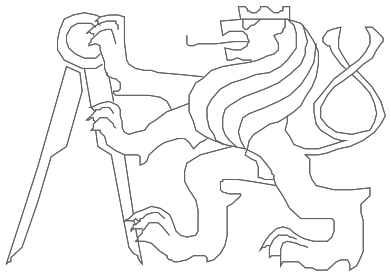
Pokročilé architektury počítačů

05

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



České vysoké učení technické, Fakulta elektrotechnická

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

Výklad syntaxe a sémantiky instrukce: například `lw`

`lw` – load word - čtení slova z datové paměti

Description	A word is loaded into a register from the specified address
Operation:	$t = MEM[$s + offset];
Syntax:	lw $t, offset($s)
Encoding:	`1000 11ss ssst tttt iiii iiii iiii iiii`

Uložme (načtime) slovo z paměti na adrese 0x4 do registru č.11:

lw $11, 0x4($0)

```
1000 11ss ssst tttt iiii iiii iiii iiii
1000 1100 0000 1011 0000 0000 0000 0100
```

0 11 4

0x 8C 0B 00 04 – strojový kód instrukce lw $11, 0x4($0)

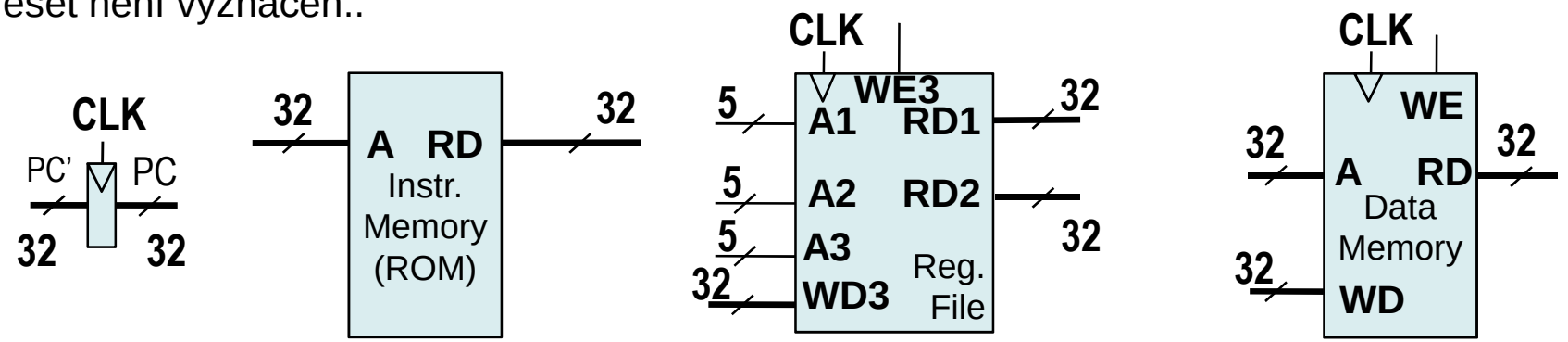
Poznámka: V registu $0 je trvale uložena konstanta 0.

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..



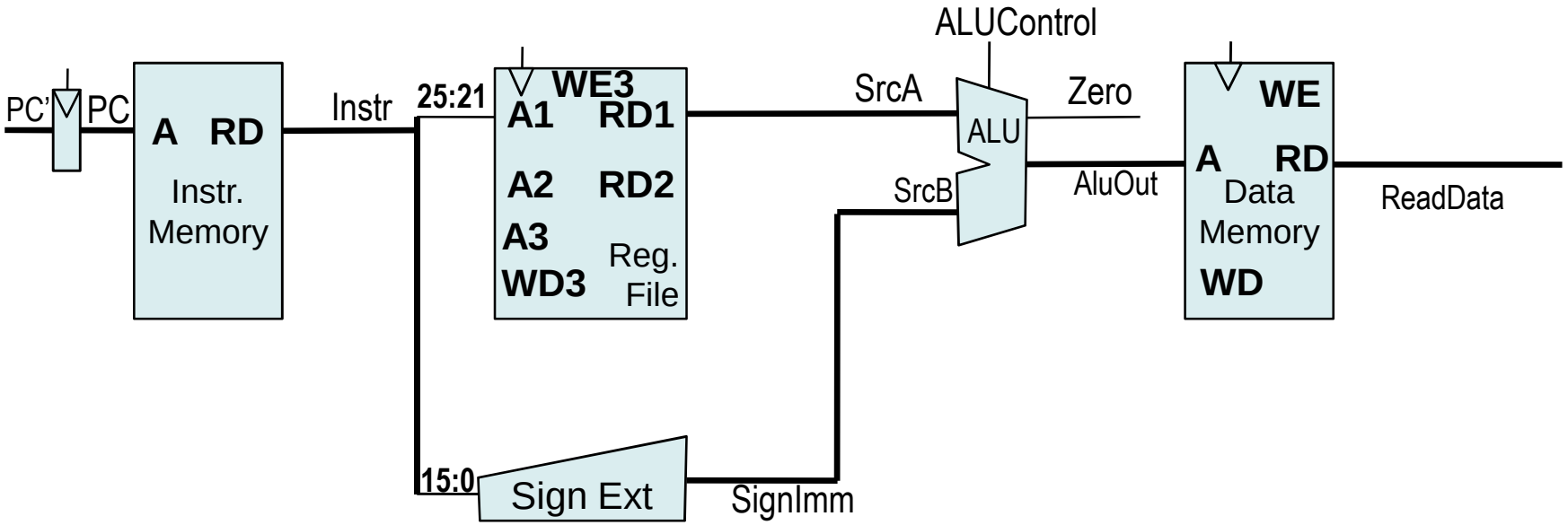
zápis s náběžnou hranou CLK při WE = 1

čtení po uplynutí dostatečně dlouhé doby

Jedno-cyklový procesor – návrh – podpora lw

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

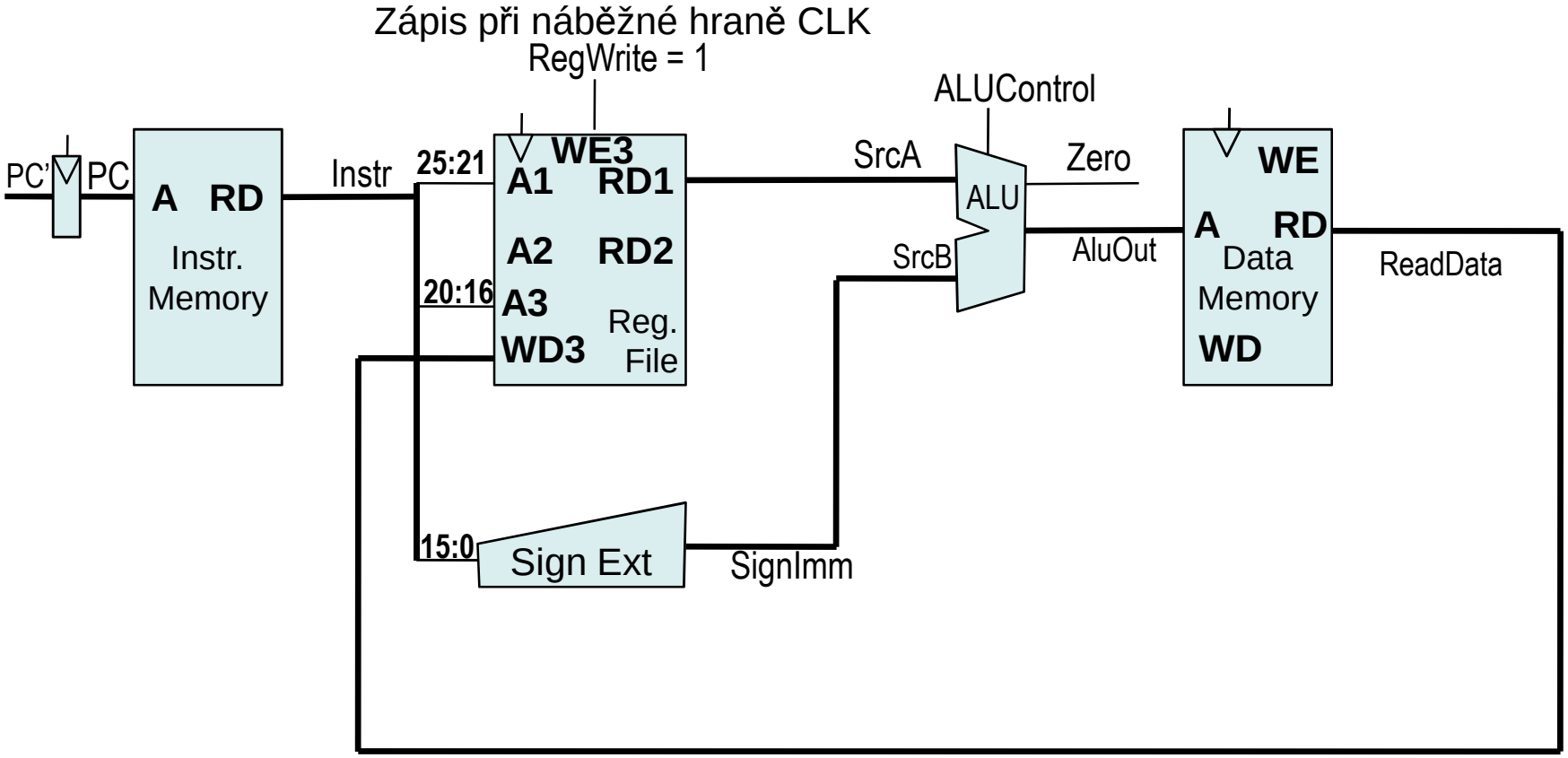
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



Jedno-cyklový procesor – návrh – podpora lw

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

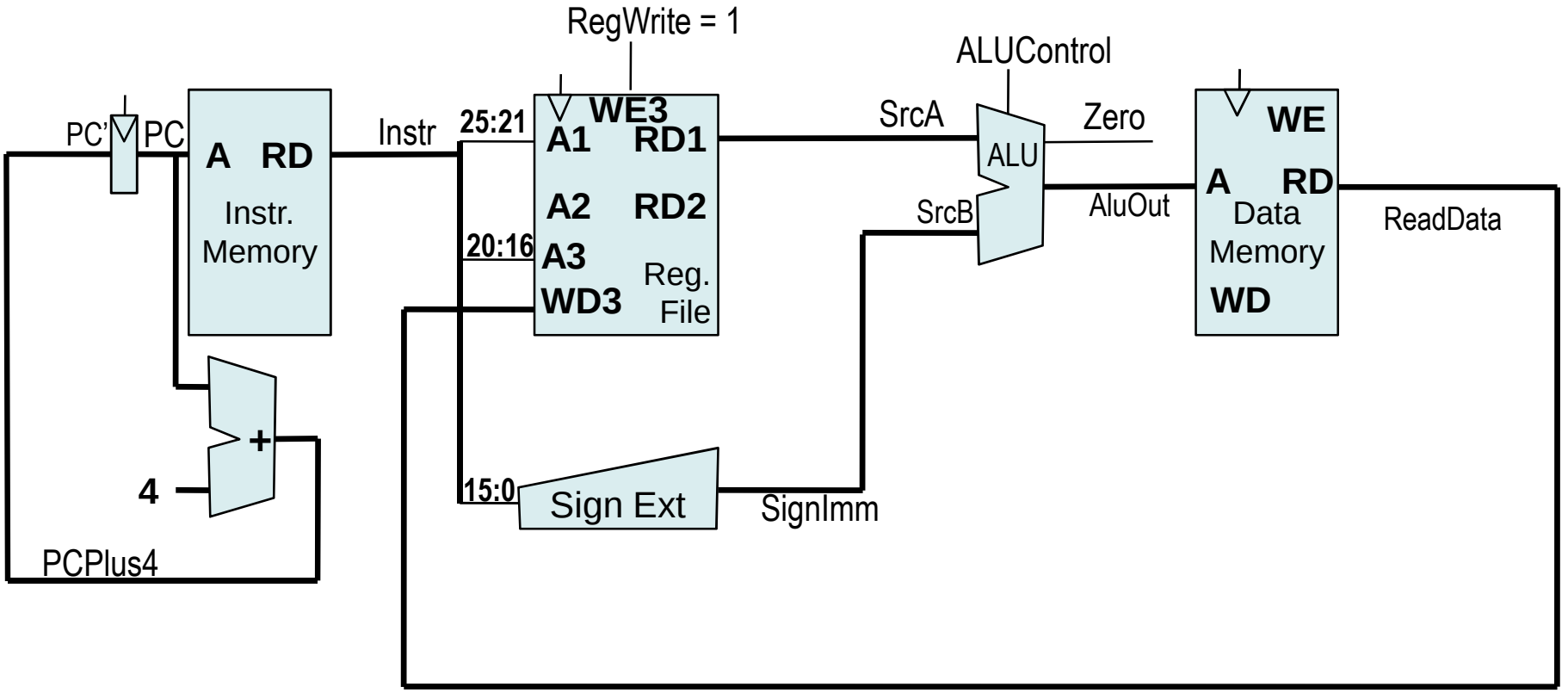
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



Jedno-cyklový procesor – návrh – podpora lw

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

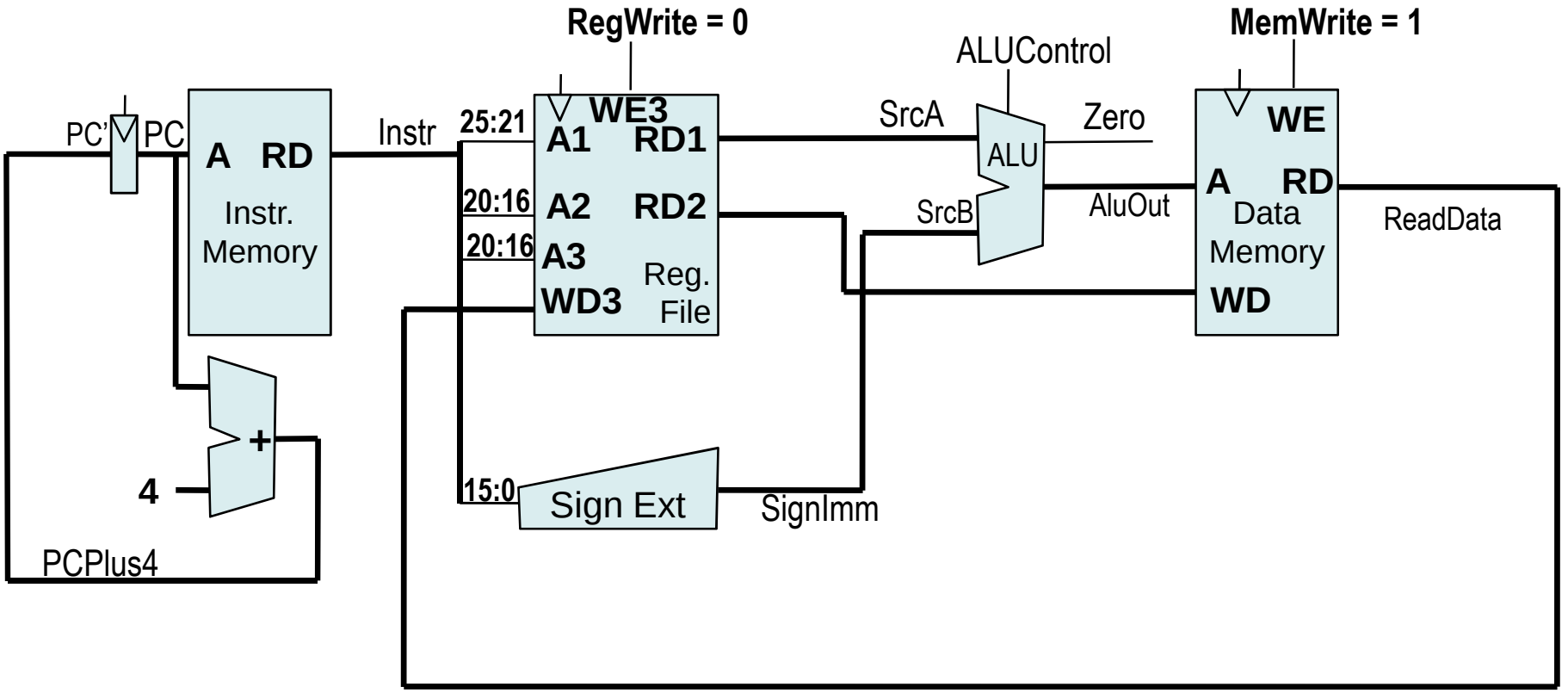
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



