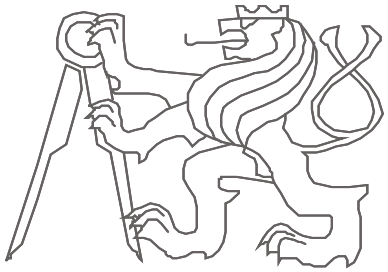
Architektury počítačů

Virtuální paměť II + Pipeline

1. duben 2019



České vysoké učení technické, Fakulta elektrotechnická

Jak využít virtuální stránky?

Vizualizace atria se studovnami

Technická knihovna v Dejvicích
- virtuální projekt

Technická knihovna v Dejvicích - realita

Virtuální paměť a soubory na disku…

- Virtuální paměť rozšiřuje fyzickou paměť o prostor na disku tím, že automaticky odkládá/načítá stránky na disk (swapování). Toho lze využít…
- **Načtení programů a knihoven do paměti:**
  - Programy a knihovny jsou uloženy na disku jako binární soubory obsahující instrukce a data
  - Když chceme spustit nový program:
    - Jádro OS alokuje souvislou množinu virtuálních stránek (dostatečně velký prostor pro uchování vlastního programu a dat)
    - Poté OS aktualizuje Page table procesu (Page tables pak odkazují na soubory na disku)
    - Položky Page table jsou označeny jako Valid=0 (na disku)
  - Jakmile program běží, správa virtuální paměti načte program do paměti automaticky…
  - Viz **mmap()** – funkce alokuje virtuální stránky a nastaví položky Page table tak, aby odkazovaly na soubor na disku

Jak využít cache a virtuální paměť

What Every Programmer Should Know About Memory

http://www.akkadia.org/drepper/cpumemory.pdf

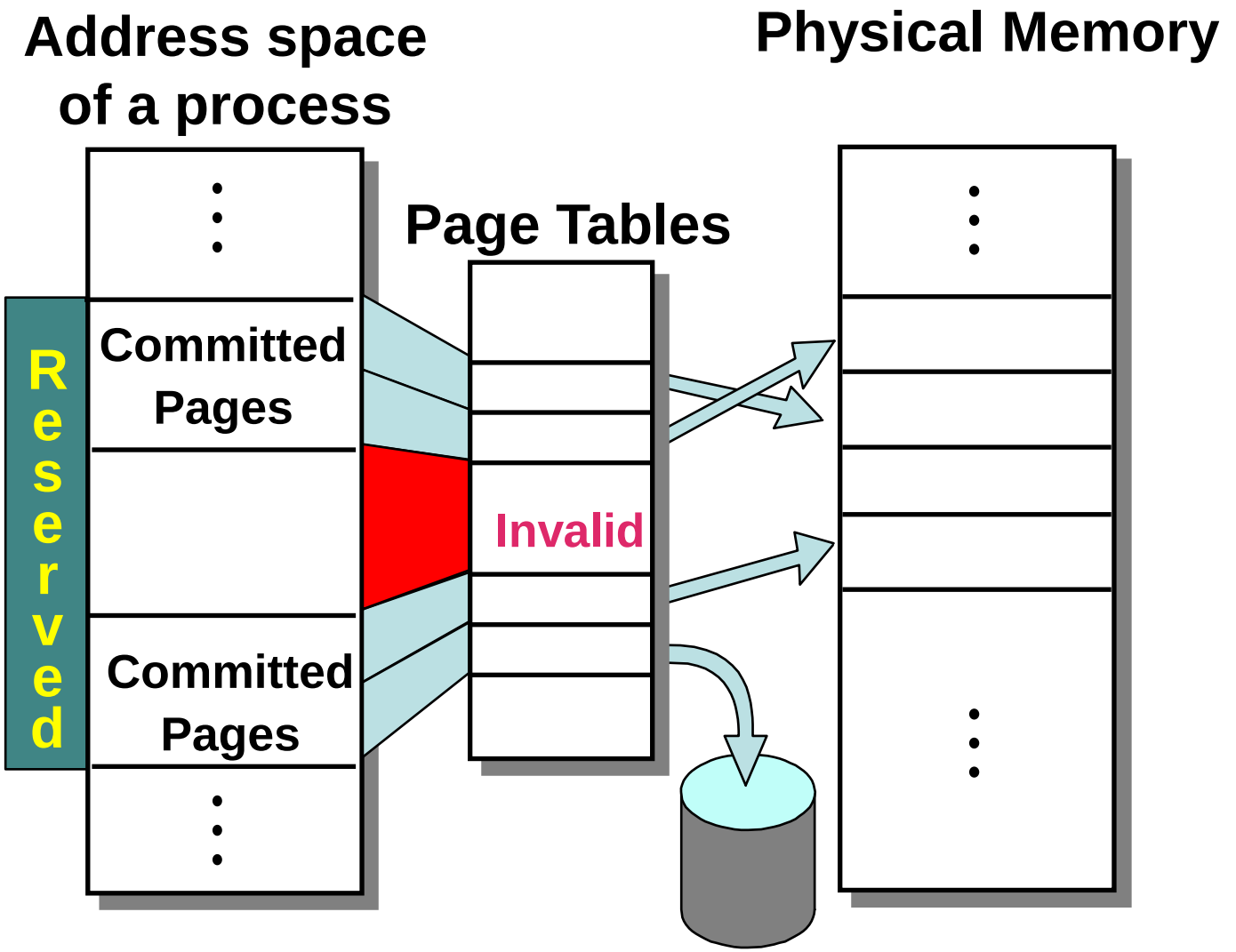
Poznámka

- Počáteční části popisují paměti i cache, a možno všem doporučit.
- Programátorské příklady ale cílí na velmi pokročilé uživatele, a začátečníky spíš zmatou.

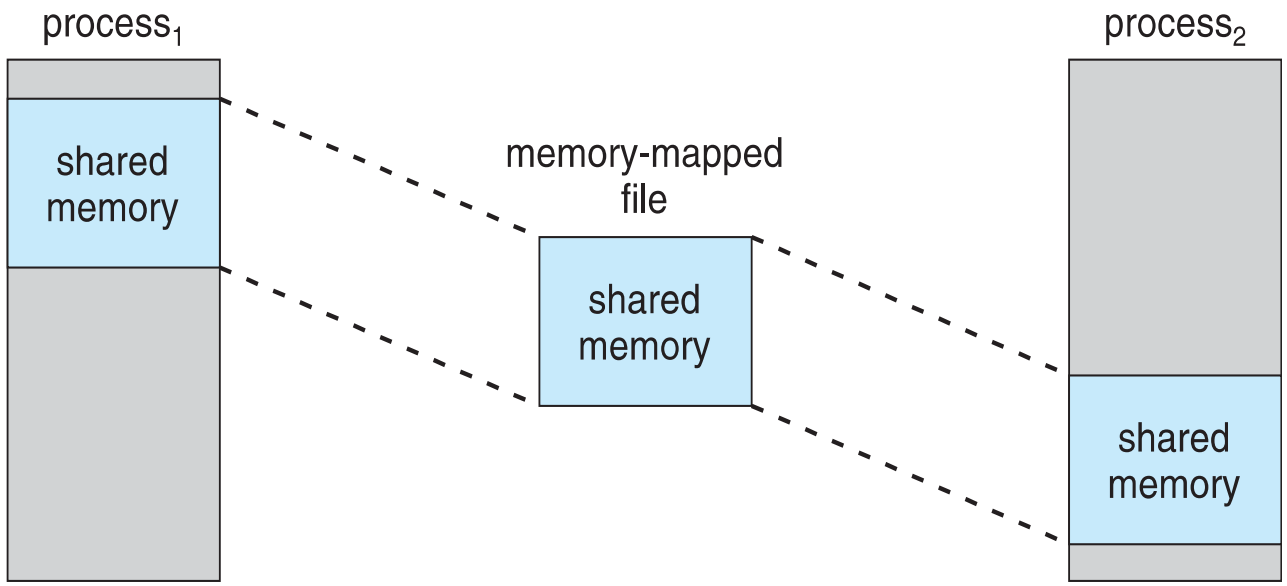
Virtuální paměť - stránkování

- Každé virtuální stránce může odpovídat nejvýš jedna fyzická stránka, obráceně to neplatí, takže:
- Na jednu konkrétní fyzickou stránku může být namapováno několik virtuálních stránek. Co to přináší?
- Můžeme sdílet paměť napříč různými procesy nebo vlákny (data nebo kód – OS načte sdílené knihovny jenom jednou), můžeme poskytnout jiná oprávnění (přístupová práva).
- Pokud se program snaží přistoupit do stránky způsobem, který neodpovídá jeho oprávněním, CPU generuje *General protection fault*
- handler pro General protection fault – typická reakce je ukončení procesu

