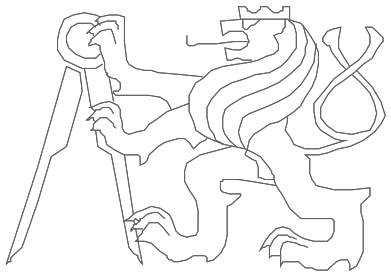
Architektura počítačů

04

Zřetězené vykonávaní instrukcí; Hazardy;
Vyvažování stupňů zřetězení a časování; Superzřetězení



České vysoké učení technické, Fakulta elektrotechnická

Úkol dnešní přednášky

- Navrhneme jednoduchý výpočetní systém pozůstávající z procesoru, oddělené instrukční a datové paměti. Procesor bude podporovat instrukce: `add, sub, and, or, slt, addi, lw, sw` a `beq` ve formátu instrukční sady MIPS. Součástí procesoru bude řídicí jednotka a aritmeticko-logická jednotka.
- Co dokáže takový procesor? Například:

```
if(a<=b)
    a = a-b;
else
    for(int i=0; i!=c; i++)
        a += i;
a = a | 3;
while(1)
    ;
```

ISA a mikroarchitektura

- Uvažujme tři typy instrukcí dle tabulky:

Typ	31…					0
R	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**rd**(5), 15:11	**shamt**(5)	**funct**(6), 5:0
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0		
J	**opcode**(6), 31:26	**address**(26), 25:0				

- všechny R instrukce -> opcode=000000, funct – operace
- rs – source, rd – destination, rt – source/destination
- shamt – při operacích posunu, immediate – přímý operand

- K dispozici je 32 pracovních registrů

ISA a mikroarchitektura

Pracovní registry:

Name	Number	Use
$zero	$0	constant 0
$at	$1	assembler temporary
$v0–$v1	$2–$3	values for function returns and expression evaluation
$a0–$a3	$4–$7	function arguments
$t0–$t7	$8–$15	temporaries
$s0–$s7	$16–$23	saved temporaries
$t8–$t9	$24–$25	temporaries
$k0–$k1	$26–$27	reserved for OS kernel
$gp	$28	global pointer
$sp	$29	stack pointer
$fp	$30	frame pointer
$ra	$31	return address

ADD – *Add (with overflow)*

Description	Adds two registers and stores the result in a register
Operation:	$d = $s + $t;
Syntax:	add $d, $s, $t
Encoding:	`0000 00ss ssst tttt dddd d000 0010 0000`

Realizujme součet dvou registrů

add $17, $0, $10

```
0000 00ss ssst tttt dddd d000 0010 0000
0000 0000 0000 1010 1000 1000 0010 0000
          0    10   17
```

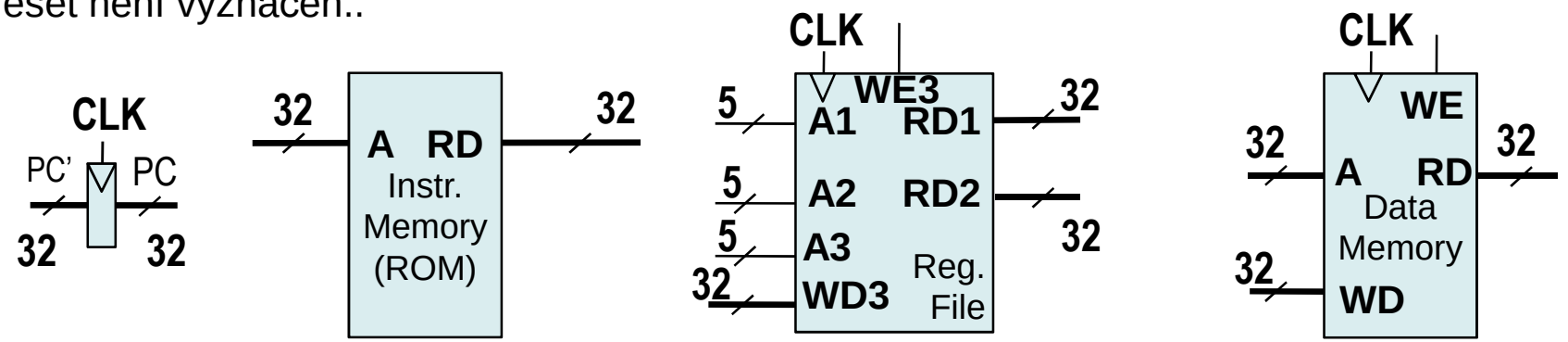
0x 00 0A 88 20 – strojový kód instrukce add $17, $0, $10

Jedno-cyklový procesor – návrh

- IPC rovno 1
- podpora: add, sub, and, or, slt, addi, lw, sw, beq
- dvě části: datapath, control

aritmetické/logické instrukce; paměťové; větvení; instrukce typu I

reset není vyznačen..



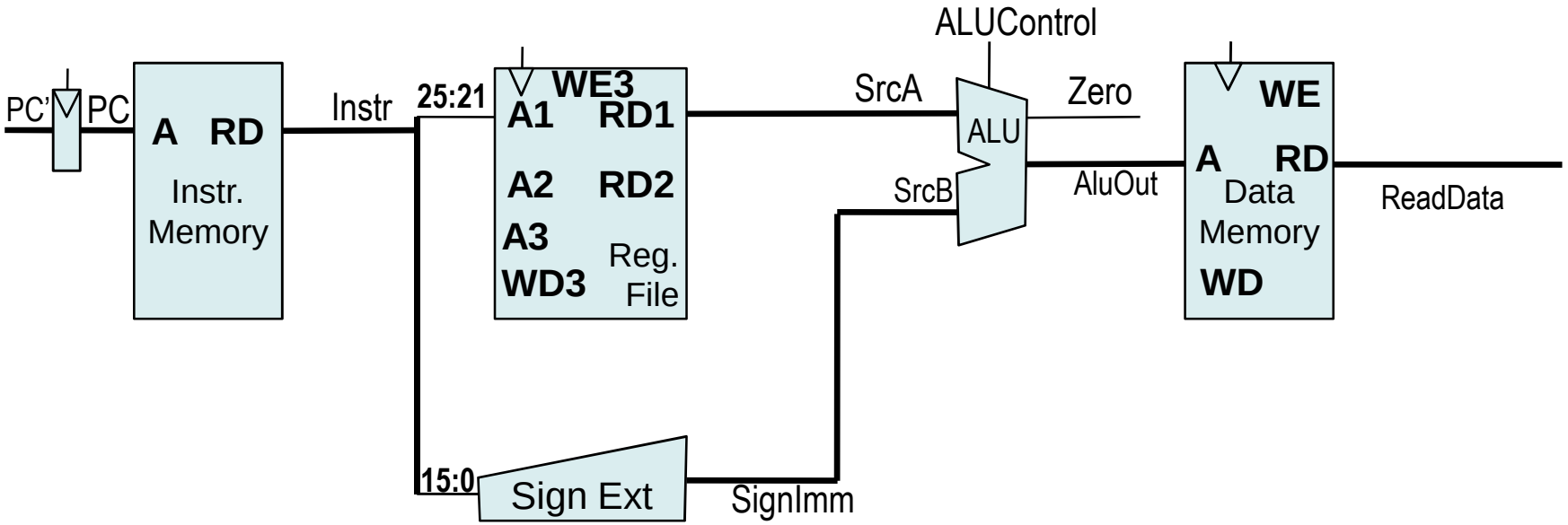
zápis s náběžnou hranou CLK při WE = 1

čtení po uplynutí dostatečně dlouhé doby

Jedno-cyklový procesor – návrh – podpora lw

- **lw**: typ I, rs – bázová adresa, imm – offset, rt – kde uložit

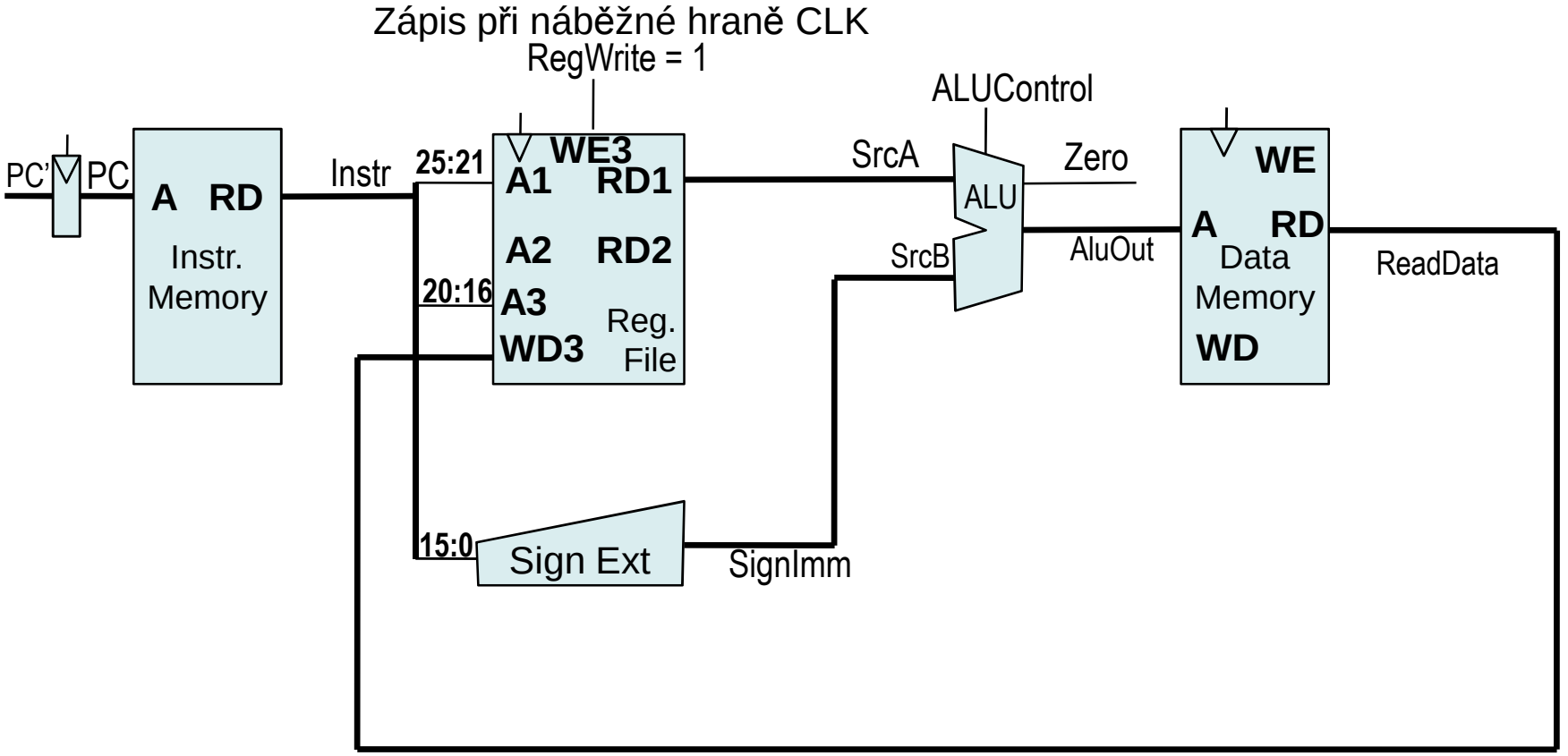
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



Jedno-cyklový procesor – návrh – podpora lw

- **lw**: typ I, rs – bázová adresa, imm – offset, rt – kde uložit

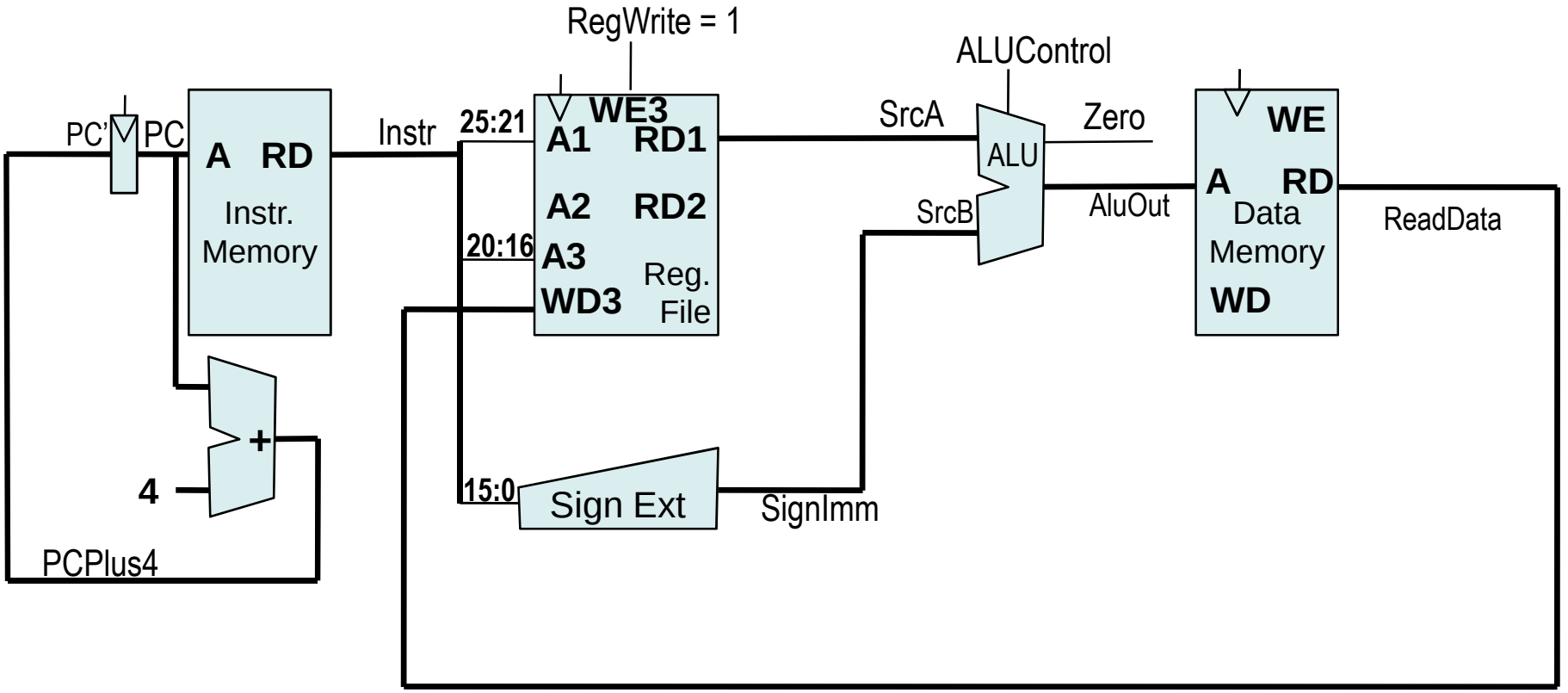
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