Jedno-cyklový procesor – návrh – podpora sw

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

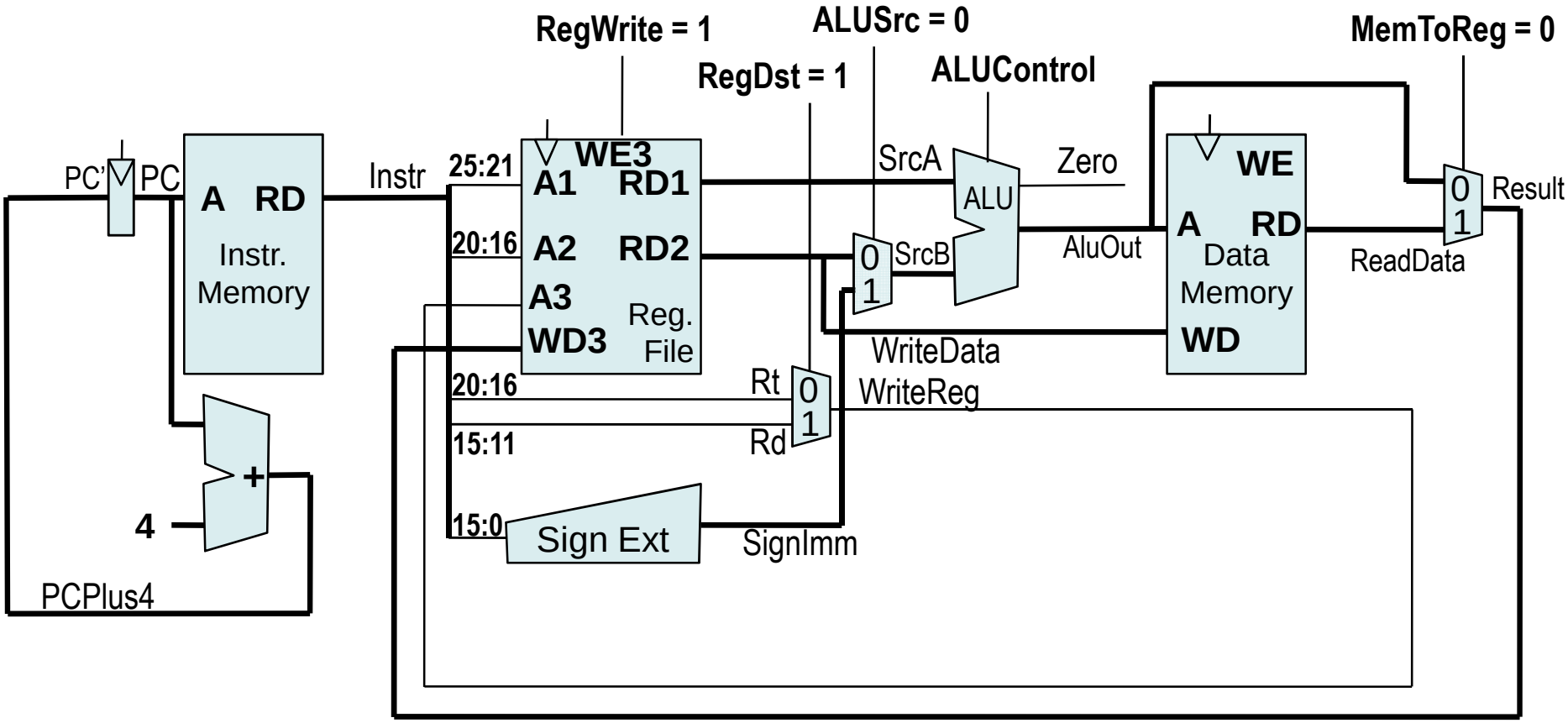
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



Jedno-cyklový procesor – návrh – podpora add

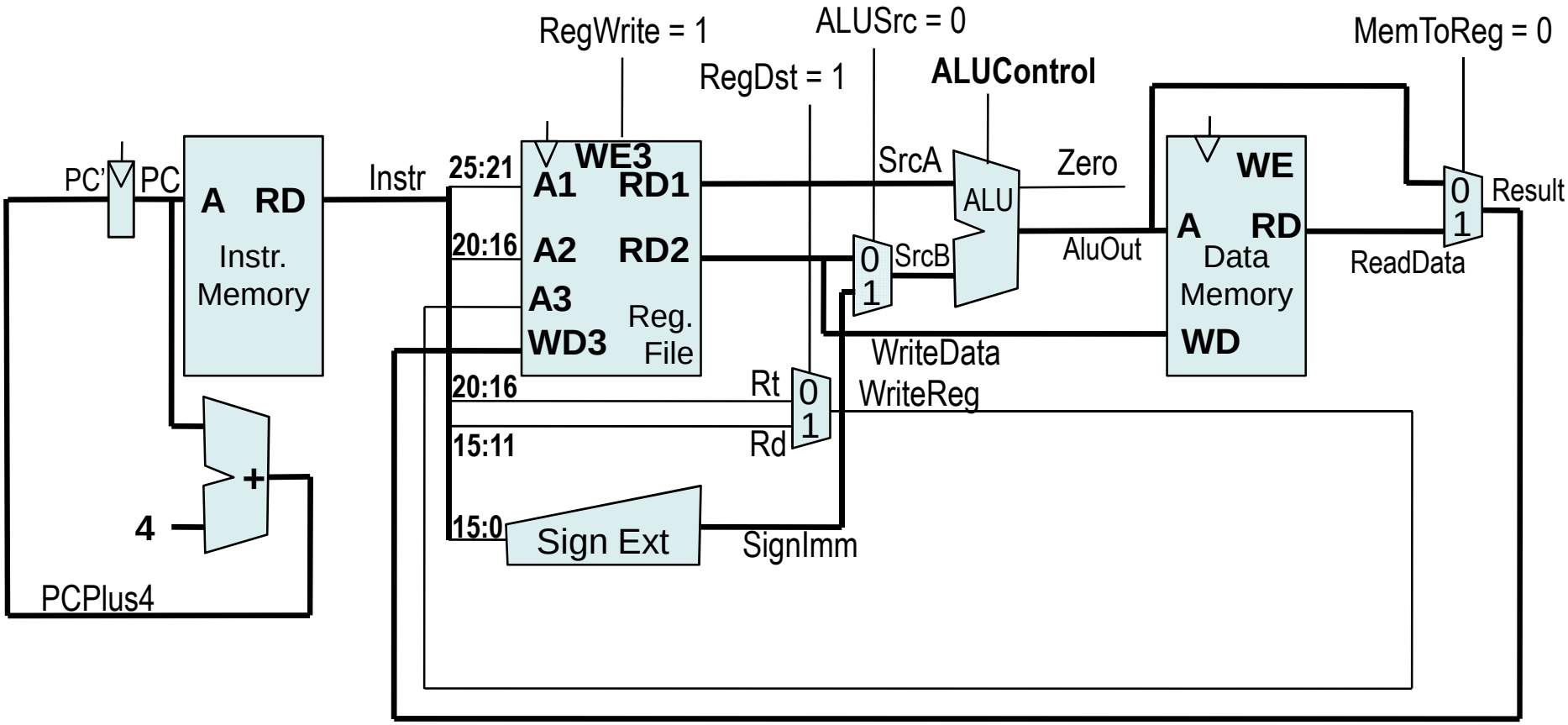
- add: typ R; rs, rt – zdroje, rd – cíl, funct – operace součtu

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

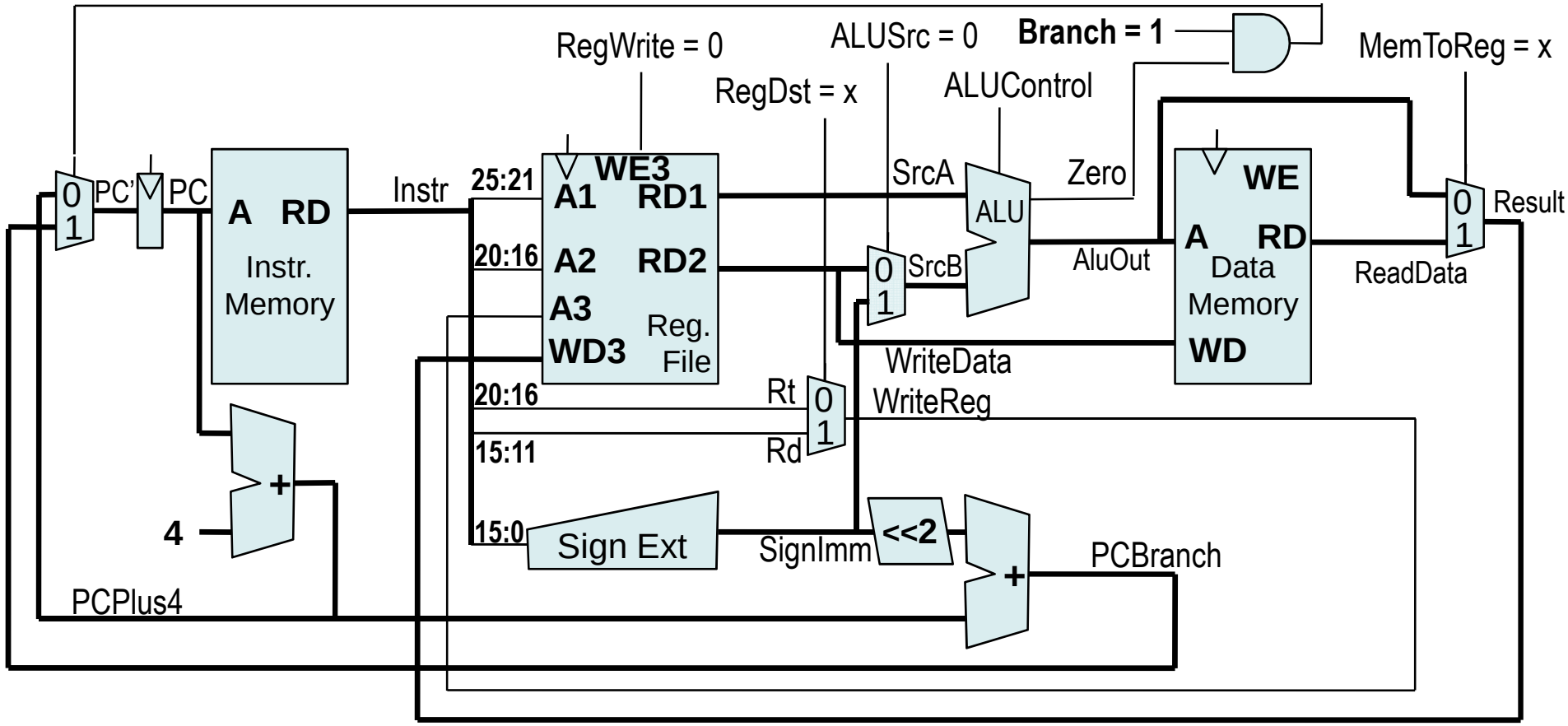
- jediné v čem se liší od add je operace ALU -> datapath beze změny; rozdíl v ALUControl



Jedno-cyklový procesor – návrh – podpora beq

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

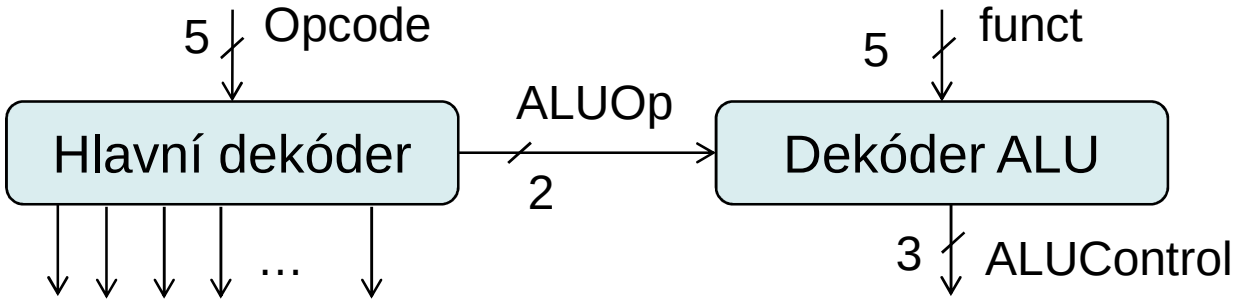
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0



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 základě **opcode** a **funct**



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

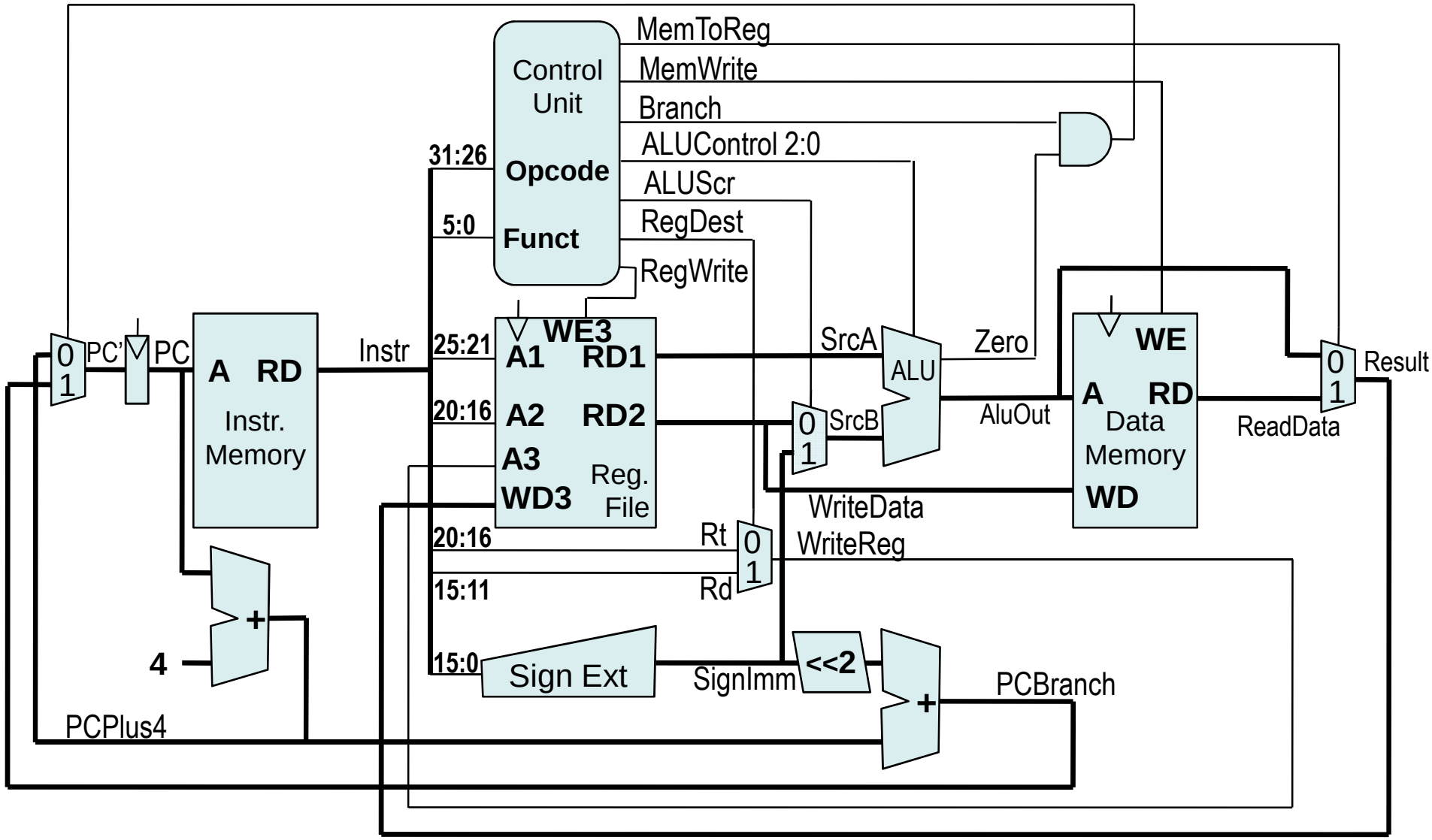
	Opcode	RegWrite	RegDst	ALUSrc	ALUOp	Branch	Mem Write	MemTo Reg
R typ	000000	1	1	0	10	0	0	0
lw	100011	1	0	1	00	0	0	1
sw	101011	0	X	1	00	0	1	X
beq	000100	0	X	0	01	1	0	X

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..