Reserving and Committing of Address Space



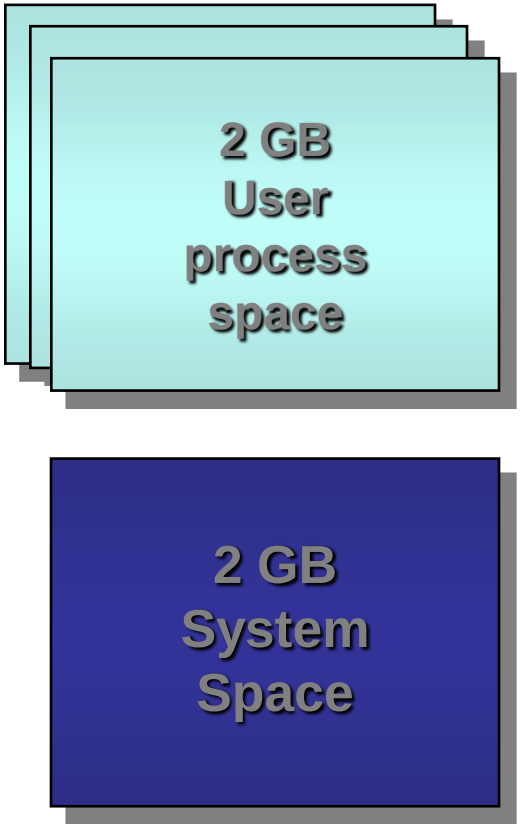
Shared Memory



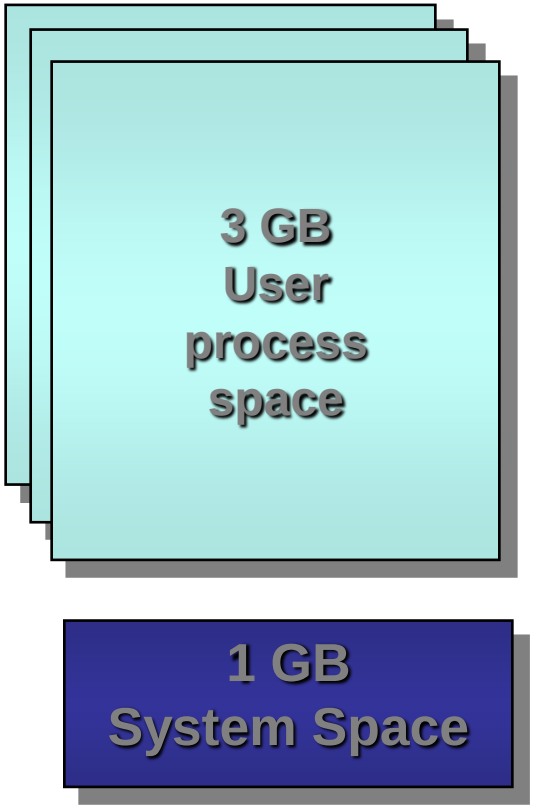
32-bit x86 Address Space

- 32-bits = 4 GB

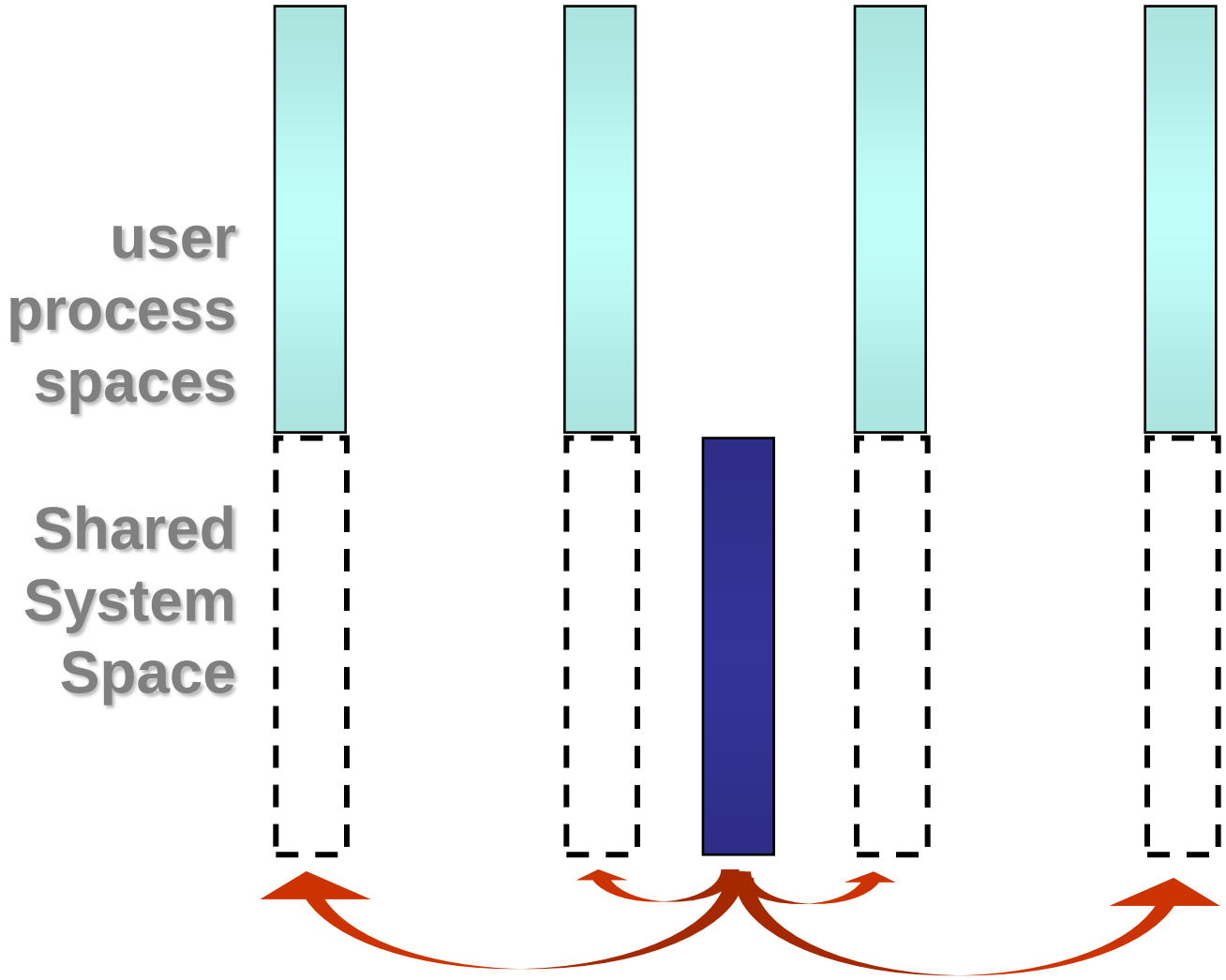
Default Windows



3 GB Linux user space or Extended Windows

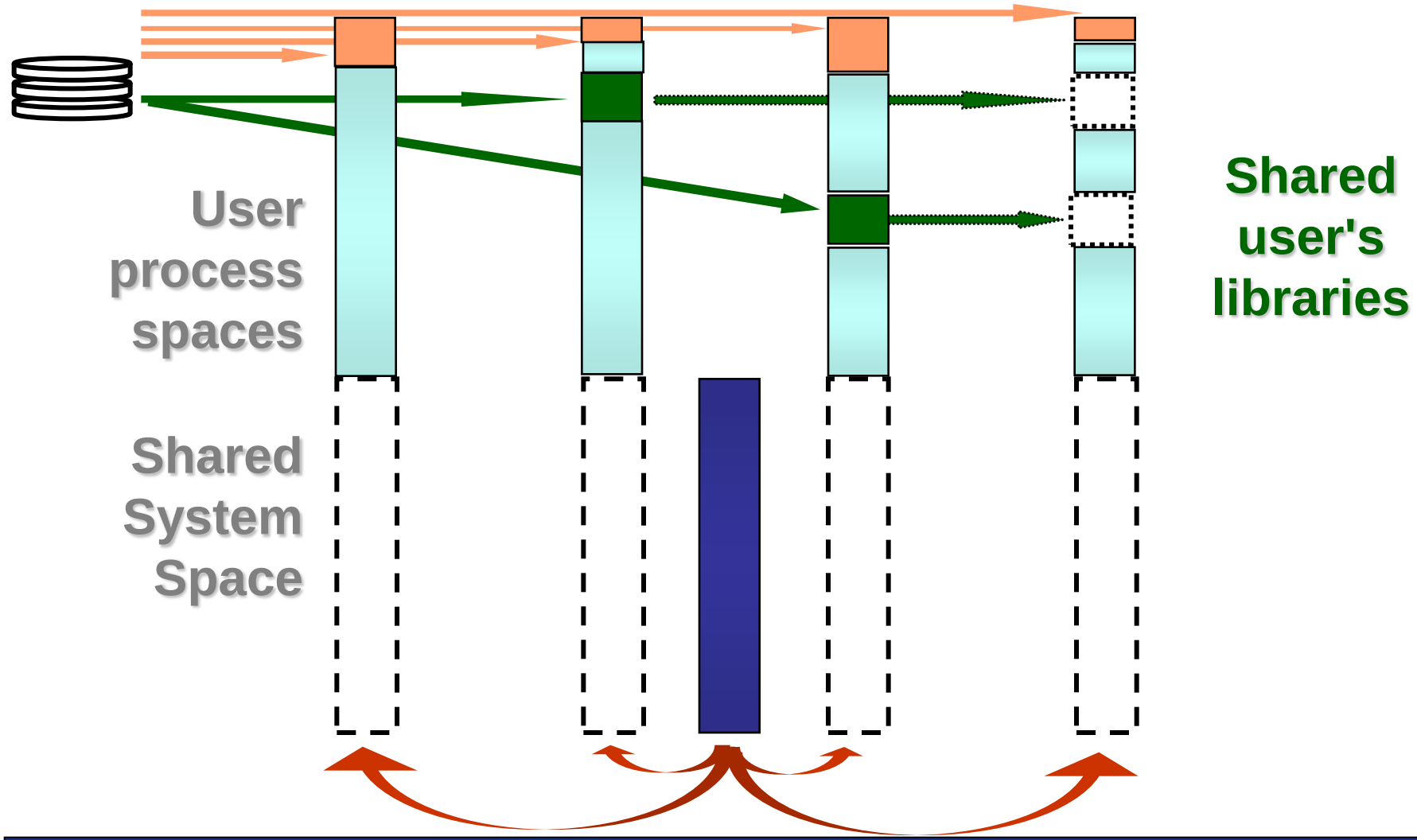


Address Spaces

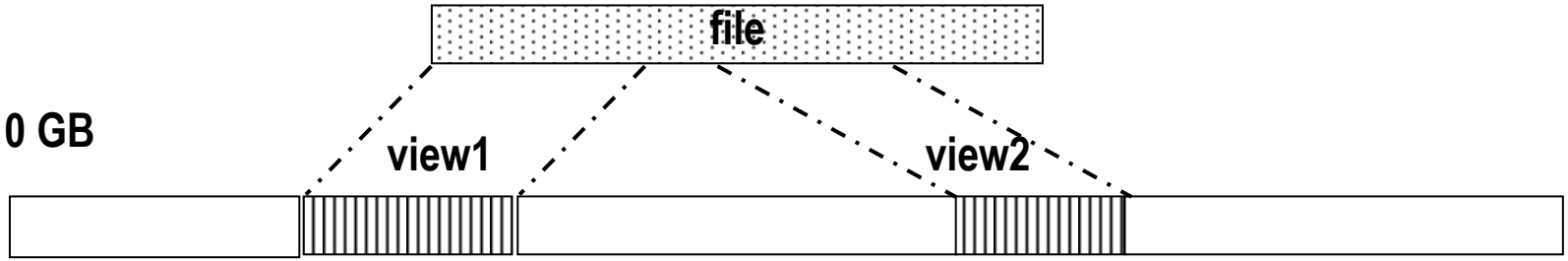


Address Spaces

Exe files and other read only files are only mapped into address spaces



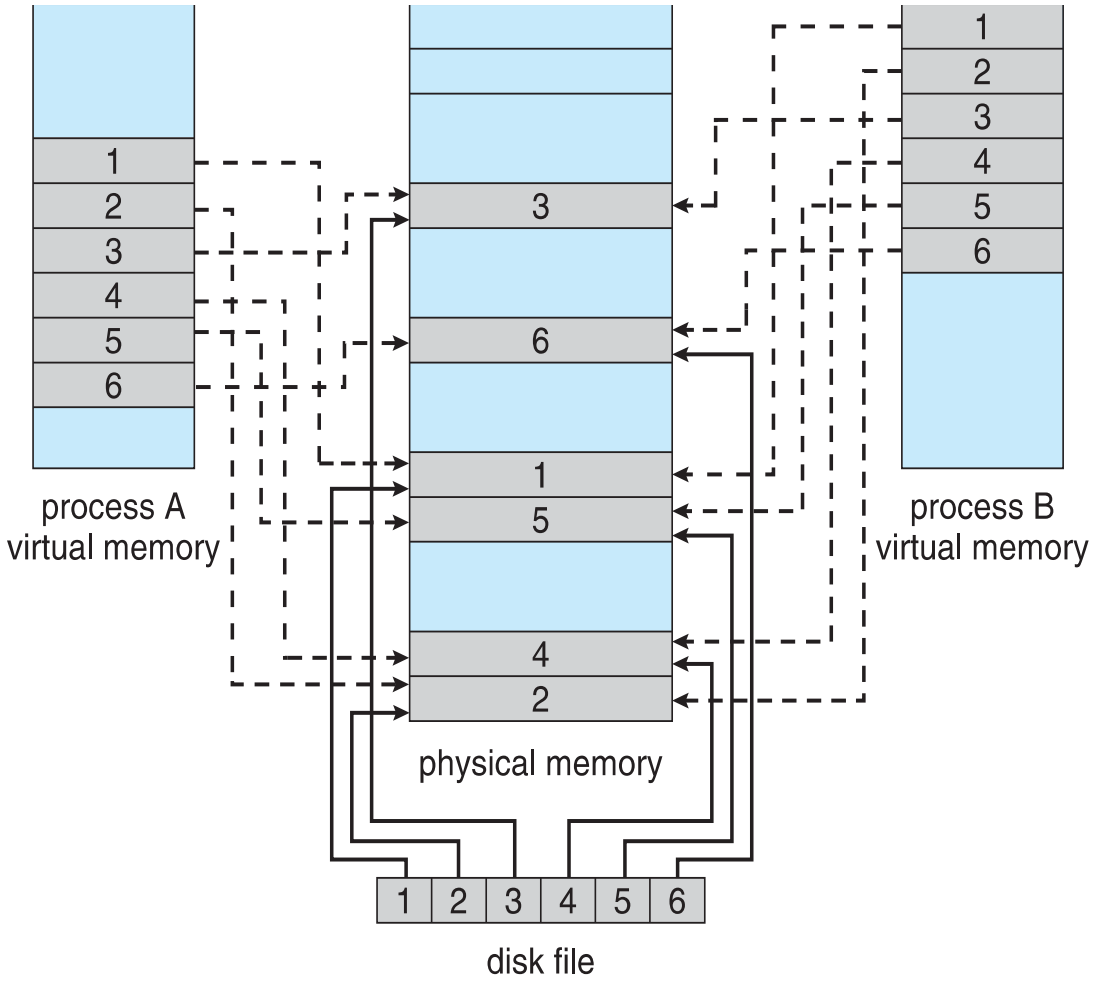
Memory Mapped Files



Notes:

- Memory mapped files are approximately from 1.5 to 3 times slower than file streams in case of **fully sequential reads or writes** because they need the construction of page table entries.
- Memory mapped files are much faster than file streams only if we need random access to file data. In such case, we can also gain benefits from cache.

Memory Mapped File



Mapping of Hardware in Linux

It is simplified part of the code that you use in your semester project

```
int fd = open("/dev/mem",  /* we ask for physical memory addresses */
              O_RDWR /* with read and write access */
              | O_SYNC /* and non-cached for /dev/mem */
              );
unsigned char *mem = (unsigned char *) mmap(
              NULL, /* kernel selects virtual address */
              0x4000 /* our required size*/,
              PROT_READ | PROT_WRITE, /* allow read and write*/
              MAP_SHARED, /* visible to other processes*/
              fd,  /* handle of an already opened file */
              0x43c40000 /* offset in file, here I/O physical base address*/
