

# Principy počítačů a operačních systémů

Mikroarchitektura procesoru MIPS

Zimní semestr 2007/2008

# Hlavní části mikroarchitektury (1)

---

## Datová cesta

- uspořádání funkčních bloků umožňující zpracovávat instrukce a data uvnitř procesoru

## Návrh datové cesty

- vychází z cílové instrukční sady
- identifikace prvků datové cesty
- propojení prvků a návrh řízení

# Procesor MIPS (1)

## Registry

- 32 general-purpose registrů (r0-r31)
  - ♦ s0-s7, t0-t9, zero, a0-a3, v0-v1, gp, fp, sp, ra, at

## Operace

- aritmetika registr/registr, registr/immediate
- přesuny registr/registr, registr/paměť
  - ♦ load/store architektura
- nepodmíněné skoky, skoky do podprogramu
  - ♦ HW neimplementuje zásobník
- speciální instrukce

# Procesor MIPS (2)

## Základní typy instrukcí

- základní formát (32-bitů)

6b	5b	5b	5b	5b	6b
----	----	----	----	----	----

- r-format (aritmetické instrukce s registry)

op	rs	rt	rd	shamt	funct
----	----	----	----	-------	-------

- i-format (přesuny, větvení, přímé operandy)

op	rs	rt	address/immediate
----	----	----	-------------------

- j-format (nepodmíněný skok)

op	target address
----	----------------

# MIPS: Návrh datové cesty

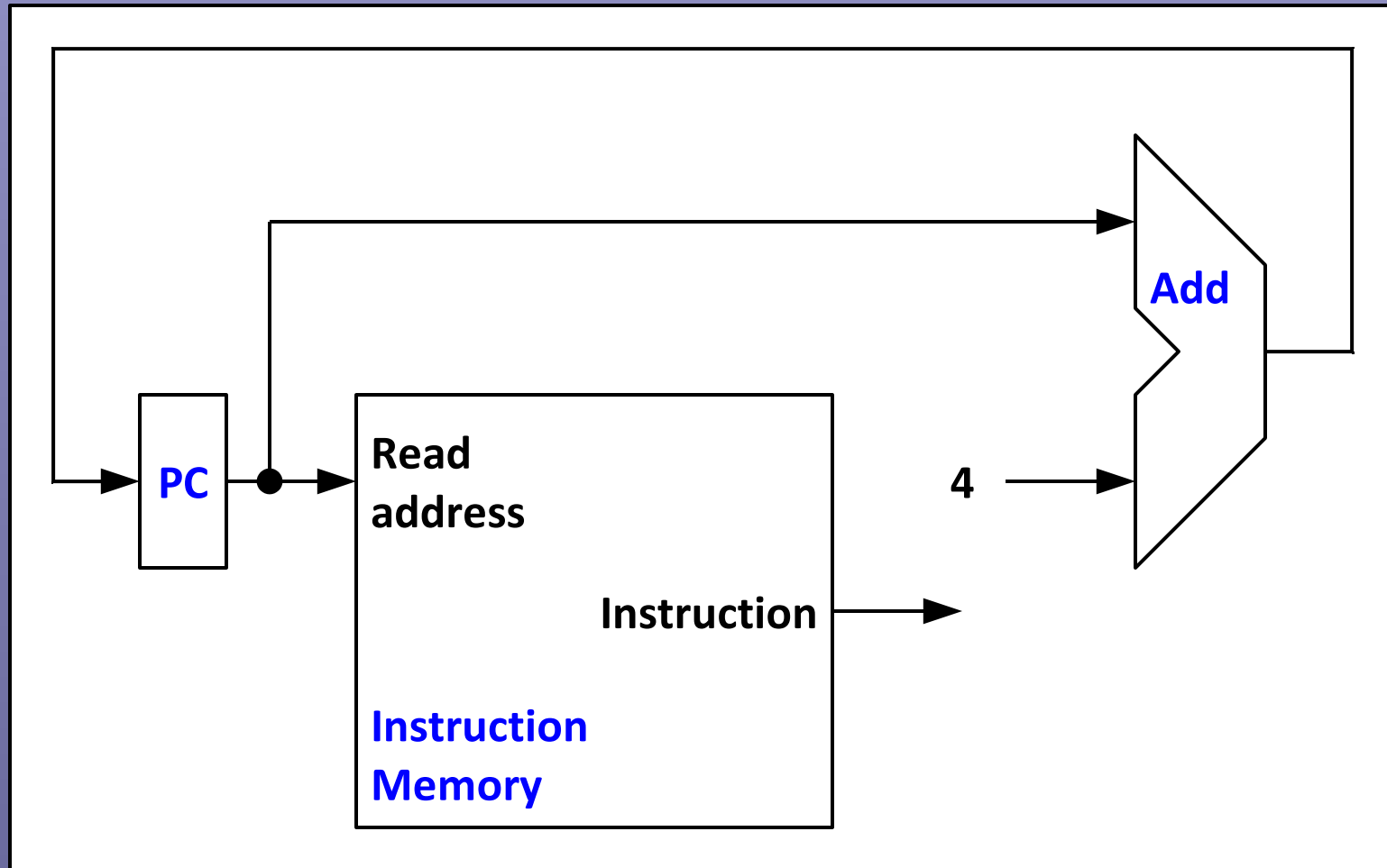
---

## Obecný postup zpracování instrukcí

- přečíst instrukci z paměti
- přečíst 1 či 2 registry
- provést operaci odpovídající instrukčnímu kódu
  - ♦ přečíst/zapsat registry z/do paměti
  - ♦ provést operaci s registry
  - ♦ podmíněný/nepodmíněný skok

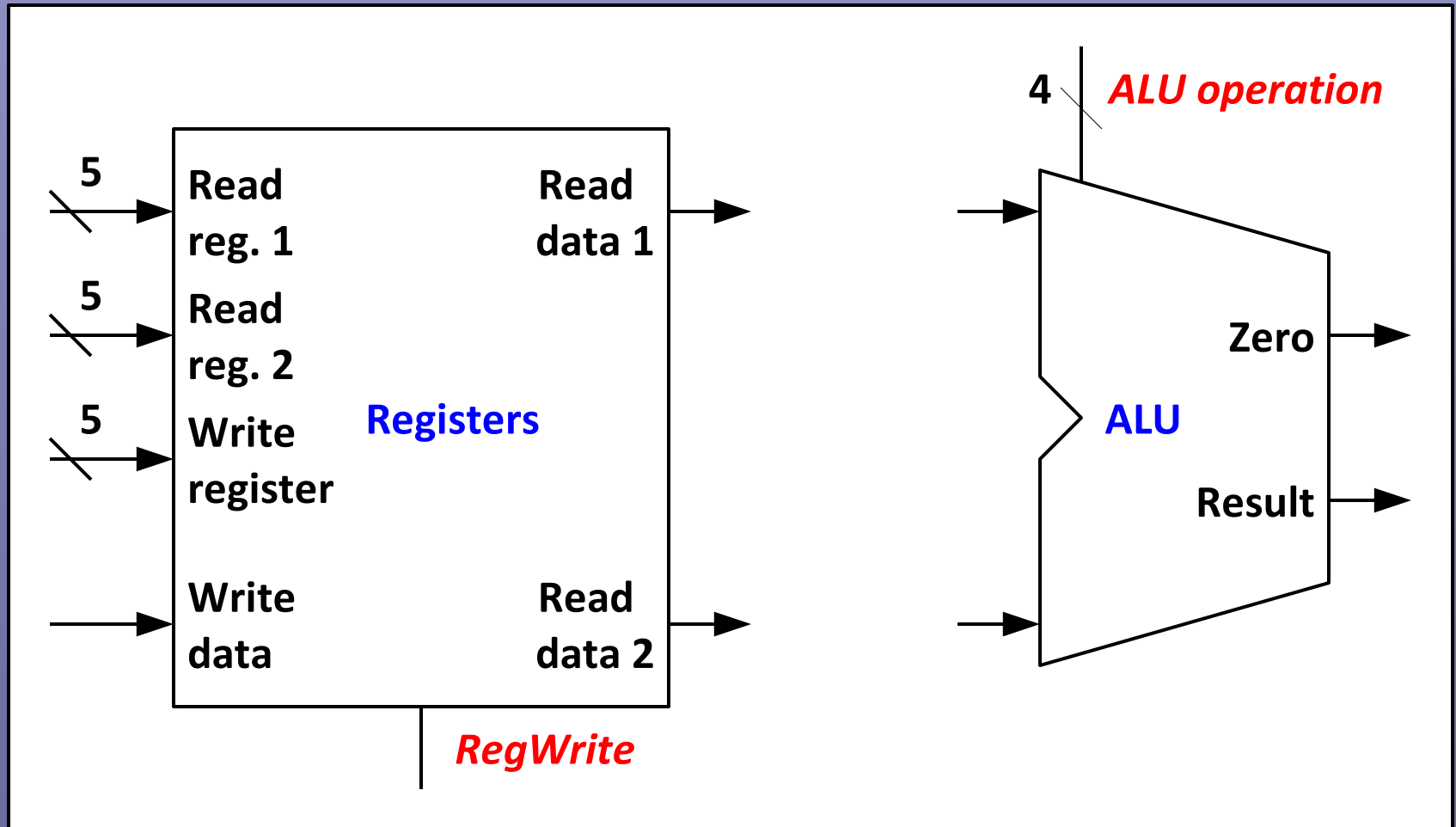
# MIPS: Prvky datové cesty (1)

## Čtení instrukce



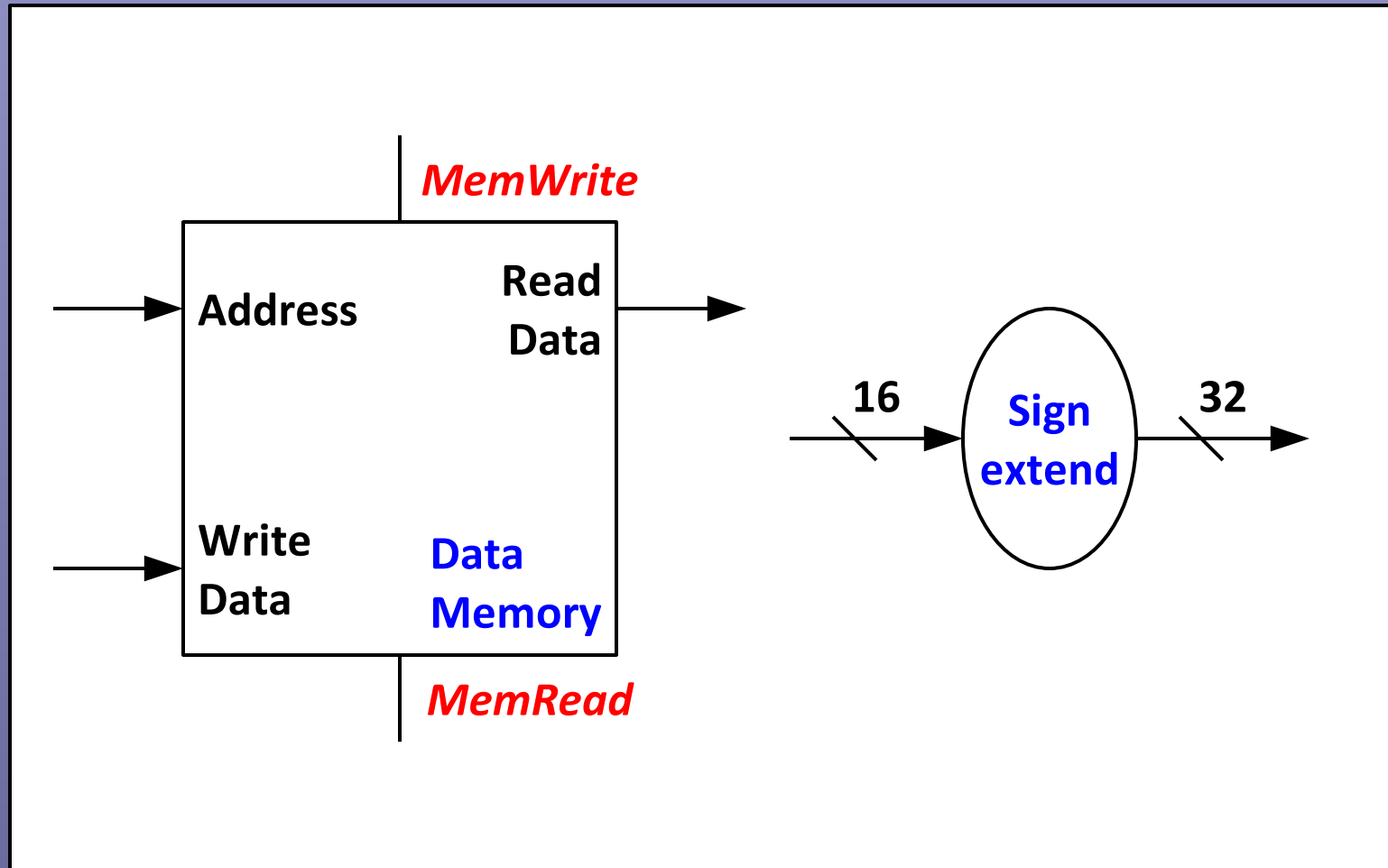
# MIPS: Prvky datové cesty (2)