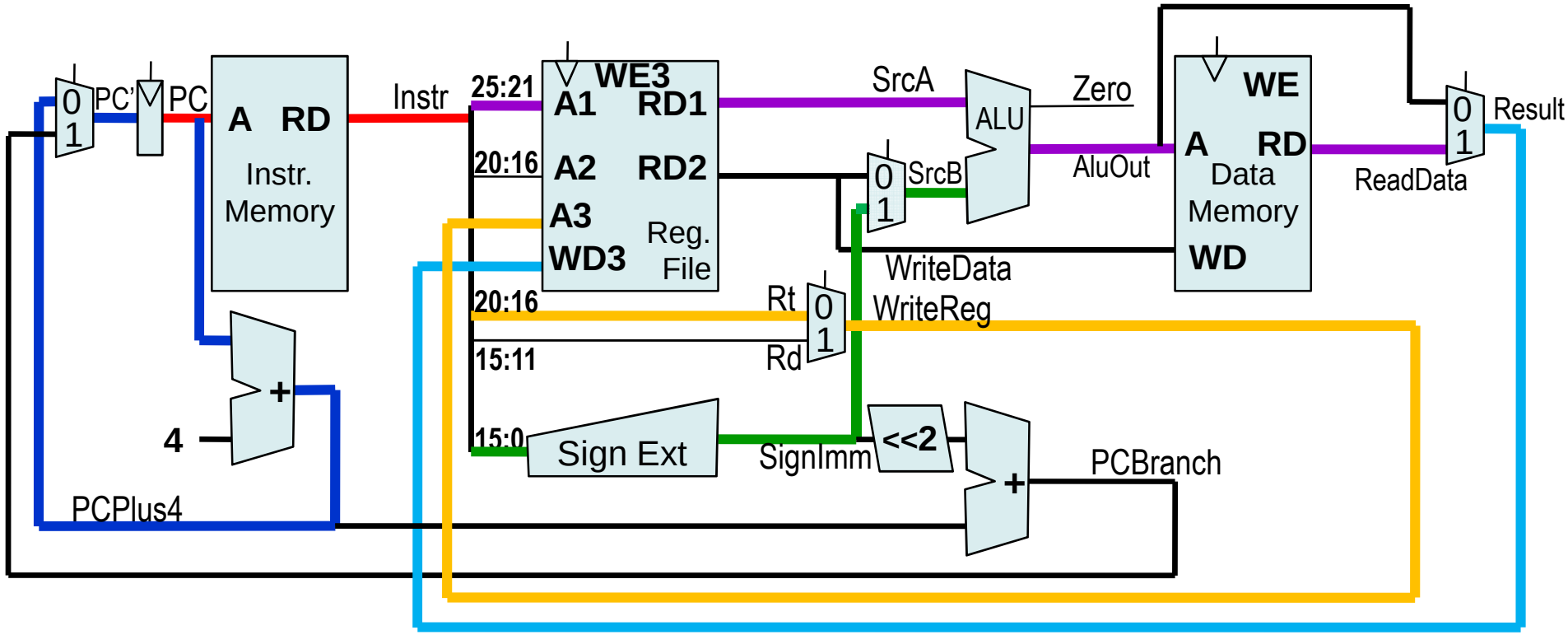
Jedno-cyklový procesor



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

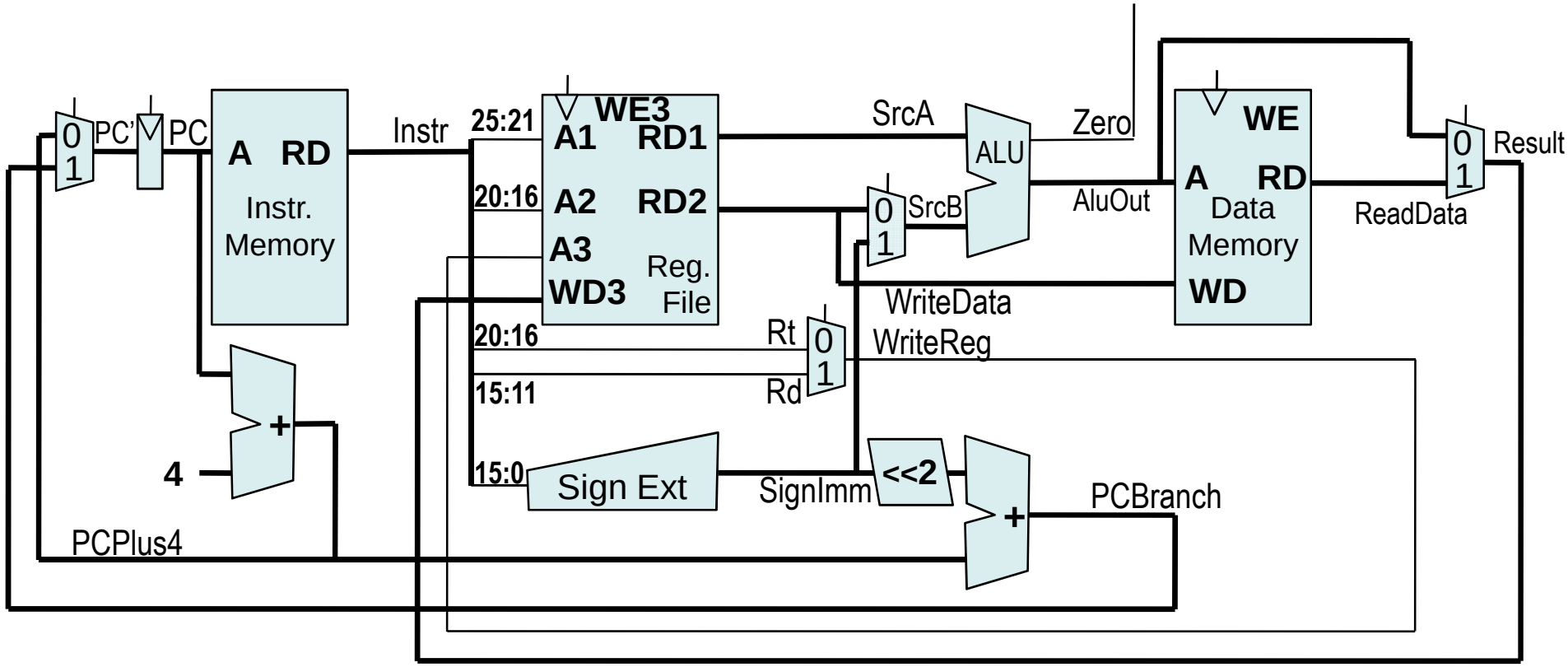
ID – dekódování instrukce a načtení registrů specifikovaných v instrukci, provedení testu na rovnost registrů (kvůli možnému větvení), znaménkové rozšíření offsetu, výpočet cílové adresy pro případ větvení (zn. rozš. offset + PC)

EX – operace ALU

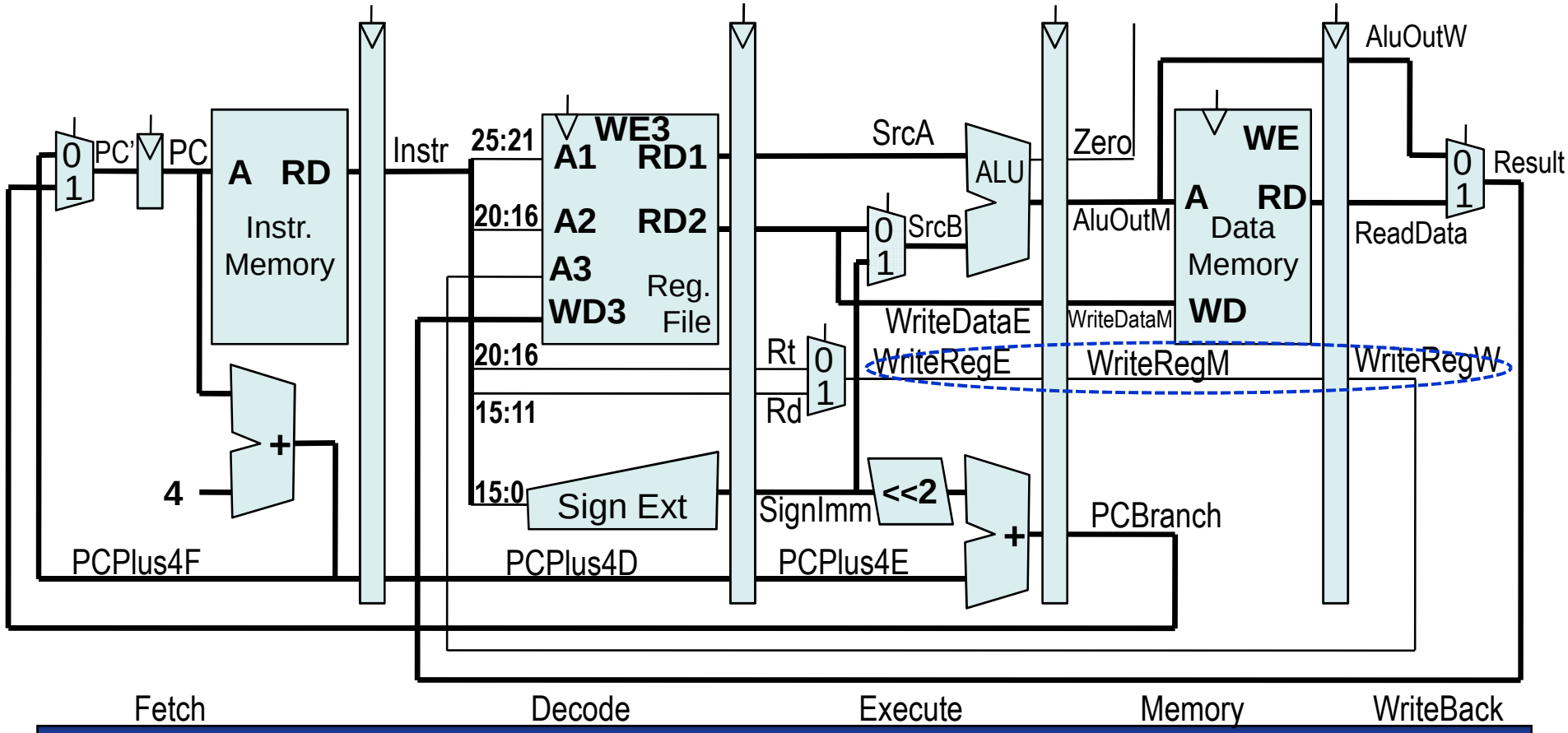
MEM – v případě instrukce *load* /store – čtení/zápis do paměti

