B35APO: Architektury počítačů

Lekce 05. Zřetězené zpracování
Pipelining

Pavel Píša
pisa@fel.cvut.cz

Petr Štěpán
stepan@fel.cvut.cz

18. dubna, 2025

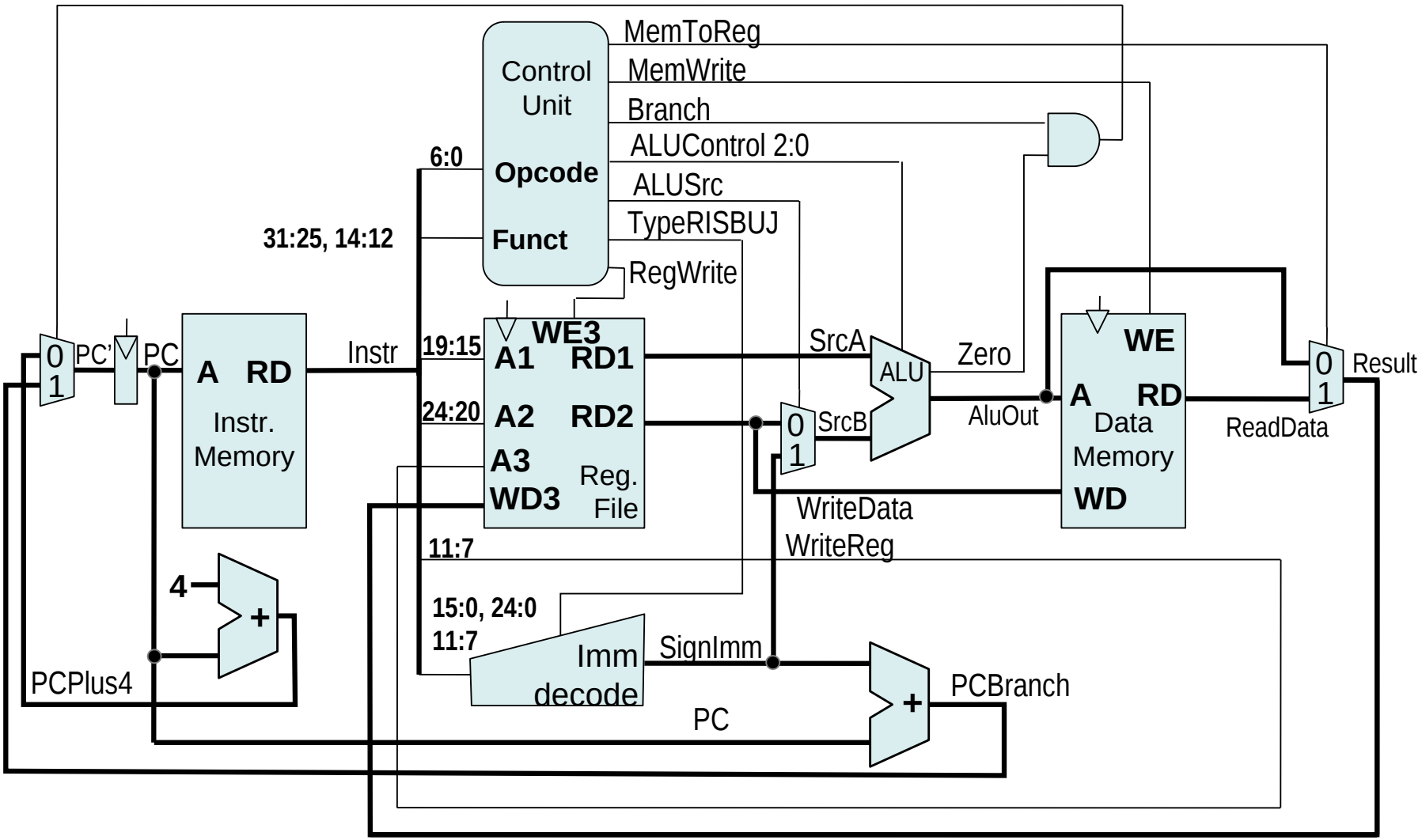
Obsah

Cíl dnešní přednášky

- Navrhnout úpravy procesoru z lekce 3 tak, aby využíval zřetězené zpracování (pipelining) a tím ho bylo možné zrychlit
- Stále uvažujeme následující instrukce: `add`, `sub`, `and`, `or`, `slt`, `addi`, `lw`, `sw`, a `beq`
- Kódování instrukcí také zůstává

Typ	31	30...25	24...21	20	19...15	14...12	11...8	7	6...0
R	fnct7		rs2		rs1	fnct3	rd		opcode
I	imm[11:0]				rs1	fnct3	rd		opcode
S	imm[11:5]		rs2		rs1	fnct3	imm[4:0]		opcode
B	imm [12]	imm [10:5]	rs2		rs1	fnct3	imm[4:1]	imm [11]	opcode
U	imm[31:12]						rd		opcode
J	imm [20]	imm[10:1]		imm [11]	imm[19:12]		rd		opcode

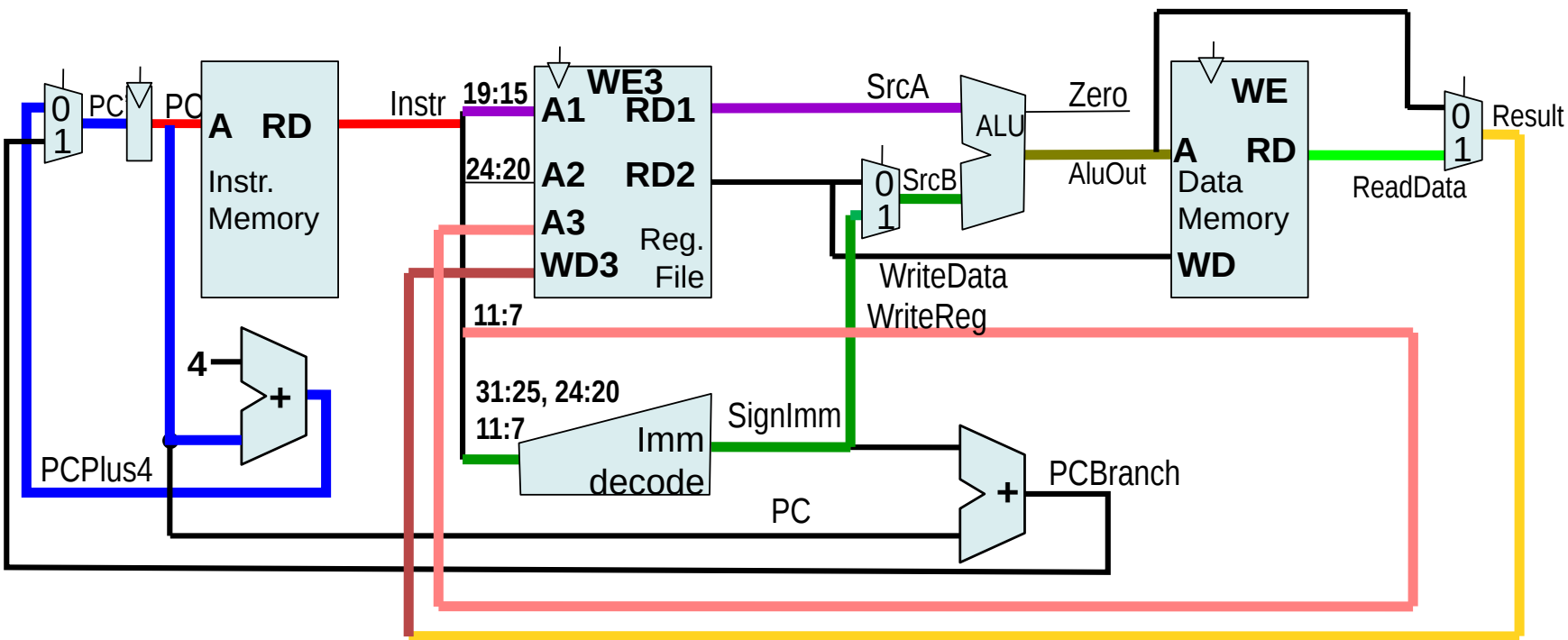
Jednocyklový procesor s pamětí (z přednášky 3)



Propustnost jednocyklového procesoru je omezená

- Maximum instrukcí za sekundu $IPS = IC/T = IPC_{avk} \cdot f_{clk}$
- Omezené nejdelší dobou průchodu signálu od zapamatované hodnoty na vstup zápisu do paměti, registru (critical path)
- V našem návrhu omezení z kritické cesty instrukce `lw`

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



Kritická cesta omezující propustnost

$f_{clk} = 1/T_C$ kde T_C je perioda hodin/čas na zpracování jednoho cyklu

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

Uvažujeme následující zpoždění a časové požadavky

$$
\begin{aligned}
t_{PC} &= 30 \text{ ns} \\
t_{Mem} &= 300 \text{ ns} \\
t_{RFread} &= 150 \text{ ns} \\
t_{ALU} &= 200 \text{ ns} \\
t_{Mux} &= 20 \text{ ns} \\
t_{RFsetup} &= 20 \text{ ns}
\end{aligned}
$$

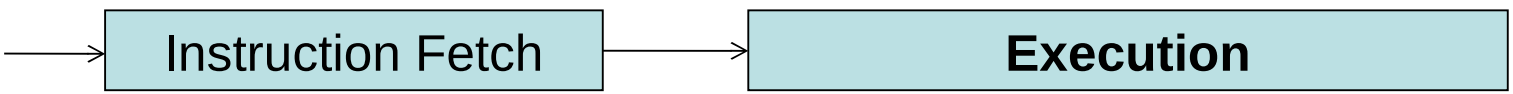
tak $T_C = 1020\,\text{ns} \rightarrow f_{clkmax} = 980\,\text{kHz}$

$IPS = IC/T = IPC_{avg} \cdot f_{clk}$

$IPS = 1 \cdots 980e3 = 980\,000$ instrukcí za sekundu

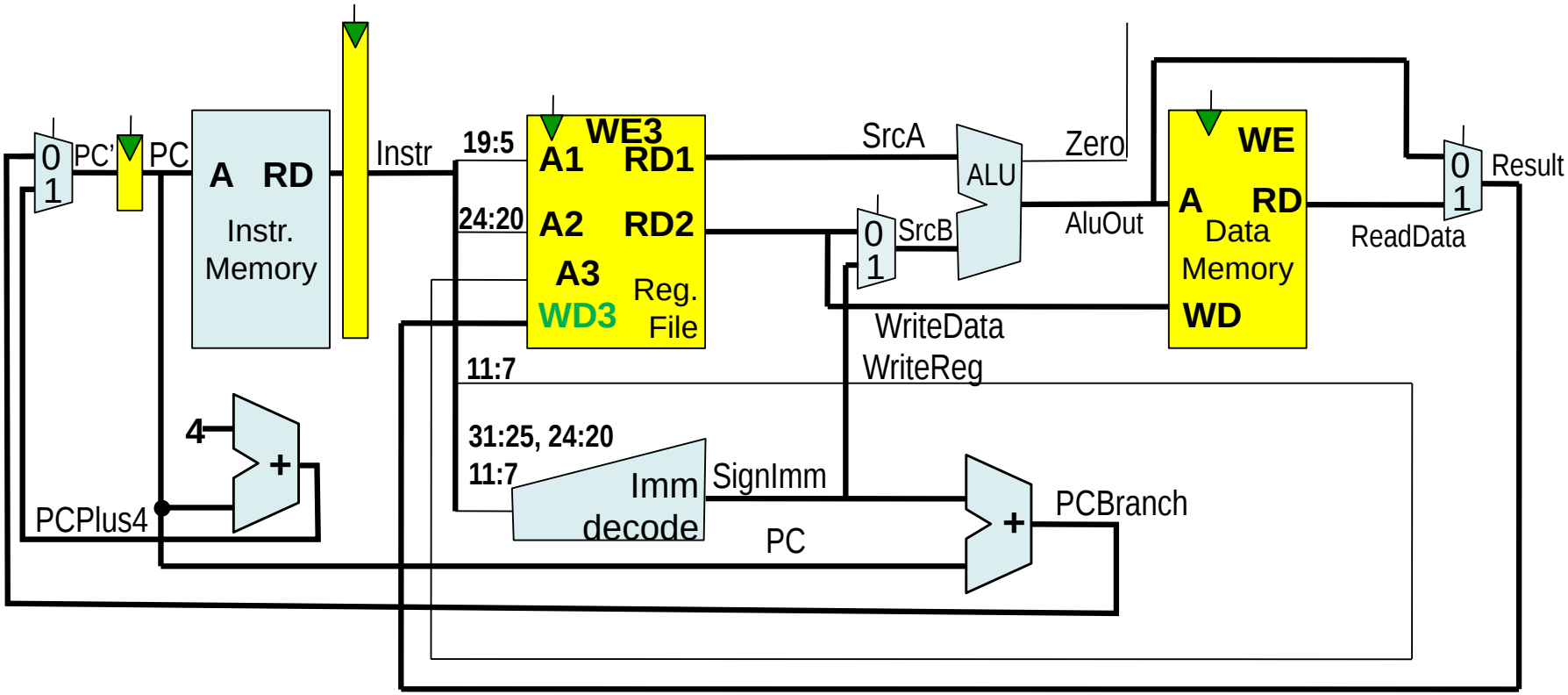
Rozdělení načítání a vykonávání instrukcí

- Načtení instrukce a čtení z paměti ($2 \times t_{Mem}$) představuje obvykle značnou část celkové doby potřebné na cyklus
- Ke zkrácení doby cyklu přispěje, pokud je instrukce načtena již v předchozím cyklu



1. Načtení instrucke (instruction fetch) a zvýšení čítače instrukcí $PC = PC + 4$
2. Vlastní vykonání (execution) instrukce až v následujícím cyklu

Úprava procesoru pro přednačítání instrukcí



- ↓ v obrázku představuje hodinový vstup aktivní na vzestupnou hranu hodinového signálu

Kritická cesta v případě přednačítání

Pro již dříve uvažované parametry

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

- Uvažujeme $T_{C_{fetch}} = t_{PC} + t_{Mem}$ probíhající paralelně s výkonáváním instrukce
- $T_{C_{exec}} = t_{RFread} + t_{ALU} + t_{Mem} + t_{Mux} + t_{RFsetup}$

po dosazení

- Uvažujeme $T_{C_{fetch}} = 30 + 300 = 330\,\text{ns}$
- $T_{C_{exec}} = 150 + 200 + 300 + 20 + 20 = 690\,\text{ns}$
- $T_{C_{fetch}} < T_{C_{exec}}$ tedy $T_C = T_{C_{exec}} = 690\,\text{ns}$
- $\rightarrow f_C = 1.45\,\text{MHz} \rightarrow IPS = 1\,450\,000$

Bez úprav dojde ke zpoždění skokových instrukcí (`beq`), viz poději

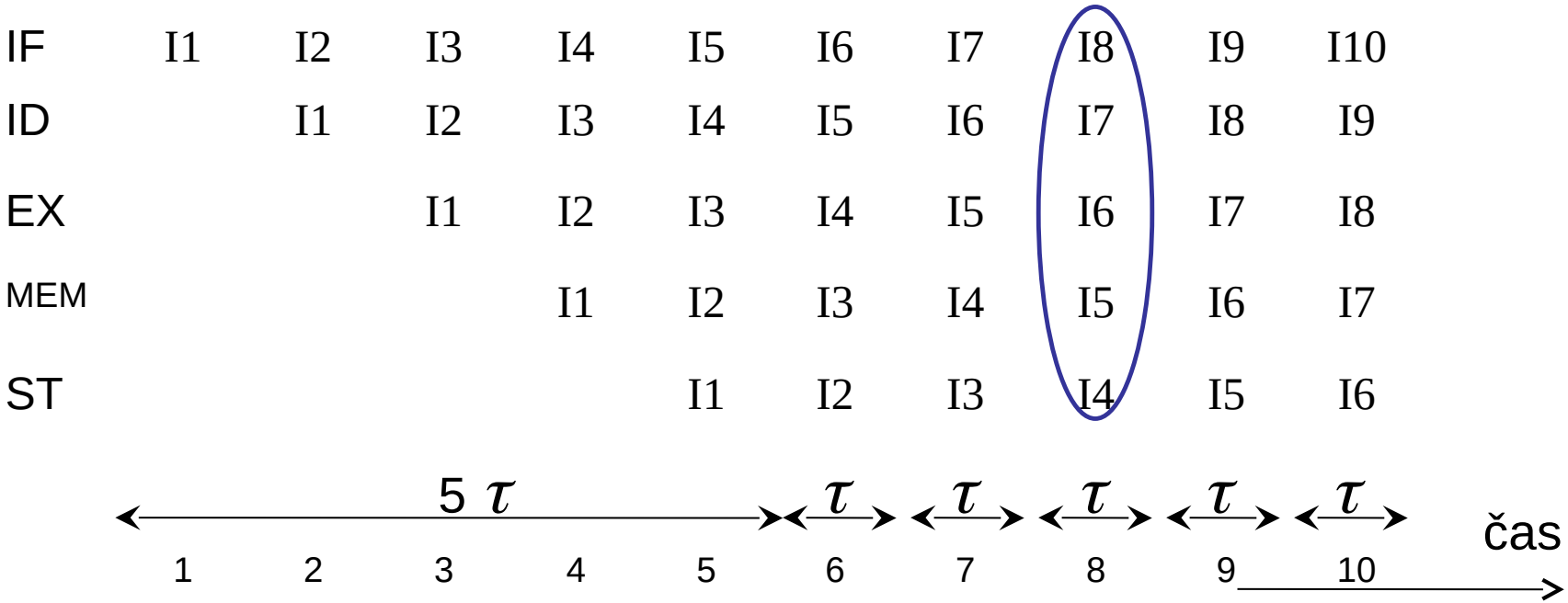
Obsah

Zřetězené vykonávání instrukcí – pipelining

Budeme uvažovat rozdělení vykonávání instrukcí do pěti stupní (stages).

