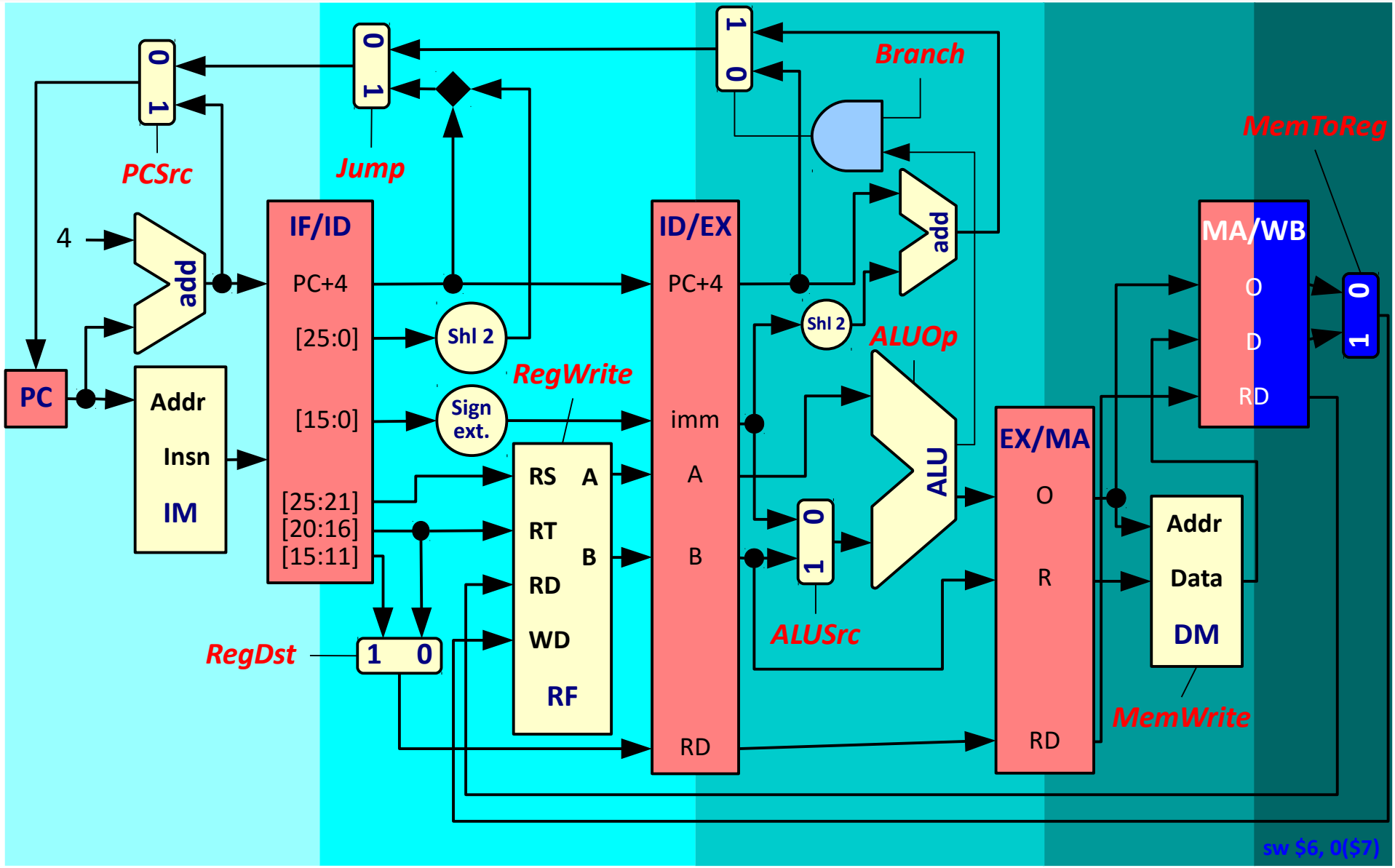
Zpracování 3 instrukcí, krok 5



Zpracování 3 instrukcí, krok 6



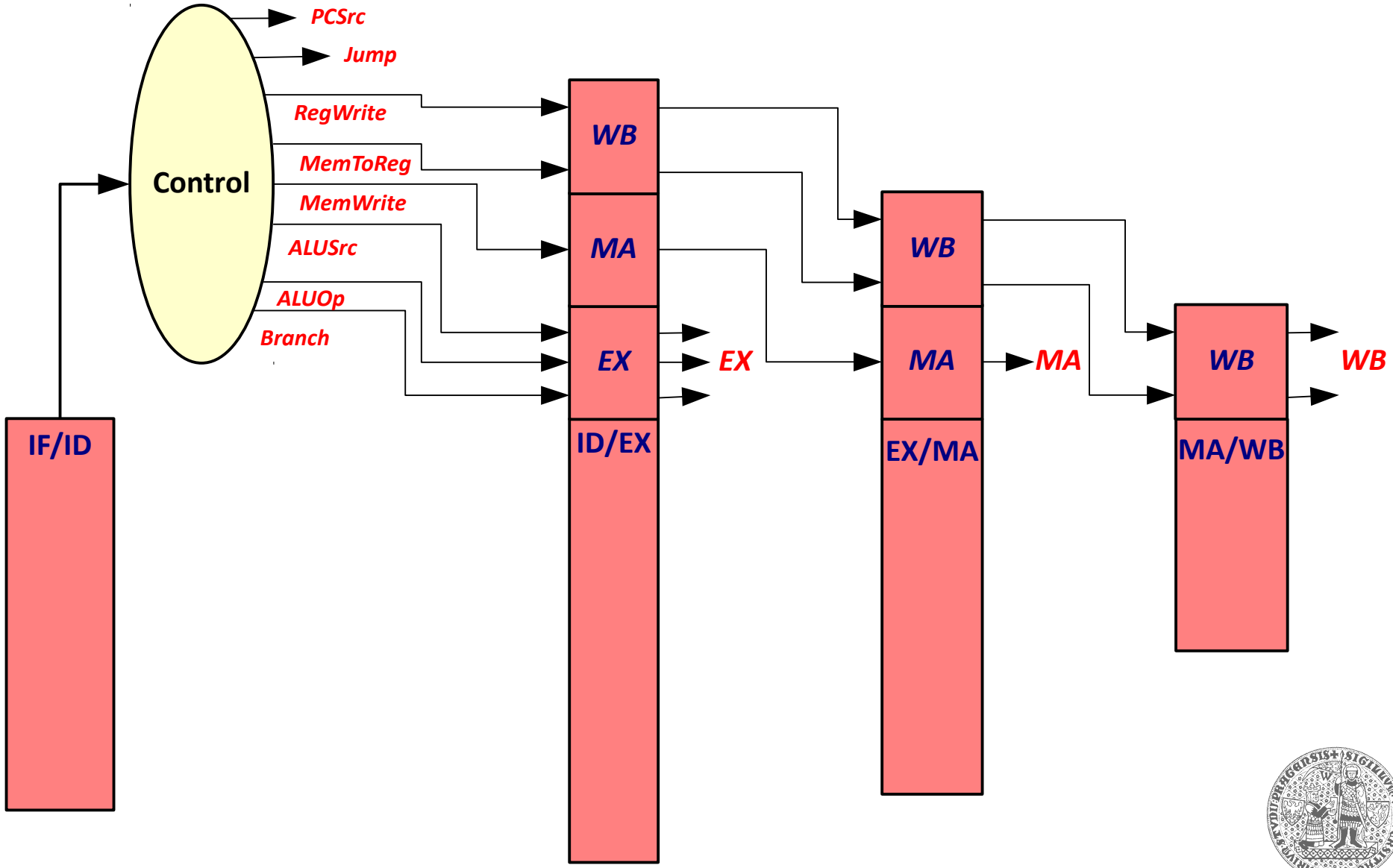
Zpracování 3 instrukcí, krok 7



Řízení pipeline

- **Různé přístupy**
  - Využití jednocyklového řadiče
    - Signály je potřeba aktivovat postupně
      - Kombinační obvod nebo ROM dekóduje operační kód instrukce
      - Stejně jako datová cesta je i cesta pro řídící signály zřetězena pomocí latch registrů pro jednotlivé úseky
        - Každá instrukce si nese řídící signály „s sebou“
  - Využití vícecyklového řadiče
    - Komplikované řešení
      - Jeden složitý konečný automat
      - Hierarchické konečné automaty (pro každý úsek pipeline)

Řízení pipeline (2)



Výkonnost zřetězené datové cesty

- **Jednocyklová datová cesta**
  - takt = **5*0ns***, ***CPI=1*** ⇒ **50*ns*** na instrukci
- **Vícecyklová datová cesta**
  - 20% branch (3T), 20% load (5T), 60% ALU (4T)
  - takt = ***11ns, CPI≈ (20% × 3) + (20% × 5) + (60% × 4) = 4***
  - ***44ns*** na instrukci
- **Zřetězená datová cesta**
  - takt = ***12ns*** (cca 50ns/5 kroků)
  - ***CPI = 1*** (v každém cyklu je dokončena 1 instrukce)
    - ***12ns*** na instrukci? Ve skutečnosti ***CPI = 1 + režie > 1***
  - ***CPI = 1.5*** ⇒ ***18ns*** na instrukci

Proč nelze dosáhnout CPI = 1?