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

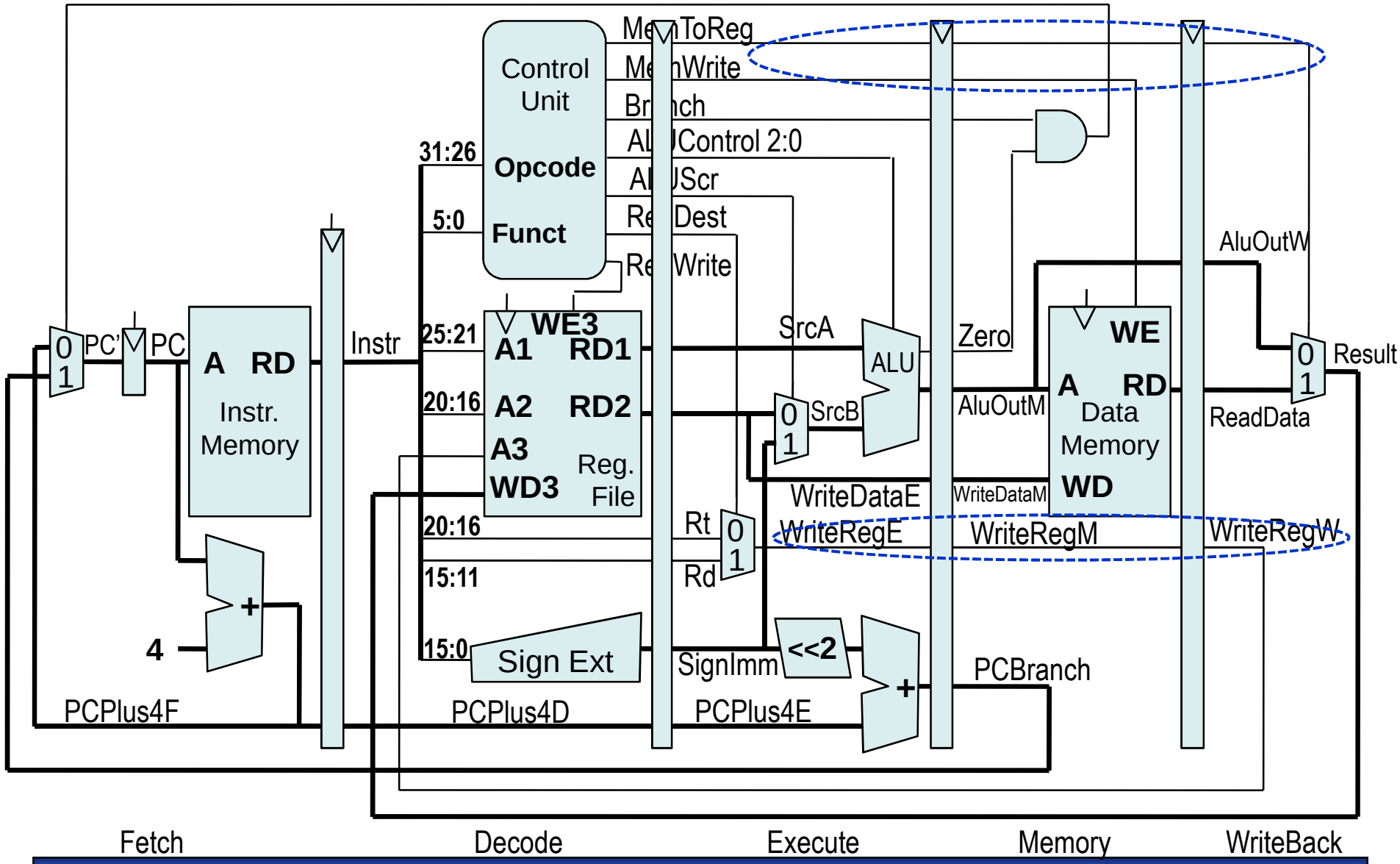
Nezřetězené vykonávání



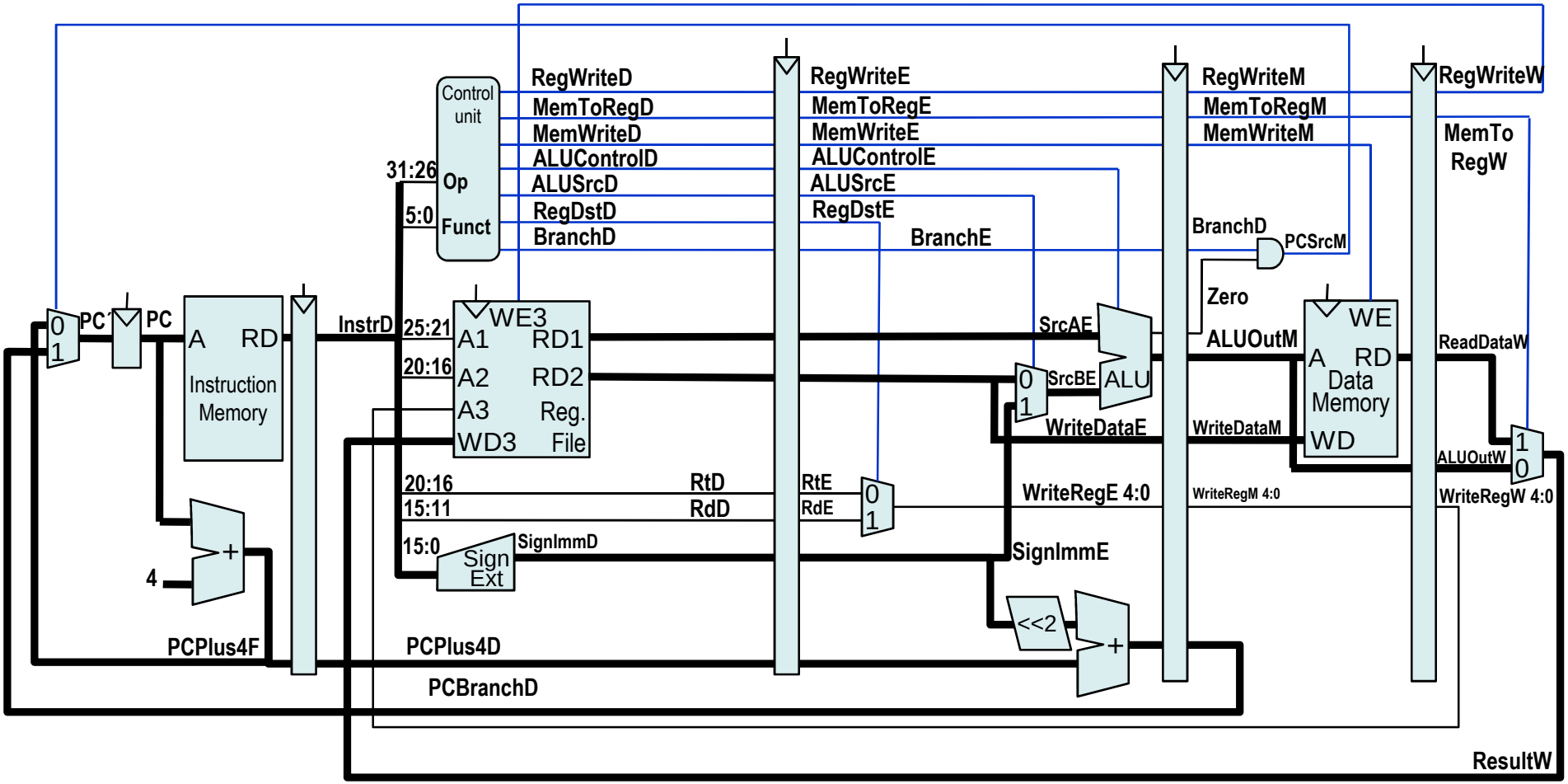
Zřetězené vykonávání



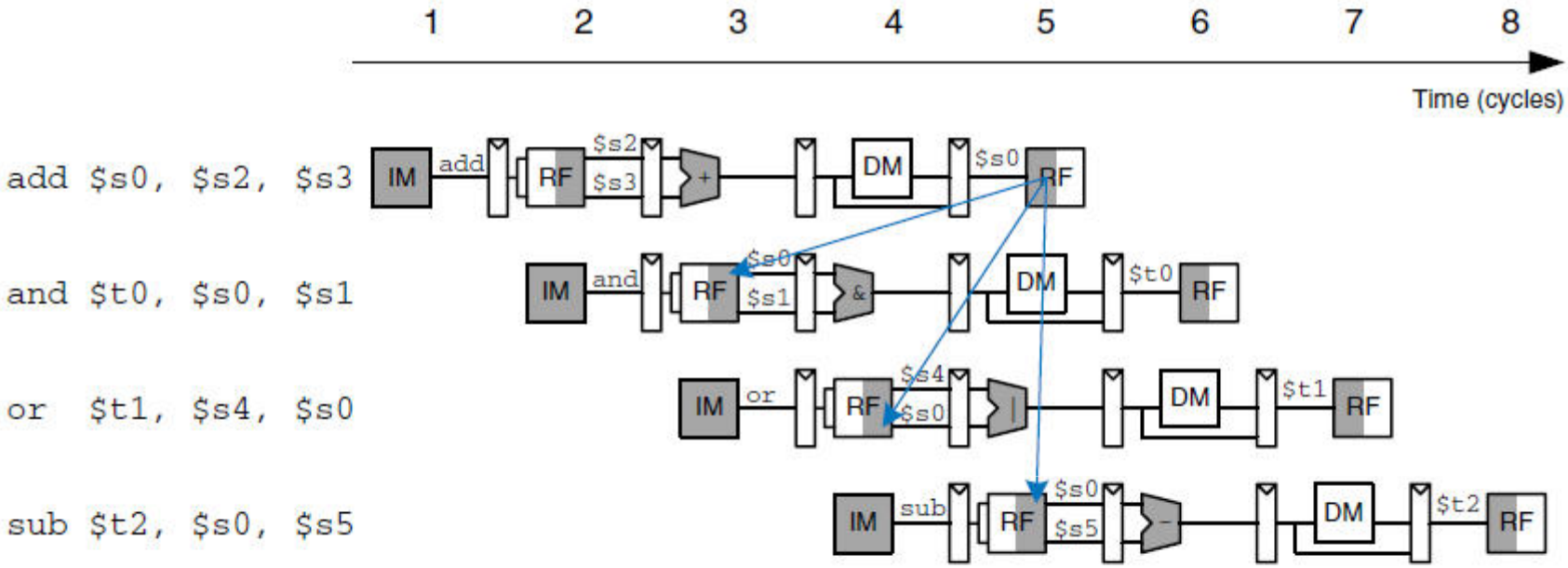
Zřetězené vykonávání



Totéž, pouze zmenšeno a překresleno…

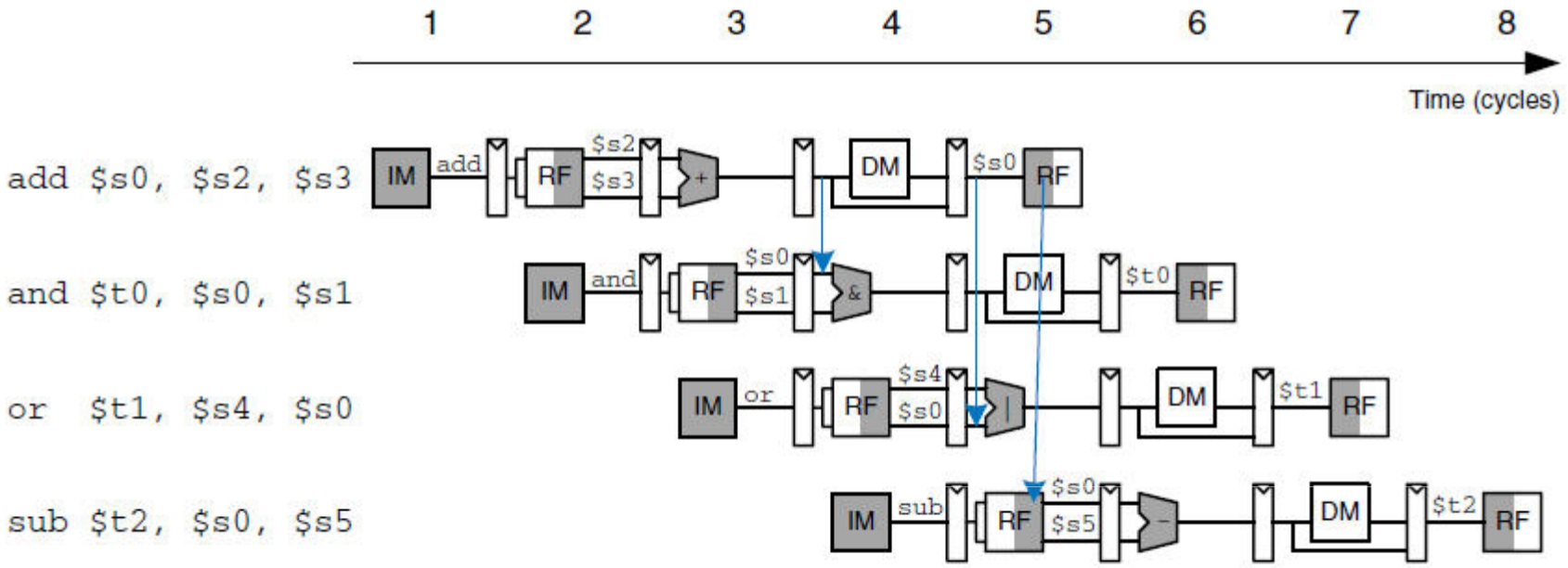


Vznik datových hazardů



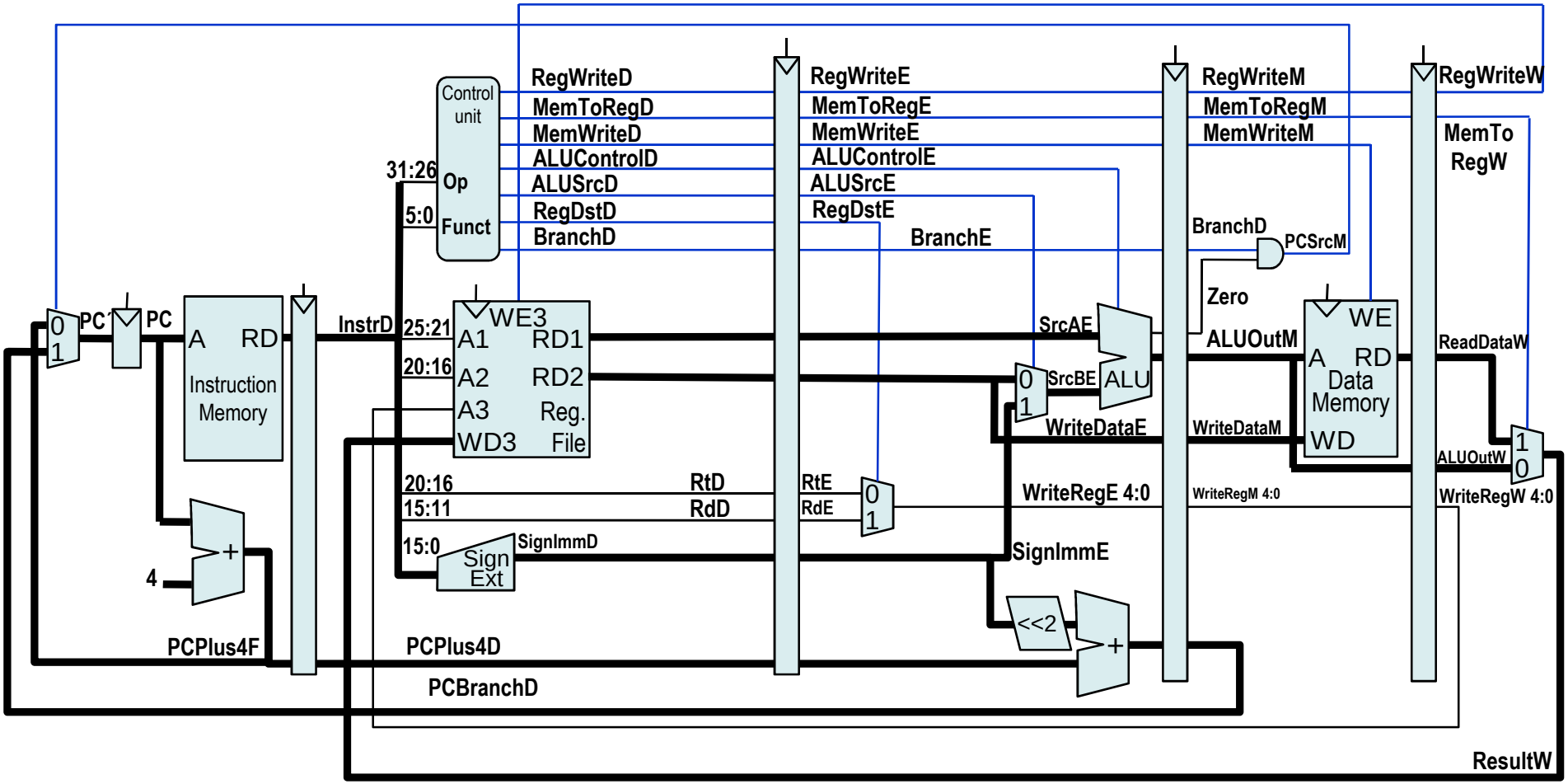
- Pracovní registry (Register File) – přístup v dvou fázích (Decode, WriteBack) – zápis v první polovině cyklu, čtení ve druhé..
- RAW hazard…
- Jak je možné řešit tento hazard a nedegradovat výkon pipeline?

Řešení datových hazardů přeposíláním (forwarding)

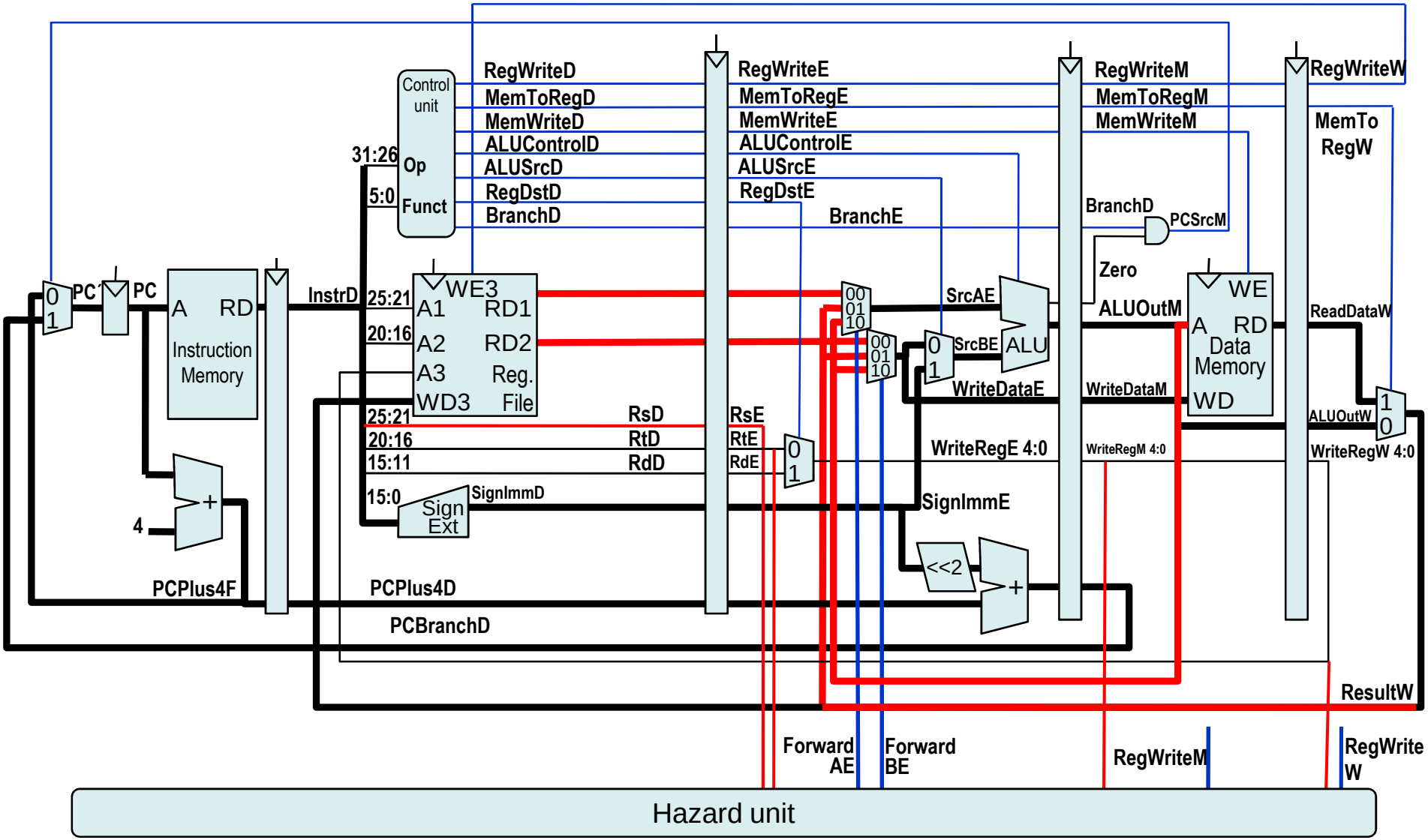


- Pokud výsledek vzniká dřív než jej následující instrukce skutečně potřebují je možné tento hazard řešit přeposíláním (forwarding)
- nastává když se použité zdrojové registry instrukce ve stupni E shodují s cílovým registrem ve stupni M nebo WB
- proto musejí být čísla těchto registrů z těchto stupňů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

