

# Principy počítačů a operačních systémů

Mikroarchitektura

Zimní semestr 2007/2008

# Vnitřní uspořádání (mikroarchitektura)

## Architektura z pohledu návrháře

- návrh a uspořádání funkčních bloků realizující konkrétní ISA
- kompozice funkčních celků
  - ♦ nejde pod úroveň logických obvodů

## Hlavní části

- datová cesta (data path)
- řízení (control)

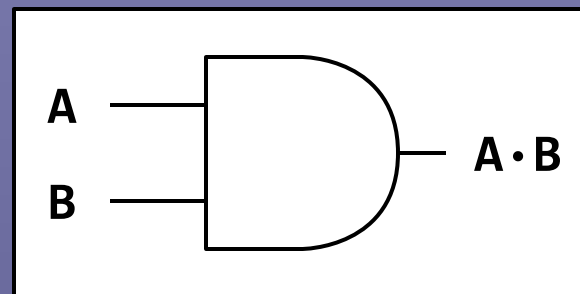
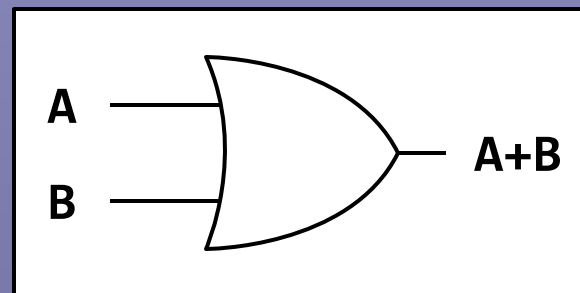
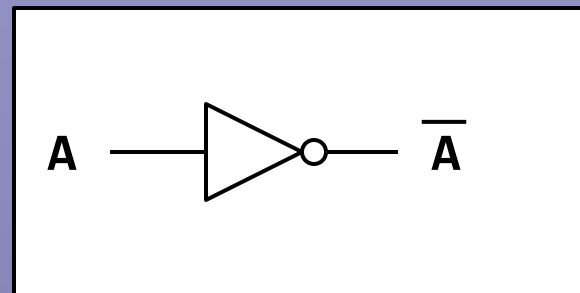
# Číslicové systémy (1)

## Booleova algebra

- algebraický zápis logických výrazů
- manipulace s výrazy

## Logické obvody

- realizace logických funkcí
- hradla realizují základní funkce
  - ♦ NOT, OR, AND



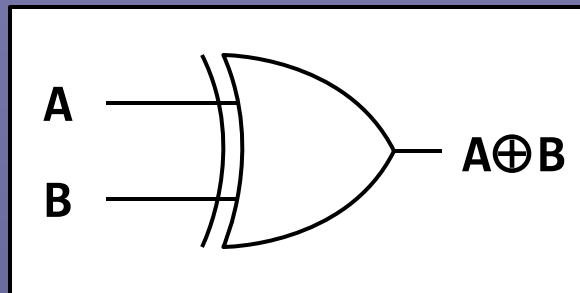
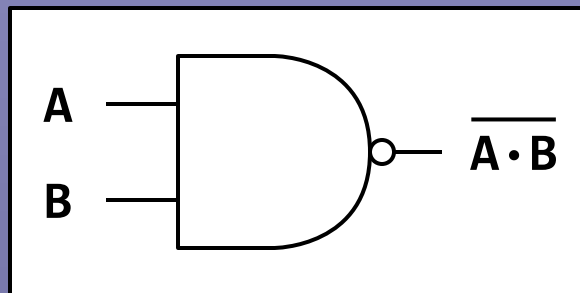
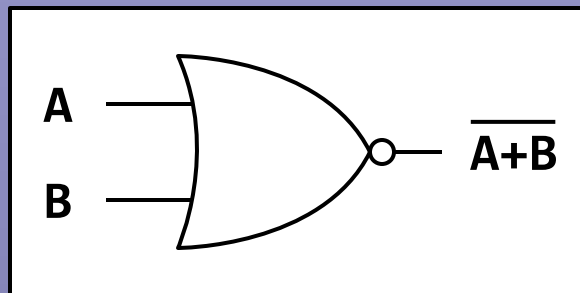
# Číslicové systémy (2)

## Odvozené funkce

- hradla realizují funkce
  - ♦ NOR, NAND, XOR

## Praktická realizace

- Booleova algebra
  - ♦ stačí NOT + 1 základní operace
  - ♦ NAND/NOR zahrnuje negaci
- pouze hradla NAND/NOR
  - ♦ závisí na technologií výroby



# Číslicové systémy (3)

---

## Kombinační obvody

- funkce logických proměnných
- výstup závisí pouze na aktuálním vstupu

## Jednoduché funkční bloky

- multiplexory, demultiplexory
- kodéry, dekodéry

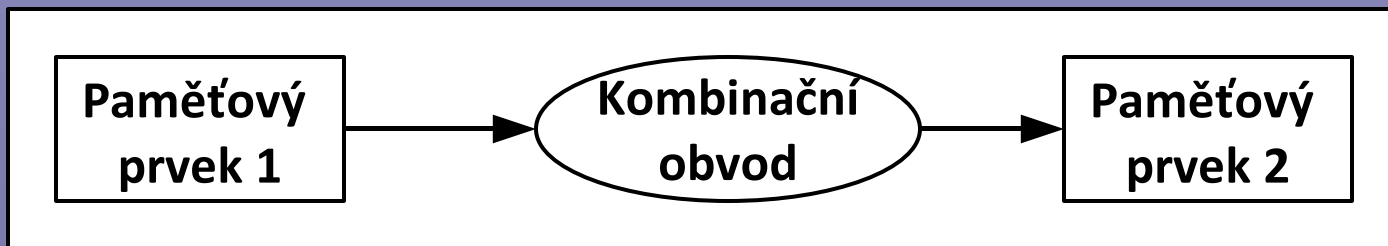
## Složitější funkční bloky

- části ALU

# Číslicové systémy (4)

## Sekvenční obvody

- kombinační obvody + paměťové prvky
  - ♦ paměťové prvky udržují stav
  - ♦ vstup a aktuální stav určuje výstup a následující stav

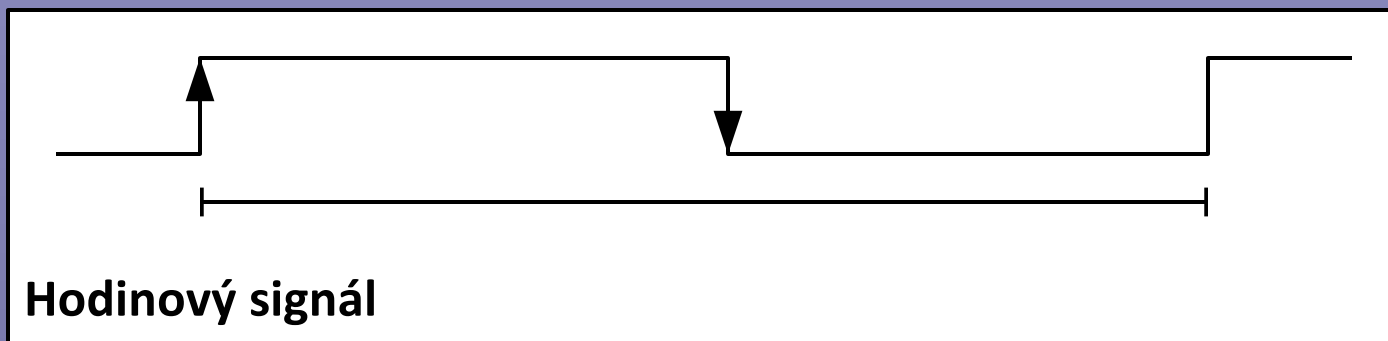


- synchronní/asynchronní
  - ♦ způsob a okamžik změny stavu

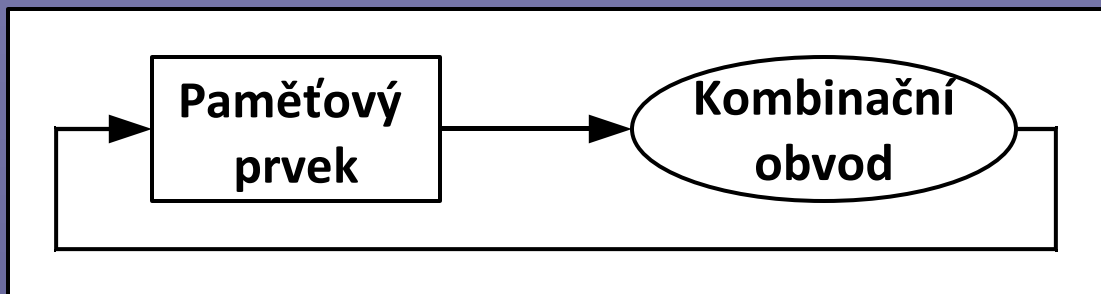
# Číslicové systémy (5)

## Synchronní sekvenční obvody

- hodinový signál synchronizuje změny stavu
  - ♦ náběžná/sestupná hrana, perioda



- změna stavu během jednoho cyklu



# Logická realizace součtu (1)

## Poloviční sčítačka

- součet dvou 1-bitových čísel
  - ♦ vstupy: operand  $a$ , operand  $b$
  - ♦ výstupy: součet  $s$ , přenos  $c$
- bez přenosu z nižšího řádu
- součet jako logická funkce
  - ♦  $s = a \text{ XOR } b$
  - ♦  $c = a \text{ AND } b$

$a$	$b$	$s$	$c$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



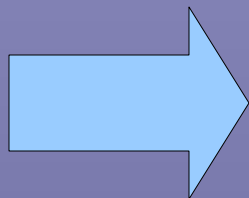
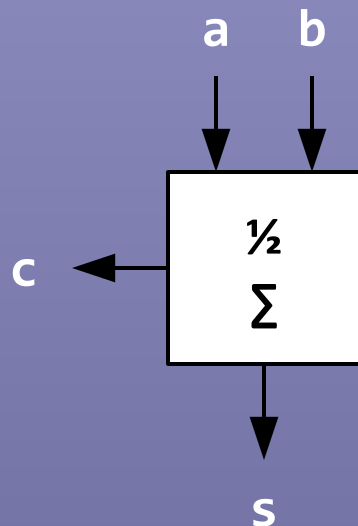
$a$	$b$	XOR	AND
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



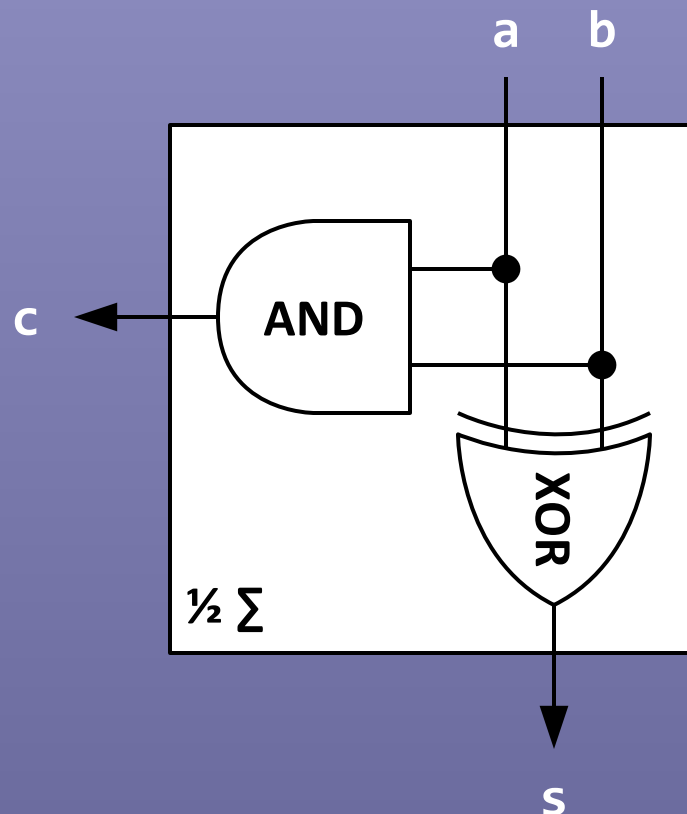
# Logická realizace součtu (2)

## Poloviční sčítačka

- funkční blok



- logický obvod



# Logická realizace součtu (3)

## Úplná sčítačka

- součet čísel a přenosu z nižšího řádu
  - ♦ 3 vstupy: čísla  $a$ ,  $b$ , přenos z nižšího řádu  $c_i$
  - ♦ 2 výstupy: součet  $s$ , přenos do vyššího řádu  $c_o$
- realizuje součet tří čísel
  - ♦  $s = a + b + c_i = (a + b) + c_i$
  - ♦ přenosy vznikají při obou součtech

$c_i$	$a$	$b$	$s$	$c_o$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

# Logická realizace součtu (4)

## Úplná sčítačka

### ▪ $(a + b)$

$$s_1 = a \text{ XOR } b$$

$$c_1 = a \text{ AND } b$$

a	b	$s_1$	$c_1$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

### ▪ výsledný přenos

$$c = c_1 \text{ OR } c_2$$

$c_1$	$c_2$	$c_o$
0	0	0
0	1	1
1	0	1
1	1	1

### ▪ $(a + b) + c_i$

$$s = s_1 \text{ XOR } c_i$$

$$c_2 = s_1 \text{ AND } c_i$$

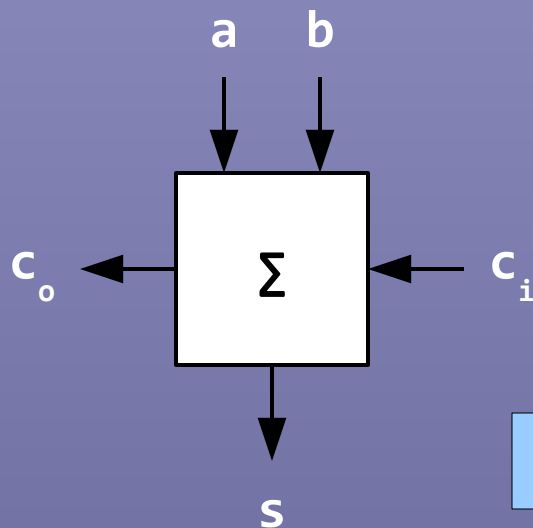
$c_i$	$s_1$	s	$c_2$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$c_i$	a	b	s	$c_o$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

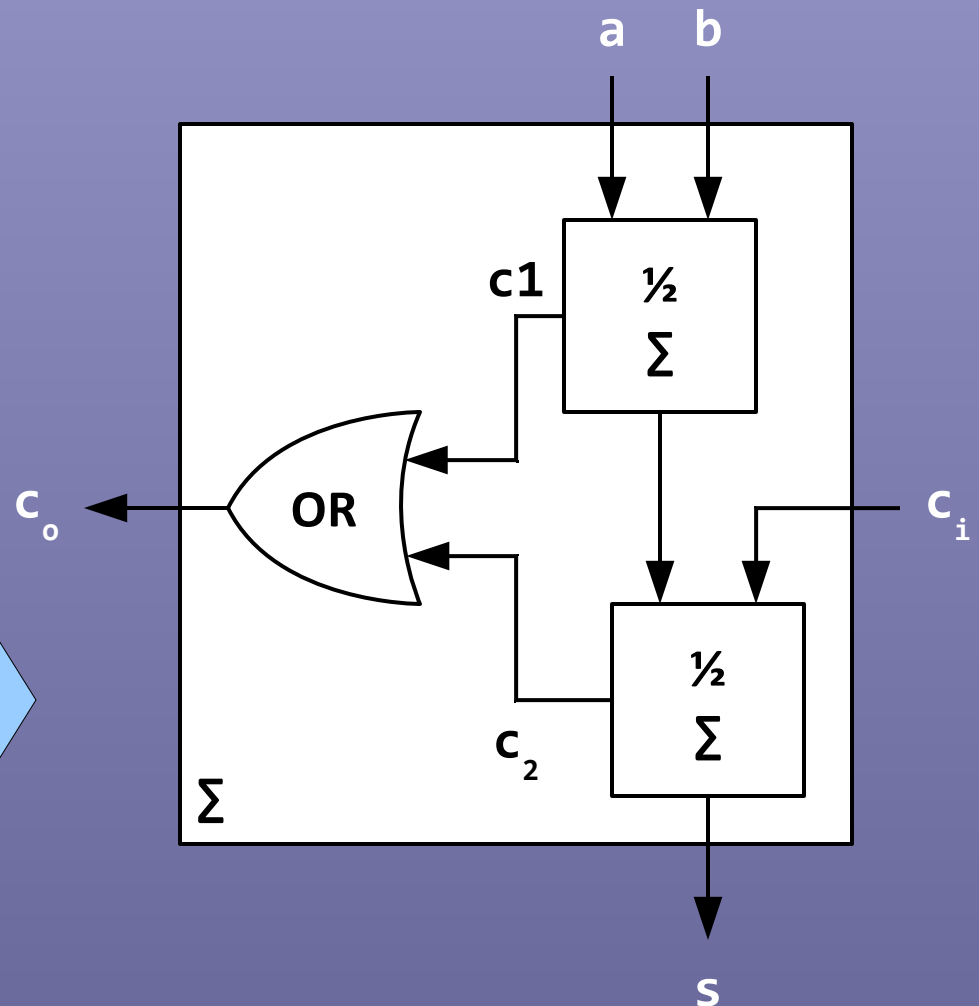
# Logická realizace součtu (5)

## Úplná sčítačka

- funkční blok



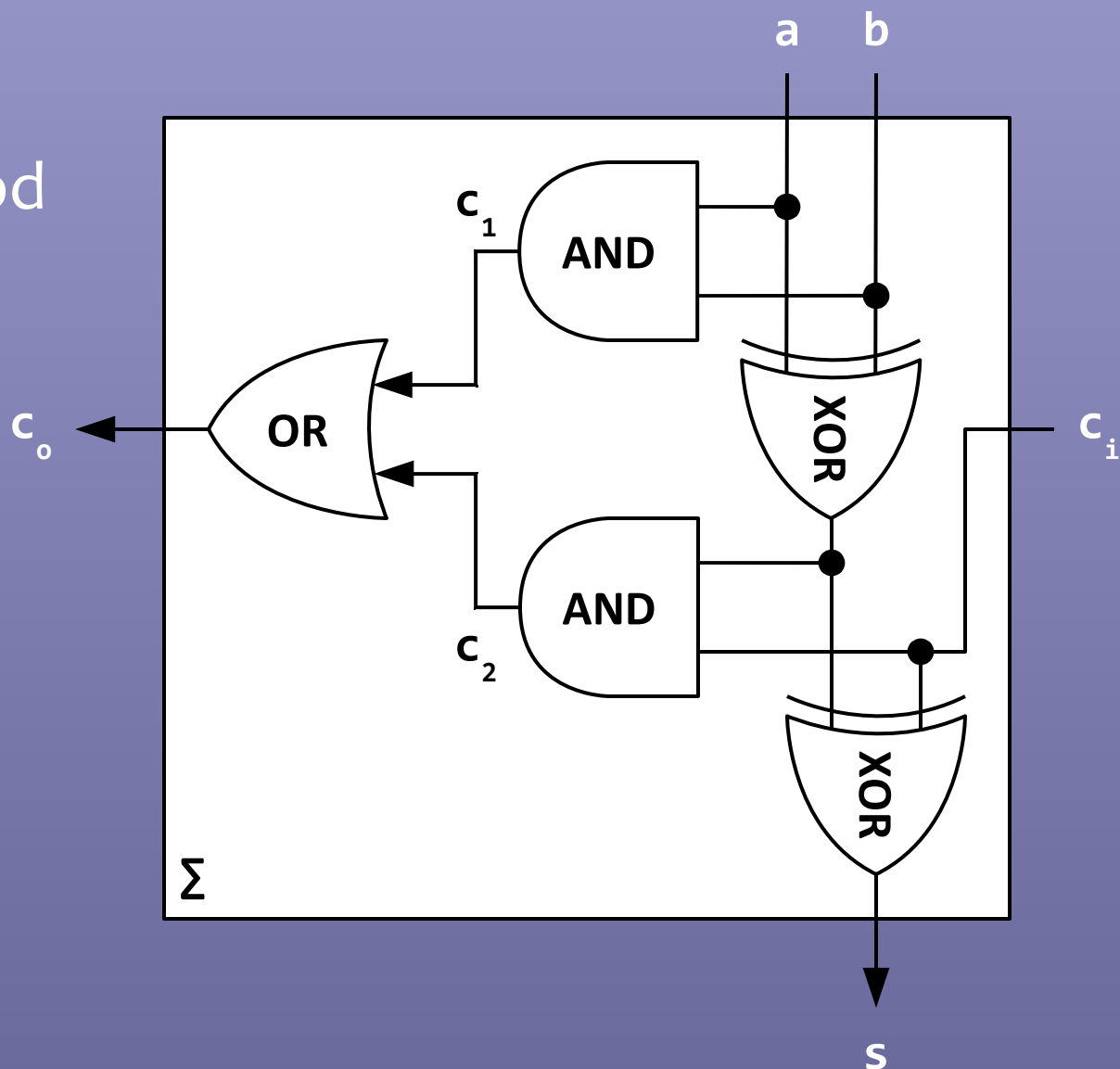
- vnitřní struktura



# Logická realizace součtu (6)

## Úplná sčítačka

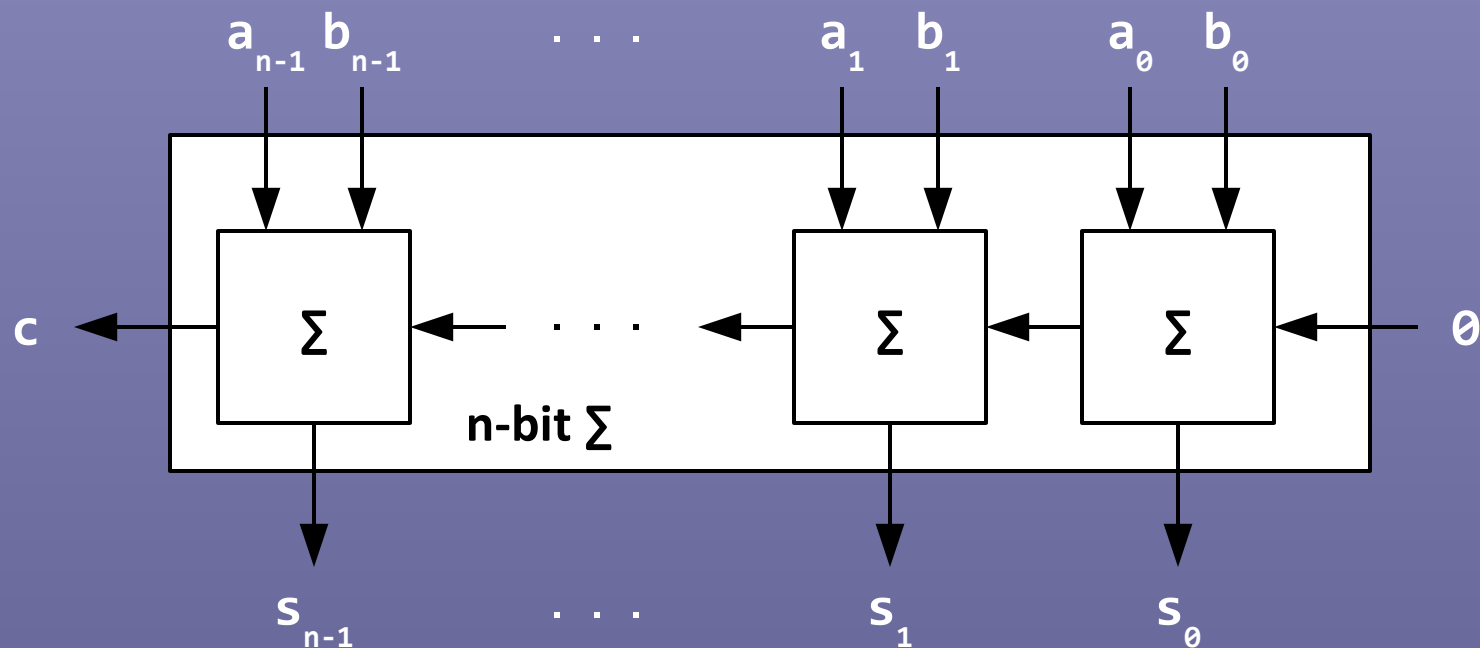
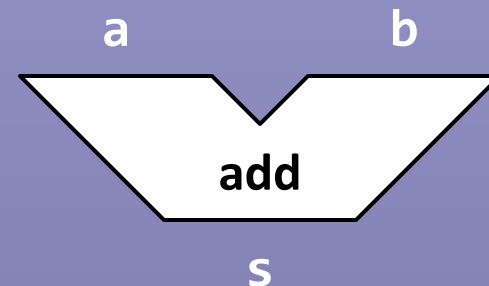
- logický obvod