- **Reálná pipeline**
  - CPI = 1 + *penalizace za zpoždění*
    - Penalizace odvozena z frekvence výskytu zpoždění a počet taktů zpoždění
      - Ani velké penalizace nevadí, pokud nenastávají příliš často
      - Penalizace ovlivňuje ideální počet stupňů pipeline
  - **Zpoždění** (*stall*) je prázdný krok v pipeline
    - Vložen s cílem ošetřit hazard pipeline
  - **Hazard**
    - Situace, která ohrožuje iluzi nezávislosti jednotlivých instrukcí

Typy hazardů v pipeline

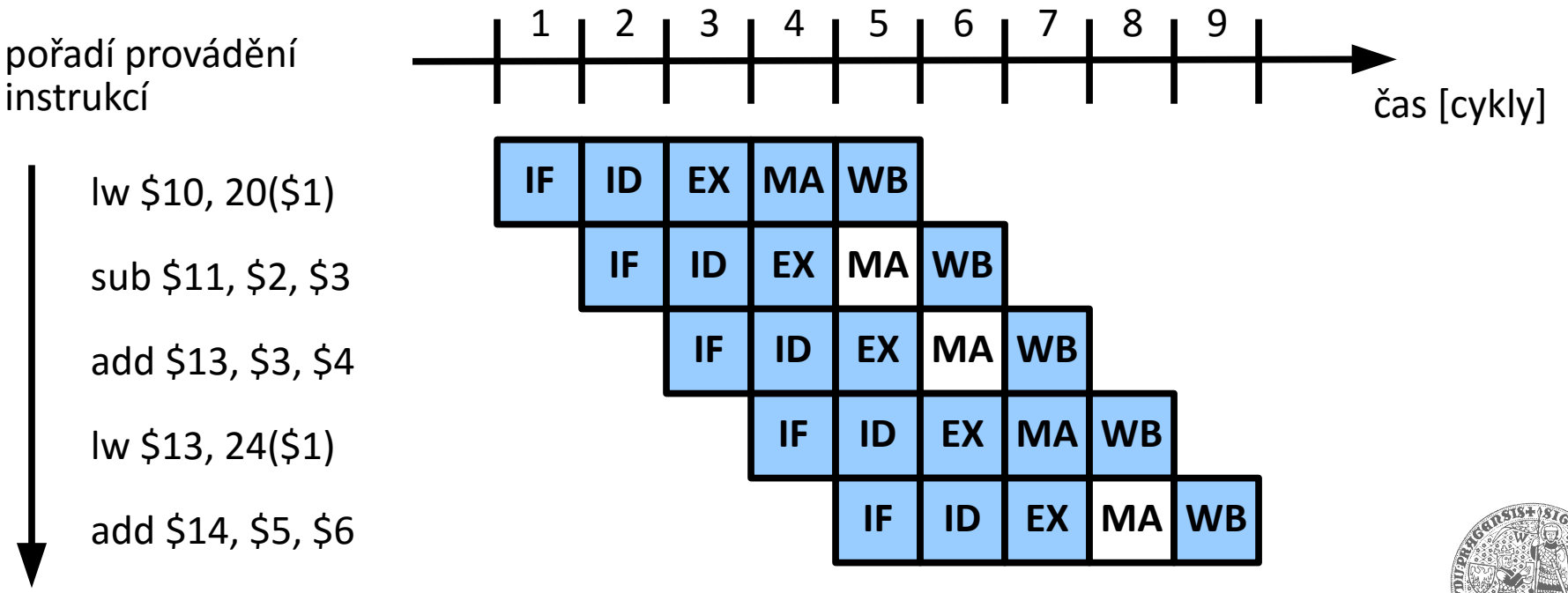
- **Strukturální hazard**
  - Hardware nepodporuje danou kombinaci instrukcí
  - Současné použití sdíleného prostředku z více stupňů pipeline
    - Řešení v našem případě: Oddělení instrukční a datové paměti
      - Reálně: Oddělená instrukční a datová cache

Typy hazardů v pipeline (2)

- **Datový hazard**
  - Instrukce nemá k dispozici data pro vykonání
    - Hodnoty argumentů jsou výsledky operací instrukce, která je ještě stále v pipeline
    - Nutnost počkat na předchozí instrukce
- **Řídící hazard**
  - Nutno učinit rozhodnutí před vykonáním instrukce
  - Podmíněný skok je zpracováván až ve 3. kroku
    - V té době už pipeline zpracovává 2 další instrukce

Pipeline diagramy

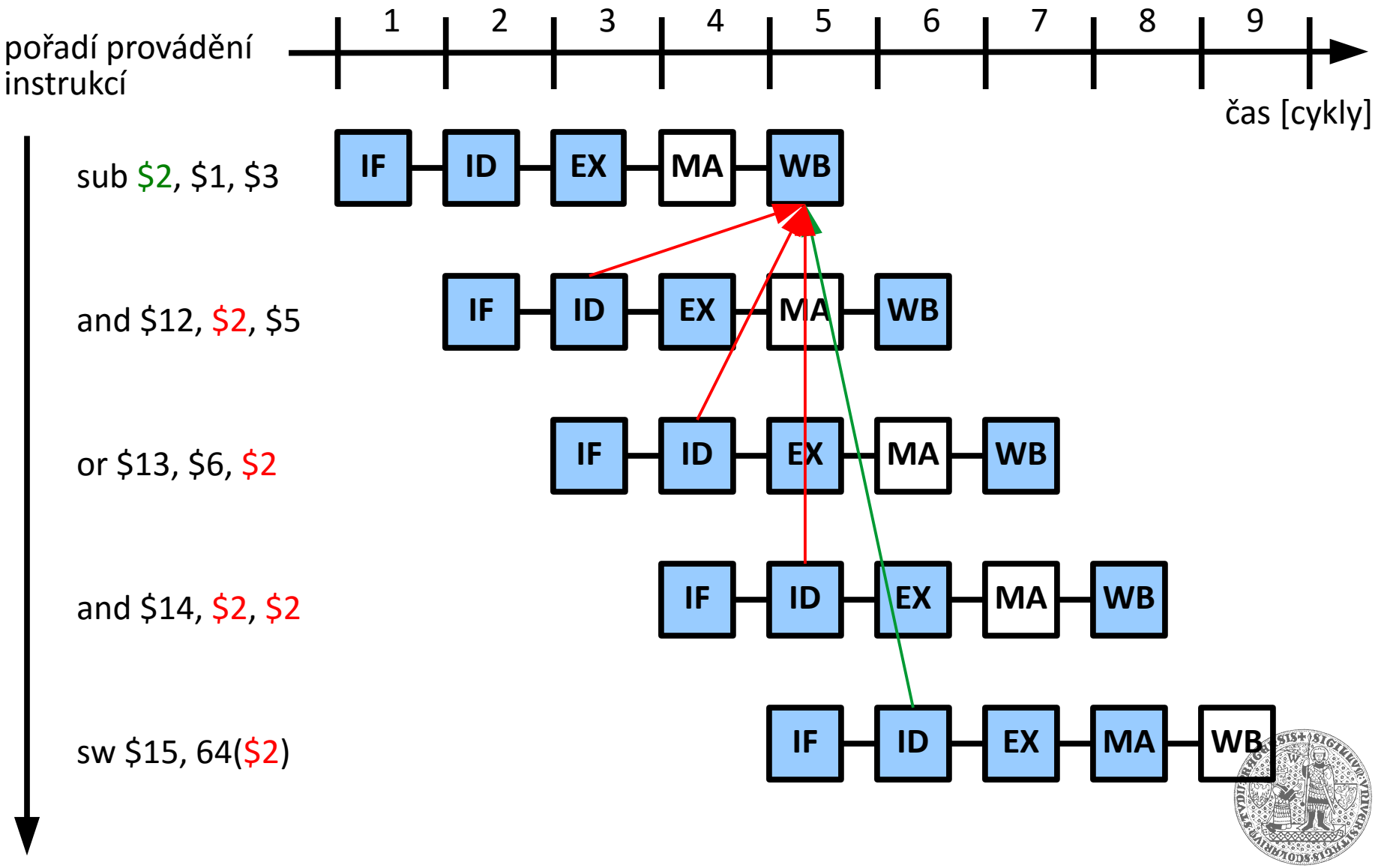
- **Zjednodušená reprezentace pipeline**
  - Všechny úseky trvají právě 1 cyklus
  - Diskrétní čas v hodinových cyklech



Datový hazard

- **Závislosti mezi operandy instrukcí**
  - Operand je výsledek operace předchozí instrukce
  - Operand je obsah paměti čtený předchozí instrukcí
  - Zjišťování závislostí (při návrhu)
    - Graf
      - Vrcholy = aktivní prvky pipeline v čase
      - Hrany = řídící nebo datové signály
      - Závislosti = hrany vedoucí „dopředu v čase“
  - Zjišťování závislostí (v hardwaru)
    - Vyhodnocování čísel registrů a paměťových adres všech instrukcí právě prováděných v pipeline

Datový hazard (2)