Jedno-cyklový procesor – návrh – podpora lw

- **lw**: typ I, rs – bázová adresa, imm – offset, rt – kde uložit

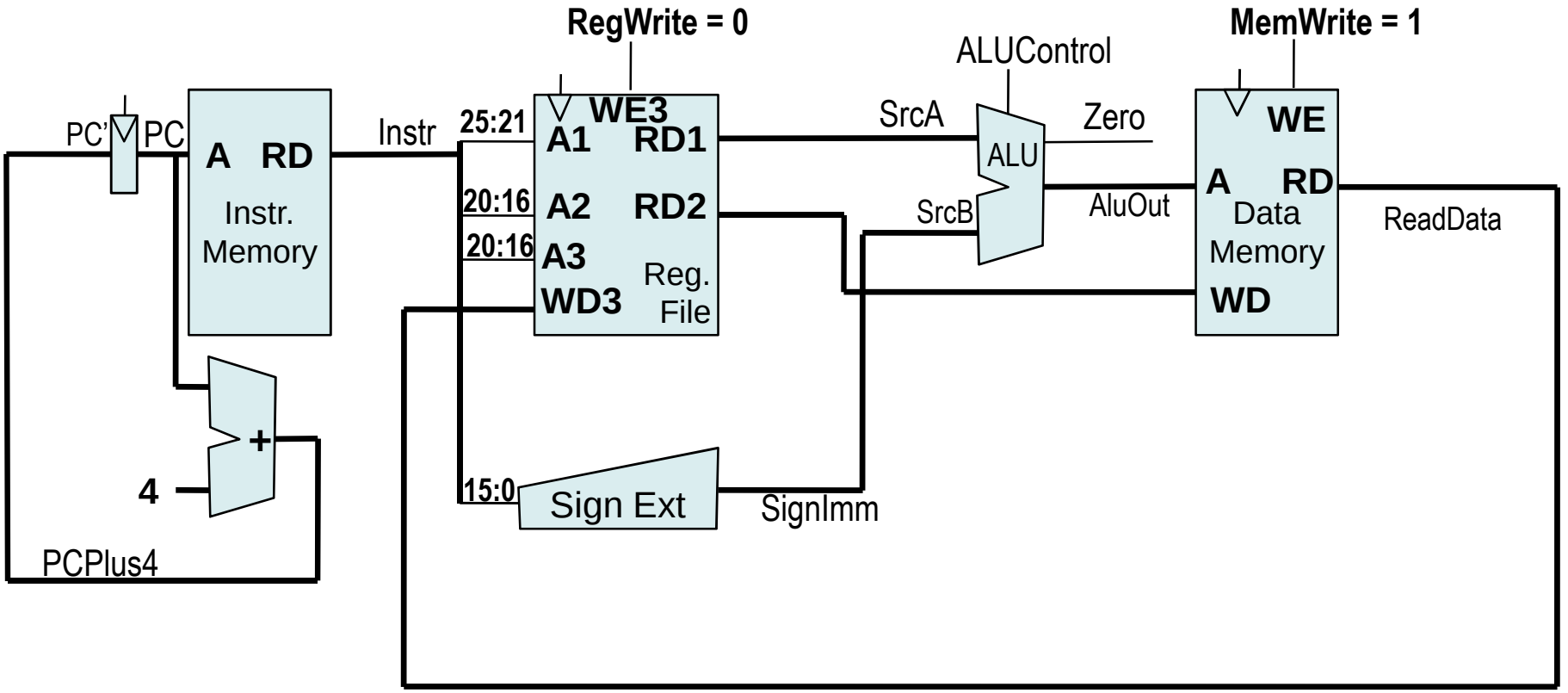
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



Jedno-cyklový procesor – návrh – podpora sw

- **sw**: typ I, rs – bázová adresa, imm – offset, **rt – co zapsát**

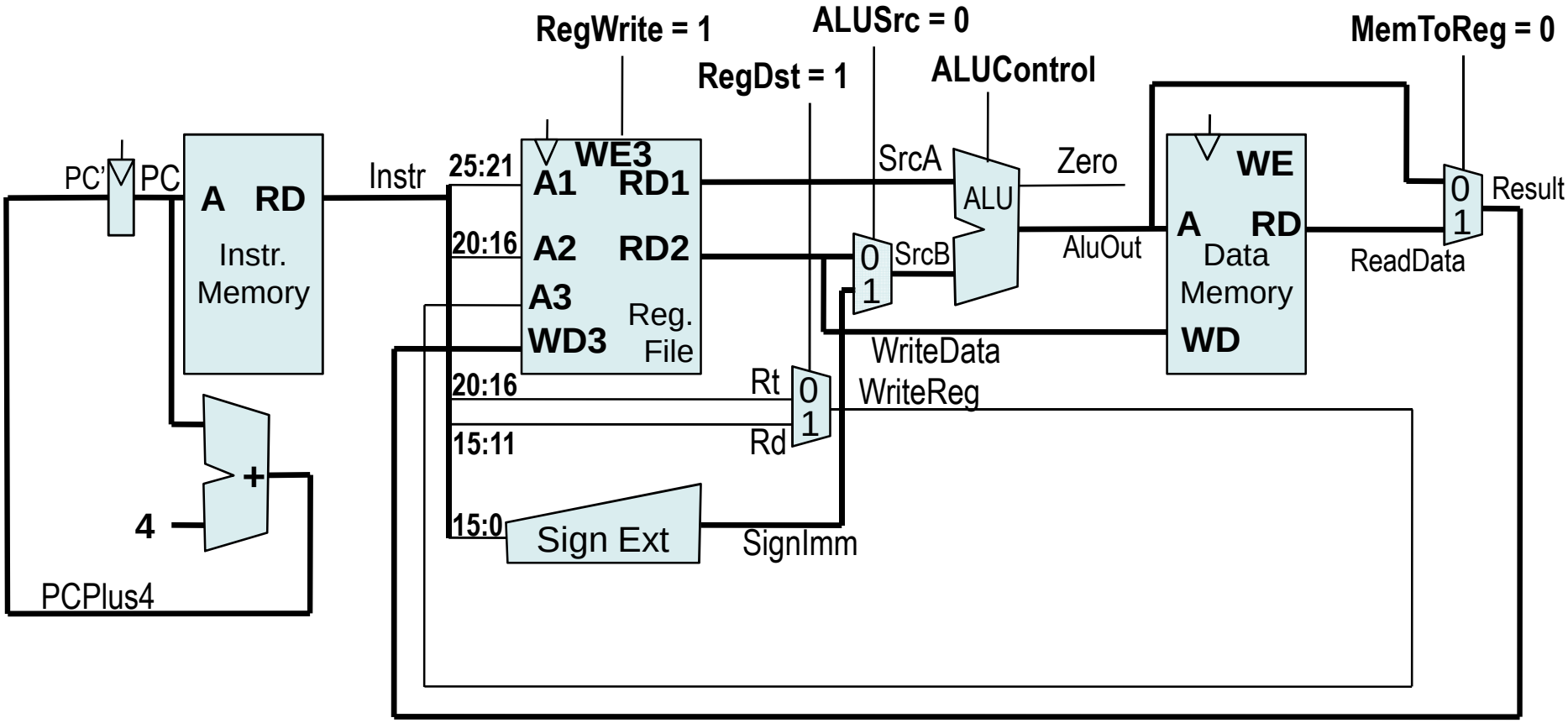
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



Jedno-cyklový procesor – návrh – podpora add

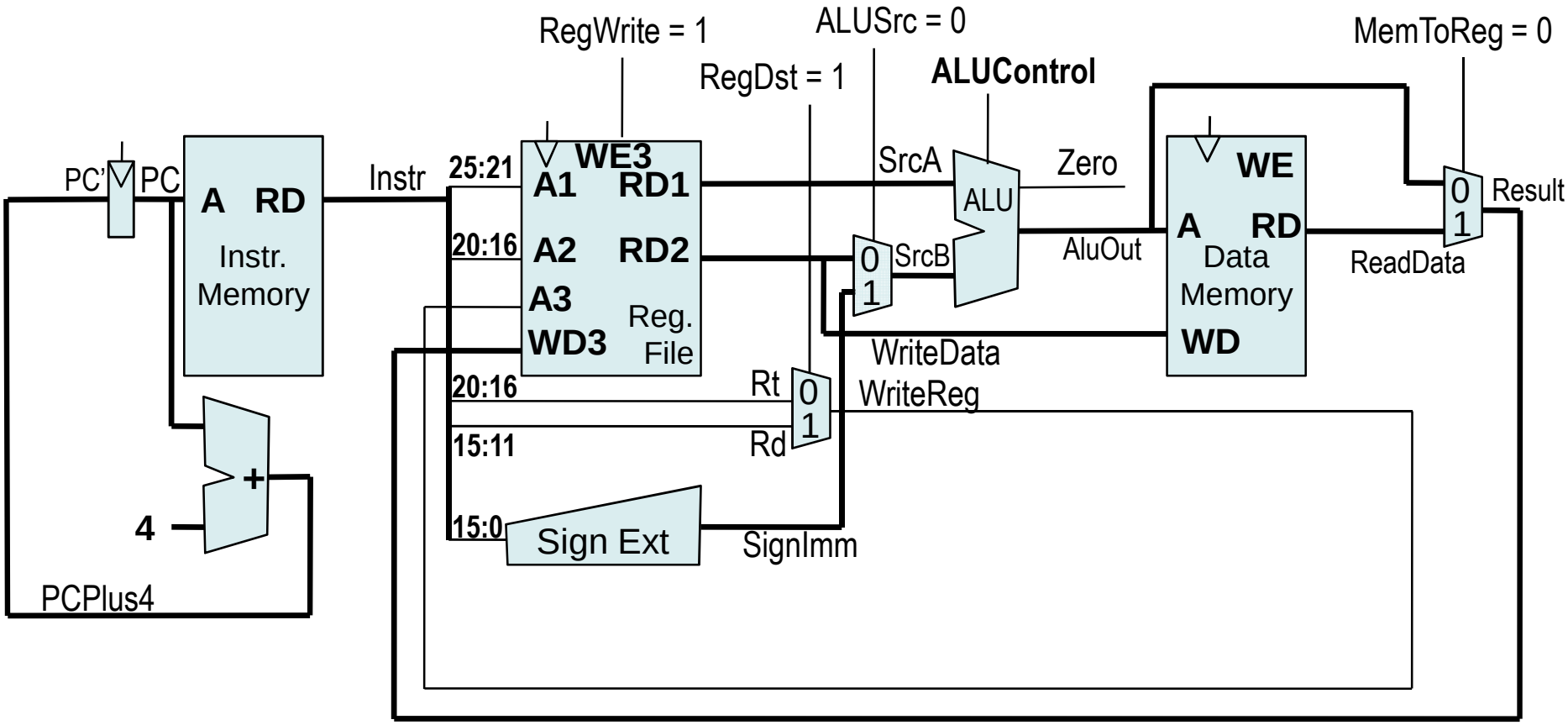
- add: typ R; rs, rt – zdroje, rd – cíl, funct – operace součtu

R	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**rd**(5), 15:11	**shamt**(5)	**funct**(6), 5:0



Jedno-cyklový procesor – návrh – podpora sub, and, or, slt

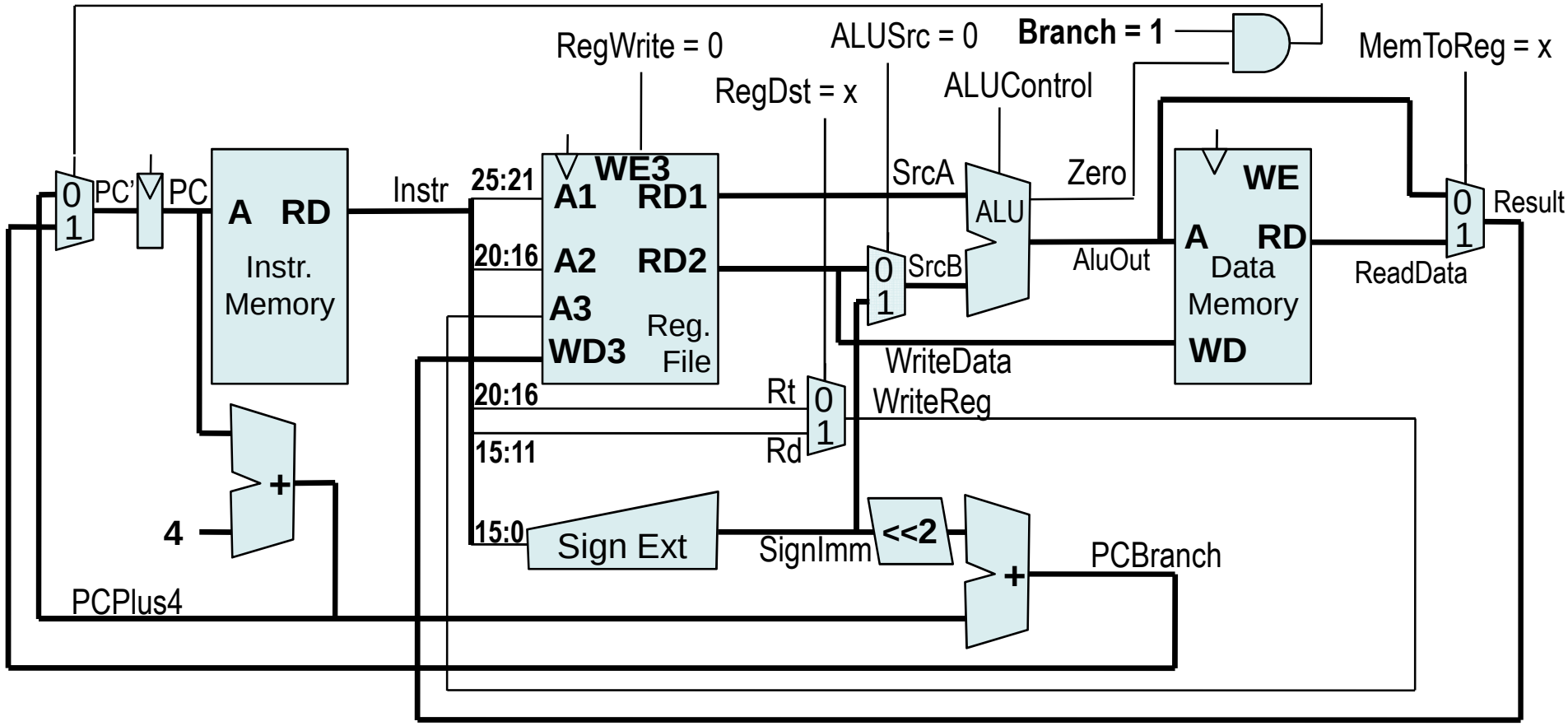
- jediné v čem se liší od add je operace ALU -> datapath beze změny; rozdíl v ALUControl



Jedno-cyklový procesor – návrh – podpora beq

- beq – branch if equal; imm – offset; PC´ = PC+4 + SignImm*4

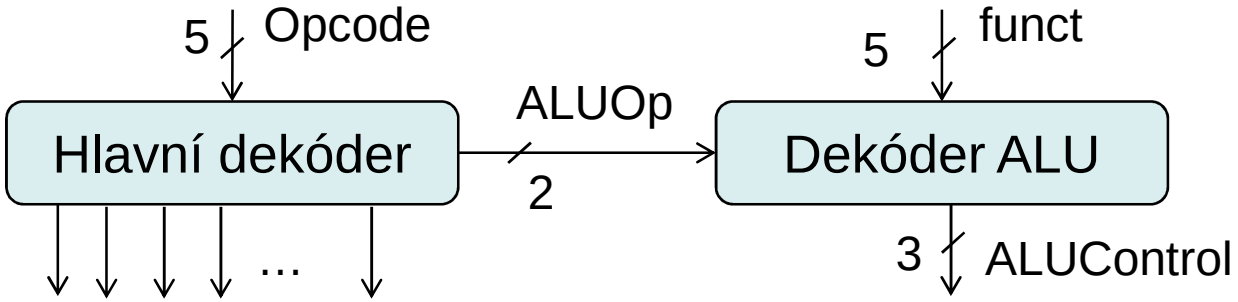
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



Jedno-cyklový procesor – návrh – řídicí část

R	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**rd**(5), 15:11	**shamt**(5)	**funct**(6), 5:0
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0		
J	**opcode**(6), 31:26	**address**(26), 25:0				