                                                    );
```

Note: For simplification, we have supposed that the size and offset are already align to page size.

Problémy hierarchických pamětí

Některé problémy hierarchických pamětí?

- Koherence pamětí. Definice viz další slajd
- Jednoprocesorové (jednojádrové) stroje.
  - Řeší D-bit a migrační strategie Write-back.
- Multiprocesory se společnou i sdílenou pamětí – řešení je mnohem složitější. Používá se mj.
  - Společná sběrnice: Snooping (s odposlechem, slíděním), MESI protokol,
  - Broadcast (s rozesíláním),
  - Directories (adresáře).
- Je obsahem předmětu A4M36PAP.

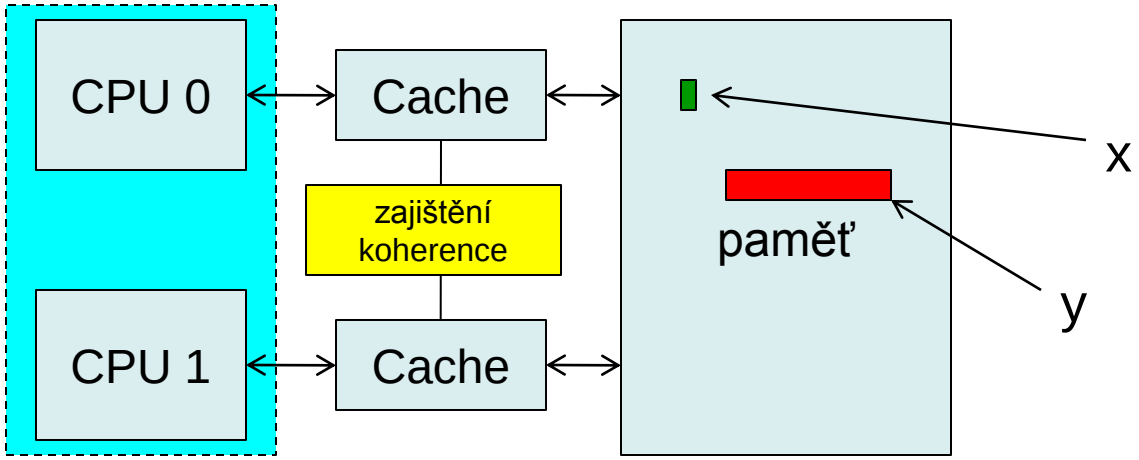
Definice koherence

- Řekneme že multiprocesorový **paměťový systém je koherentní,** jestliže výsledek jakéhokoli provádění programu je takový, že pro každé paměťové místo je možné sestavit myšlené sériové pořadí čtení a zápisů k tomuto paměťovému místu, a platí:

- 1. Paměťové operace k danému paměťovému místu pro každý proces jsou provedeny v pořadí, ve kterém byly spuštěny tímto procesem.

- 2. Hodnoty vracené každou operací čtení jsou hodnotami naposledy provedené operace zápis do daného paměťového místa vzhledem k sériovému pořadí.

Problém koherence



Proto je důležité, aby byl systém paměťově koherentní – viz *cache coherence*

Nicméně i v paměťově koherentním systému může nevhodný programátorský styl vést k značnému zpomalení běhu programu…

Příklad A:

Vlákno 0:	Vlákno 1:
…	…
x=1;	x=3;
…	…
if(x==1)	

Příklad B:

Vlákno 0:	Vlákno 1:
…	…
y[1]=1;	y[0]=0;
…	…
y[3]=3;	y[2]=2;
…	…
y[5]=5;	y[4]=4;

Nechť x je sdílená proměnná, y sdílené pole.