1. **IF** načtení instrukce **Instruction Fetch** – přivedení **PC** na adresový vstup paměti, načtení instrukce a paralelně příprava $PC = PC + 4$
2. **ID** dekódování instrukce **Instrukction Decode** – dekódování operačního znaku (opcode), přímého operand a načtení registrů podle polí `rs1` a `rs2`
3. **EX** vykonání instrukce **EXecution** – provedení požadované funkce, průchod hodnot registrů a konstant ALU
4. **MEM** paměťové přístupy **MEMory** – pokud je požadovaný, proběhne v tomto stupni zápis (**sw**) nebo čtení (**lw**) paměti
5. **WB** aktualizace registrů **WriteBack** – zápis výsledku do pole registrů pro meziregistrové instrukce a paměti

Překrývající se zřetězené vykonávání instrukcí

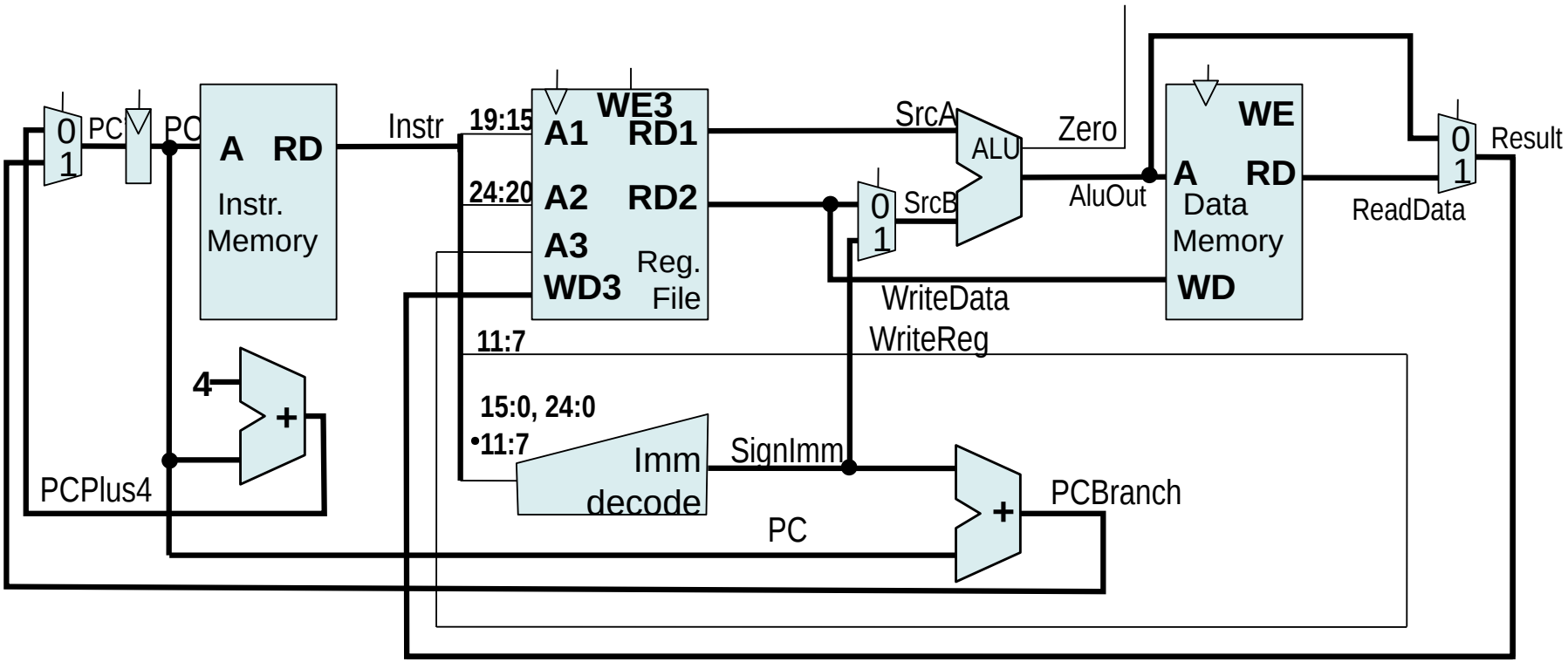


$\tau = max\{\tau_i\}_{i=1}^{k}$, kde τ_i je zpožení v jednotlivých stupních
Vykonání n instrukcí pro k stupňové zřetězené zpracování trvá

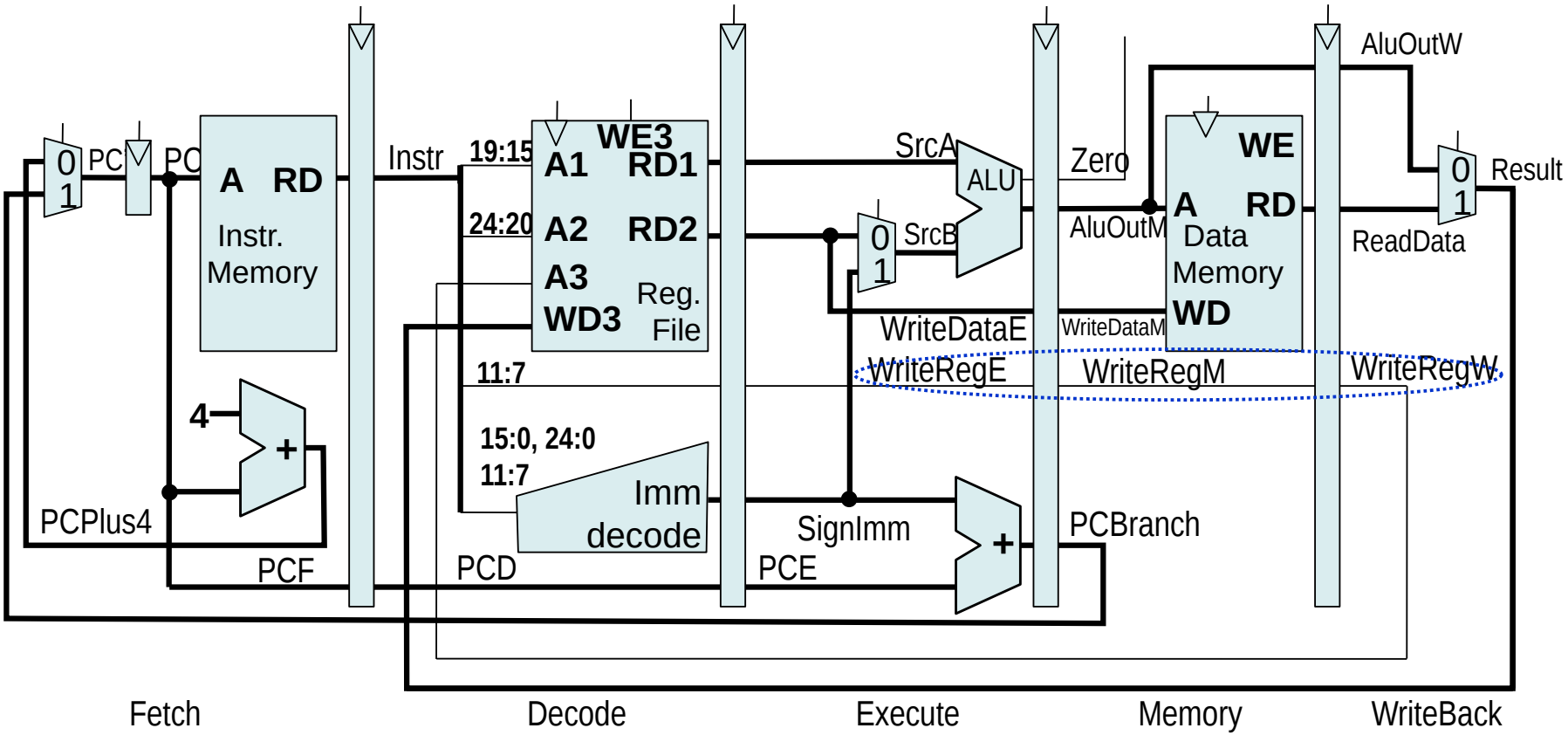
$$T_{k,n} = k \cdot \tau + (n-1)\tau$$

Zrychlení (Speedup) $S_{k,n} = \frac{T_{1,n}}{T_{k,n}} = \frac{nk\tau}{k\tau+(n-1)\tau}$ $\lim_{n\to\infty} S_k = k$

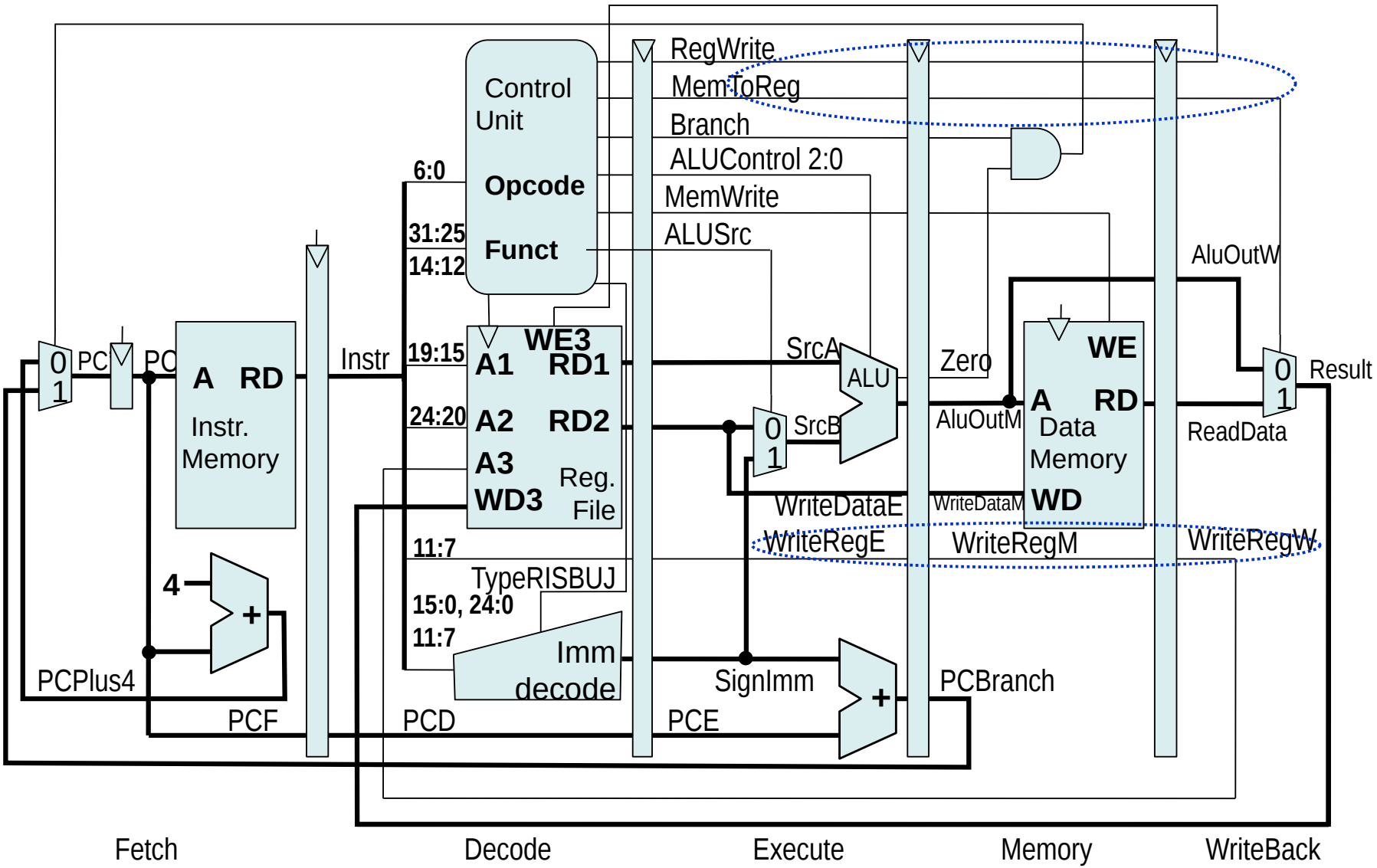
Jednocyklový procesor (z přednášky 3) – datová cesta



Zřetězený pětistupňový návrh – datová cesta



Zřetězený pětistupňový návrh včetně řídicí jednotky



Obsah

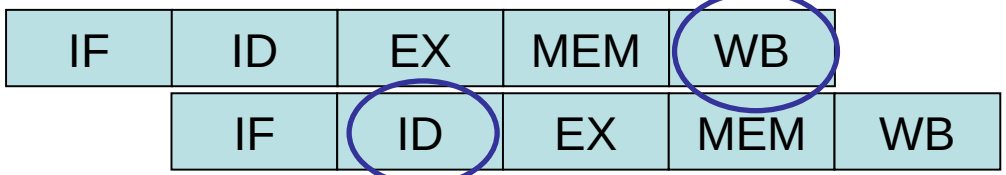
Datové a řídící hazardy a jejich řešení

Datové hazardy

add zapisuje novou hodnotu do **X1**

add x1,x2,x3

sub x4,x1,x3



sub načte starou hodnotu **X1**

tok instrukcí a očekávané předání výsledků

Řídící hazardy

Vyhodnocení podmínky a výpočet **PC**

Nastavení **PC** na cíl skoku

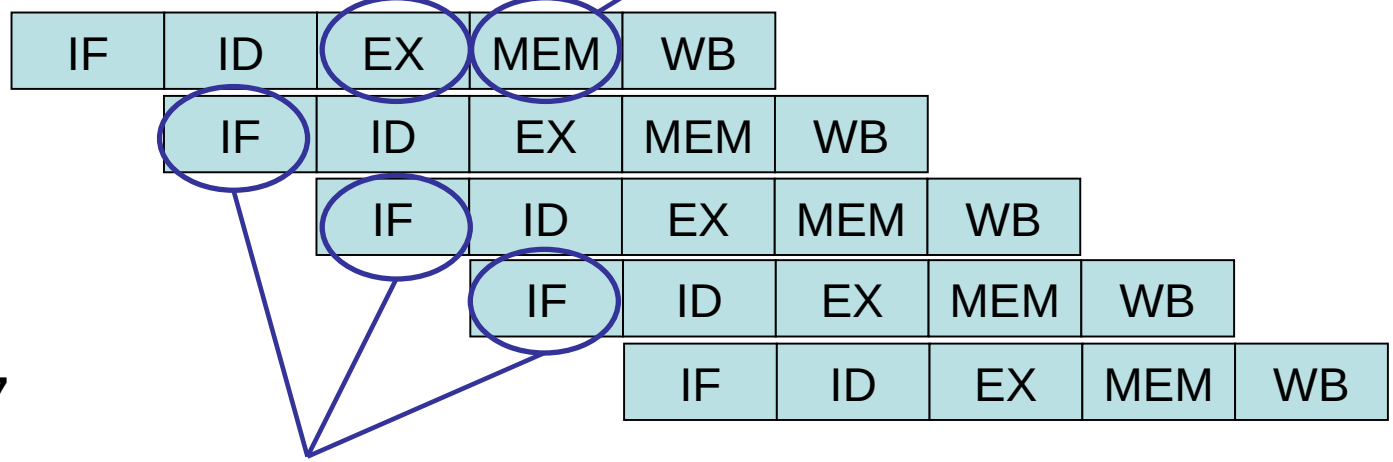
beq x3,x4,m1

add x6,x1,x2

instruction 3

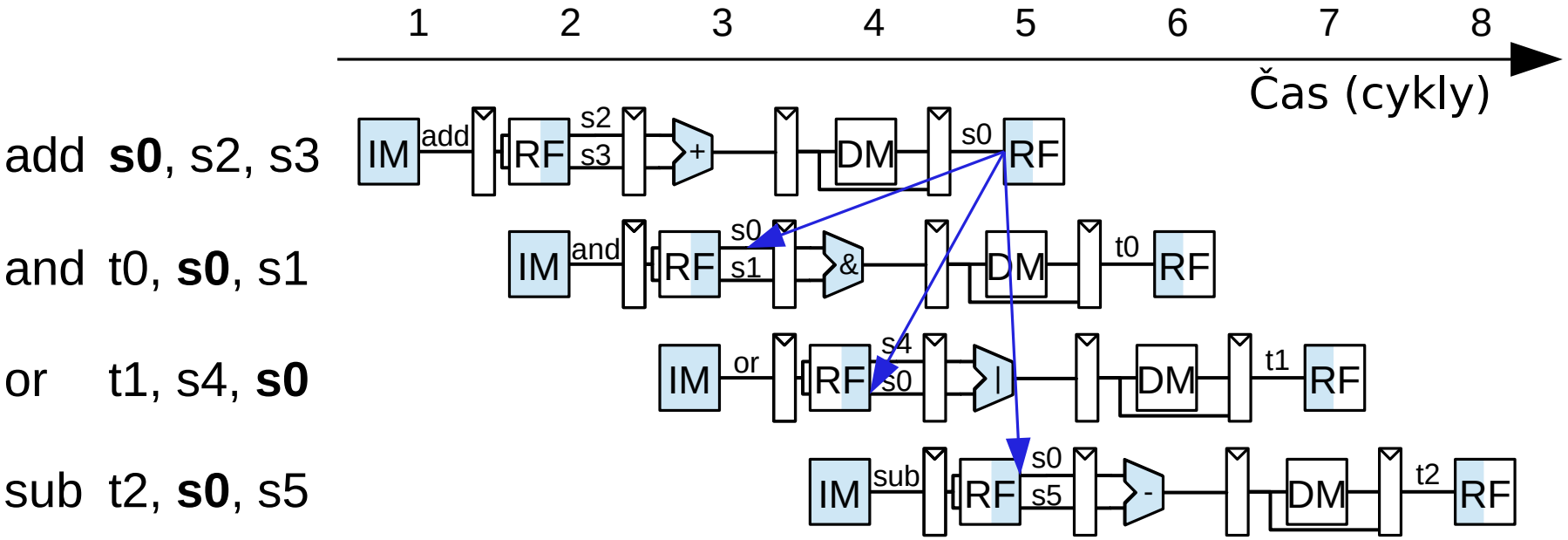
instruction 4

m1: add x4,x6,x7



Mají být tyto instrukce načtené a je správné je vykonat?