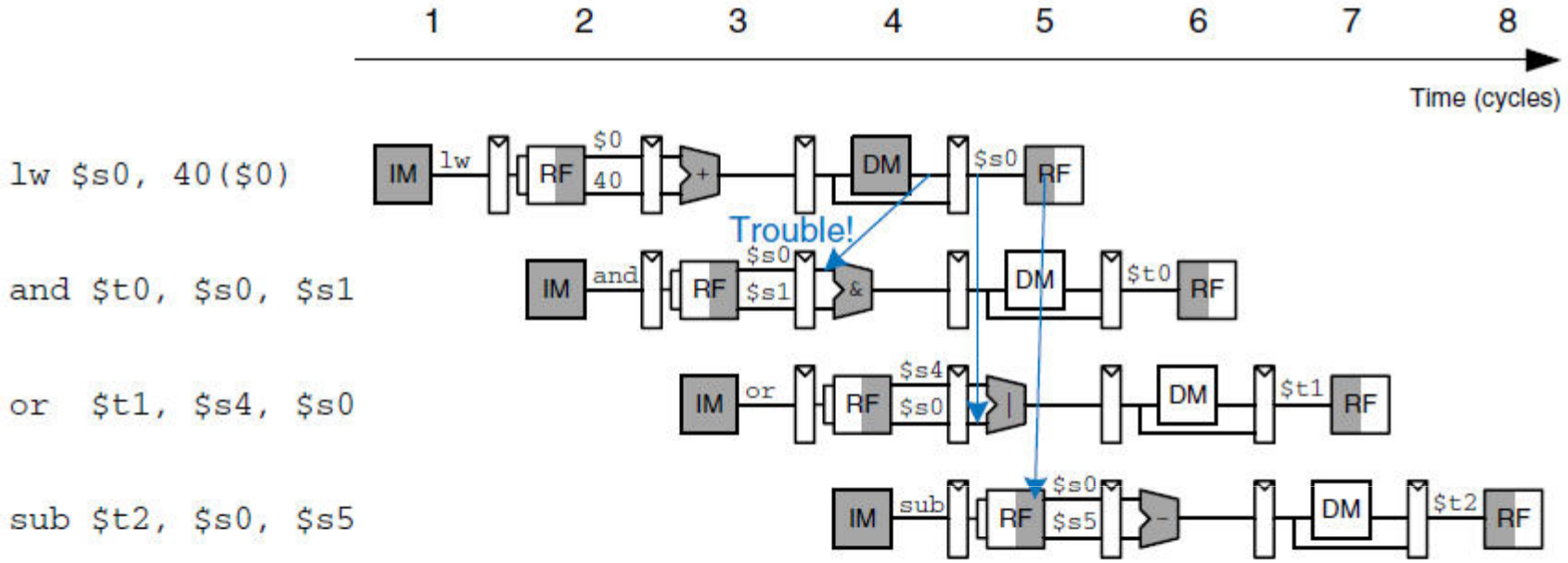
Stávající procesor



Řešení datových hazardů přeposíláním (forwarding)

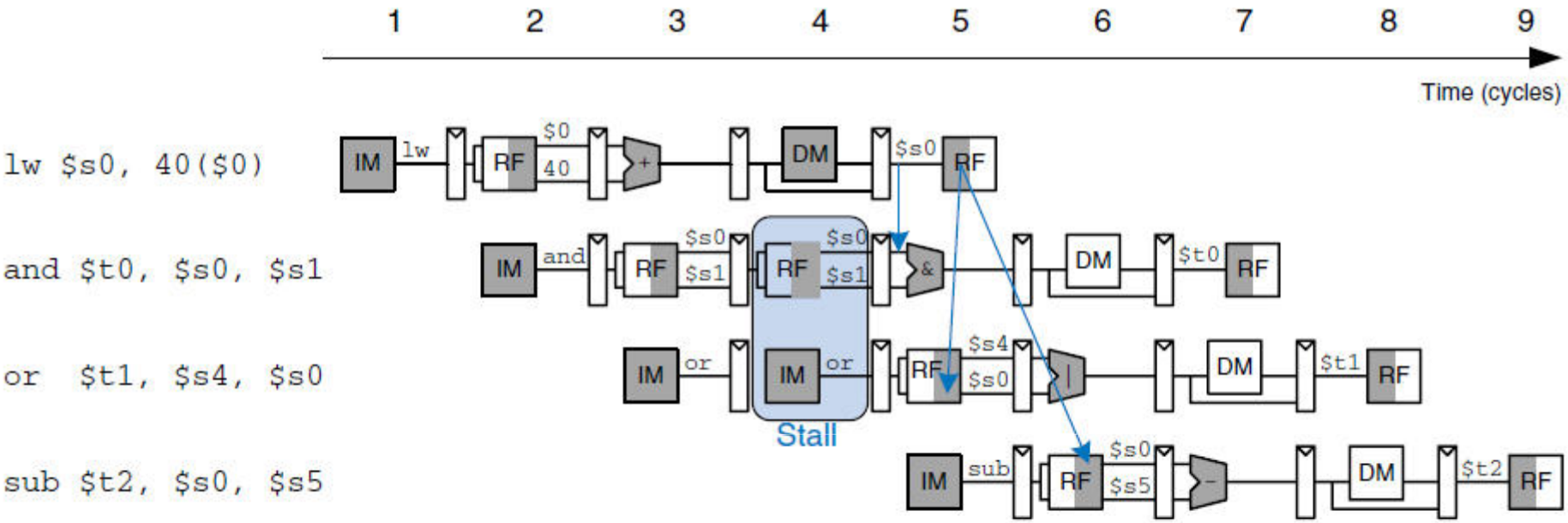


Řešení datových hazardů pozastavením (stall)



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

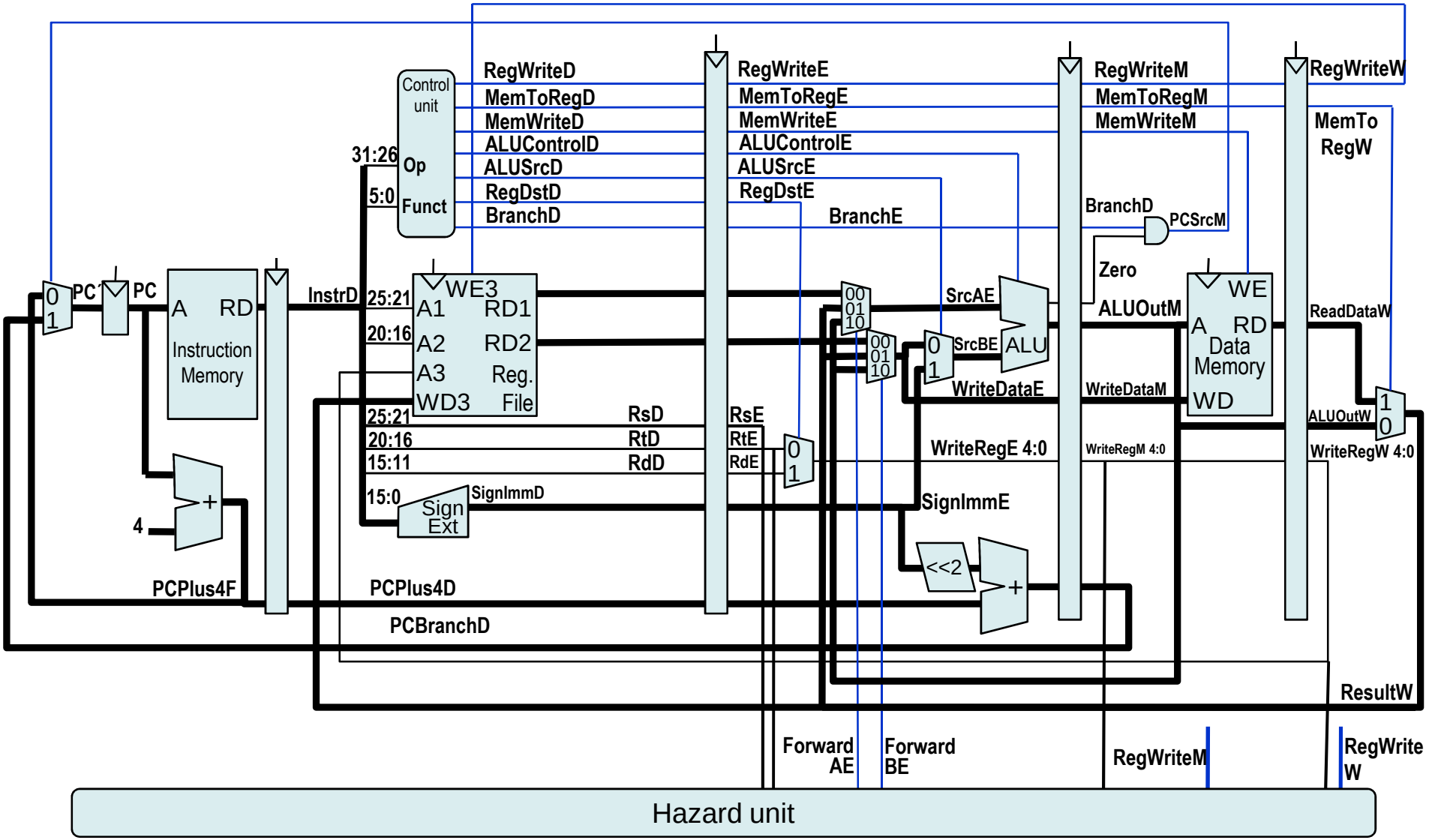
Řešení datových hazardů pozastavením (stall)



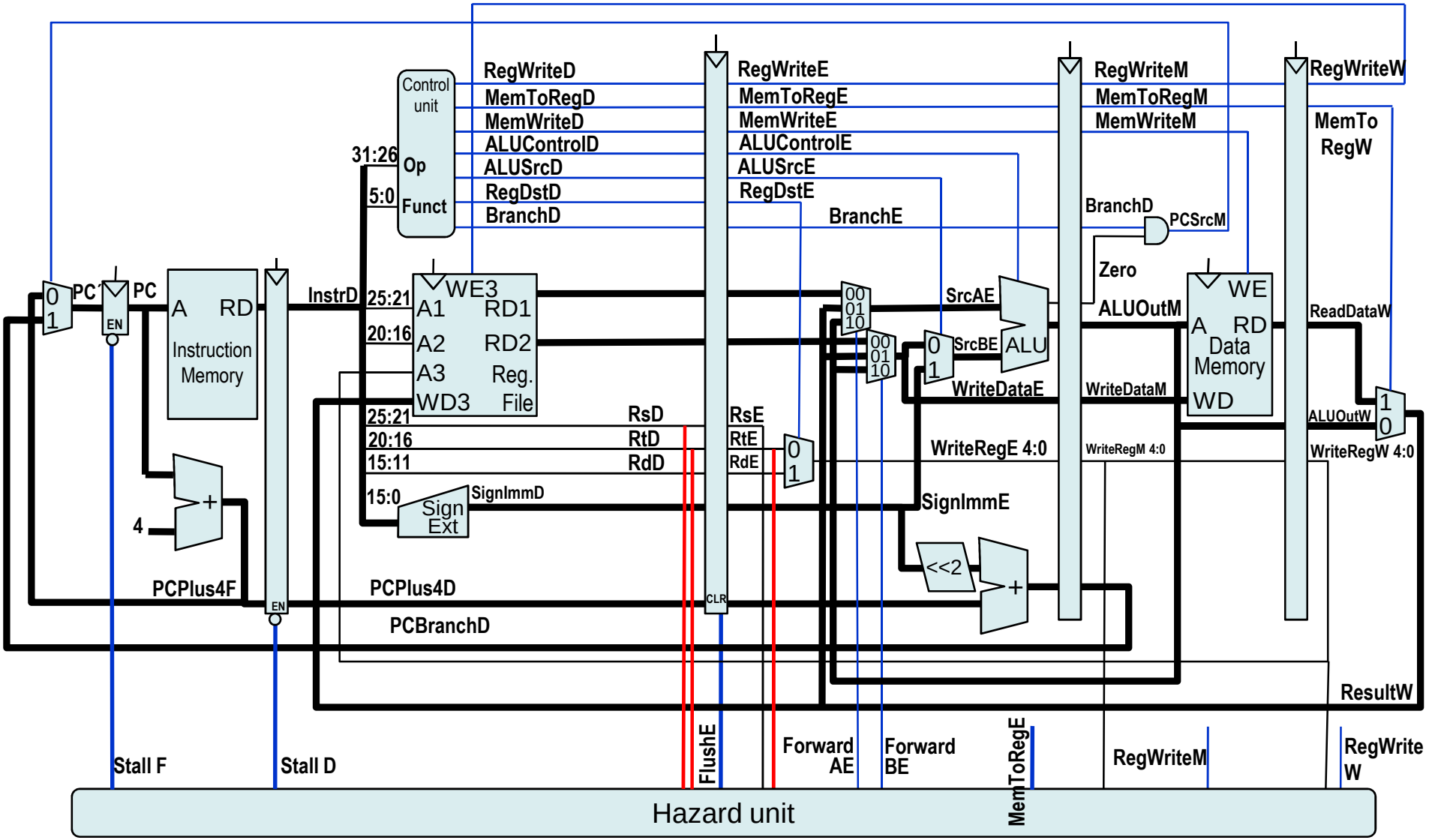
- pozastavení se dosáhne podržením hodnoty mezistupňových registrů
- výsledky z kolizního stupně se musejí „ztratit“ – řídicí signály umožňující měnit stav (kontext) procesoru (zápis pracovních registrů nebo do paměti, řízení povolení větvení) se nulují
- obojí se dosáhne přidáním řídicích vodičů k mezistupňovým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

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

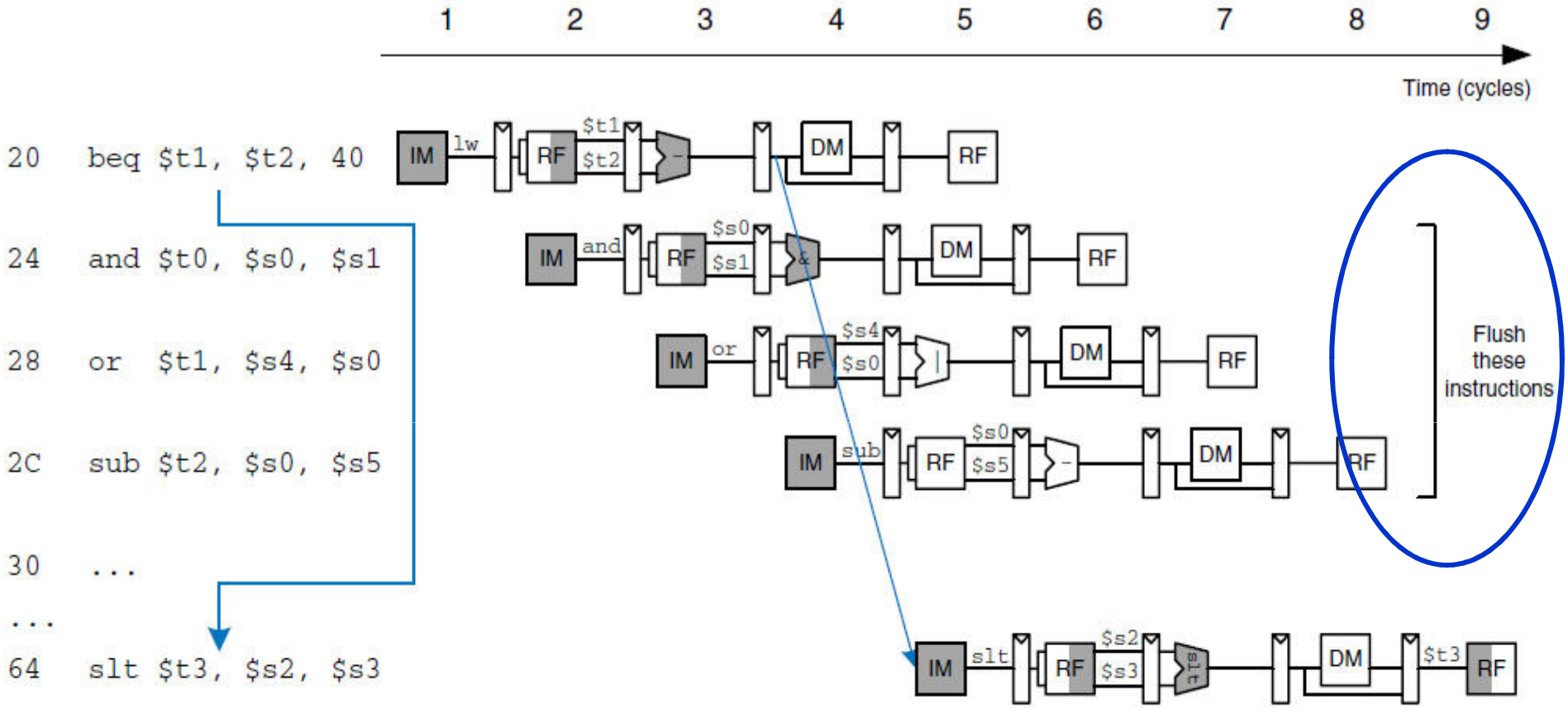
Stávající procesor



Řešení datových hazardů pozastavením (stall)