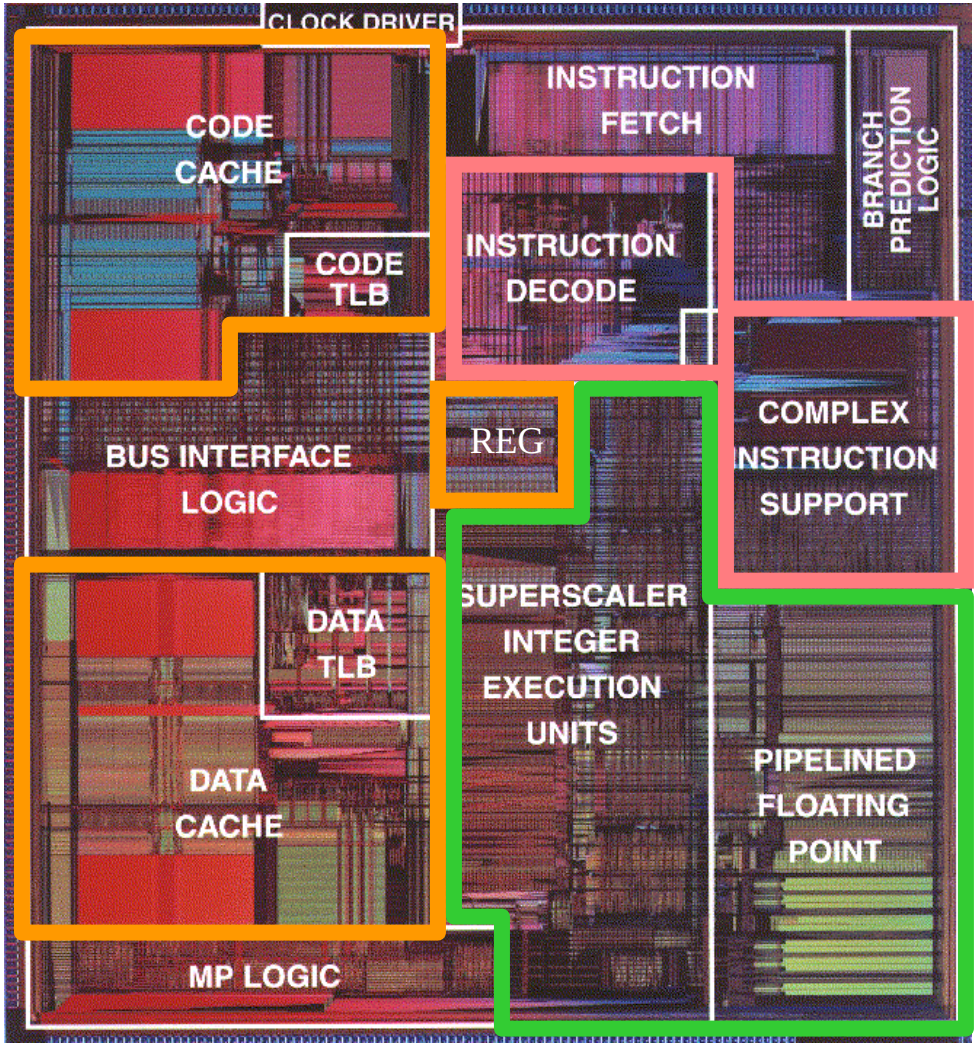
Srovnání **Virtual memory** versus **Cache**

Virtuální paměť TLB	**Cache**
Stránka	Blok/řádek
Page Fault	Read/Write Miss
Velikost stránky: 512 B – 8 KB (4kB)	Velikost bloku: 8 – 128 B (64B)
Plně asociativní, N-cestná	(DM), N-cestná, plně asociativní
Výběr oběti: LRU	LRU, ARC, CAR
Write Back	Write Back

LRU - Last Recently Used
ARC - Adaptive Replacement Cache (LRU+LFU)
CAR - Clock (improvement of FIFO) with Adaptive Replacement

- Pozn.: TLB virtuální paměti může být plně asociativní, ale pro větší TLB typicky bývá jen 4-cestná.
- Rozumíte pojmům?
  - Co je oběť?
- Závěr: každé adjektivum vyjadřuje něco jiného…

Single Core - Intel Pentium



MP Logic = Multiprocessing logic for bus arbitration and cache coherency

Source: Intel

Aprilum-Cat® procesory pro řešení DCOT

Definice DCOT

Don't Care-Output Tasks

Definice DCOT

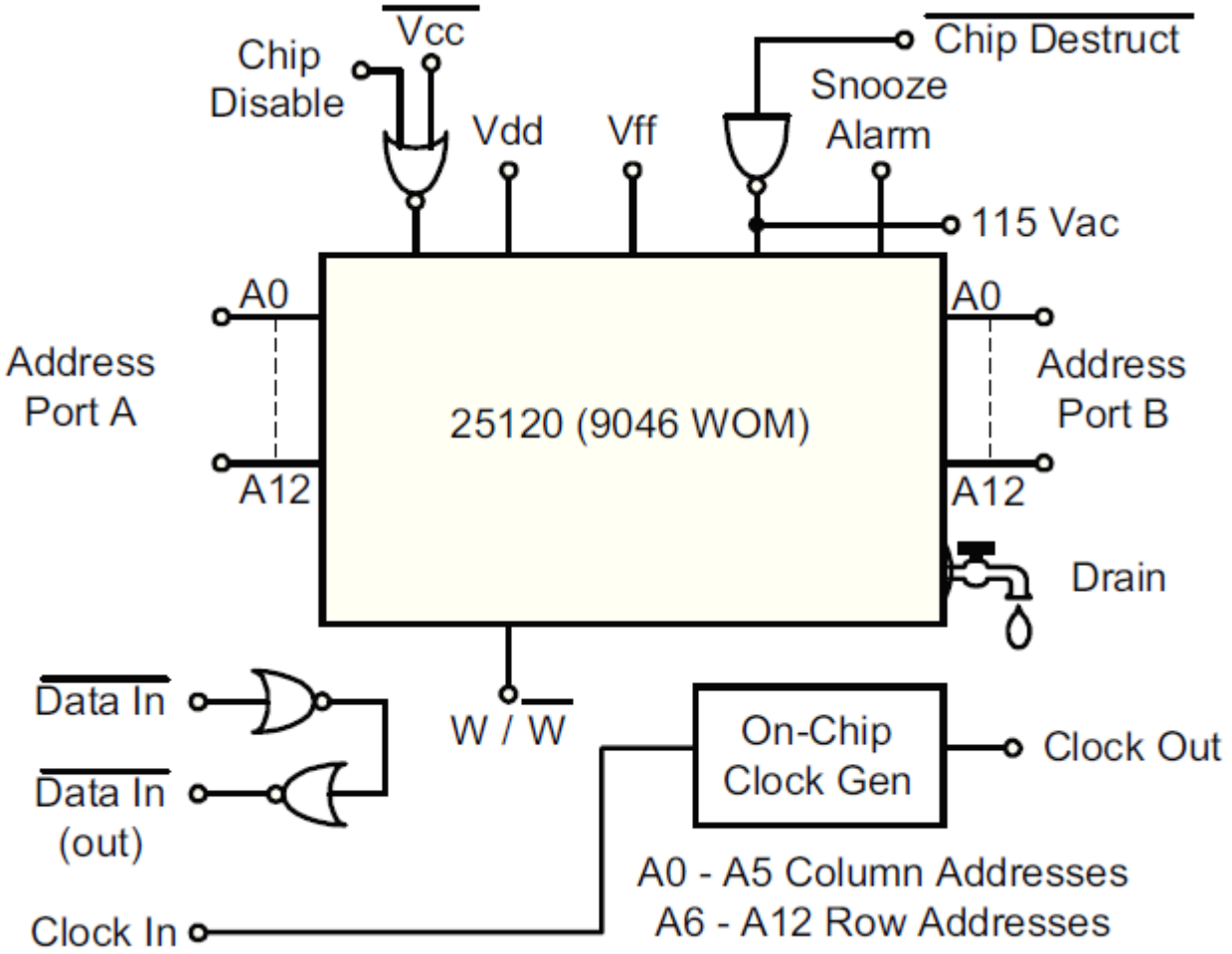
- celá úloha či její část, v níž se počítá něco, co nikoho nezajímá, avšak nelze to nespočítat.
- Musí se tedy vykázat náročnost k dosažení celkového zdárného vyřešení,
- ale nikdo si výsledek nikdy pořádně nepřečte.
- DCOT výpočty zatěžují životní prostředí zvýšenou spotřebou energie, což vede k nárůstu globálního oteplení,
- a tak potřebujeme způsob šetrnější k životnímu prostředí.