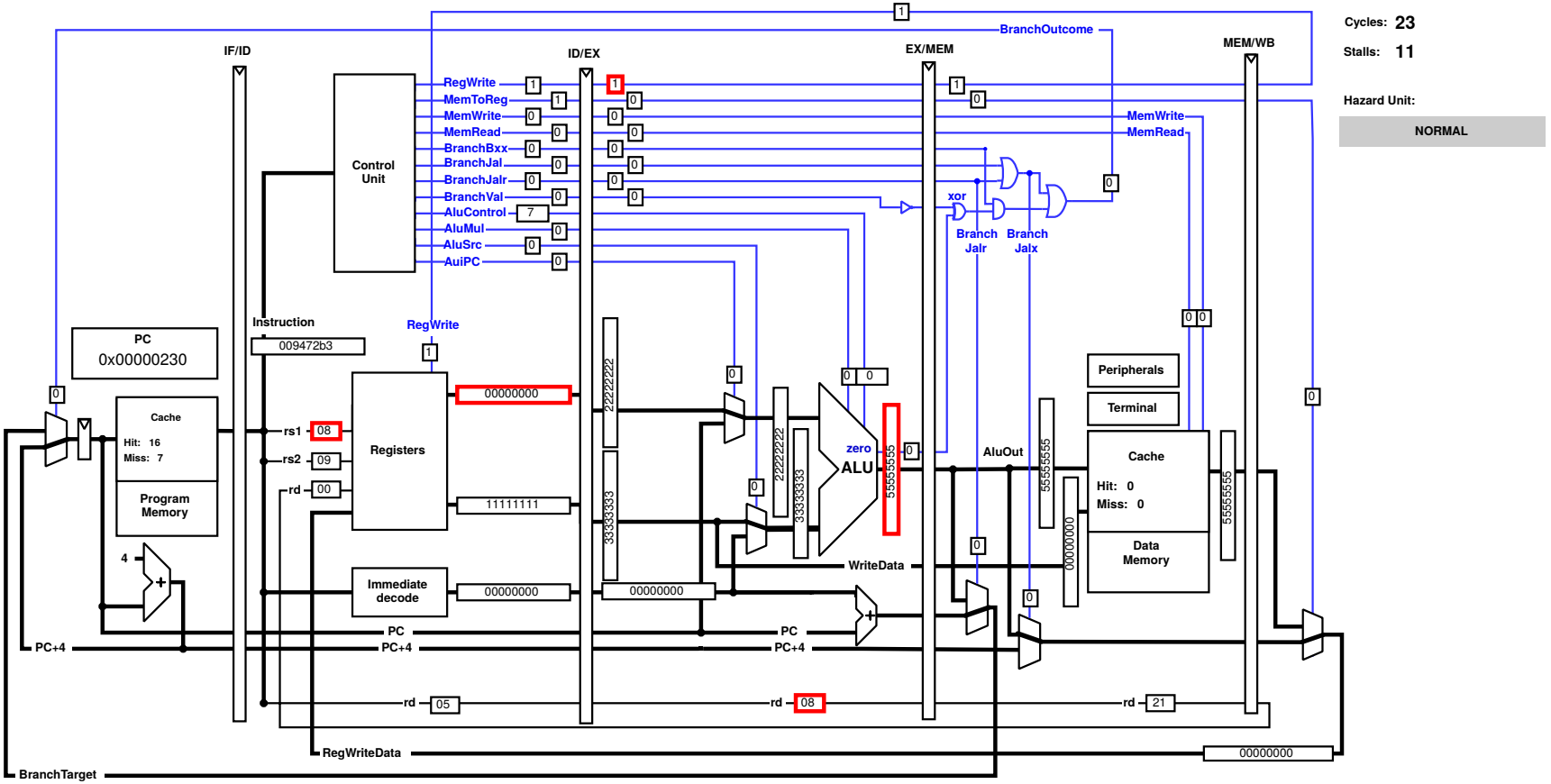
Zdroj datových hazardů



- K zápisníkové paměti (register file) se přistupuje ze dvou stupňů (Decode, WriteBack) – zápis se uskutečňuje v první půli cyklu, čtení v druhé $\Rightarrow$ v instrukci sub pro vstop **s0** hazard nevzniká
- Čtení-po-zápisu (Read After Write – RAW) hazard vzniká u instrukcí **and** a **or** při čtení **s0** v cyklech 3 a 4
- Jak zamezit takovému hazardu bez degradace propustnosti pipeline?

Datový hazard v instrukci ”and”sledovaný v simulátoru

or x6, x20, **x8**	and x5, **x8**, x9	add **x8**, x18, x19	addi x21, x21, 1365	nop
or t1, s4, **s0**	and t1, **s0**, s1	add **s0**, s2, s3	addi s5, s5, 1365	nop

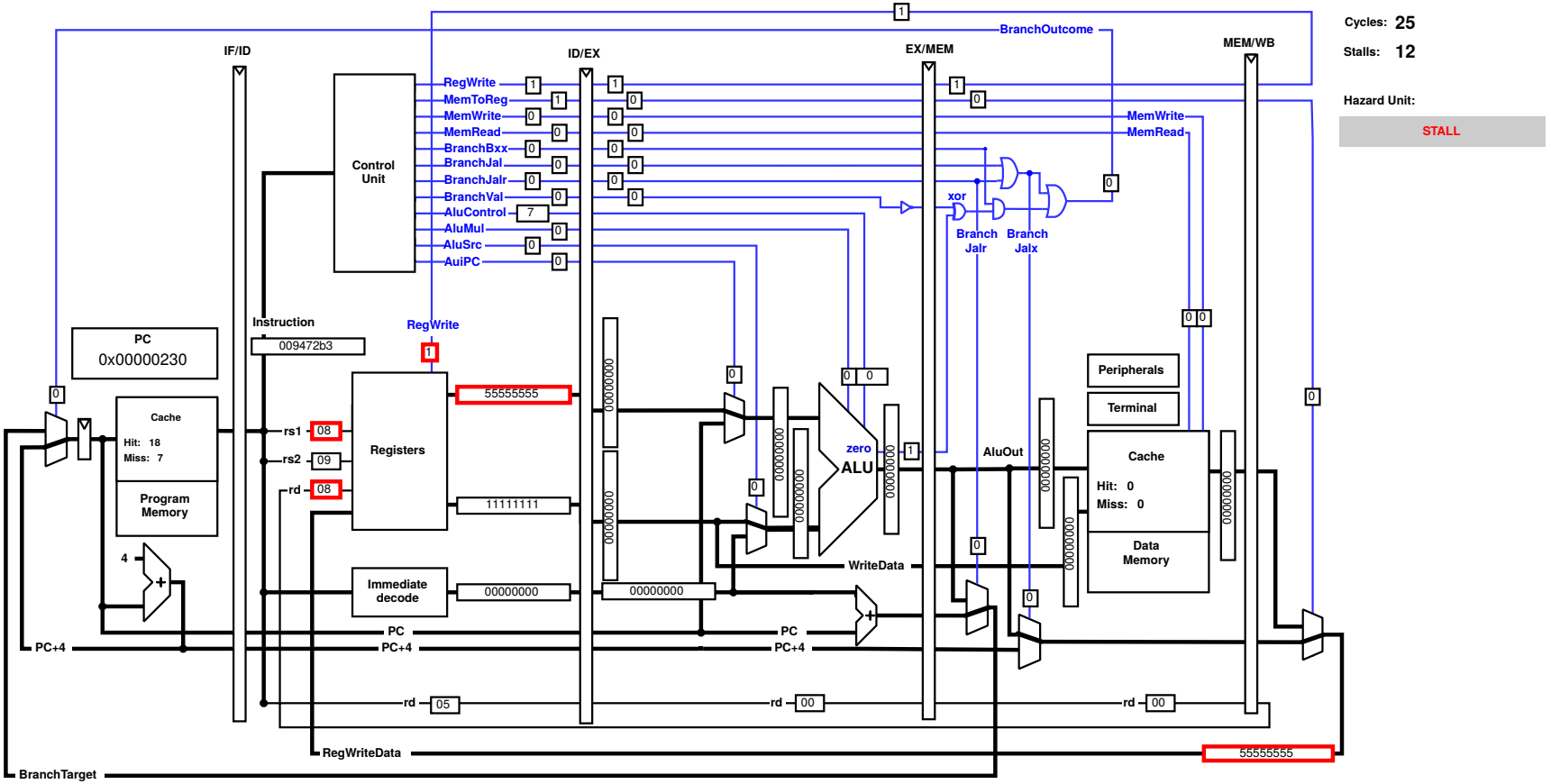


QtRvSim `https://github.com/cvut/qtrvsim`

`https://gitlab.fel.cvut.cz/b35apo/stud-support/-/blob/master/seminaries/qtrvsim/hazards-from-lecture/alu-hazards.S`

Vyřešení datového hazardu pozastavením (stall)

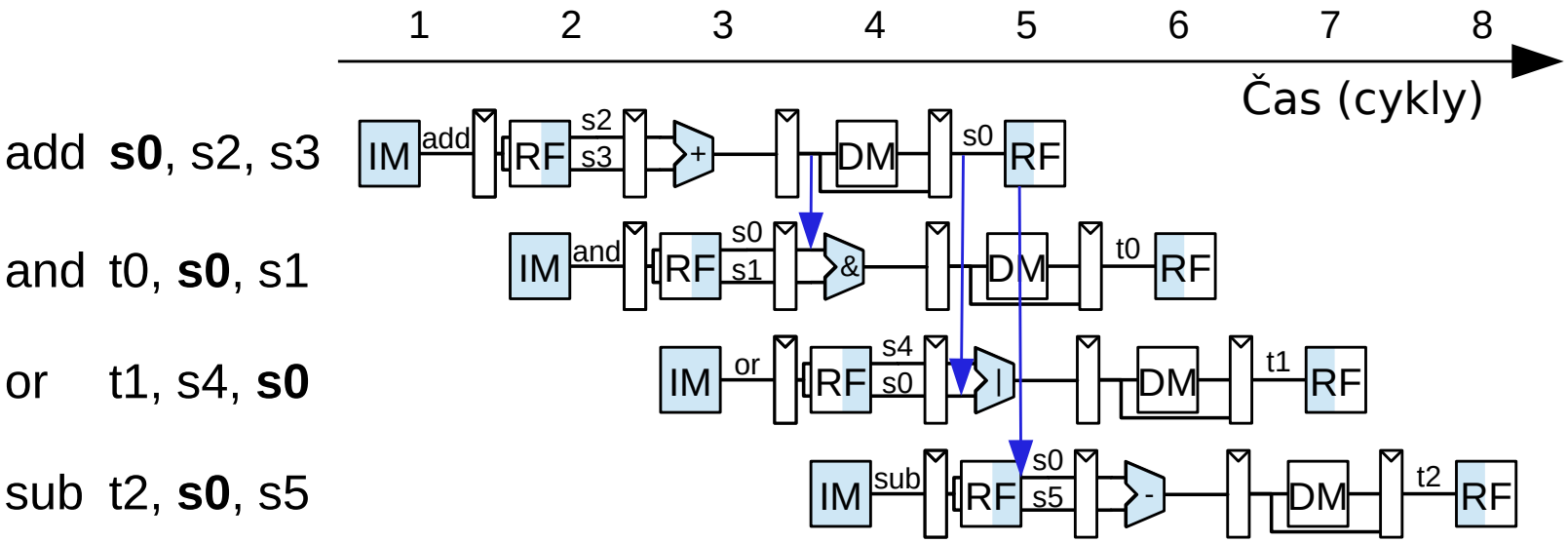
or x6, x20, **x8**	and x5, **x8** , x9	nop	nop	add **x8** , x18, x19
or t1, s4, **s0**	and t1, **s0** , s1	stall	stall	add **s0** , s2, s3



QtRvSim `https://github.com/cvut/qtrvsim`

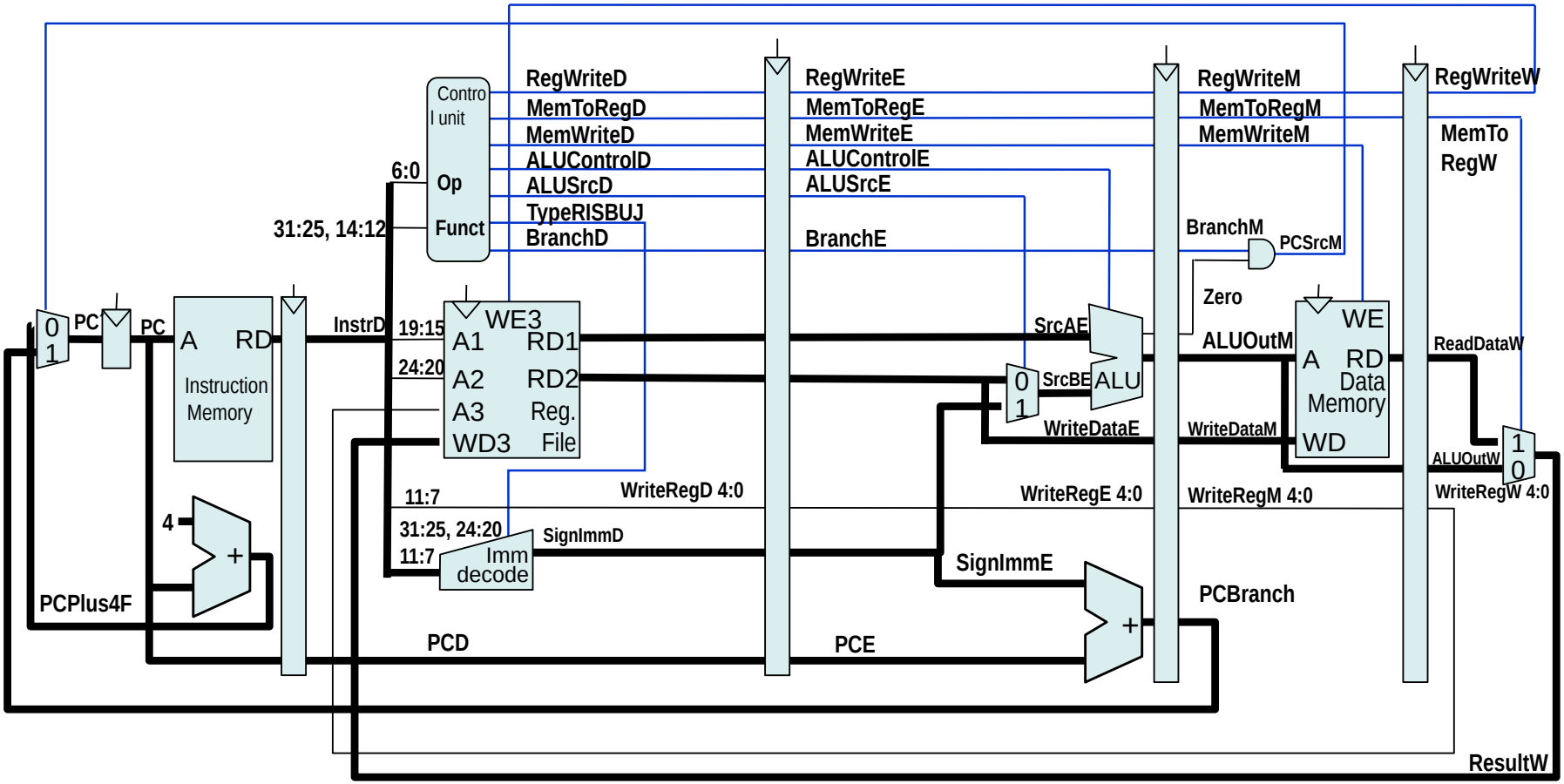
`https://gitlab.fel.cvut.cz/b35apo/stud-support/-/blob/master/seminaries/qtrvsim/hazards-from-lecture/alu-hazards.S`

Rešení datových hazardů přeposíláním (forwarding)



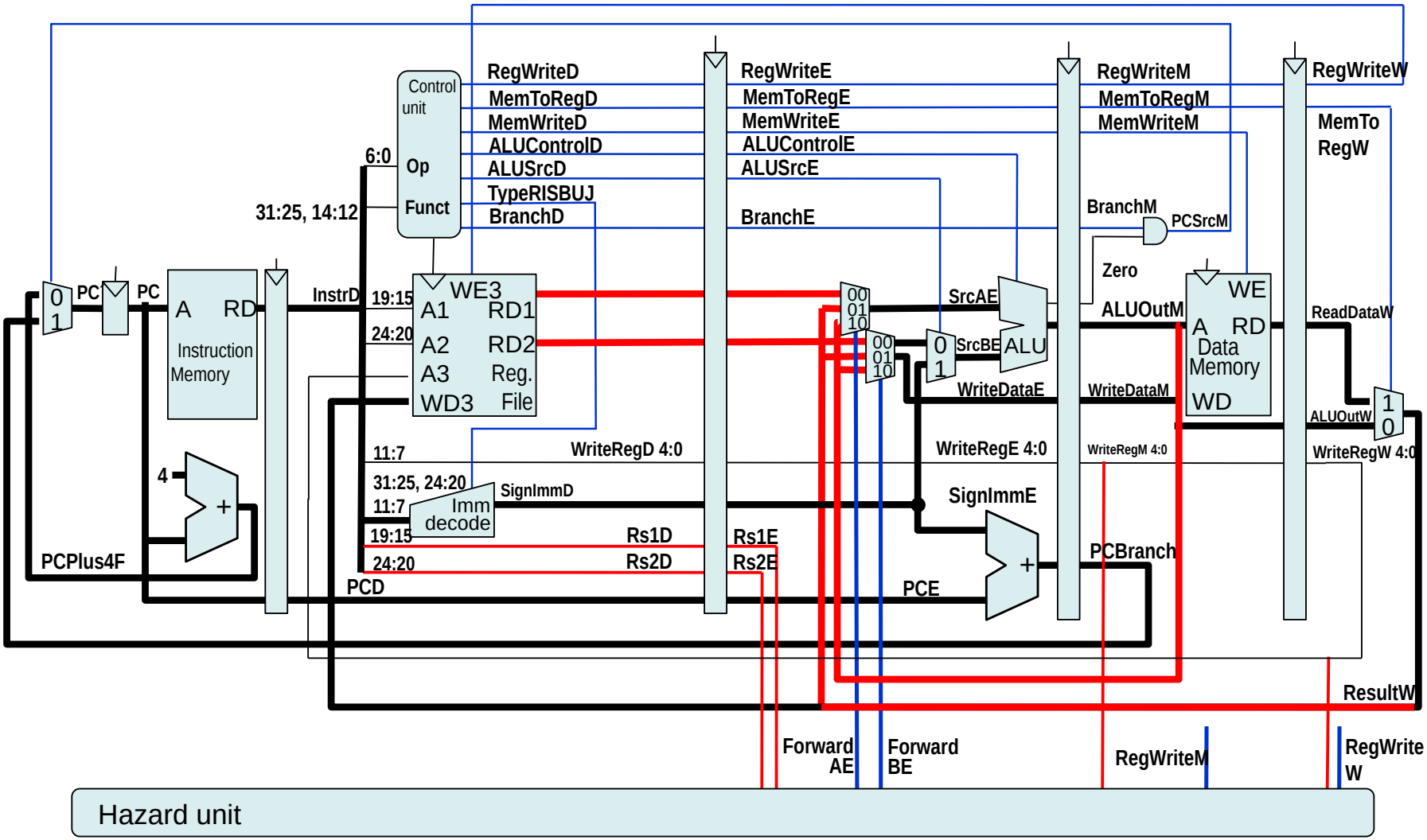
- pokud je výsledek k dispozici (vypočítaný) před vykonáním následující instrukce/í, která/é vyžadují výsledek, tak je možné datový hazard vyřešit přeposláním hodnoty
- K datovému hazardu v uvažovaném návrhu dochází, pokud některý zdrojový registr (**rs1**, **rs2**) v stupni **EX** odovídá cílovému registru ve stupni **MEM** nebo **WB** (vyjma X0/zero)
- Čísla registrů z daných stupňů jsou přivedena na jednotku řešení hazardů (Hazard Unit – HU)