## Operace s daty v registrech



# MIPS: Prvky datové cesty (3)

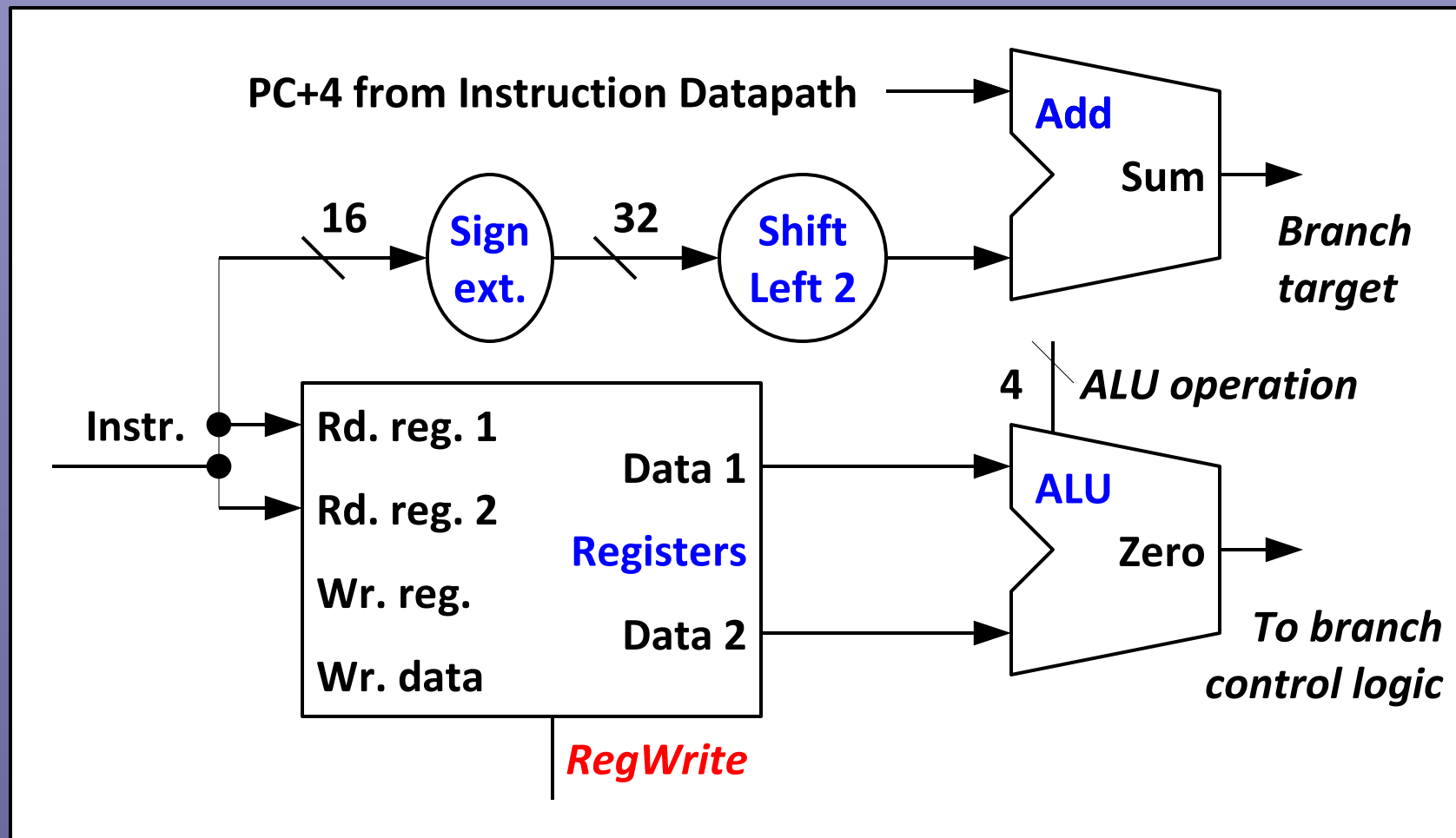
Čtení/zápis z/do paměti



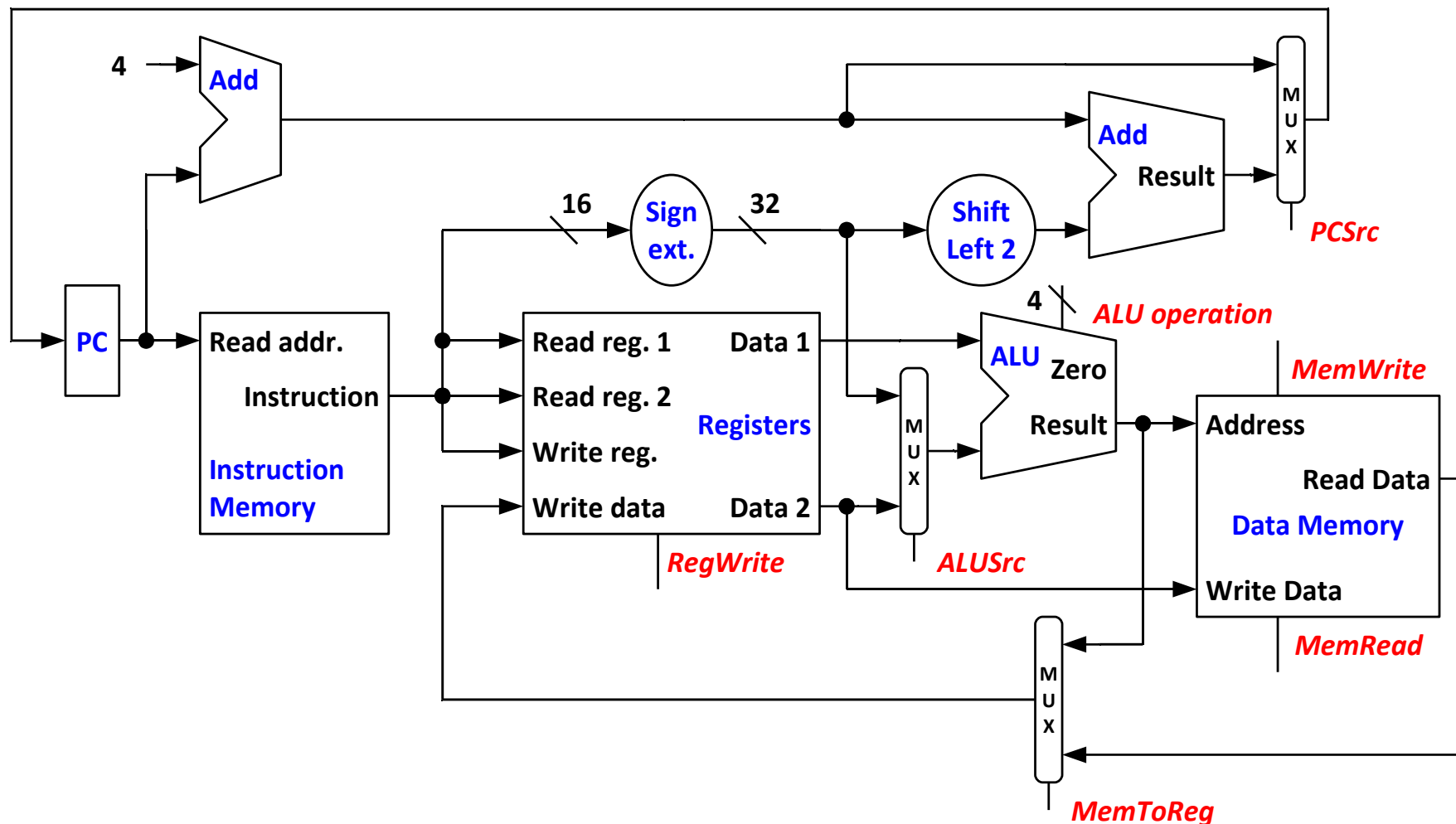


# MIPS: Prvky datové cesty (4)

## Podmíněný skok



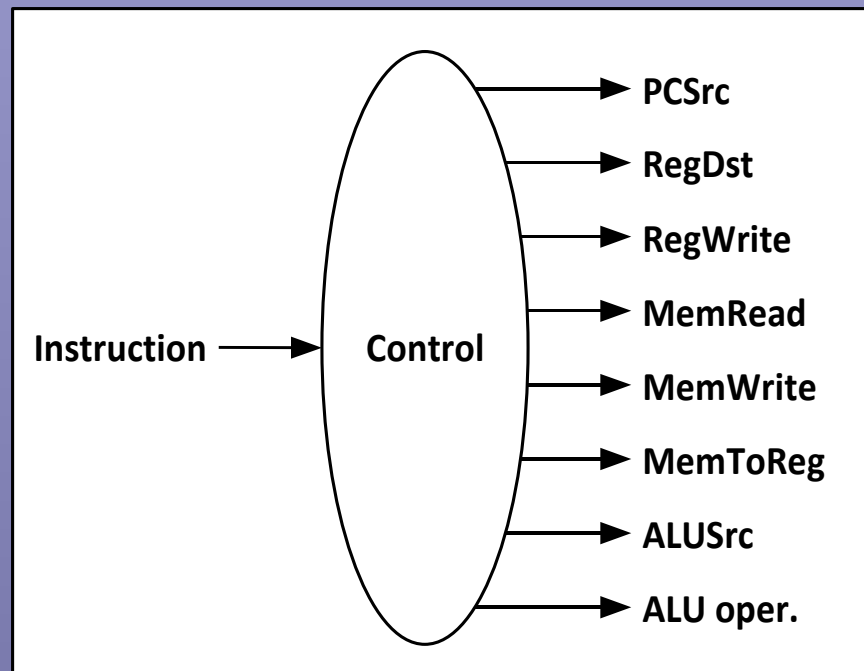
# MIPS: Datová cesta



# MIPS: Řízení datové cesty

## Řídící signály

- programový čítač
- registrové pole
- paměť pro data
- ALU



## Podporované instrukce

- load word, store word, branch equal
- add/subtract, logical and/or, set on less than

# Návrh řízení datové cesty (1)

## Kombinační obvod

- nastavuje řídicí signály datové cesty
- dekodér z instrukčního kódu

## Formát instrukčního kódu

- významně ovlivňuje složitost a rychlost dekodéru
  - ♦ pravidelnost, symetrie

## Víceúrovňové dekódování

- rozdělení řídicího obvodu do více bloků
  - ♦ podřízený blok pro řízení ALU
- zjednodušení návrhu řídicí části

# Řadič ALU

3 režimy práce v závislosti na instrukci

- add, subtract, funct

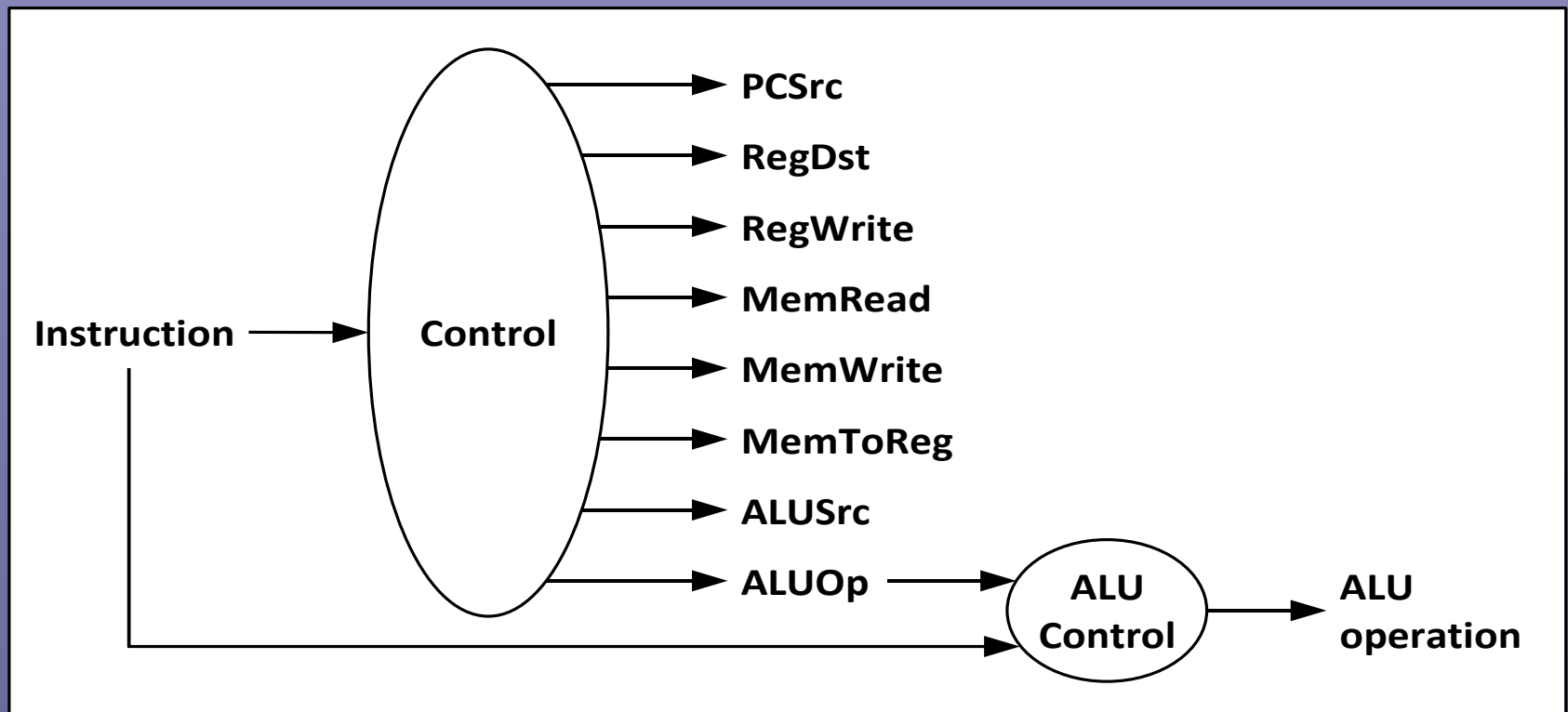
op	rs	rt	rd	shamt	funct
op	rs	rt	address/immediate		
op	target address				

Operační kód	Operace	Funct						Požadovaná operace ALU	ALU control	ALUOp	
		F5	F4	F3	F2	F1	F0			Op1	Op2
LW	load word	X	X	X	X	X	X	add	0010	0	0
SW	store word	X	X	X	X	X	X	add	0010	0	0
BEQ	branch equal	X	X	X	X	X	X	subtract	0110	X	1
R-TYPE	add	1	0	0	0	0	0	add	0010		
R-TYPE	subtract	1	0	0	0	1	0	subtract	0110		
R-TYPE	AND	1	0	0	1	0	0	and	0000	1	X
R-TYPE	OR	1	0	0	1	0	1	or	0001		
R-TYPE	set on less than	1	0	1	0	1	0	set on less than	0111		

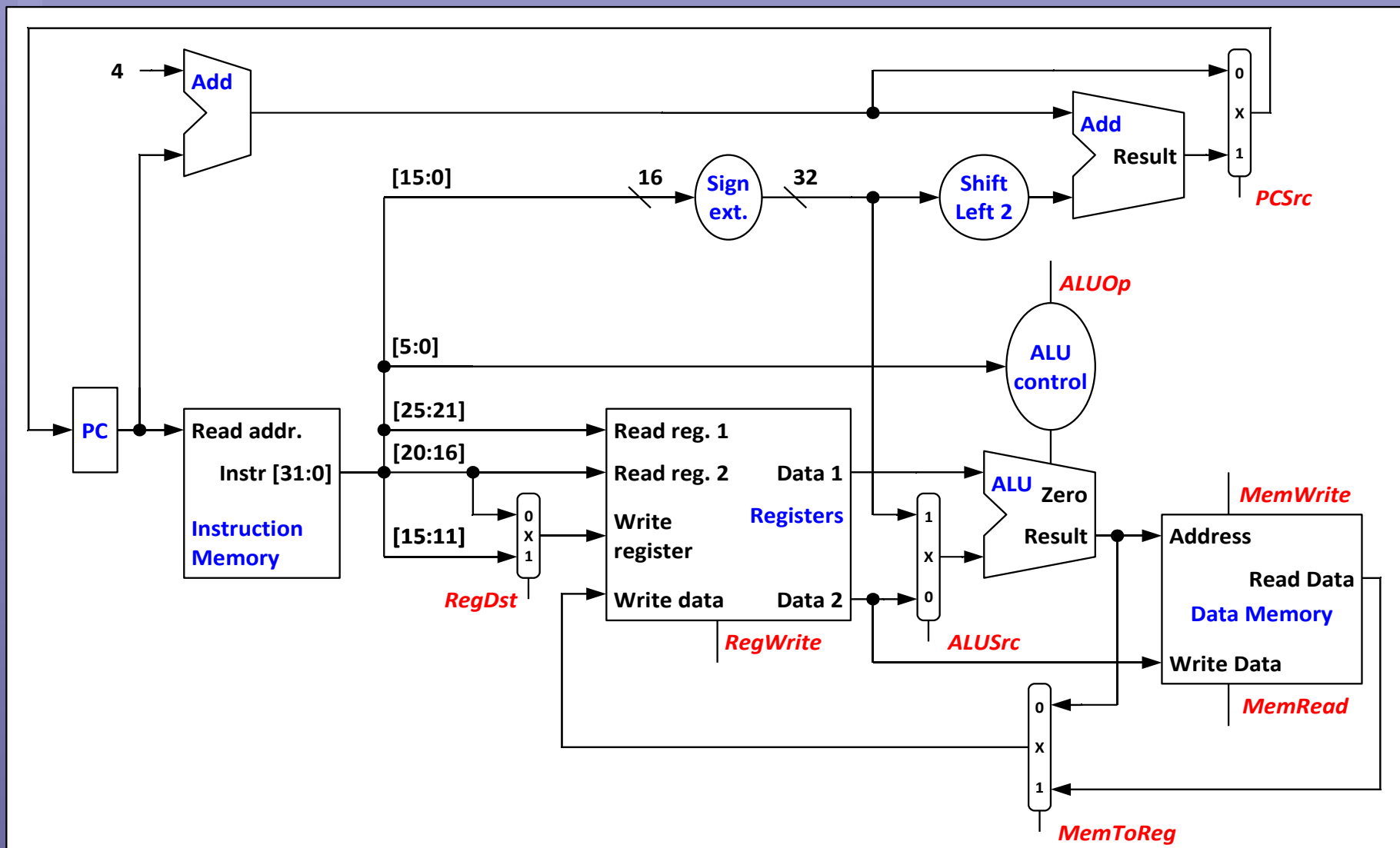
# Víceúrovňové dekódování

## Řadič ALU

- zjednodušení řízení ALU ze 4 bitů na 2
  - ♦ podle typu instrukce, funkci nastavuje ALU Control



# MIPS: Datová cesta s řadičem ALU



# Návrh hlavního řadiče (1)

## Dekódování instrukce

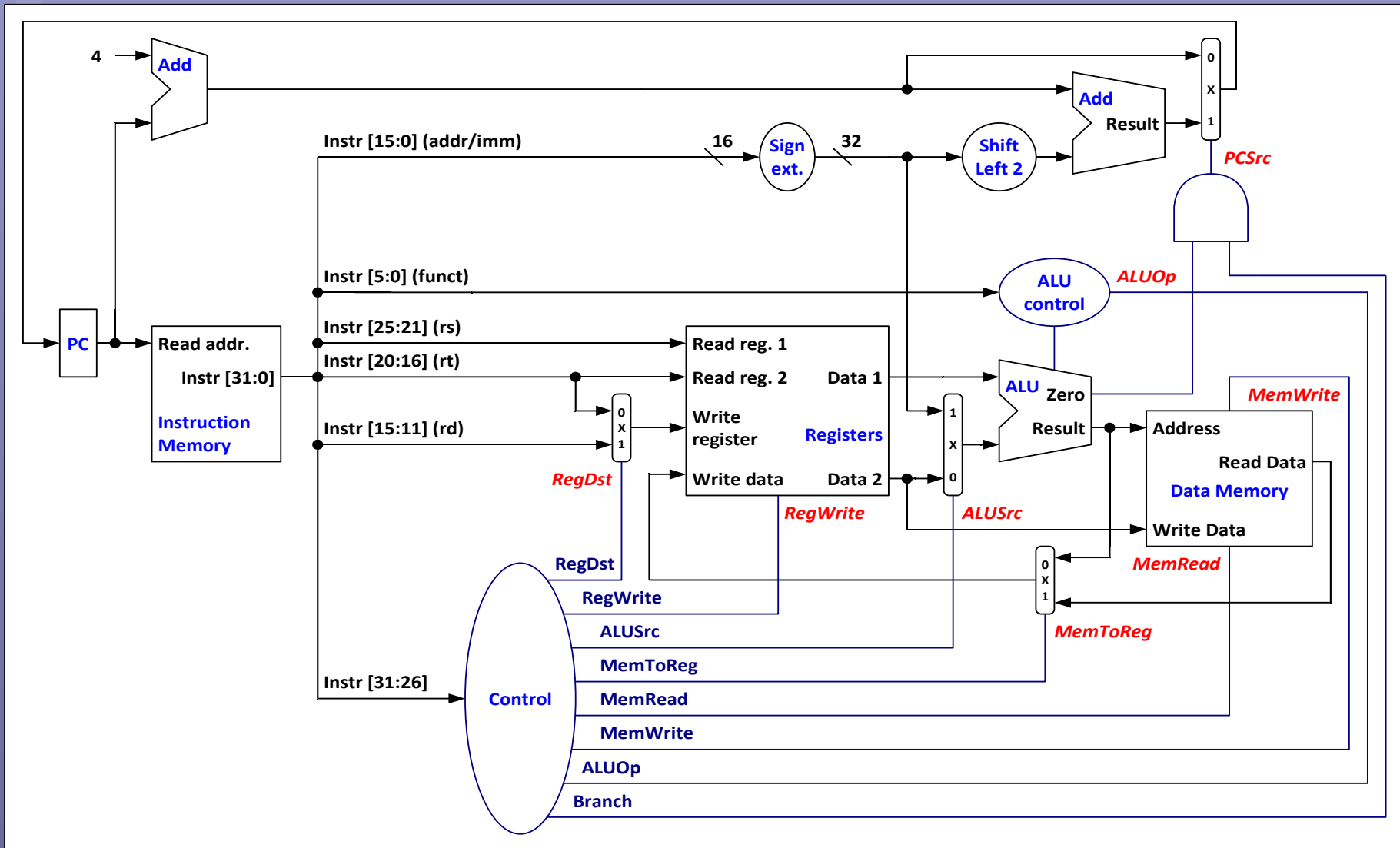
op	rs	rt	rd	shamt	funct
op	rs	rt	address/immediate		
op	target address				