- řídicí signály na základe **opcode** a **funct**



ALUOp	
00	součet
01	rozdíl
10	podle funct
11	-nepoužito-

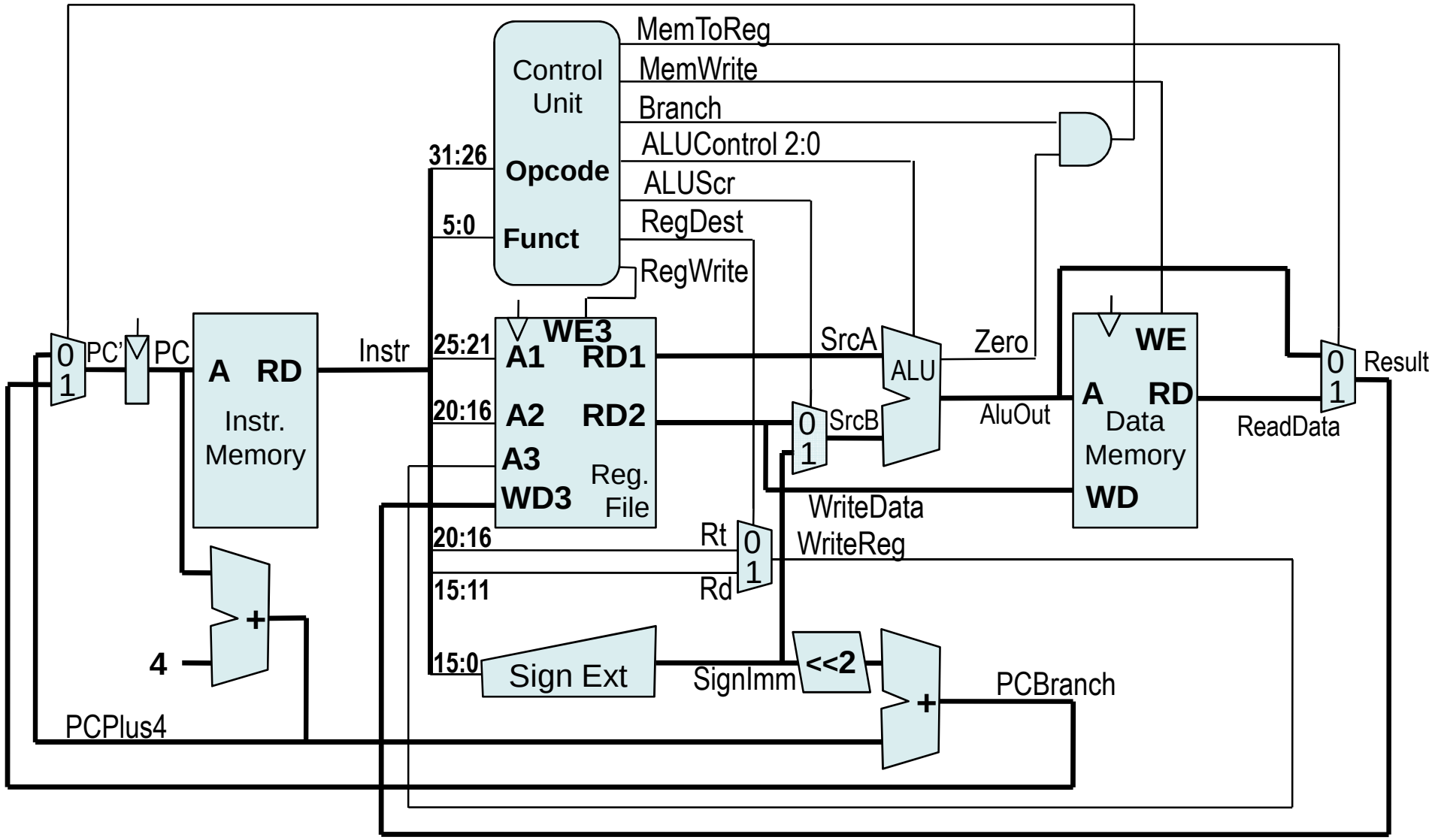
	Opcode	RegWrite	RegDst	ALUSrc	ALUOp	Branch	Mem Write	MemTo Reg
R typ	000000	1	1	0	10	0	0	0
lw	100011	1	0	1	00	0	0	1
sw	101011	0	X	1	00	0	1	X
beq	000100	0	X	0	01	1	0	X

Jedno-cyklový procesor – návrh – řídicí část

ALUOp	Funct	ALUControl
00	X	010 (add)
01	X	110 (sub)
1X	add (100000)	010 (add)
1X	sub (100010)	110 (sub)
1X	and (100100)	000 (and)
1X	or (100101)	001 (or)
1X	slt (101010)	111 (set less than)

- řídicí jednotka (hlavní dekóder instrukcí + dekóder ALU) je kombinační obvod -> návrh obvodu je triviální..

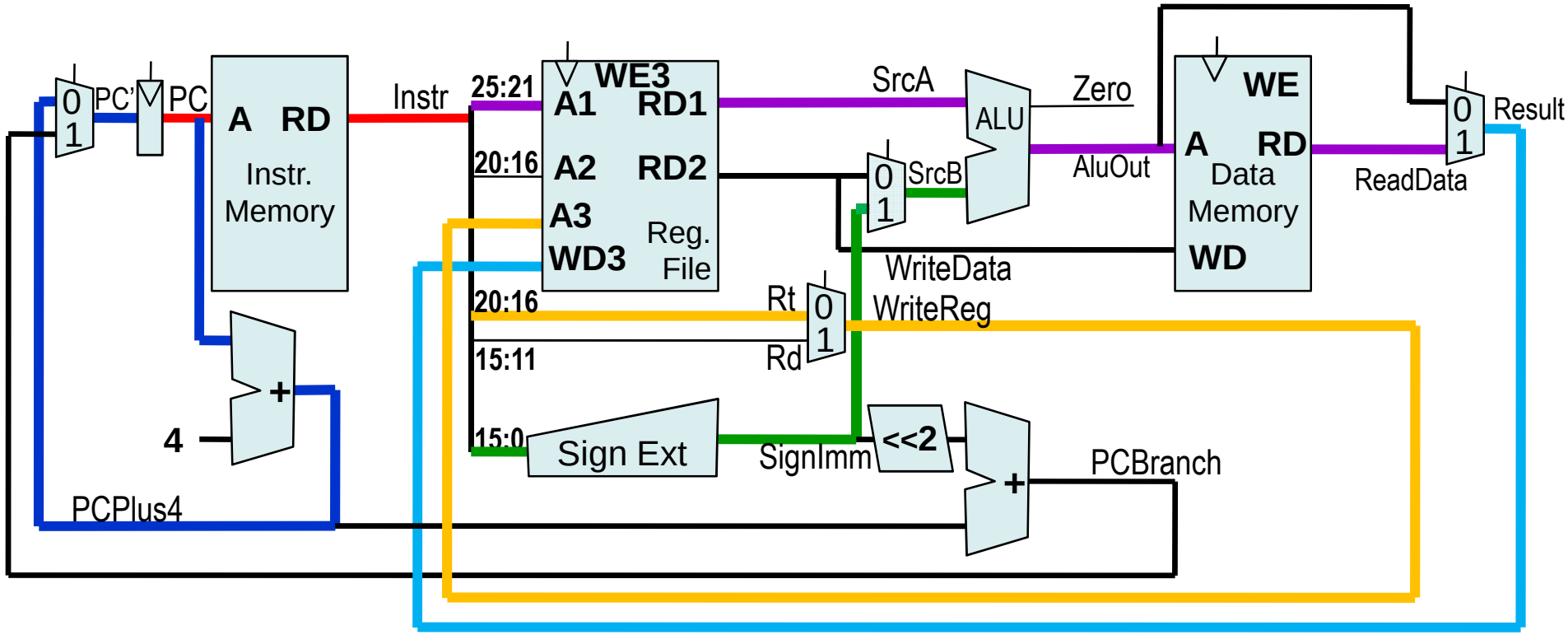
Jedno-cyklový procesor



Jedno-cyklový procesor – výkon: IPS = IC / T = $IPC_{str}.f_{CLK}$

- Jaká může být maximální frekvence procesoru?
- Zpoždění na kritické cestě – instrukce lw:

$$Tc = t_{PC} + t_{Mem} + t_{RFread} + t_{ALU} + t_{Mem} + t_{Mux} + t_{RFsetup}$$



Jedno-cyklový procesor – výkon: IPS = IC / T = $IPC_{str}.f_{CLK}$

- $Tc = t_{PC} + t_{Mem} + t_{RFread} + t_{ALU} + t_{Mem} + t_{Mux} + t_{RFsetup}$
- Předpokládejme:

  t_{PC} = 30 ns
  t_{Mem} = 300 ns
  t_{RFread} = 150 ns
  t_{ALU} = 200 ns
  t_{Mux} = 20 ns
  $t_{RFsetup}$ = 20 ns

  Pak Tc = 1020 ns --> $f_{CLK\ max}$ = 980 kHz,

  IPS = 1. 980e3 = 980 000 instrukcí za sekundu

Zřetězené vykonávaní instrukcí

Předpokládejme, že vykonání instrukce můžeme rozdělit do 5 stupňů:

IF – Instruction Fetch, ID – Instruction decode (and Operand Fetch),
EX – Execute, MEM – Memory Access, WB – Write Back

a dále $\tau = \max \{ \tau_i \}_{i=1}^{k}$, kde τ_i je čas šíření (*propagation delay) v i*-tém stupni.

IF – poslání PC do paměti a vybrání aktuální instrukce. Aktualizace PC = PC+4

ID – dekódování instrukce a načtení registrů specifikovaných v instrukci, provedení testu na rovnost registrů (kvůli možnému větvení), znaménkové rozšíření offsetu, výpočet cílové adresy pro případ větvení (zn. rozš. offset + PC)

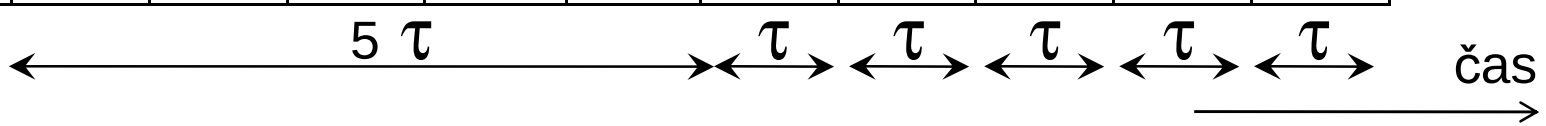
EX – operace ALU

MEM – v případě instrukce *load* /store – čtení/zápis do paměti

WB – v případě instrukcí typu register-register nebo instrukce *load* – zápis výsledku do RF (výsledek může přicházet z ALU nebo paměti)

Paralelizmus na úrovni instrukcí - zřetězení

IF	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
ID		I1	I2	I3	I4	I5	I6	I7	I8	I9
EX			I1	I2	I3	I4	I5	I6	I7	I8
MEM				I1	I2	I3	I4	I5	I6	I7
ST					I1	I2	I3	I4	I5	I6
	1	2	3	4	5	6	7	8	9	10



- Čas vykonání *n* instrukcí *k*-stupňové pipeline:

$$T_k = k.\tau + (n-1)\,\tau$$

Předpoklad: ideálně vyvážená pipeline

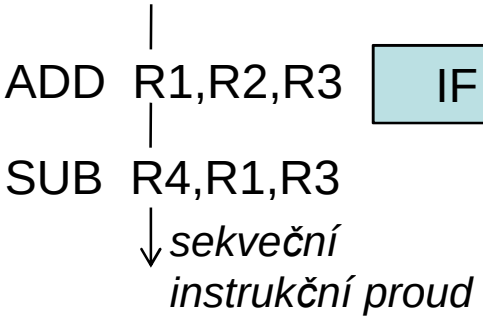
- Zrychlení: $S_k = \frac{T_1}{T_k} = \frac{nk\tau}{k\tau + (n-1)\tau}$ $\lim_{n\to\infty} S_k = k$

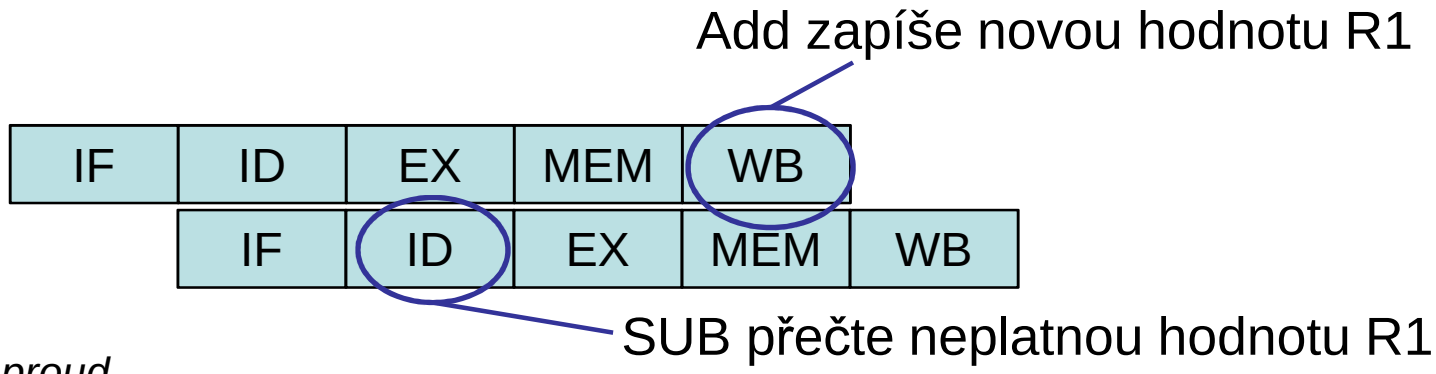
Paralelizmus na úrovni instrukcí - zřetězení

- Neredukuje čas vykonání individuální instrukce, právě naopak..
- Hazardy:
  Strukturální (řešeny duplikací),
  datové (důsledek datových závislostí) a
  řídící (instrukce měnící PC)...
- Hazardy způsobují (mohou způsobovat) pozastavení vykonávání (stall) nebo vyprázdnění pipeline
- Pozn. : Hlubší pipeline (více stupňů) znamená méně hradel v každém stupni a tím pádem možnost zvýšit pracovní frekvenci procesoru.., avšak více stupňů znamená i vyšší režii (nutnost lépe řadit instrukce do pipeline)

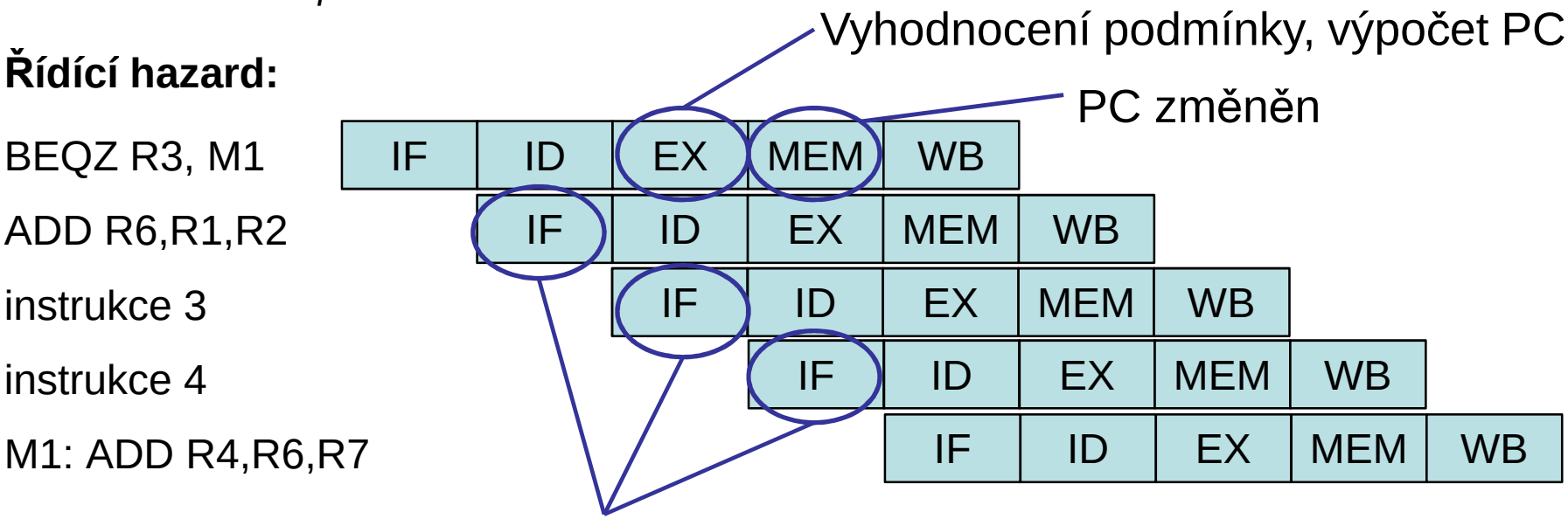
Paralelizmus na úrovni instrukcí - Narušení sémantiky