Zopakovaný předchozí návrh a příprava na přeposílání



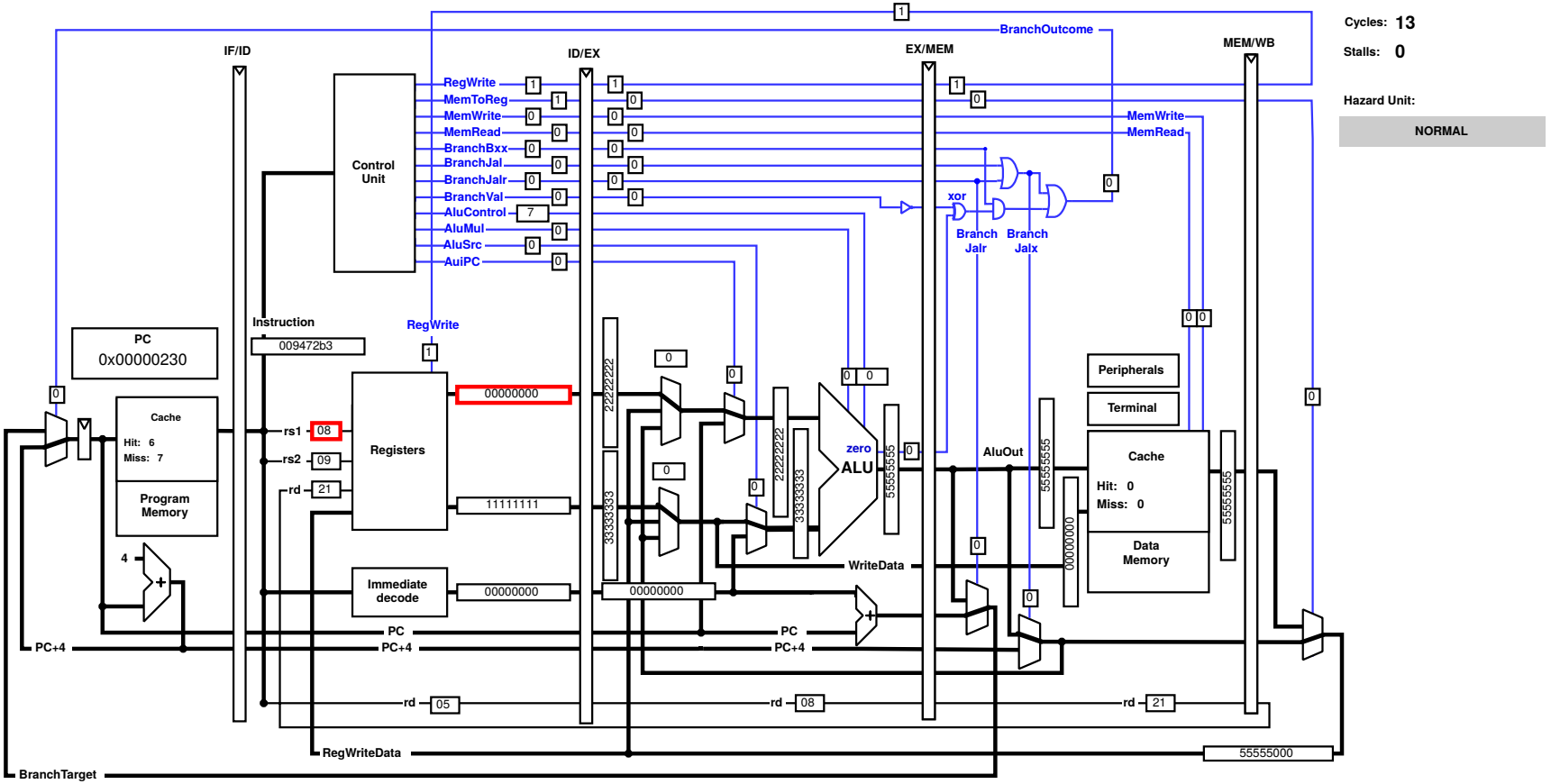
Je potřeba znát předchozí **A1** (**rs1**) a **A2** (**rs2**) v **EX**. Signály **RegWrite** z **MEM** a **WB** musí být také monitorované aby potvrzovaly, že registr určený **WriteReg** v **MEM** a **WB** je opravdu zapisovaný.

Návrh procesoru s přeposíláním z MEM a WB do EX



QtRvSim – datové hazardy řešené přeposíláním

or x6, x20, **x8**	and x5, **x8** , x9	add **x8** , x18, x19	addi x21, x21, 1365	lui x21, 0x55555
or t1, s4, **s0**	and t1, **s0** , s1	add **s0** , s2, s3	addi s5, s5, 1365	lui s5, 0x55555

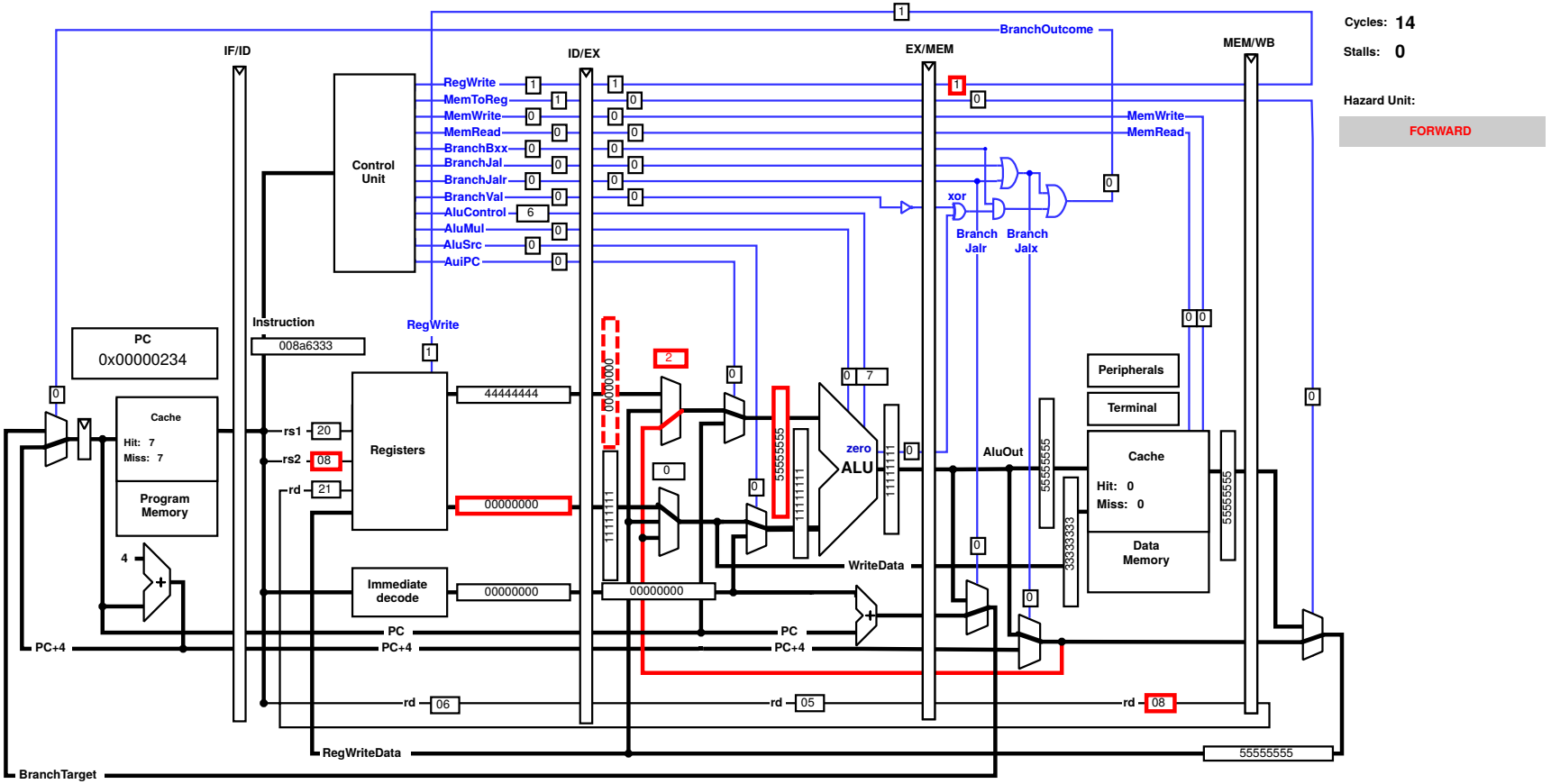


QtRvSim `https://github.com/cvut/qtrvsim`

`https://gitlab.fel.cvut.cz/b35apo/stud-support/-/blob/master/seminaries/qtrvsim/hazards-from-lecture/alu-hazards.S`

QtRvSim – přeposílání vstupu instrukce ”and”z MEM

sub x7, **x8**, x21	or x6, x20, **x8**	and x5, **x8** , x9	add **x8** , x18, x19	addi x21, x21, 1365
sub t2, **s0**, s5	or t1, s4, **s0**	and t1, **s0** , s1	add **s0** , s2, s3	addi s5, s5, 1365

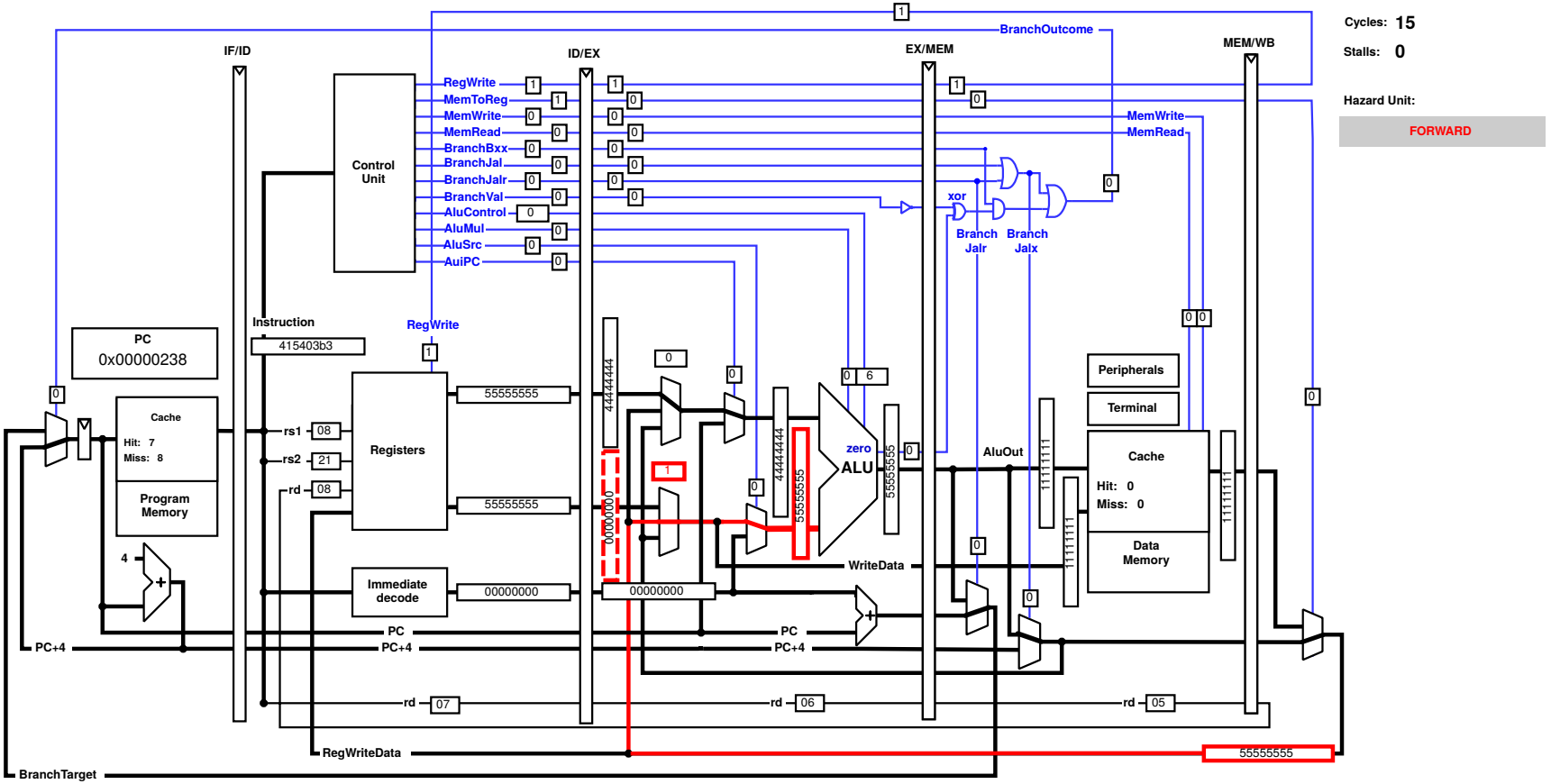


QtRvSim `https://github.com/cvut/qtrvsim`

`https://gitlab.fel.cvut.cz/b35apo/stud-support/-/blob/master/seminaries/qtrvsim/hazards-from-lecture/alu-hazards.S`

QtRvSim – přeposílání vstupu instrukce ”or”z WB

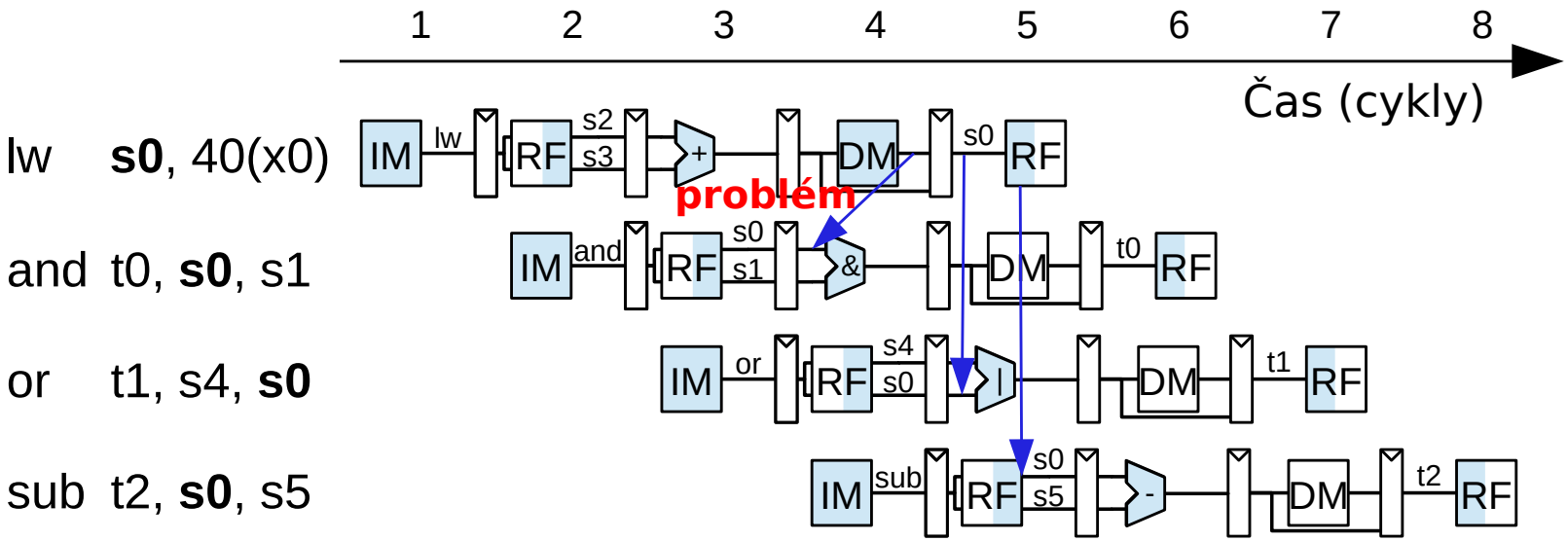
unknown	sub x7, **x8** , x21	or x6, x20, **x8**	and x5, **x8** , x9	add **x8** , x18, x19
unknown	sub t2, **s0** , s5	or t1, s4, **s0**	and t1, **s0** , s1	add **s0** , s2, s3



QtRvSim `https://github.com/cvut/qtrvsim`

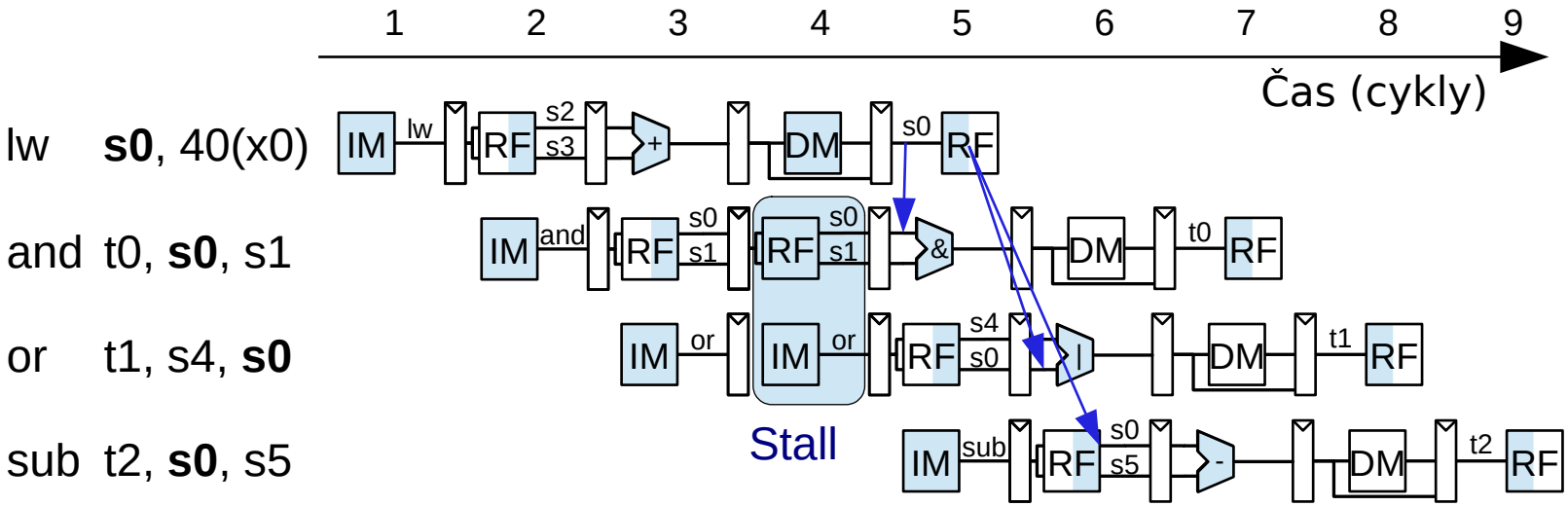
`https://gitlab.fel.cvut.cz/b35apo/stud-support/-/blob/master/seminaries/qtrvsim/hazards-from-lecture/alu-hazards.S`