# Logická realizace součtu (7)

## Sčítačka dvou n-bitových čísel

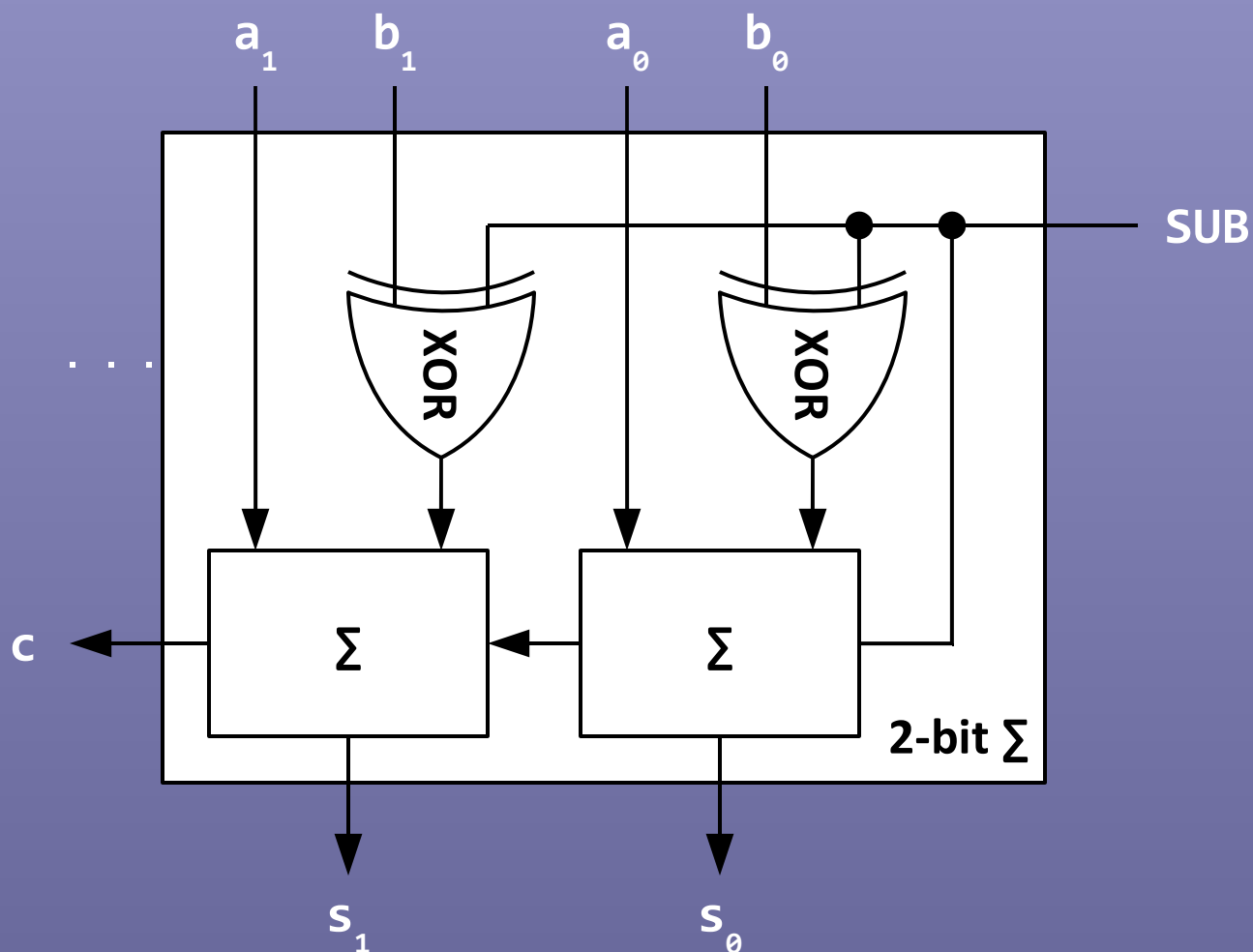
- funkční blok
- vnitřní struktura



# Logická realizace rozdílu

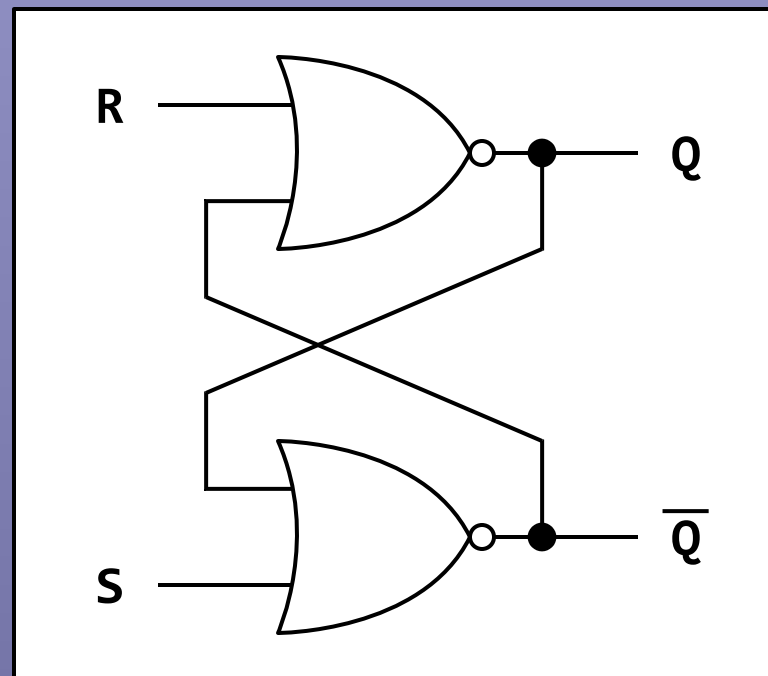
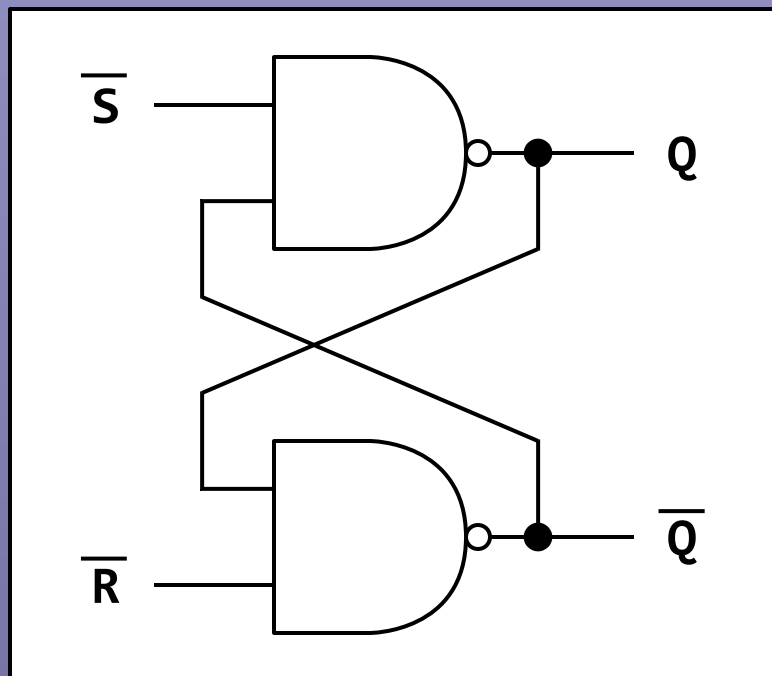
## 2-bitová sčítačka/odčítačka

a	b	$\oplus$
0	0	0
0	1	1
1	0	1
1	1	0



# Paměťové prvky (1)

## Klopný obvod typu R-S (latch)



$\neg R$	$\neg S$	$Q_n$	$\neg Q_n$
0	0	?	?
0	1	0	1
1	0	1	0
1	1	$Q_{n-1}$	$\neg Q_{n-1}$

NAND	a	b	NOR
1	0	0	1
1	0	1	0
1	1	0	0
0	1	1	0

R	S	$Q_n$	$\neg Q_n$
0	0	$Q_{n-1}$	$\neg Q_{n-1}$
0	1	1	0
1	0	0	1
1	1	?	?



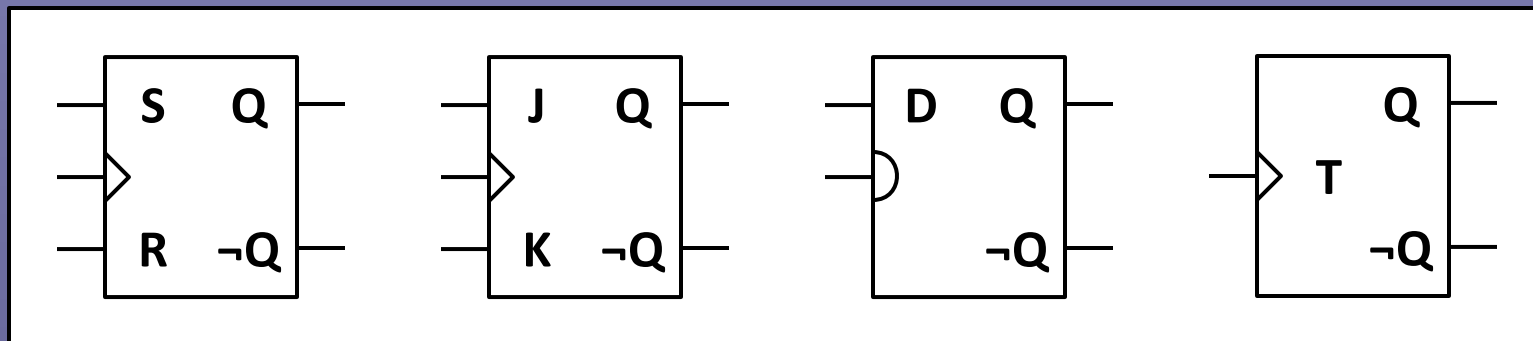
# Paměťové prvky (2)

## Složitější klopné obvody

- R-S s hodinovým vstupem (clocked R-S latch)
- R-S master/slave (R-S flip-flop)
- J-K master/slave (J-K flip-flop)
  - ♦ umí invertovat vlastní stav

## Odvozené obvody a značení

- D latch, D flip-flop, T flip-flop

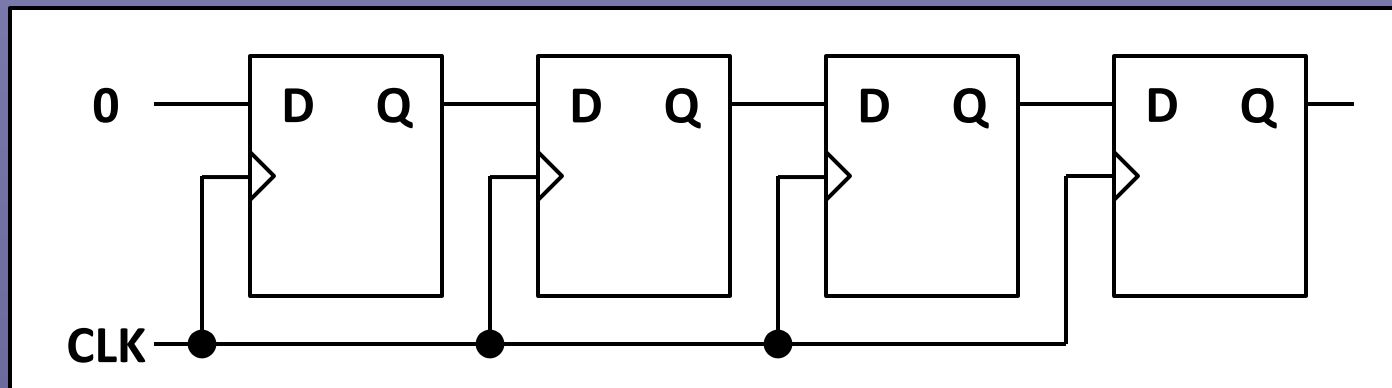


# Paměťové prvky (3)

## n-bitový registr

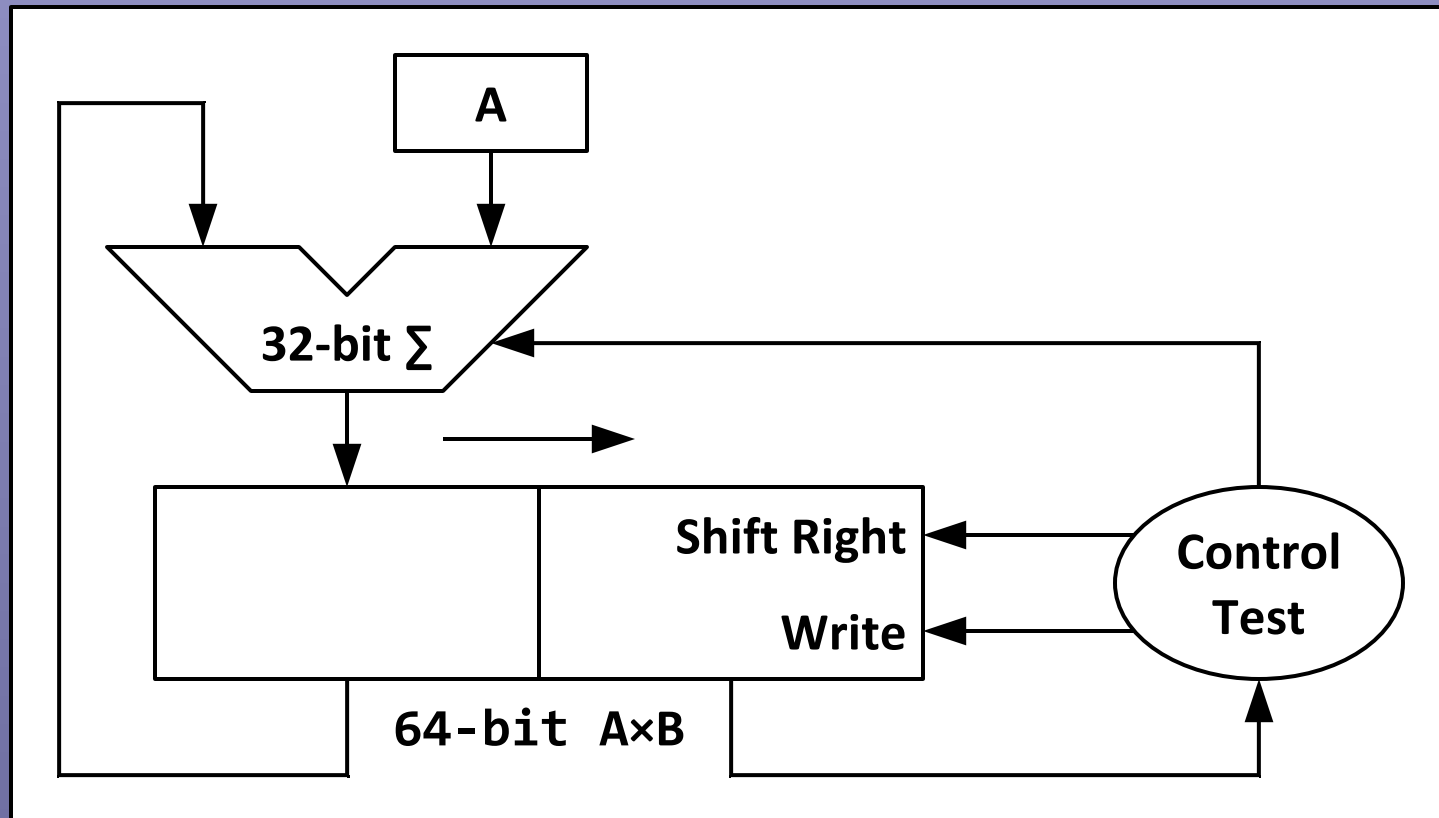
- blok klopných obvodů typu D řízených stejným hodinovým signálem
- vstupy: data  $d_{n-1} \dots d_o$ , hodiny  $clk$ ,
- výstupy: data  $q_{n-1} \dots q_o$