- řídicí signály plně určeny operačním kódem

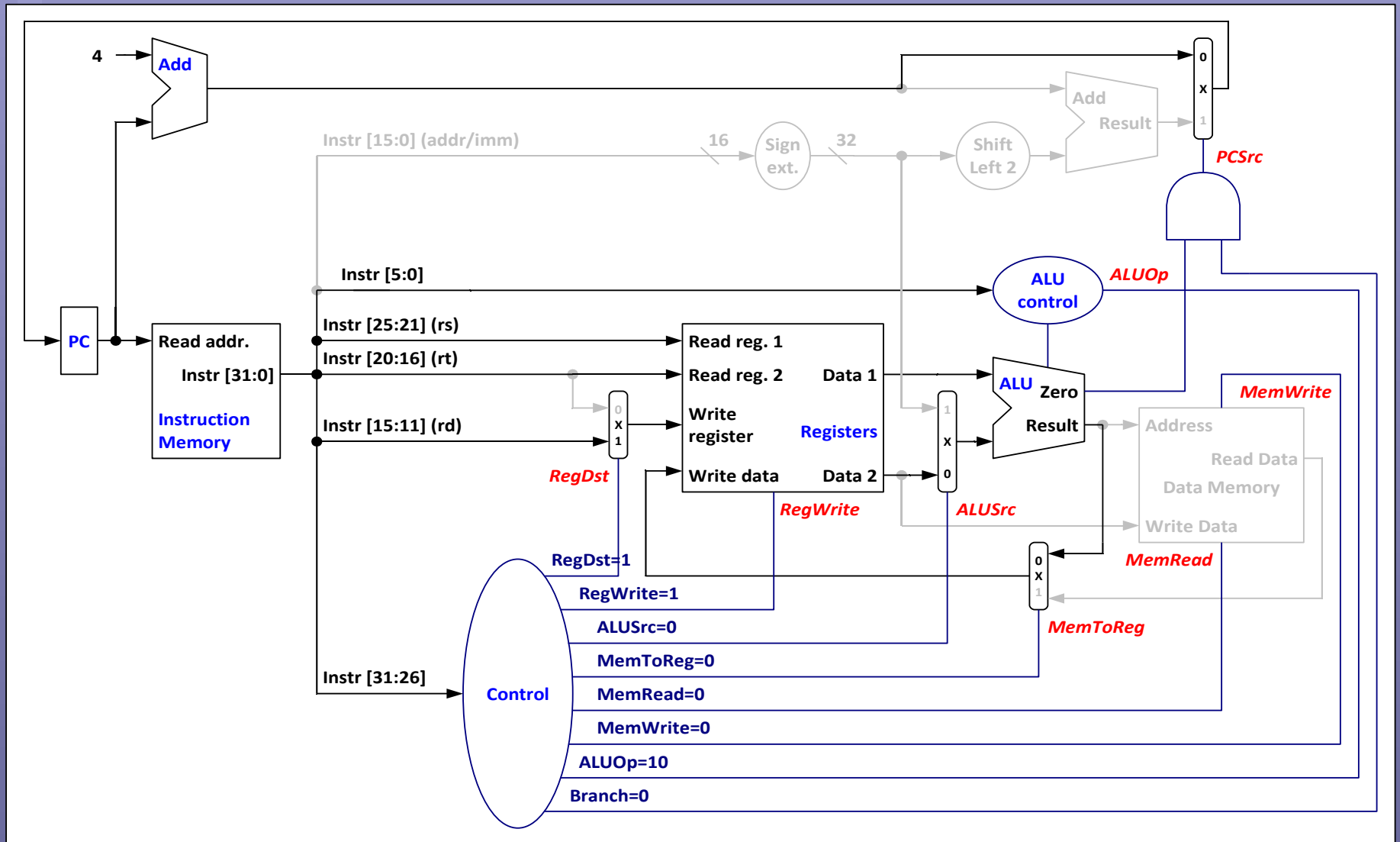
Instrukce	RegDst	ALUSrc	MemToReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp2
R-format	1	0	0	1	0	0	0	1	0
LW	0	1	1	1	1	0	0	0	0
SW	X	1	X	0	0	1	0	0	0
BEQ	X	0	X	0	0	0	1	0	1



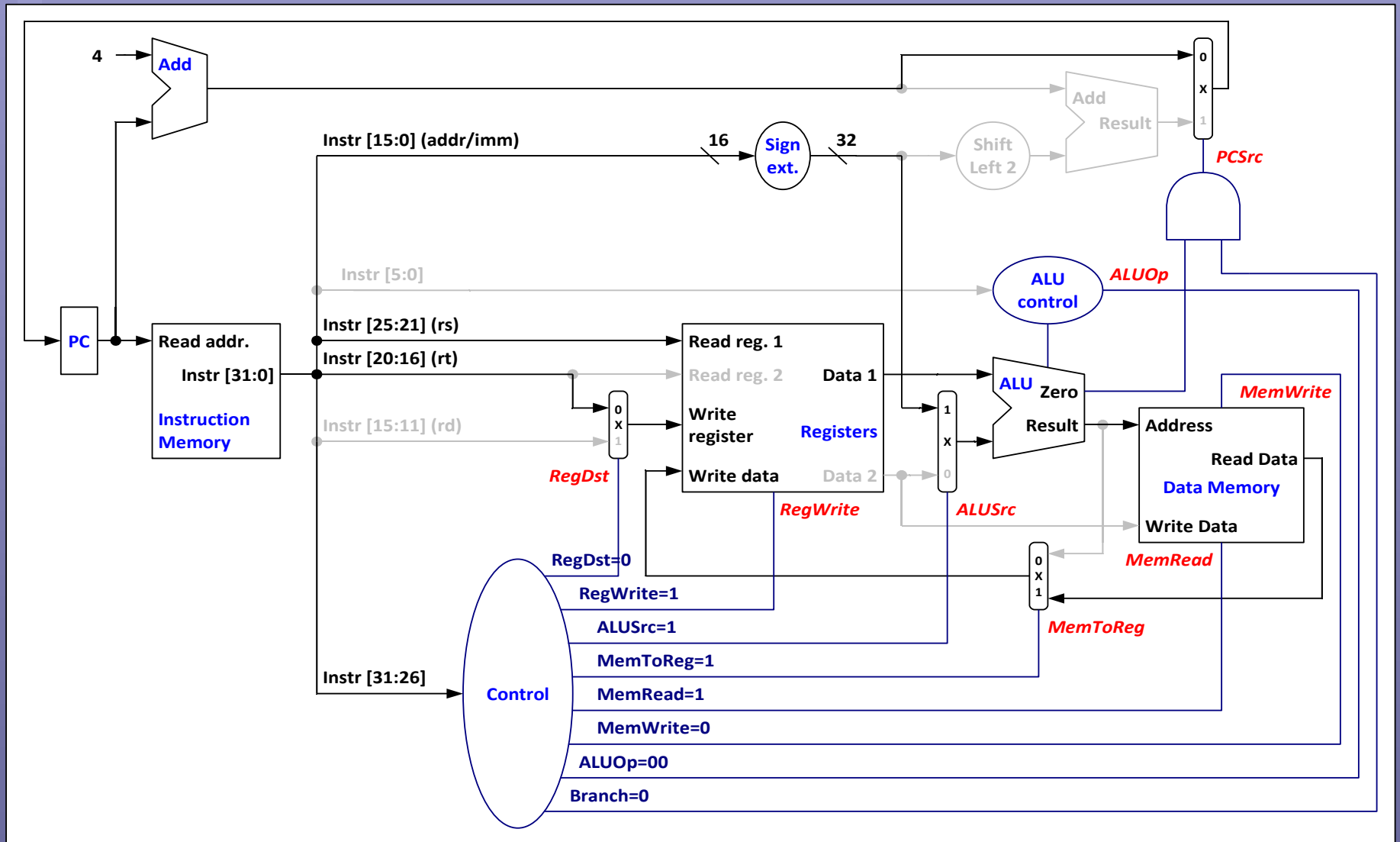
# MIPS: Datová cesta s hlavním řadičem



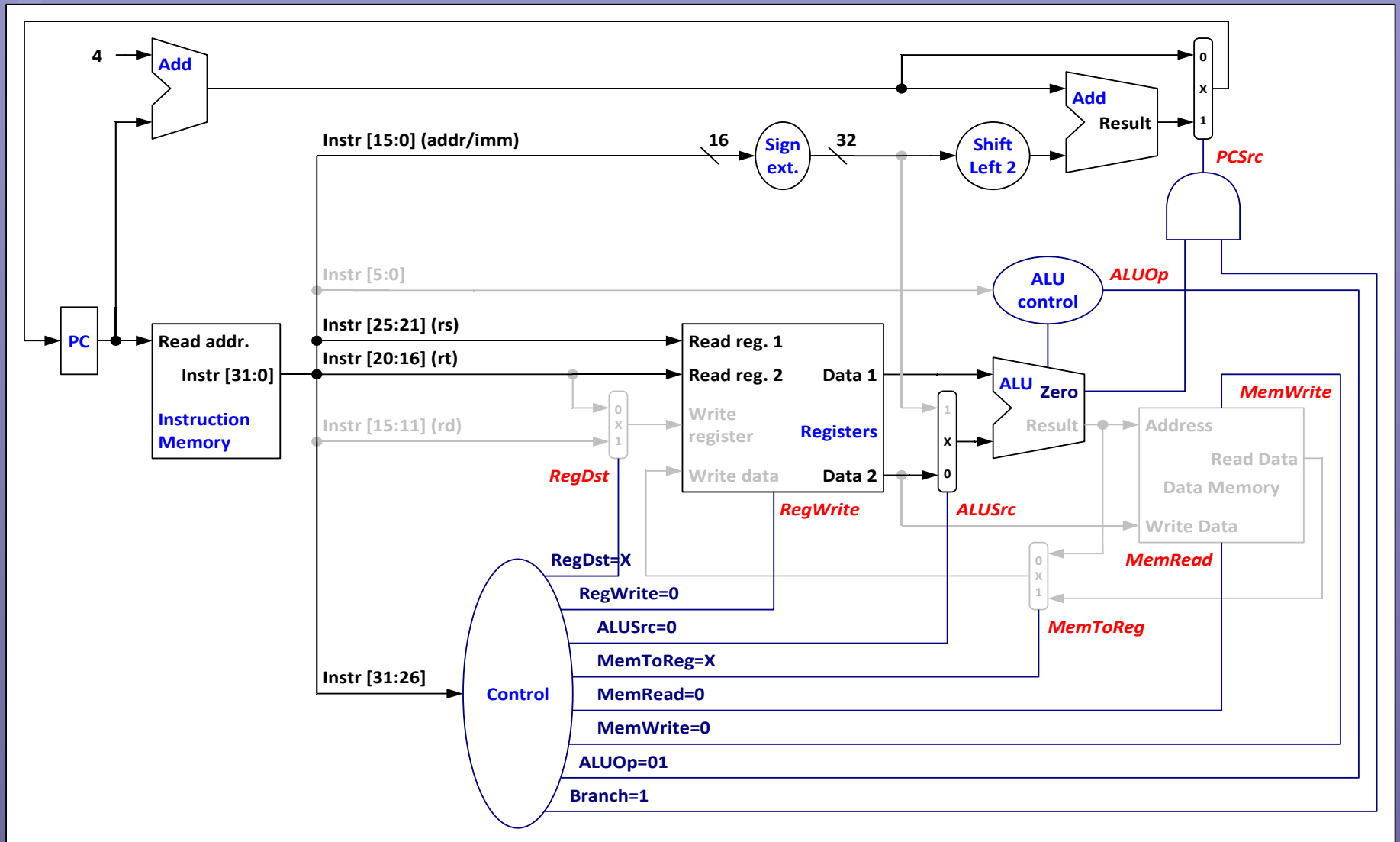
# MIPS: Instrukce typu R-format



# MIPS: Instrukce Load Word



# MIPS: Instrukce Branch Equal



# Návrh hlavního řadiče (2)

## Jednocyklový radič

- instrukce trvá 1 takt
- kombinační obvod

## Hlavní nevýhody

- délka cyklu odpovídá délce nejdelší instrukce
  - ♦ ve sporu s “optimize for common case”
- duplicitní prvky v datové cestě

Vstup nebo výstup	Signál	R-format	LW	SW	BEQ
Vstupy	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Výstupy	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemToReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp2	0	0	0	1

# Více-cyklový radič

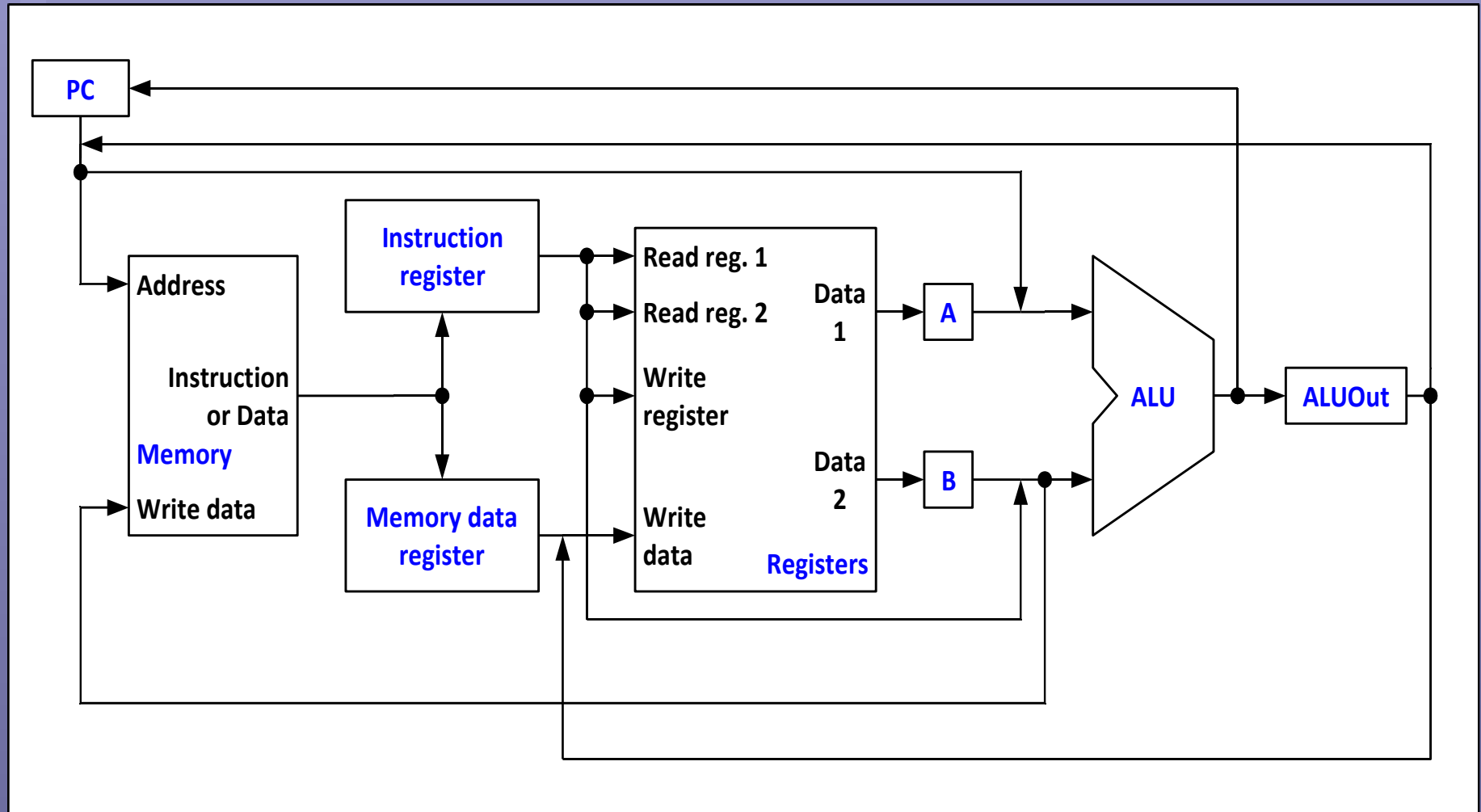
## Základní princip

- instrukce rozdělena do kroků
- v každém taktu proveden 1 krok
  - ♦ počet taktů se pro různé instrukce liší
  - ♦ instrukční cyklus vs. strojový cyklus
- nutno uschovávat mezivýsledky

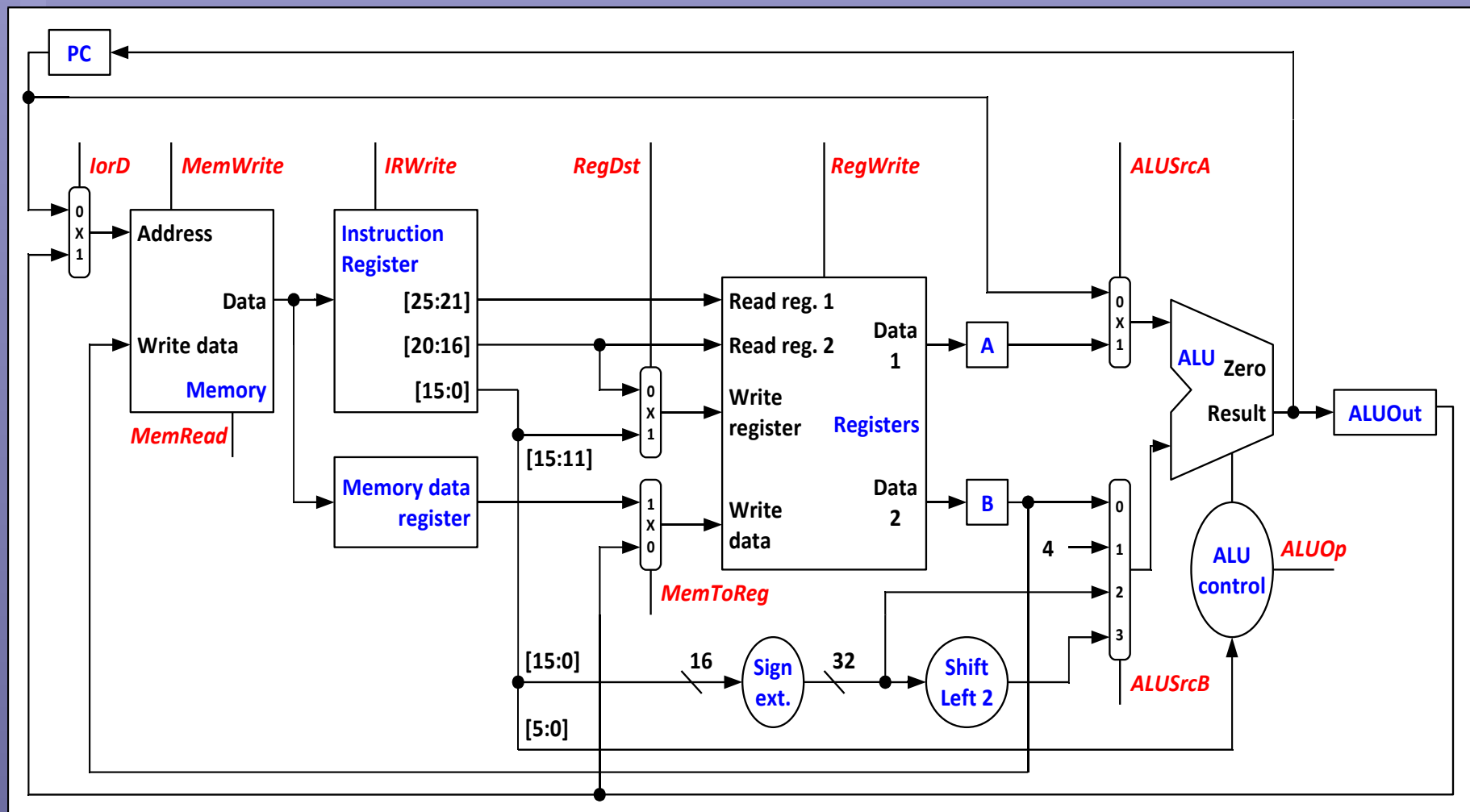
## Hlavní výhody

- vyšší výkon a efektivita
  - ♦ aproximace proměnné délky cyklu
  - ♦ není nutné duplikovat některé funkční prvky