Přetrvávající problém v instrukci ”lw”– řešení pozastavení



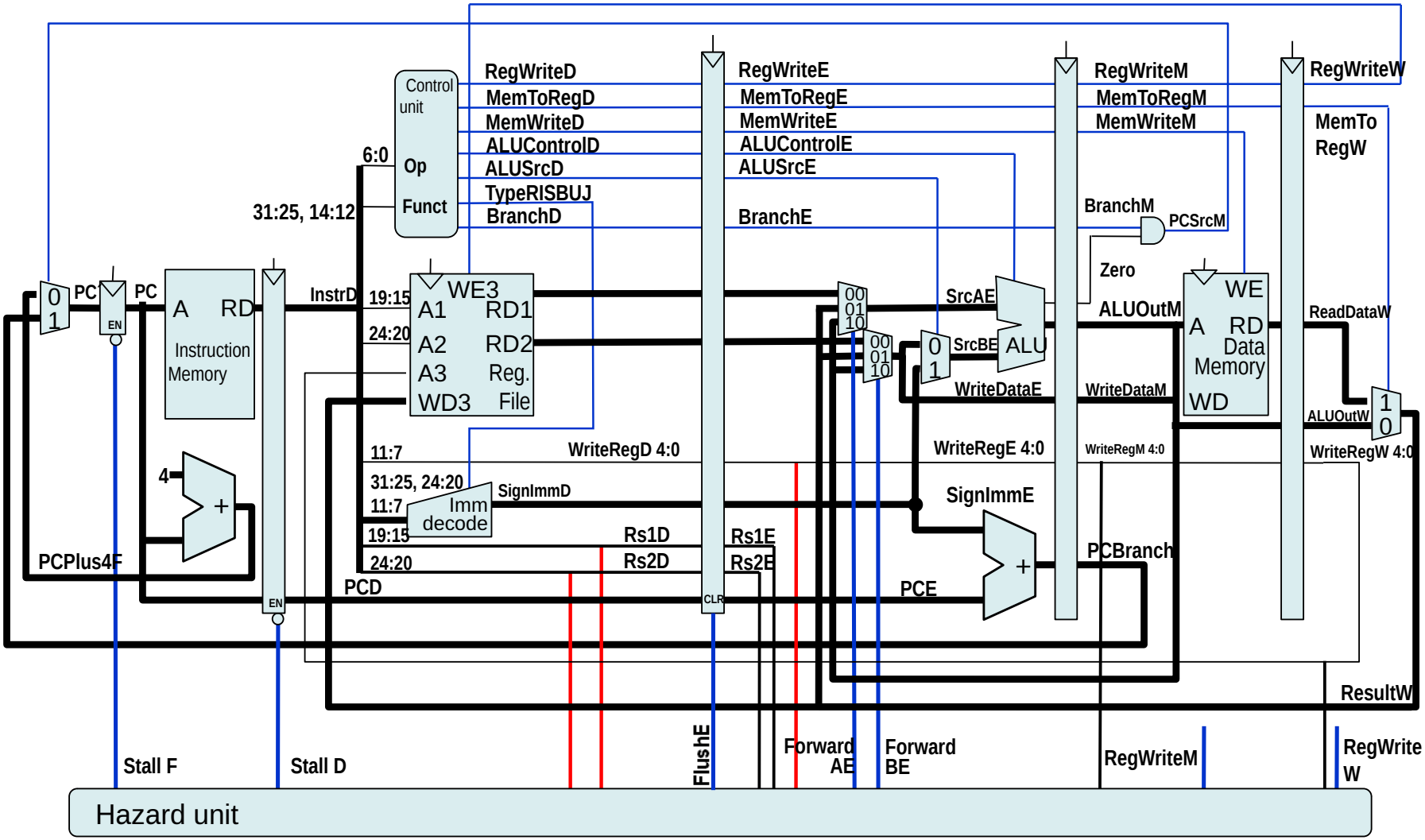
- Pokud následující instrukce vyžaduje data dříve, než jsou k dispozici v procesoru, tak je nutné pipeline pozastavit (**stall**) a do pokračování pipeline jsou vloženy stall/**nop** stavy
- Pozastavení řeší hazardy ale degraduje propustnost
- Stupně, které předchází stupni dotčenému hazardem, jsou pozastaveny, dokud nejsou všechny dalšími instrukcemi požadované výsledky k dispozici – výsledky jsou pak přeposlané na všechna místa, kde jsou požadované

Řešení datových hazardů pozastavením (pipeline stall)



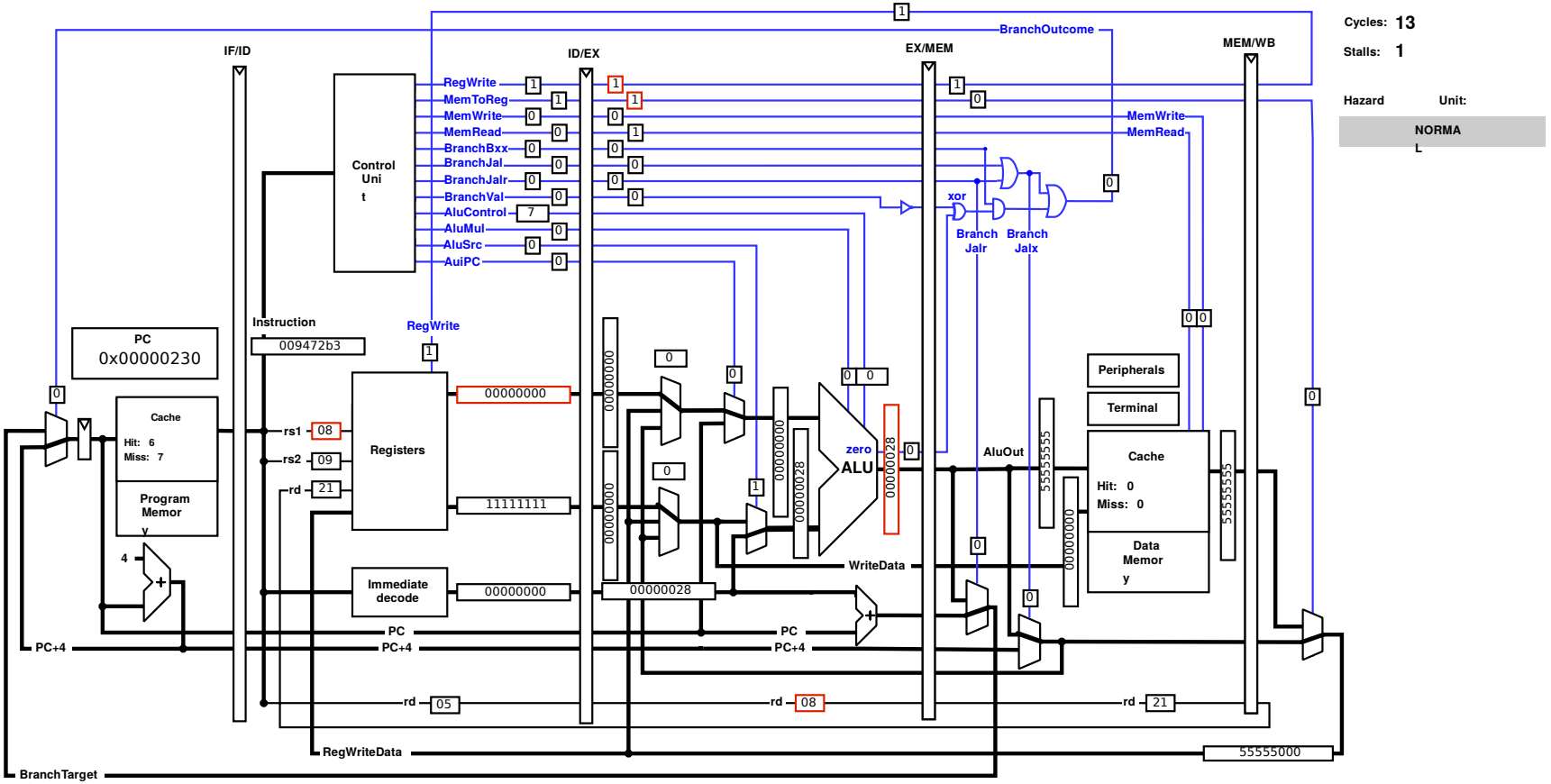
- Pozastavení je realizované podržením obsahu mezistavových (interstage) registrů (hradlováním jejich hodinového signálu nebo nahrávacích signálů)
- Výsledky ze stupňů s chybějícími daty se nepoužijí, vynulují se signály pro zápis do paměti a registrů i ostatní řídicí signály
- Obojího je dosaženo přidáním řídicích signálů pro pozdržení a nebo nulování mezistupňových registrů

Řešení zbývajících datových hazardů pozastavením



QtRvSim – datový hazard v ”lw”řešený pozastavením

or x6, x20, **x8**	and x5, **x8**, x9	lw **x8**, 40(x0)	addi x21, x21, 1365	lui x21, 0x55555
or t1, s4, **s0**	and t0, **s0**, s1	lw **s0**, 40(zero)	addi s5, s5, 1365	lui s5, 0x55555

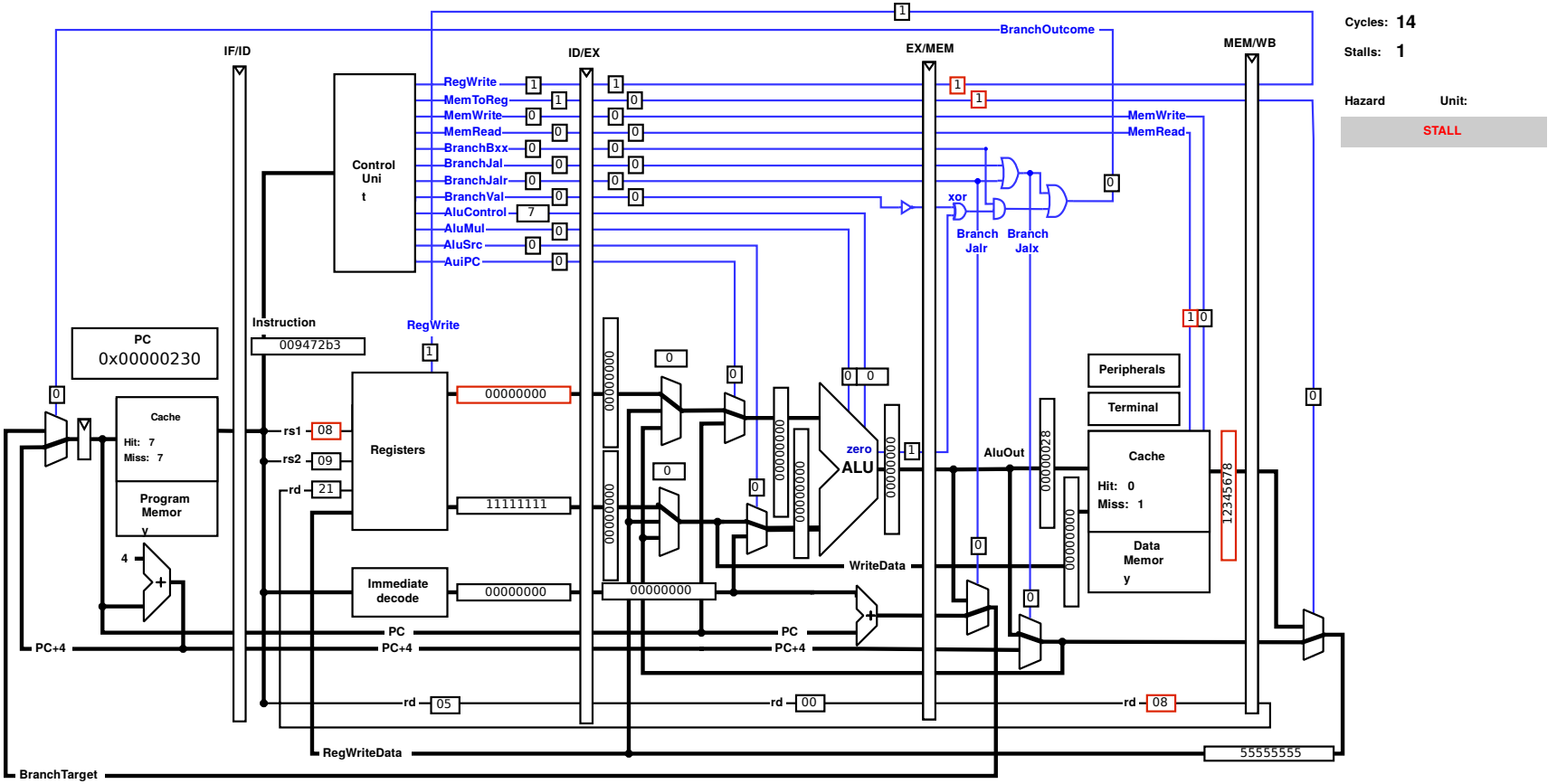


QtRvSim `https://github.com/cvut/qtrvsim`

`https://gitlab.fel.cvut.cz/b35apo/stud-support/-/blob/master/seminaries/qtrvsim/hazards-from-lecture/lw-hazards.S`

QtRvSim – datový hazard v ”lw”řešený pozastavenímm

or x6, x20, **x8**	and x5, **x8**, x9	nop	lw **x8**, 40(x0)	addi x21, x21, 1365
or t1, s4, **s0**	and t0, **s0**, s1	stall	lw **s0**, 40(zero)	addi s5, s5, 1365

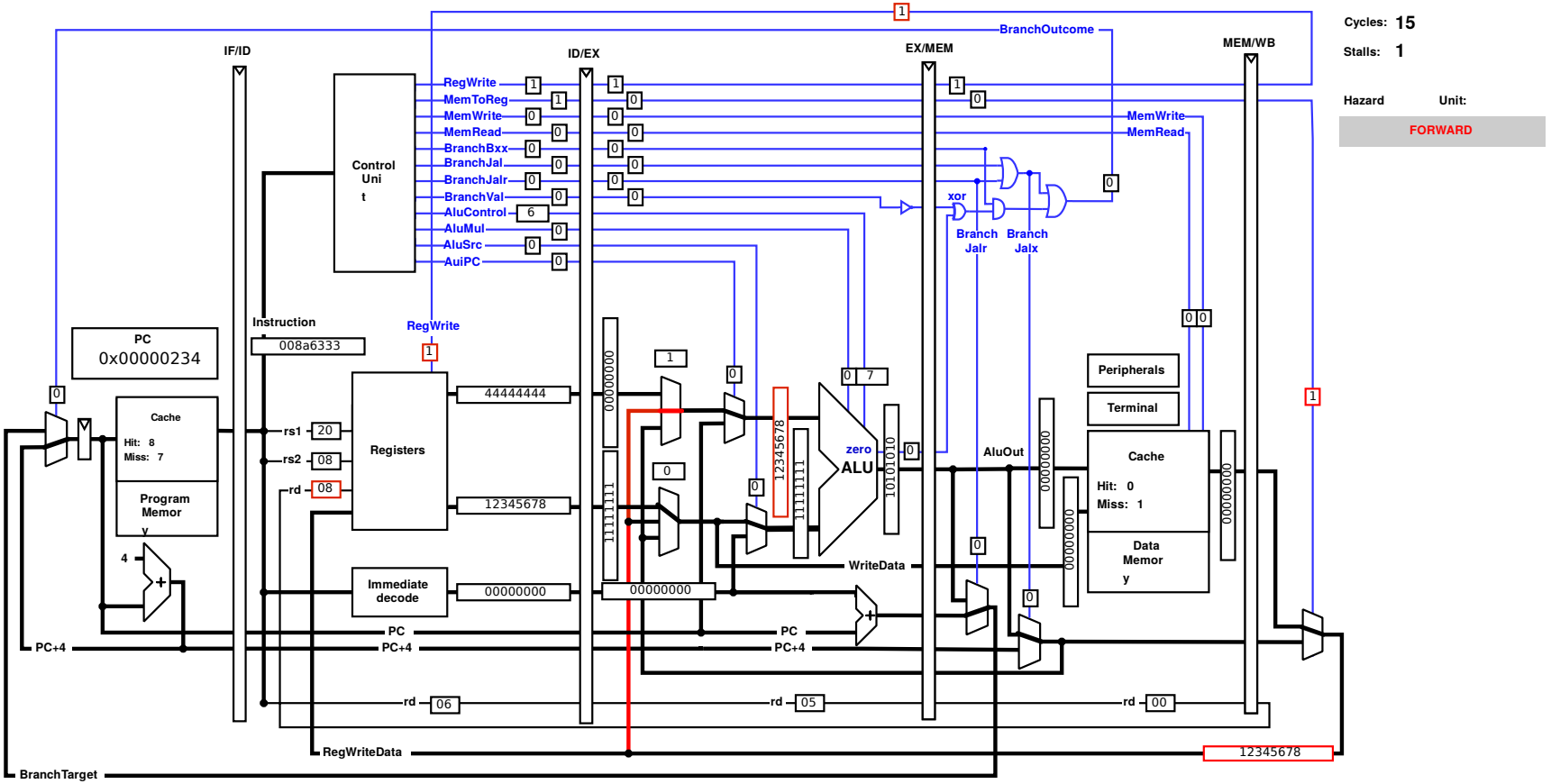


QtRvSim `https://github.com/cvut/qtrvsim`

`https://gitlab.fel.cvut.cz/b35apo/stud-support/-/blob/master/seminaries/qtrvsim/hazards-from-lecture/lw-hazards.S`

QtRvSim – datový hazard v ”lw”řešený pozastavením

sub x7, **x8**, x21	or x6, x20, **x8**	and x5, **x8**, x9	nop	lw **x8**, 40(x0)
sub t2, **s0**, s5	or t1, s4, **s0**	and t0, **s0**, s1	stall	lw **s0**, 40(zero)