## Posuvný registr

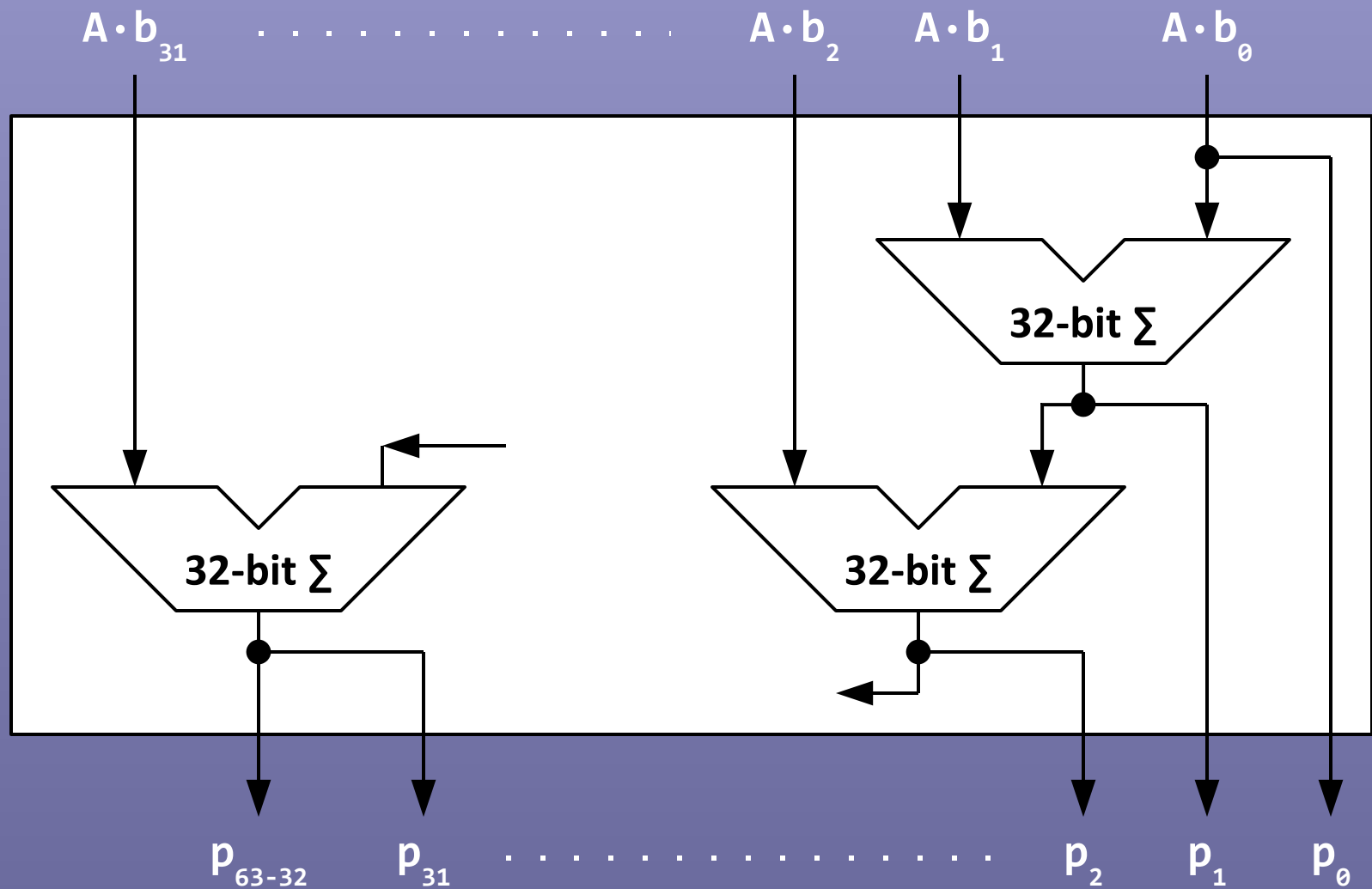


# Logická realizace násobení (1)

## 32-bitová sekvenční násobička

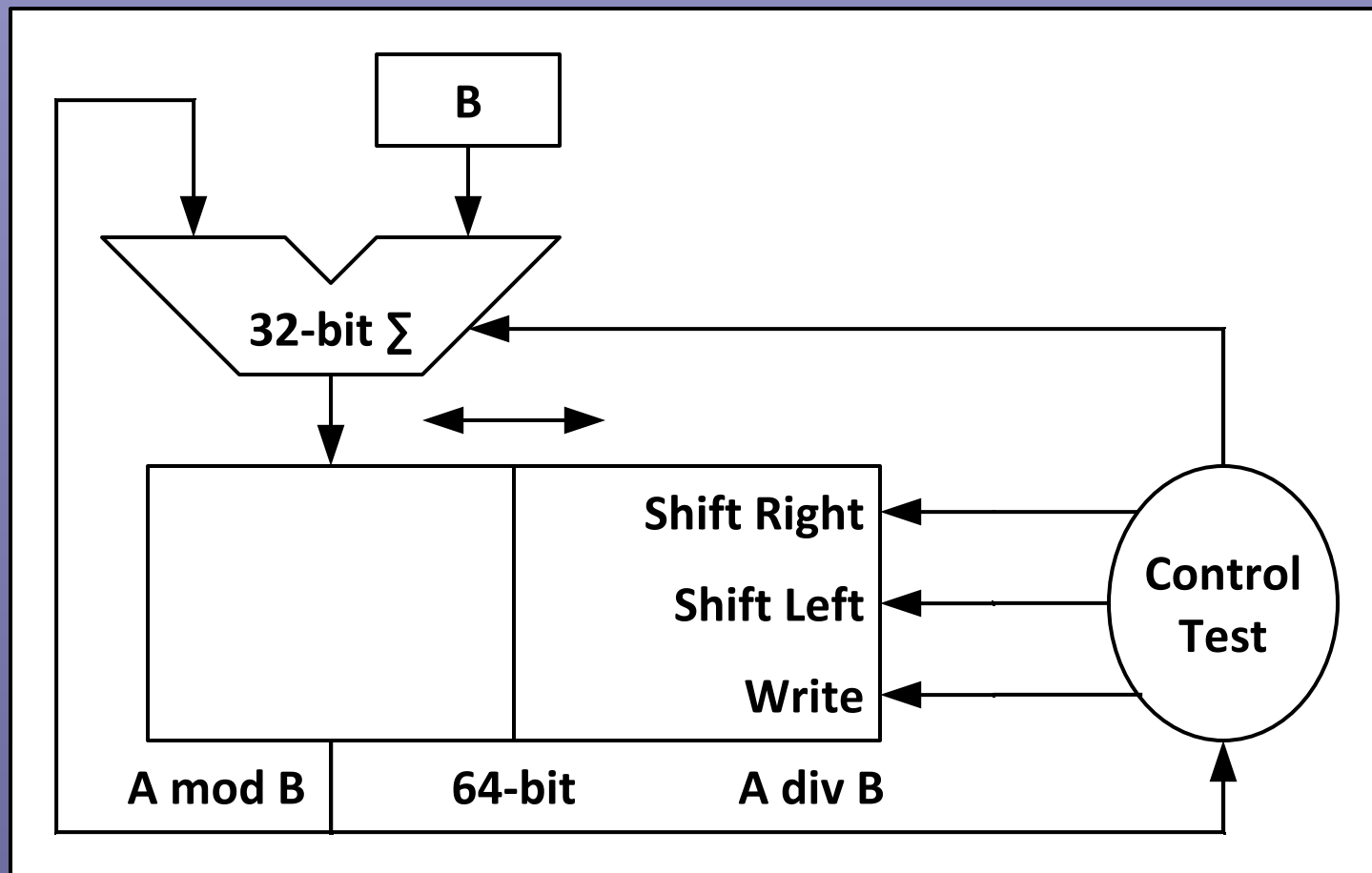


# Logická realizace násobení (2)



# Logická realizace dělení

## 32-bitová sekvenční dělička



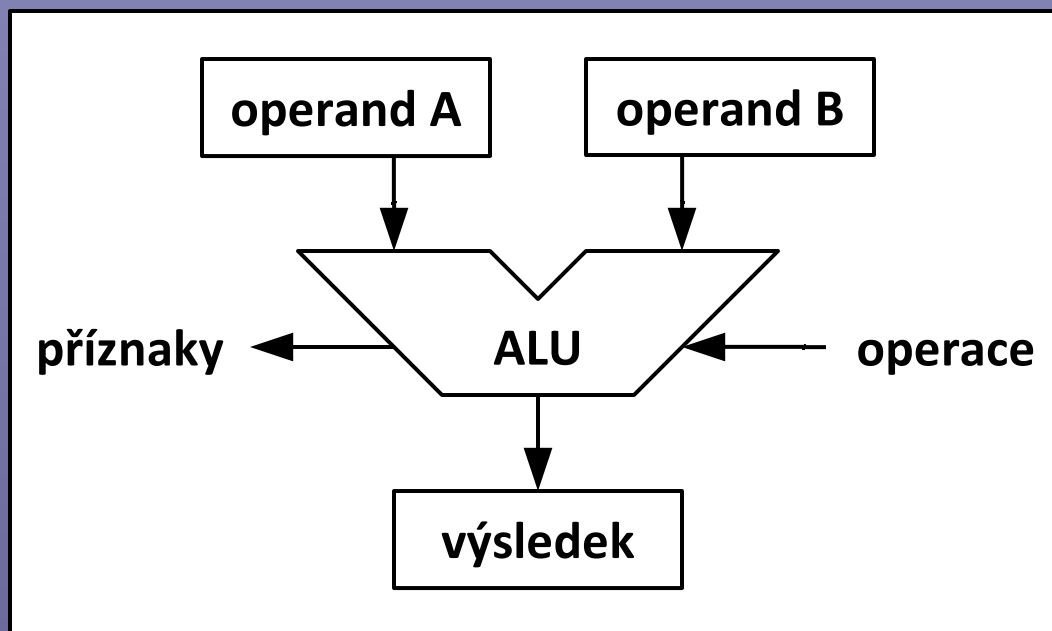
# Aritmeticko-logická jednotka

## Vstupy

- operandy
- operace: sčítání, odčítání, násobení, ..., porovnání

## Výstupy

- příznaky: přenos, nulový výsledek
- výsledek



# 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
----	----------------

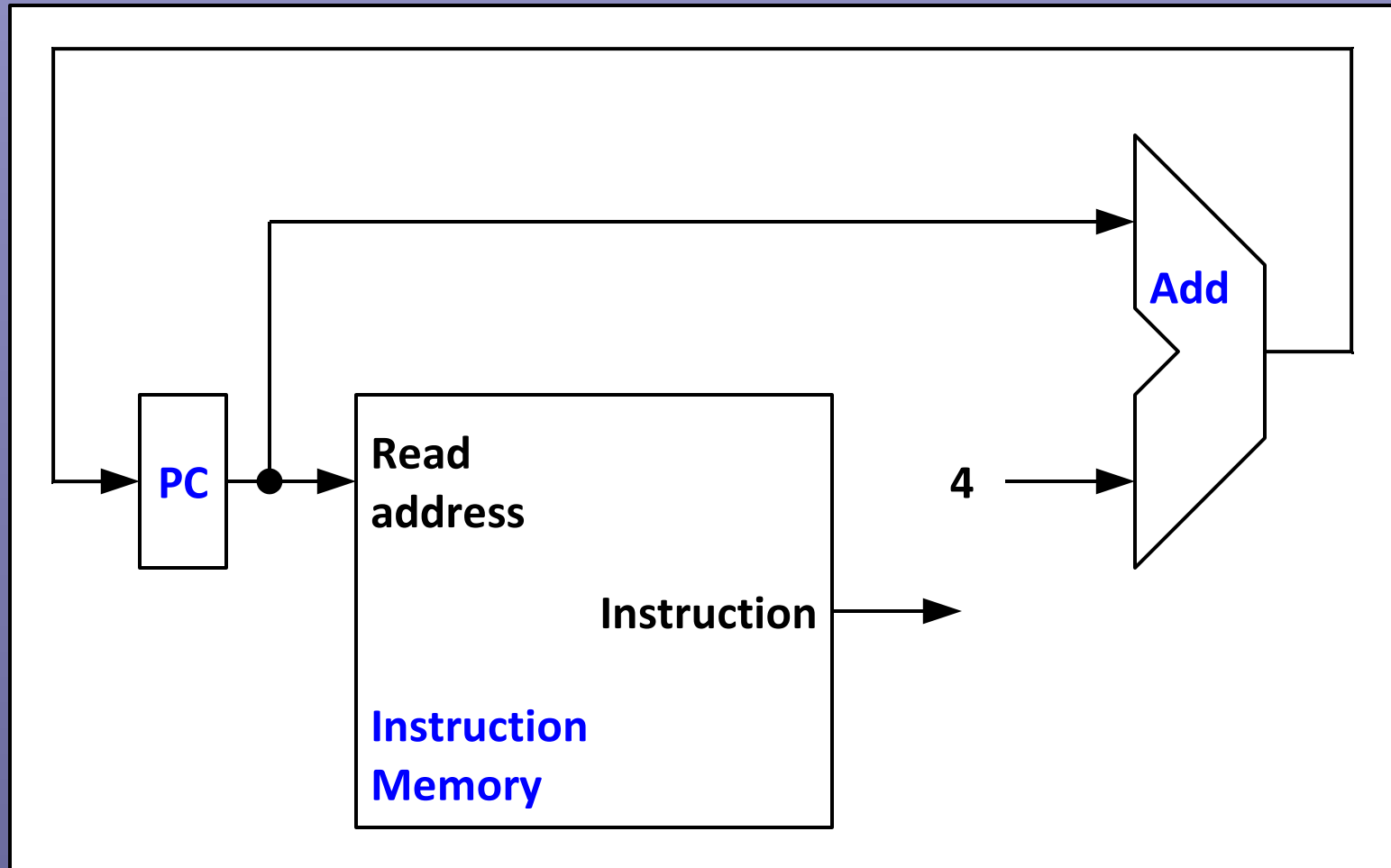
# MIPS: Návrh datové cesty

## 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

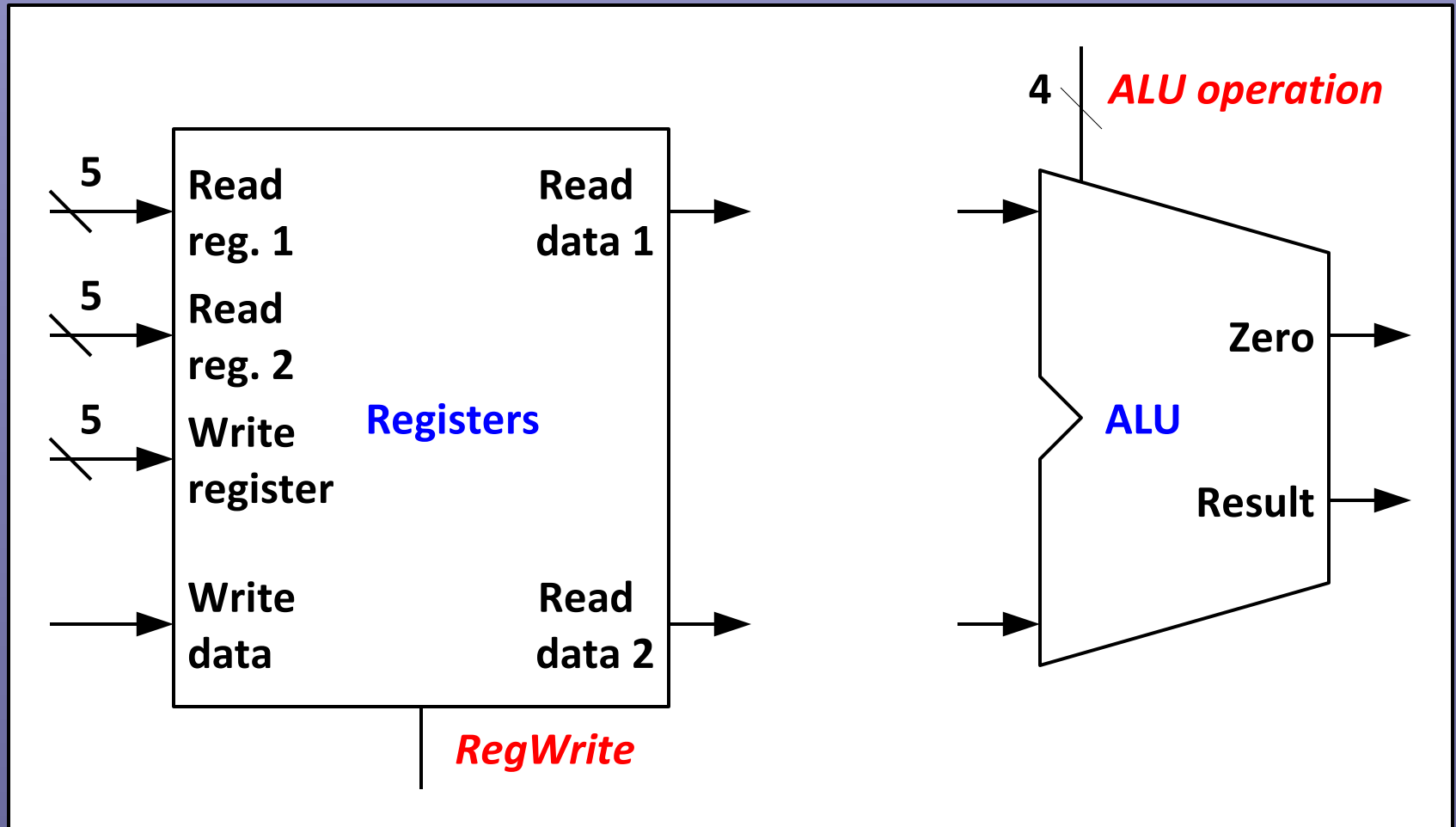
# MIPS: Prvky datové cesty (1)

## Čtení instrukce



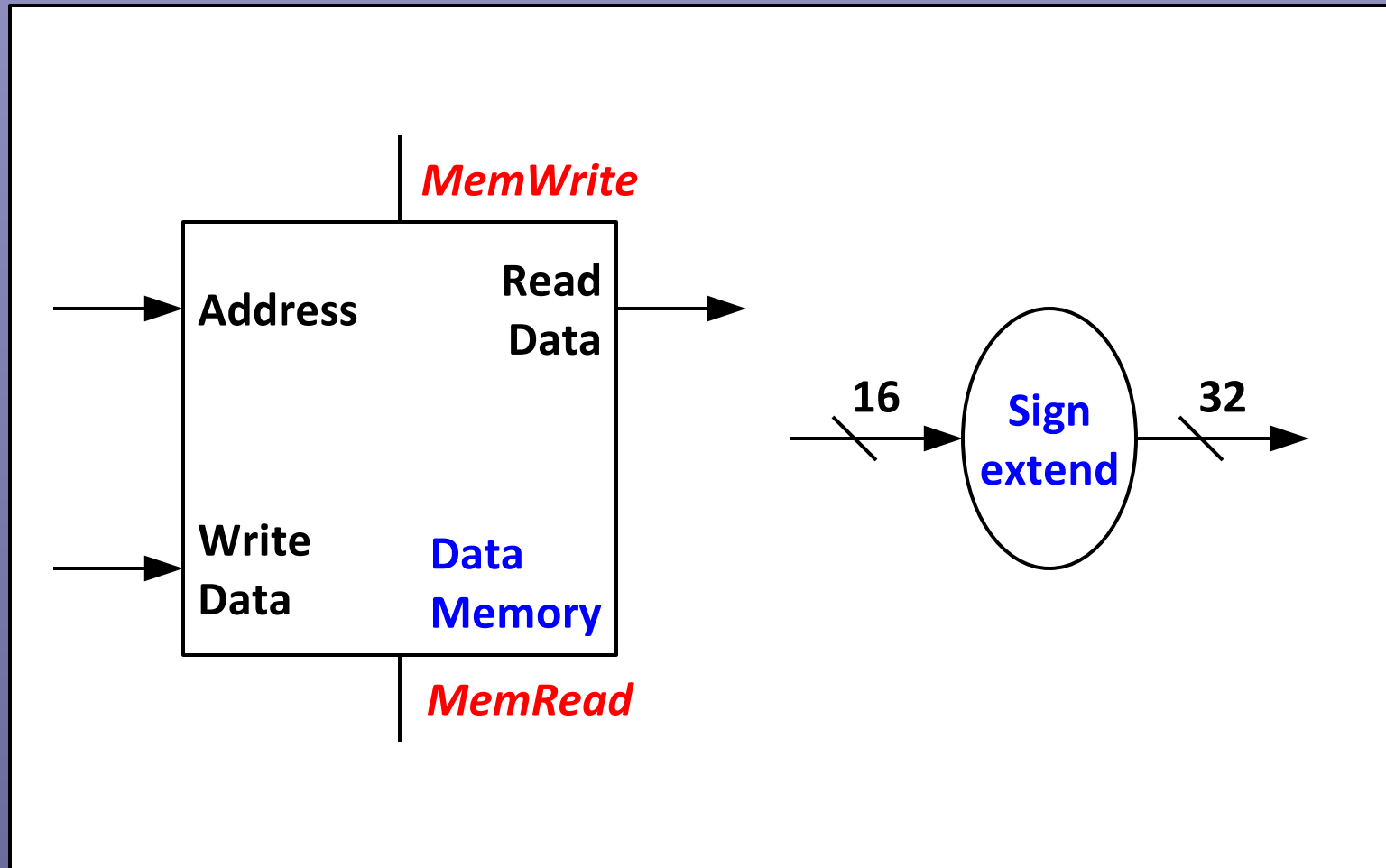
# MIPS: Prvky datové cesty (2)

## Operace s daty v registrech



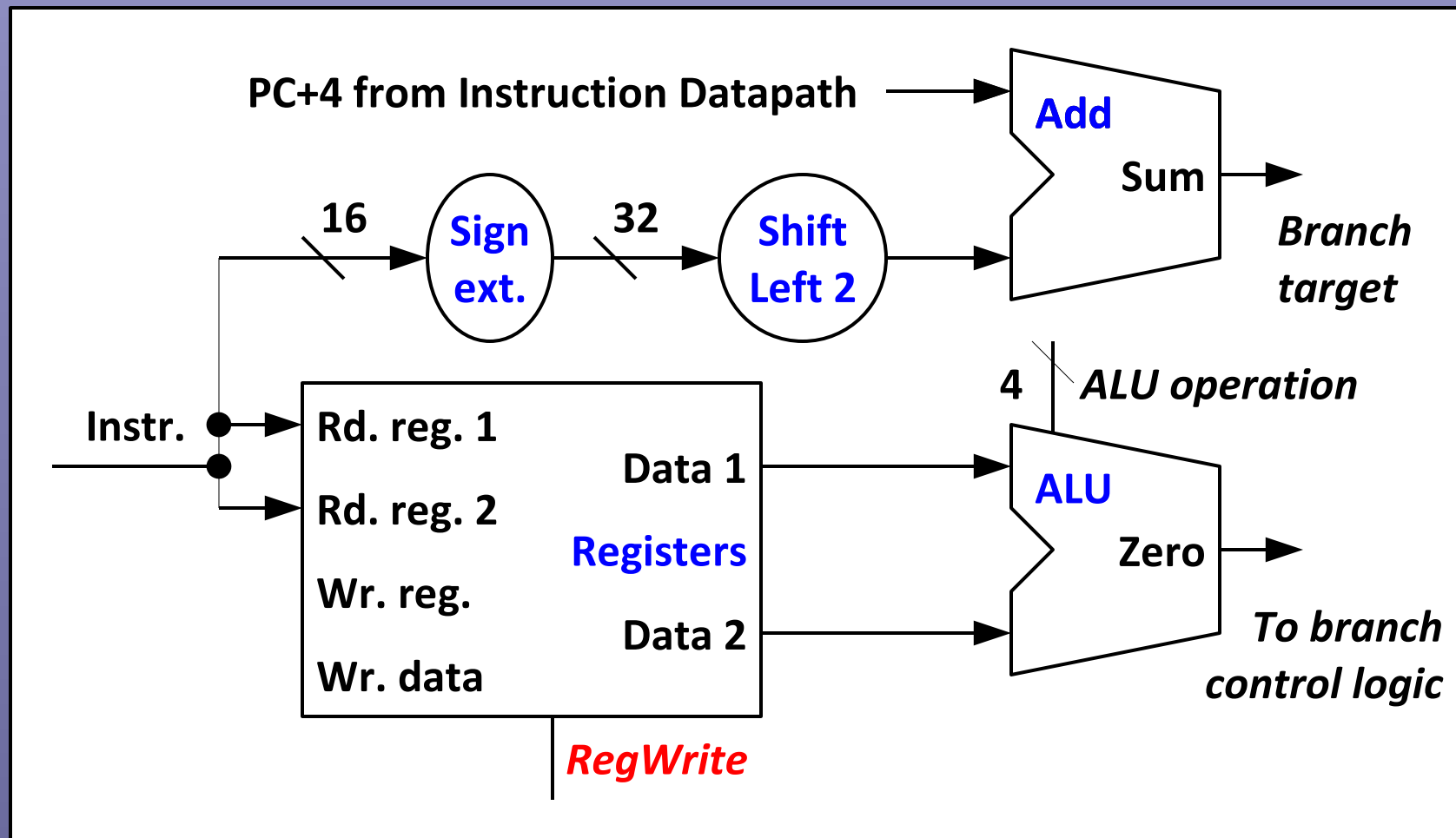
# MIPS: Prvky datové cesty (3)