Datový hazard:



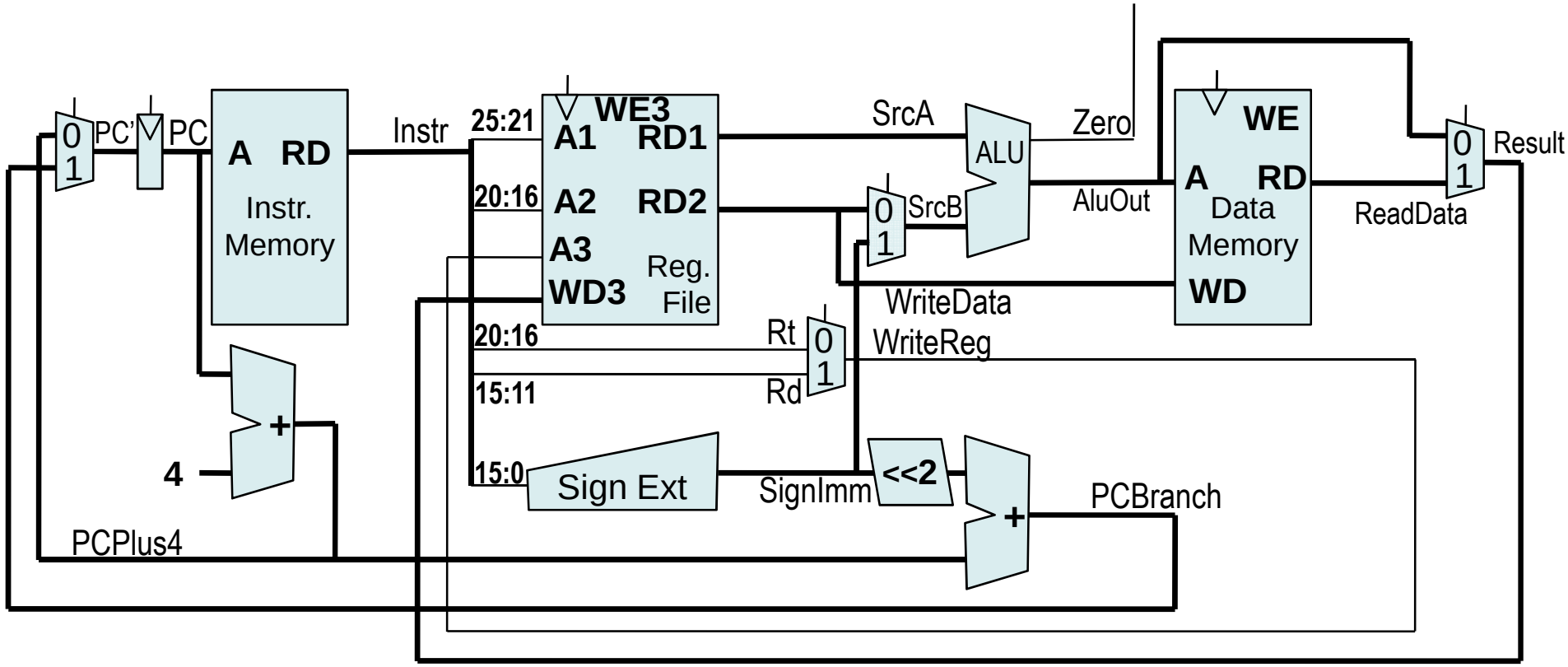


Řídící hazard:

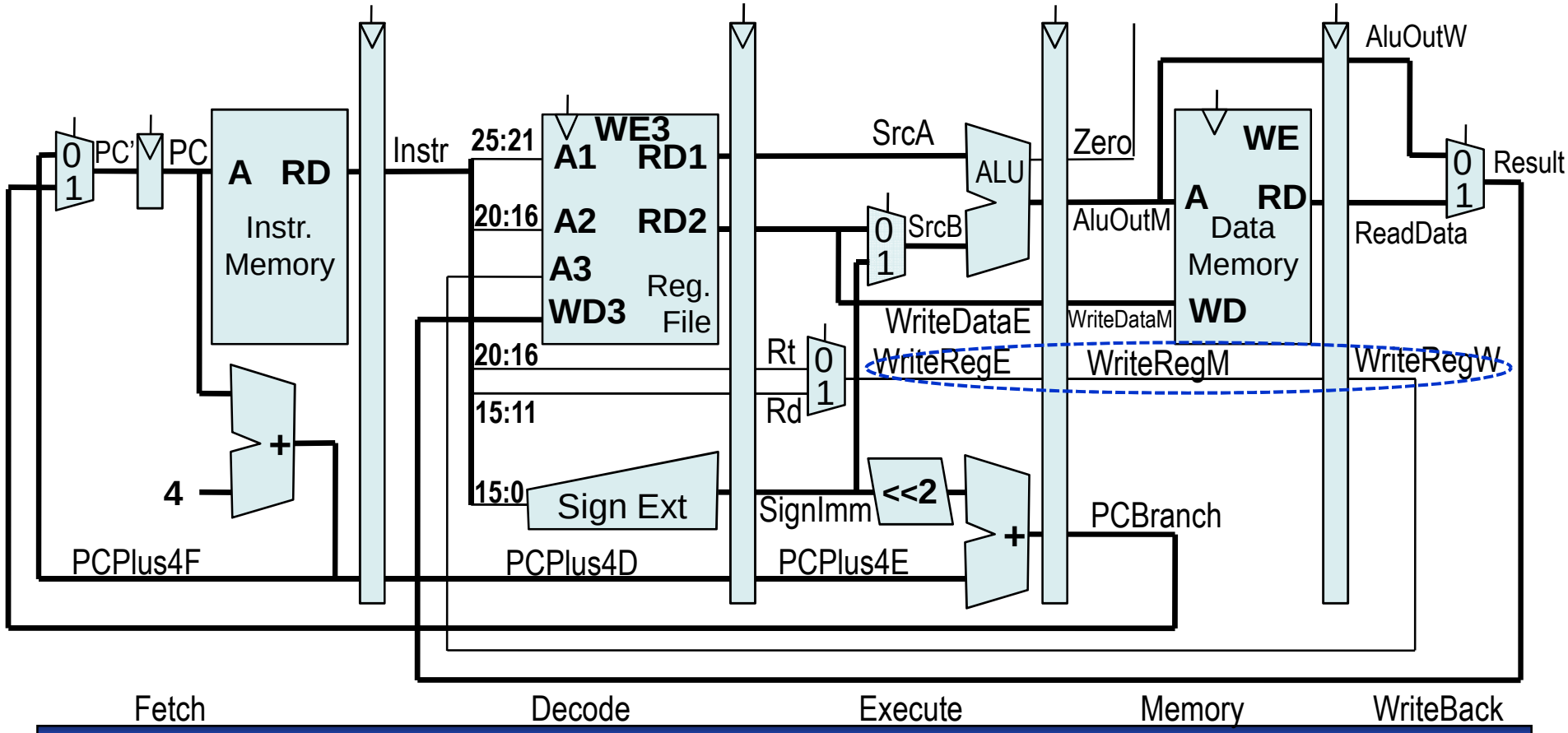


Měli být přineseny (a následně spracovávany) tyto instrukce?