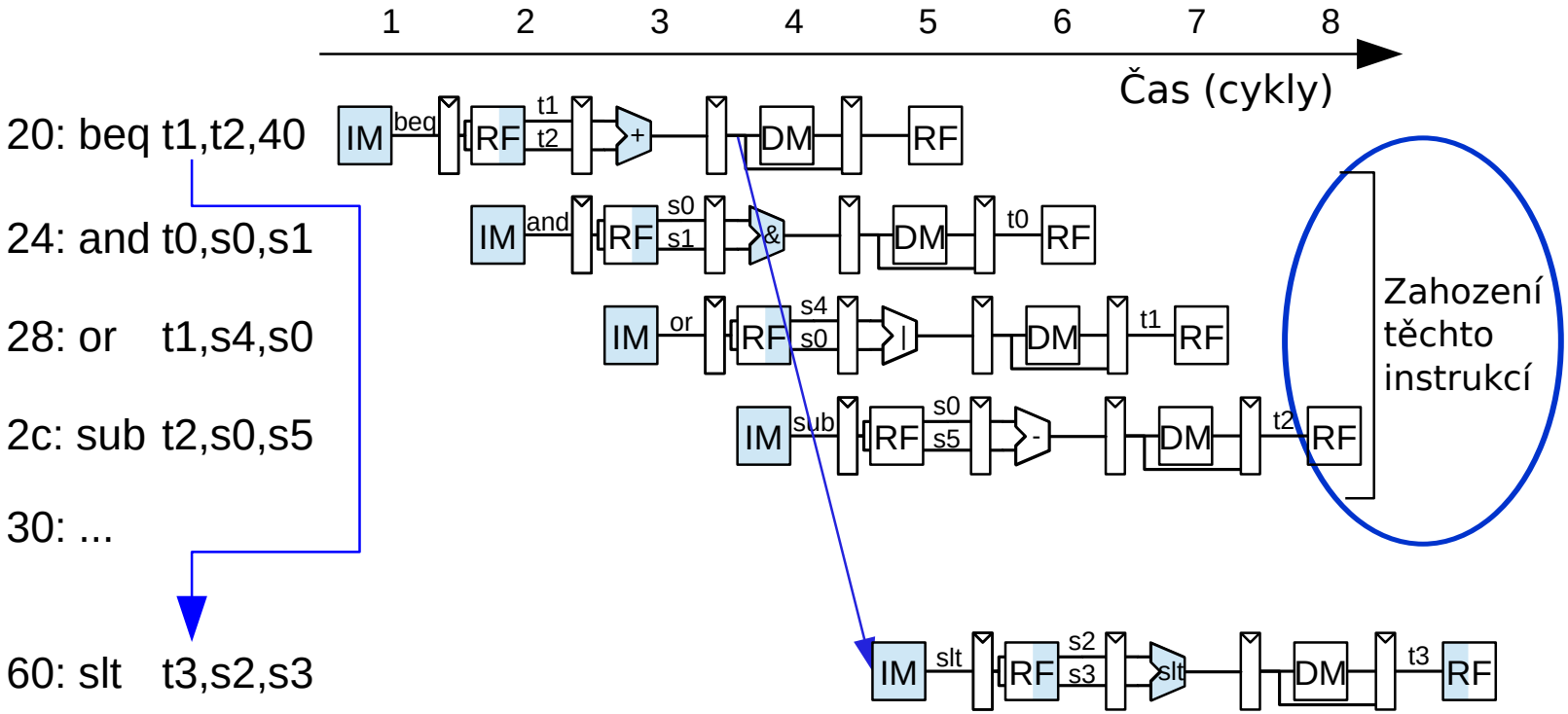


QtRvSim `https://github.com/cvut/qtrvsim`

`https://gitlab.fel.cvut.cz/b35apo/stud-support/-/blob/master/seminaries/qtrvsim/hazards-from-lecture/lw-hazards.S`

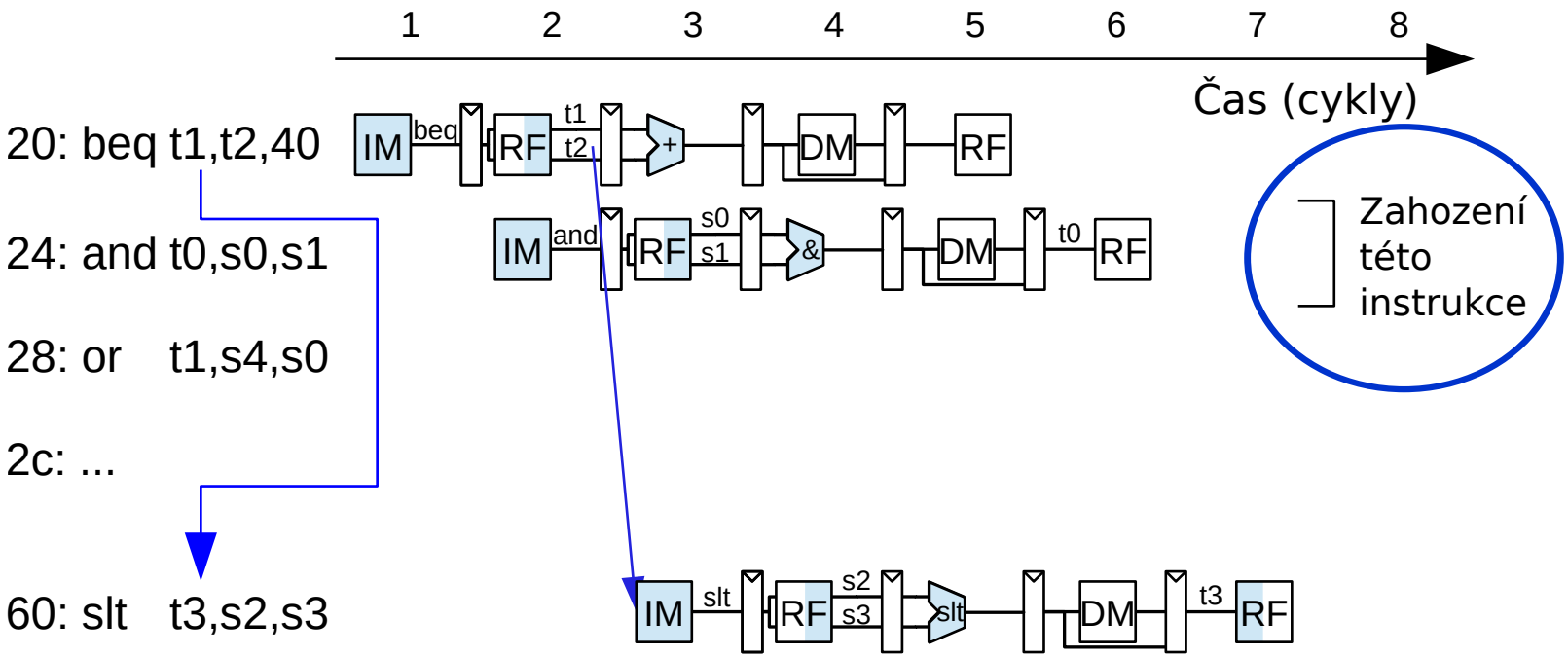
Obsah

Řídicí (control) hazardy (instrukce branch a jump)



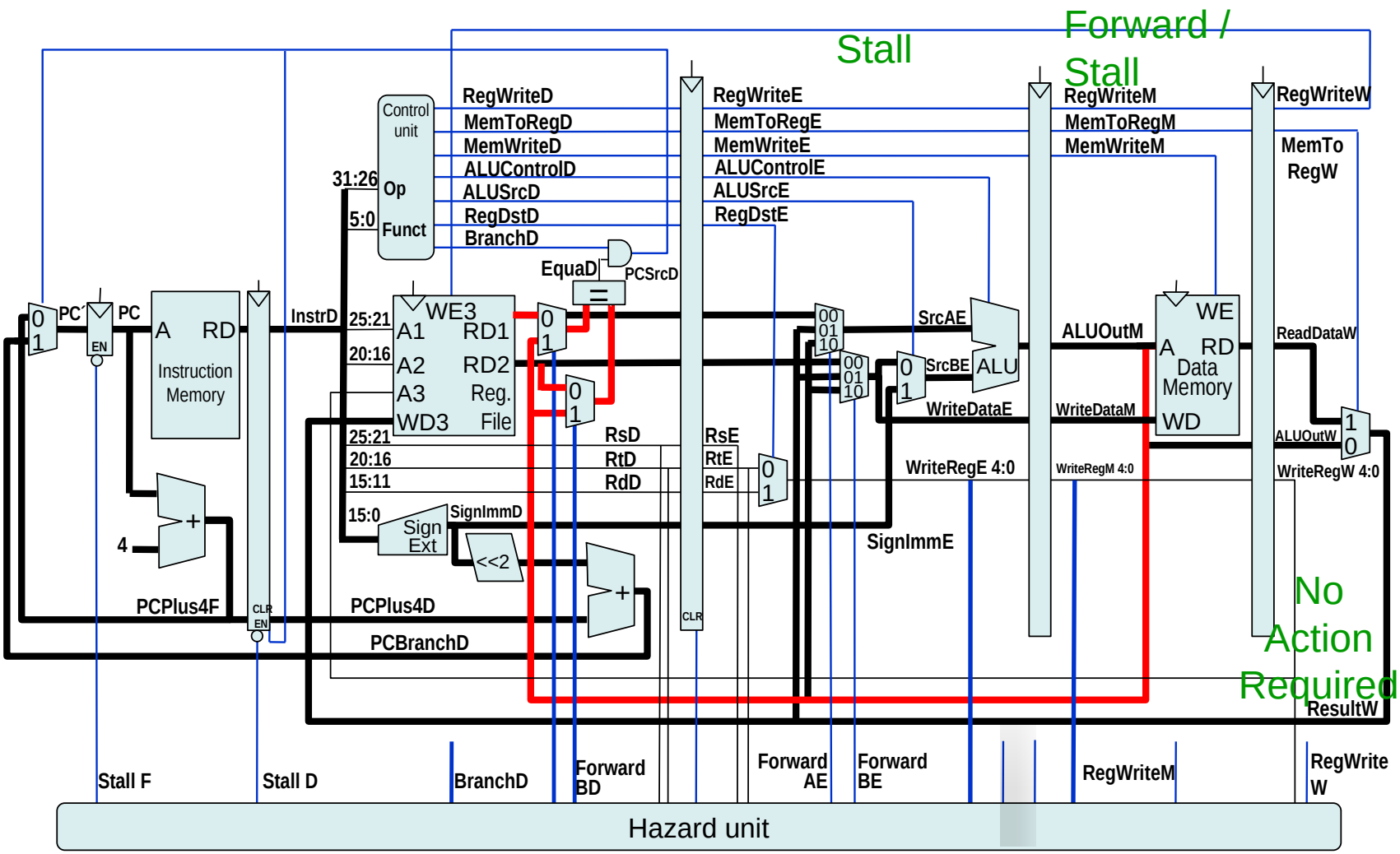
- Výsledek není známý dříve než ve čtvrtém cyklu, poté se nahraje do PC a tak první instrukce na cílové adrese skoku může být načtena až v cyklu pátém
- Skoky takto představují zásadní omezení využití času výkonných jednotek v pipeline

Alternativa, přesunout rozhodování skoků dříve

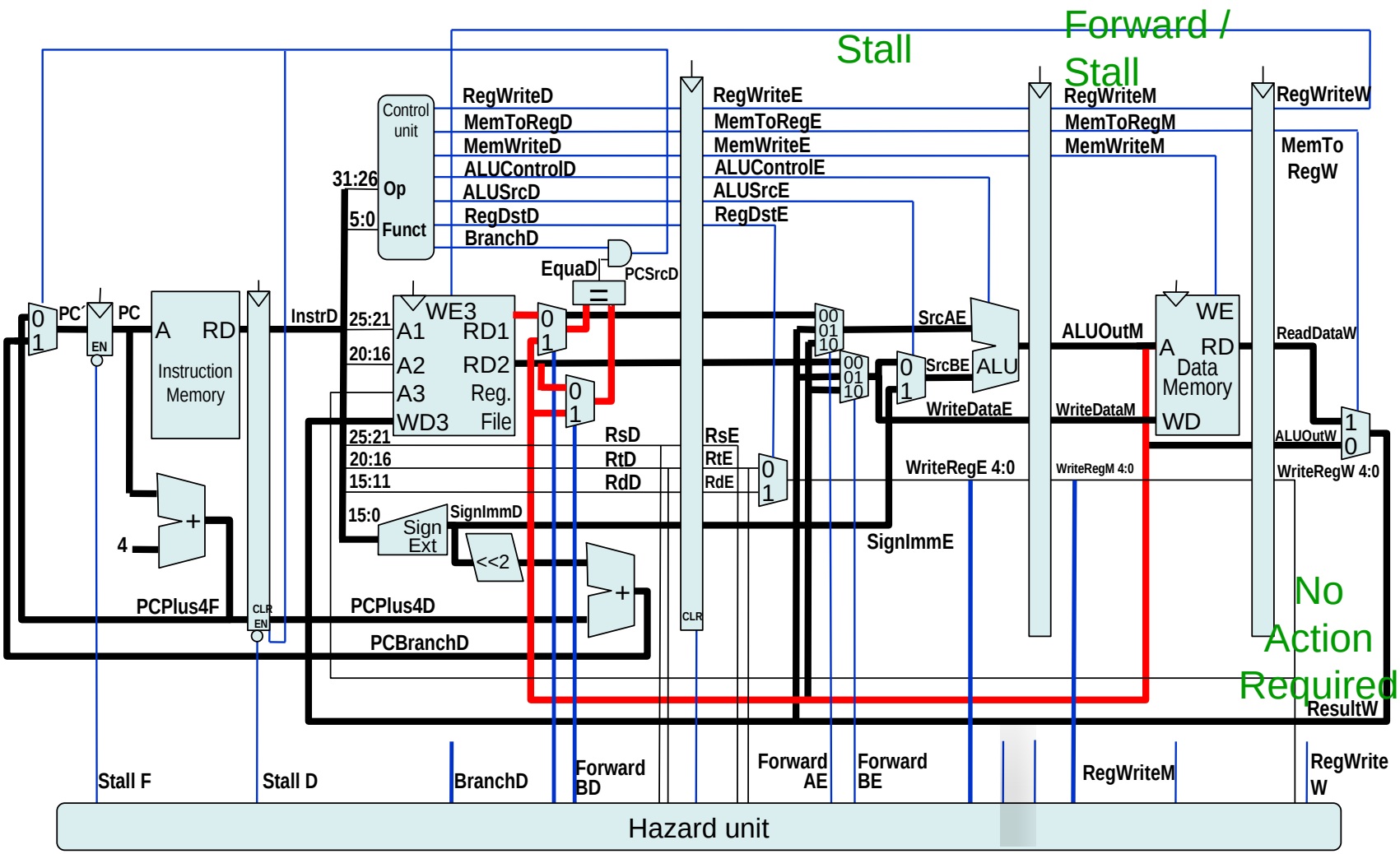


- Vzniká strukturální hazard, **ALU** je potřeba ve stupni **EX** i **ID**.
- Strukturální hazardy lze řešit duplikací prvků. Kompletní sčítačka, odčítačka nebo porovnání na velikost trvá dlouho, řešením může být omezit podmíny skoků jen na shodu/neshodu (stačí xor mezi operandy a nor přes bity)

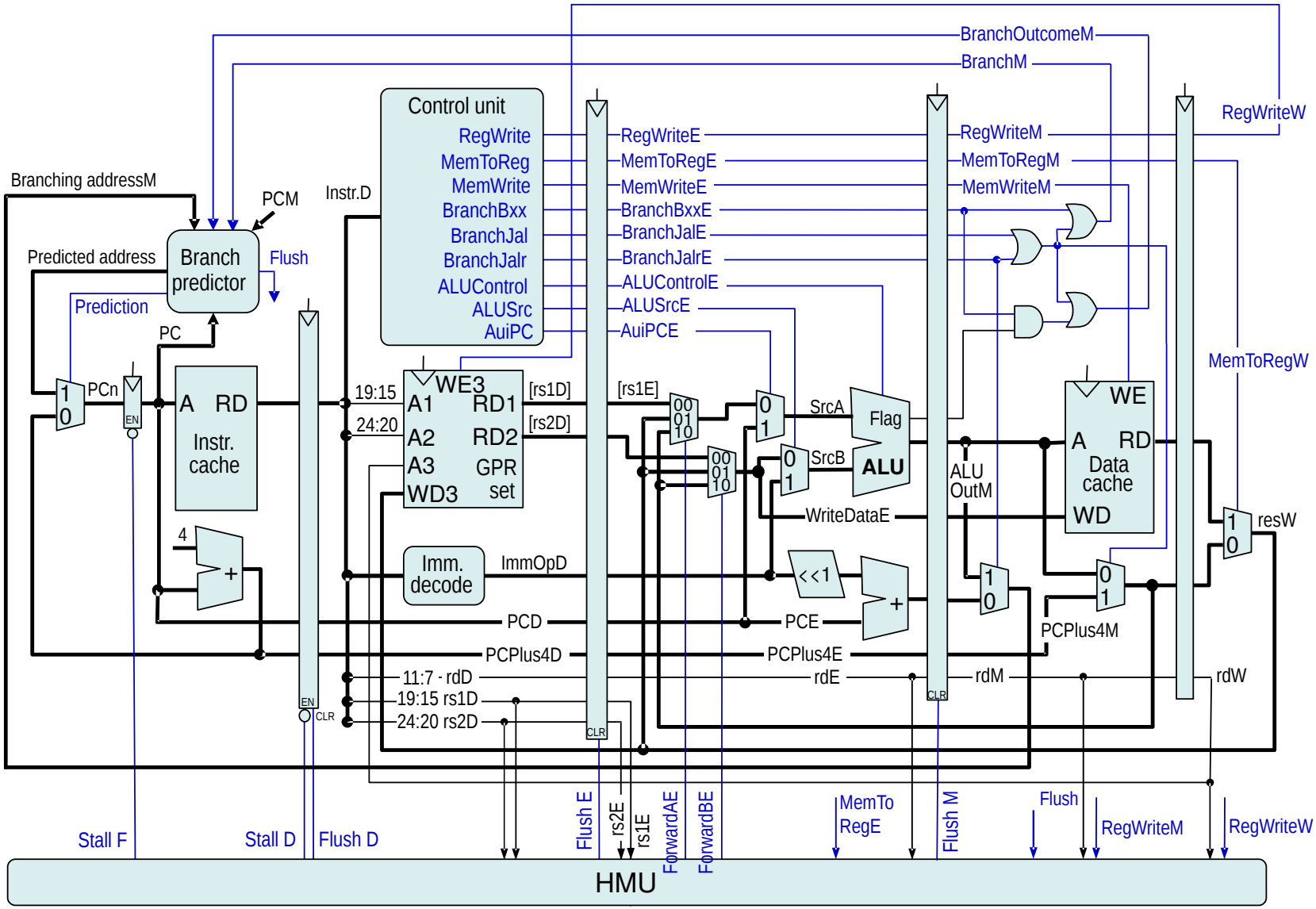
Přesun vykonání skoků do ID – architektura MIPS