# MIPS: Princip více-cyklové datové cesty



# MIPS: Více-cyklová datová cesta





# Návrh více-cyklové datové cesty

## Rozdělení instrukcí do kroků

- sekvenční a paralelní části vykonání instrukce

## Instrukční cyklus (MIPS)

- načtení instrukce
- dekódování instrukce a přečtení registrů
- vykonání instrukce, výpočet adresy, dokončení větvení
- přístup do paměti a zapsání výsledku
- dokončení čtení z paměti

# MIPS: Instrukční cyklus (1)

## Načtení instrukce

- $IR \leq \text{Memory}[PC]$ 
  - ♦ přečtení instrukce do instrukčního registru
- $PC \leq PC + 4$ 
  - ♦ posun PC na adresu další instrukce

# MIPS: Instrukční cyklus (2)

## Dekódování instrukce a přečtení registrů

- $A \leq \text{Reg}[\text{IR.rs}]$ 
  - ♦ přečtení obsahu zdrojového registru 1
- $B \leq \text{Reg}[\text{IR.rt}]$ 
  - ♦ přečtení obsahu zdrojového registru 2
- $\text{ALUOut} \leq \text{PC} + (\text{sign-extend}(\text{IR.addr}) \ll 2)$ 
  - ♦ výpočet adresy podmíněného skoku
  - ♦ pokud instrukce není skok, nevadí

# MIPS: Instrukční cyklus (3)

Vykonání, výpočet adresy, dokončení větvení

- Přístup do paměti
  - ♦  $ALUOut \leq A + \text{sign-extend}(IR.addr)]$
  - ♦ obsah zdrojového registru + offset
- Aritmeticko-logická operace
  - ♦  $ALUOut \leq A \text{ funct } B$
- Podmíněný skok
  - ♦ if (A == B) then  $PC \leq ALUOut$
  - ♦ adresa skoku z předchozího kroku
- Nepodmíněný skok

# MIPS: Instrukční cyklus (4)

## Přístup do paměti, zápis výsledku

- Přístup do paměti (load)
  - ♦  $MDR \leftarrow Memory[ALUOut]$
  - ♦ obsah paměti přečten do pomocného registru
- Přístup do paměti (store)
  - ♦  $Memory[ALUOut] \leftarrow B$
  - ♦ obsah registru zapsán do paměti
- Aritmeticko-logická operace
  - ♦  $Reg[IR.rd] \leftarrow ALUOut$
  - ♦ výsledek operace zapsán do cílového registru

# MIPS: Instrukční cyklus (5)

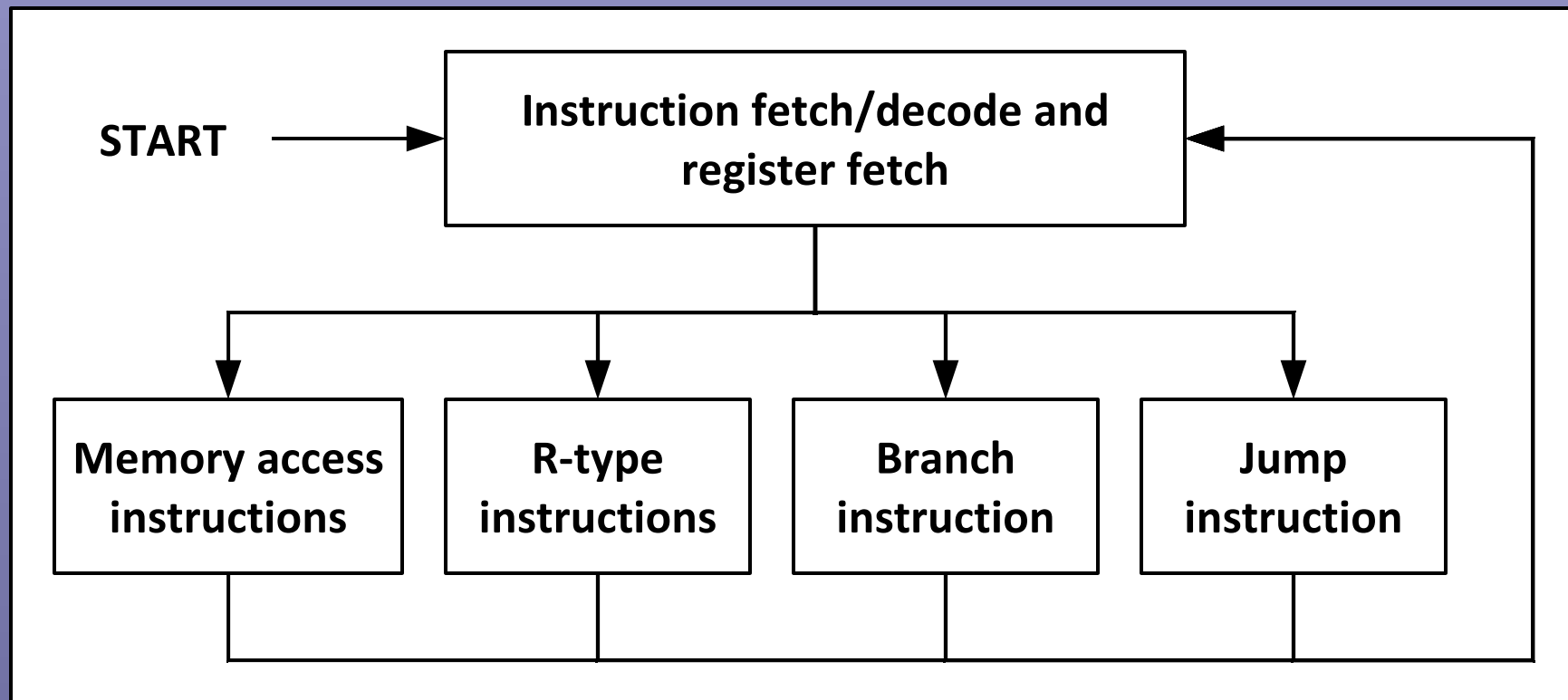
---

## Dokončení čtení z paměti

- Přístup do paměti (load)
  - ♦  $\text{Reg}[\text{IR.rt}] \leq \text{MDR}$
  - ♦ zápis přečtené hodnoty do registru

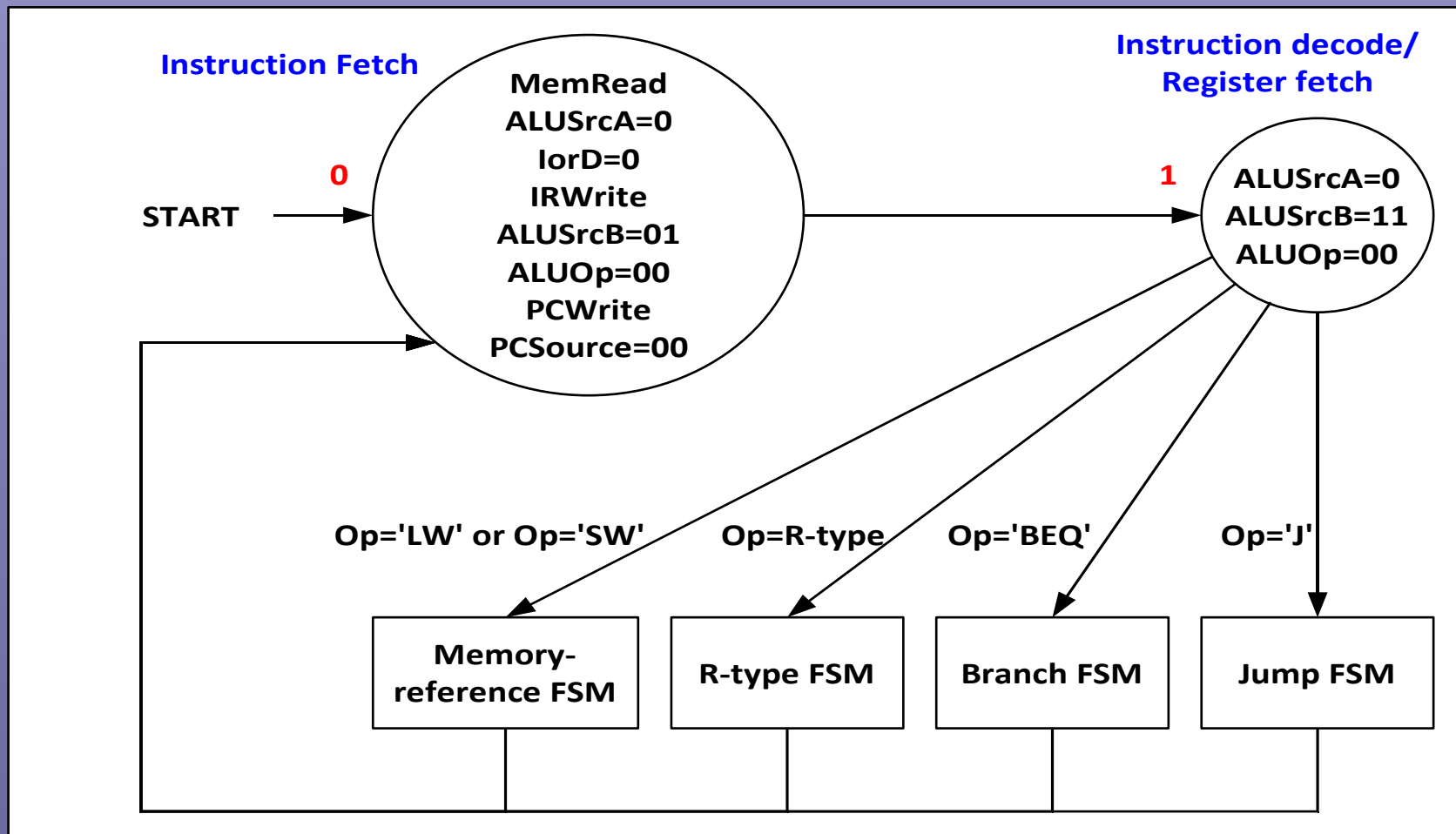
# MIPS: Řadič jako konečný automat (1)

## Hrubé schéma



# MIPS: Řadič jako konečný automat (2)

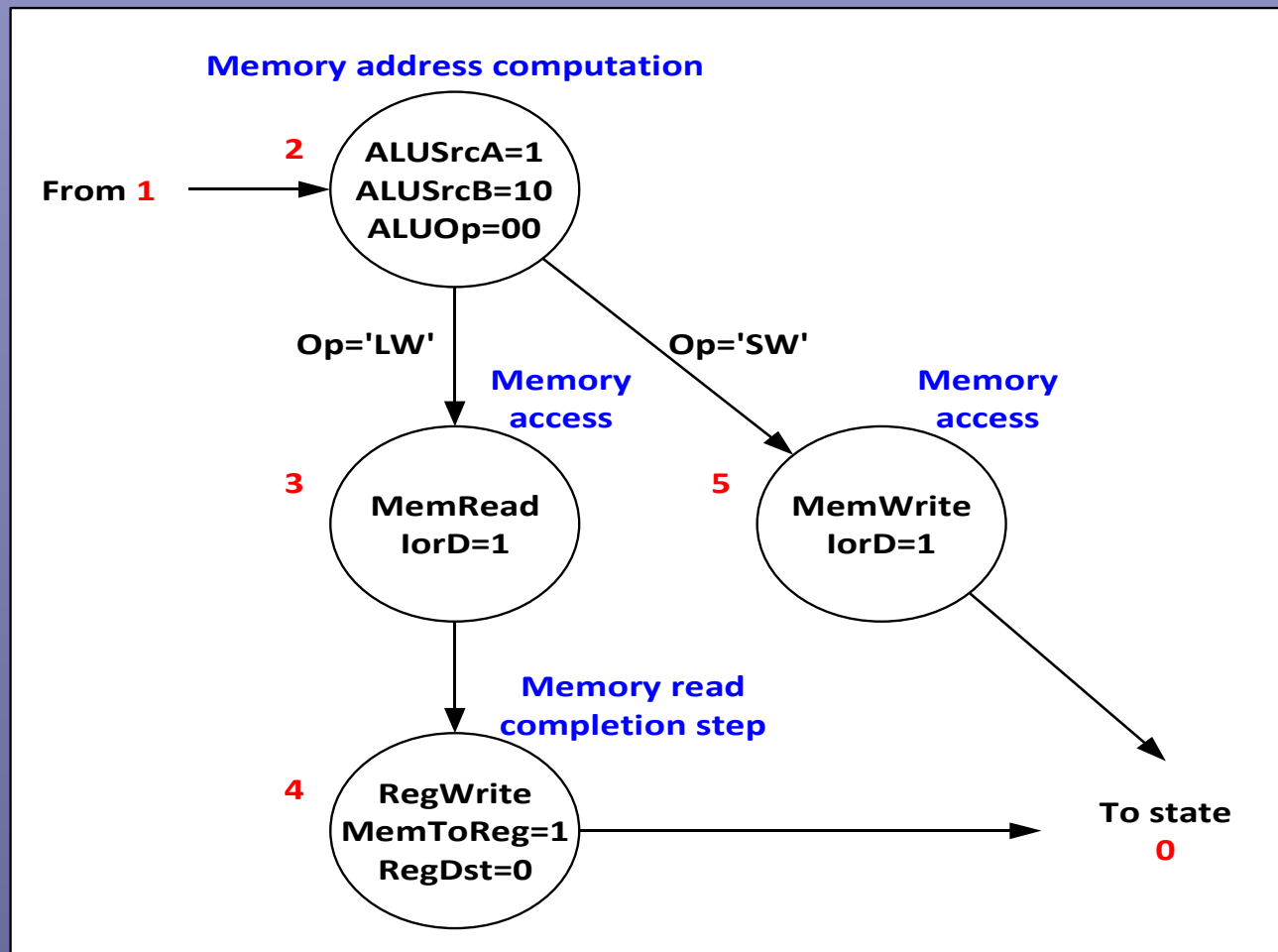
## Instruction fetch & decode





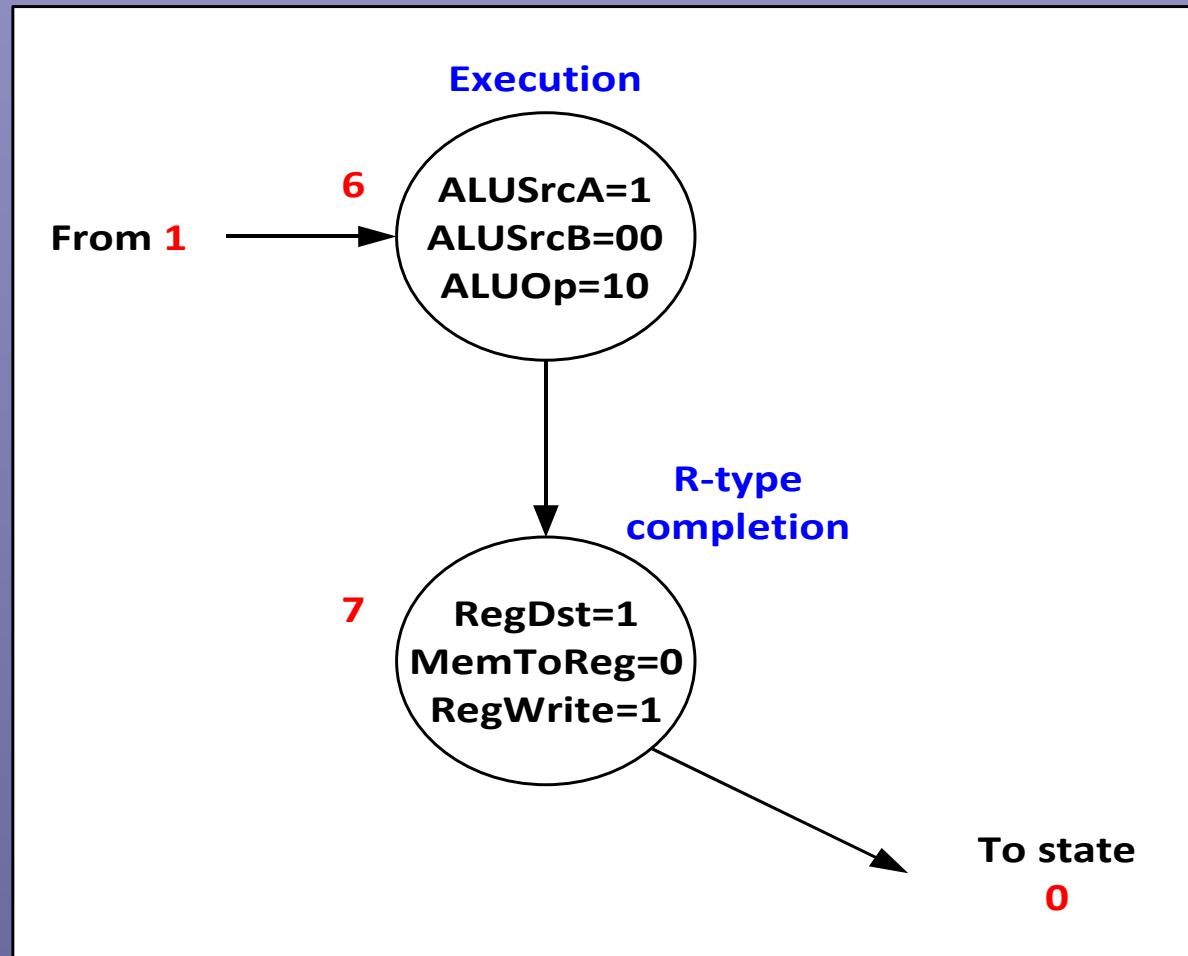
# MIPS: Řadič jako konečný automat (3)