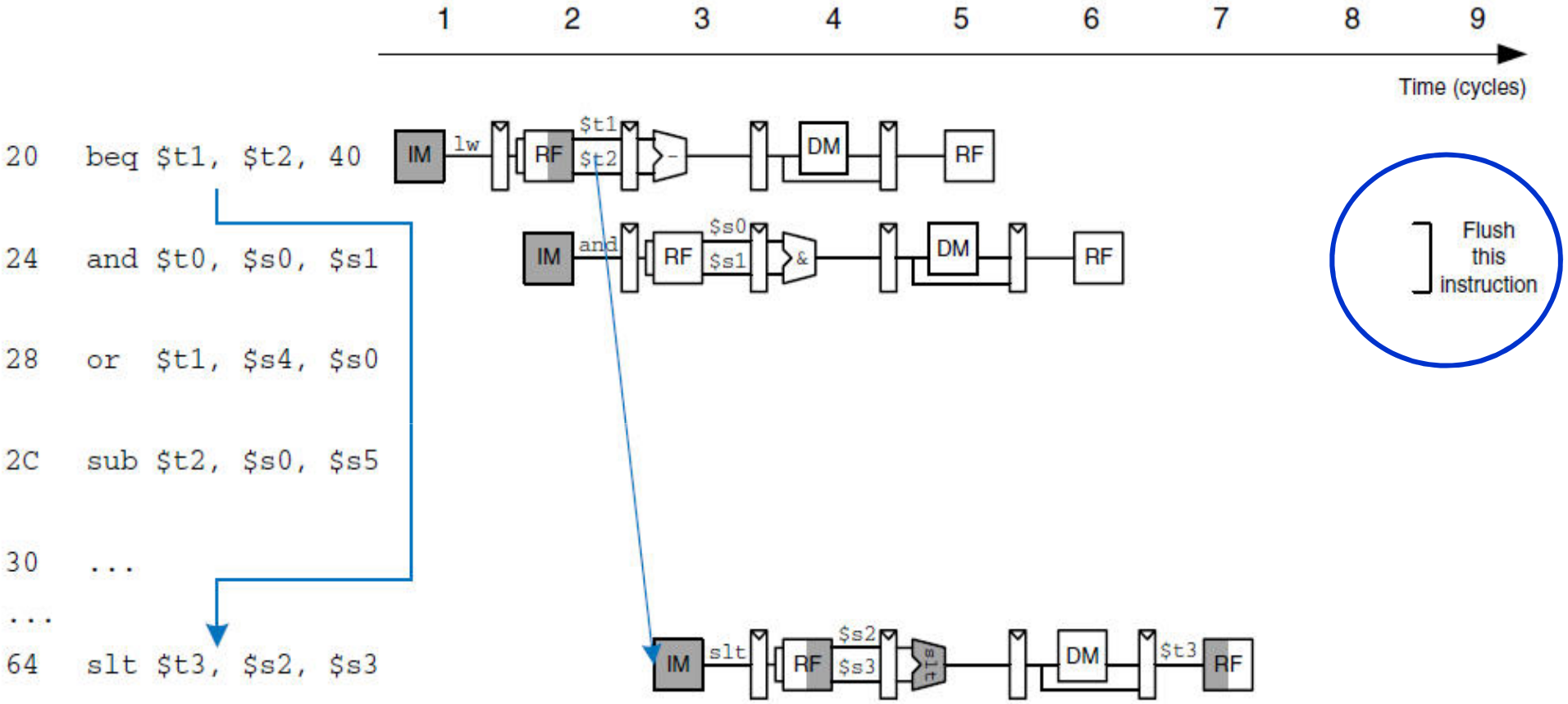


Řídicí hazardy



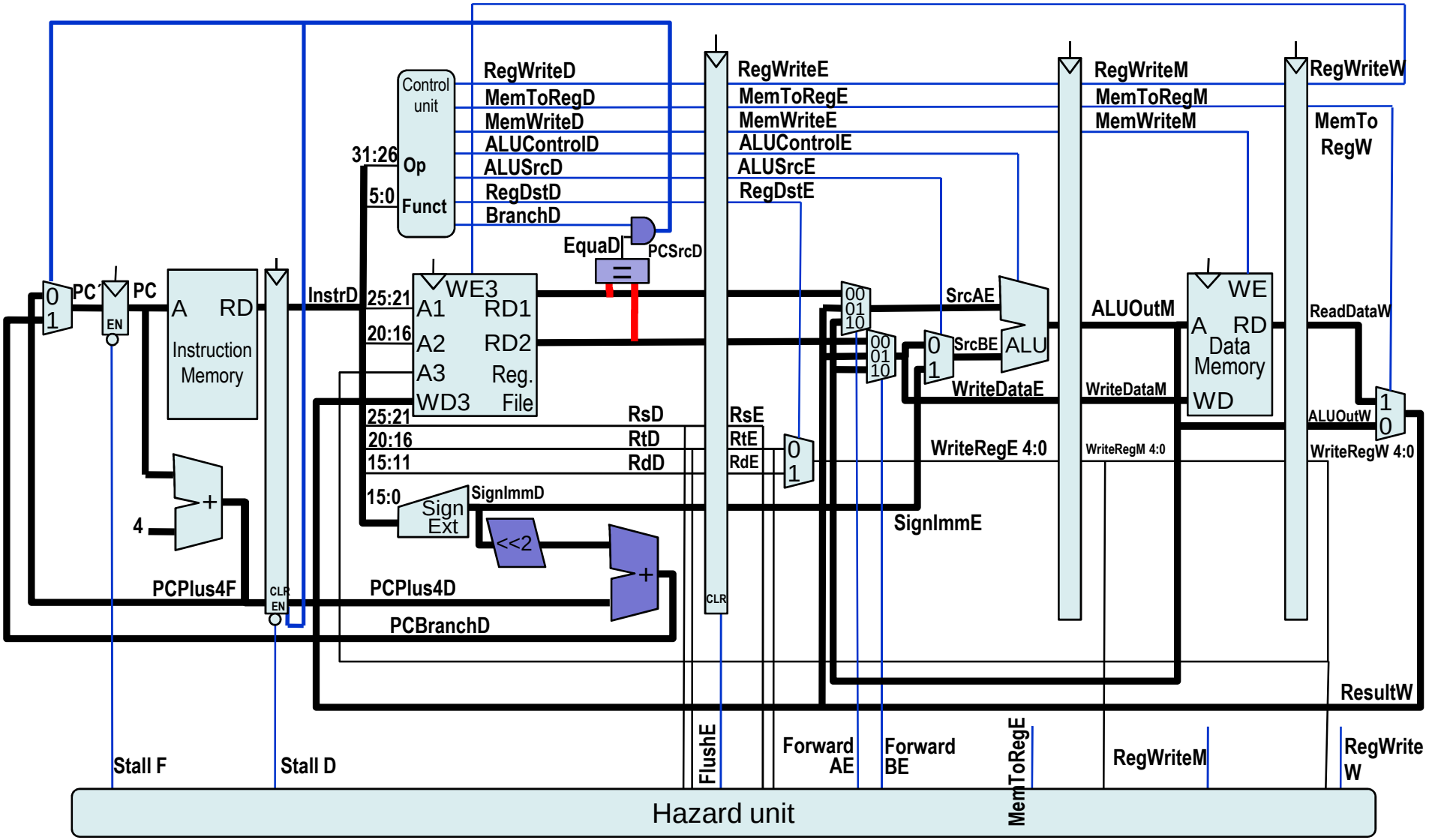
- výsledek porovnání je znám až v 4. cyklu.. Proč?

Řídicí hazardy – raději znát výsledek dříve..

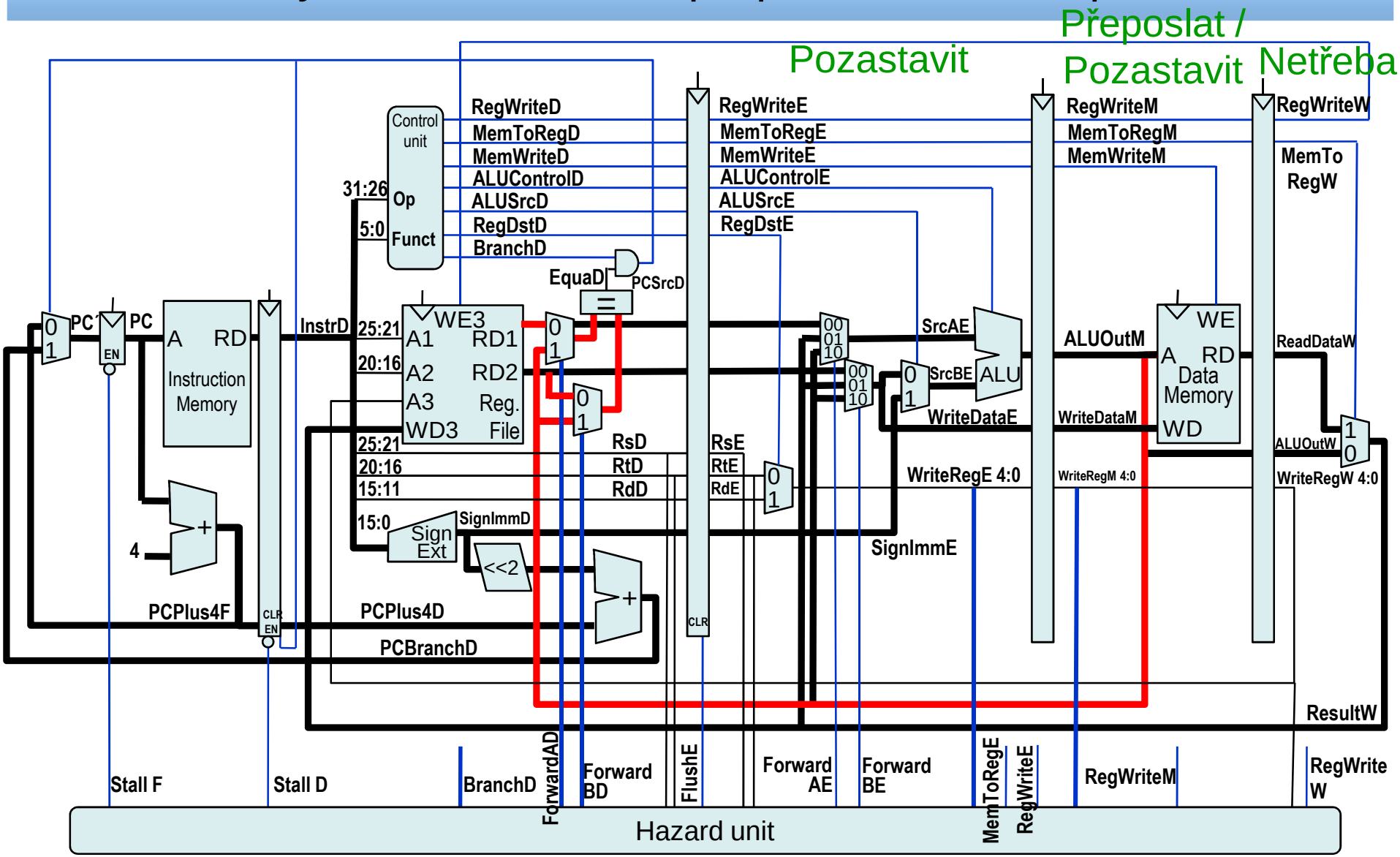


- pokud dokážeme stanovit výsledek porovnání už v 2. cyklu můžeme redukovat tzv. „misprediction penalty“
- přesun rozhodování dopředu může zavést nové RAW hazardy..

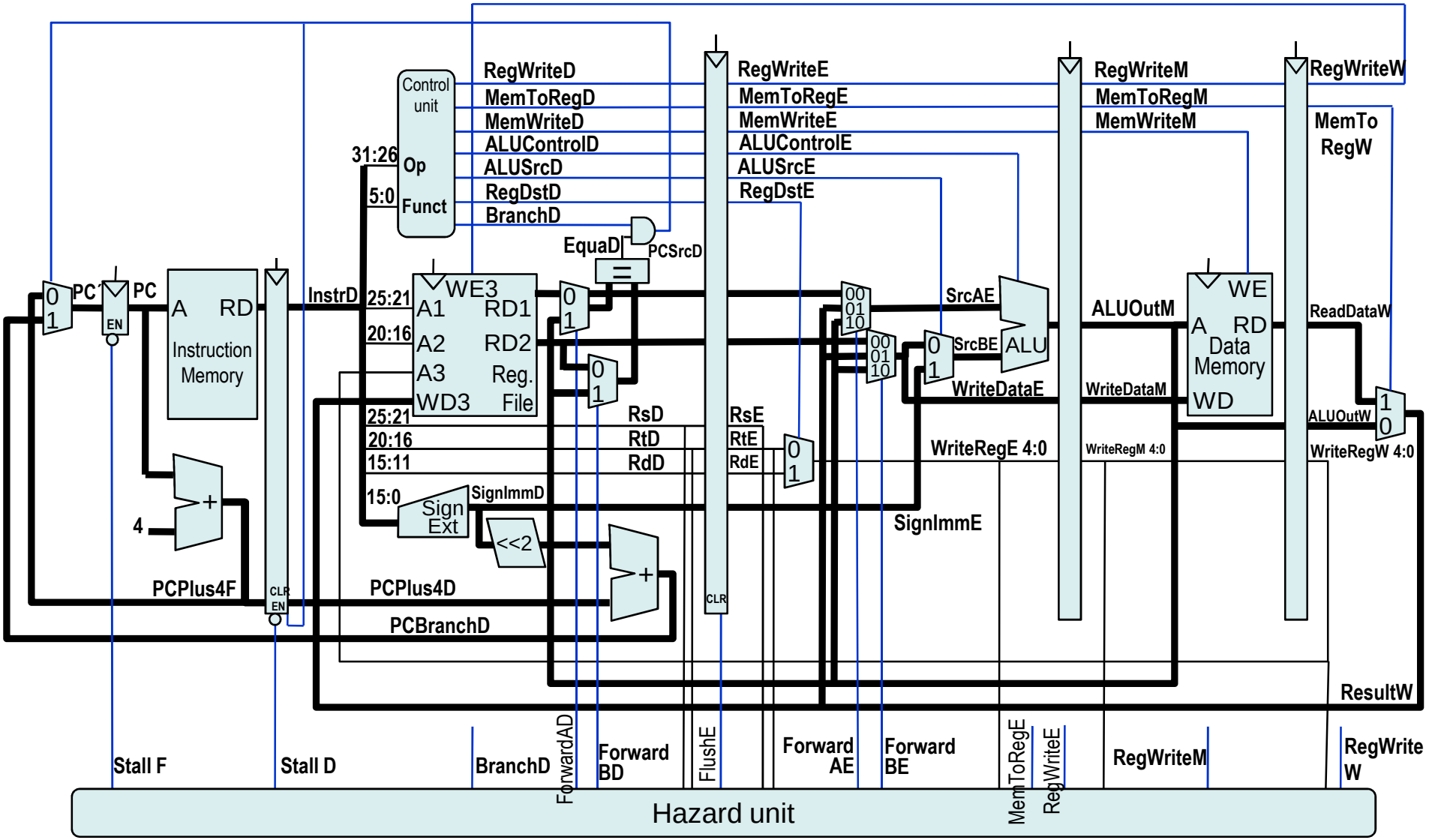
Řešení řídicích hazardů vyprázdněním (flush)