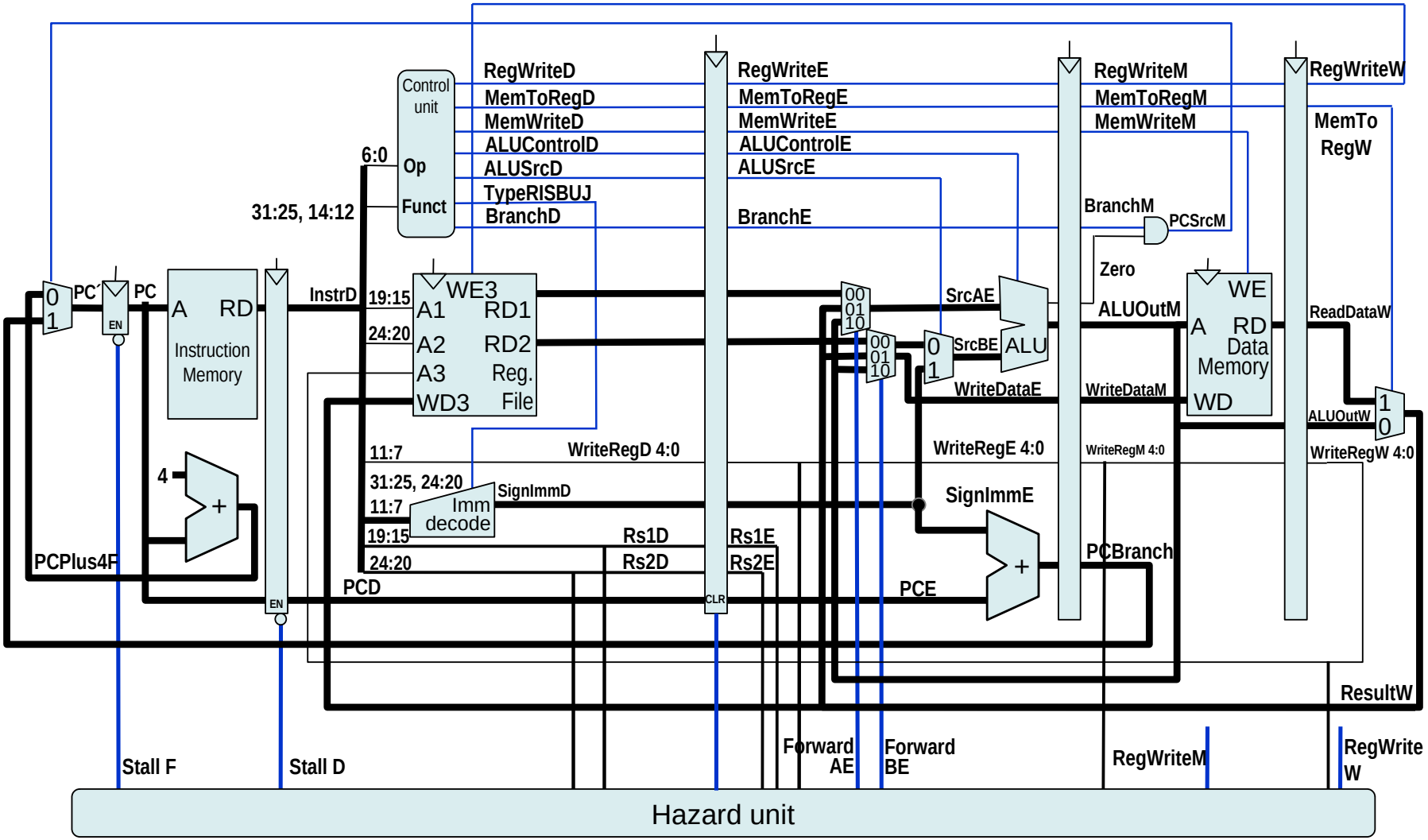
Vyřešení hazardů na vstupech komparátoru – MIPS



Predikce skoků a spekulativní vykonávání instrukcí (příště)



Výsledek návrhu zřetězeného procesoru – RISC-V

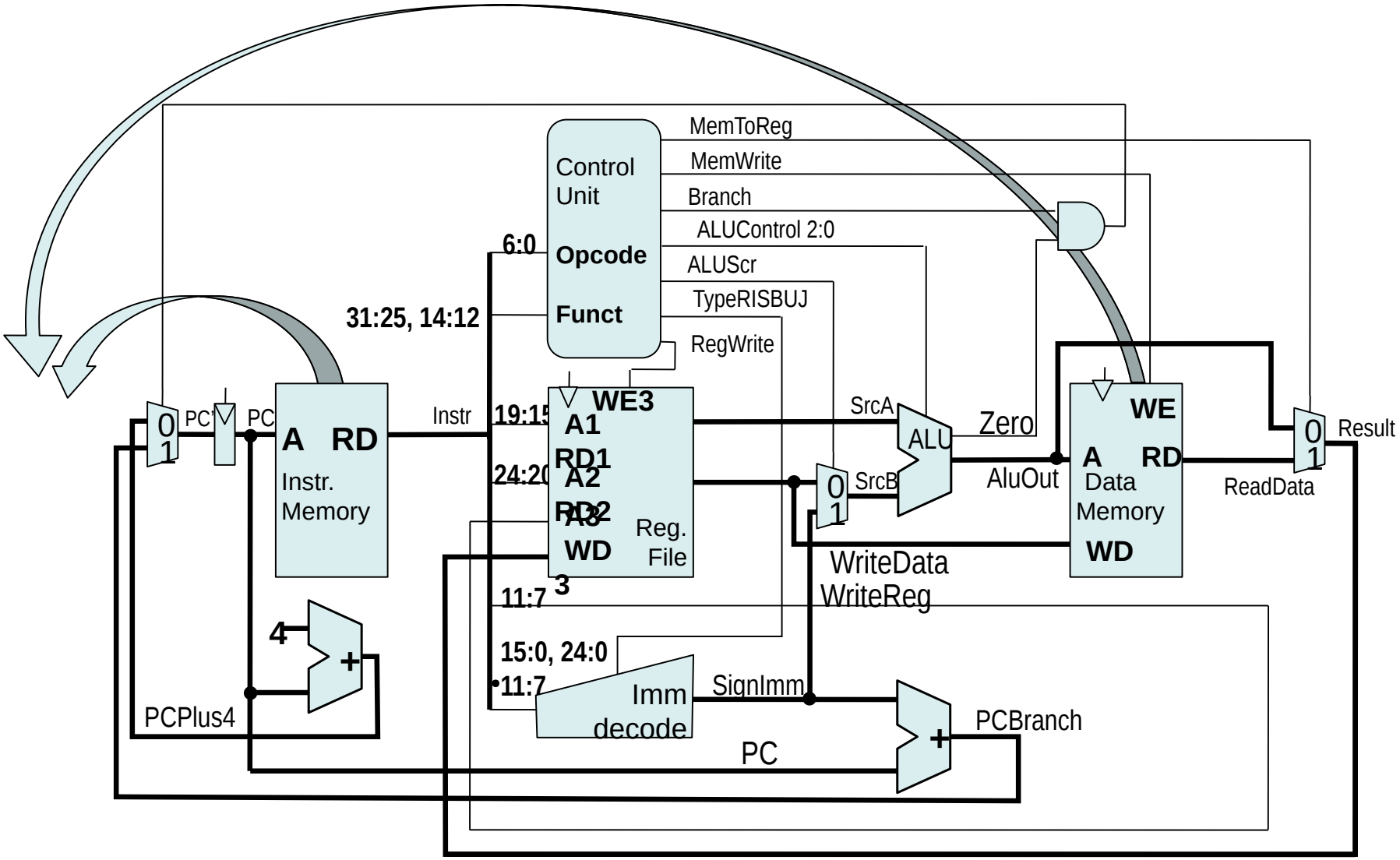


Propustnost navrženého zřetězeného procesoru

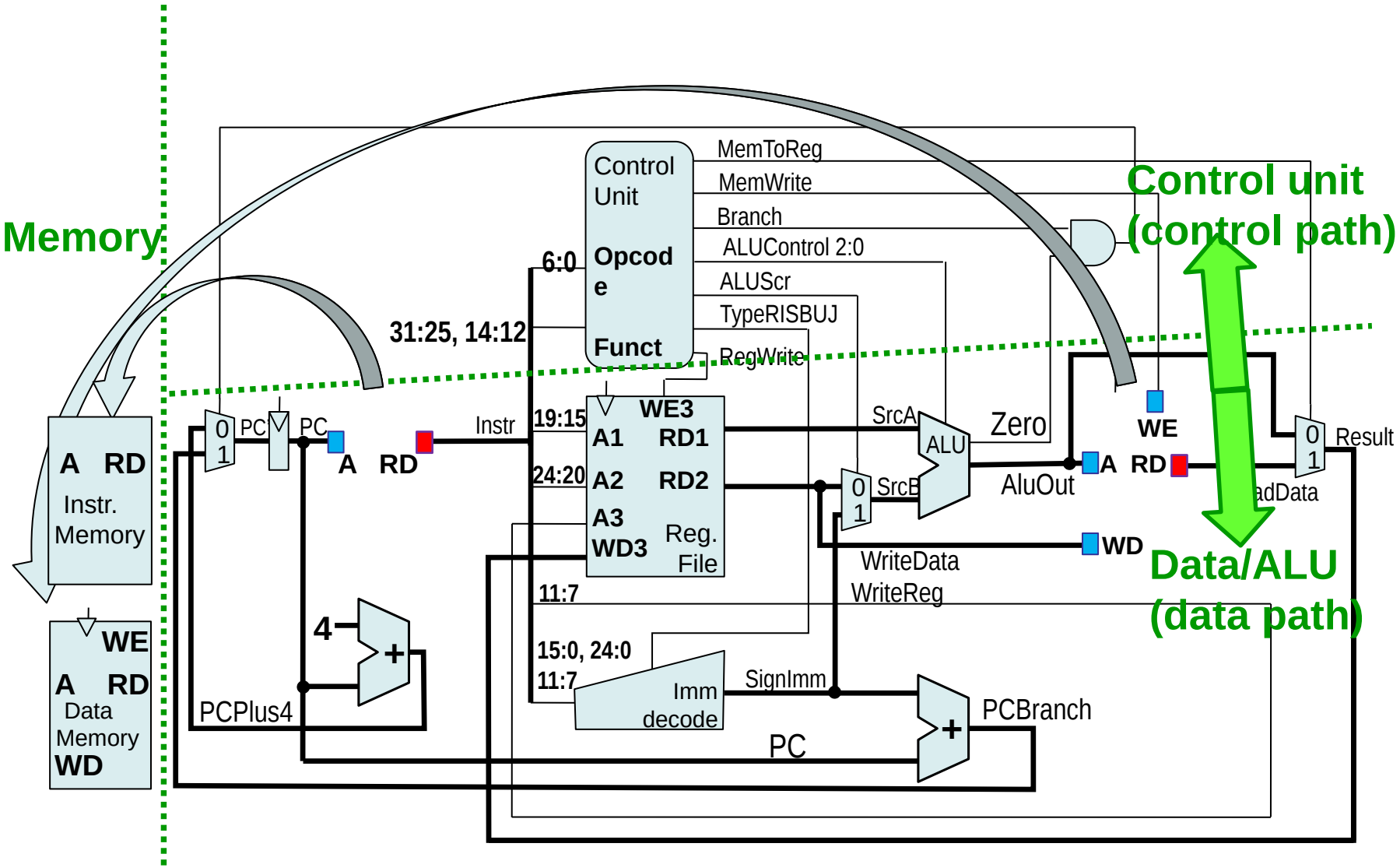
- Jaká bude maximální dosažitelná frekvence procesoru?
- Který stupeň je nejpomalejší?
- Minimální přípustná perioda cyklu je daná nejpomalejším stupněm
- V našem případě se jedná o paměti
  $T_C = 300\,\text{ns} \rightarrow f_{clk\,max} = 3\,333\,\text{kHz}$
  pokud neuvažujeme cykly navíc na plnění pipeline (žádná pozastavení a vyprázdnění například při skocích) tak pro ideální $IPC = 1$
  $IPS = 1 \cdot 3\,333e3 = 3\,333\,000$ instrukcí za sekundu
- Zavedení pětistupňového zřetězeného zpracování vedlo k zvýšení propustnosti $3\,333\,000/980\,000 = 3.4\times$ za předpokladu $IPC = 1$

Obsah

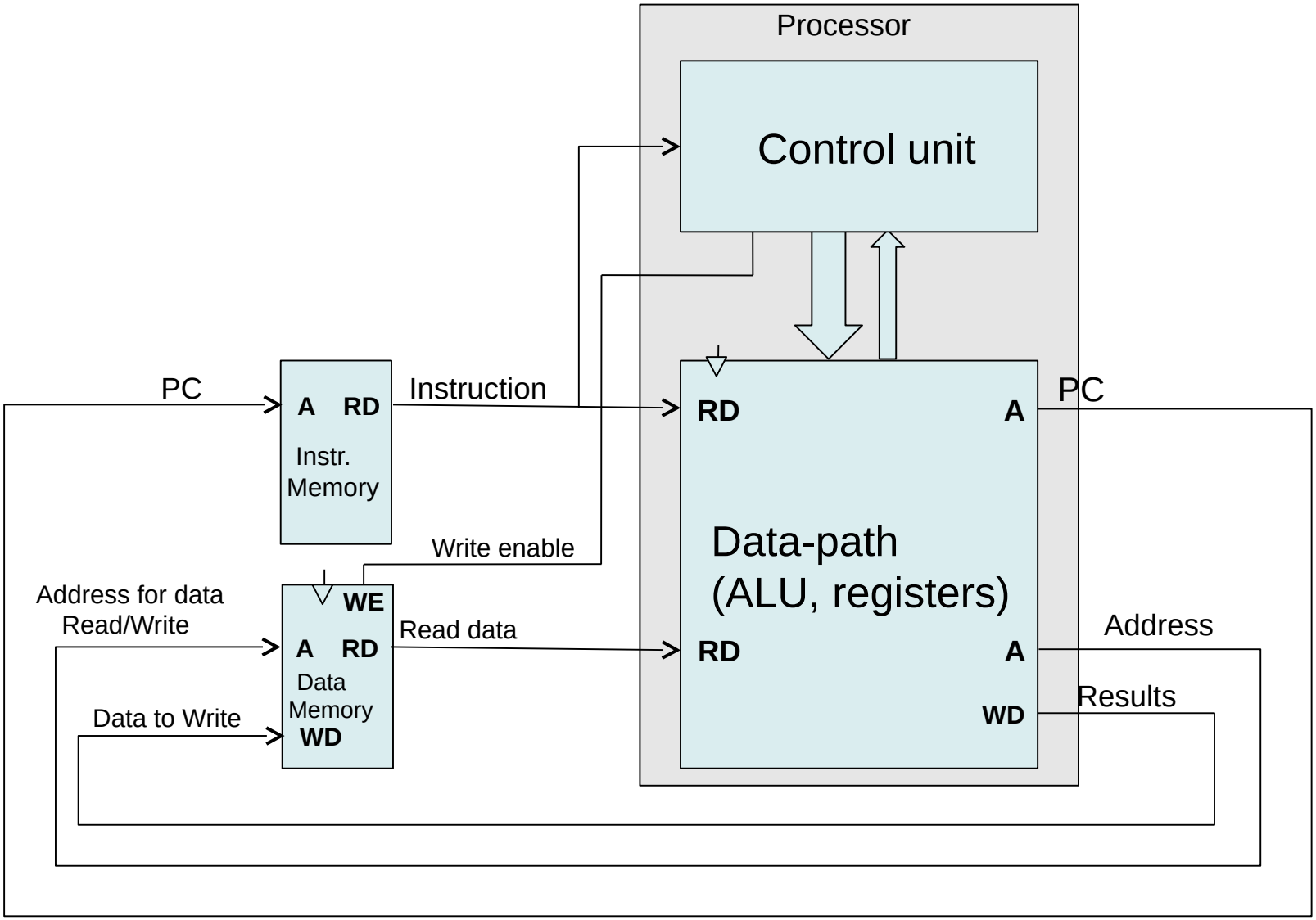
Jak koresponduje návrh s reálným procesorem s připojenou pamětí (pro zjednodušení neuvažujeme pipeline)