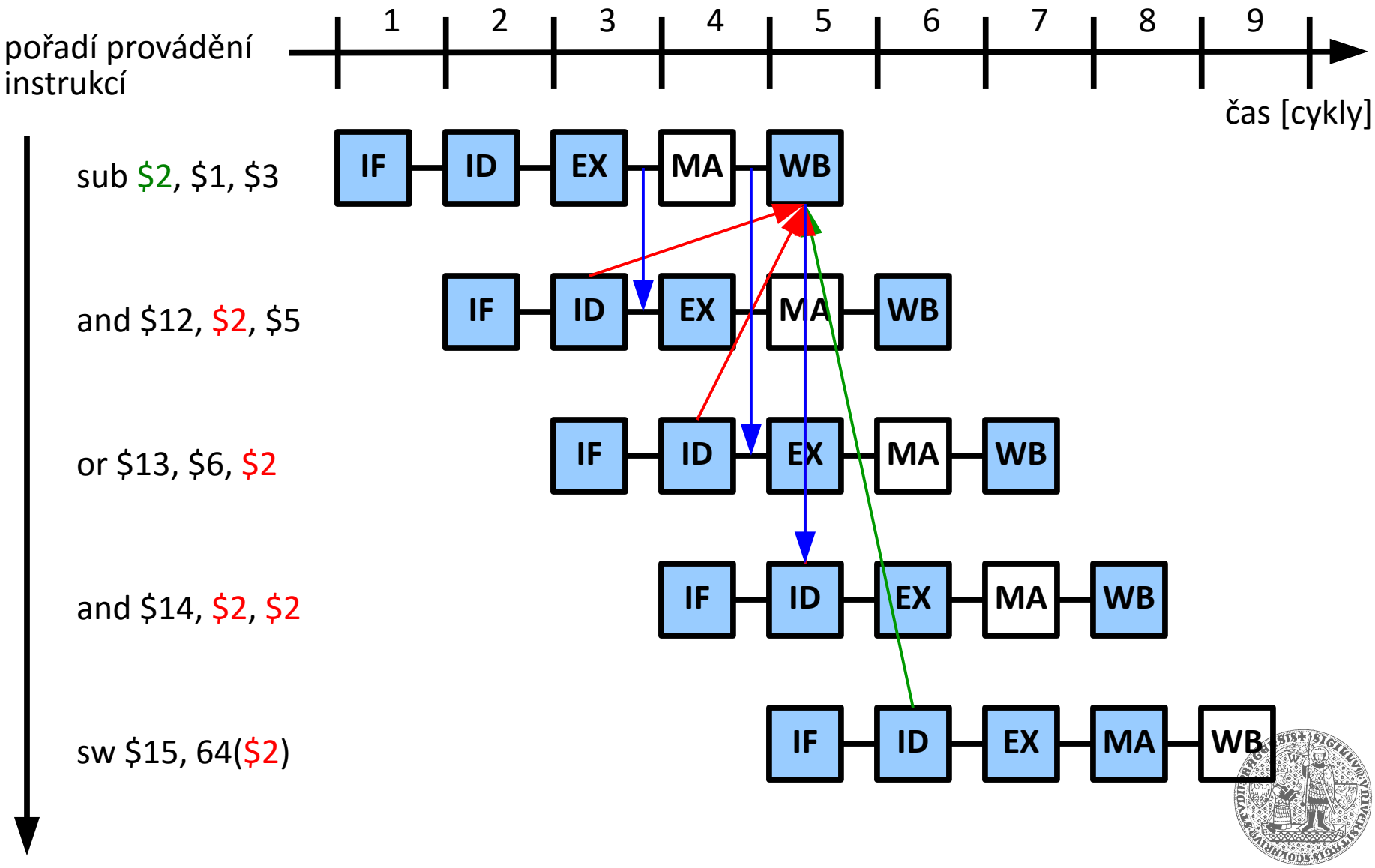
Ošetření datových hazardů

- **Na úrovni překladače (*software interlock*)**
  - Řazení instrukcí tak, aby došly do pipeline až v okamžiku, kdy mají k dispozici všechny operandy
    - Mezi dvojici závislých instrukcí vložit nezávislou instrukci (nepotřebují žádný registr, který je předmětem závislosti)
      - V nejhorším případě použití prázdné instrukce (*nop*)
  - Nevhodné řešení z hlediska kompatibility
    - ***MIPS** = **M**icroprocessor without **I**nterlocked **P**ipeline **S**tages*

Ošetření datových hazardů (2)

- ***Forwarding/bypassing***
  - Poskytnutí mezivýsledku následující instrukci
    - Je-li výsledek v některém z následujících pipeline registrů (pro předchozí instrukci), lze jej přečíst přímo z tohoto pipeline registru a nečekat na registrové pole
  - *Forwarding unit*
    - Zdrojový operand vykonávané instrukce je cílový operand výsledku dřívější instrukce
      - EX/MA.RD := ID/EX.RS
      - EX/MA.RD := ID/EX.RT
      - MA/WB.RD := ID/EX.RS
      - MA/WB.RD := ID.EX.RT

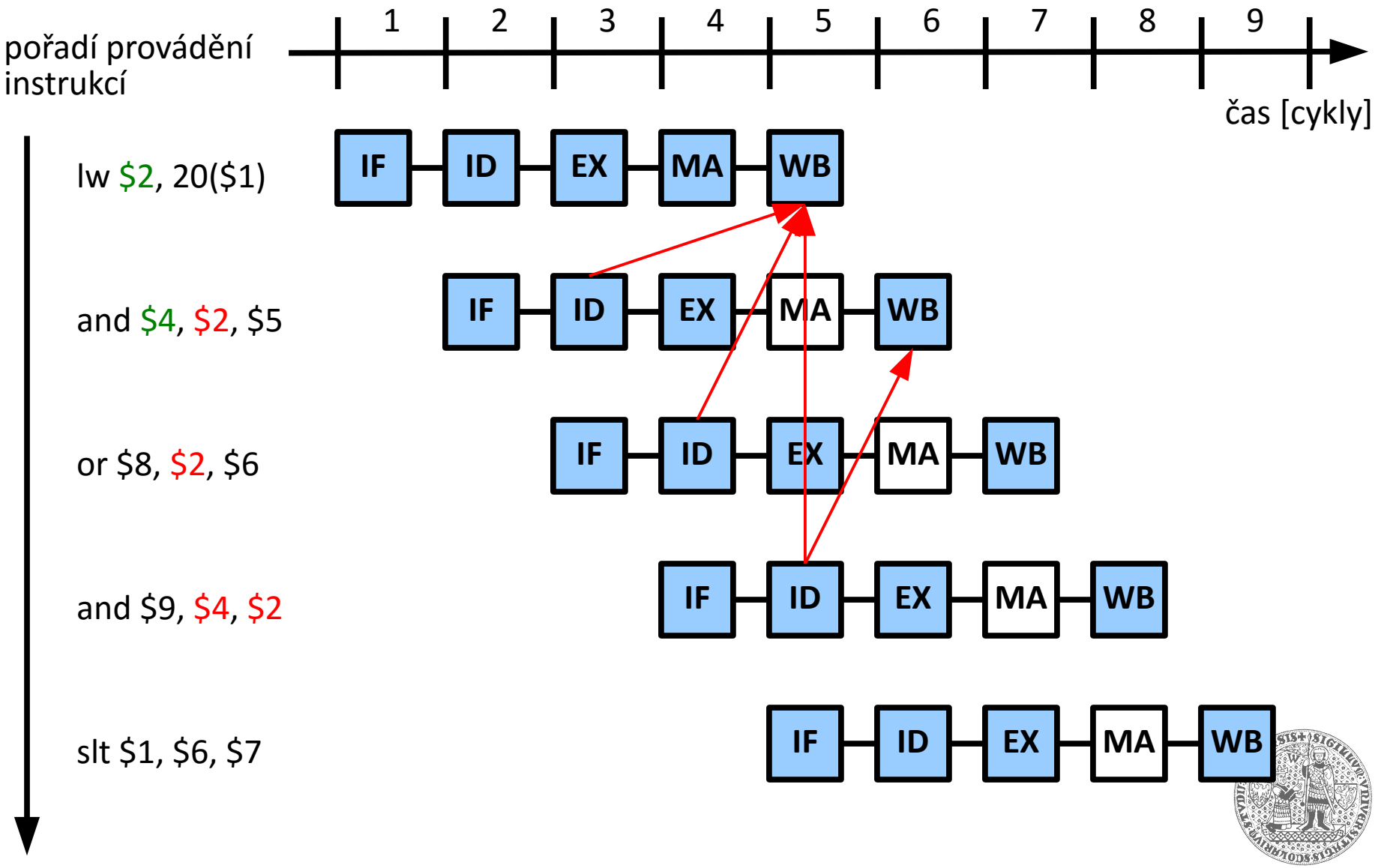
Datový hazard – forwarding/bypassing



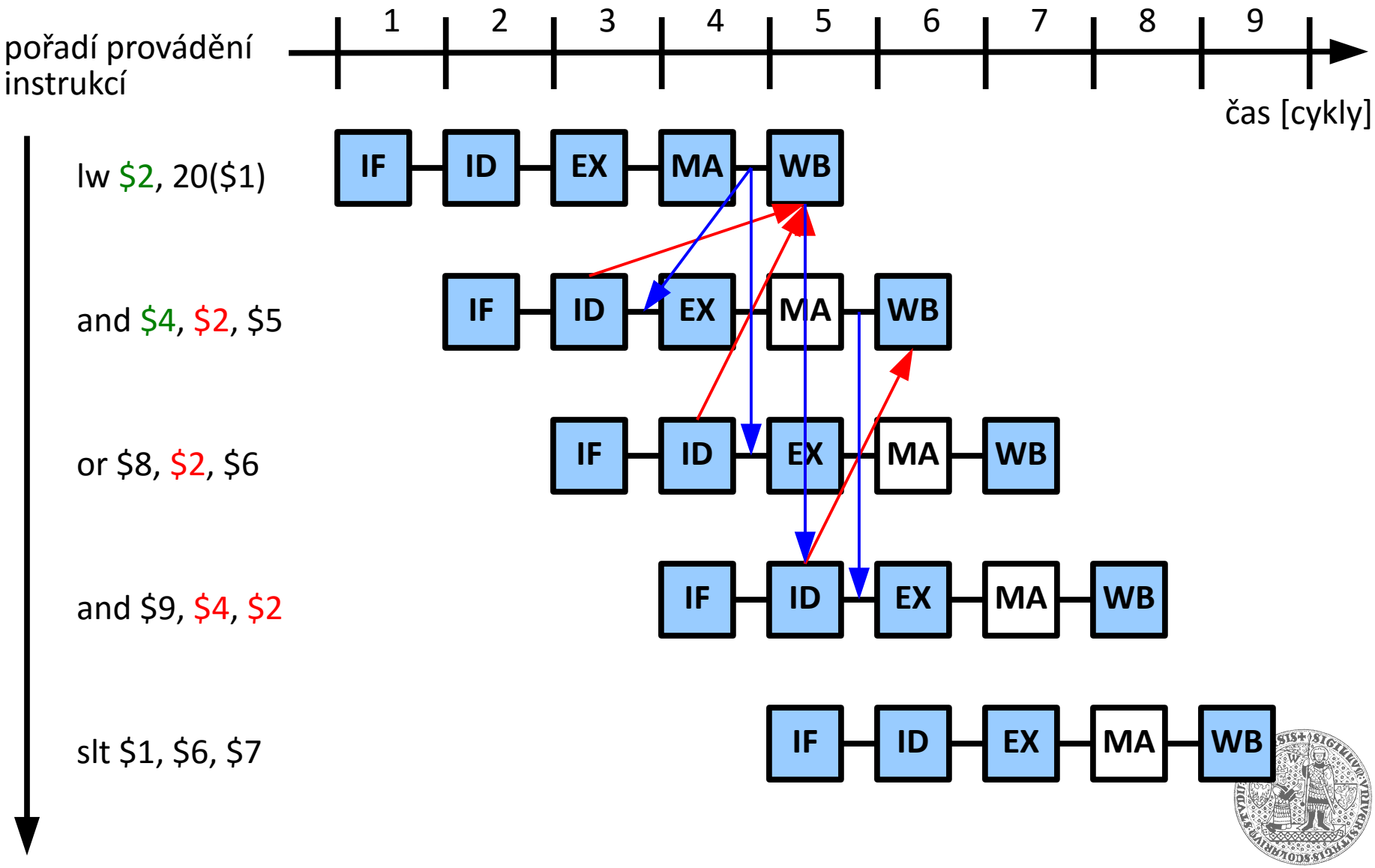
Ošetření datových hazardů (3)