Uložení výsledků DCOT

- starší způsob WOM, Write-Only Memory, *viz* *Signetics 25120 WOM datasheet*

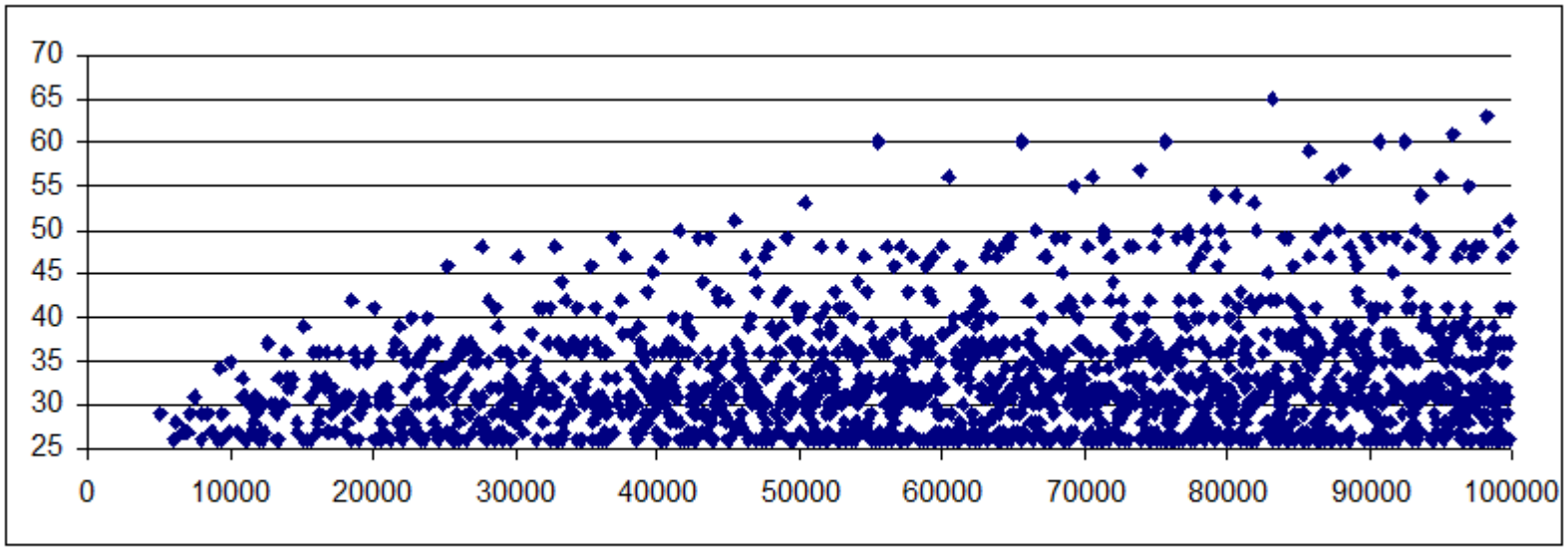
Nevýhody:

- potřeba střídavého žhavícího napětí V^{FF}=~6.3 V
- nutnost i Drain odvodu pro časté "memory leaks"



Palindromická paměť pro DCOT

- další varianta je využití palindromů ke snížení paměti pro DCOT úlohy, *viz např. Kolář J.: Čísla palindromická z pohledu výpočetní techniky, Palindromická konference 2002 PAKO OKAP 2002, ČVUT 2002*
- *Uložení palindromu potřebuje jen poloviční paměť, takže opakováním palindromizace lze teoreticky dosáhnout i ekologického bezpaměťového počítače. Zde bude však nutný ještě další výzkum.*



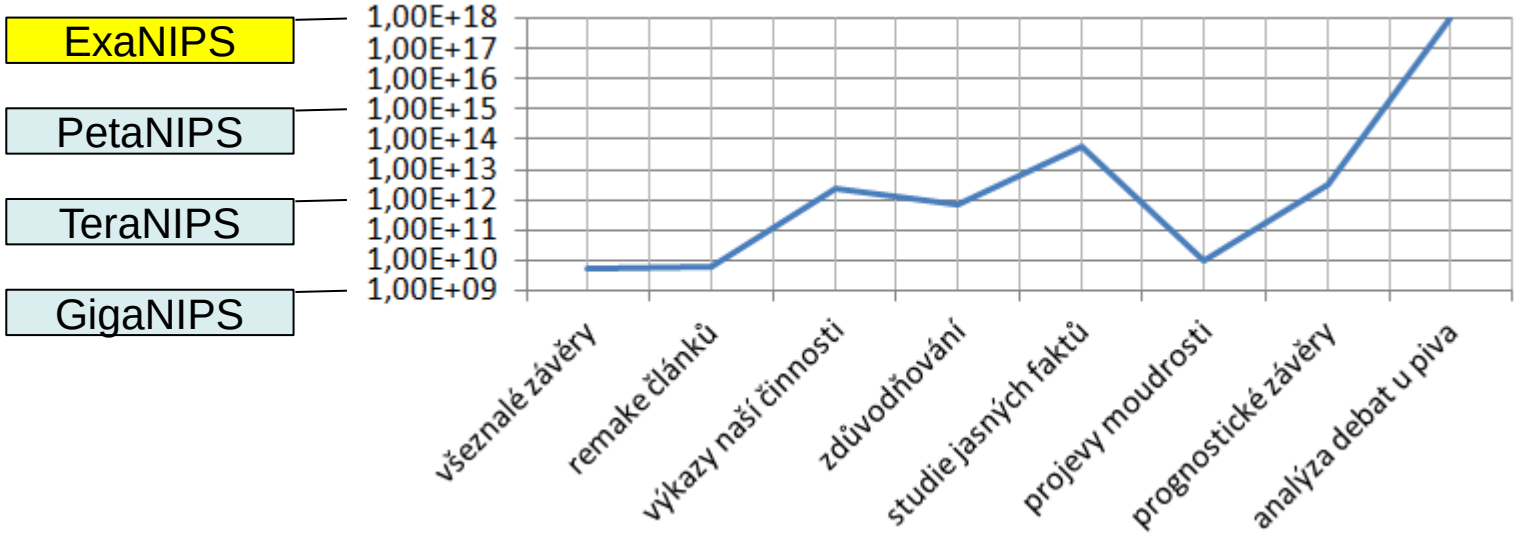
Obr. 1 – Palindromické váhy prvních 100 000 přirozených čísel. (větší než 25)

Procesory Aprilum-Cat® **se SISC jádrem**

Praktičtější způsob se opírá o neuronové překladače, které v úlohách DCOT části, ty pošlou je do SISC jádra (Single Instruction Set Core) a poté jen vygenerují výsledky. V SISC jádru:

- úplně odpadá instrukční cache a lze vynechat i datovou a stránkování,
- čímž se dosáhne extrémně vysokého zrychlení výpočtu,
  které se leckde blížící až Exa-NIPS [Nop-Instructions Per Second]

SISC jádro bude zahrnuto v procesorech řady Aprilum-Cat®



Předběžné benchtesty úloh na prototypech Aprilum-Cat®

* Pipeline

Source: Chinesse Press Automation

Další část dnešní přednášky

- Modifikujeme jednocyklový procesor z 2. přednášky na zřetězený procesor.
- Procesor bude podporovat instrukce:
  `add, sub, and, or, slt, addi, lw, sw` a `beq`

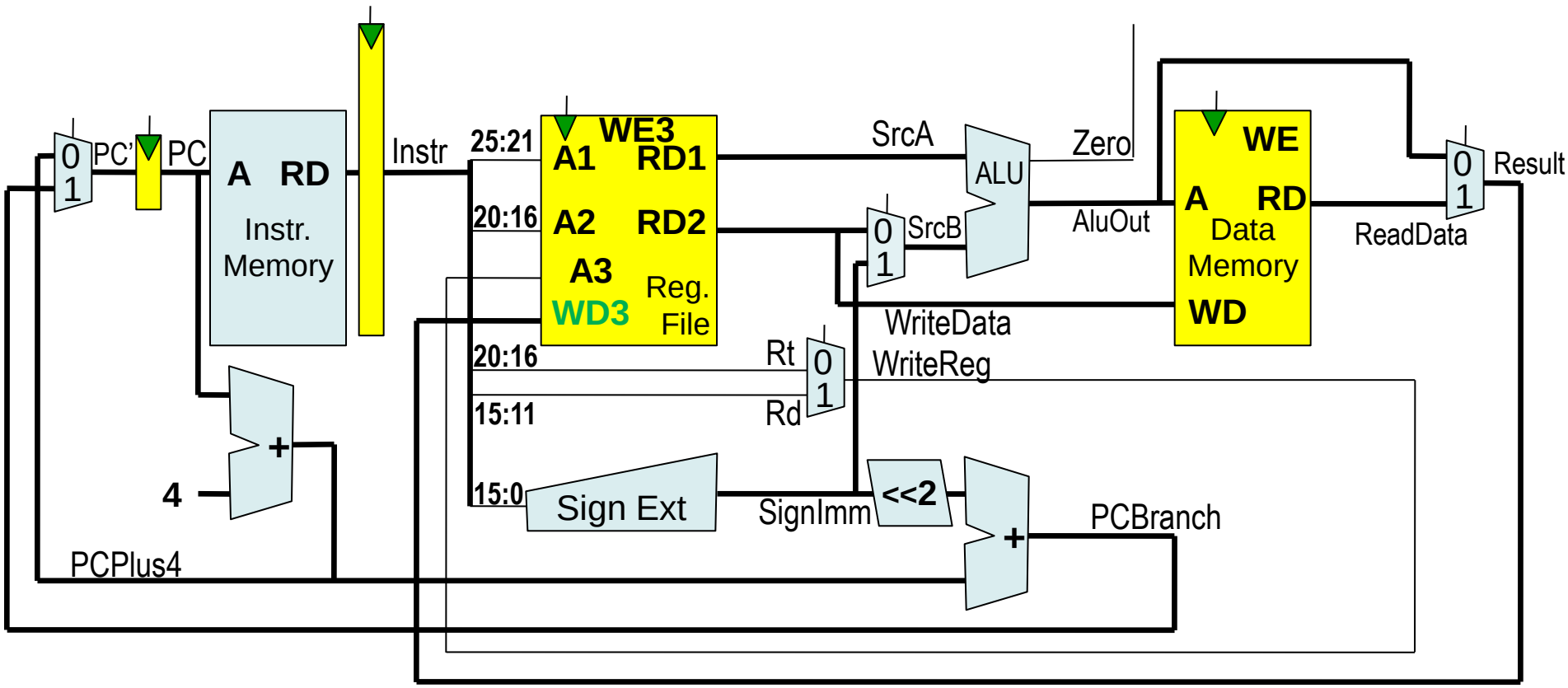
Typ	31…					0
R	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**rd**(5), 15:11	**shamt**(5)	**funct**(6), 5:0
I	**opcode**(6), 31:26	**rs**(5), 25:21	**rt**(5), 20:16	**immediate** (16), 15:0		
J	**opcode**(6), 31:26	**address**(26), 25:0				