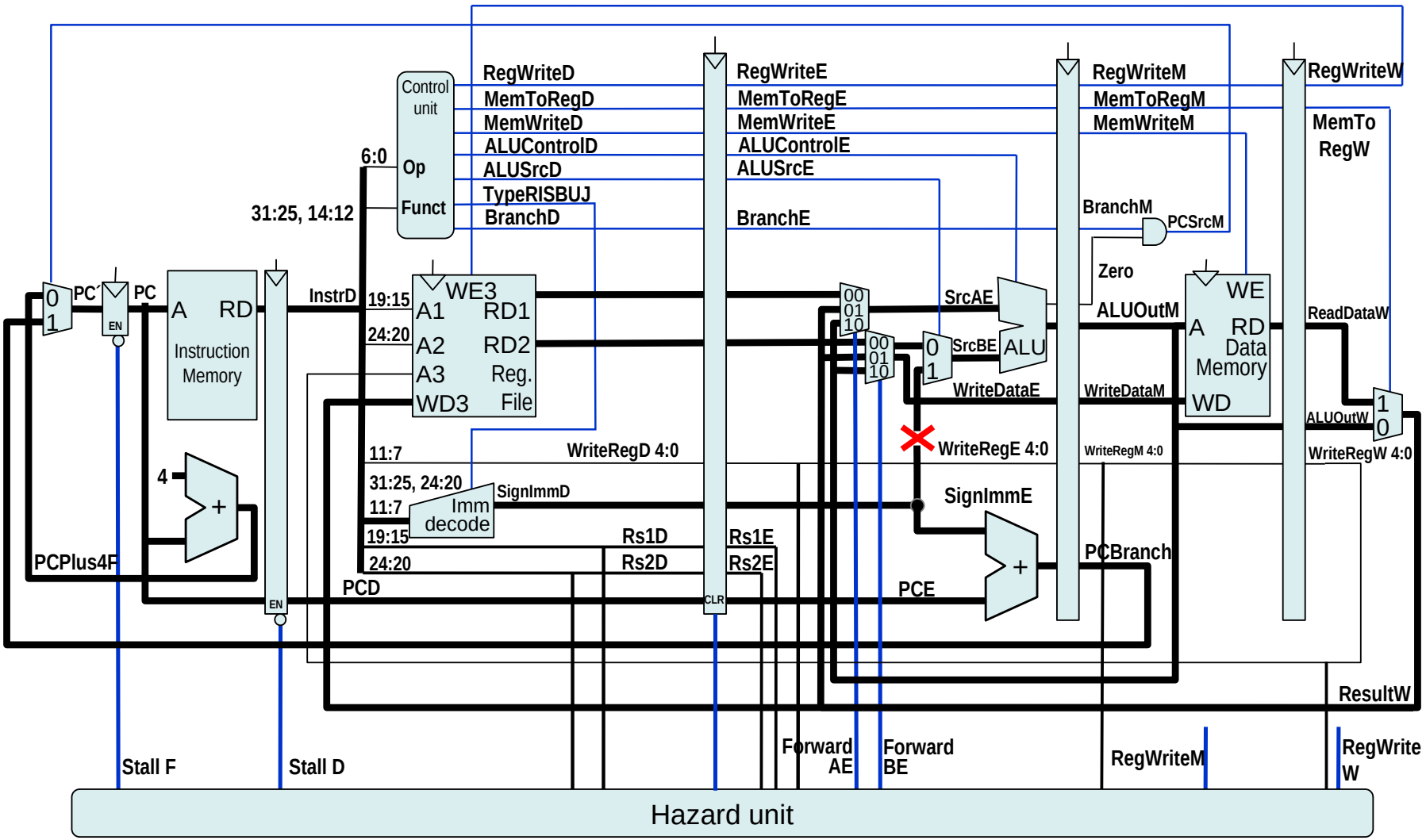
Pamět již nebudeme uvažovat na čipu



Vracíme se k původnímu rozdělení procesor, paměť



Která instrukce selže při přerušení spoje

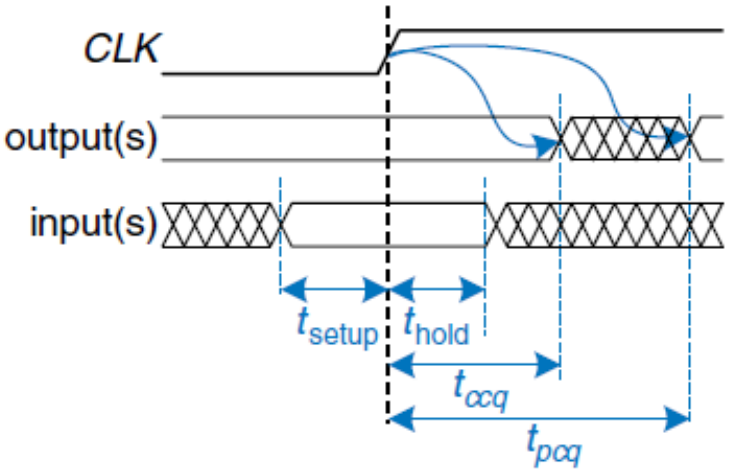


A) add **B)** addi **C)** beq **D)** or ?

Obsah

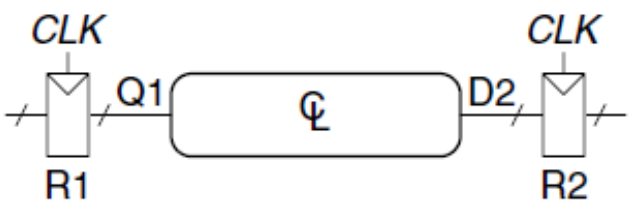
Requirements for Pipeline Timing

- The timing/AC characteristics of synchronous sequential circuit :

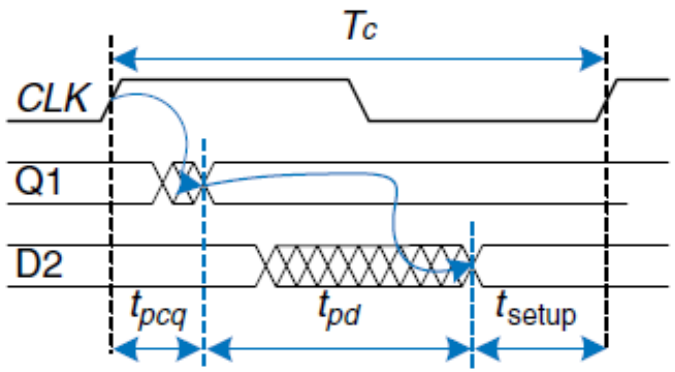


t_{setup} – inputs setup time
t_{hold} – inputs hold time

- Signal integrity constrain for the setup time before the clock:



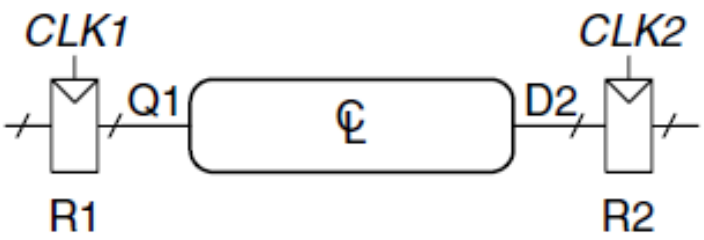
$$Tc >= t_{pcq} + t_{pd} + t_{setup}$$

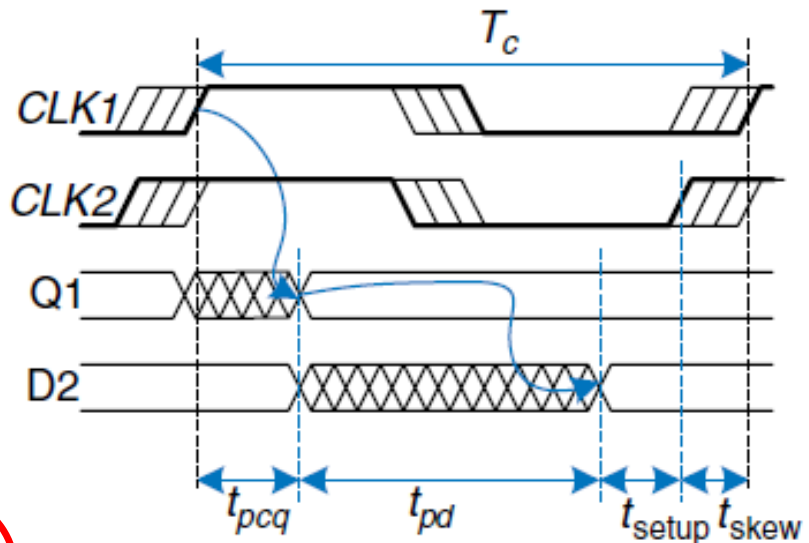


t_{pd} – combinatorial logic propagation delay

Requirements for Pipeline Timing

Constraint for the setup time (consider the clock distribution jitter):



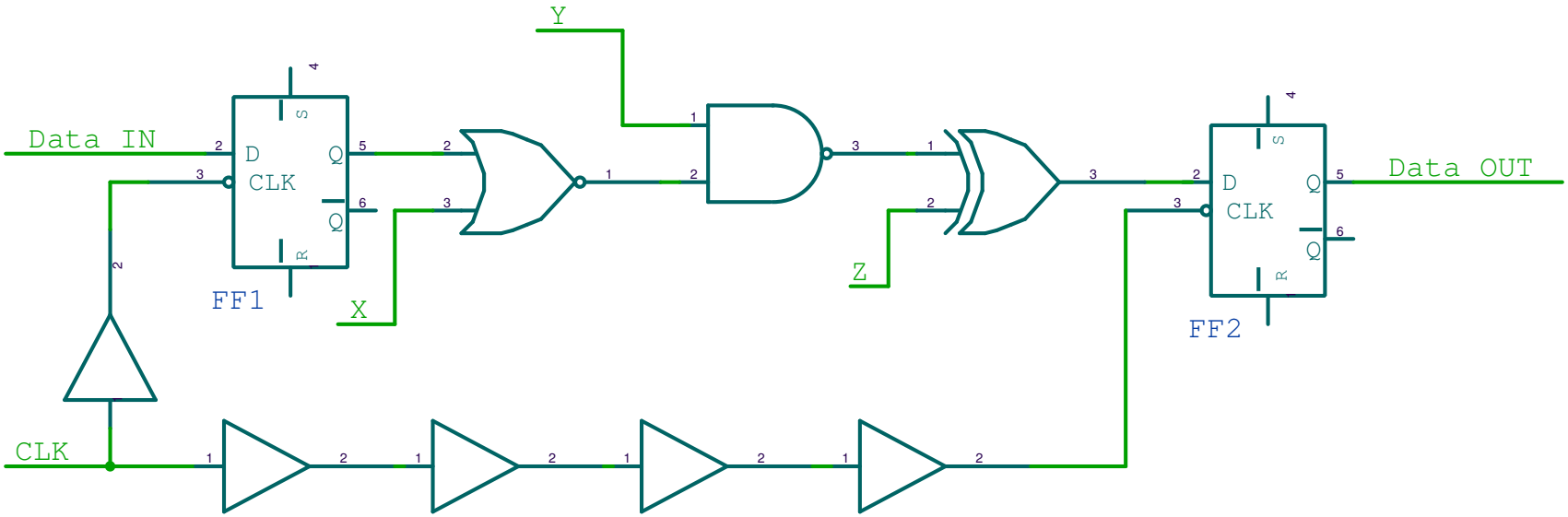


$$Tc >= t_{pcq} + t_{pd} + t_{setup} + t_{skew}$$

Clock distribution jitter is limiting factor,
if it reaches or exceeds value of t_{pd}

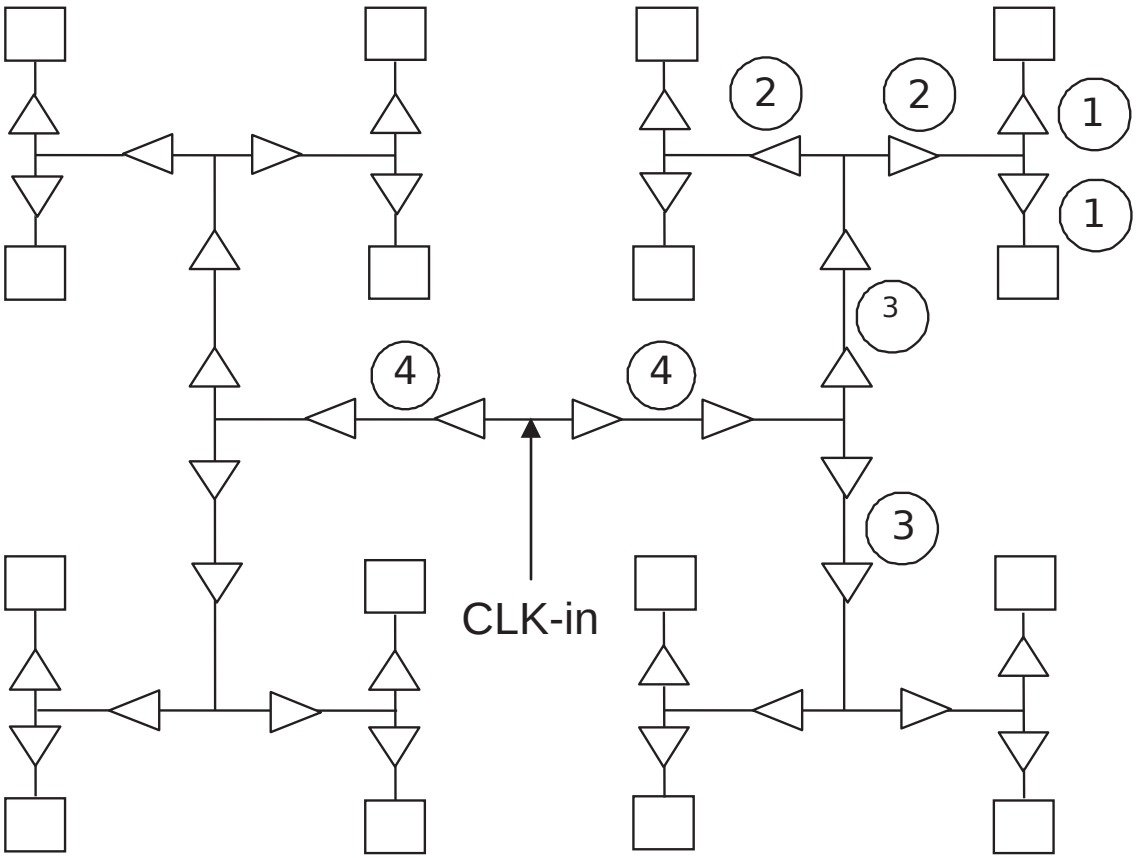
(too deep pipeline / too many stages…)

Clock Distribution Network Skew



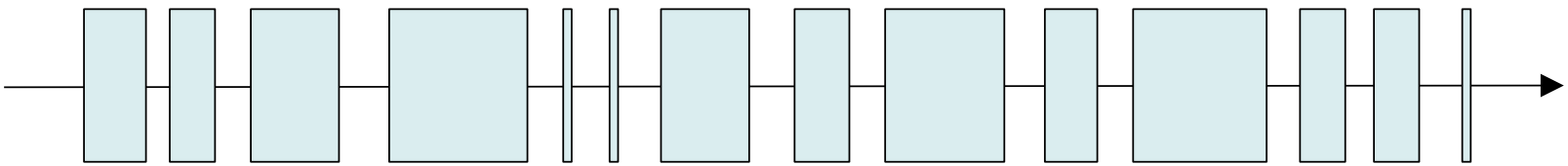
- **Positive Clock Skew** – clock arrives at the capturing sequential later than it arrives at the launching sequential
- **Negative Clock Skew** – clock arrives at the launching sequential later than it arrives at the capturing sequential
- **Local Clock Skew** – skew between any two sequentials with a valid timing path between them.
- **Global Clock Skew** – clock skew between any two sequentials in the design irrespective of whether a timing paths exists between them

Clock Distribution Network – H-tree



source: Tawfik, S., Kursun, V.: Clock Distribution Networks with Gradual Signal Transition Time Relaxation for Reduced Power Consumption.

Pipeline Stages Balancing

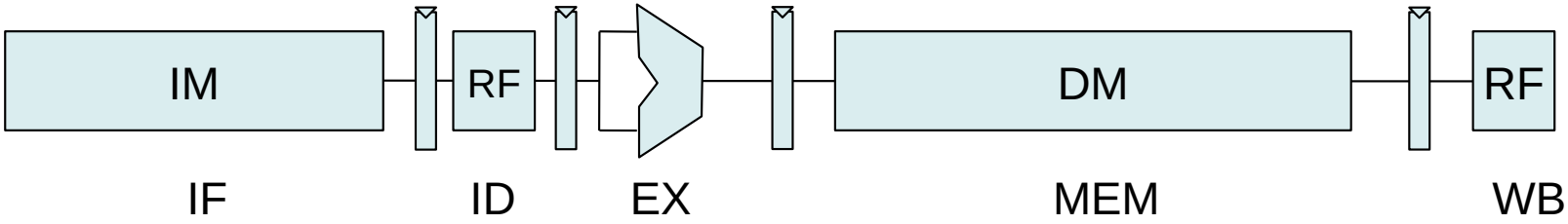


(applies to tree based adder, multiplier, (unrolled) iterative divider..)

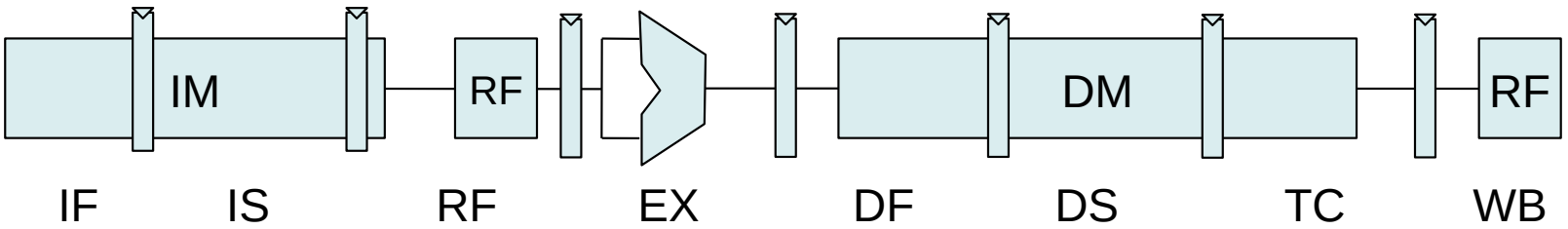
- **Balancing**: the goal is to divide the processing into N stages in such way, that stage propagation delays are roughly the same...
- The number of stages reflects preference of performance (throughput) versus latency.

Superpipeline and Beyond

- Not well balanced 5-stage pipeline:



- Deeper pipeline is result of decomposing stages into more stages



- It allows CPU to work at higher frequencies but introduces many problems as well...
- Complex forwarding, more pipeline stalls, hazards need to be solved by complex logic

Superpipeline and Beyond

P5 (Pentium) : **5**
P6 (Pentium 3): **10**
P6 (Pentium Pro): **14**
NetBurst (Willamette, 180 nm) - Celeron, Pentium 4: **20**
NetBurst (Northwood, 130 nm) - Celeron, Pentium 4, Pentium 4 HT: **20**
NetBurst (Prescott, 90 nm) - Celeron D, Pentium 4, Pentium 4 HT, Pentium 4 ExEd: **31**
NetBurst (Cedar Mill, 65 nm): **31**
NetBurst (Presler 65 nm) - Pentium D: **31**
Core : **14**
Bonnell: **16**

Haswell **14**-**19**
Cooper Lake **14**-19

K7 Architecture - Athlon : **10**-**15**
K8 - Athlon 64, Sempron, Opteron, Turion 64: **12**-**17**

AMD Zen **19**
AMD Zen2 **19**

ARM 8-9: **5**
ARM 11: **8**
Cortex A7 2-wide **8**-**10**
Cortex A8 2-wide **13**
Cortex A15 3-wide **15**-**25**

Cortex-A35 2-wide **8**
Cortex-A53 2-wide **8**
Cortex-A57 3-wide **15**
Cortex-A77 4-wide **11**–**13**
Denver 2/7-wide **13**
M4 6-wide **15**
Lightning 7-wide **16**

SiFive FE310-G000 **5**
SiFive FU540-C000 **5**

The Optimum Pipeline Depth for a Microprocessor:

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.93.4333&rep=rep1&type=pdf