Čtení/zápis z/do paměti

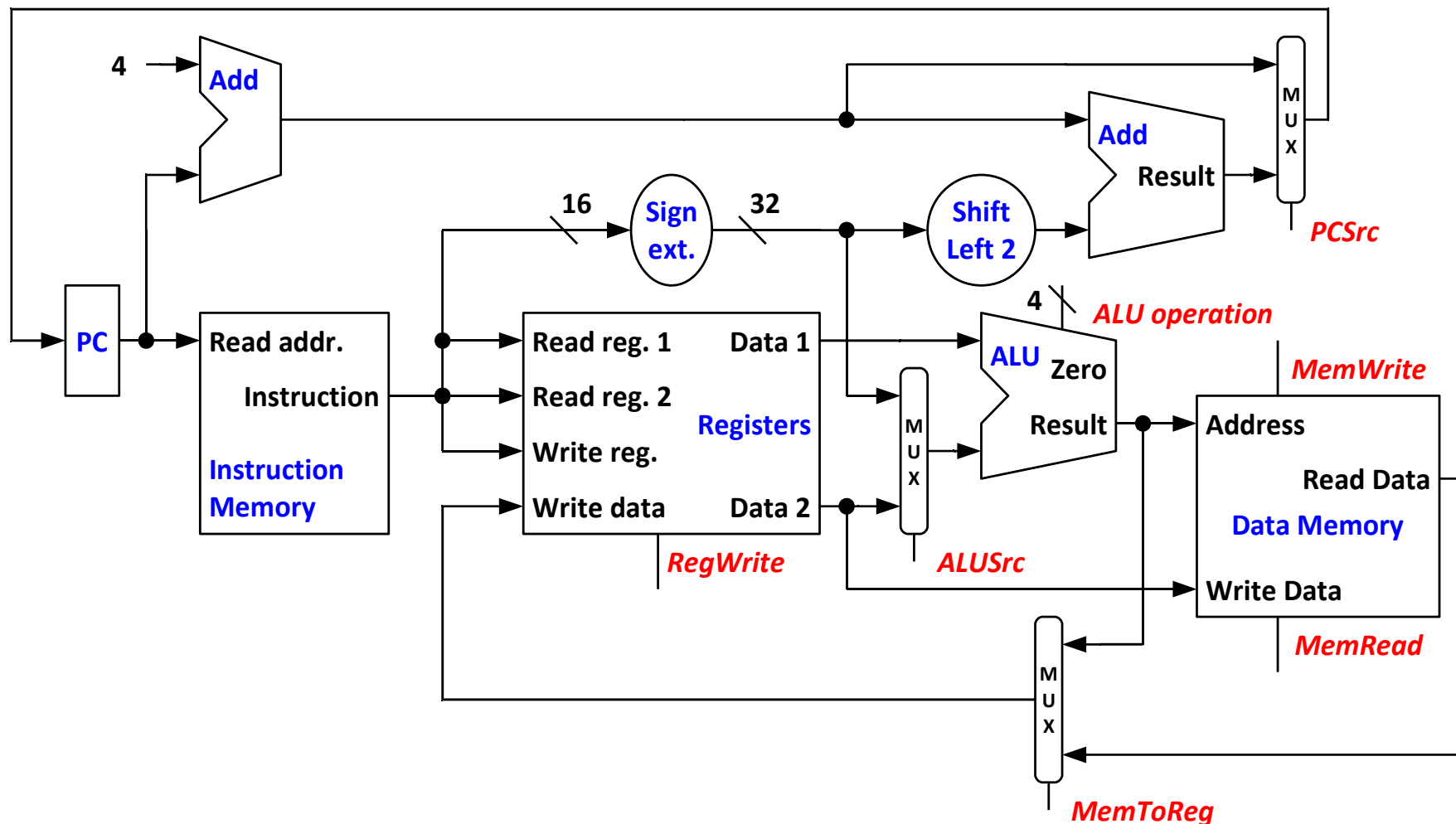


# MIPS: Prvky datové cesty (4)

## Podmíněný skok



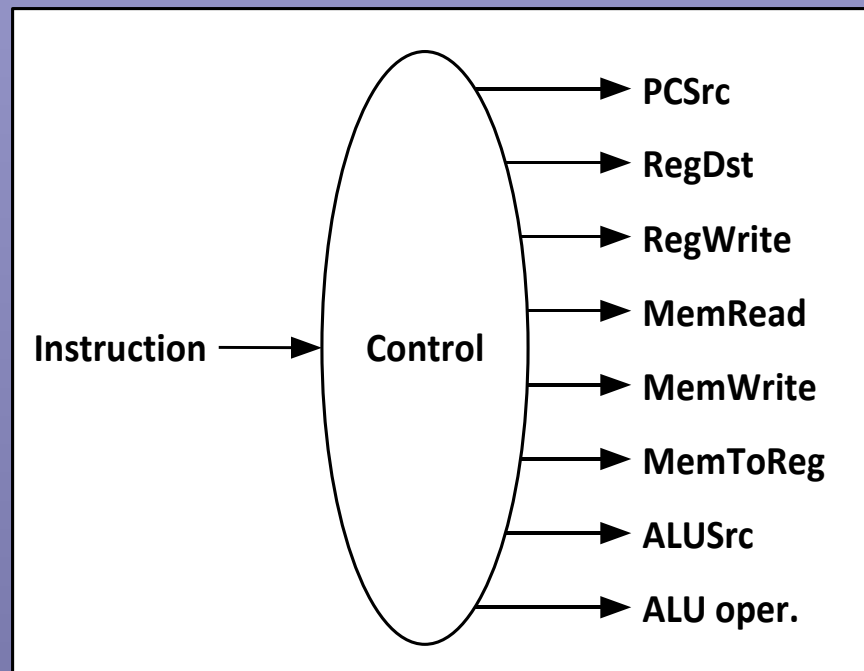
# MIPS: Datová cesta



# 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

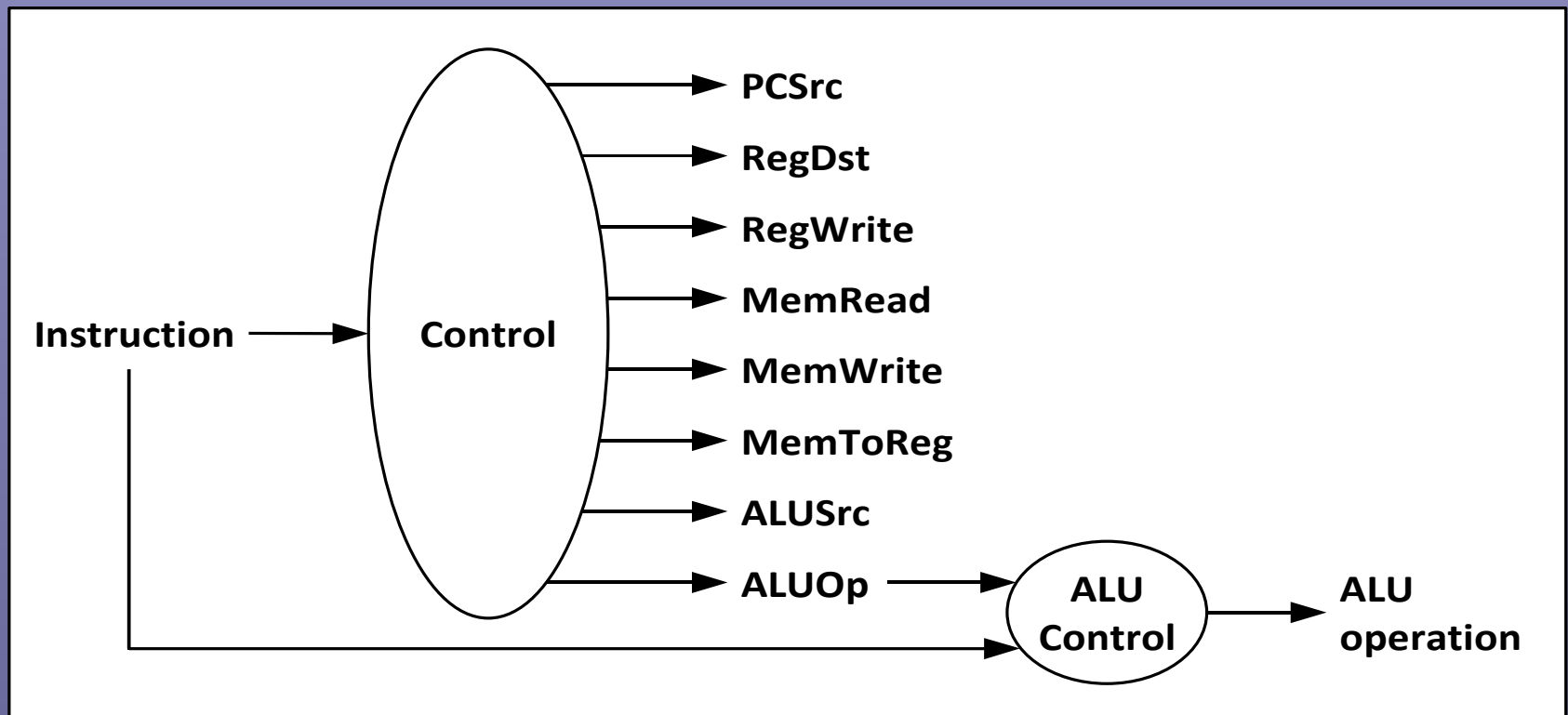
op	rs	rt	rd	shamt	funct
op	rs	rt	address/immediate		
op	target address				

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		

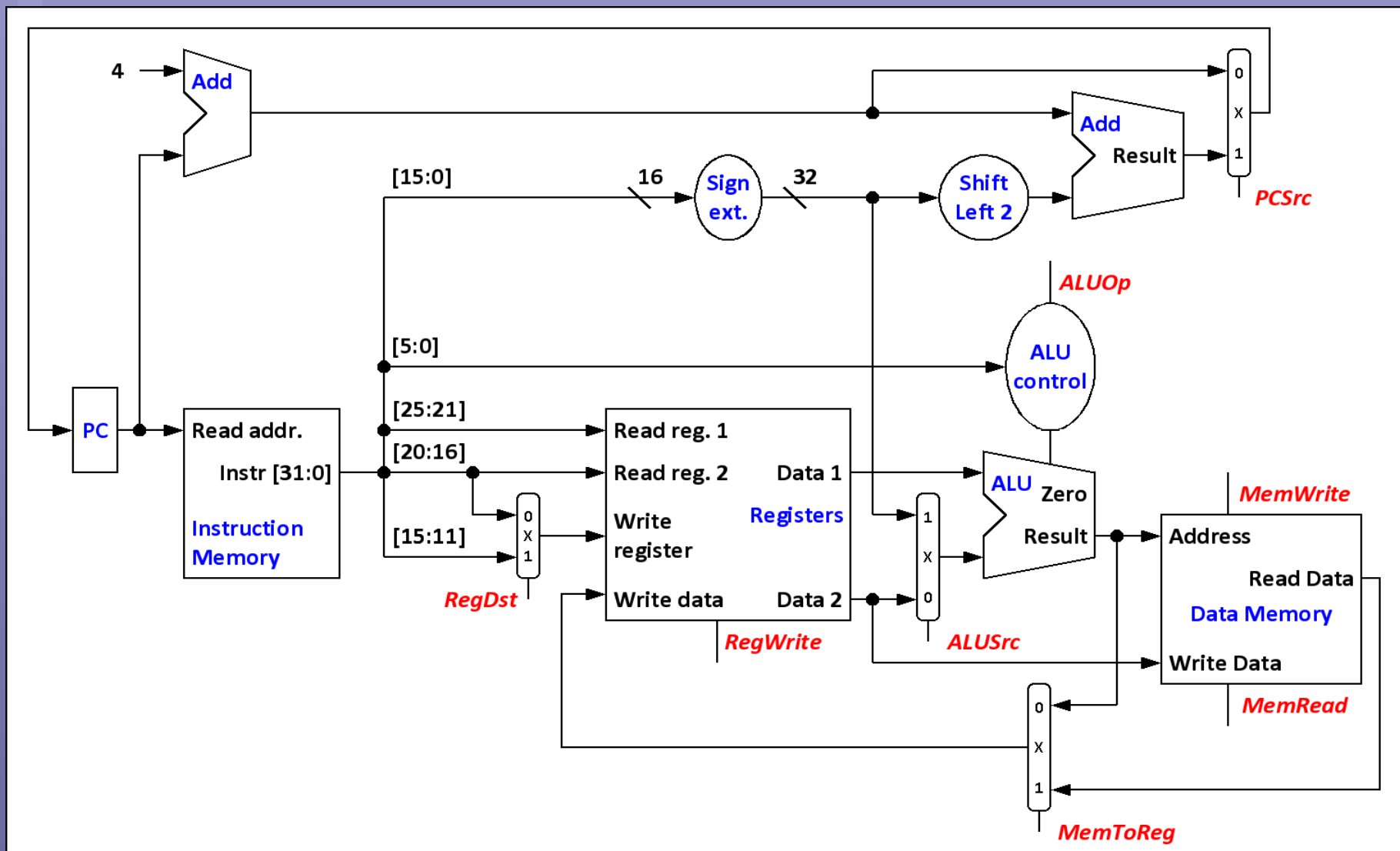
# Víceúrovňové dekódování

## 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)

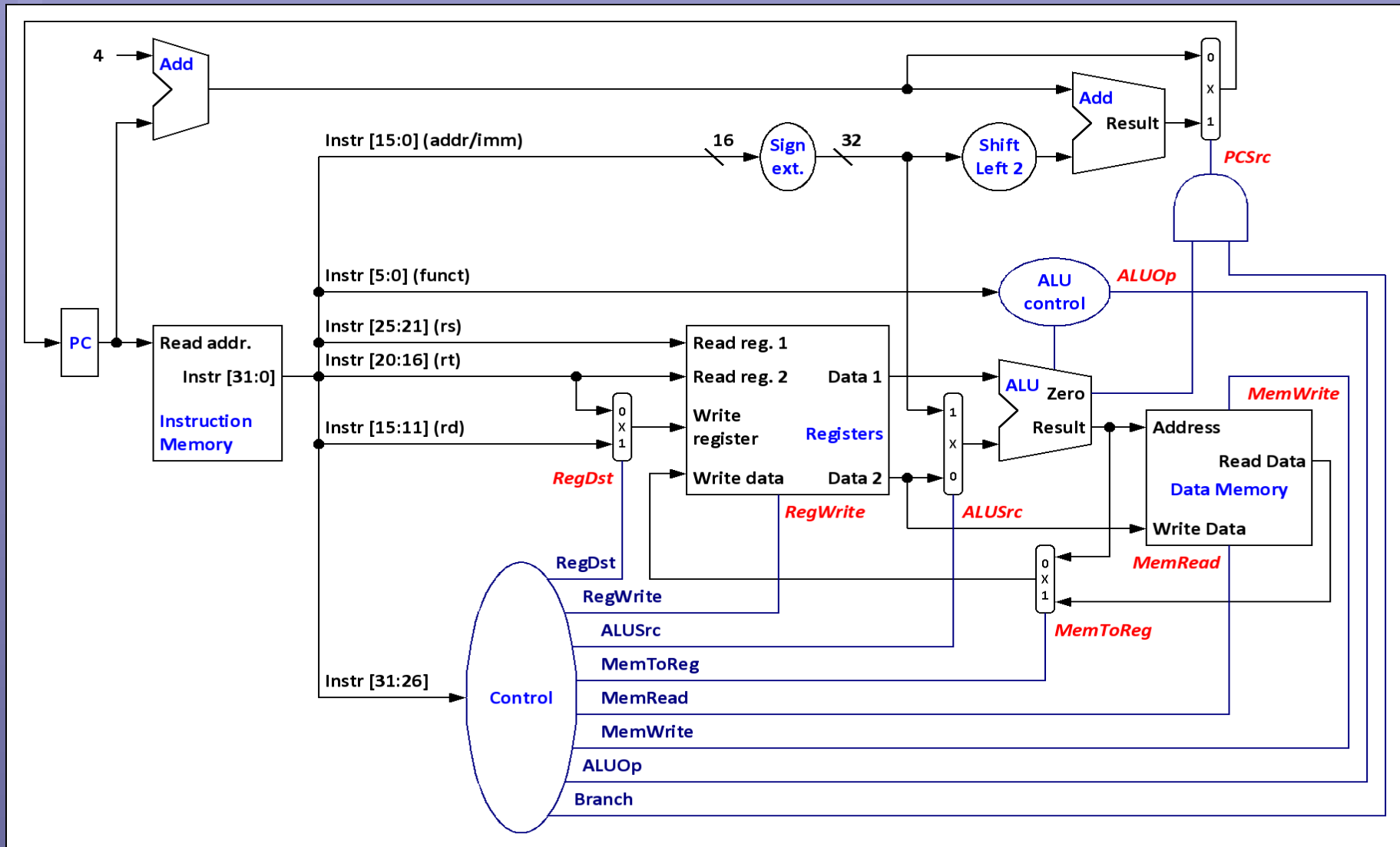
## Dekódování instrukce

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

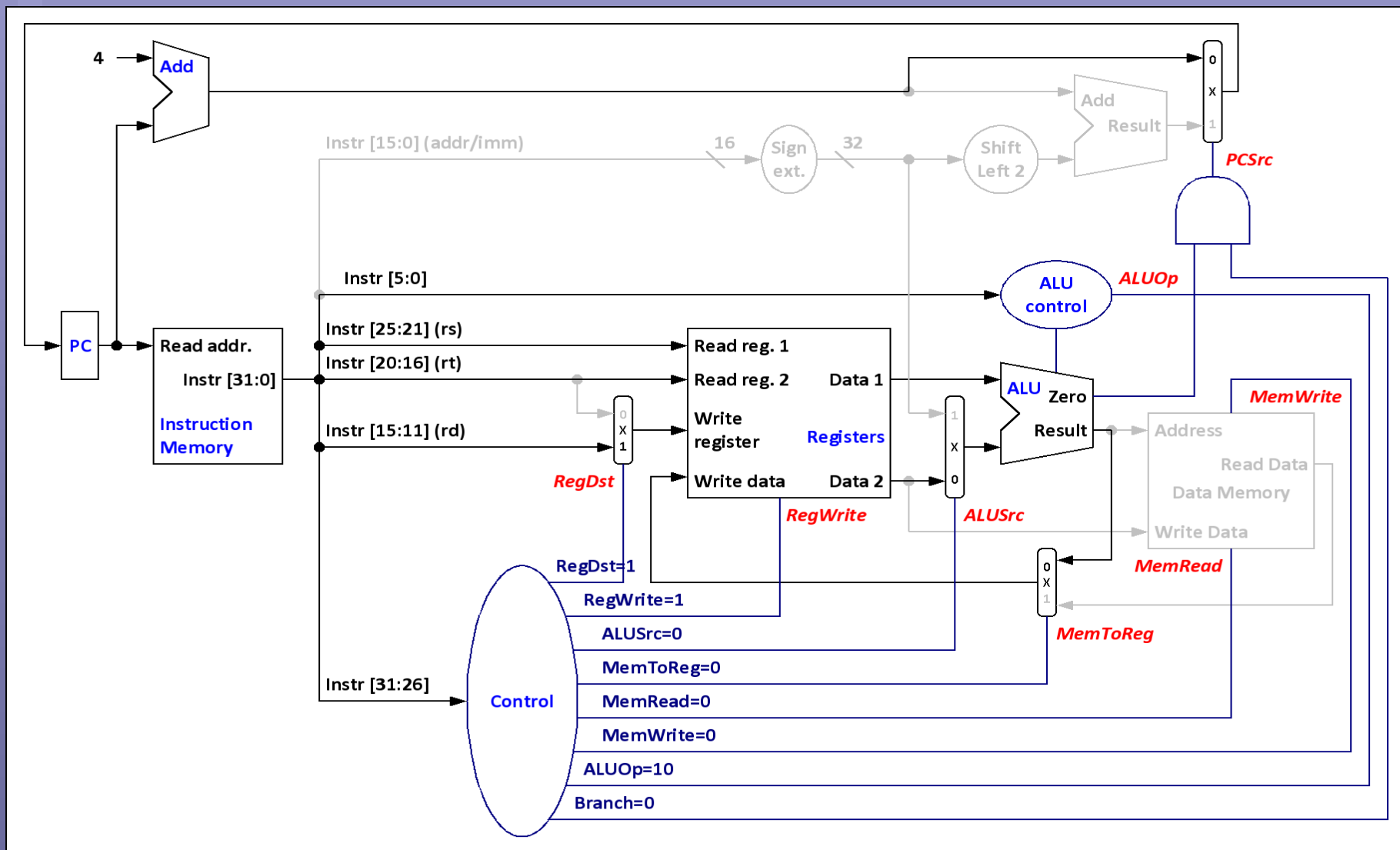
- řídicí signály plně určeny operačním kódem