- **Zpoždění instrukce v pipeline**
  - V případě použití operandu (registru) bezprostředně po jeho načtení z paměti (*load/use dependency*)
  - *Hazard detection unit*
    - Testování, zda zdrojový operand dekódované instrukce je cílový operand dřívější instrukce čtení z paměti
      - ID/EX.MemRead &&
        (ID/EX.RT == IF/ID.RS || ID/EX.RT == IF/ID.RT)

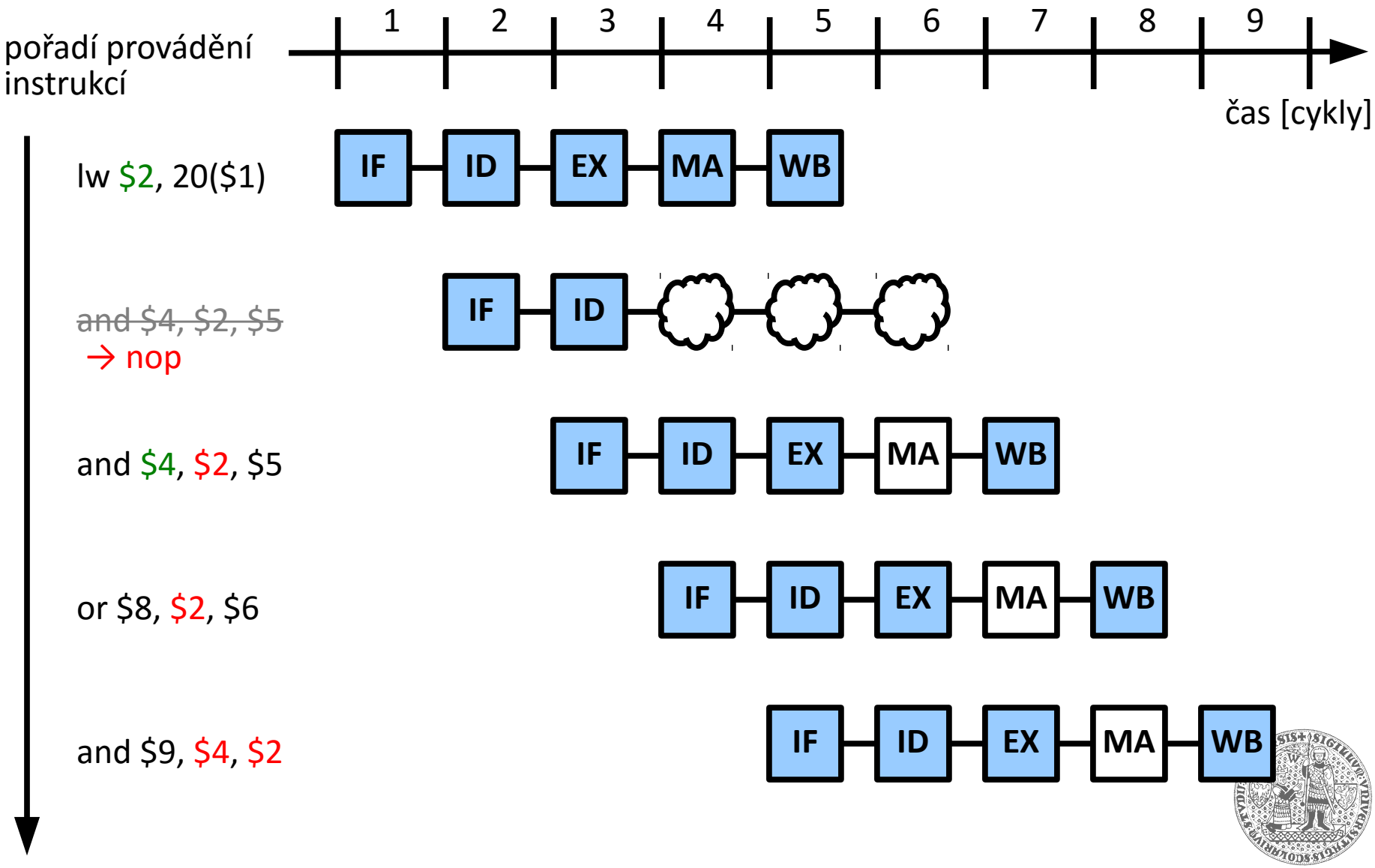
Datový hazard – load/use závislosti



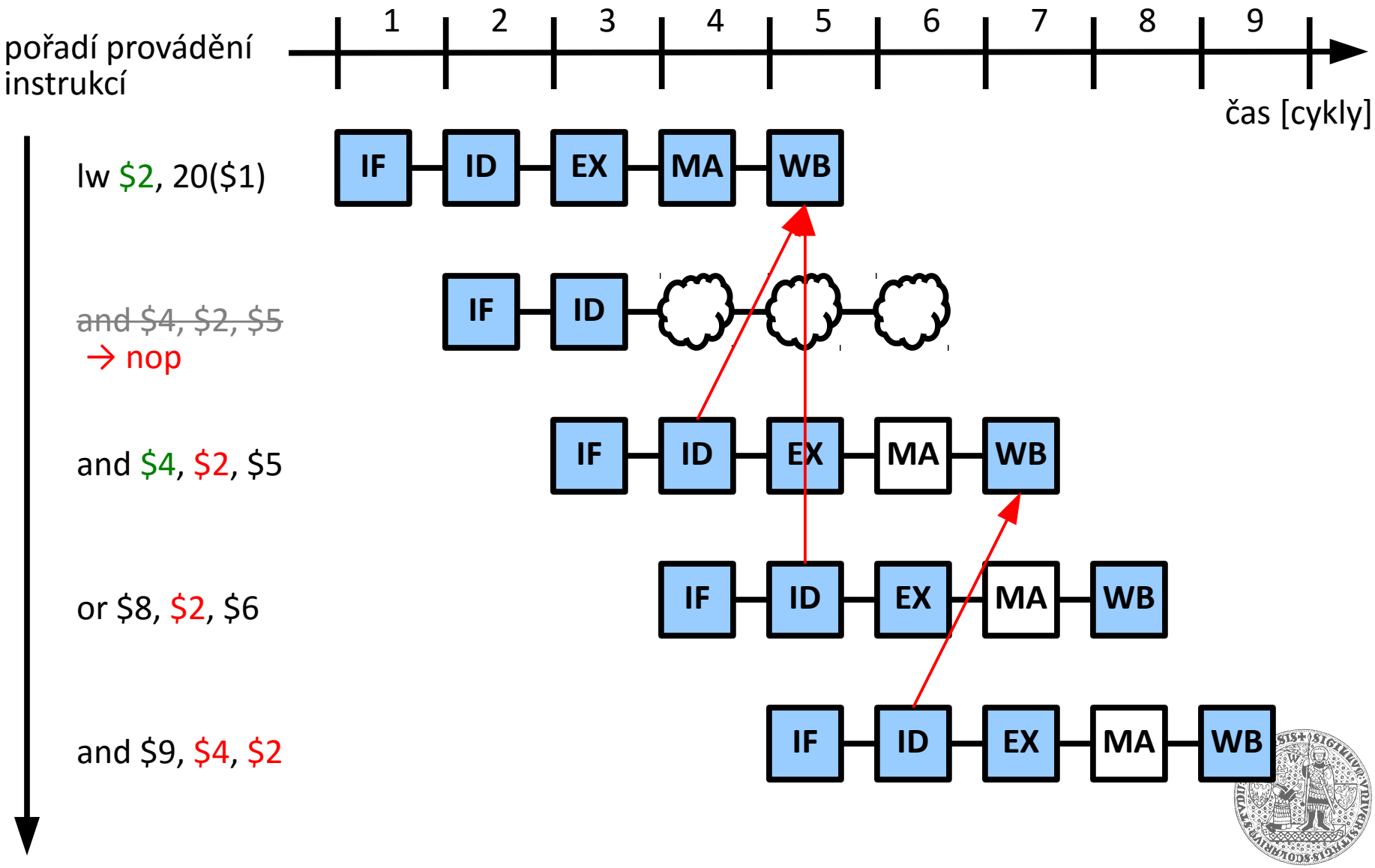
Datový hazard – load/use a forwarding



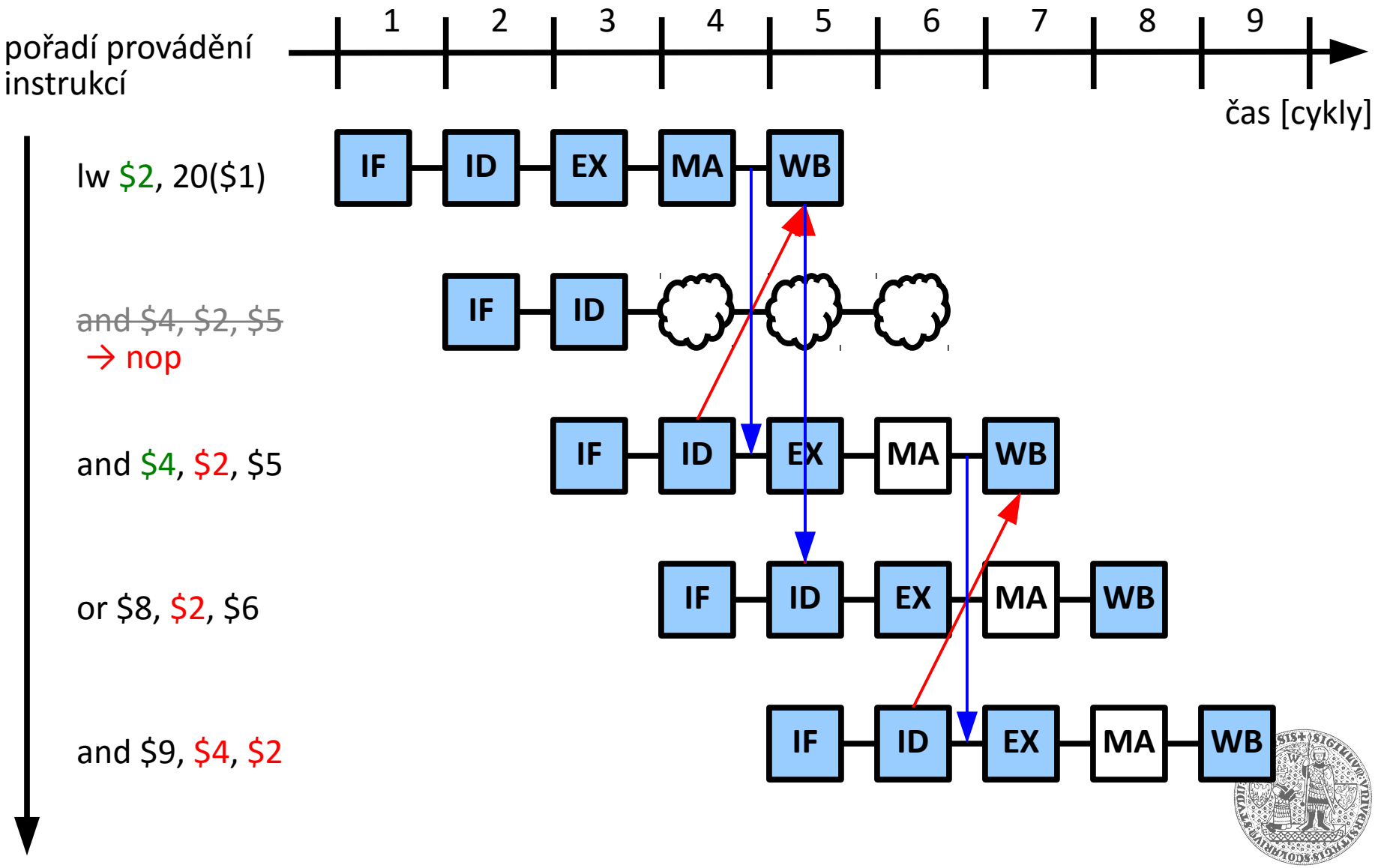
Datový hazard – zpoždění pipeline



Datový hazard – zpoždění pipeline (2)



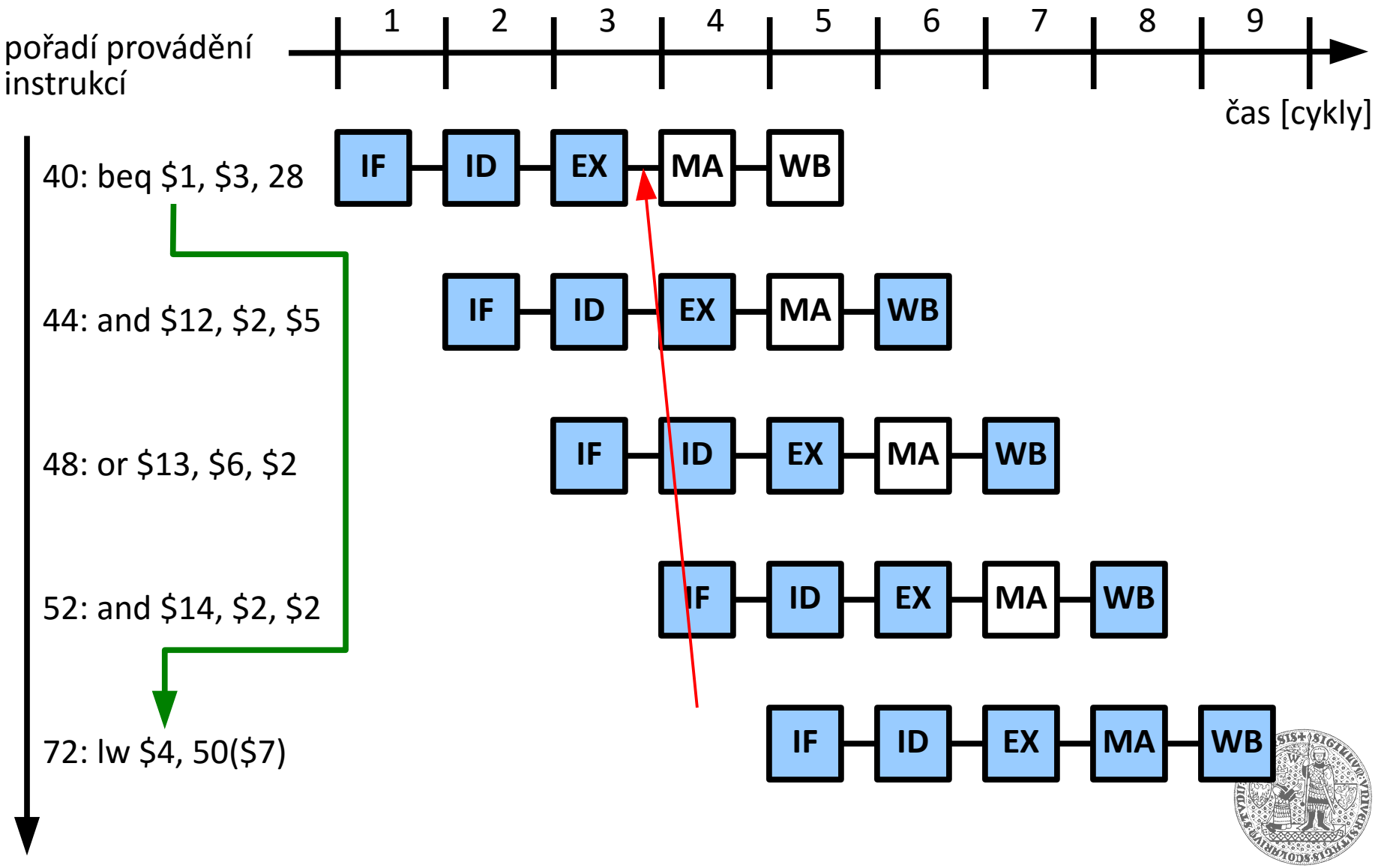
Datový hazard – zpoždění pipeline (3)



Řídící hazard

- **Odkud číst další instrukci**
  - Hodnota PC ovlivněna podmíněnými a nepodmíněnými skoky
    - Rozhodnutí závisí na výsledku, který bude k dispozici o několik taktů později, než je potřeba
  - Výjimky a přerušení
- **Ošetření řídícího hazardu**
  - Forwarding není možný
    - Cílová adresa skoku může být známa, ale vyhodnocení podmínky ne