Nezřetězené vykonávaní instrukcí

I nezřetězený MIPS procesor má dva cykly:

1. Instruction Fetch – poslání PC do paměti a vybrání aktuální instrukce. Aktualizace PC = PC+4
2. Vlastní provedení instrukce

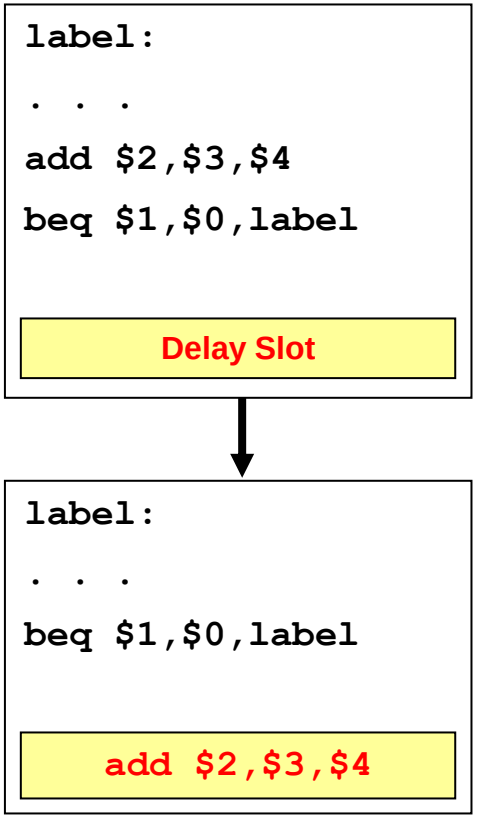
Nezřetězené vykonávání



v obrázku označuje hodinový vstup reagující na náběžnou hranu

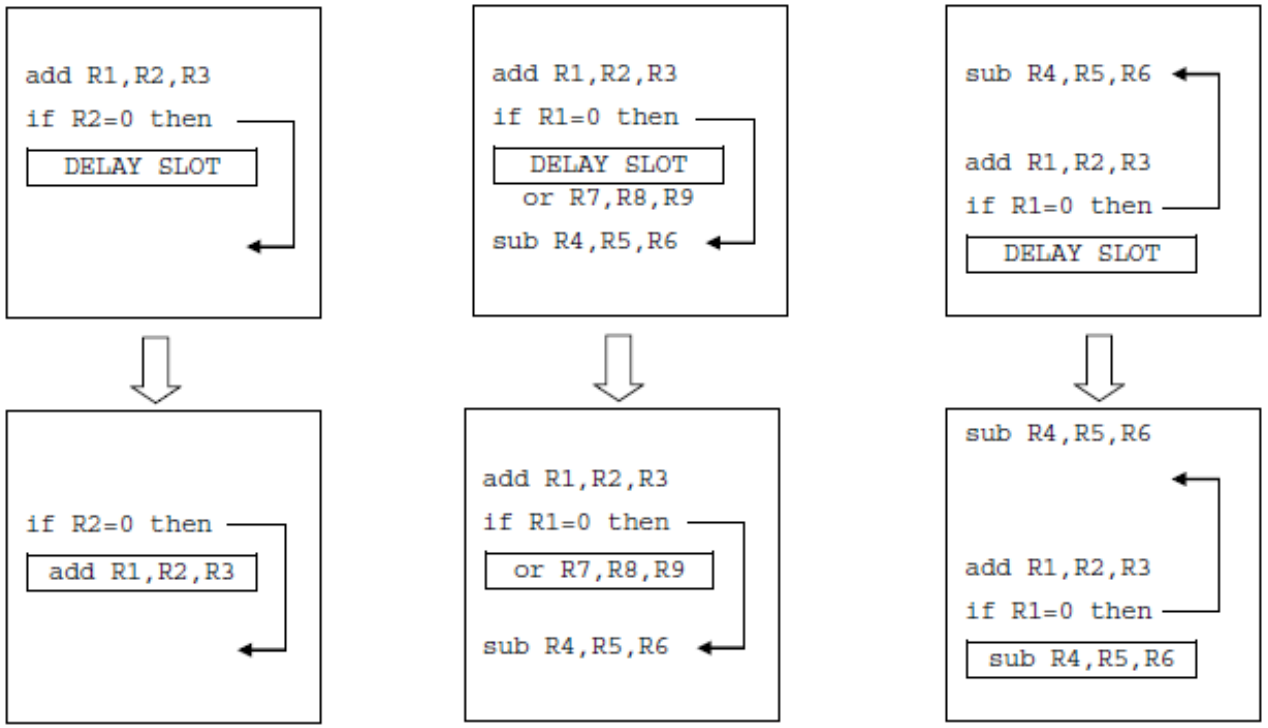
Delay Slot

- Define branch to take place after the next instruction
- MIPS defines one delay slot
- Compiler fills the branch delay slot
  - By selecting an independent instruction from before the branch
  - Must be okay to execute instruction in the delay slot whether branch is taken or not
- If no suitable instruction is found
  - then the compiler fills delay slot with a NOP



Delay slot…

- Úkolem kompilátoru je tedy zabezpečit, že následující instrukce nacházející se v **branch delay slotu** budou platné a užitečné. Na níže uvedených obrázcích jsou ilustrovány tři alternativy jak je možné plnit delay slot.

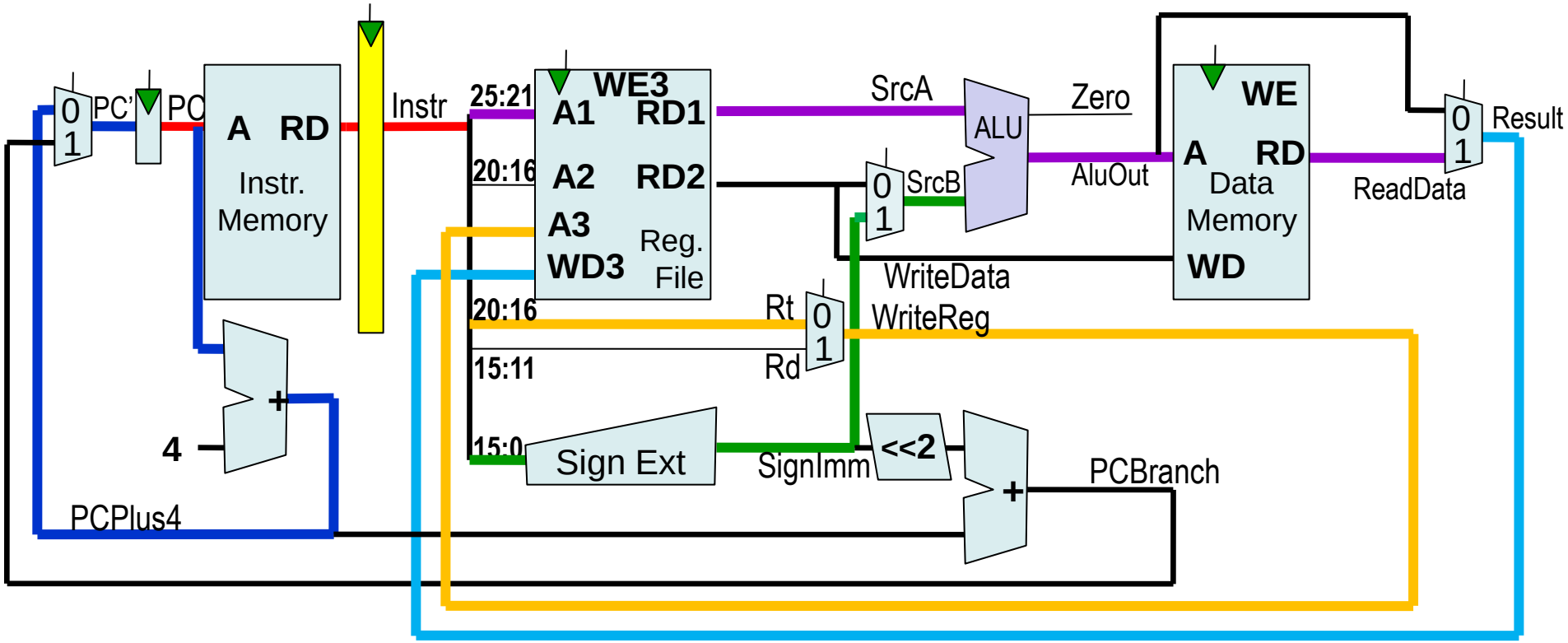


Nejsnadnější způsob (zároveň nejméně efektivní) spočívá ve vyplnění delay slot prázdnou instrukci – nop.