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

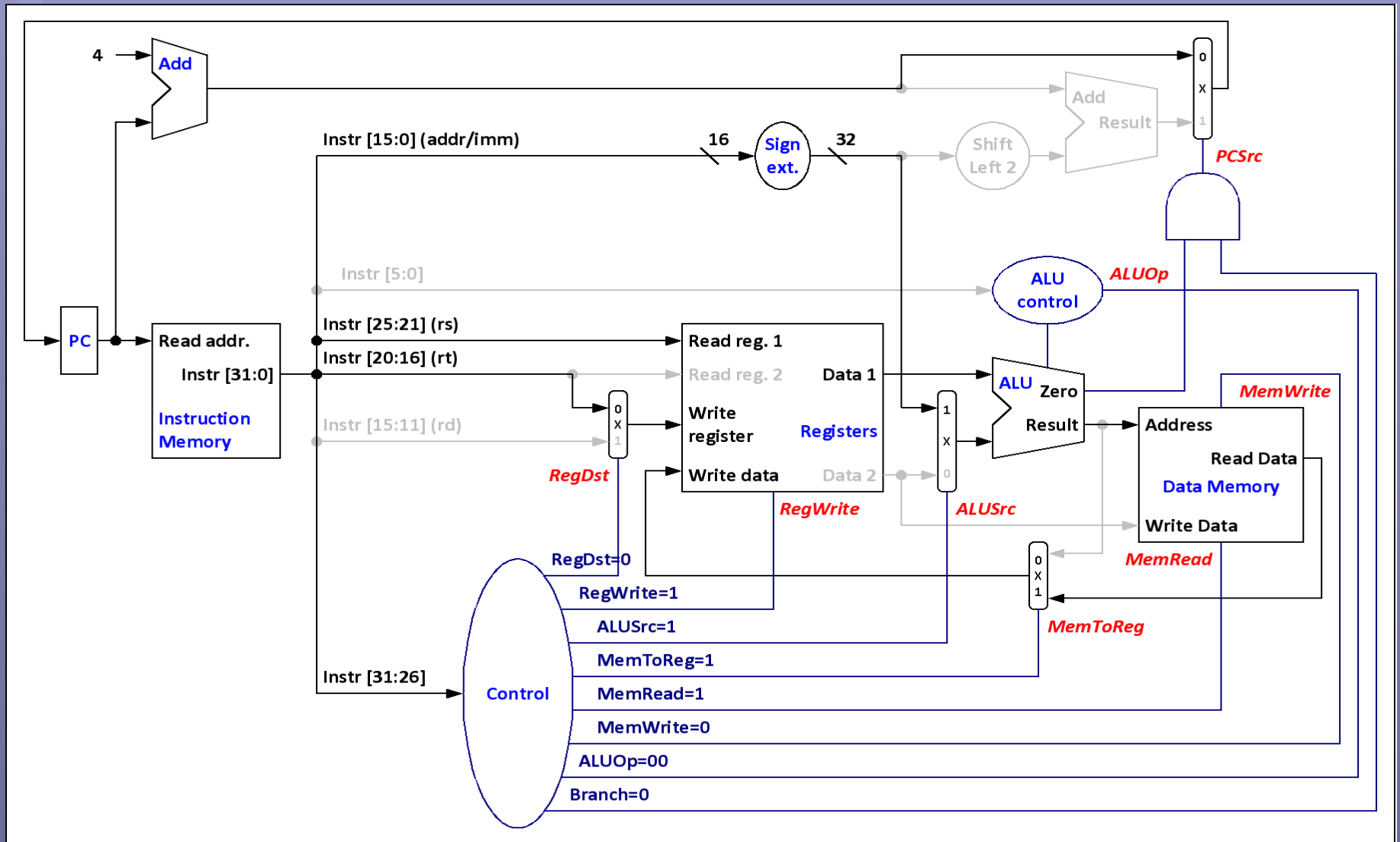
# MIPS: Datová cesta s hlavním řadičem



# MIPS: Instrukce typu R-format

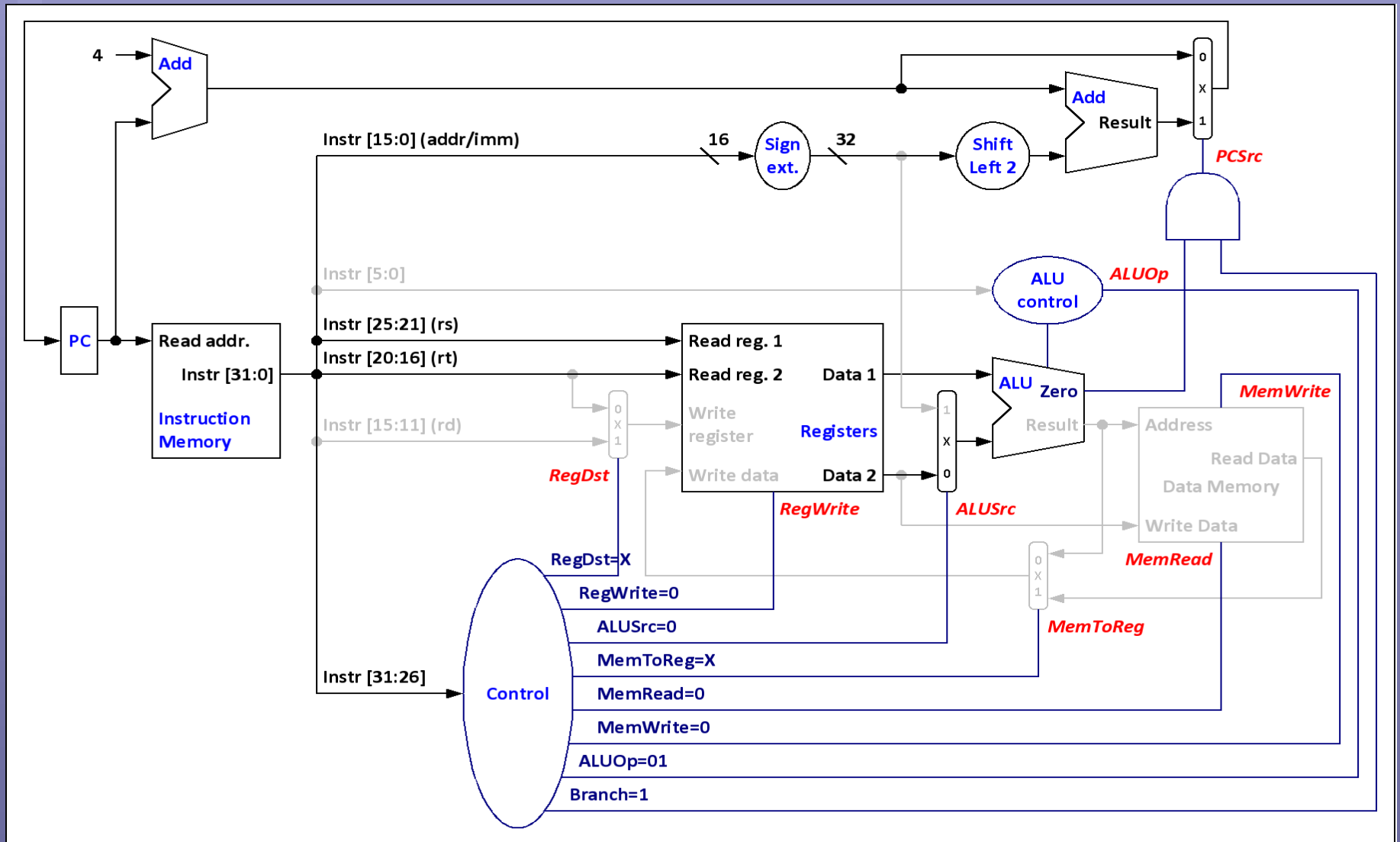


# MIPS: Instrukce Load Word





# 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

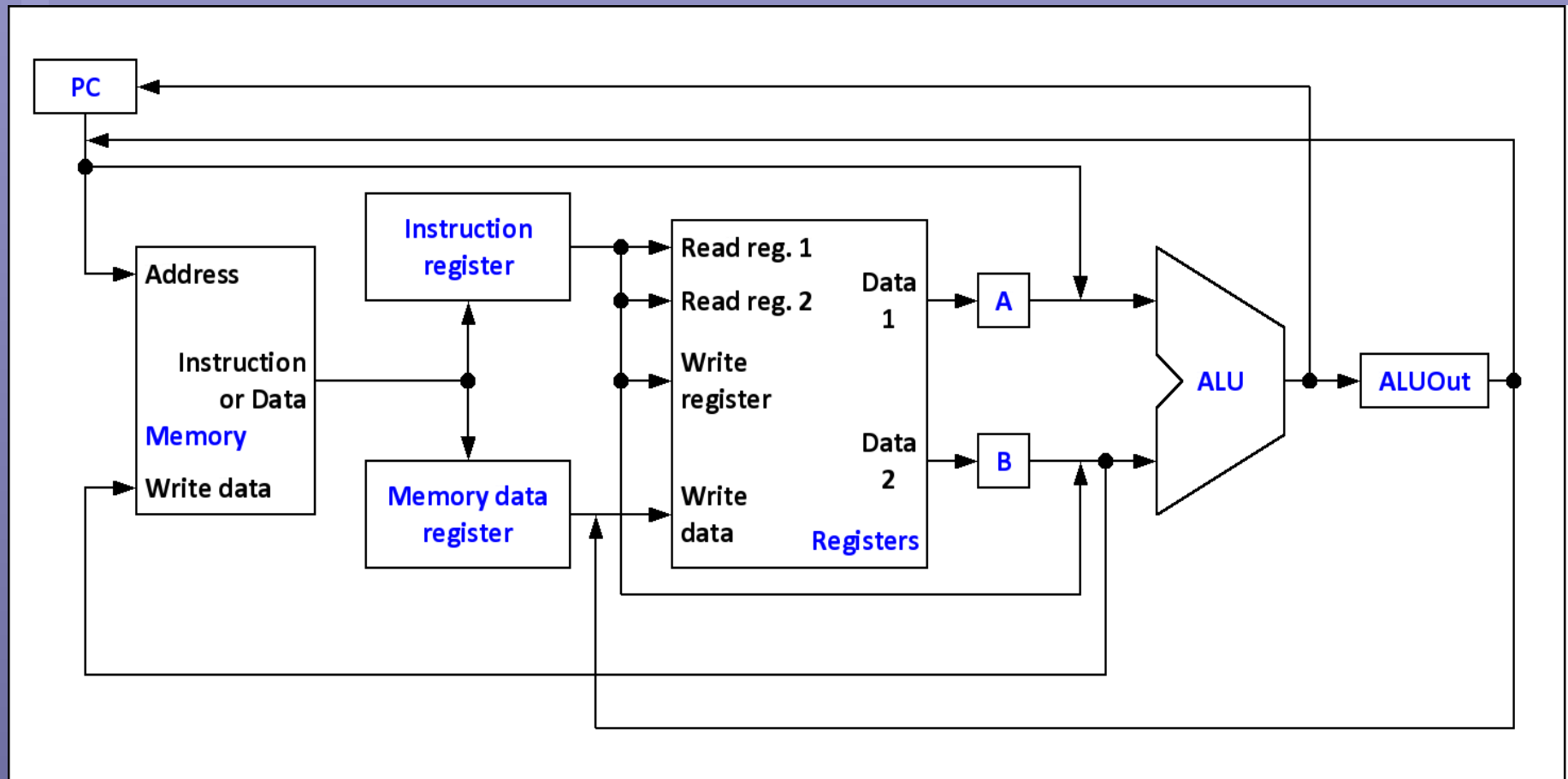
## Základní princip

- 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

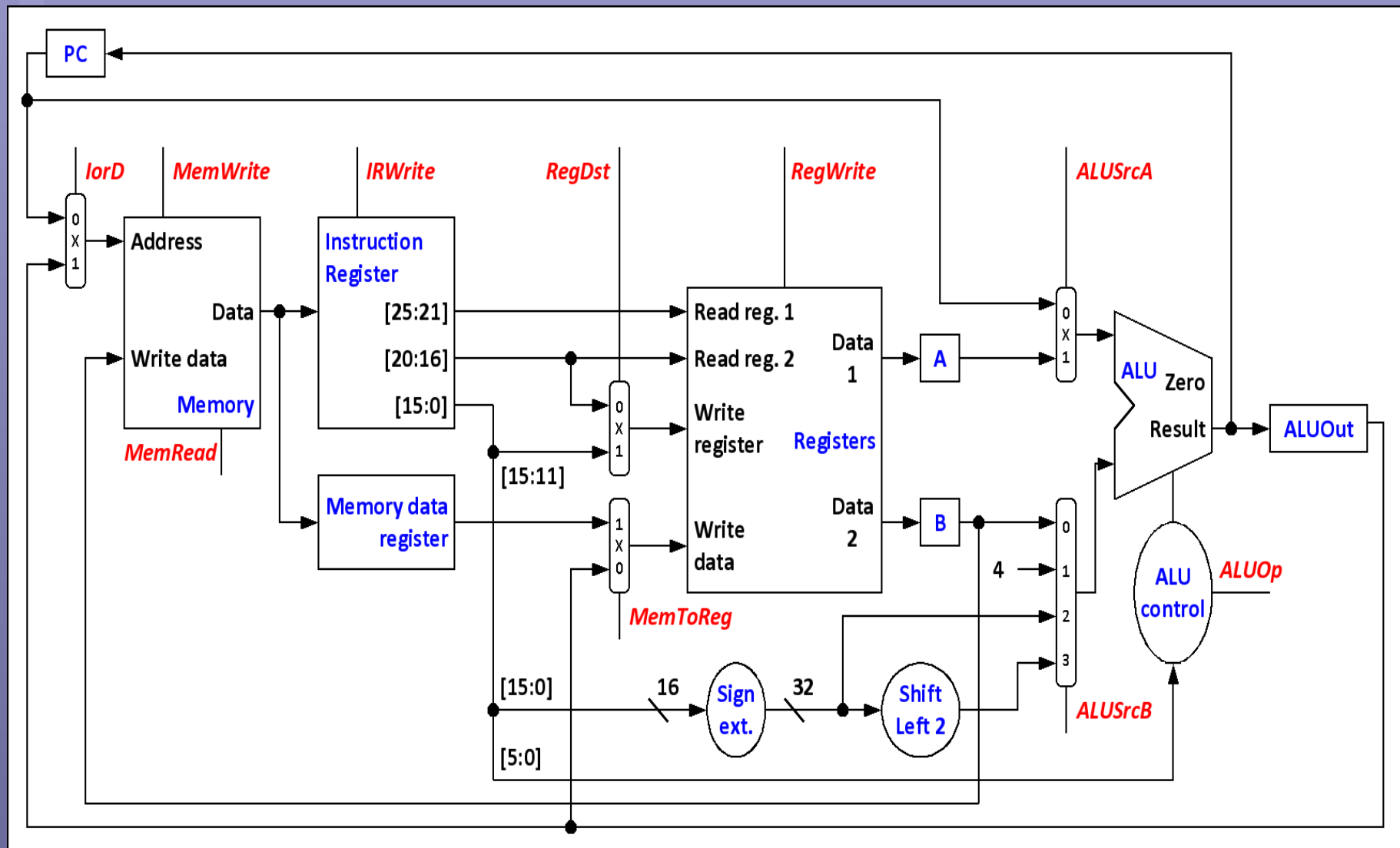
## Hlavní výhody

- 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

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

# MIPS: Instrukční cyklus (5)

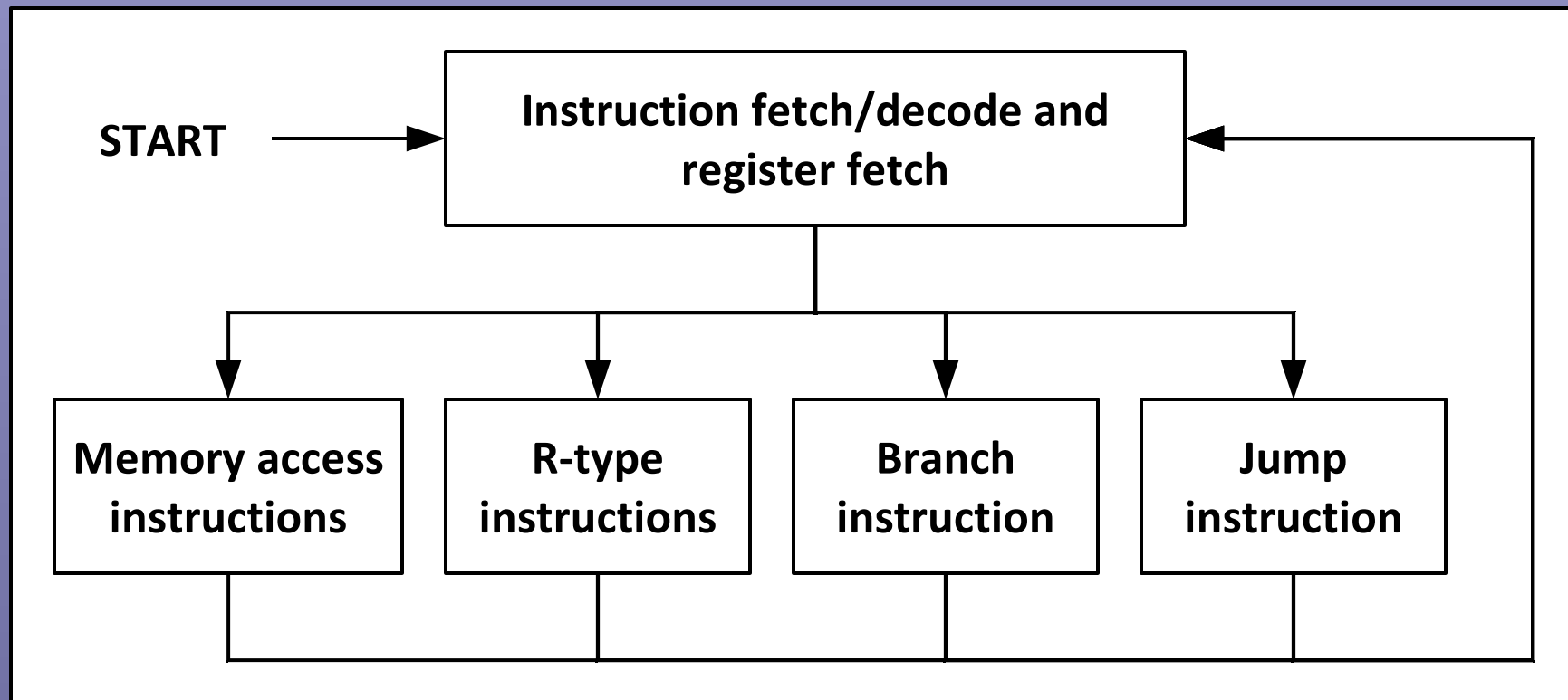
---

## Dokončení čtení z paměti

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

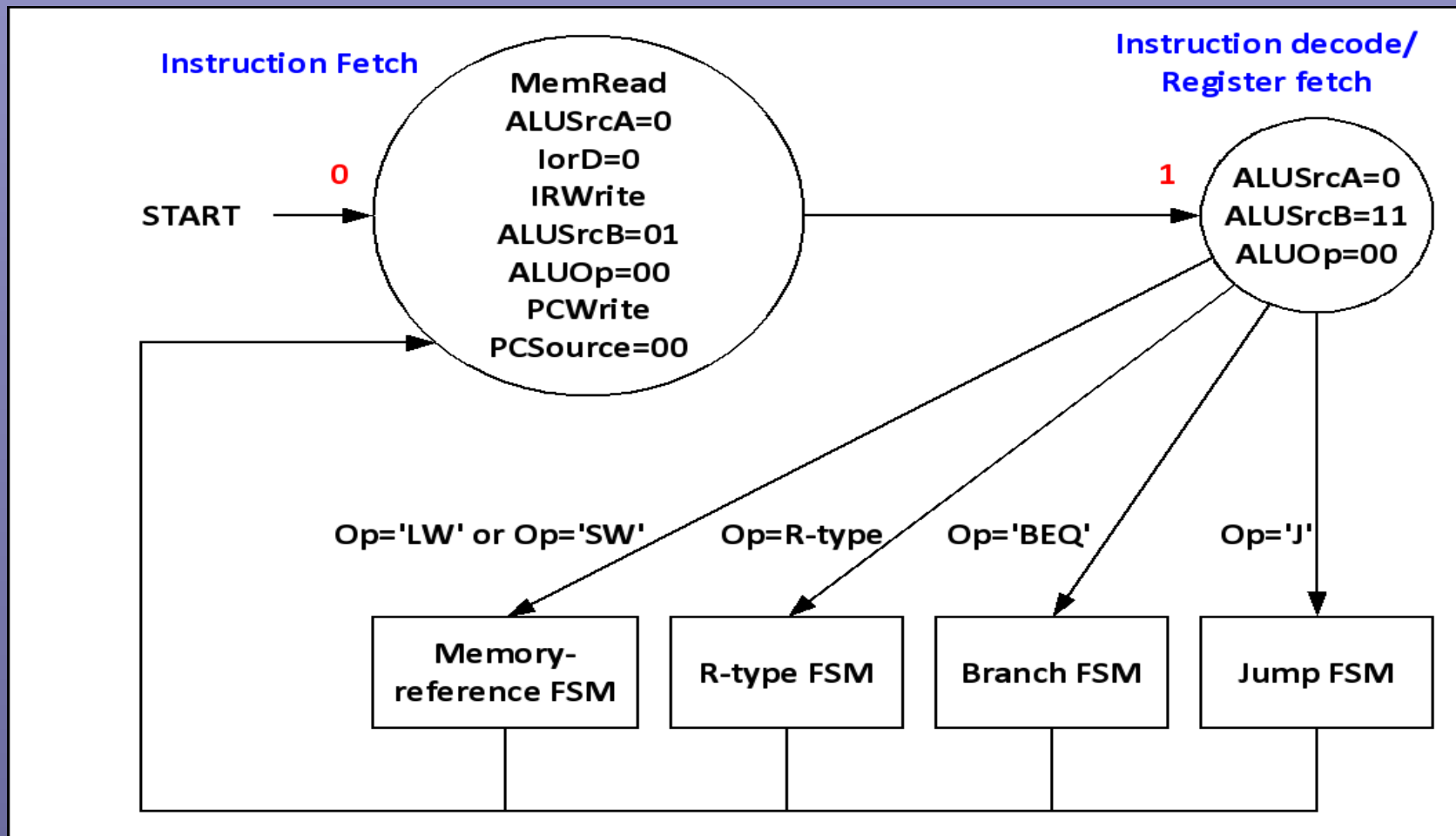
# MIPS: Řadič jako konečný automat (1)