  - **Cíl:** Minimalizovat zpoždění pipeline

Řídící hazard – větvení



Ošetření řídícího hazardu

- **Zpozdit pipeline do dokončení skoku**
- **Snaha udržet stále plnou pipeline**
  - Odkud číst další instrukce?
    - Zrychlené vyhodnocení skoku
      - Přesun výpočtu adresy skoku a vyhodnocení podmínky z úseku EX do úseku ID
      - Forwardování mezivýsledků
      - Předpokladem je jednoduchá podmínka skoku (např. test na rovnost)
      - Zpoždění 1 cyklus při skoku
    - Zpožděný skok (*branch delay slot*)
      - Vždy se vykoná 1 instrukce po instrukci skoku

Ošetření řídícího hazardu (2)

- **Snaha udržet stále plnou pipeline**
  - Odkud číst další instrukce?
    - *Branch target buffer*
      - Cache cílových adres specifických instrukcí
    - Spekulativní provádění instrukcí
      - Provádění instrukcí „naslepo“ bez vyhodnocení podmínky skoku
      - Pokud se později zjistí, že se má provádět jiná varianta, pipeline se vyprázdní (*rollback*)
      - U složitých procesorů může znamenat částečnou virtualizaci registrového souboru a store bufferů (zápisy do paměti)