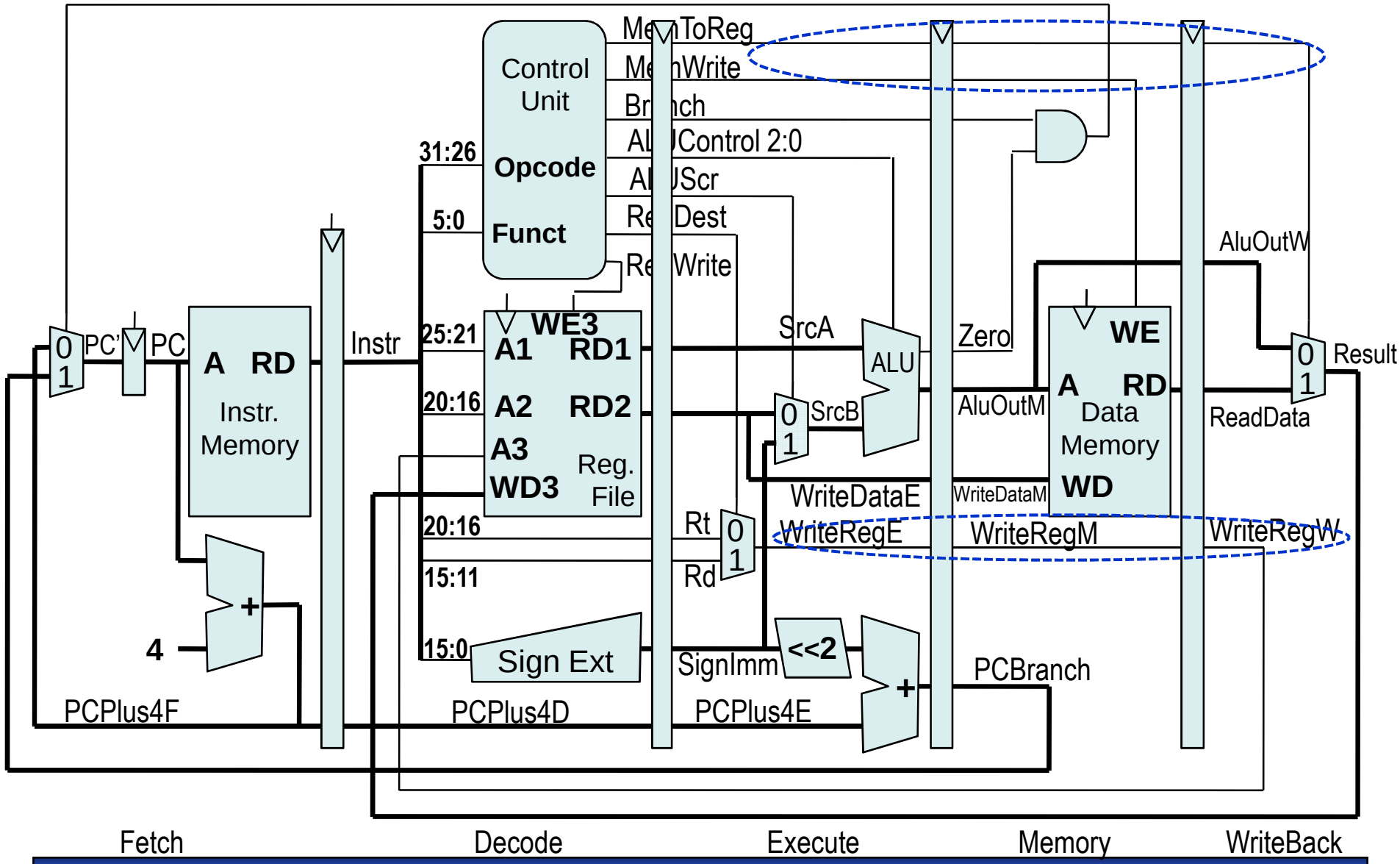
Nezřetězené vykonávání



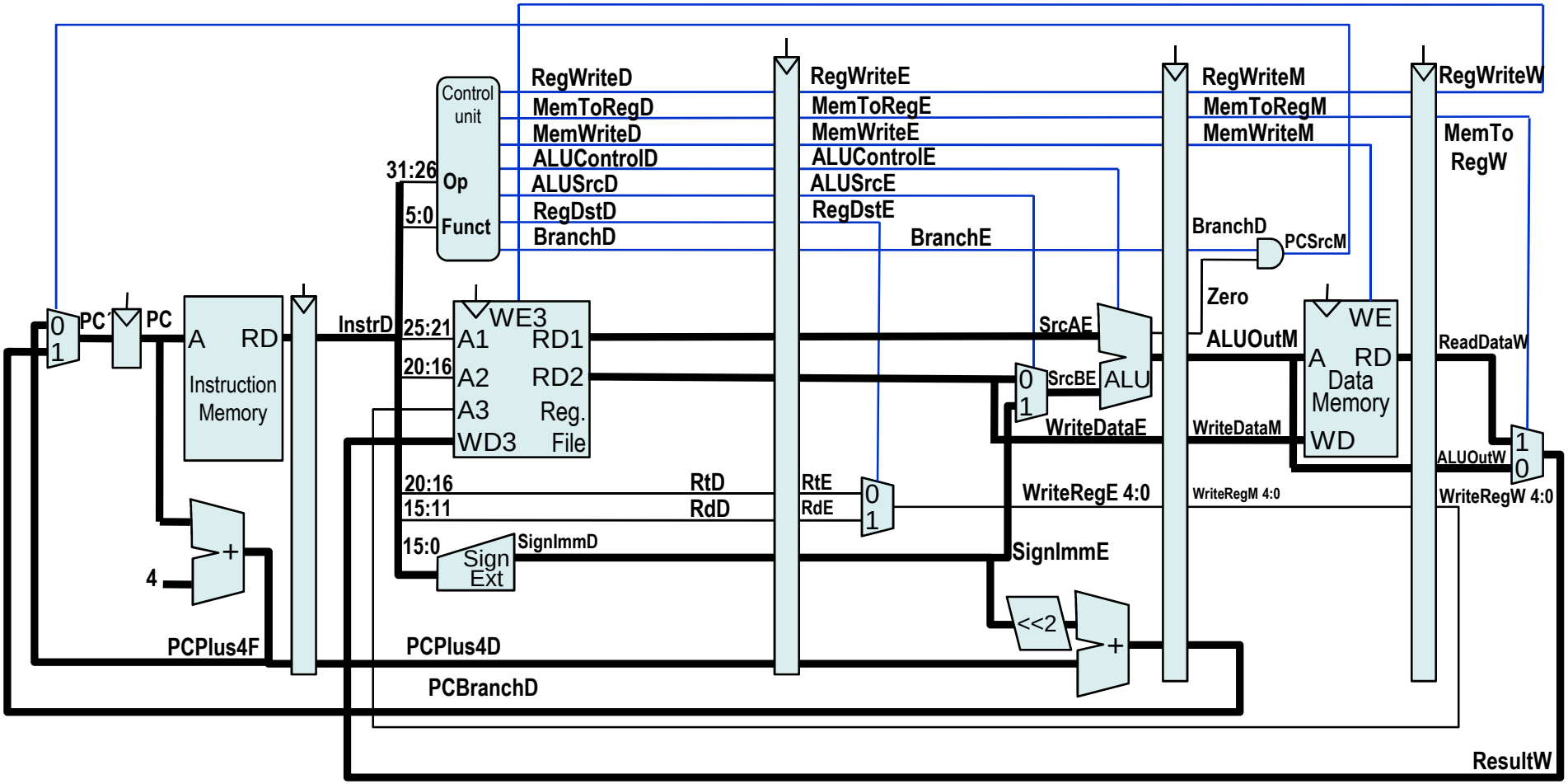
Zřetězené vykonávání



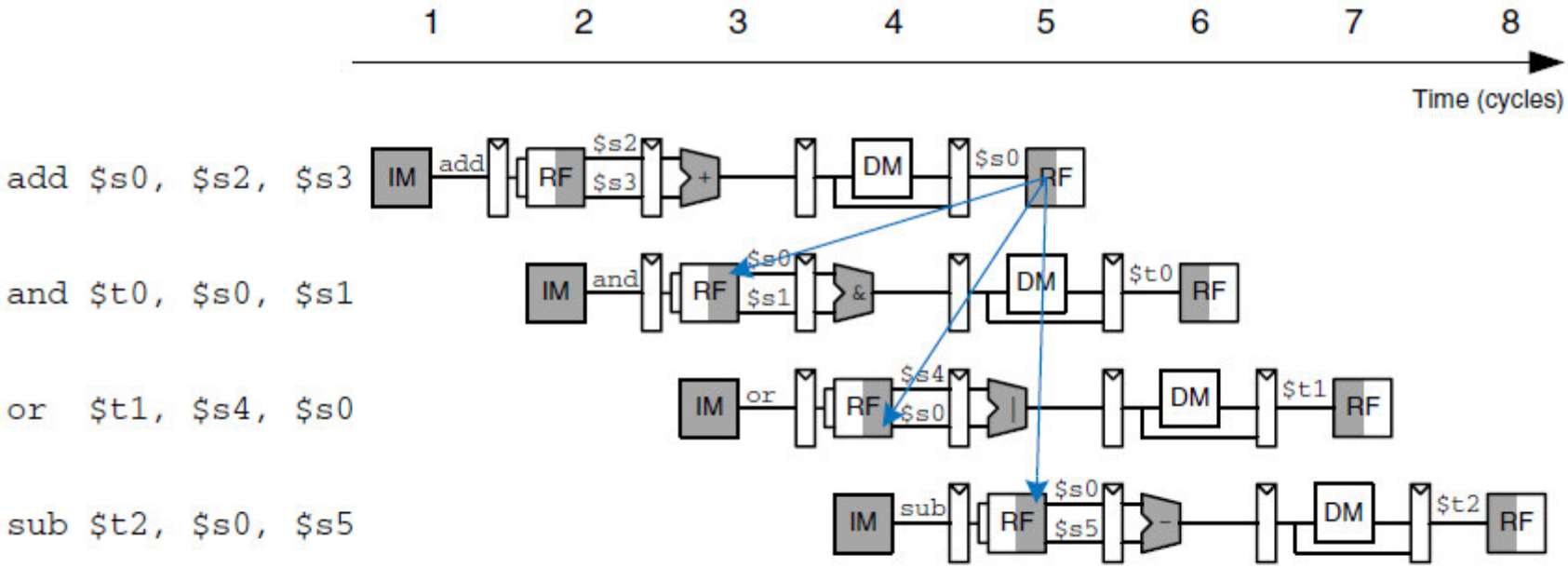
Zřetězené vykonávání



Totéž, pouze zmenšeno a překresleno…

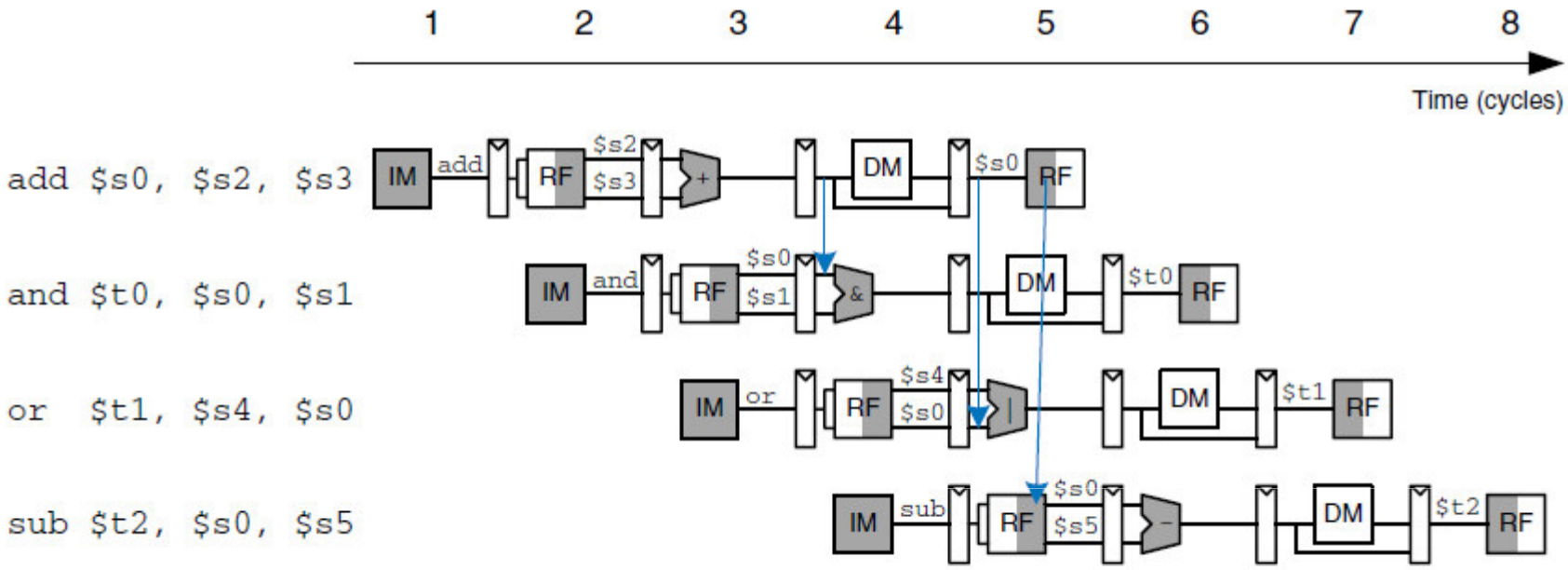


Vznik datových hazardů



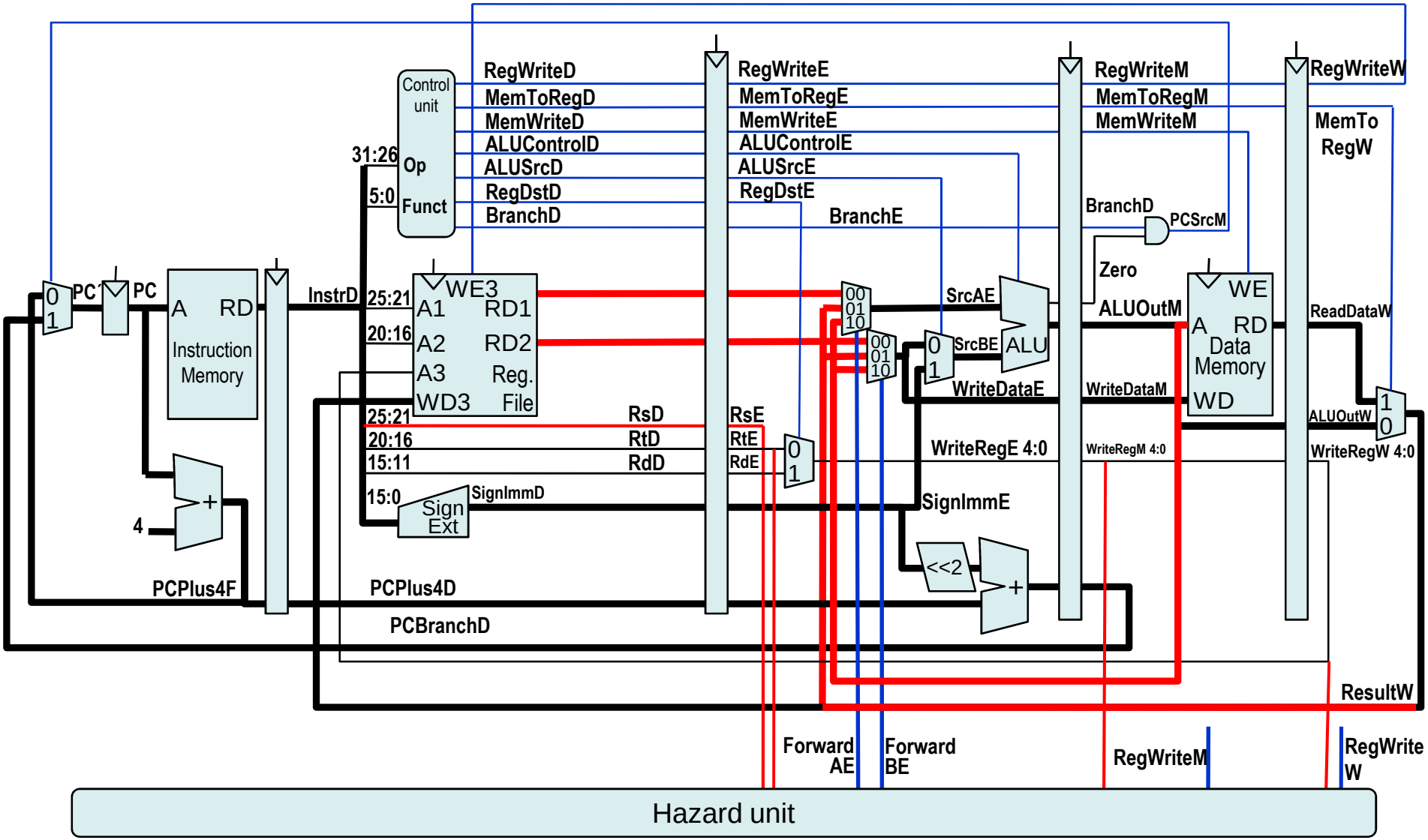
- Pracovní registry (Register File) – přístup v dvou fázích (Decode, WriteBack) – zápis v první polovině cyklu, čtení ve druhé..
- RAW hazard…
- Jak je možné řešit tento hazard a nedegradovat výkon pipeline?

Řešení datových hazardů přeposíláním (forwarding)

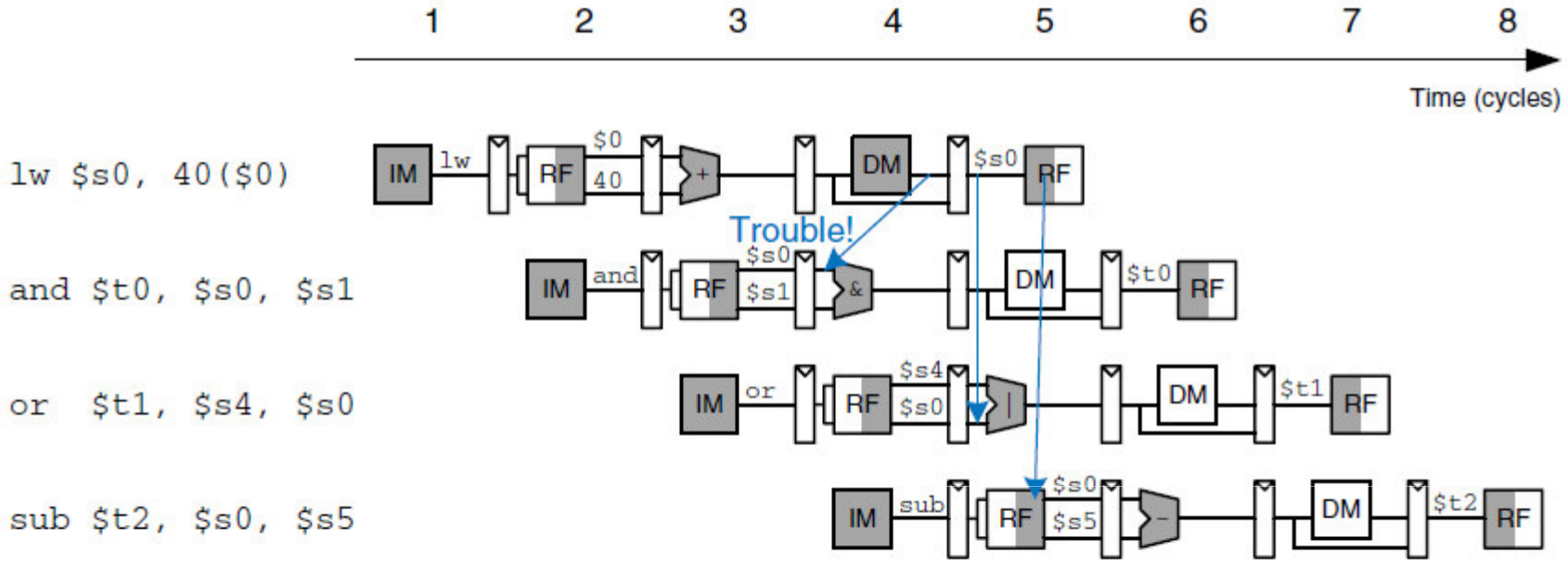


- Pokud výsledek vzniká dřív než jej následující instrukce skutečně potřebují je možné tento hazard řešit přeposíláním (forwarding)
- nastává když se shodují použité registry instrukce ve stupni E s M nebo WB
- proto musejí být čísla těchto registrů z těchto stupňů posílána do Hazard Unit
- taktéž zda cílový registr bude skutečně použit k zápisu – RegWrite z M a WB (netřeba přeposílat když se nezapisují..)

Řešení datových hazardů přeposíláním (forwarding)

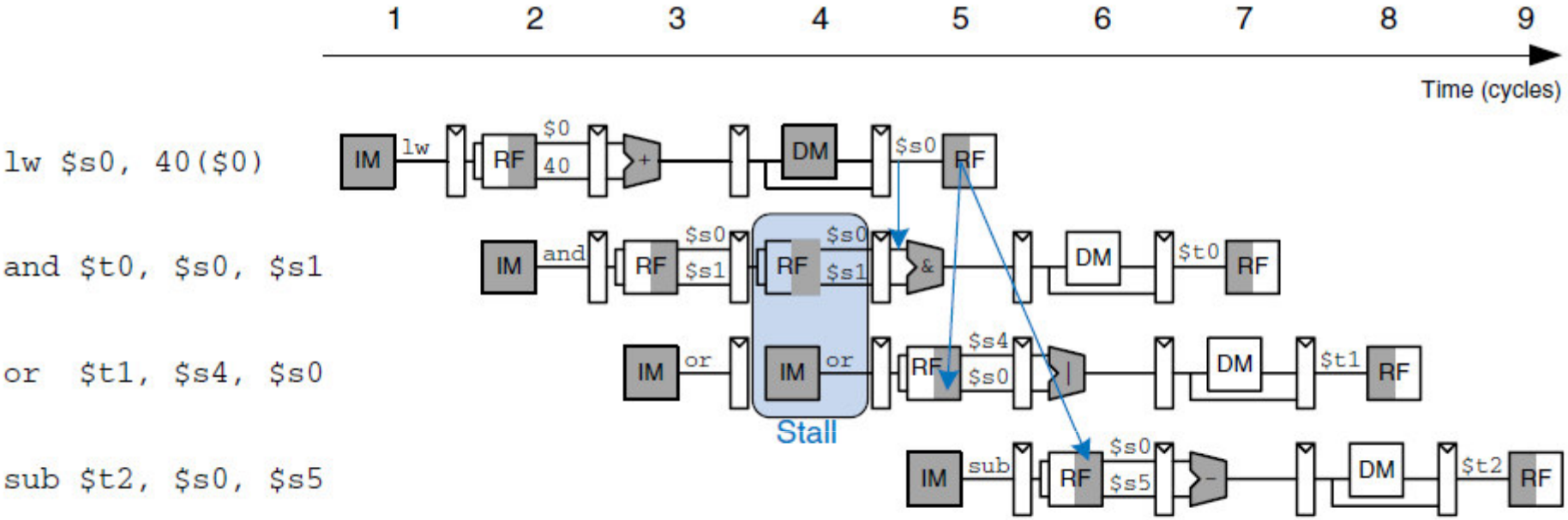


Řešení datových hazardů pozastavením (stall)