## Hrubé schéma



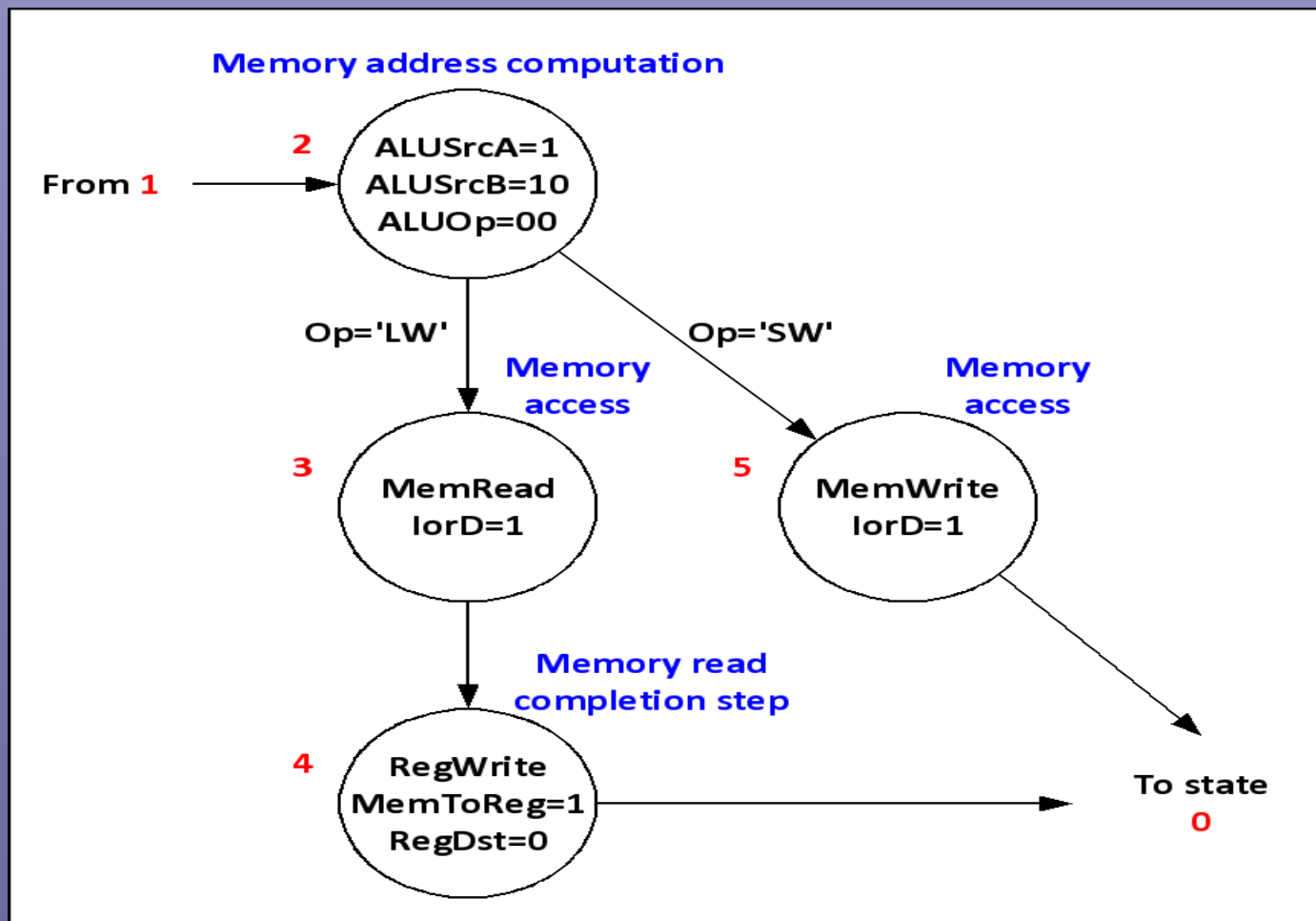
# MIPS: Řadič jako konečný automat (2)

## Instruction fetch & decode



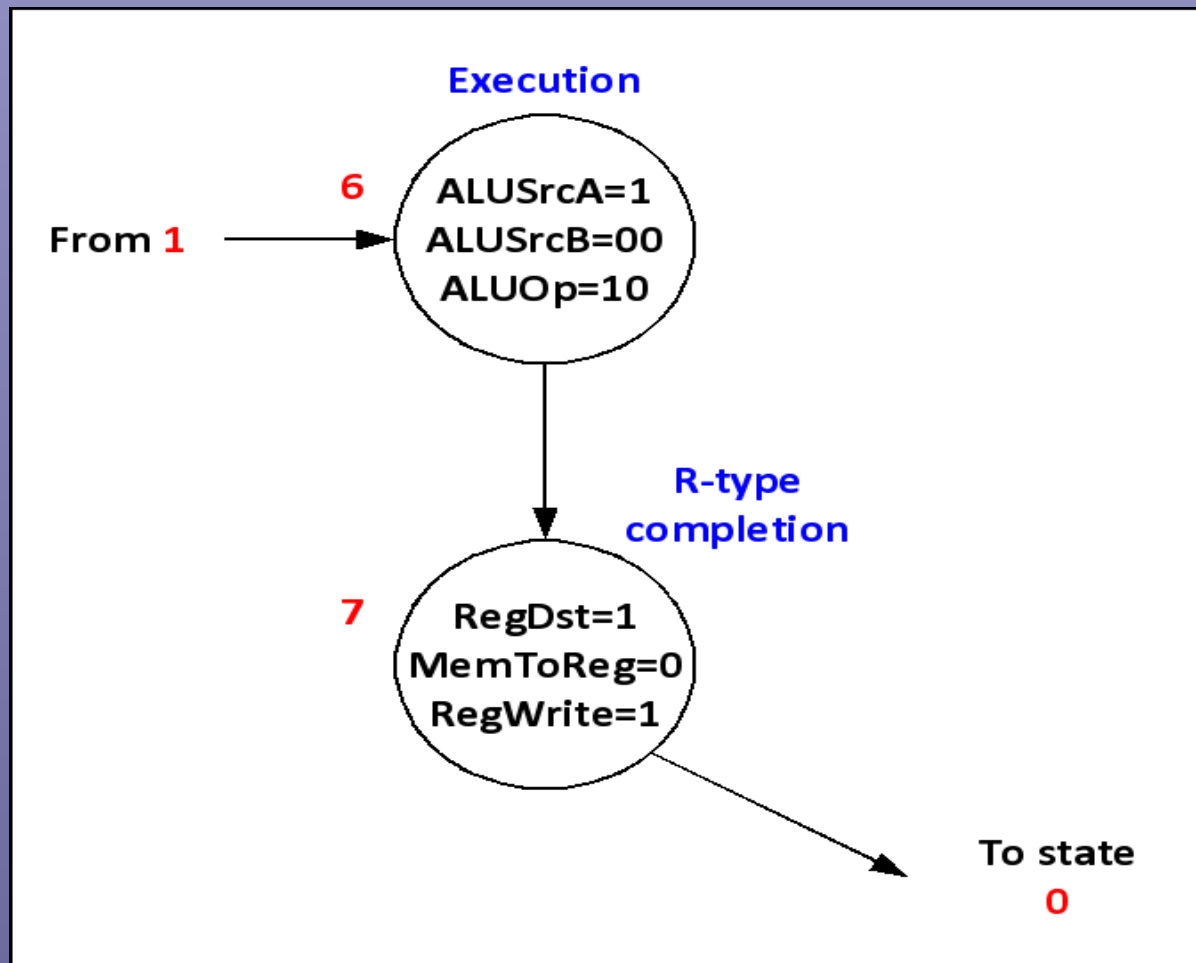
# MIPS: Řadič jako konečný automat (3)

## Memory reference FSM



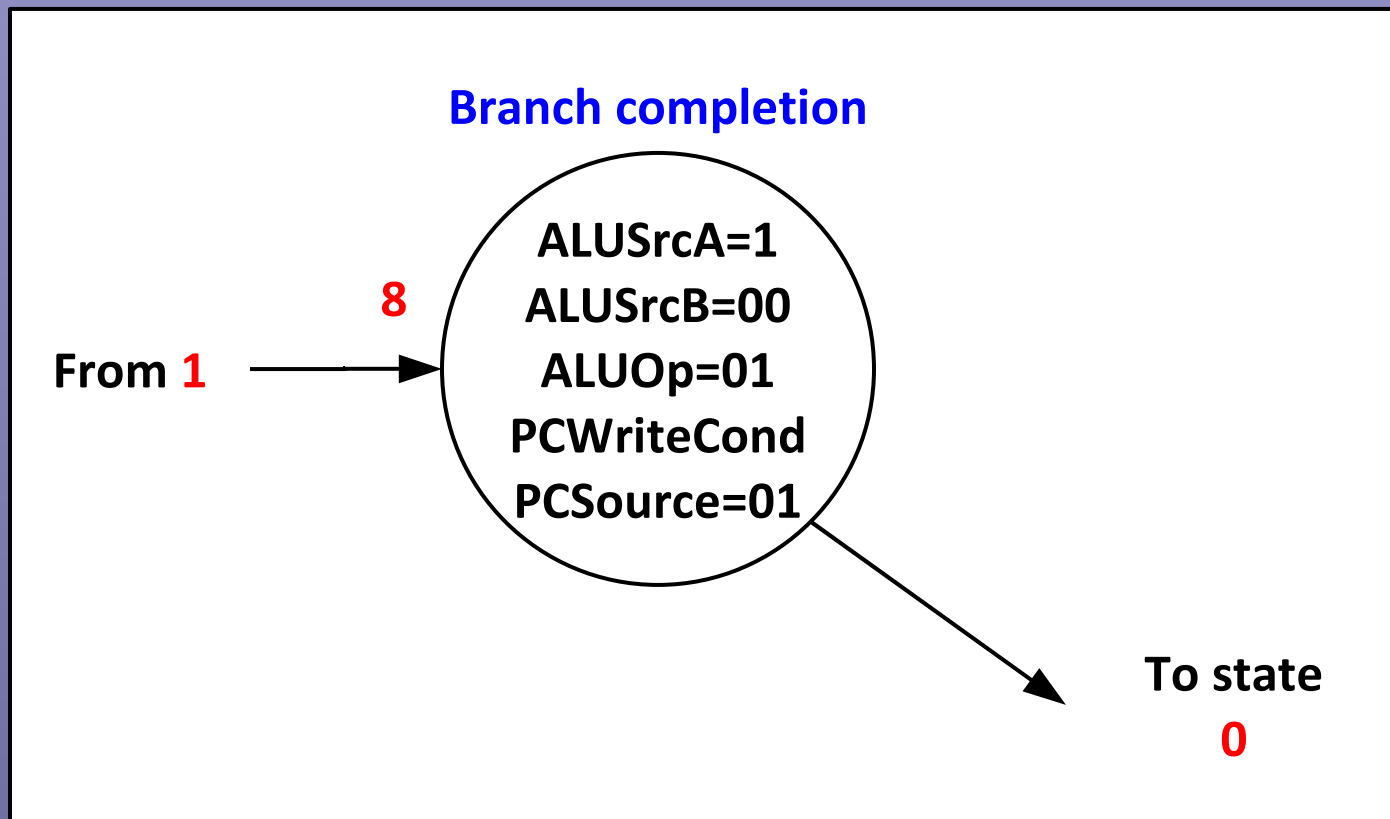
# MIPS: Řadič jako konečný automat (4)

## R-type instruction



# MIPS: Řadič jako konečný automat (5)

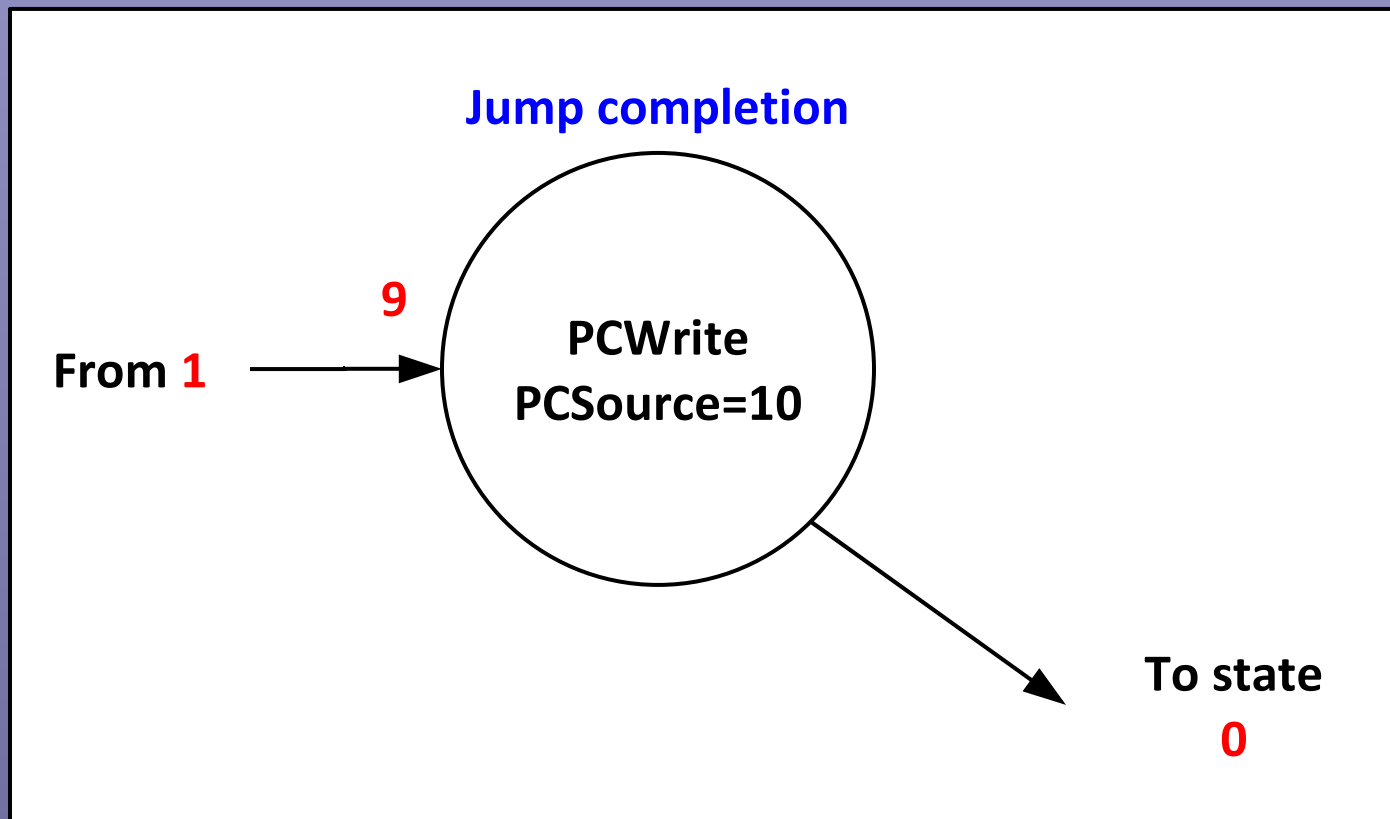
## Branch instruction



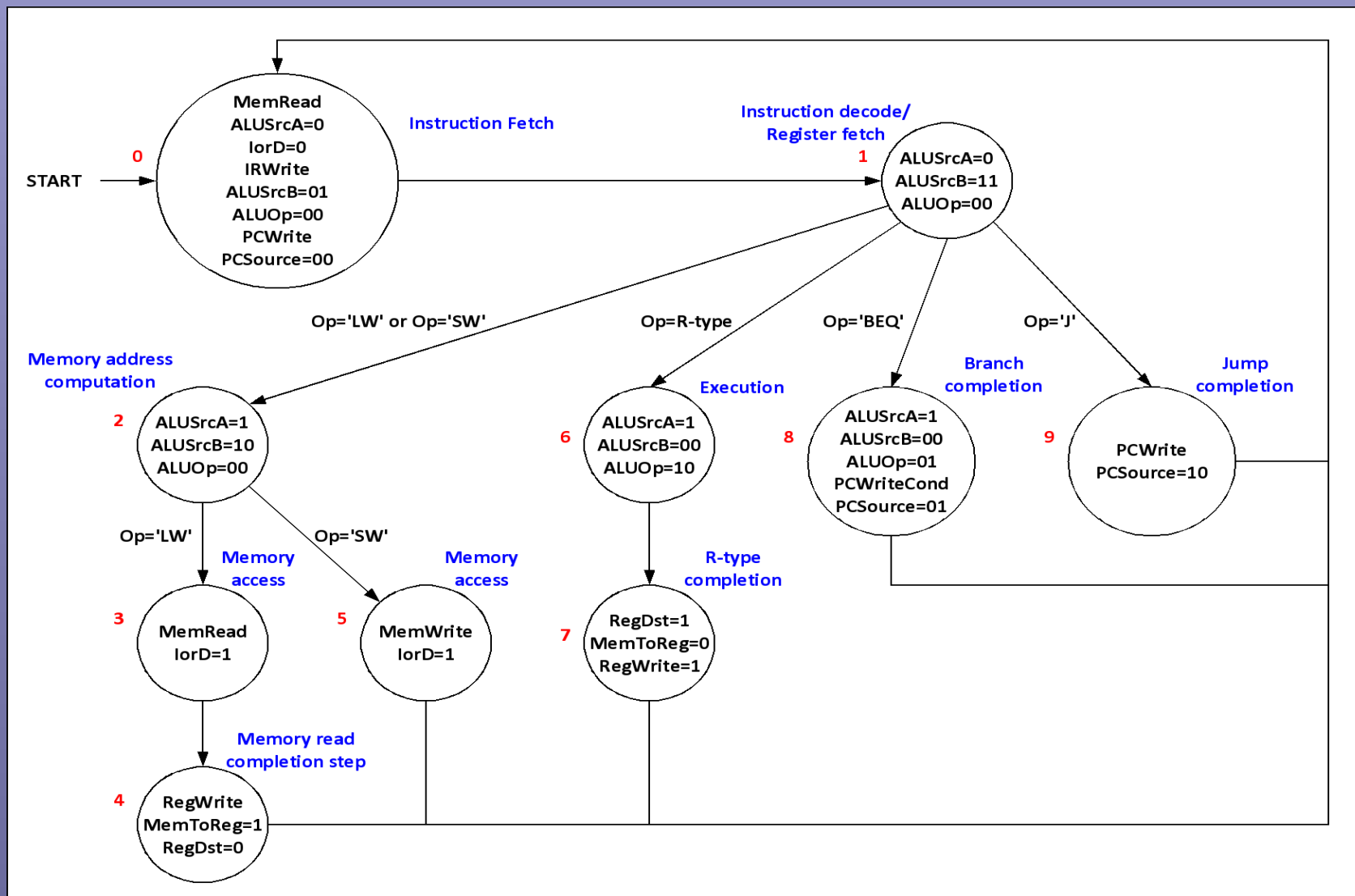


# MIPS: Řadič jako konečný automat (6)

## Jump instruction



# MIPS: Řadič jako konečný automat (7)



# Výjimky a přerušení

## Neočekávaná změna toku provádění instrukcí

- jiná příčina než jump/branch
- vnitřní (exception)
  - ♦ aritmetické přetečení
  - ♦ nedefinovaná instrukce
  - ♦ vyvolání služby operačního systému
  - ♦ selhání hardware
- vnější (interrupt)
  - ♦ periferní zařízení
  - ♦ selhání hardware