## Memory reference FSM



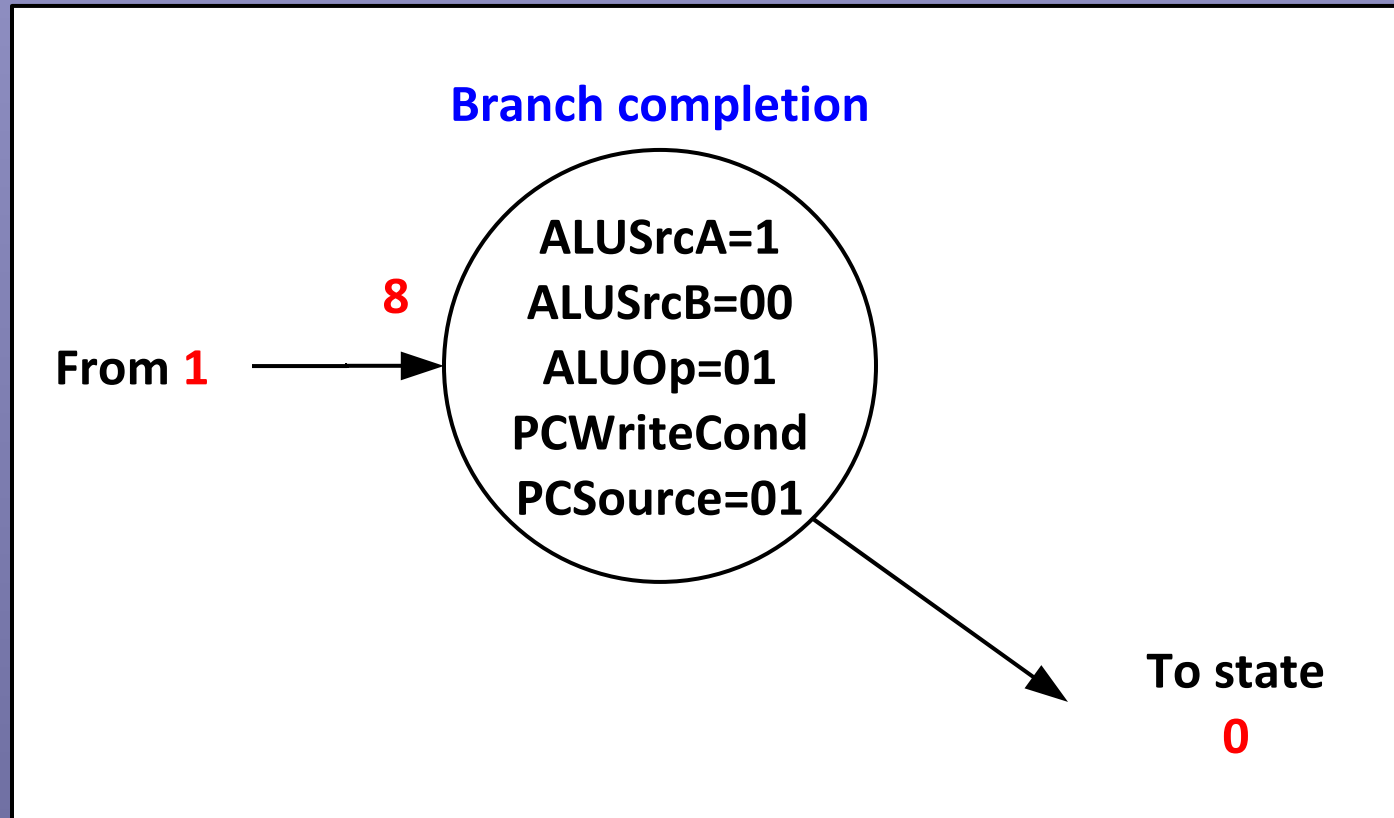
# MIPS: Řadič jako konečný automat (4)

## R-type instruction



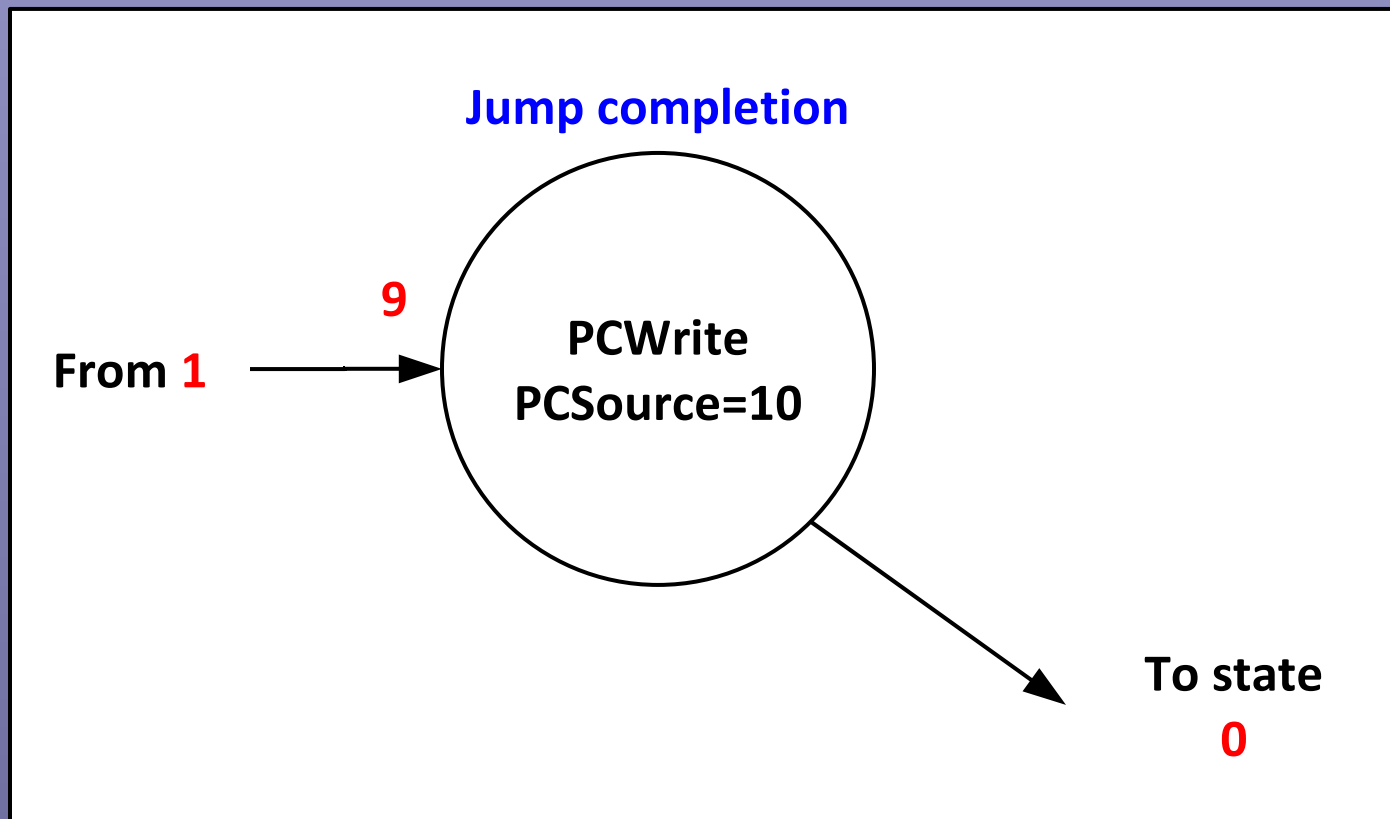
# MIPS: Řadič jako konečný automat (5)

## Branch instruction

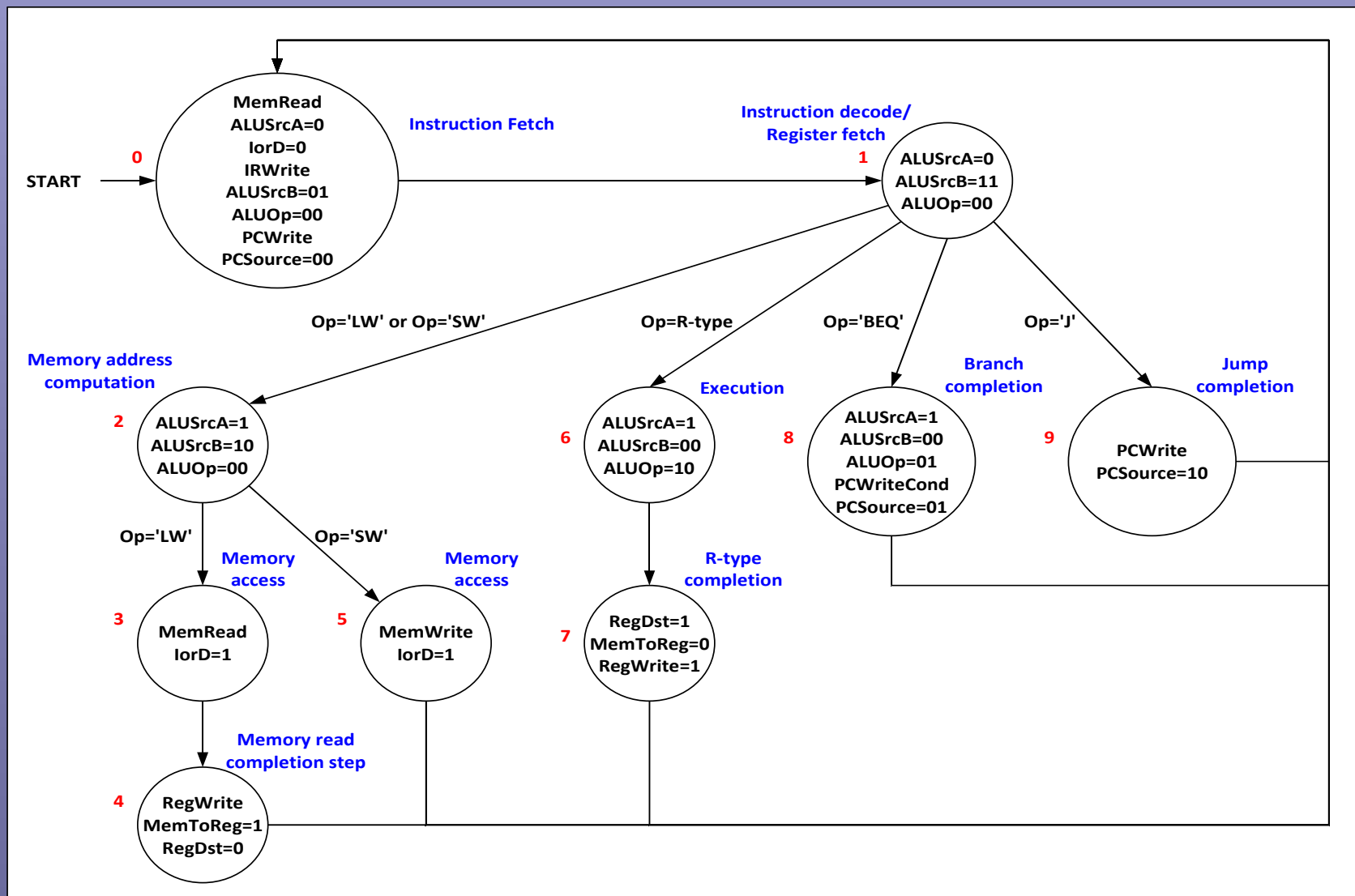


# MIPS: Řadič jako konečný automat (6)

## Jump instruction



# MIPS: Řadič jako konečný automat (7)



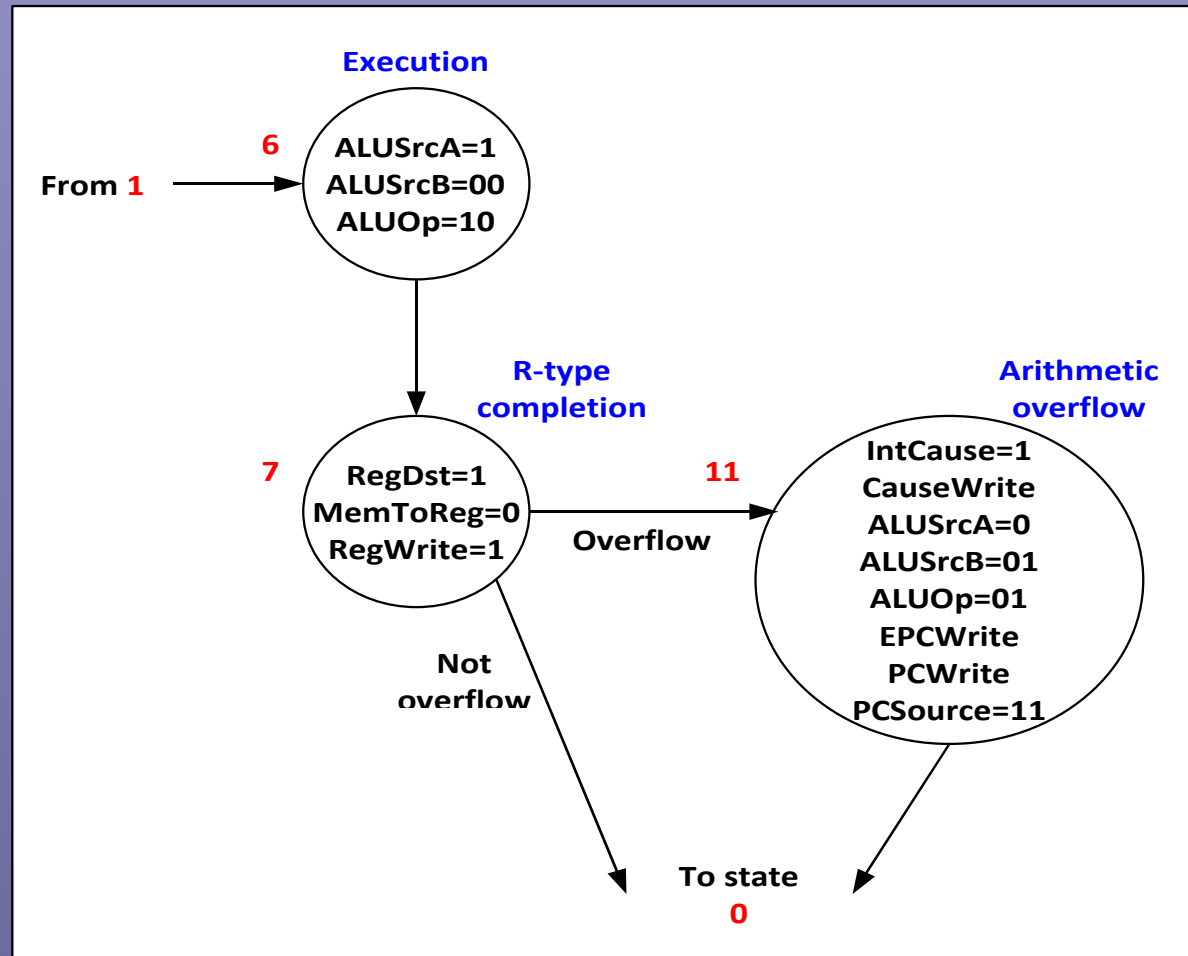
# Podpora výjimek a přerušení

## Hardware

- zastavení vykonávání instrukce
  - ♦ důležité je zachovat korektní stav procesoru
- zajistit možnosti identifikace příčiny
  - ♦ příznak indikujícího příčinu
  - ♦ případně další upřesňující informace
- uschovat adresu instrukce, při které výjimka nastala
- skok na adresu obslužné rutiny
  - ♦ stejná adresa pro všechny typy výjimek
  - ♦ různé adresy pro různé výjimky