Predikce skoku

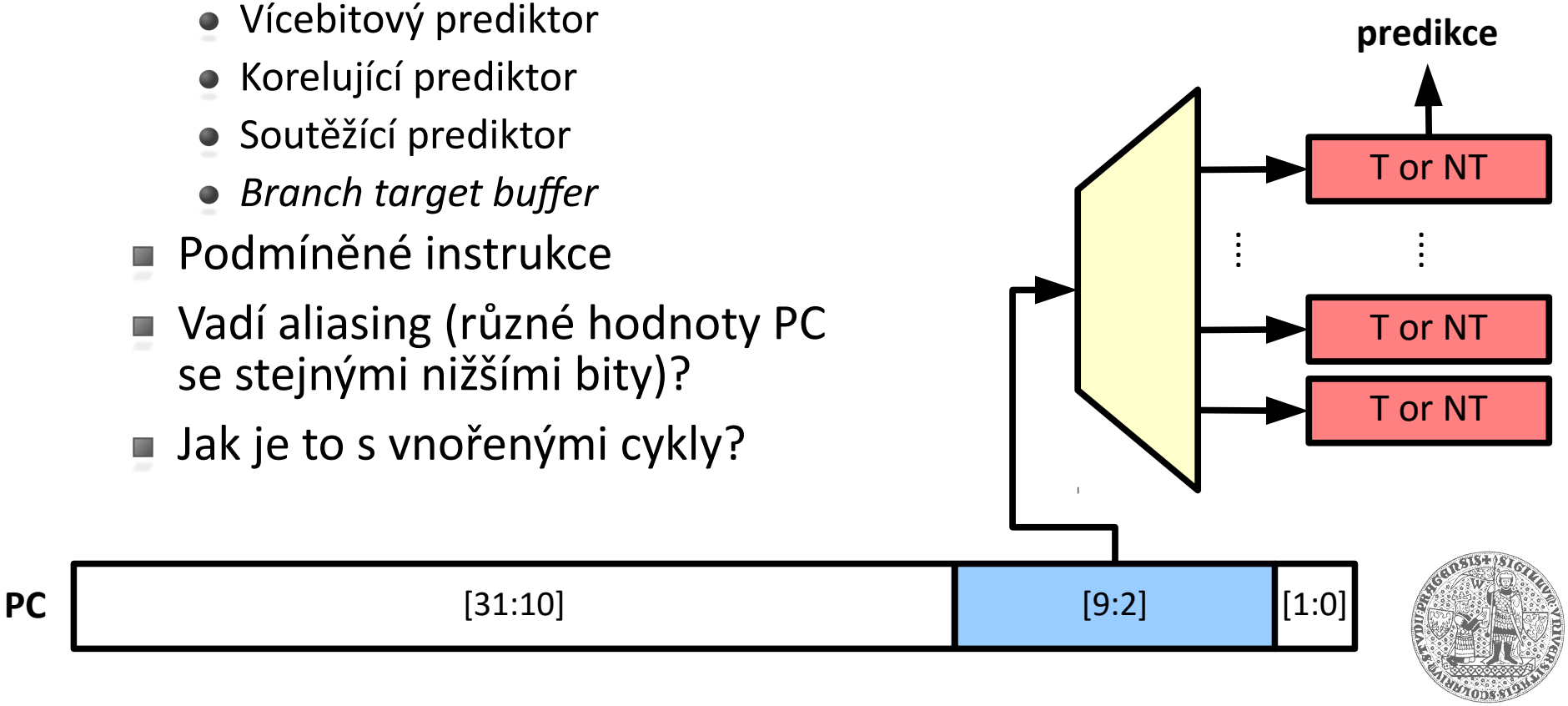
- **Statická predikce**
  - Predikce nezávisí na historii
  - Bez nápovědy
    - Heuristika určena hardwarem
    - Obvykle se předpokládá, že se skok neprovede
    - Méně často složitější heuristiky (vzdálenost skoku apod.)
  - S nápovědou
    - Pravděpodobnější chování skoku určeno bitem v instrukci

Predikce skoku (2)

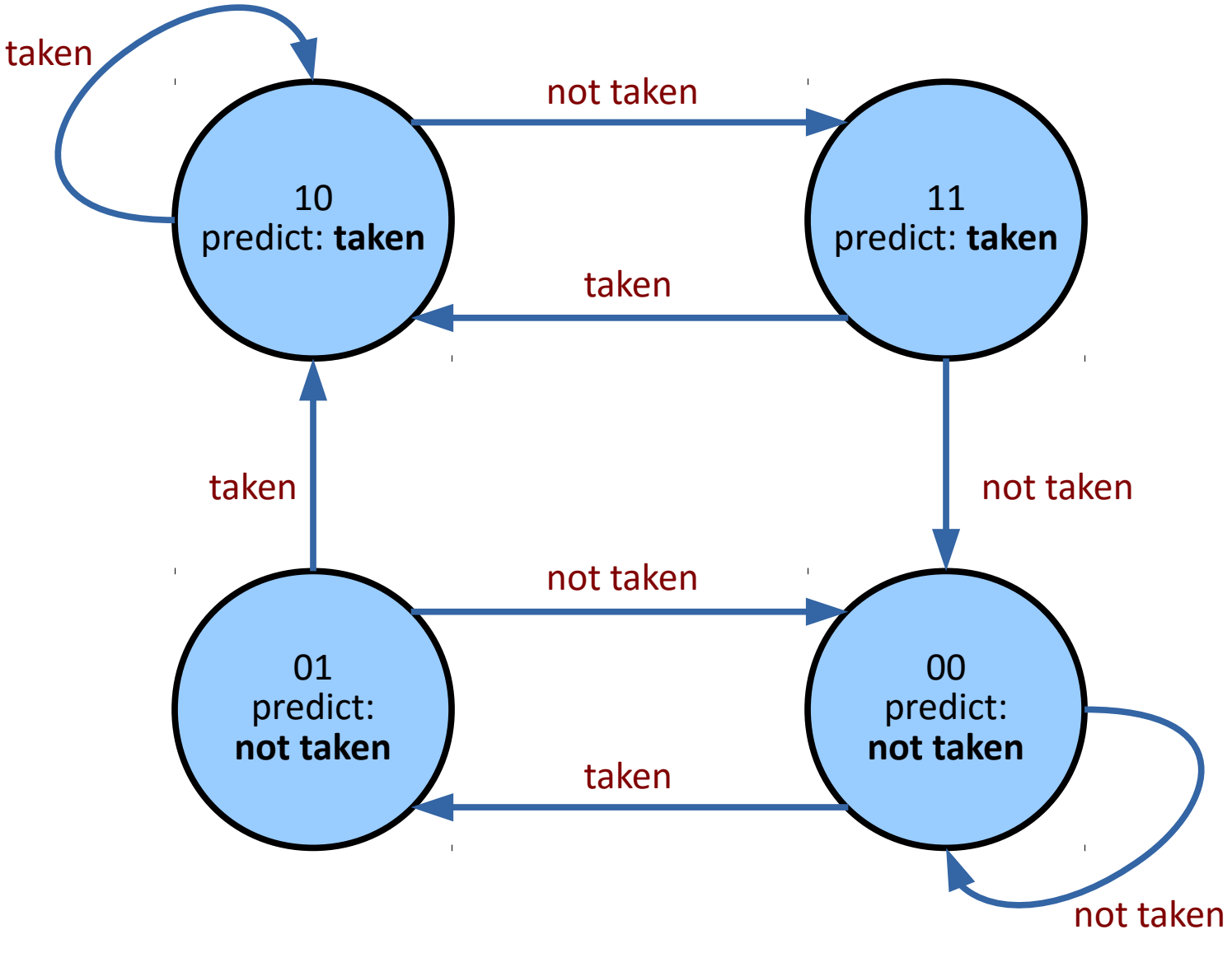
- **Dynamická predikce**
  - Predikce na základě historie
  - *Branch prediction buffer (history table)*
    - Predikce stejného chování jako v minulosti
    - Cykly většinou skáčí na začátek → 2 špatné predikce
      - Iniciální chování může být vylepšeno nápovědou
  - 2-bitový prediktor
    - Musí se dvakrát splést, než změní „názor“

Branch history table

- **Základní prediktor**
  - Tabulka bitů predikce indexována částí PC
  - Rozšíření
    - Vícebitový prediktor
    - Korelující prediktor
    - Soutěžící prediktor
    - *Branch target buffer*
  - Podmíněné instrukce
  - Vadí aliasing (různé hodnoty PC se stejnými nižšími bity)?
  - Jak je to s vnořenými cykly?



2-bitový prediktor skoků



Zřetězené zpracování a výjimky

- **V pipeline je *k* instrukcí současně**
  - Která instrukce způsobila výjimku?
    - Informace se musí propagovat v pipeline registrech
  - Když nastane více výjimek, kterou obsloužit dříve?
    - Tu, která patří k instrukci dříve v pořadí
  - Obsluha výjimky
    - Důraz na zachování konzistentního stavu procesoru
      - Data z pipeline registrů se nikam nezapisují (registrový soubor a paměť obsahují hodnoty před výskytem výjimky)
    - Před obsluhou nutno vyprázdnit pipeline (rollback)
      - Stejná logika jako při spekulativním provádění skoků

Prodlužování pipeline

- **Trend: Delší pipeline**
  - 486 (5 stupňů), Pentium (7 stupňů), Pentium III (12 stupňů), Pentium 4 (20 – 31 stupňů), Core (14 stupňů)
  - Důsledky
    - Vyšší taktovací frekvence
      - Zvýšení není lineární, od určité délky způsobuje pokles výkonu
        - Pentium 4 na 1 GHz je pomalejší než Pentium III na 800 MHz
    - Obecně vyšší CPI
      - Vyšší penalizace za špatně predikované skoky
      - Zpoždění u hazardů, které nelze vyřešit bypassem

Zvyšování počtu pipeline

- ***Flynn bottleneck***
  - Teoretické omezení skalární pipeline
    - V každém stupni pouze 1 instrukce → CPI = IPC = 1
    - V praxi není možné ji dosáhnout (hazardy)
    - Výnosy z prodlužování pipeline rychle klesají
- **Superskalární (multiple issue) pipeline**
  - Dnešní procesory typicky 4 pipeline
  - Využití paralelismu na úrovni instrukcí
    - Nezávislé instrukce se dají vykonávat současně

Paralelismus na úrovni instrukcí

- **Optimální plánování instrukcí překladačem**
  - Nutné i pro skalární pipeline (omezení hazardů)
  - Pro superskalární pipeline složitější
    - Kolik nezávislých proudů instrukcí lze v programu najít?
      - **Ideální případ:** Kopírování bloku paměti
        - Při rozbalení smyčky dostáváme spoustu nezávislých instrukcí
      - V běžných programech zdaleka ne tolik
    - **Alternativa:** *Simultaneous multithreading* (SMT)

Simultaneous multithreading

- **Zpracování více nezávislých vláken současně**
  - Na úrovni superskalární pipeline
    - Kódy nezávislých vláken by na sobě měly být z definice nezávislé, takže využijí vícenásobnou pipeline efektivněji
    - Energeticky efektivnější než implementace více jader
      - Zvětšení registrového souboru a logiky čtení instrukcí
      - Ostatní logika procesoru zůstává prakticky nezměněna
    - Detekovatelná softwarem (logické procesory)
    - **Nevýhody:** Implicitně sdílené zdroje (cache, přístup do paměti)
    - *Intel Hyper-Threading Technology*

Temporal multithreading

- **Úprava principu SMT pro jednu pipeline**
  - Pragmaticky: Přepínání vláken na úrovni procesoru
  - **Jemné** (*fine-grained*)
    - Přepnutí vlákna s každou instrukcí
    - Niagara (Sun UltraSPARC T1)
  - **Hrubé** (*coarse-grained*)
    - K přepnutí dojde tehdy, pokud zpracování aktuální instrukce způsobí zpozdění (pipeline stall, cache miss, page fault)
    - Montecito (Intel Itanium 2)