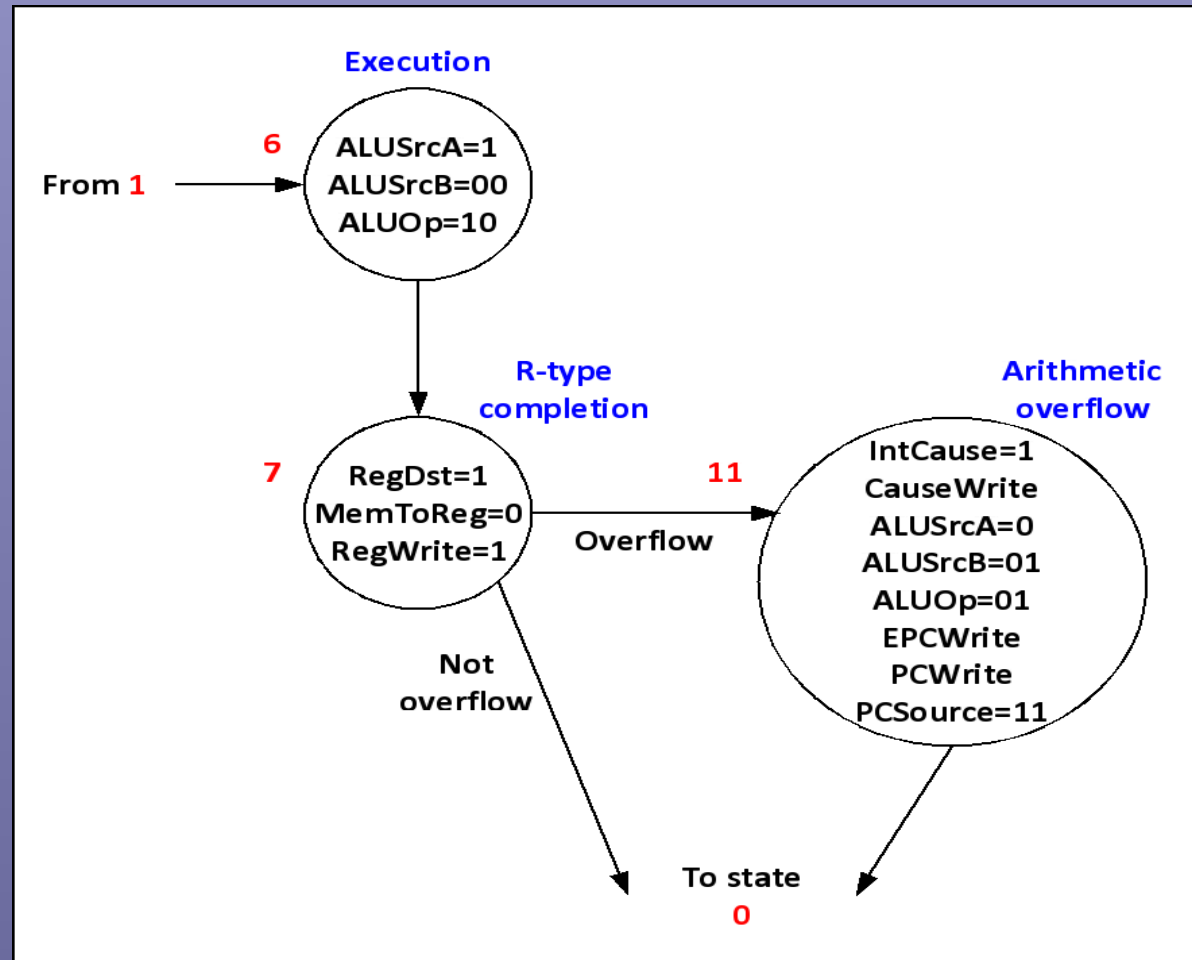
# Podpora výjimek a přerušení

## 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

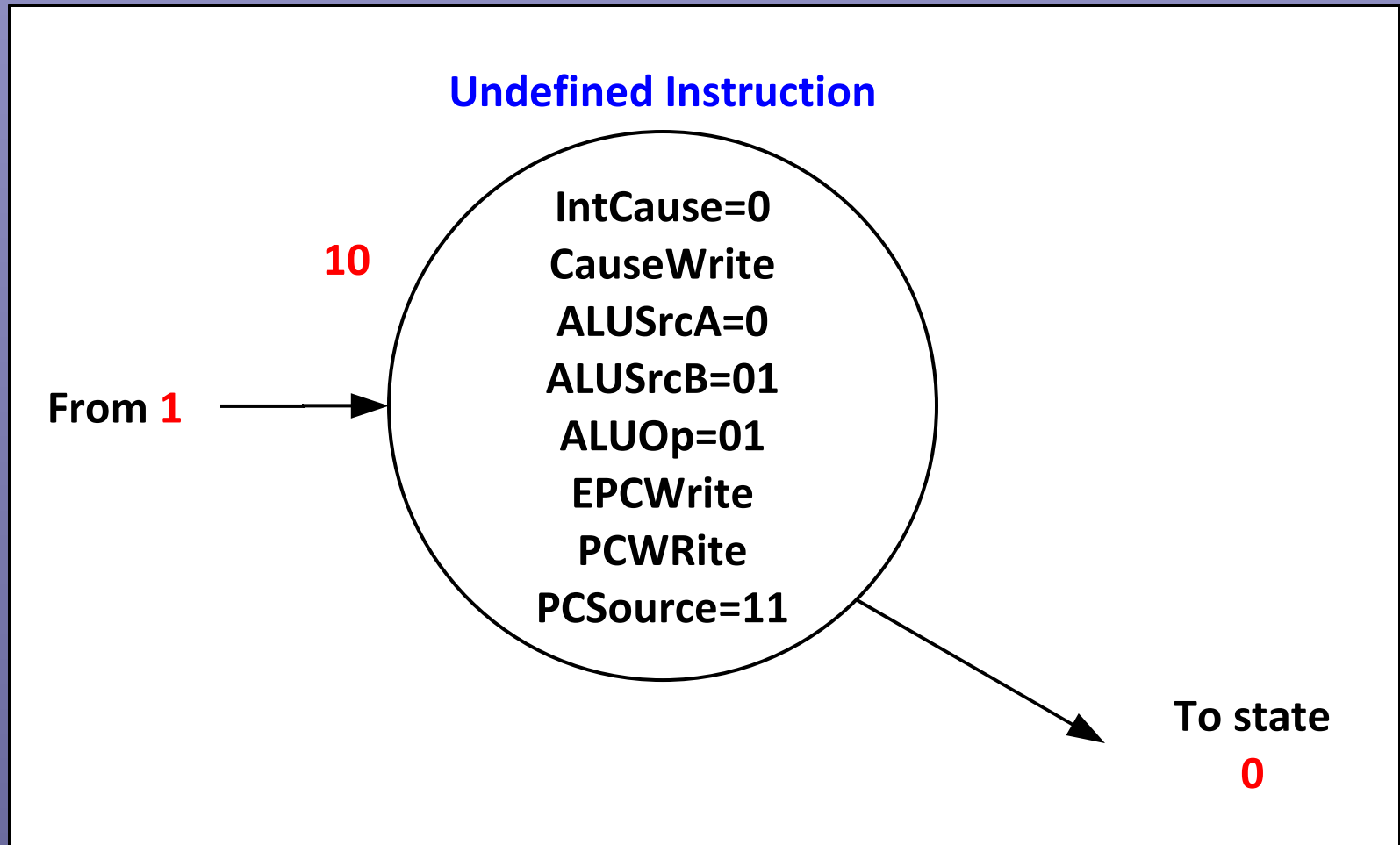
# MIPS: Podpora výjimek: přetečení

## R-type instruction



# MIPS: Podpora výjimek: neplatná instrukce

## Undefined instruction



# Obsluha výjimek a přerušení

## Realizuje software

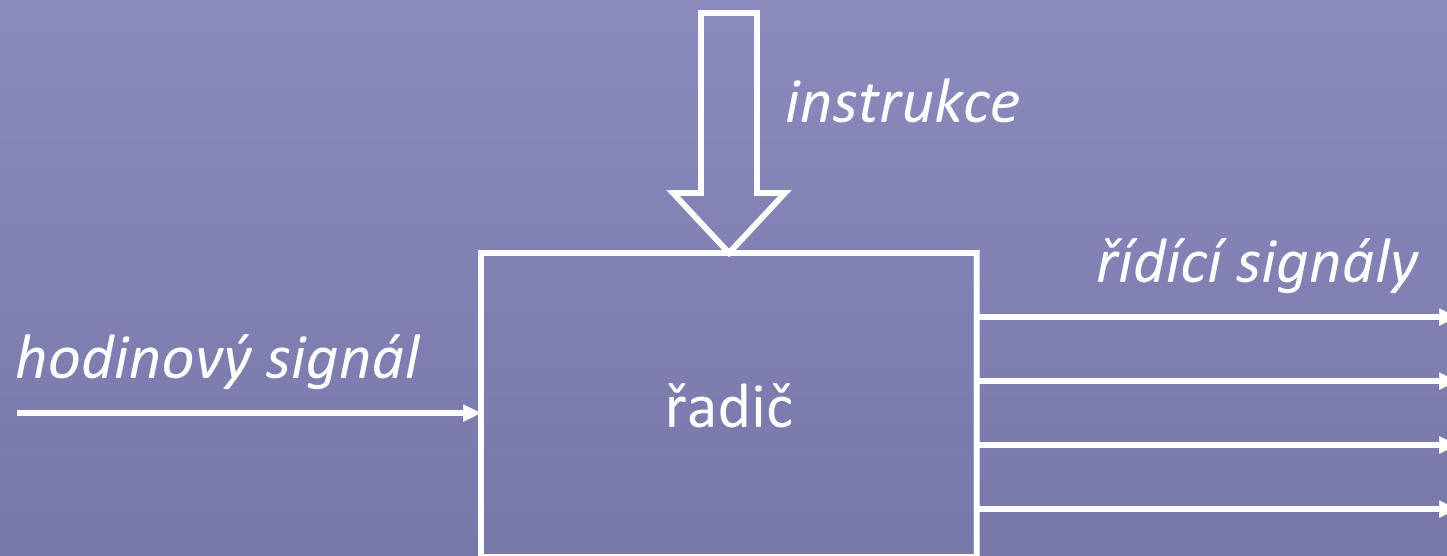
- uschování stavu původního výpočtu
- zjištění příčiny výjimky/přerušení
- obsluha příslušného typu výjimky
  - ♦ může dojít k ukončení výpočtu
- obnovení stavu původního výpočtu
- návrat do původního programu
  - ♦ pokračovat následující instrukcí
  - ♦ restartovat instrukci, která výjimku vyvolala





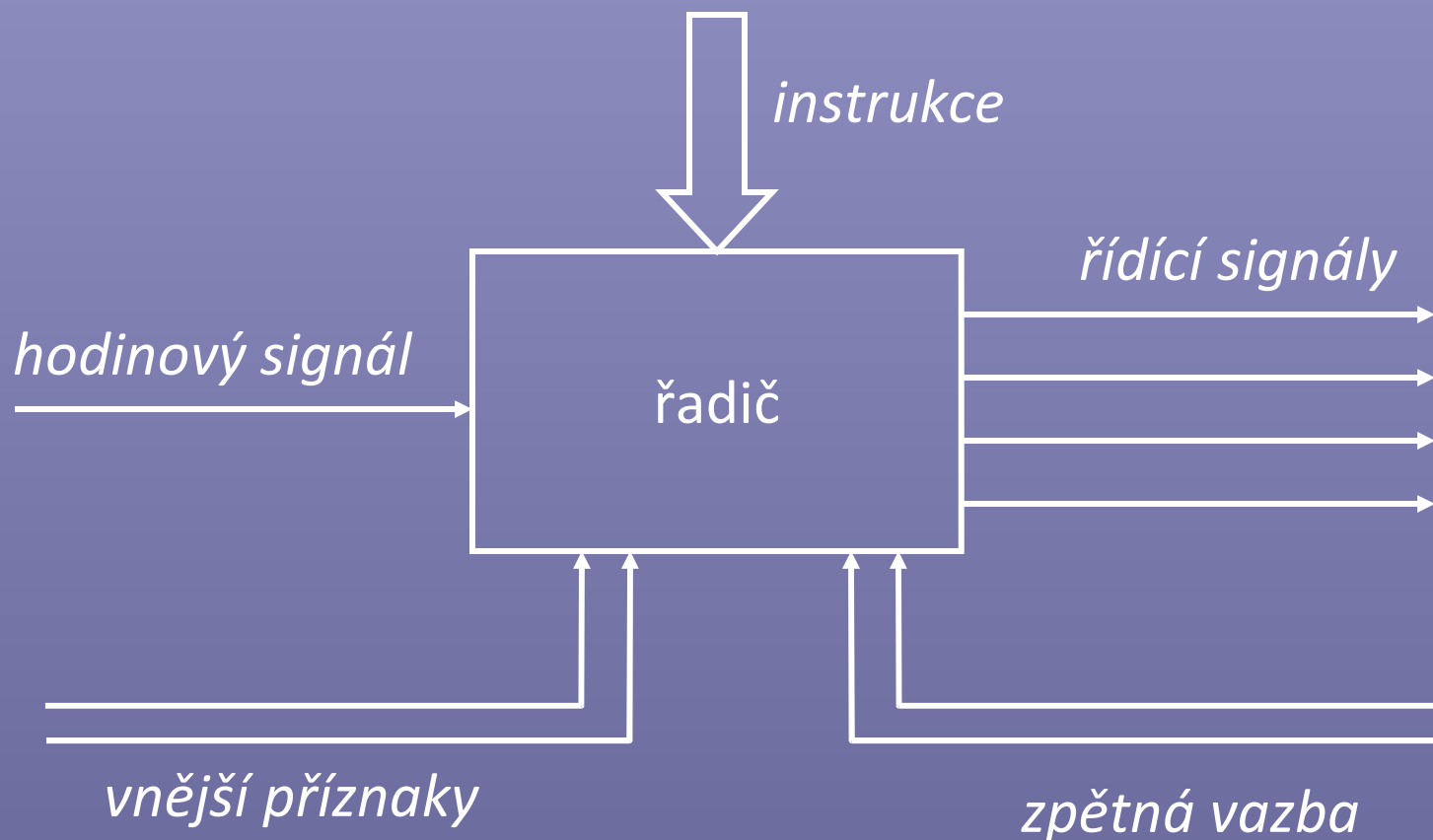
# Obecné schéma řadiče (1)

## Direktivní řadič



# Obecné schéma řadiče (2)

## Zpětnovazební řadič



# Realizace více-cyklového řadiče

---

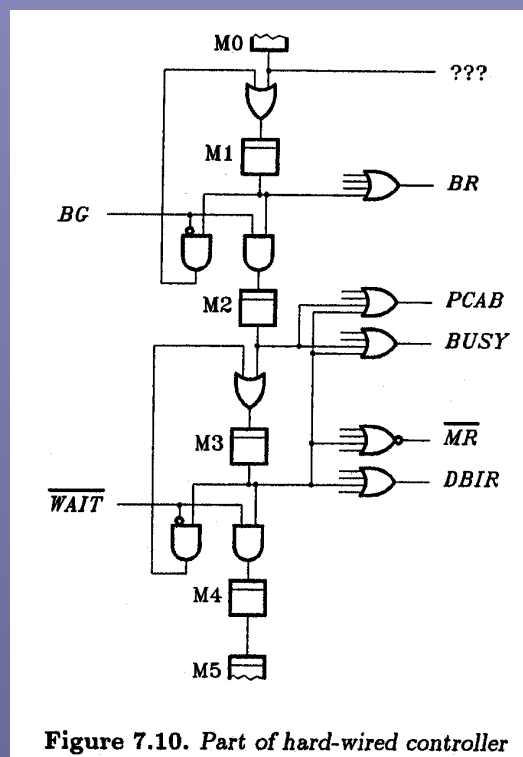
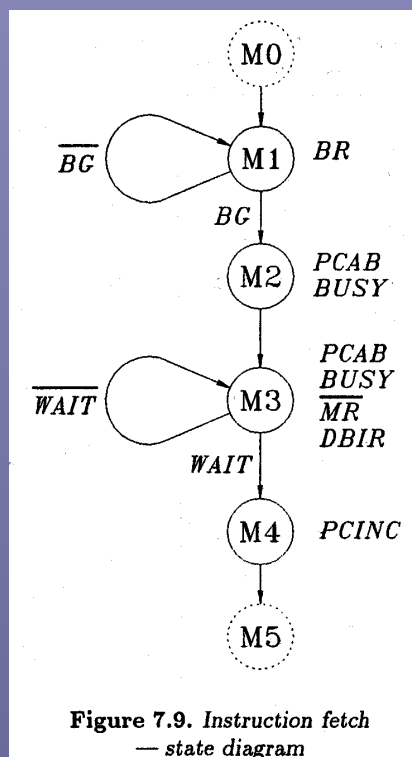
## Realizace konečného automatu

- sekvenční obvod
  - ♦ realizace závisí na reprezentaci vnitřního stavu
- obvodové řešení
  - ♦ stavový registr, kombinační logika
  - ♦ posuvný řetězec klopných obvodů
- mikroprogramování
- nanoprogramované

# Obvodový řadič

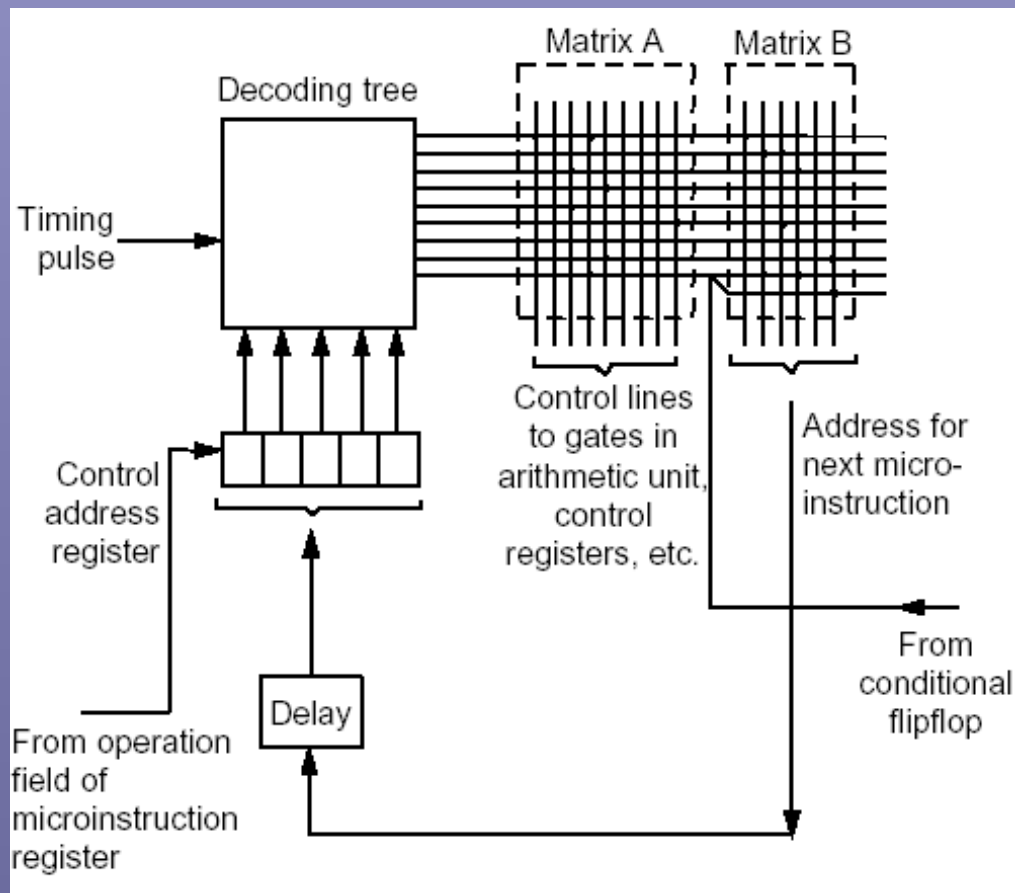
## Obvodová realizace konečného automatu

- přechody stavovým diagramem
- standardní metody sekvenční logiky

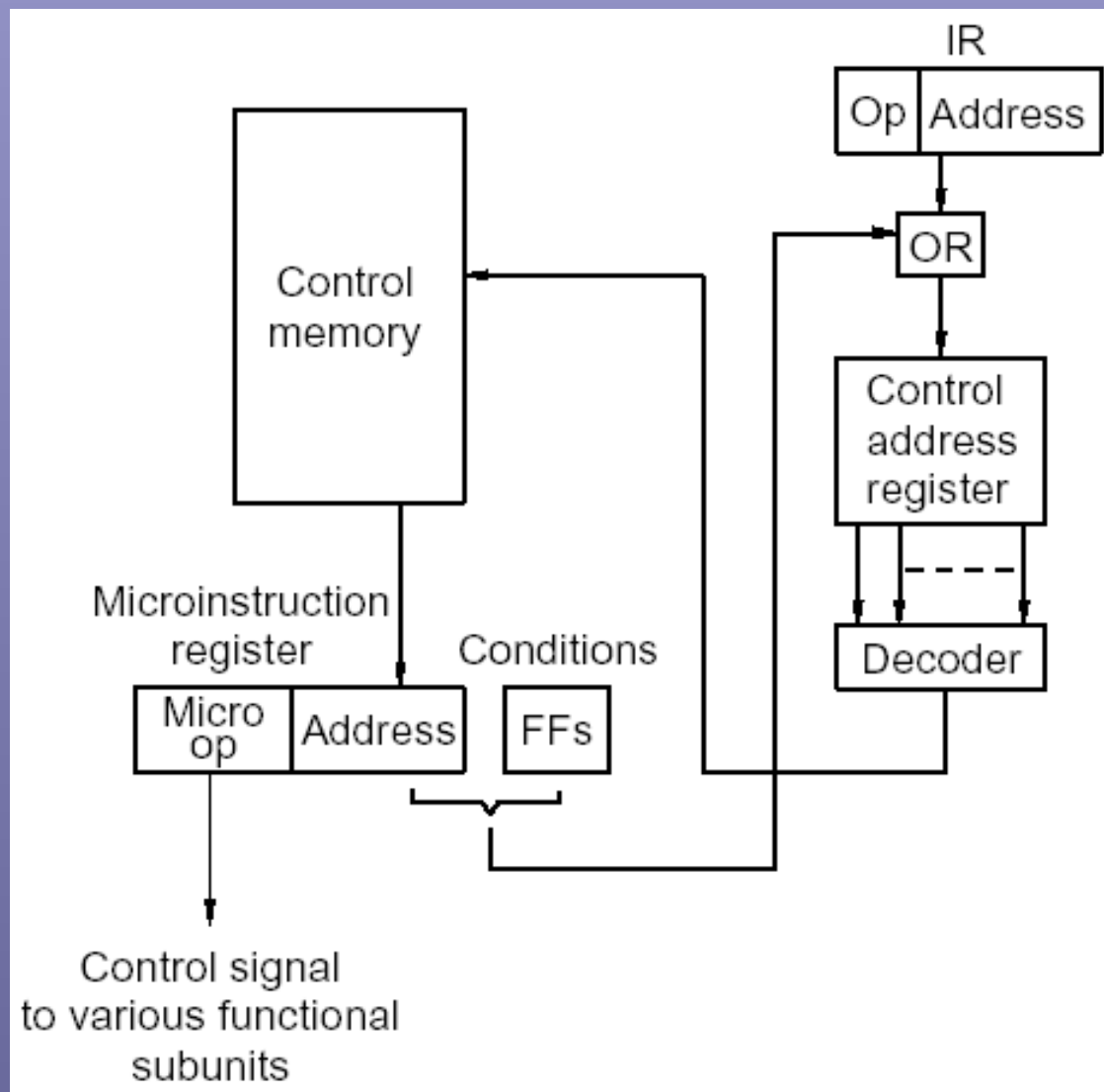


# Mikroprogramování

- Maurice V. Wilkes, 1951



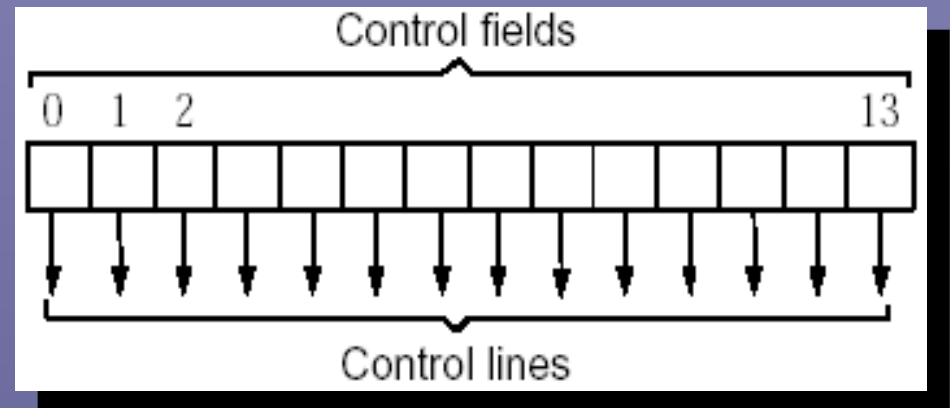
# Provádění mikrokódu



# Horizontální formát

## Přímá reprezentace

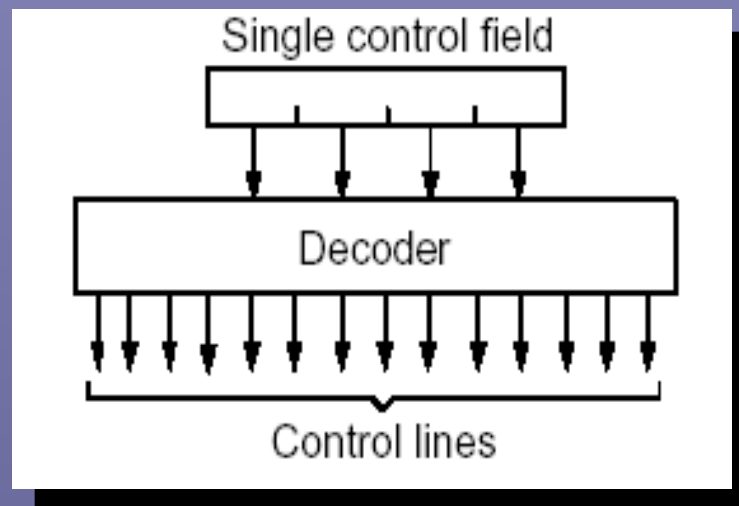
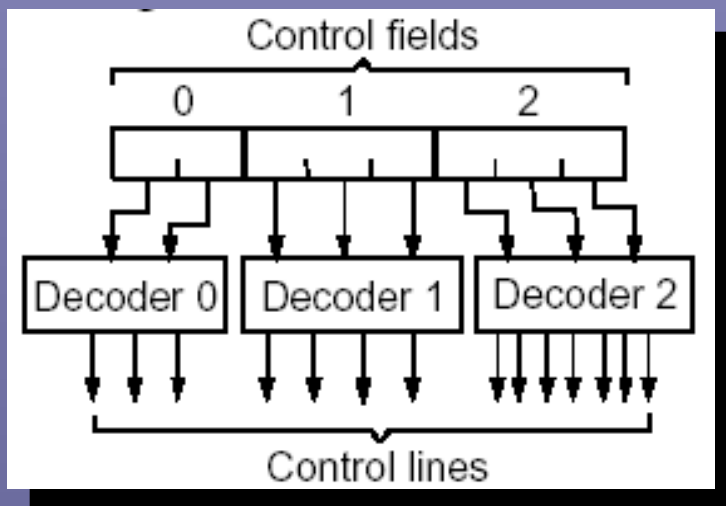
- mikroinstrukce obsahuje přímo hodnoty řídicích signálů
- není třeba dekódovat → rychlost
- libovolná kombinace → pružnost
- velké nároky na prostor