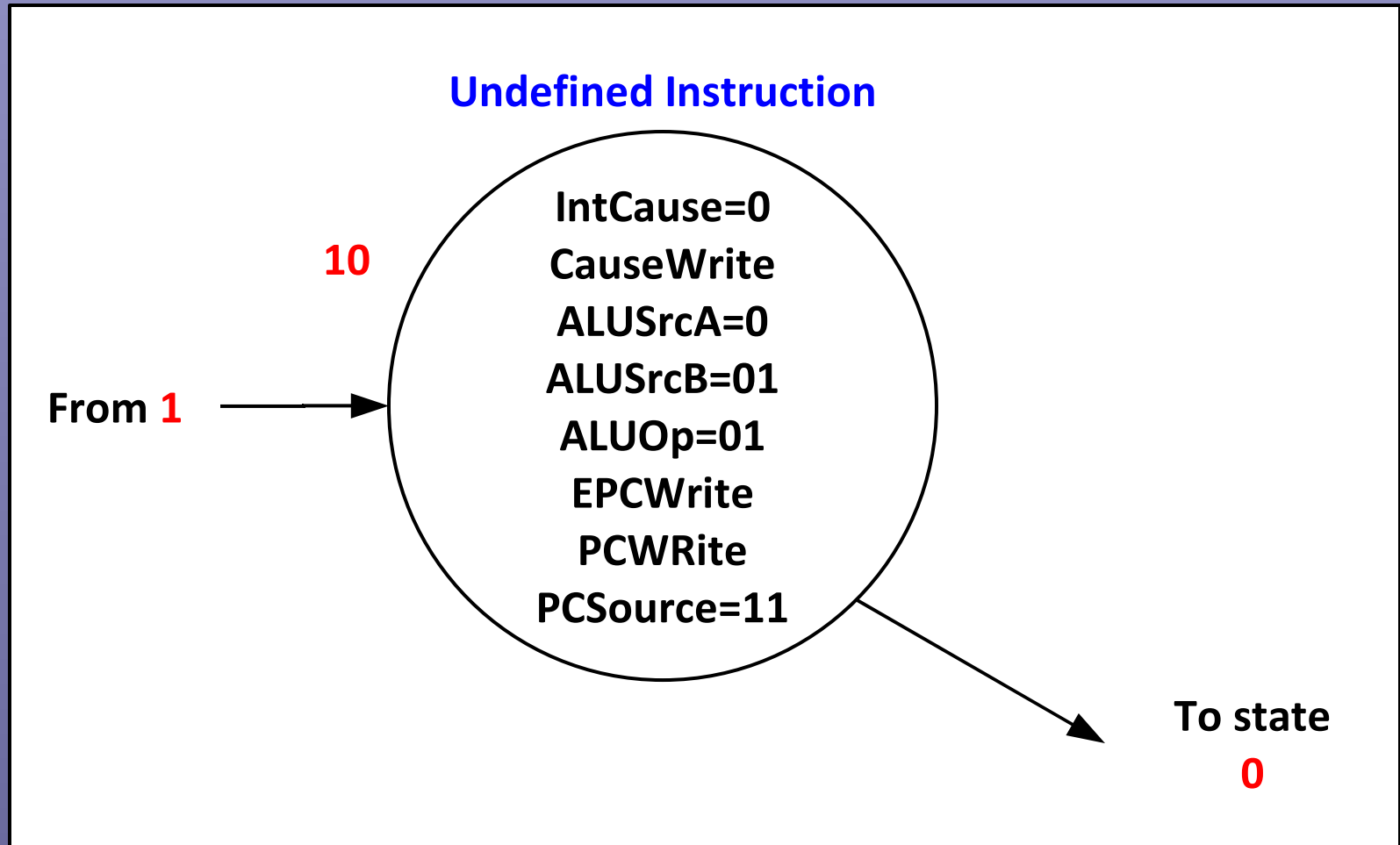
# MIPS: Podpora výjimek: přetečení

## R-type instruction



# MIPS: Podpora výjimek: neplatná instrukce

## Undefined instruction





# Proudové zpracování (1)

## 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

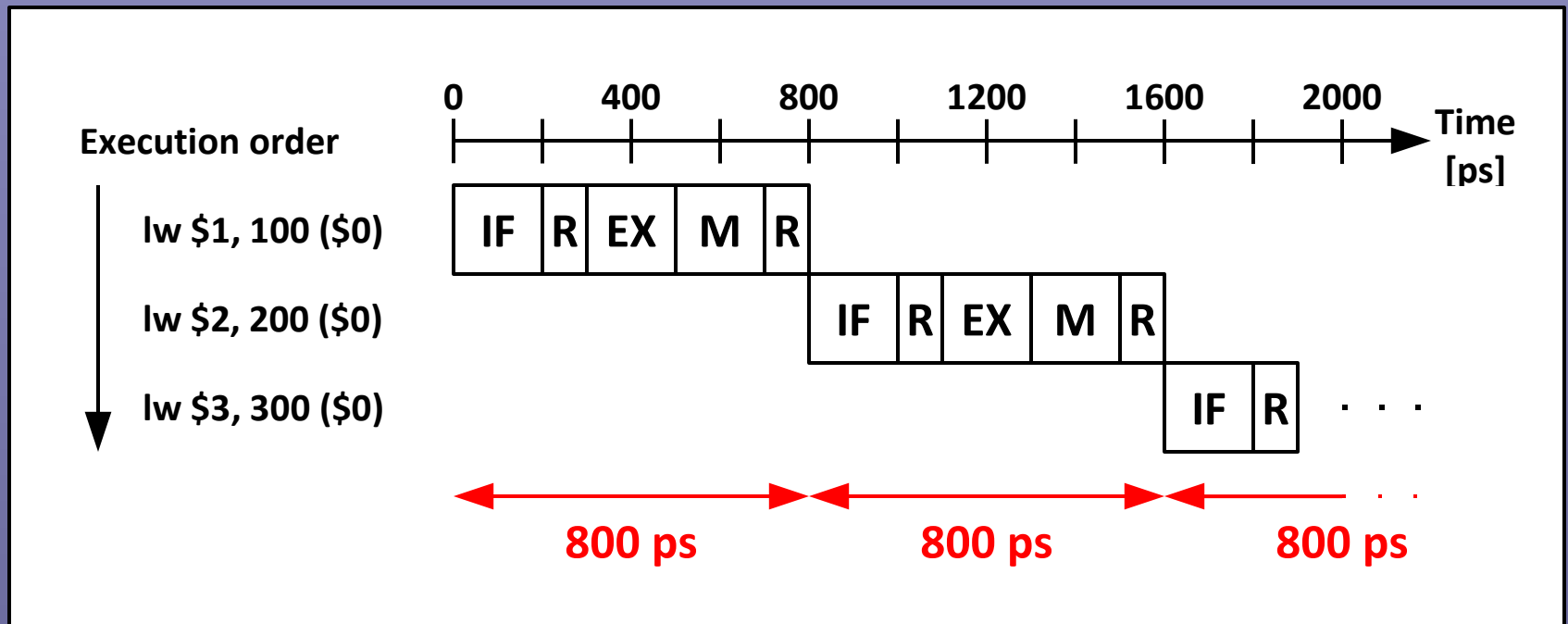
## Jednocyklové vs. proudové zpracování

- výkon závisí na nejpomalejší operaci

# Proudové zpracování (2)

## Jednocyklová datová cesta

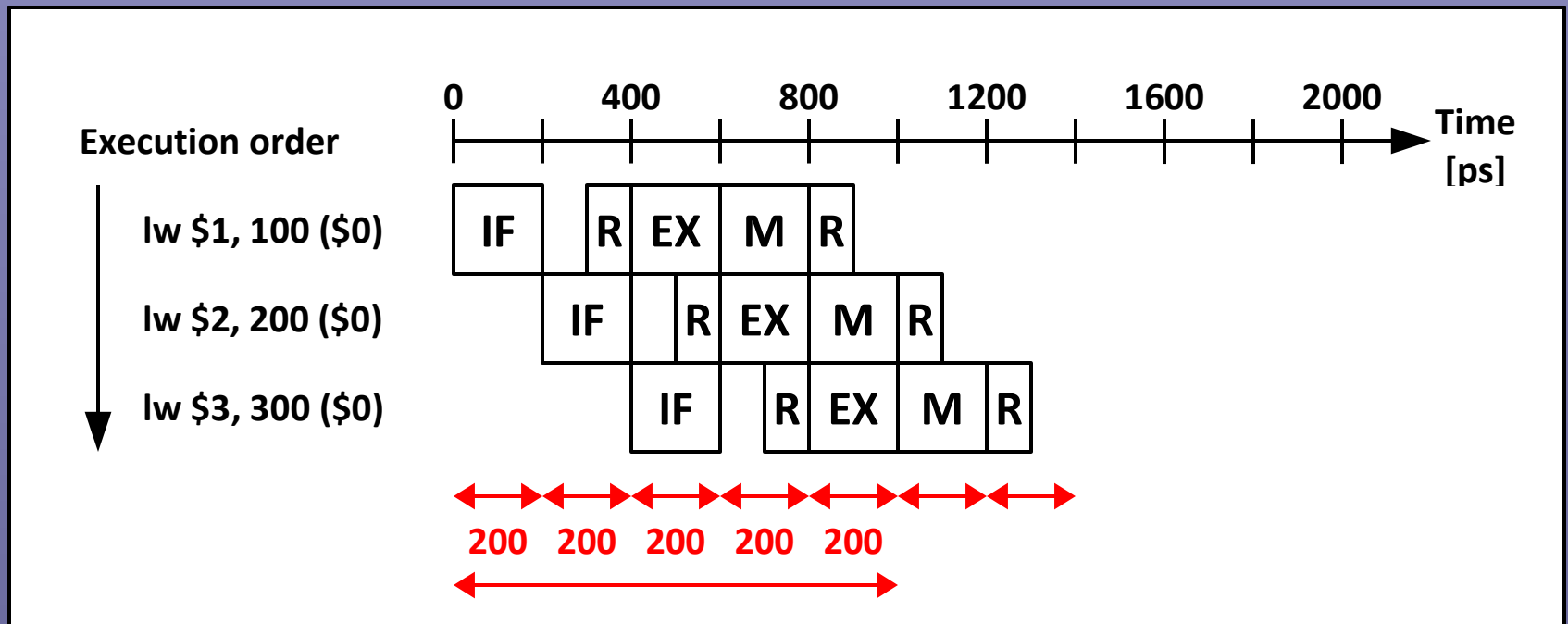
- instrukce *load word* (nejdelší)
  - ♦ práce s registry 100ps, ostatní 200ps



# Proudové zpracování (3)

## Proudová datová cesta

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



# Proudové zpracování (4)

## Návrh instrukční sady (MIPS)

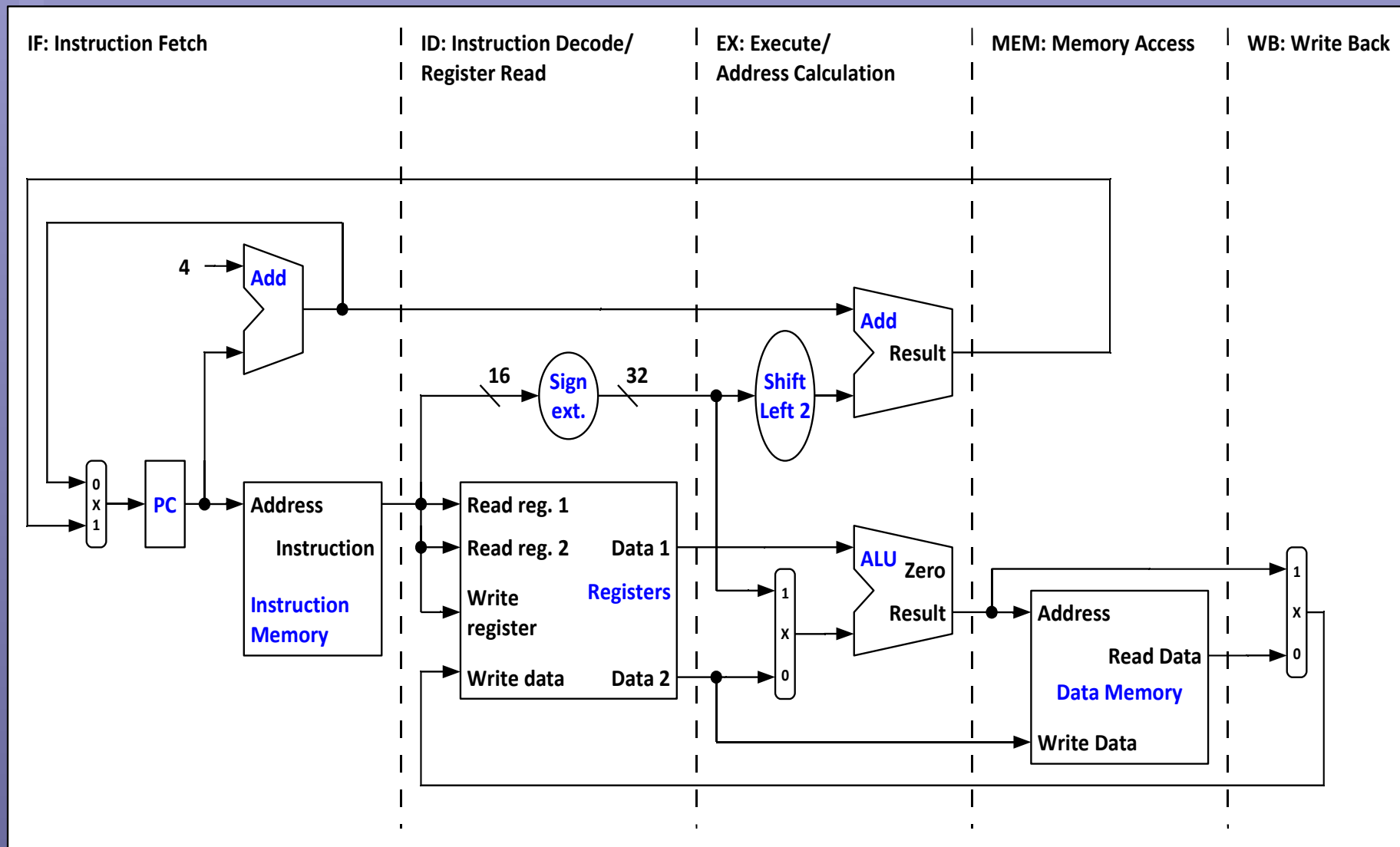
- instrukční kód stejné délky
  - ♦ jednoduché načítání instrukcí do pipeline
- omezený počet formátů instrukčního kódu
  - ♦ čtení registrů probíhá zároveň s dekodováním instrukce
- paměťové operandy pouze v load/store operacích
  - ♦ výpočet adresy ve fázi vykonání instrukce
  - ♦ přístup do paměti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