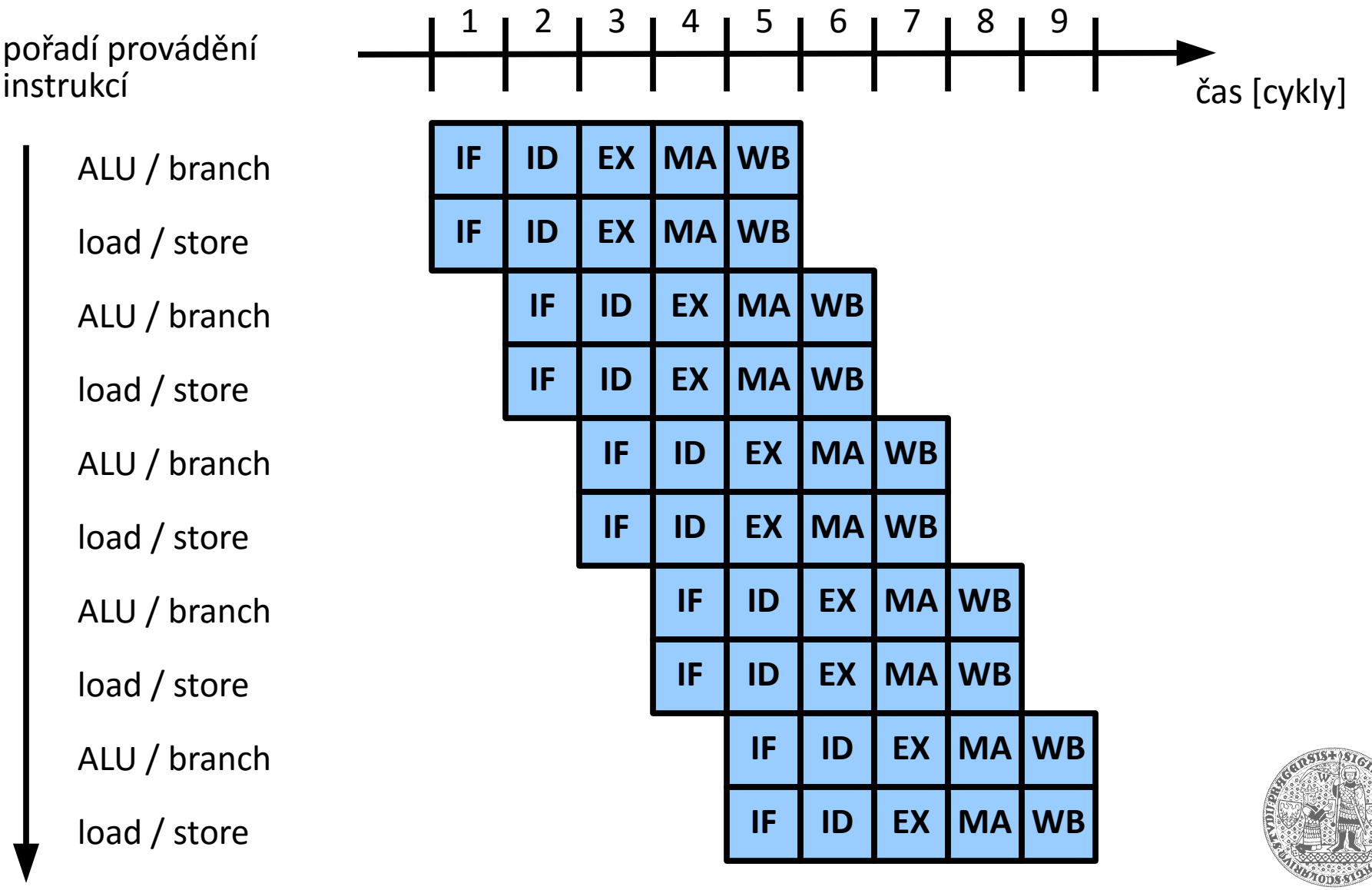
Typická superskalární pipeline

- **Čtení instrukcí**
  - Celý blok cache (16, 32 nebo 64 bytů), 4 – 16 instrukcí
  - Predikce jednoho skoku v každém cyklu
- **Paralelní dekódování instrukcí**
  - Detekce závislostí a hazardů
- **Víceportové, vícenásobné registrové pole**
- **Více výkonných jednotek**
  - Různé ALU, forwarding/bypassing logika mezi nimi
- **Přístup do paměti**

Static multiple issue

- **Plánování superskalární pipeline překladačem**
  - ***Issue packet***
    - Sada instrukcí, které se mají provést současně
    - Sloty v issue packetu nejsou ortogonální
      - *Very Long Instruction Word* (VLIW)
      - *Explicit Parallel Instruction Computer* (EPIC)
  - Výkon silně závisí na překladači
    - Vhodné plánování instrukcí (plnění instrukčních slotů)
    - Některé nebo všechny důsledky datových a řídících hazardů řeší překladač
    - Statická predikce skoků

Příklad: Multiple issue MIPS



Příklad: Multiple issue MIPS (2)

- **Změny oproti single issue**
  - Načítání 64bitových instrukcí, zarovnání na 8 bytů
    - Nevyužitý slot může obsahovat NOP
  - Registrové pole pro přístup z obou slotů
  - Samostatná sčítačka pro výpočet efektivní adresy
- **Nevýhody**
  - Vyšší latence při použití výsledků
    - Registrové operace 1 instrukce, load 2 instrukce
    - Složitější plánování instrukcí překladačem
  - Penalizace v důsledku hazardů jsou dražší

Příklad: Multiple issue MIPS (3)

- **Jak naplánovat následující kód?**

```
Loop:  lw     $t0, 0($s1)
       addu   $t0, $t0, $s2
       sw     $t0, 0($s1)
       addi   $s1, $s1, -4
       bne    $s1, $zero, Loop
```

	ALU or branch insn	Data transfer insn	Clock cycle
Loop:		lw $t0, 0($s1)	1
	addi $s1, $s1, -4		2
	addu $t0, $t0, $s2		3
	bne $s1, $zero, Loop	sw $t0, 4($s1)	4

- **Výkonnost?**
  - 5 instrukcí, 4 takty → CPI = 0.8 (místo ideálních 0.5)

Příklad: Multiple issue MIPS (4)

- **Program po rozbalení 4 iterací cyklu**

	ALU or branch insn	Data transfer insn	Clock cycle
Loop:	addi $s1, $s1, -16	lw $t0, 0($s1)	1
		lw $t1, 12($s1)	2
	addu $t0, $t0, $s2	lw $t2, 8($s1)	3
	addu $t1, $t1, $s2	lw $t3, 4($s1)	4
	addu $t2, $t2, $s2	sw $t0, 16($s1)	5
	addu $t3, $t3, $s2	sw $t1, 12($s1)	6
		sw $t2, 8($s1)	7
	bne $s1, $zero, Loop	sw $t3, 4($s1)	8

- **Přejmenování registrů (překladačem)**
  - nutné k odstranění falešných závislostí vzniklých při rozbalení cyklu
  - místo $t0 překladač použil jiný registr pro každou iteraci

Příklad: Itanium (IA-64)

- **Hlavní rysy**
  - Mnoho registrů
    - 128 general purpose, 128 floating point, 8 branch, 64 condition
    - Registrová okna s podporou přetečení do paměti
  - EPIC *instruction bundle*
    - Svazek instrukcí vykonávaný současně
    - Pevný formát, explicitní závislosti
      - **Stop bit:** Indikace, zda následující bundle závisí na aktuálním
  - Podpora spekulace a eliminace větvení
    - Instrukce se provede, ale teprve později se rozhodne, zda její efekt bude trvalý (provede se rollback)

Příklad: Itanium (IA-64) (2)

- **Další zajímavé vlastnosti**
  - *Instruction group*
    - Skupina instrukcí bez datových závislostí
    - Odděleny instrukcí se stop bitem
      - Kvůli dopředné kompatibilitě (rozšíření počtu pipelines)
  - Struktura *instruction bundle*
    - 5 bitů template (použité výkonné jednotky)
    - 3 × 41 bitů instrukce
    - Většina instrukcí může záviset na podmínkovém registru

Dynamic multiple issue

- **Dynamické plánování instrukcí v pipeline**
  - Optimální využití paralelismu, eliminace hazardů a prostojů
  - Zpracování instrukcí mimo pořadí
    - Zachování iluze sekvenčního pořadí
  - Překladač se snaží procesoru plánování usnadnit
- **Spekulativní provádění instrukcí**
  - Provedení instrukcí s potenciálně špatnými operandy nebo bez zaručení využití výsledku
  - Mechanismus rollbacku podobně jako při predikci skoků

Příklad: dynamické plánování instrukcí

```
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)
```

Out-of-order execution

- **Procesor je řízen datovými závislostmi**
  - Kolize mezi jmény registrů v instrukcích
    - RAW (*Read After Write*, *true data dependency*)
      - Výsledek instrukce je používán jako operand následující
    - WAW (*Write After Write*, *output dependency*)
      - Dvě instrukce zapisují do stejného registru (výsledek musí odpovídat pozdější instrukci)
    - WAR (*Write After Read*, *anti-dependency*)
      - Zatímco jedna instrukce zpracovává data, další instrukce tato data změní
  - WAW a WAR lze vyřešit přejmenováním registrů
    - Procesor má více fyzických registrů než definuje architektura