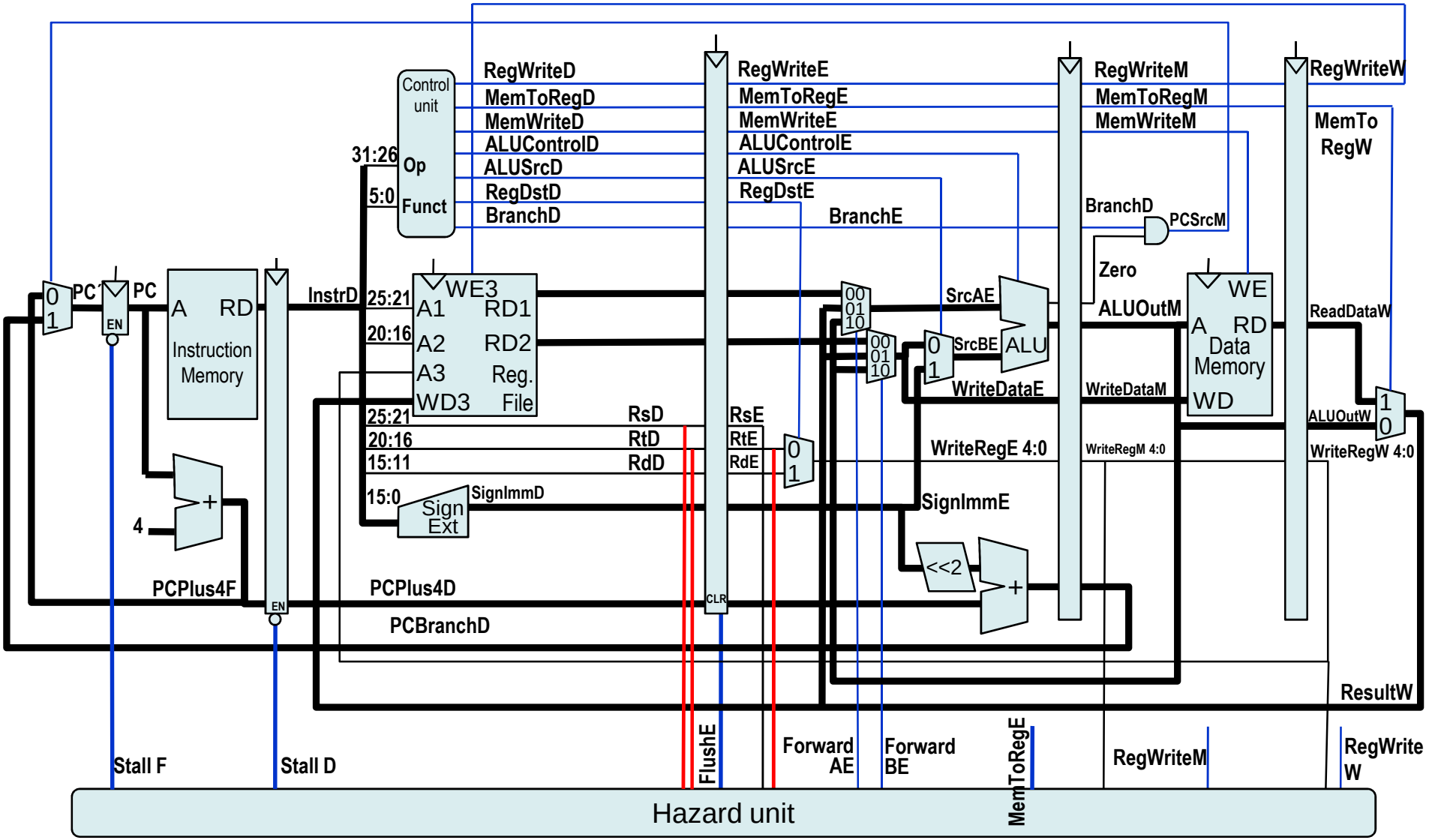
- Pokud následující instrukce potřebují výsledek dřív než skutečně vzniká je možné tento hazard řešit pozastavením (stall)
- Pozastavení pipeline je prostředkem řešení hazardů; nezvyšuje ILP
- stupně pipeline předcházející stupni kde hazard vzniká jsou pozastaveny do doby, než jsou k dispozici výsledky požadované následujícími instrukcemi – ty jsou pak přeposílány (forwarding)

Řešení datových hazardů pozastavením (stall)

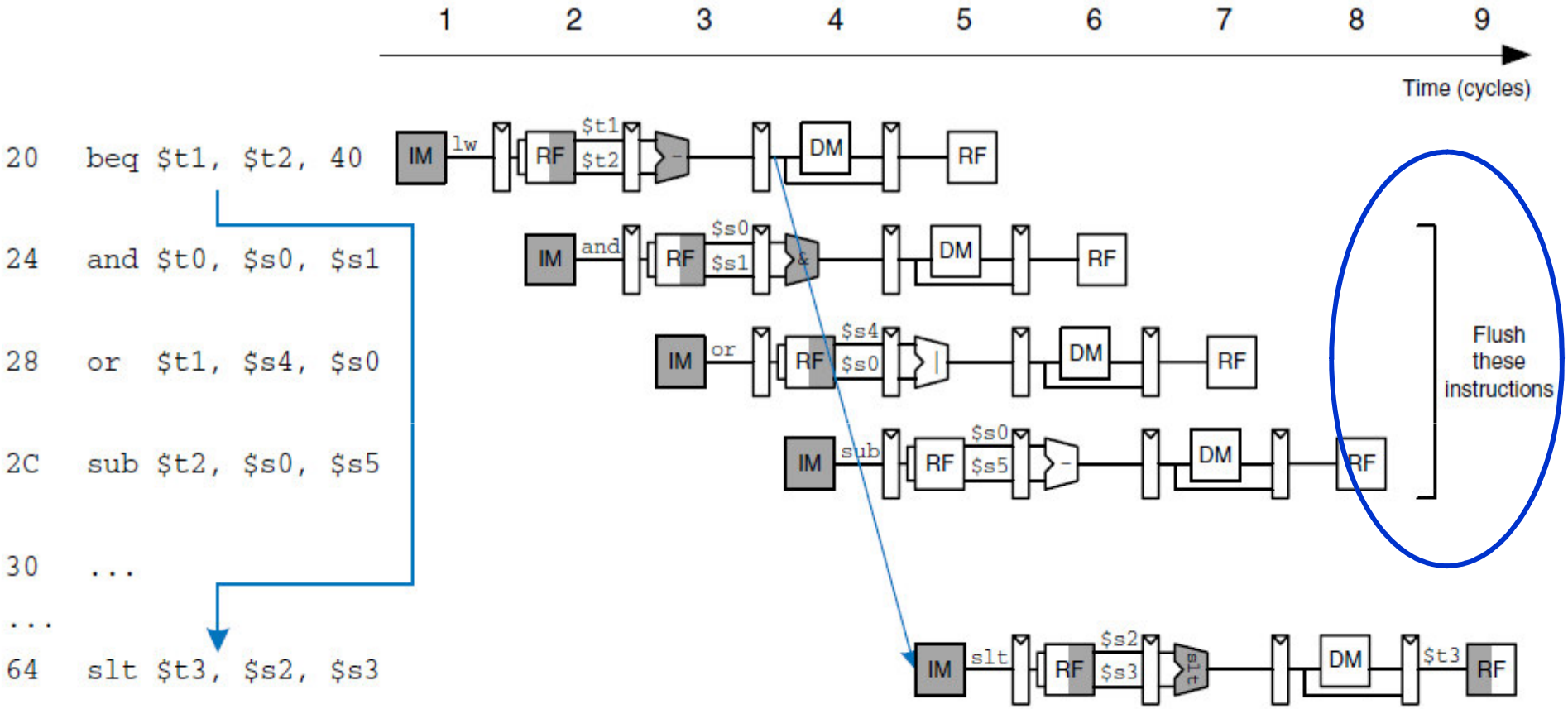


- pozastavení se dosáhne podržením hodnoty mezistupňových registrů
- výsledky z kolizního stupně se musejí „ztratit“ – řídicí signály umožňující měnit stav (kontext) procesoru (zápis pracovních registrů nebo do paměti, řízení povolení větvení) se nulují
- obojí se dosáhne přidáním řídicích vodičů k mezistupňovým registrům umožňujících měnit, uchovat nebo nulovat jejich obsah

Řešení datových hazardů pozastavením (stall)

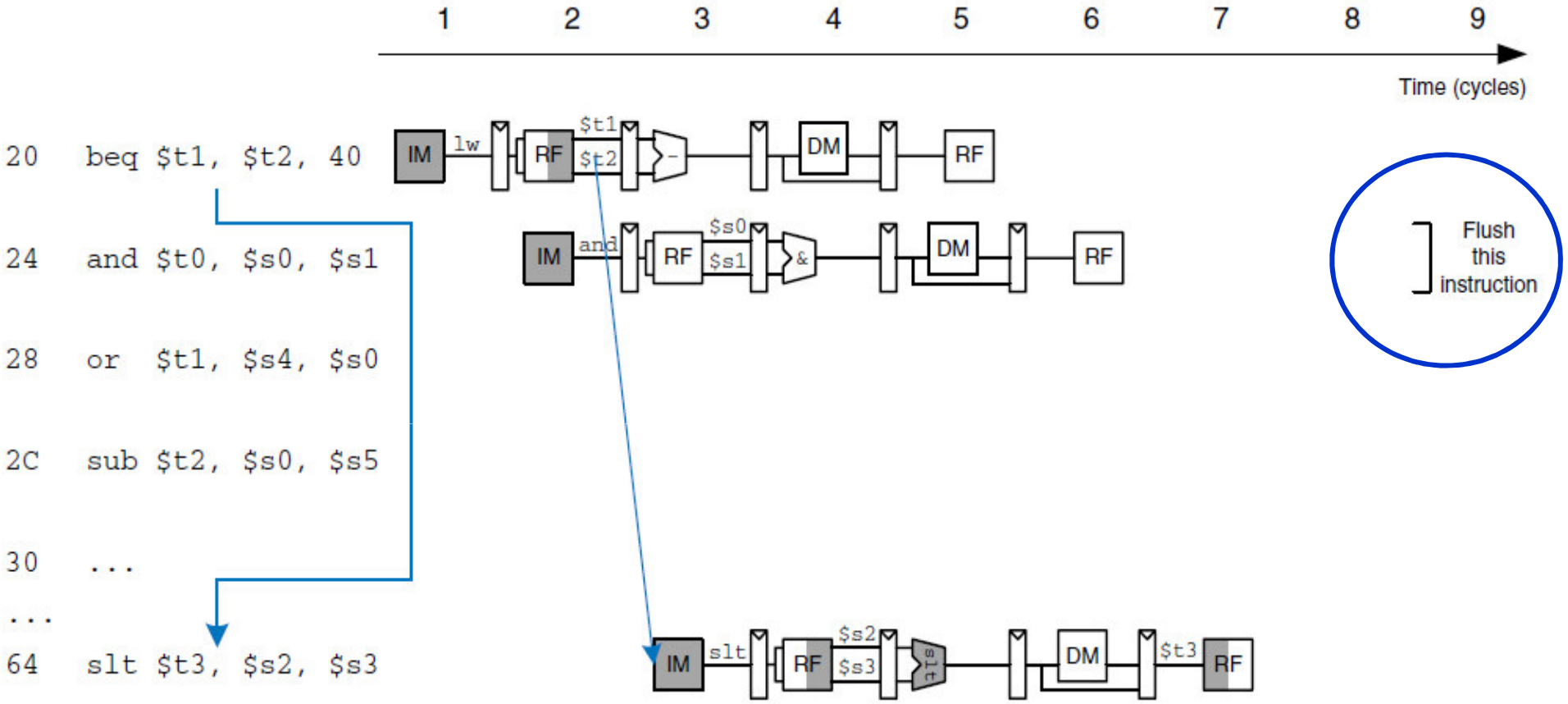


Řídicí hazardy



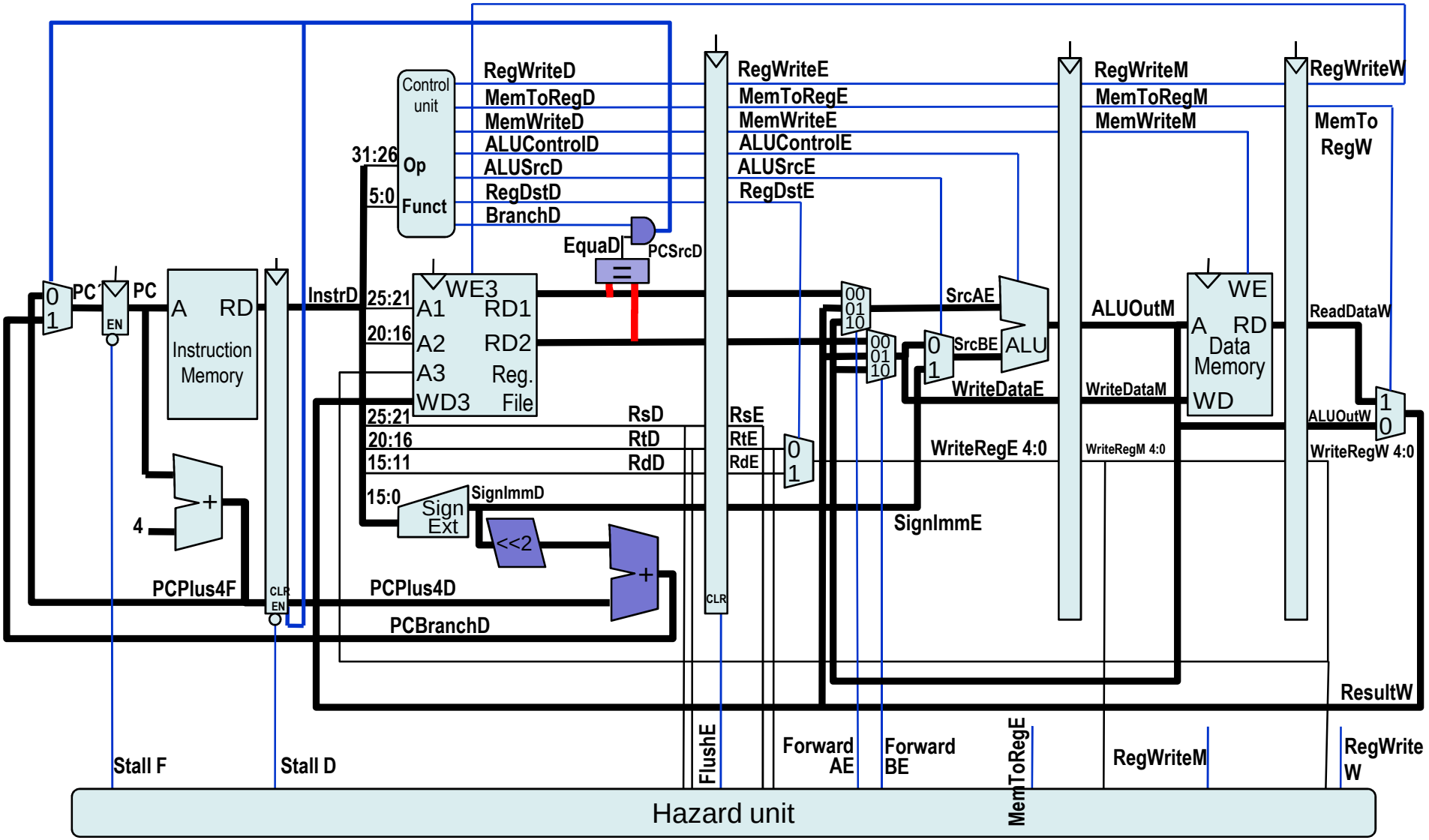
- výsledek porovnání je znám až v 4. cyklu..

Řídicí hazardy – raději znát výsledek dříve..