Řešení vzniklých RAW hazardů přeposíláním nebo pozastavením



Hotovo – navržený zřetězený procesor



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

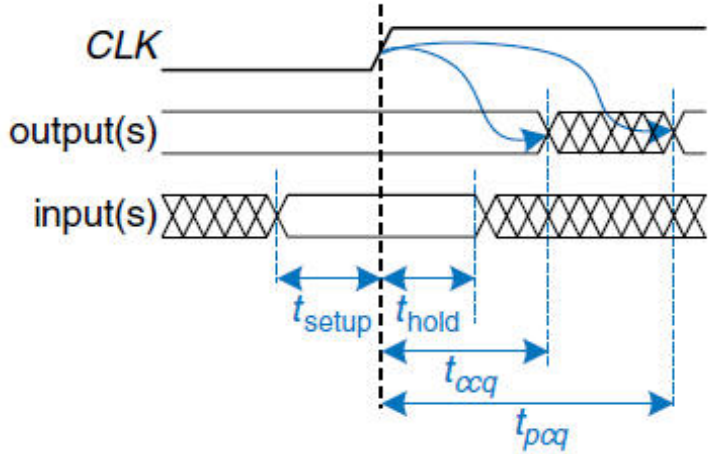
- Jaká může být maximální frekvence procesoru?
- Který stupeň je nejpomalejší?
- Dobu cyklu určuje nejpomalejší stupeň
- V našem případě:
  Tc = 300 ns --> 3 333 kHz

  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)

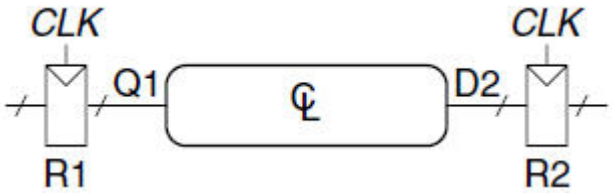
Zřetězený procesor – časování

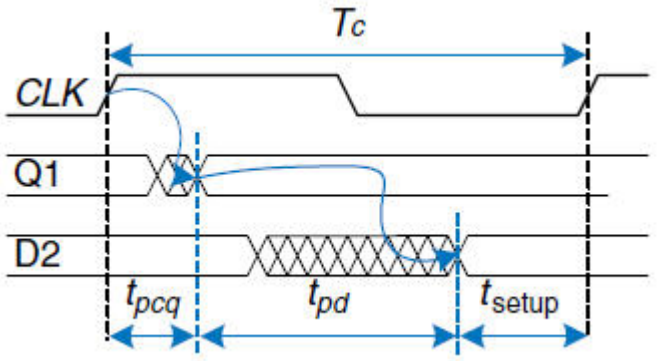
- Specifikace časování pro synchronní sekvenční obvod:



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

- Omezující podmínka na předstih signálu před hodinami:

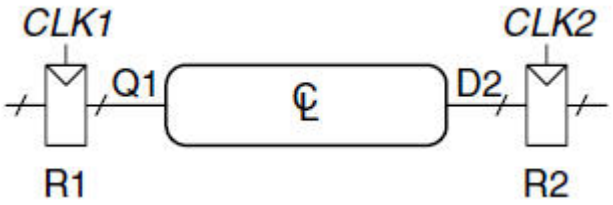


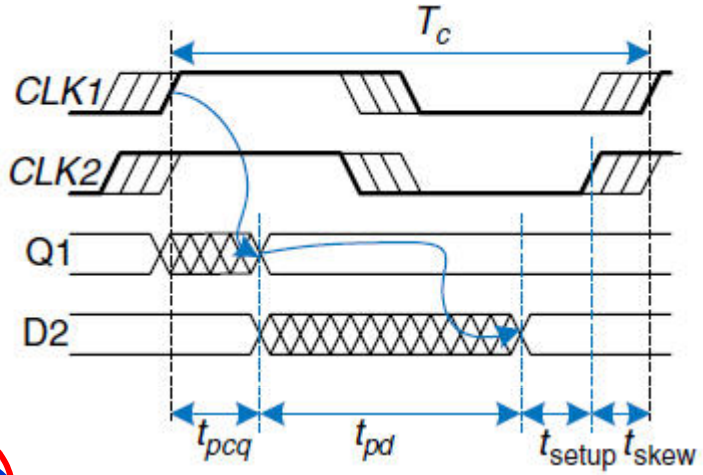


$$Tc >= t_{pcq} + t_{pd} + t_{setup}$$

Zřetězený procesor – časování

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



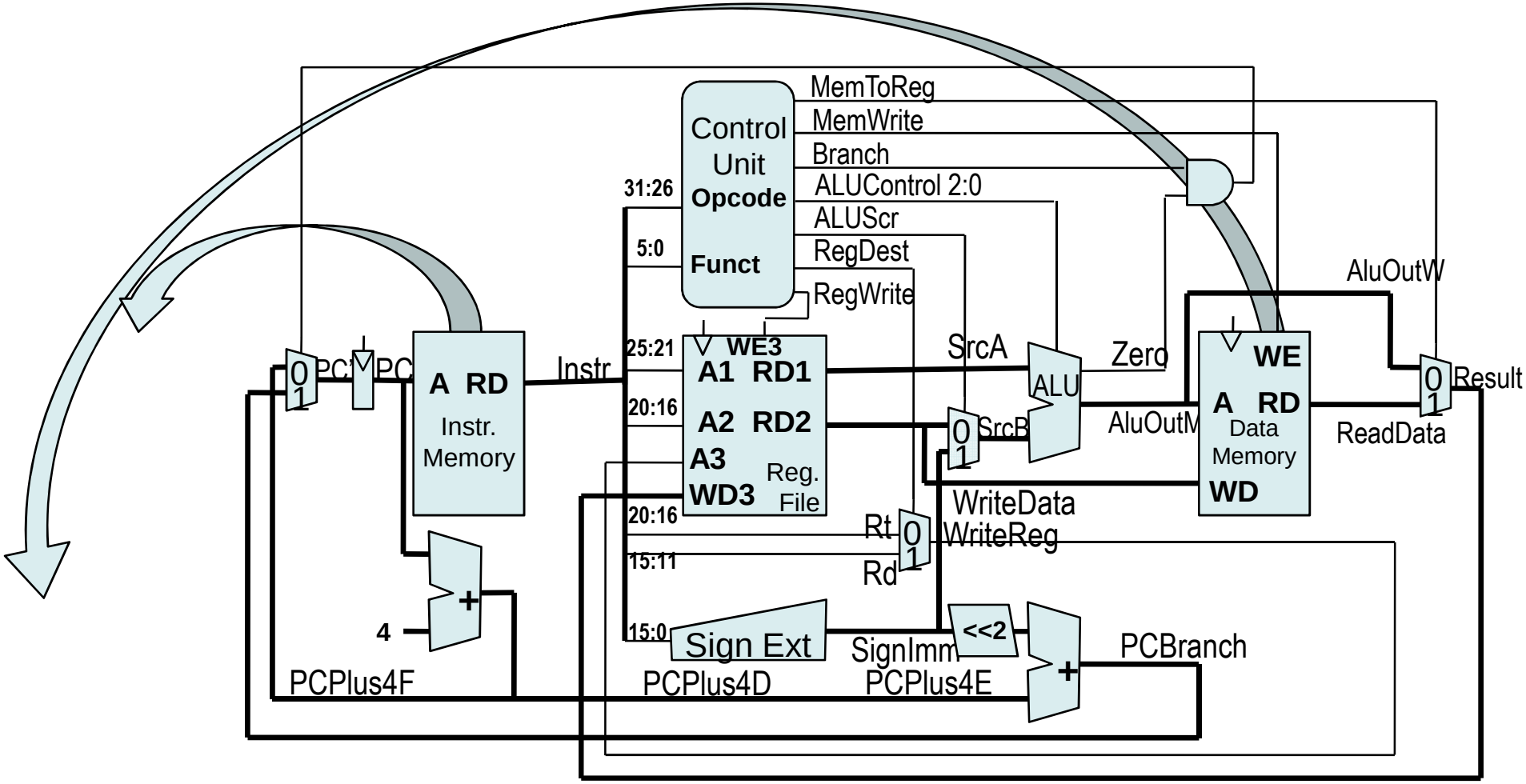


$$Tc >= t_{pcq} + t_{pd} + t_{setup} + t_{skew}$$

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

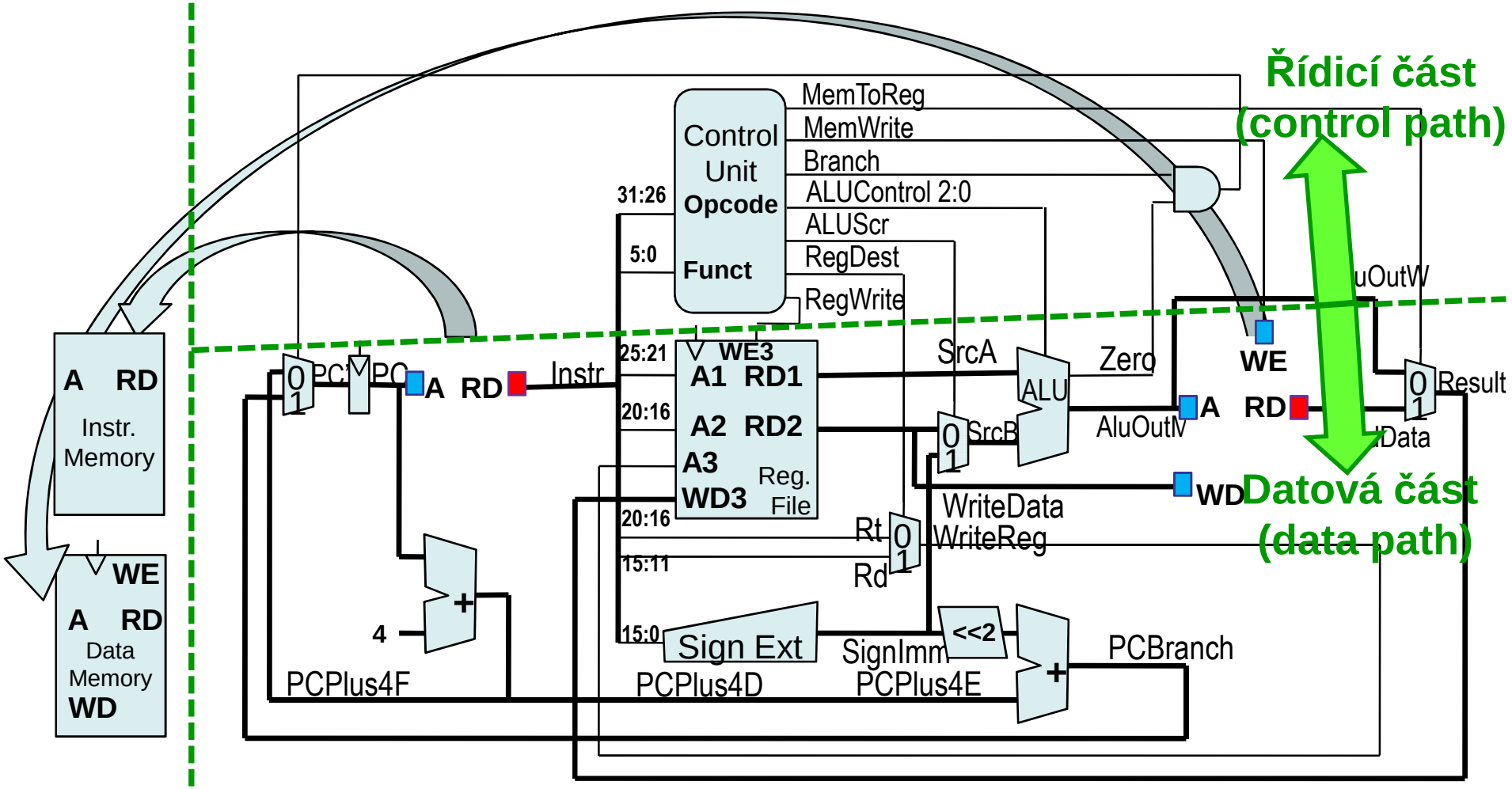
Co jsme navrhli?

Návrat k nezřetězenému procesoru

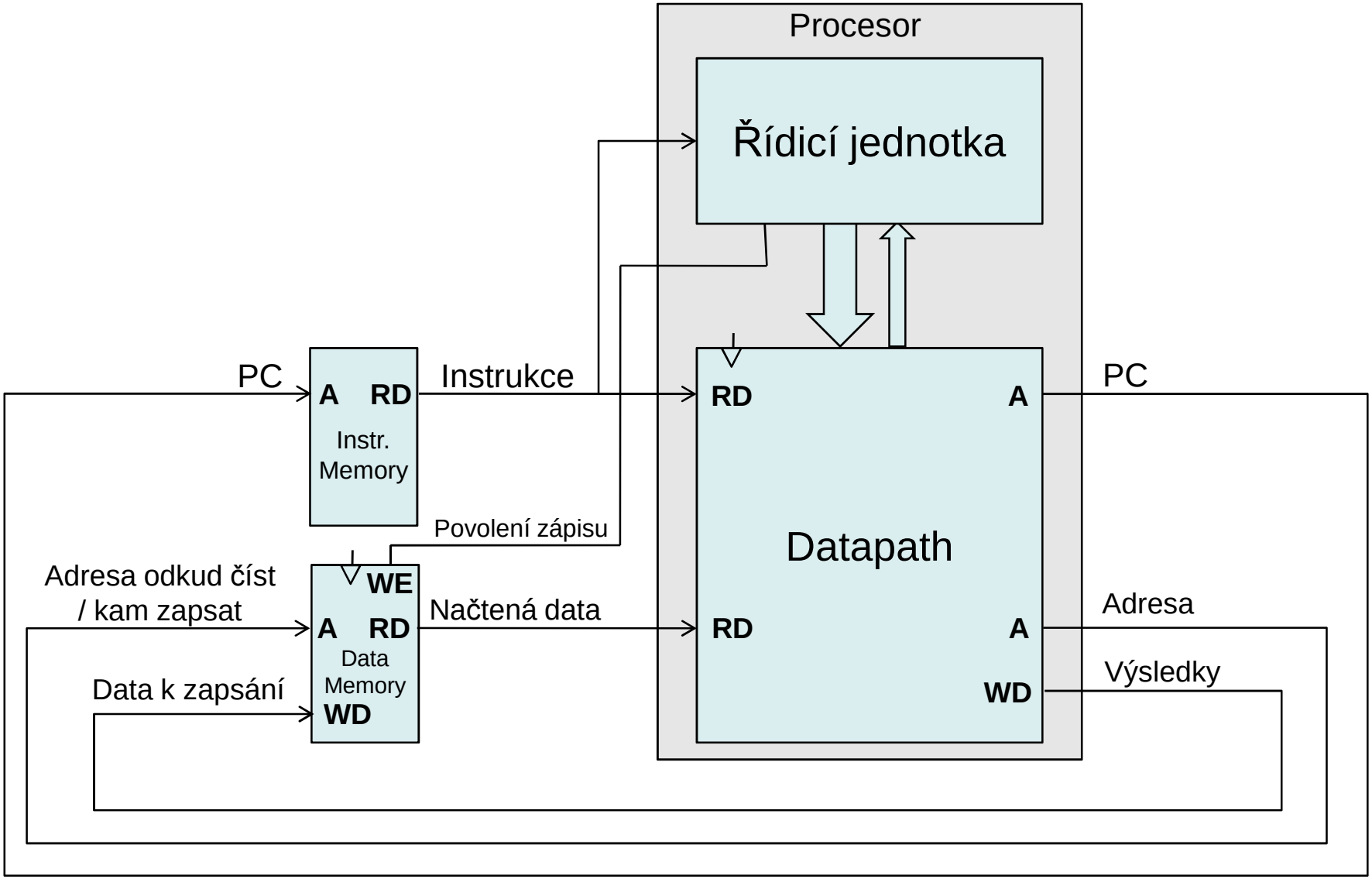


Co jsme navrhli?

Návrat k nezřetězenému procesoru



Co jsme navrhli?