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

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

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



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

- Předpokládejme:

t_{PC}	= 30 ns	t_{Mem}	= 300 ns
t_{RFread}	= 50 ns	t_{ALU}	= 200 ns
t_{Mux}	= 20 ns	$t_{RFsetup}$	= 20 ns

Při T_{fetch} prováděném paralelně s $T_{processor}$,
jelikož je vždy 30+300=T_{fetch} < $T_{processo}$ = 50+200+300+20+20 = 590 ns = 1.69 MHz -> **IPS = 1 690 000**

Zřetězené vykonávaní instrukcí

Předpokládejme, že i vykonání instrukce můžeme rozdělit do dalších stupňů:

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

a dále $\tau = \max \{ \tau_i \}^k_{i=1}$, 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

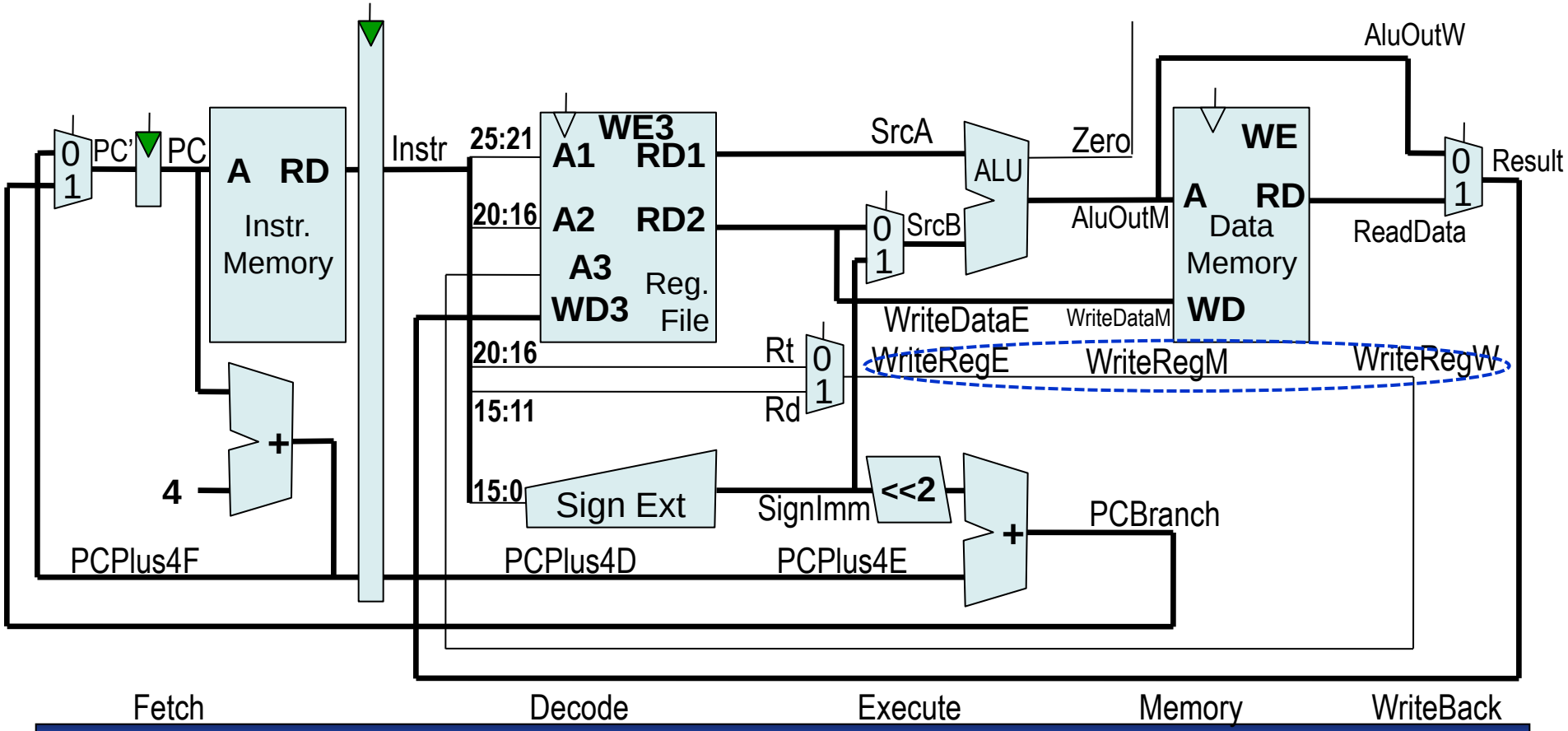
ID – dekódování instrukce a načtení registrů specifikovaných v instrukci, provedení testu na rovnost registrů (kvůli možnému větvení), znaménkové rozšíření offsetu, výpočet cílové adresy pro případ větvení (zn. rozš. offset + PC)

EX – operace ALU

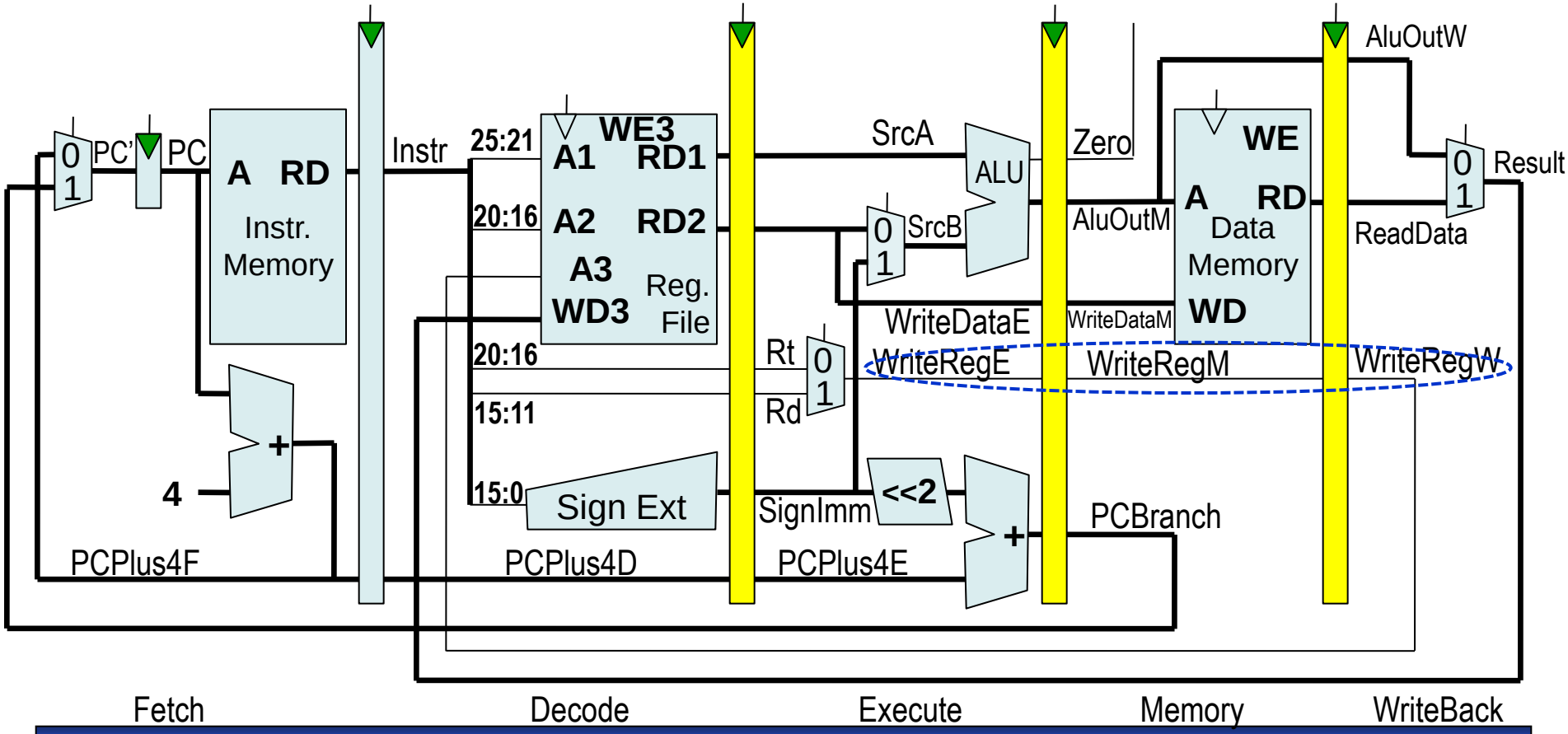
MEM – v případě instrukce *load* /store – čtení/zápis do paměti