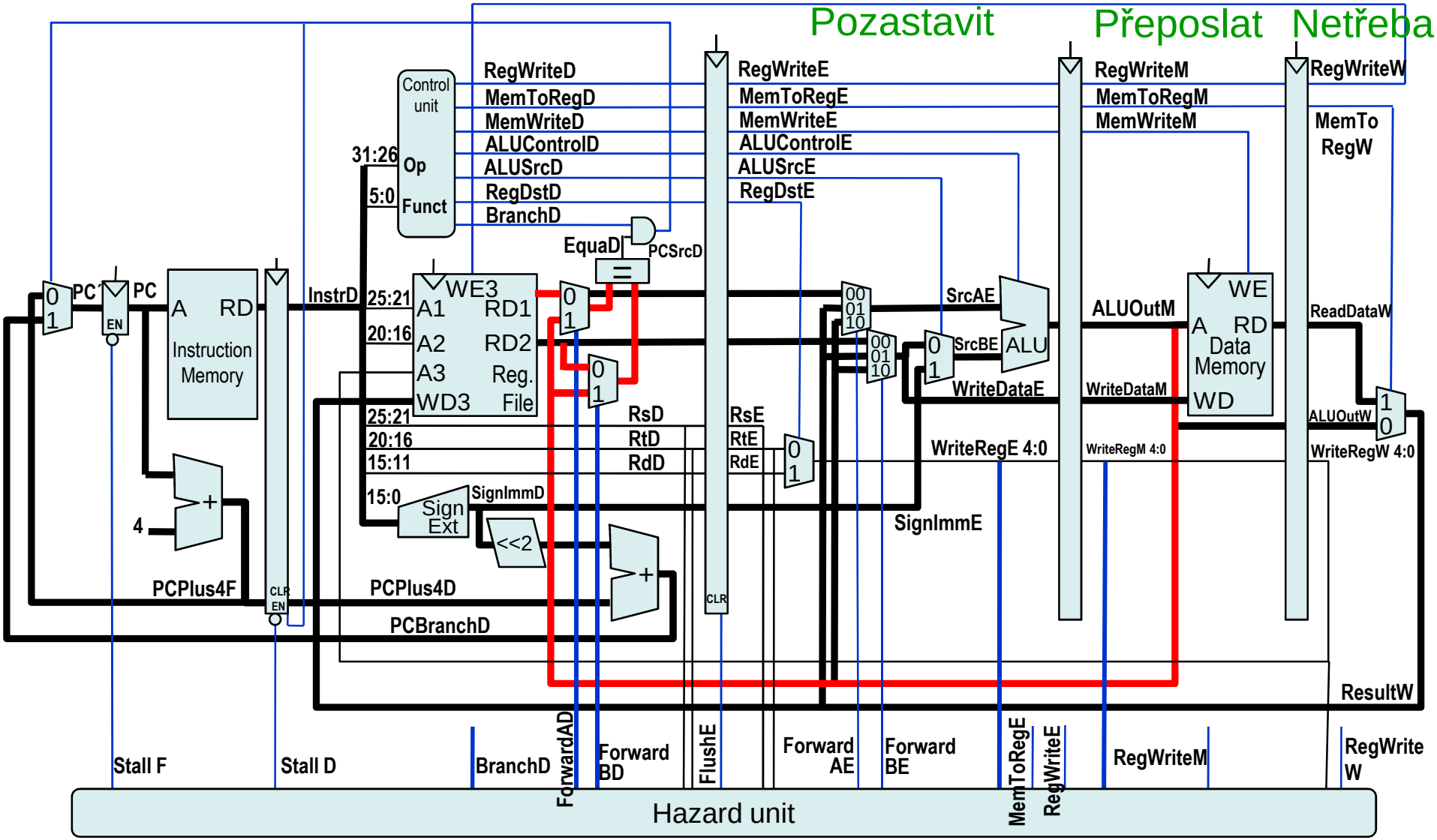


- pokud dokážeme stanovit výsledek porovnání už v 2. cyklu můžeme redukovat tzv. „misprediction penalty“
- přesun rozhodování dopředu může zavést nové RAW hazardy..

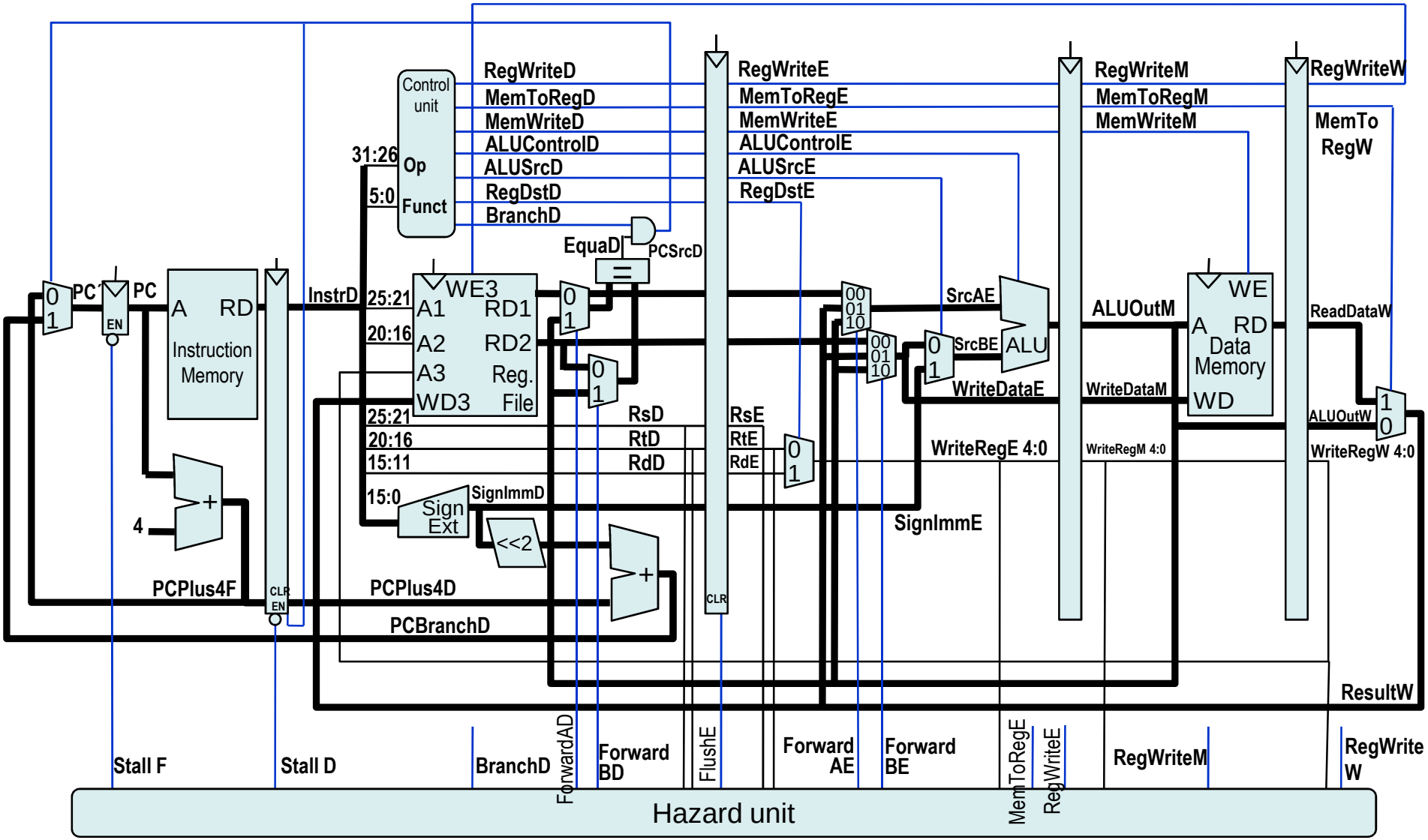
Řešení řídicích hazardů vyprázdněním (flush)



Řešení vzniklých RAW hazardů přeposíláním nebo pozastavením



Hotovo – navržený zřetězený procesor



Zřetězený procesor – výkon: IPS = IC / T = $IPC_{str}.f_{CLK}$

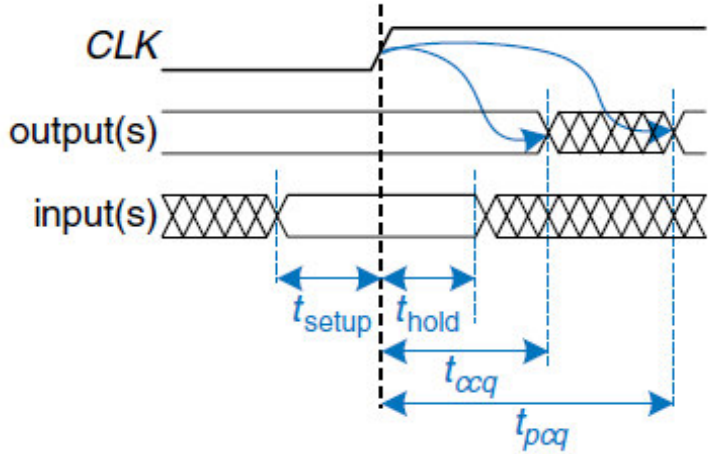
- Jaká může být maximální frekvence procesoru?
- Který stupeň je nejpomalejší?
- Dobu cyklu určuje nejpomalejší stupeň
- V našem případě:
  Tc = 300 ns --> 3 333 kHz

  Zanedbejme plnění pipeline, všechna pozastavení pipeline a všechna vyprázdnění. Pak bude IPC = 1.
  IPS = 1. 3 333e3 = 3 333 000 instrukcí za sekundu

- Zavedením 5-stupňového zřetězení jsme zlepšili propustnost 3 333 000/ 980 000 = 3,4 krát! (i za předpokladu IPC=1)

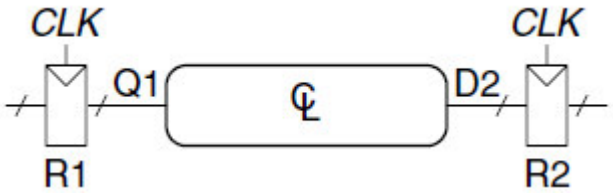
Zřetězený procesor – časování

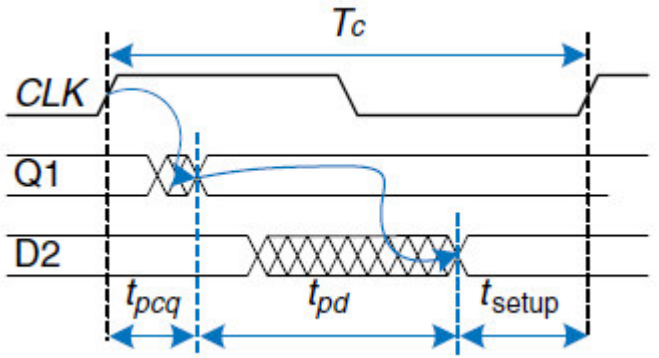
- Specifikace časování pro synchronní sekvenční obvod:



t_{setup} – předstih
t_{hold} – přesah

- Omezující podmínka na předstih signálu před hodinami:

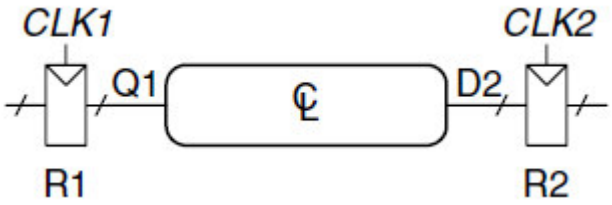


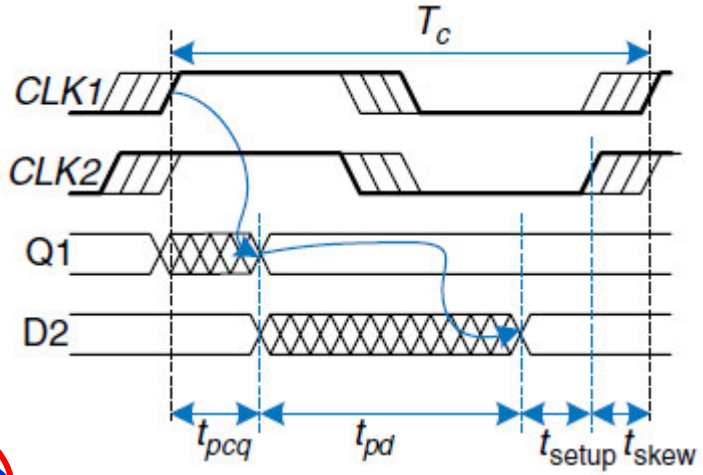


$$Tc >= t_{pcq} + t_{pd} + t_{setup}$$

Zřetězený procesor – časování

- Omezující podmínka na předstih signálu před hodinami (zohlednění nedokonalosti rozvodu hodin):



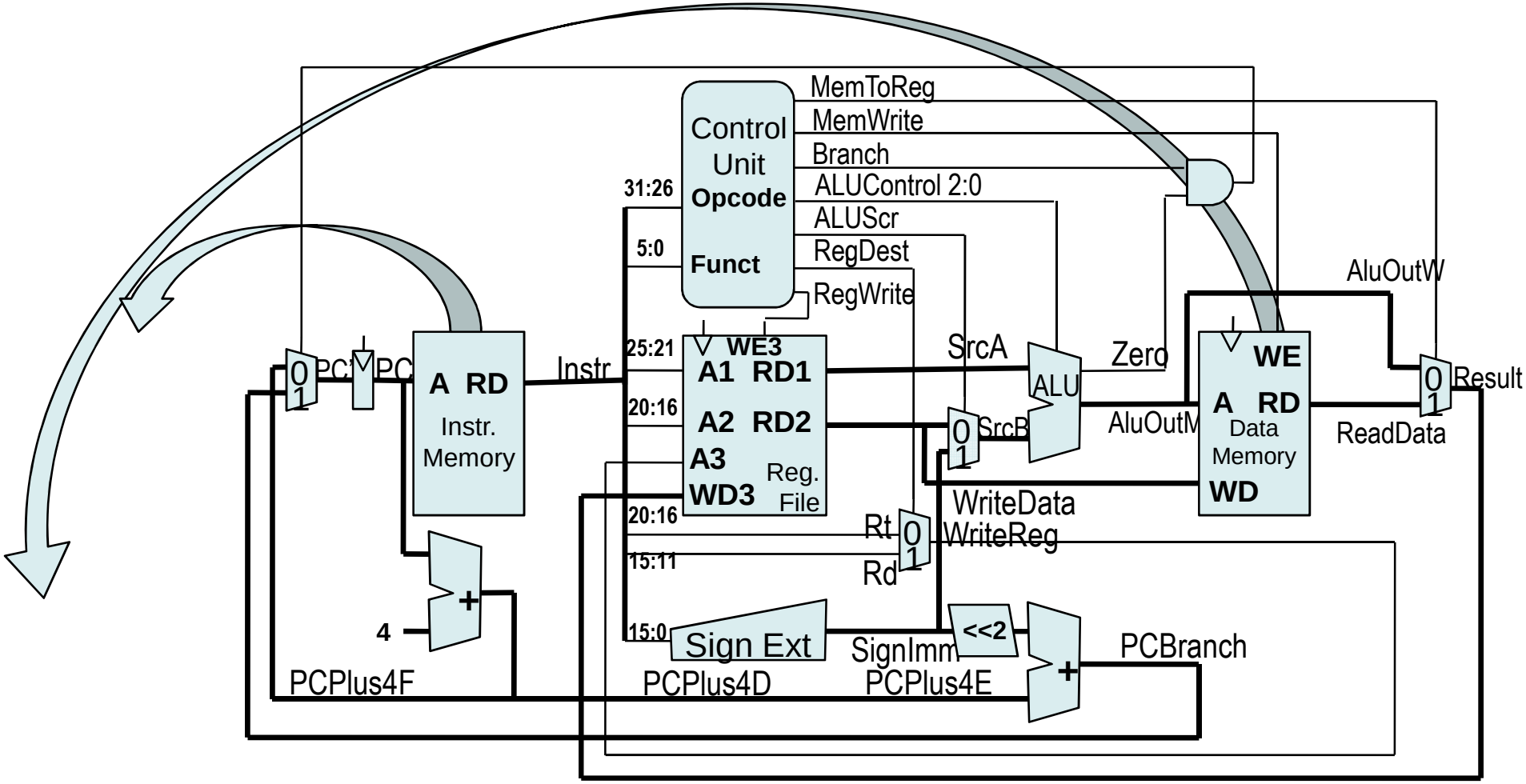


$$Tc >= t_{pcq} + t_{pd} + t_{setup} + t_{skew}$$

Představuje omezení pokud se začíná blížit
nebo dokonce dominovat nad t_{pd}
(příliš hluboká pipeline / příliš mnoho stupňů…)