# Proudové zpracování (5)

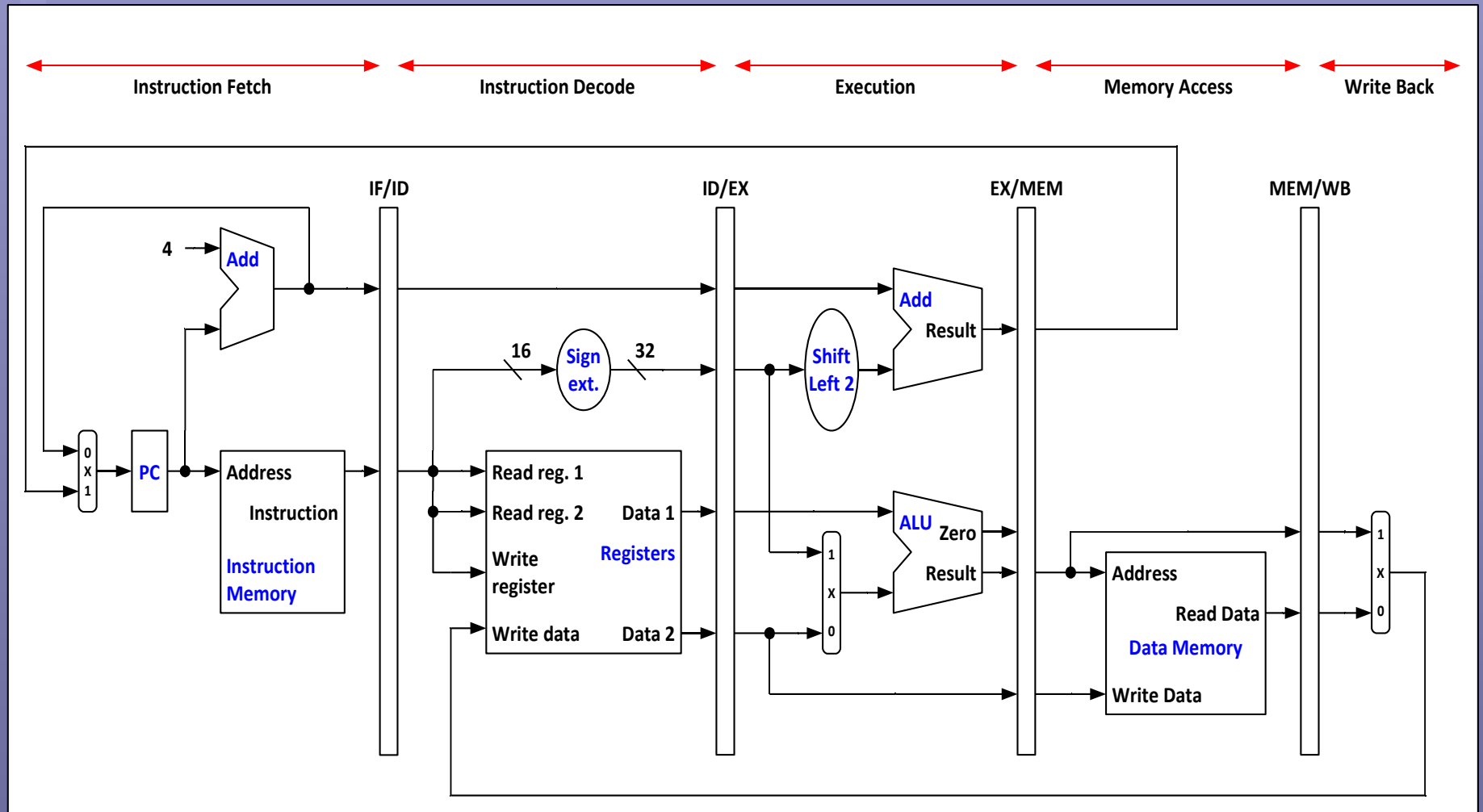
## Konstrukce datové cesty

- 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

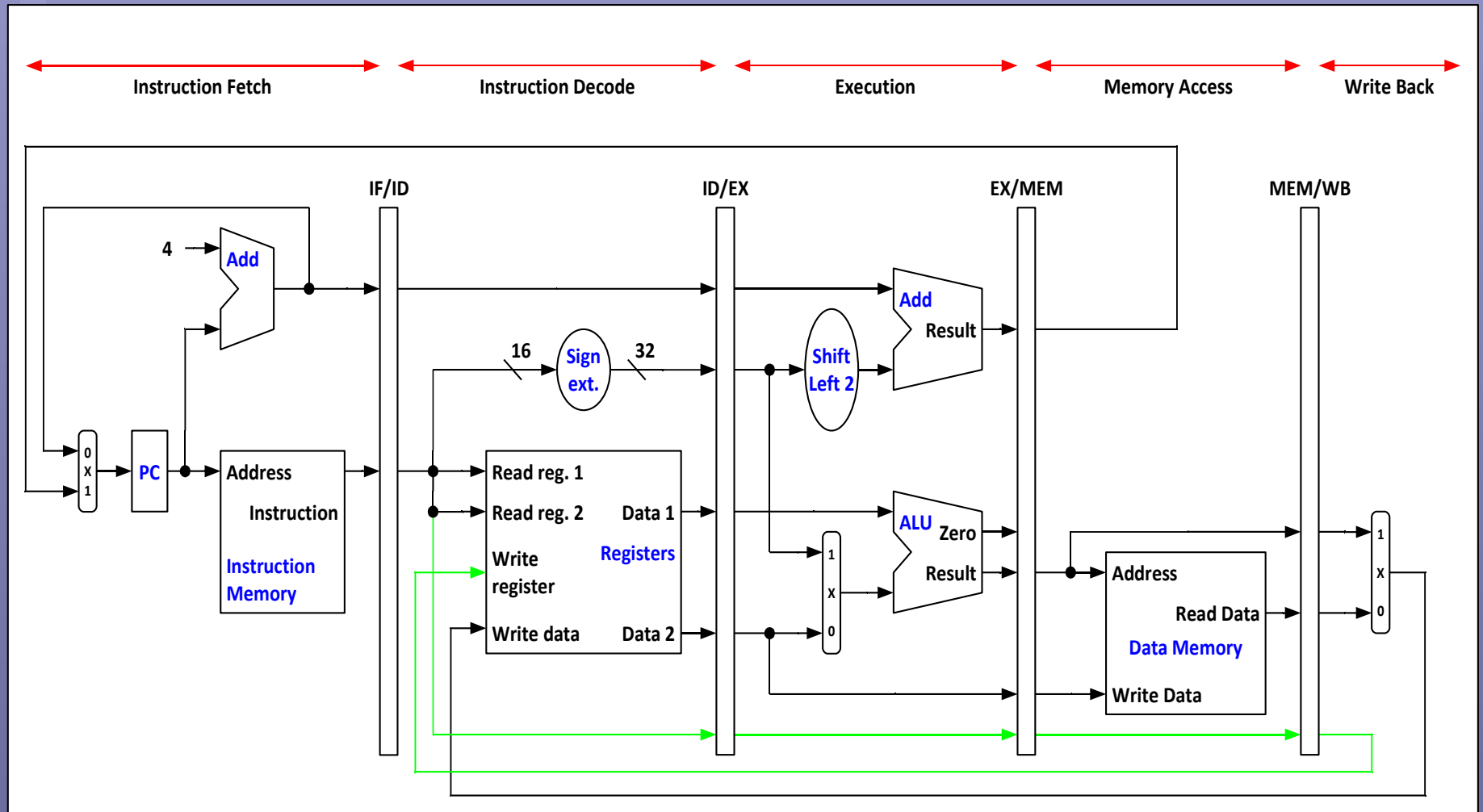
# MIPS: Jednocyklová datová cesta



# MIPS: Proudová datová cesta (1)



# MIPS: Proudová datová cesta (2)

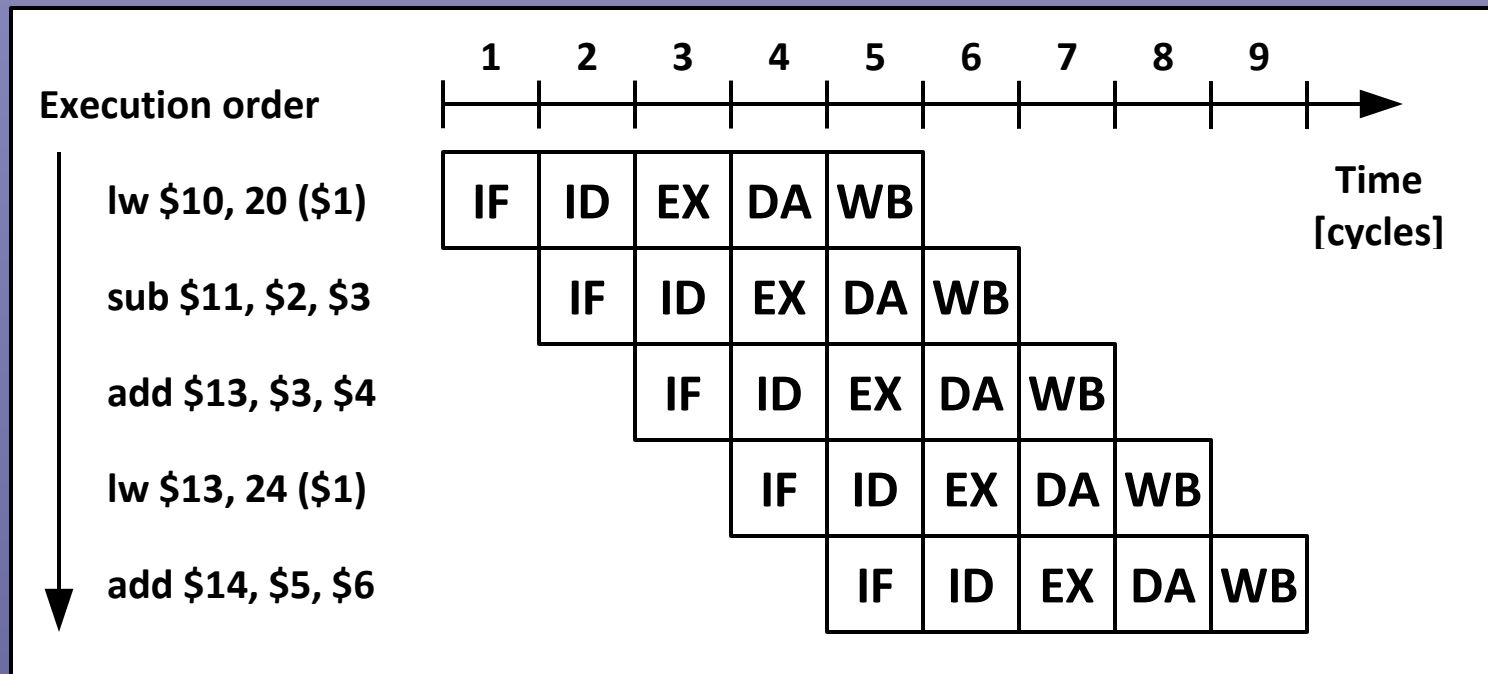




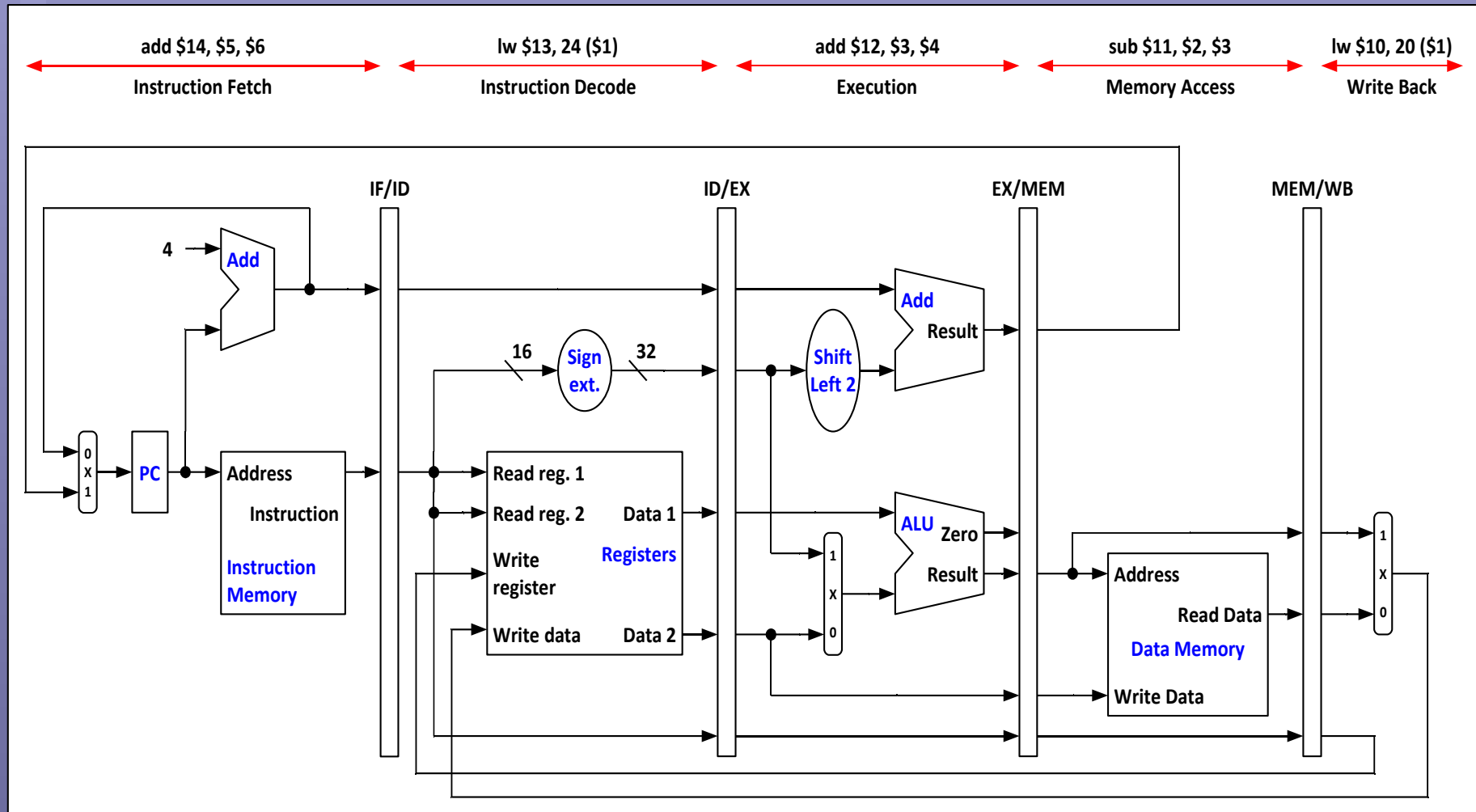
# Proudové zpracování (6)

## Reprezentace pipeline

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



# MIPS: Proudová datová cesta (3)



# Proudové zpracování (7)

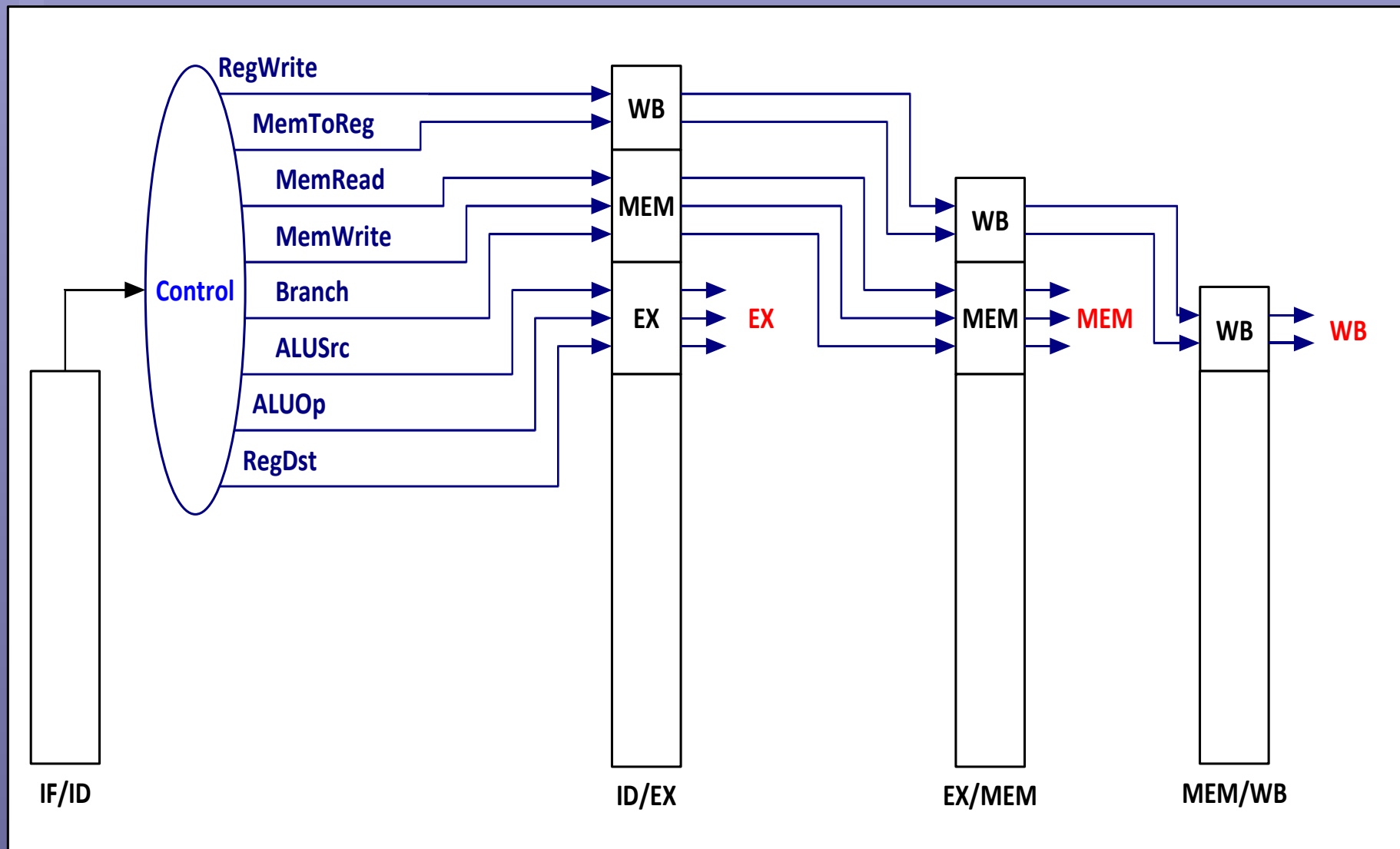
---

## 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: Řízení proudové datové cesty



# Datový hazard v proudovém zpracování (1)

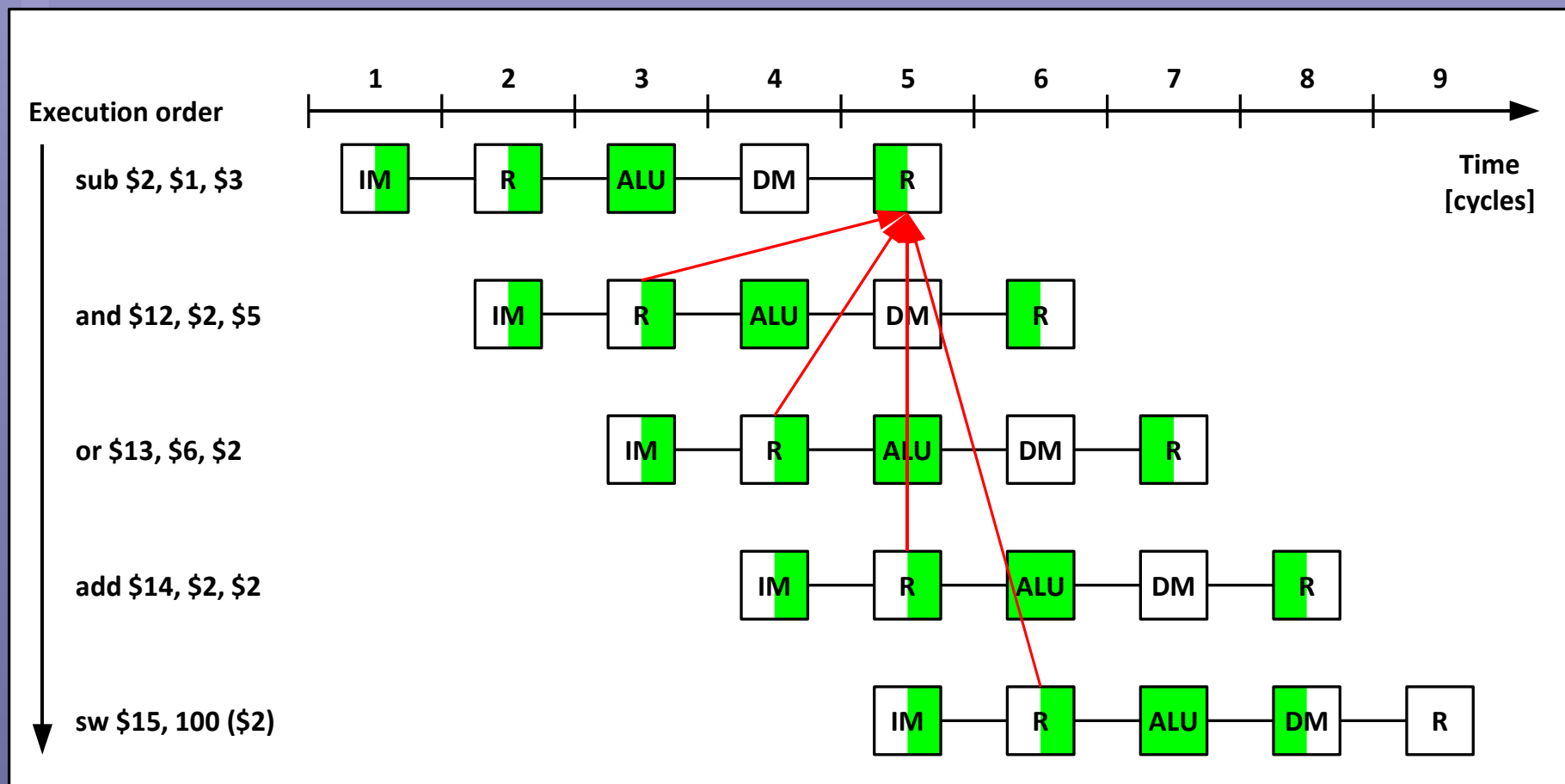
## Forwarding/bypassing

- poskytnutí mezivýsledku následující instrukci
- nelze forwardovat zpátky v čase