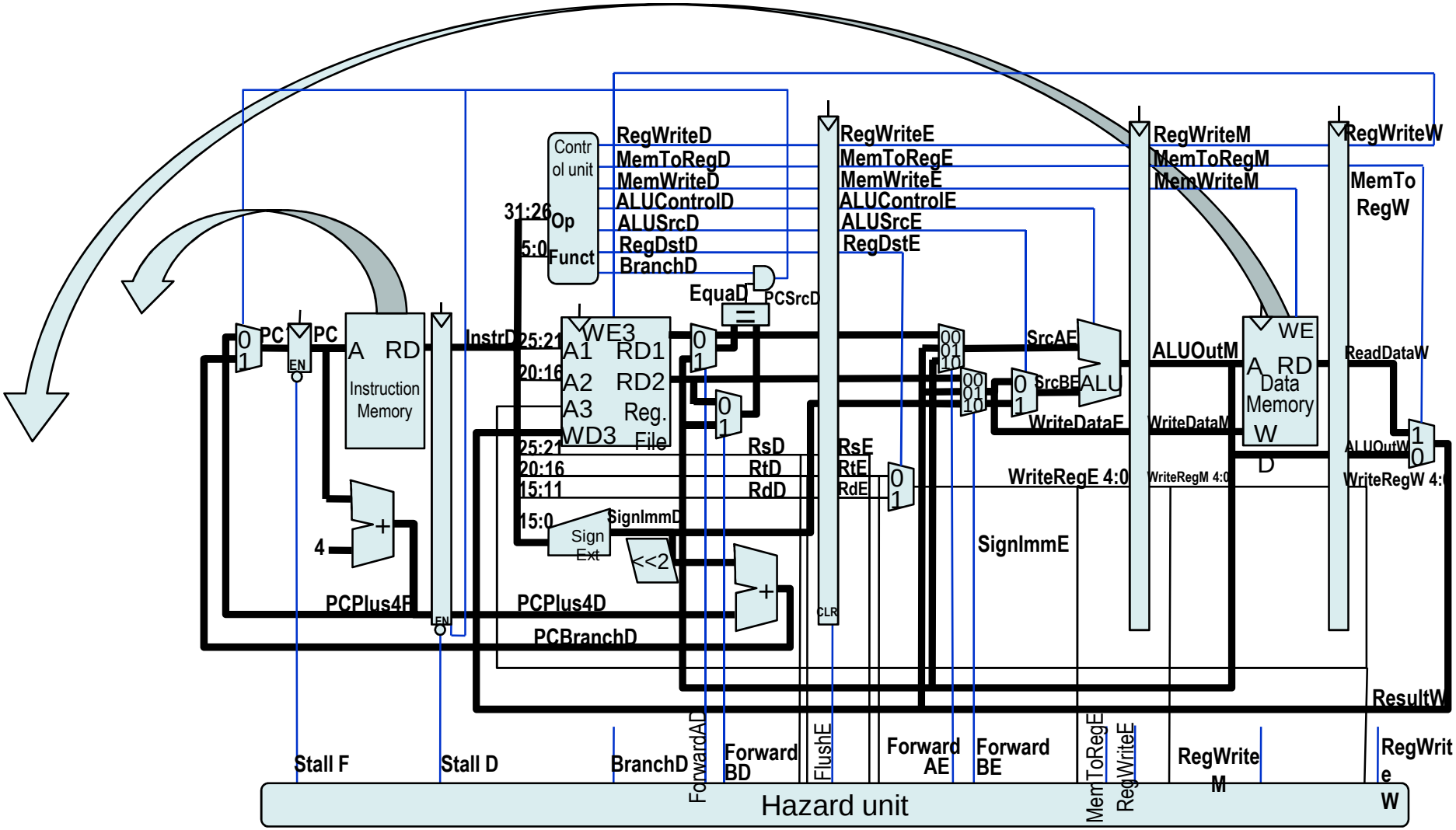


Co s paměťmi pokud jsou velké?

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

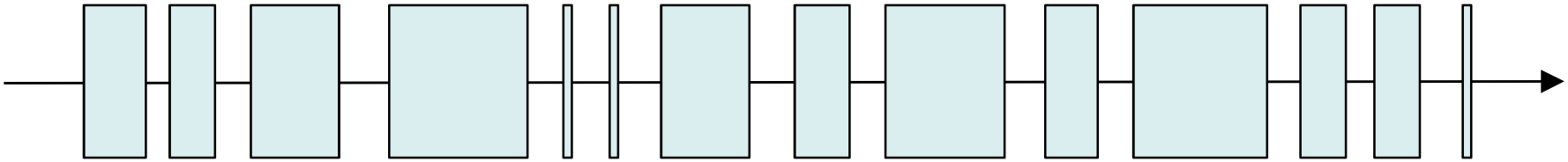
  Jak nedegradovat frekvenci procesoru??
- Řešení: Víceúrovňový paměťový systém
  (podrobněji o této problematice v 10. a 11. přednášce)

Co jsme navrhli? – zřetězená verze



Vyvažování stupňů zřetězení

Lineární zřetězení:

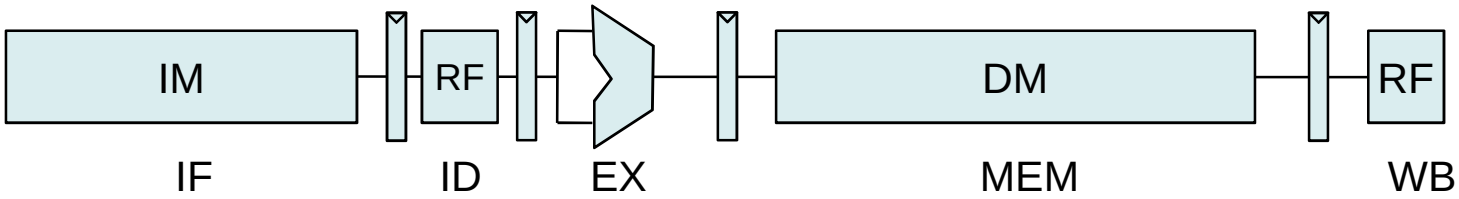


(též: stromový sumátor, stromová násobička, iterační dělička..)

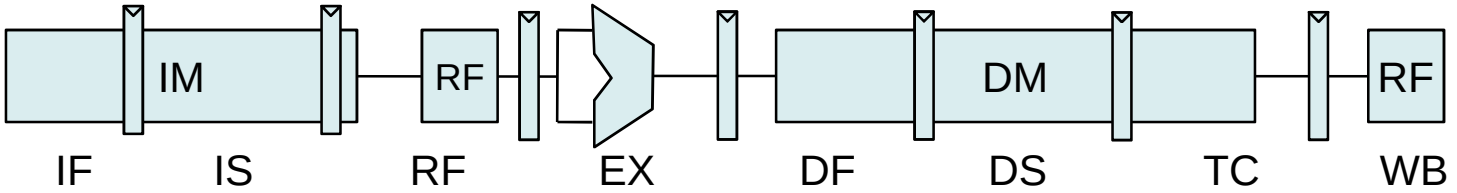
- **Vyvažování**: cílem je rozdělit jednotlivé bloky do *N* stupňů tak, aby ve zpoždění ve všech stupních bylo pokud možno stejné…
- Volba počtu stupňů závisí od preference: propustnost vs. latence

Superzřetězení

- nevyvážené 5-stupňové zřetězení:



- hlubší zřetězení vzniklá další dekompozicí



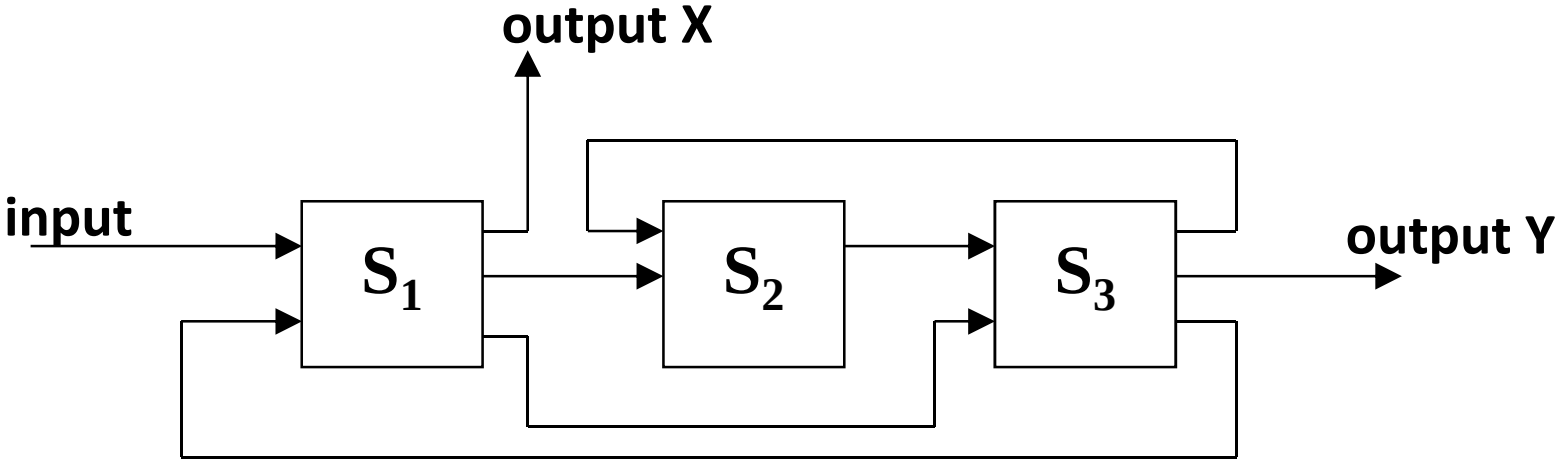
- přináší možnost dalšího zvýšení pracovní frekvence, avšak také řadu dalších problémů
- další forwarding, nárůst pozastavení pipeline, hazardy

Nelineární zřetězení - obecně

- Nelineární zřetězení: kaskáda zpracovatelských stupňů (segmentů), které jsou nelineárně spojené (dopředně a zpětně) za účelem vykonání různých funkcí v rozličných časech
- Vstupný proud údajů postupně prochází segmenty, přičemž některými z nich může (podle tabulky rezervace) procházet vícenásobně.
- Tabulka rezervace zřetězení zřetězené jednotky se zpětnou vazbou zobrazuje proces zřetězení určený obvykle na realizaci rekurentních funkcí resp. na procesy dynamického zřetězení.

Nelineární zřetězení - obecně

- Příklad – multifunkční dynamické zřetězení



- Tabulka rezervace pro funkci X a pro funkci Y:

	čas → 1	2	3	4	5	6	7	8
S1	X					X		X
S2		X		X				
S3			X		X		X	

	1	2	3	4	5	6
S1	Y				Y	
S2			Y			
S3		Y		Y		Y

Nelineární zřetězení - obecně

Rezervační tabulky:

- pro statický lineární pipeline – tabulka zobrazuje triviální proces zřetězení, při kterém vstupný proud údajů postupně prochází všemi segmenty
- pro dynamický pipeline – násobné rezervační tabulky pro vyhodnocování různých funkcí (kupř. tabulka pro X a tabulka pro Y)

Počet sloupců v rezervační tabulce se nazývá vyhodnocovací čas dané funkce (kupř. pro X je to 8 hodinových cyklů, pro Y je to 6 cyklů)

Začátek procesu zřetězeného zpracování vstupného proudu údajů v tabulce rozkladu zřetězení (rezervační tabulce) se nazývá inicializace zřetězení.

Počet jednotlivých časových intervalů (kroků) mezi počátky dvou za sebou následujících inicializací vyjadřuje latentnost zřetězení (latenci). Jinak řečeno, latence *k* znamená, že dvě inicializace jsou separované *k* cykly.

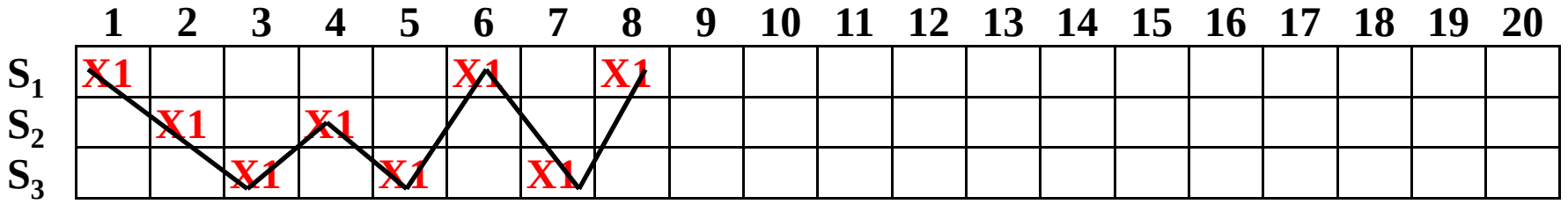
Nelineární zřetězení - obecně

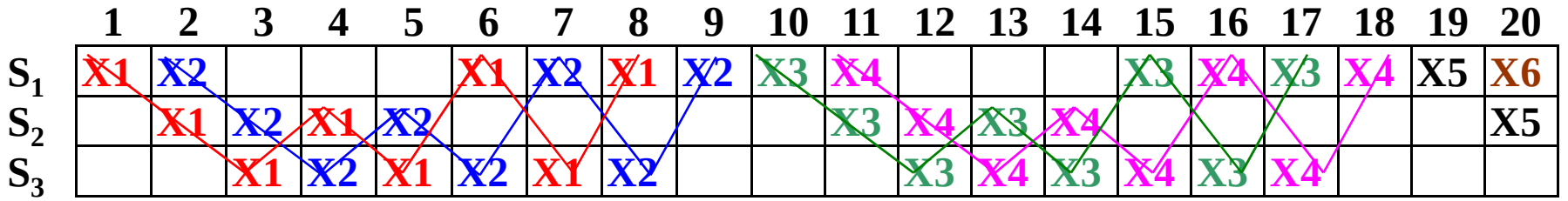
Analýza latence zřetězení – bezkolizní rozvrhovaní:

- Kolize je pokus dvou anebo více inicializací použit ten samý stupeň zřetězení v tom samém čase.
- Rozvrhovaní (Scheduling) musí vyloučit všechny kolize volbou postupnosti inicializací zřetězení.
- Zakázané latence jsou ty, které způsobují kolize.
- Hlavní cíl rozvrhovaní je **nalézt nejkratší možnou střední latenci mezi inicializacemi bez způsobení kolizí**.

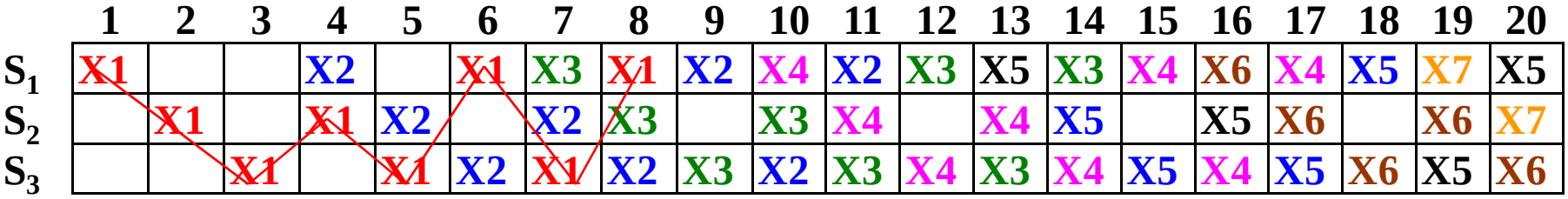
Nelineární zřetězení - obecně

Příklad:





Latency cycle (1,8) = 1,8,1,8,1,8,… střední hodnota: 4,5



Latency cycle (3) = 3,3,3,3,… střední hodnota: 3

Nelineární zřetězení - obecně

- Postupnost latencí je postupnost dovolených nezakázaných latencí mezi po sobě následujícími inicializacemi.
- Cyklus latencí je postupnost latencí, která nekonečně opakuje tu samou subpostupnost (cyklus).
- Nejvýznamnější úloha strategie řízení procesu zřetězení spočívá v určení okamihů nových inicializací funkce zřetězení. Systematické řešení této úlohy se uskutečňuje pomocí **stavového grafu inicializací**, který vyjadřuje průběžné stavy zřetězeného systému při nových inicializacích. Vrcholy grafu zobrazují v daném časovém okamihu všechny možné následujíce inicializace, přičemž jeho hrany zobrazují počet kroků mezi dvěma bezkonfliktními inicializacemi v čase t a t+1.