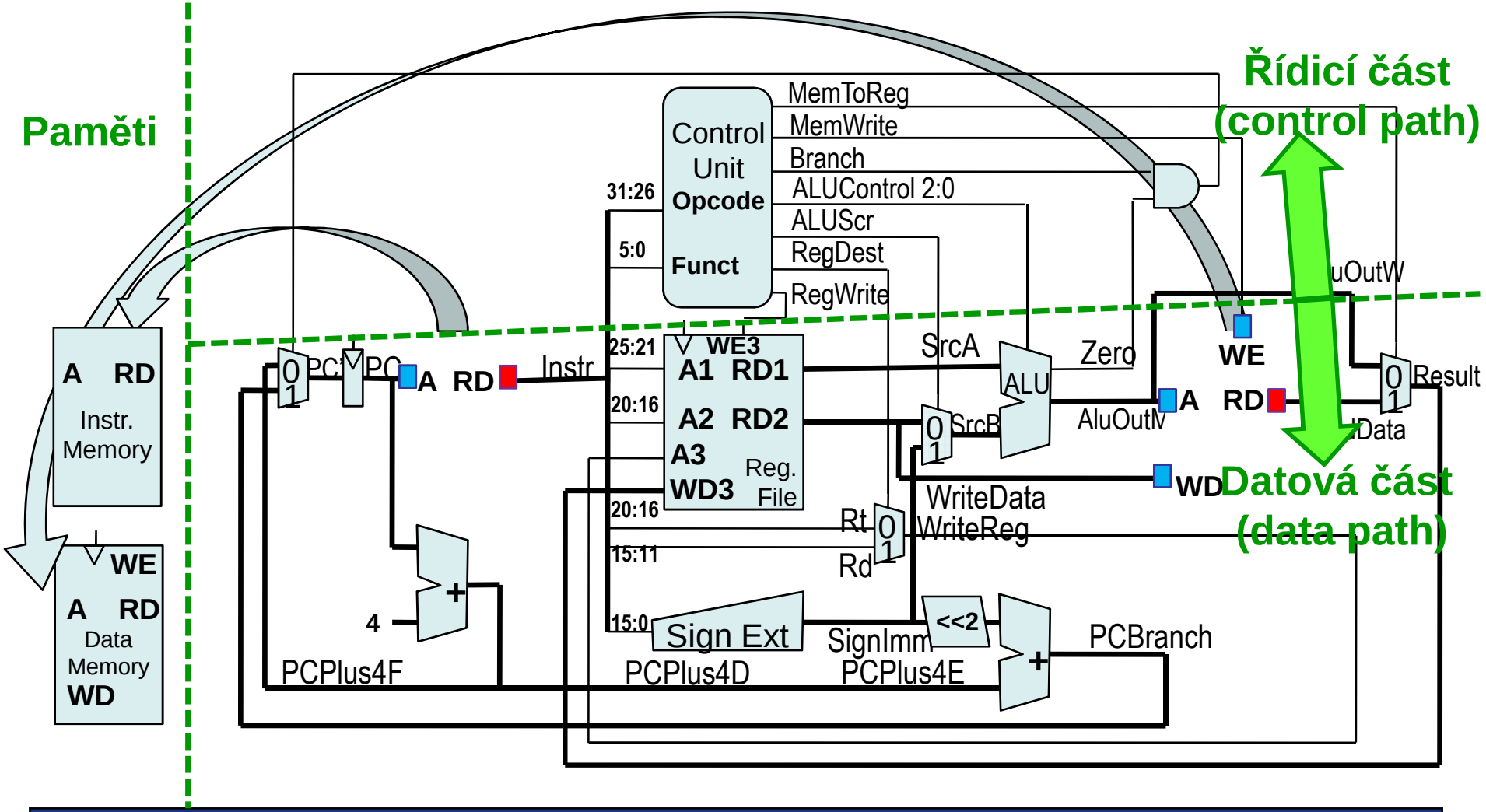
Co jsme navrhli?

Návrat k nezřetězenému procesoru

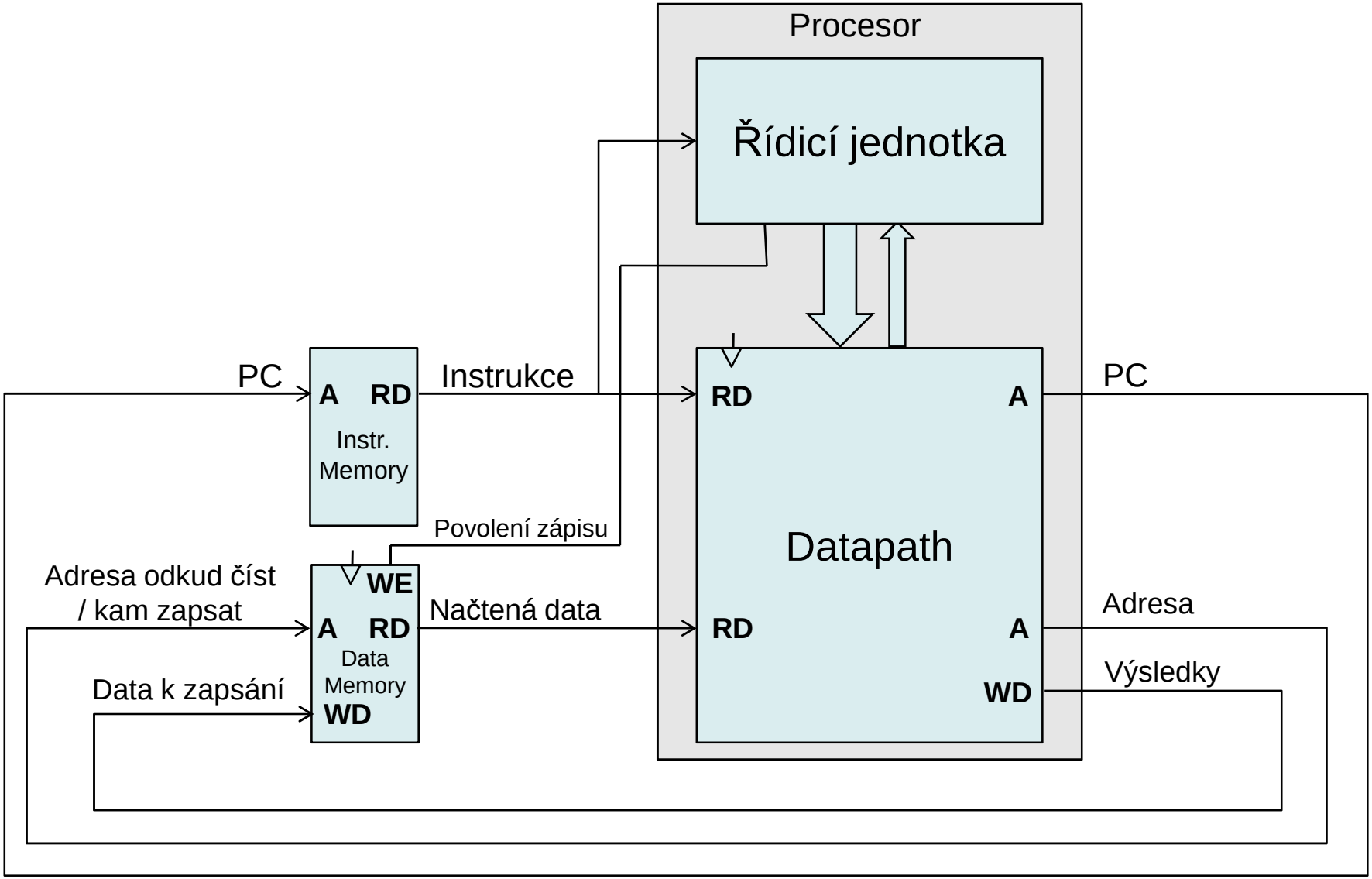


Co jsme navrhli?

Návrat k nezřetězenému procesoru



Co jsme navrhli?

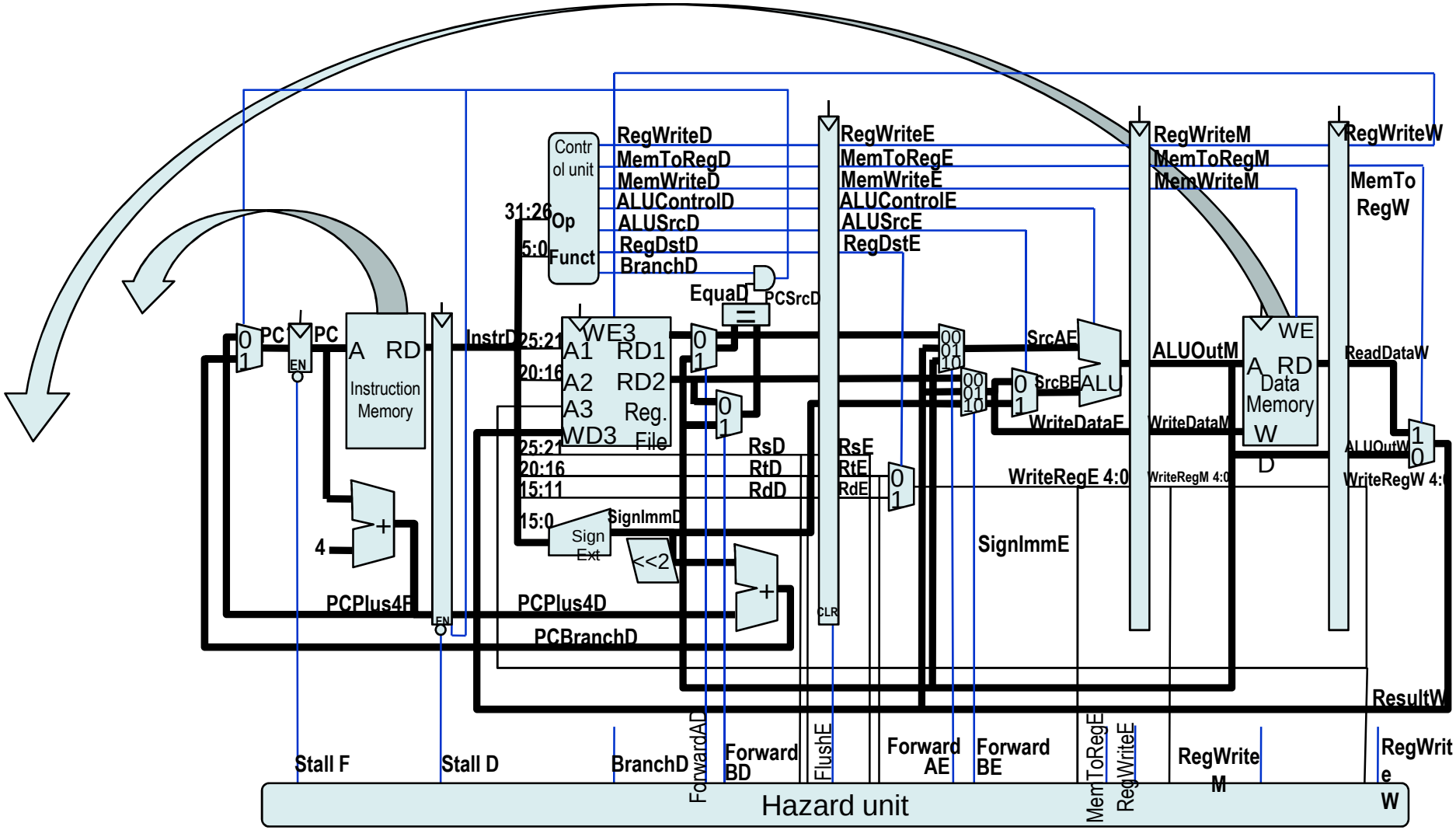


Co s paměťmi pokud jsou velké?

- Čím větší paměť, tím větší přístupová doba…
- Připomeňme si zpoždění na kritické cestě :
  $Tc = t_{PC} + t_{Mem} + t_{RFread} + t_{ALU} + t_{Mem} + t_{Mux} + t_{RFsetup}$

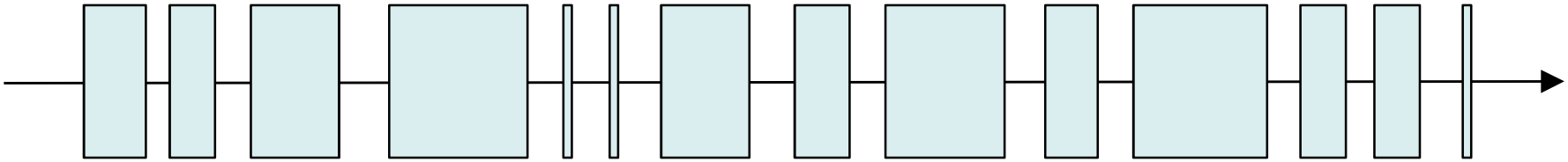
  Jak nedegradovat frekvenci procesoru??
- Řešení: Víceúrovňový paměťový systém
  (podrobněji o této problematice v 10. a 11. přednášce)

Co jsme navrhli? – zřetězená verze



Vyvažování stupňů zřetězení

Lineární zřetězení:

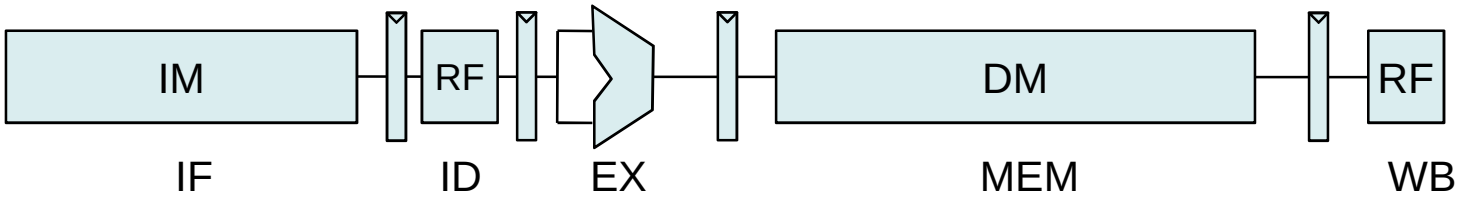


(též: stromový sumátor, stromová násobička, iterační dělička..)

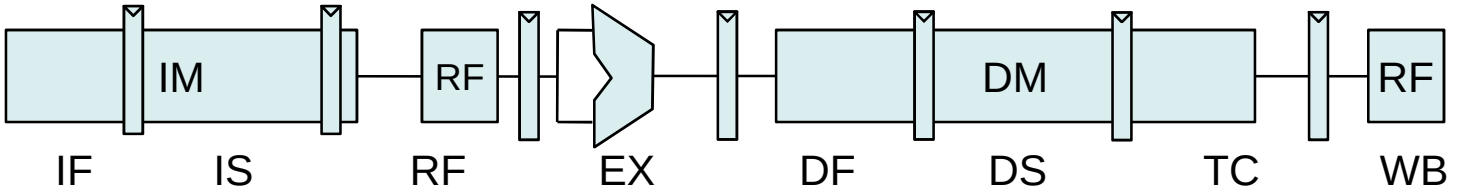
- **Vyvažování**: cílem je rozdělit jednotlivé bloky do *N* stupňů tak, aby ve zpoždění ve všech stupních bylo pokud možno stejné…
- Volba počtu stupňů závisí od preference: propustnost vs. latence

Superzřetězení

- nevyvážené 5-stupňové zřetězení:



- hlubší zřetězení vzniklá další dekompozicí



- přináší možnost dalšího zvýšení pracovní frekvence, avšak také řadu dalších problémů..
- další forwarding, nárůst pozastavení pipeline, hazardy