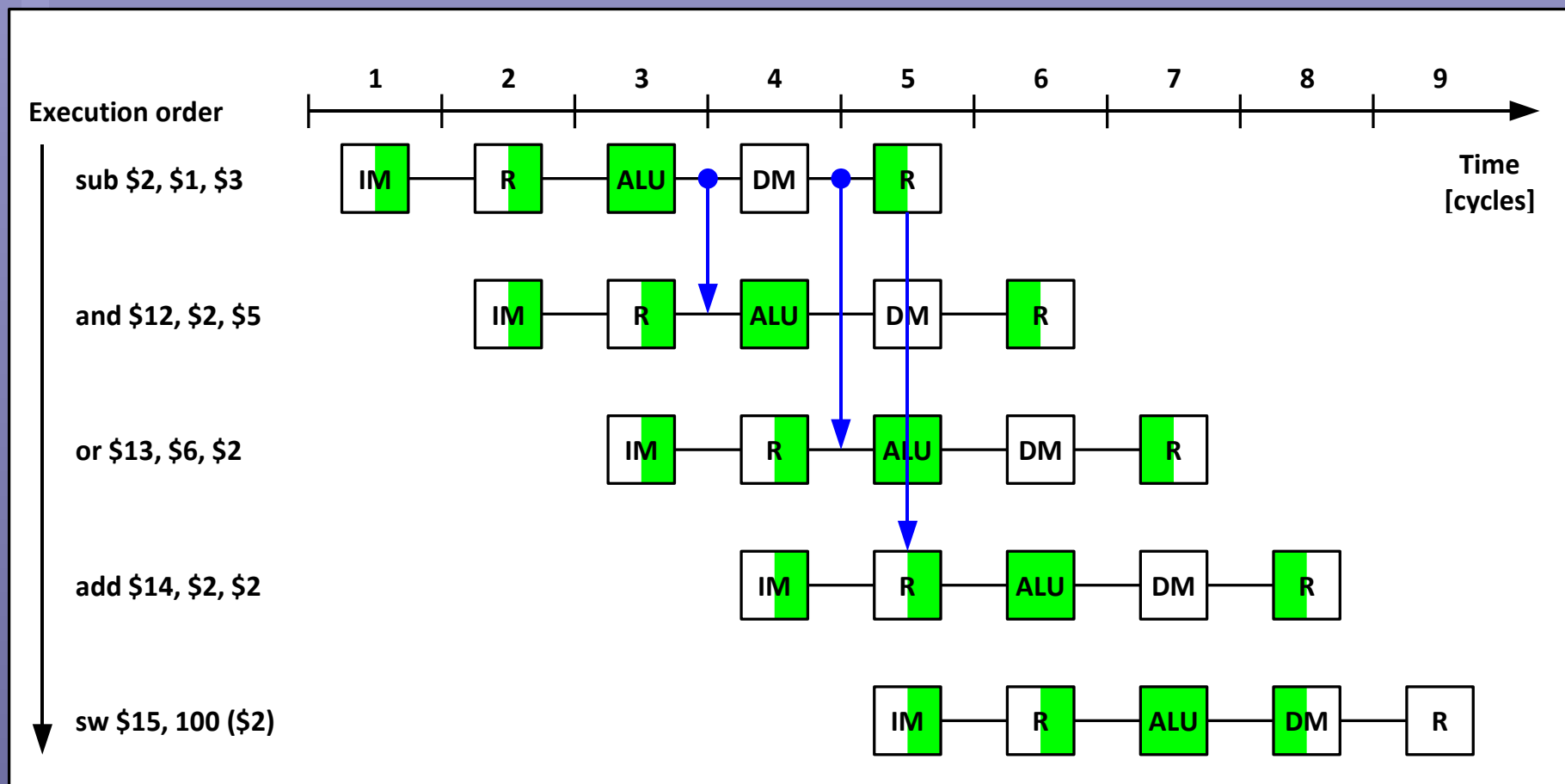
## Forwarding unit (MIPS)

- zdrojový operand vykonávané instrukce je cílový operand výsledku dřívejší 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)

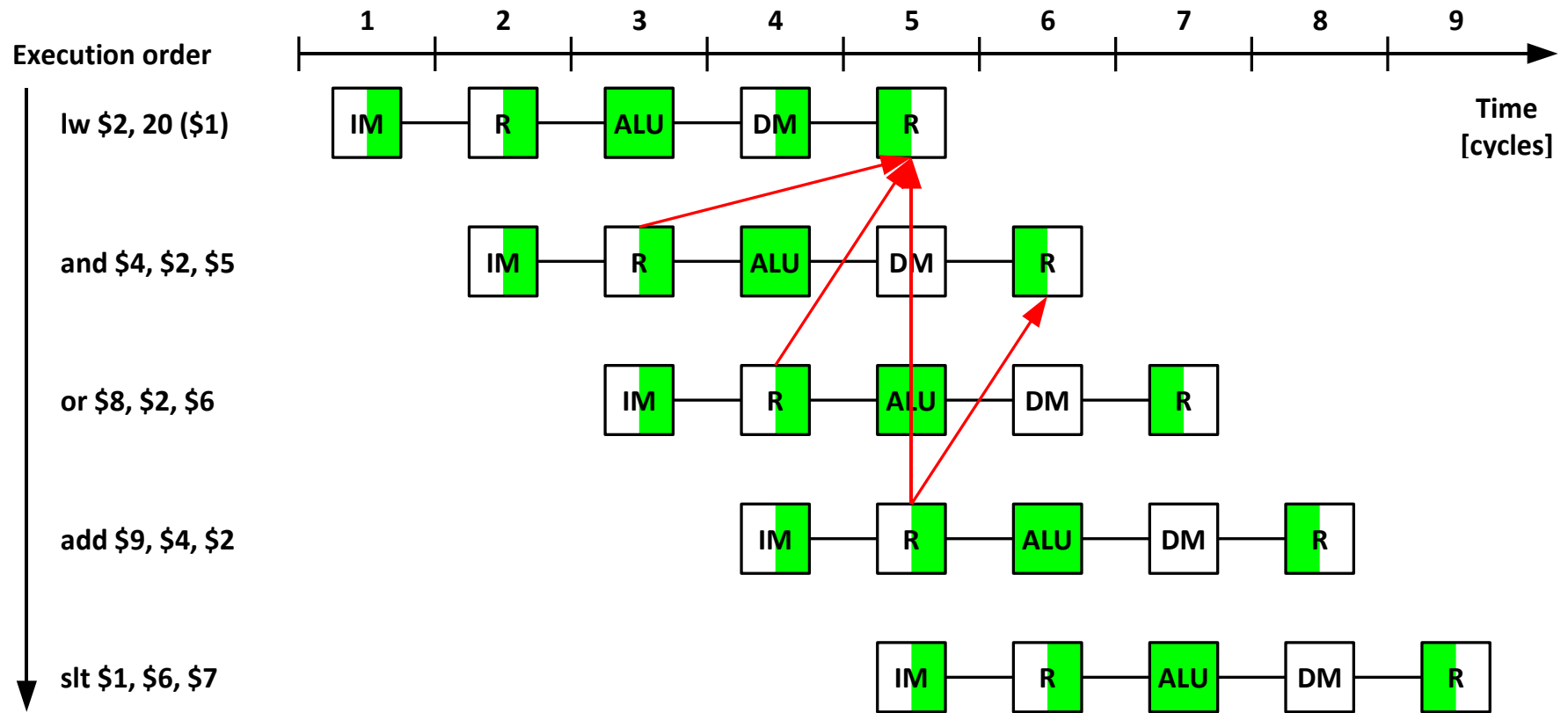
## Zpoždění instrukce v pipeline

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

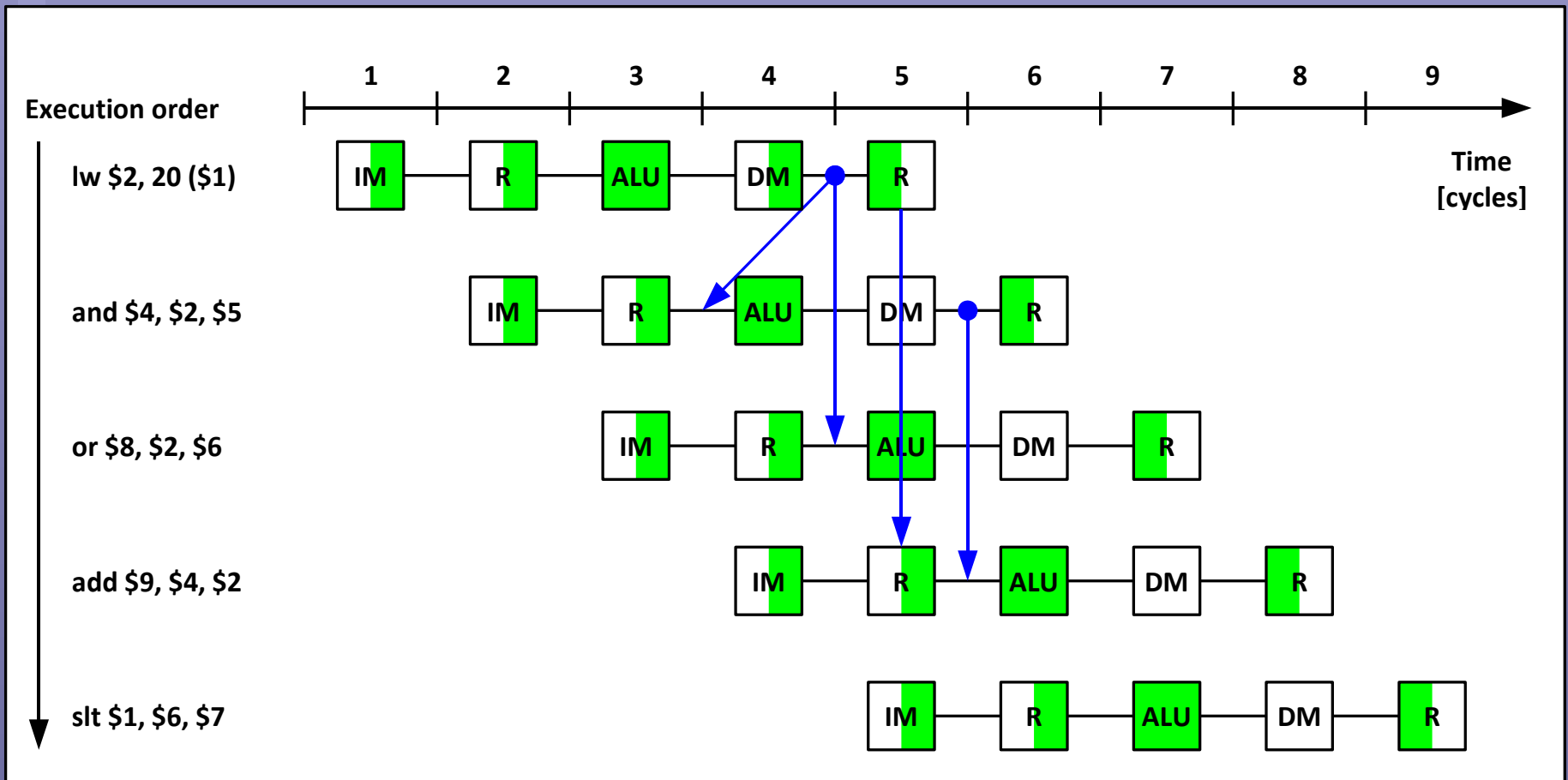
## Hazard detection unit (MIPS)

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

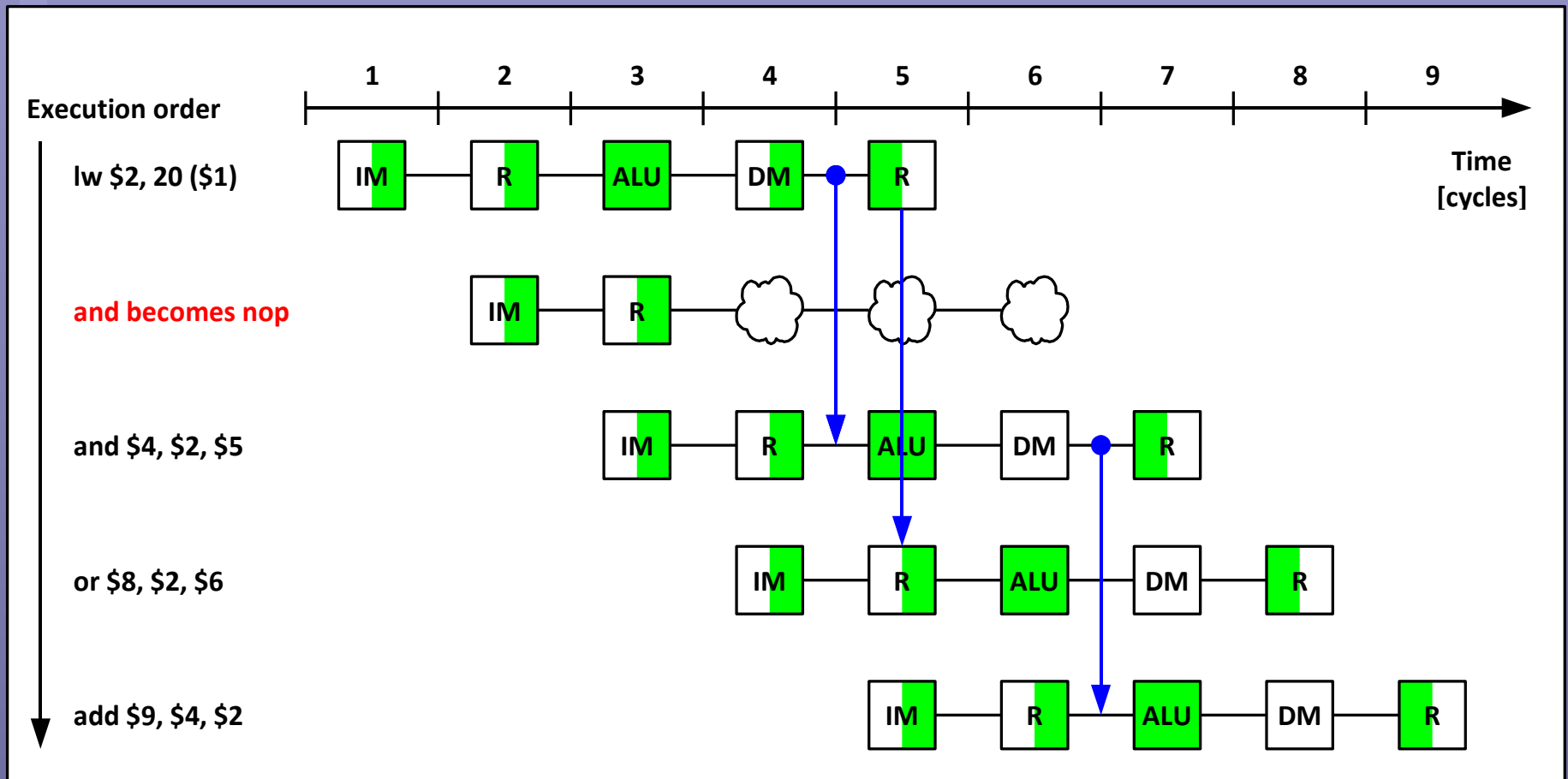
# Zpoždění v proudovém zpracování (1)



# Zpoždění v proudovém zpracování (1)



# Zpoždění v proudovém zpracování (2)



# Řídící hazard v proudovém zpracování (1)

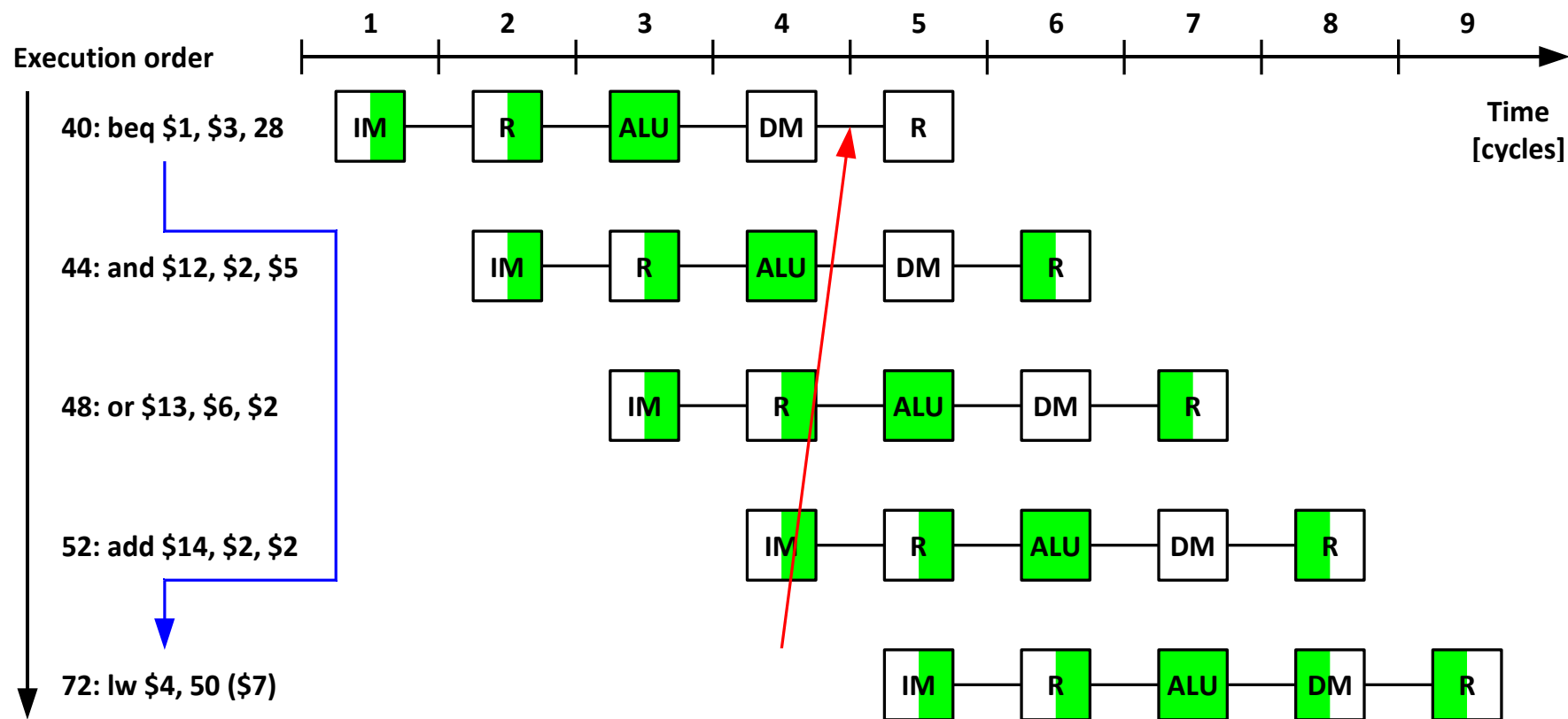
## Rozhodnutí o adrese další instrukce

- 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 při větvení (1)



# Literatura

---

D. A. Patterson, J. L. Hennessy

- Computer Organization and Design