Příklad: Eliminace WAW

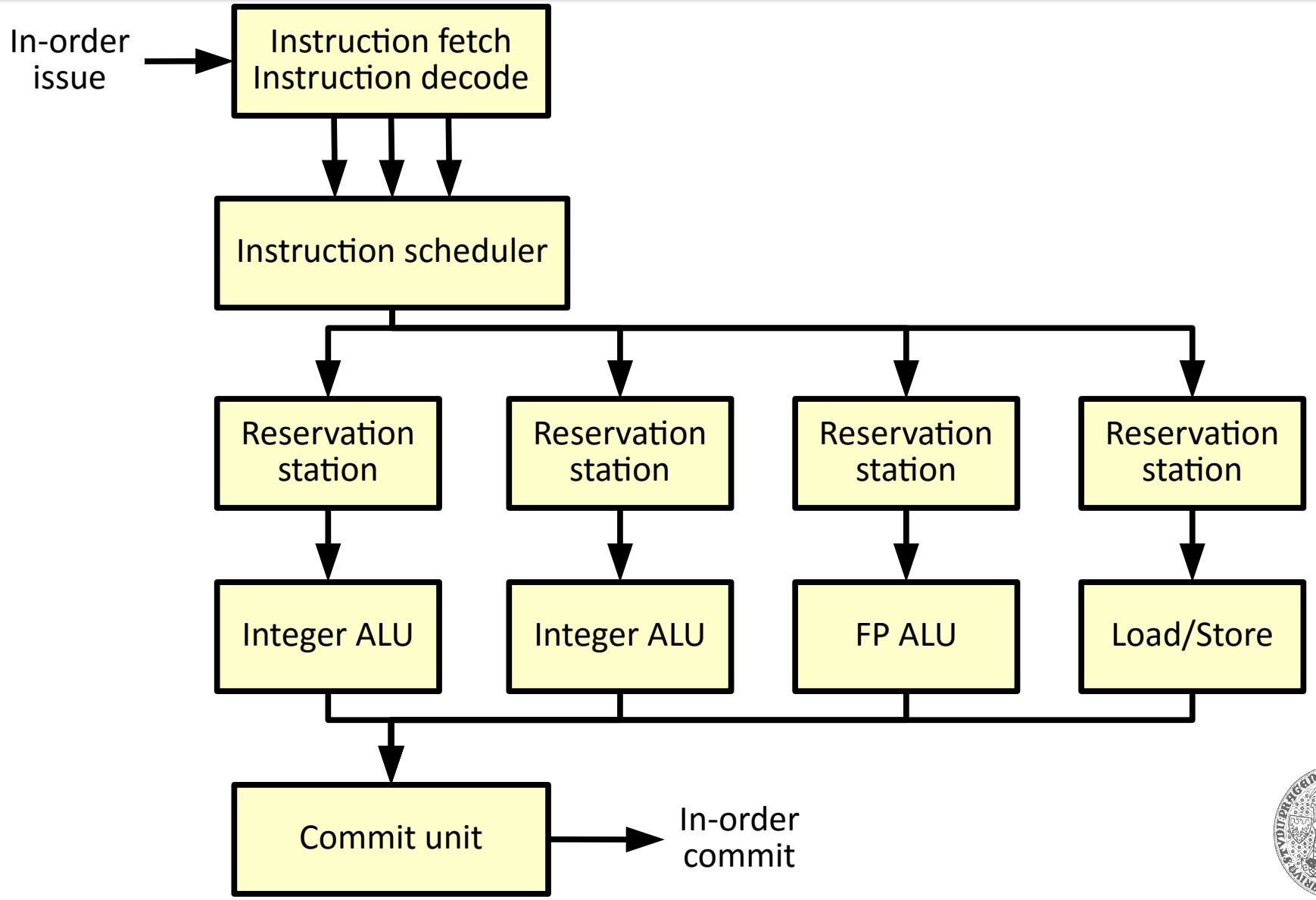
- **Kód z pohledu procesoru (po přerovnání)**

```
move r3, r7
add r3, r4, r5
move r1, r3
```

- **Po přejmenování registrů**

```
move r3, r7
add fr8, r4, r5
move r1, fr8
```

Dynamic multiple issue (2)



Out-of-order a výjimky

- **Komplikovanější než skalární pipeline**
  - Při zpracování mimo pořadí není snadné definovat místo přerušení výpočtu
    - Instrukce následující po instrukci, která vykonala výjimku, nesmí ovlivnit stav stroje
      - Některé takové instrukce už mohly být vykonány
    - Nesmí existovat nezpracované starší instrukce
    - Všechny výjimky způsobené staršími instrukcemi jsou už vyřízeny
  - Přesné (*precise*) vs. nepřesné (*imprecise*) výjimky
    - OOE + register renaming poprvé implementováno v IBM 360/91 (1969), masové nasazení až v 90. letech
    - Důvody: nepřesné výjimky + vyšší efektivita jen pro malou třídu programů

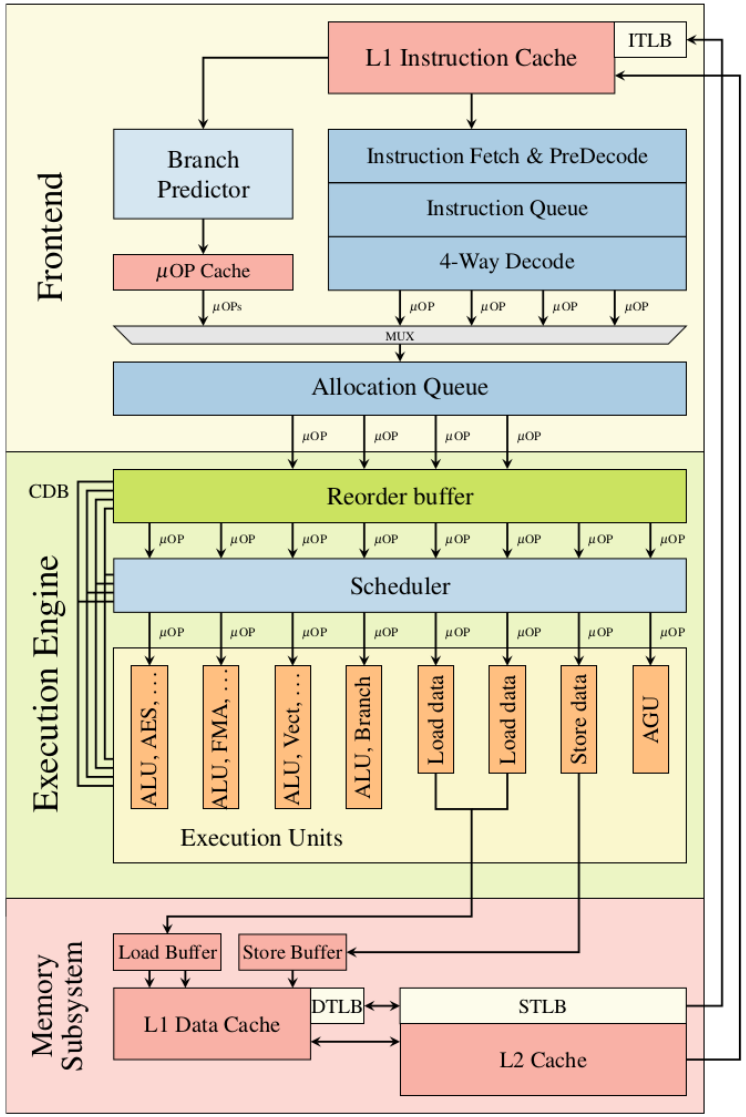
Spekulativní provádění instrukcí

- **Odhadnutí vlastností/výsledku instrukce**
  - Umožňuje zahájení zpracovávání závislých instrukcí
  - Logika pro ošetření nesprávného výsledku
    - Při překladu
      - Speciální kód pro opravu chybných spekulací
    - V procesoru
      - Spekulativní výsledky neovlivňují stav stroje, dokud nejsou potvrzeny
      - Spekulativně provedené instrukce nevyvolávají výjimky nebo vyvolávají speciální výjimky

Příklad: IA-32

- **Intel Pentium Pro až Pentium 4**
  - CISC instrukční sada implementována pomocí mikrooperací na post-RISC jádře
    - Instrukce rozkládány na mikrooperace
    - Pipeline provádí mikrooperace
  - Superskalární, out-of-order spekulativní provádění instrukcí (včetně predikce skoků a přejmenování registrů)
- **Pentium 4**
  - *Trace cache* pro urychlení dekódování instrukce

Příklad: Skylake



- **Zjednodušený pohled na mikroarchitekturu procesorů rodiny Skylake**
  - Instrukce dekódovány do mikrooperací (µOPs)
  - µOPs vykonánány out-of-order pomocí výkonných jednotech v Execution Engine
  - Reorder Buffer zodpovědný za alokaci registrů, přejmenování registrů a dokončení instrukcí
    - Navíc eliminuje přesuny mezi registry a nulovací idiomy
  - Scheduler forwarduje µOPs výkonným jednotkám podle dostupnosti dat
  - Zdroj: M. Lipp et al. Meltdown.

Návrh optimální instrukční sady

- **Statistika využití instrukcí (IBM 360)**

Skupina	Zastoupení
přesuny dat	45,28 %
řízení	28,73 %
aritmetika	10,75 %
porovnávání	5,92 %
logické operace	3,91 %
posuny, rotace	2,93 %
bitové operace	2,05 %
I/O a ostatní	0,43 %

Návrh optimální instrukční sady (2)

- **Další pozorování (IBM 360)**
  - 56 % konstant je v rozsahu ±15 (5 bitů)
  - 98 % konstant je v rozsahu ±511 (10 bitů)
  - 95 % podprogramů potřebuje pro předání argumentů méně než 24 bytů
- **Další pozorování (DEC Alpha)**
  - Typický program používá 58 % instrukční sady
  - Pro 98 % instrukcí stačí 15 % firmwaru (PAL)

Návrh optimální instrukční sady (3)

- **Historické priority**
  - Rozsáhlé instrukční sady, složité operace
  - Překlenutí sémantické mezery mezi assemblerem a vyšším programovacím jazykem
- **Aktuální priority**
  - Malé instrukční sady, jednoduché instrukce
  - Rychlejší provádění instrukcí, snadnější optimalizace (při překladu i při provádění)

Post-RISC procesory

- **Postupná konvergence CISC a RISC architektur**
  - Přidání užitečných složitějších CISCových instrukcí do RISCové instrukcní sady
  - Superskalární zpracování
  - Agresivní přerovnávání instrukcí při zpracování
    - Out-of-order speculative execution
    - Odklon od závislosti na optimalizacích překladače
  - Nové specializované jednotky, větší míra paralelismu