# Vertikální formát

## Kódovaná reprezentace

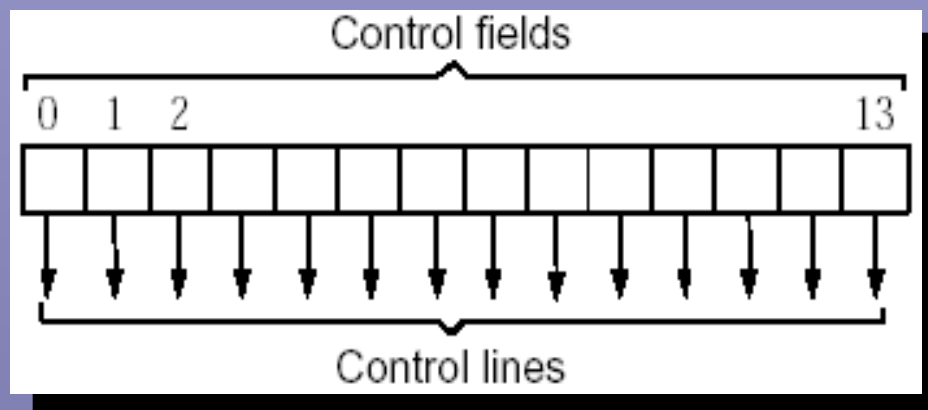
- některé kombinace se vylučují navzájem → možno zakódovat → menší objem
- nutno dekodovat → zpomalení, zesložnění
- pevný návrh → méně pružné



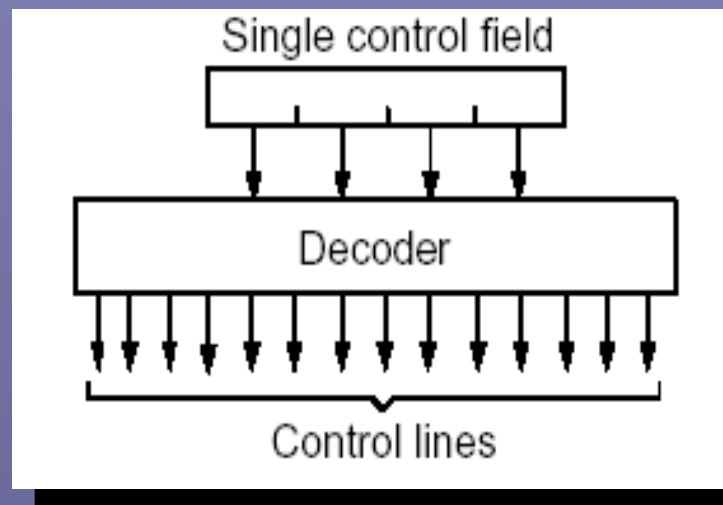
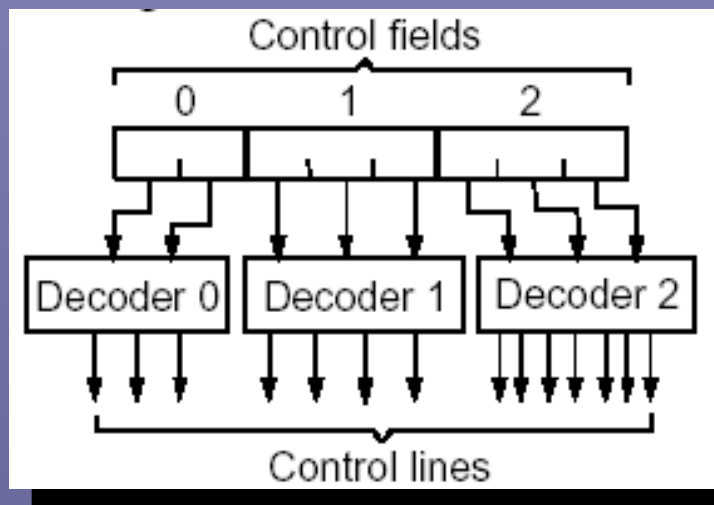


# Formát mikroinstrukce

## Horizontální



## Vertikální



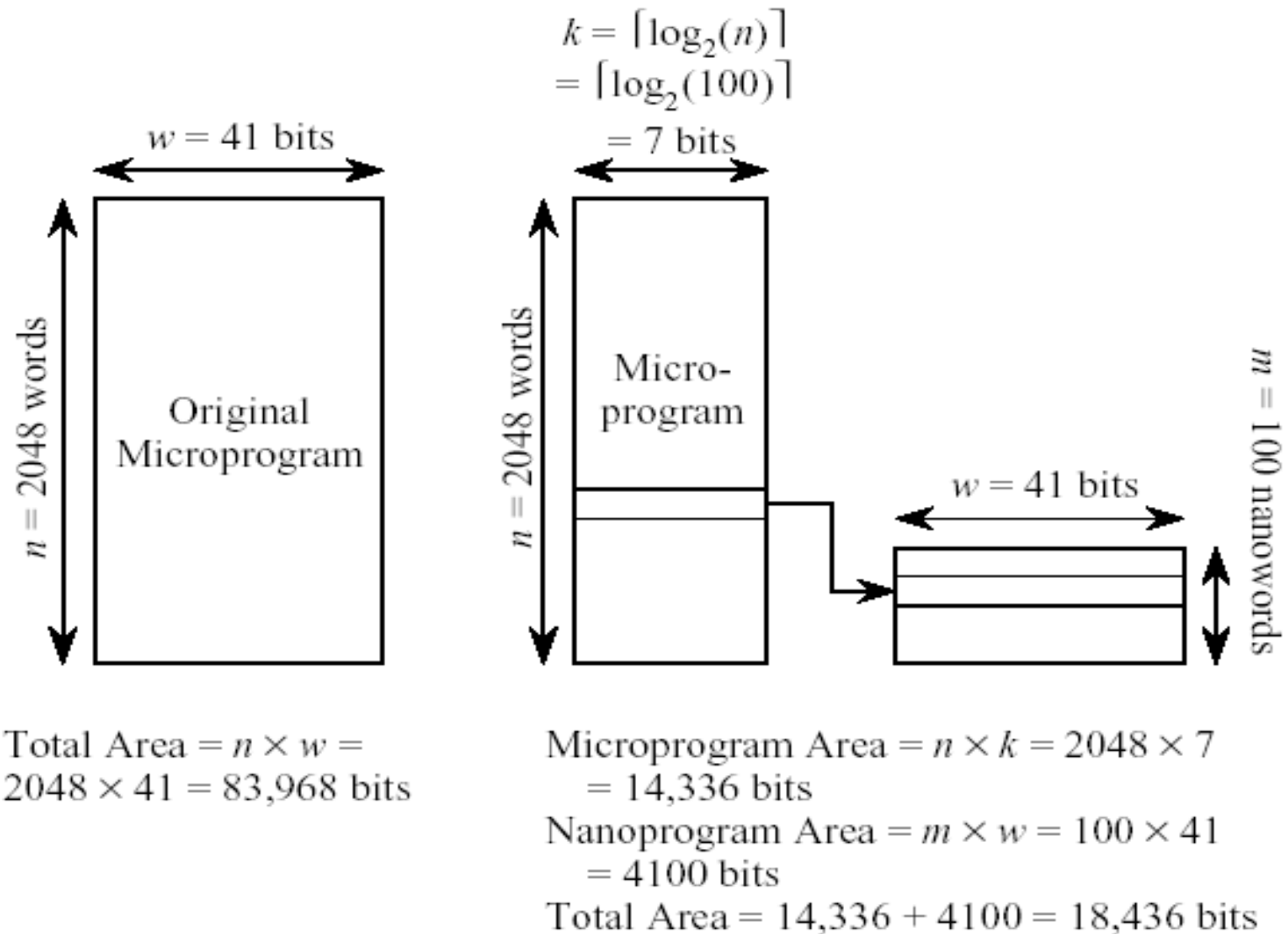
# Nanoprogramování

---

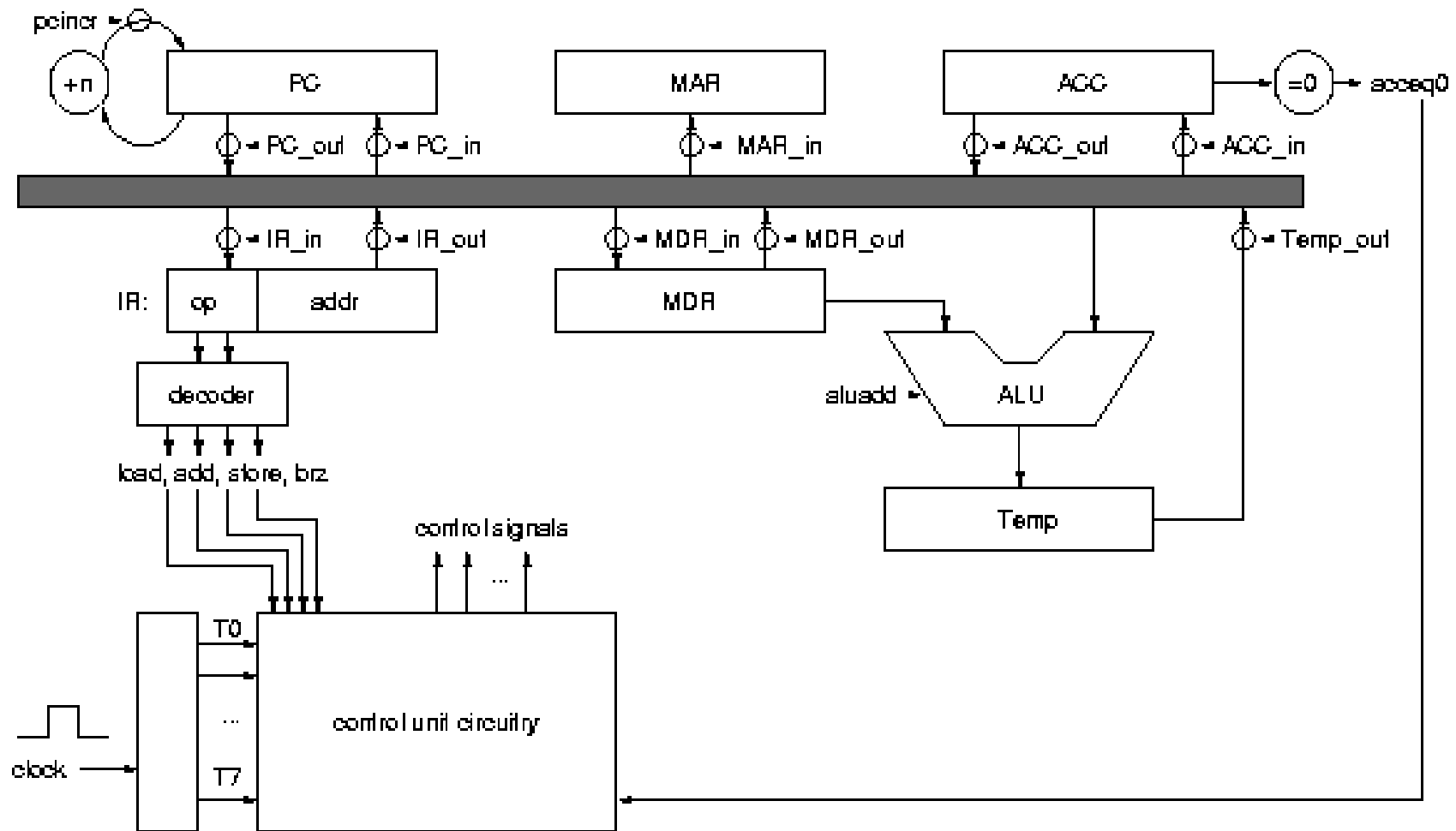
## Dvojúrovňová řídící paměť

- horní úroveň použije vertikální formát pro indexaci druhé úrovně
- nanoprogram ve druhé úrovni použije horizontální formát
- silná redukce velikosti
- zpomalení

# Srovnání mikro- a nanoprogramování



# Mikroprogramování na jednoduchém CPU



# Instrukční sada

(00) load address

ACC  $\leftarrow$  memory [address]

(01) add address

ACC  $\leftarrow$  ACC + memory [address]

(10) store address

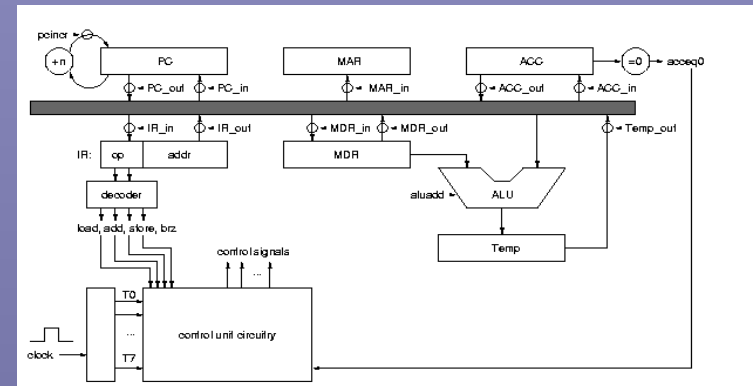
memory [address]  $\leftarrow$  ACC

(11) brz address

if (ACC == 0) PC  $\leftarrow$  address

# Řídící signály

ACC\_in : ACC  $\leftarrow$  CPU internal bus  
ACC\_out : CPU internal bus  $\leftarrow$  ACC  
aluadd : addition is selected as the ALU operation  
IR\_in : IR  $\leftarrow$  CPU internal bus  
IR\_out : CPU internal bus  $\leftarrow$  address portion of IR  
MAR\_in : MAR  $\leftarrow$  CPU internal bus  
MDR\_in : MDR  $\leftarrow$  CPU internal bus  
MDR\_out : CPU internal bus  $\leftarrow$  MDR  
PC\_in : PC  $\leftarrow$  CPU internal bus  
PC\_out : CPU internal bus  $\leftarrow$  PC  
pcincr : PC  $\leftarrow$  PC + 1  
read : MDR  $\leftarrow$  memory[ MAR ]  
TEMP\_out : CPU internal bus  $\leftarrow$  TEMP  
write : memory[ MAR ]  $\leftarrow$  MDR



# Mikroprogram pro instrukci load

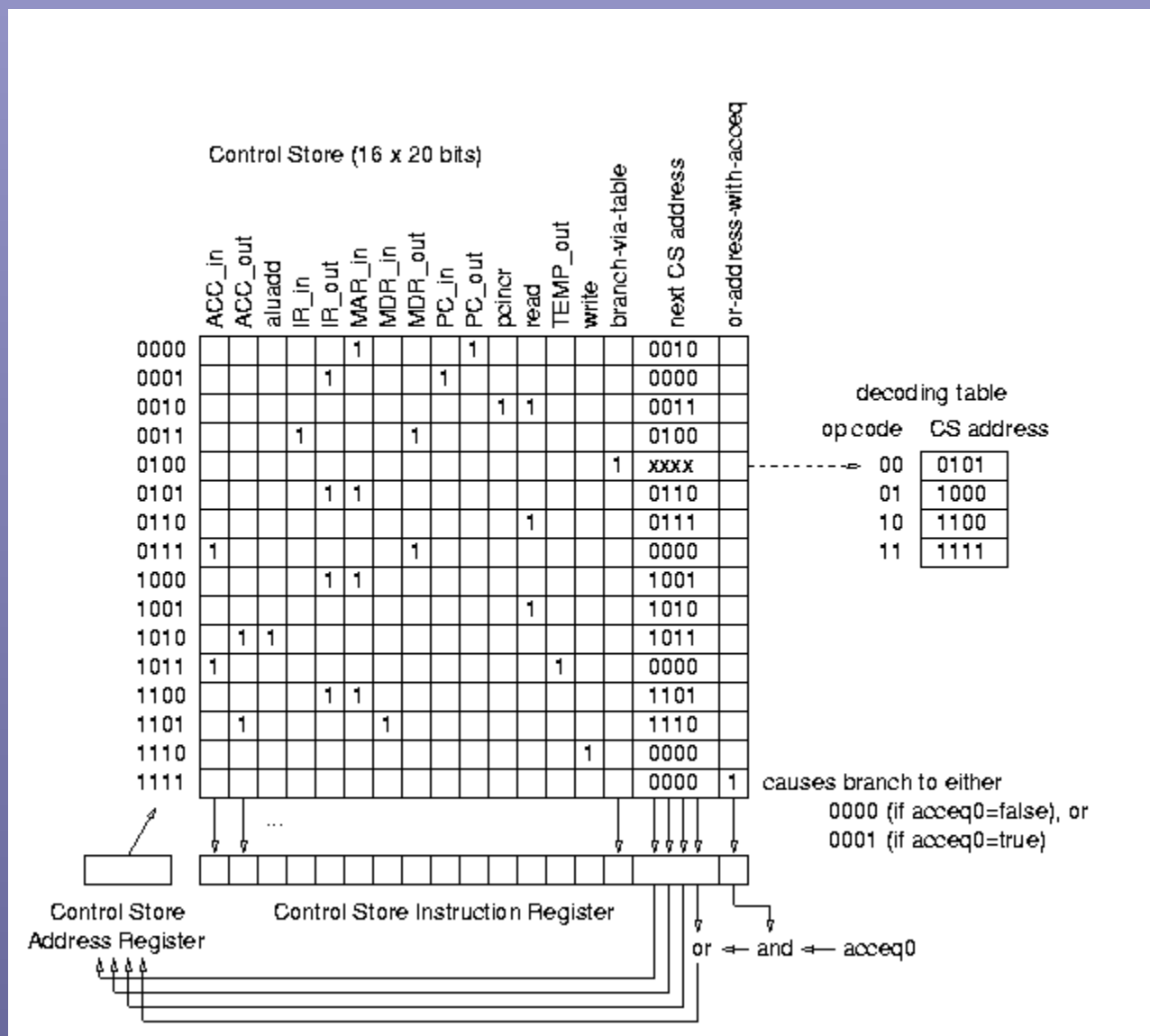
- fetch
  - T0: PC\_out, MAR\_in
  - T1: read, pcincr
  - T2: MDR\_out, IR\_in
- decode
  - T3: decode opcode in IR
- execute
  - T4: IR\_out(addr part), MAR\_in
  - T5: read
  - T6: MDR\_out, ACC\_in, reset to T0

# Logický popis řídicích signálů

```
ACC_in = (load & T6) + (add & T7)
ACC_out = (store & T5) + (add & T6)
aluadd = add & T6
IR_in = T2
IR_out = (load & T4) + (add & T4) + (store & T4)
        + (brz & acceq0 & T4)
MAR_in = T0 + (load & T4) + (add & T4) + (store & T4)
MDR_in = store & T5
MDR_out = T2 + (load & T6)
PC_in = brz & acceq0 & T4
PC_out = T0
pcincr = T1
read = T1 + (load & T5) + (add & T5)
TEMP_out = add & T7
write = store & T6
```



# Řídící paměť



# Literatura

---

K. Bigelow

- <http://www.play-hookey.com/digital/>

D. A. Patterson, J. L. Hennessy

- Computer Organization and Design