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

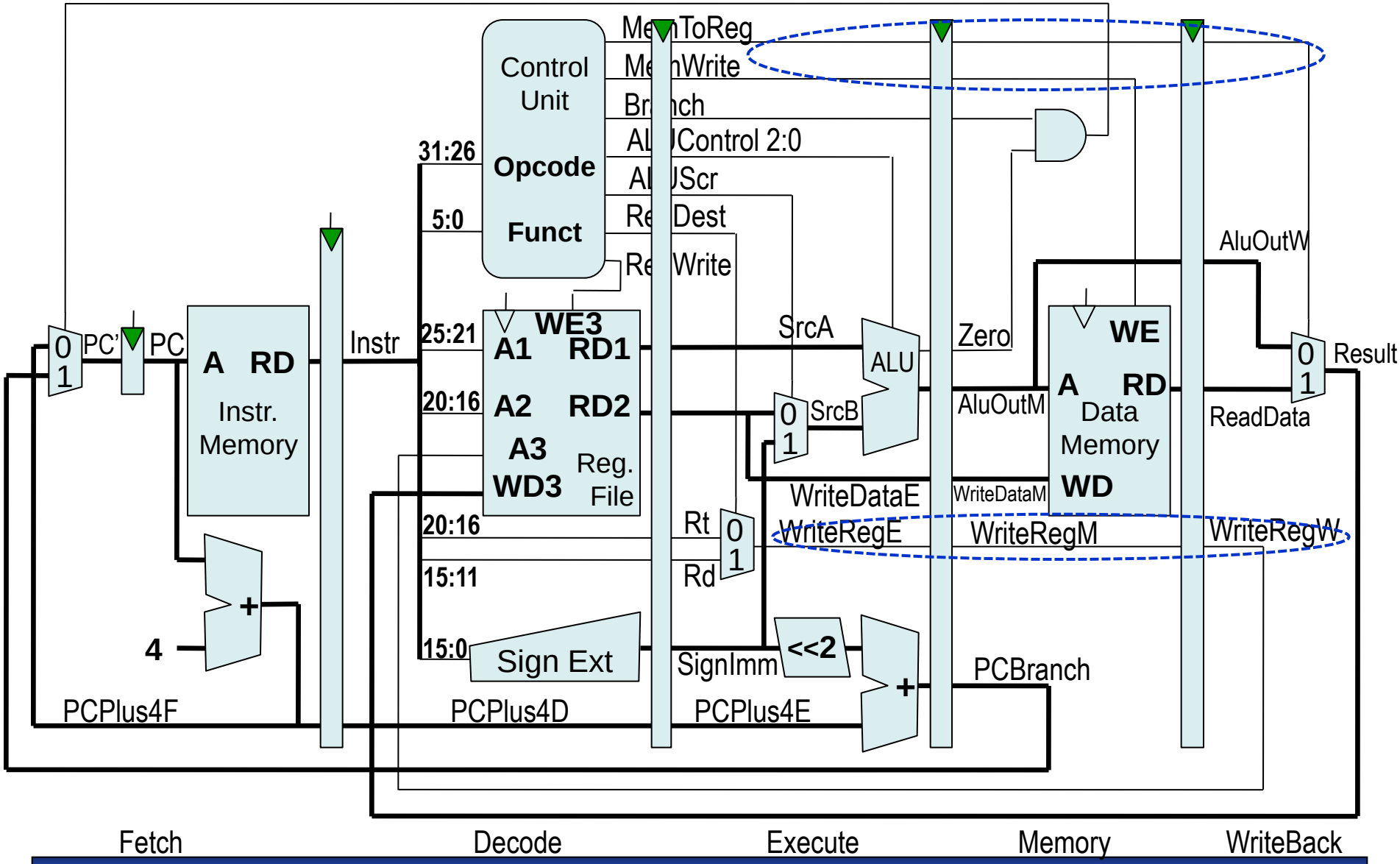
Nezřetězené vykonávání



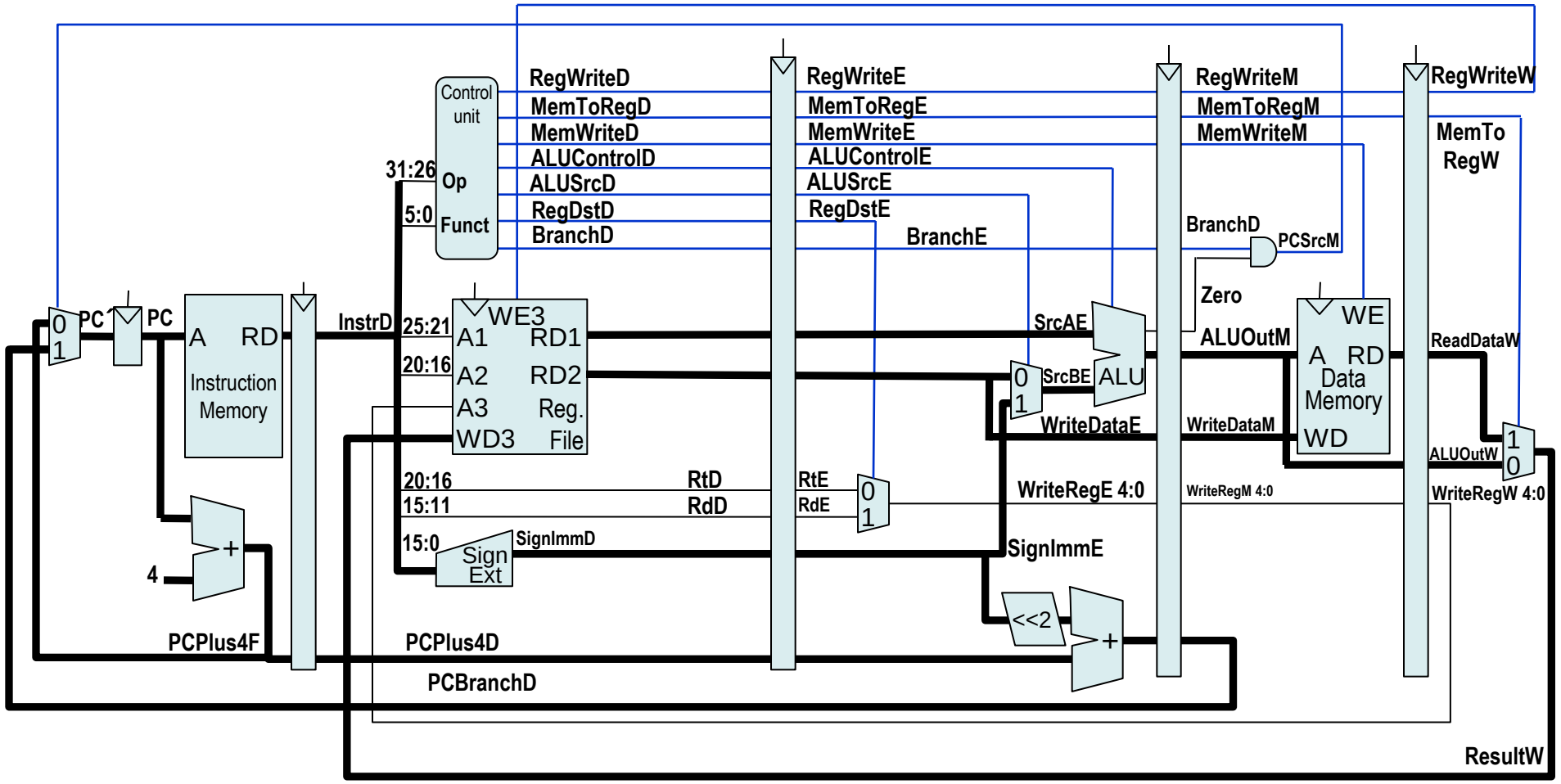
Zřetězené vykonávání



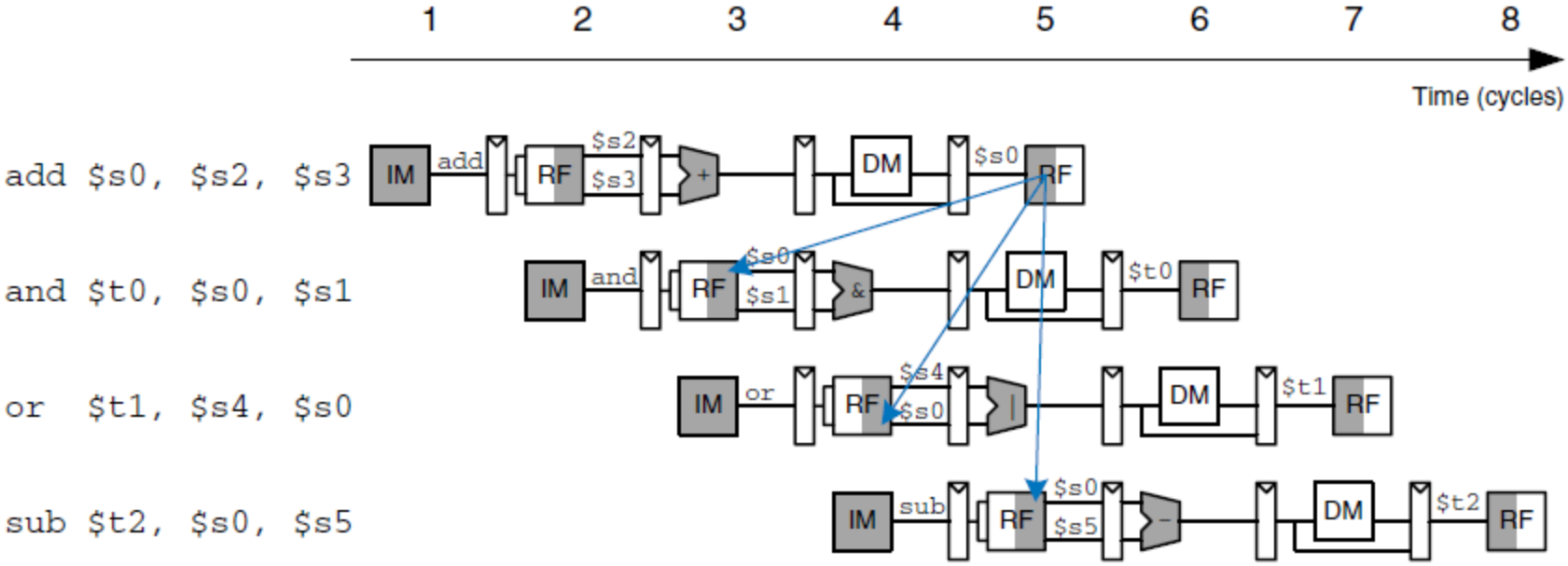
Zřetězené vykonávání



Totéž, pouze zmenšeno a překresleno…

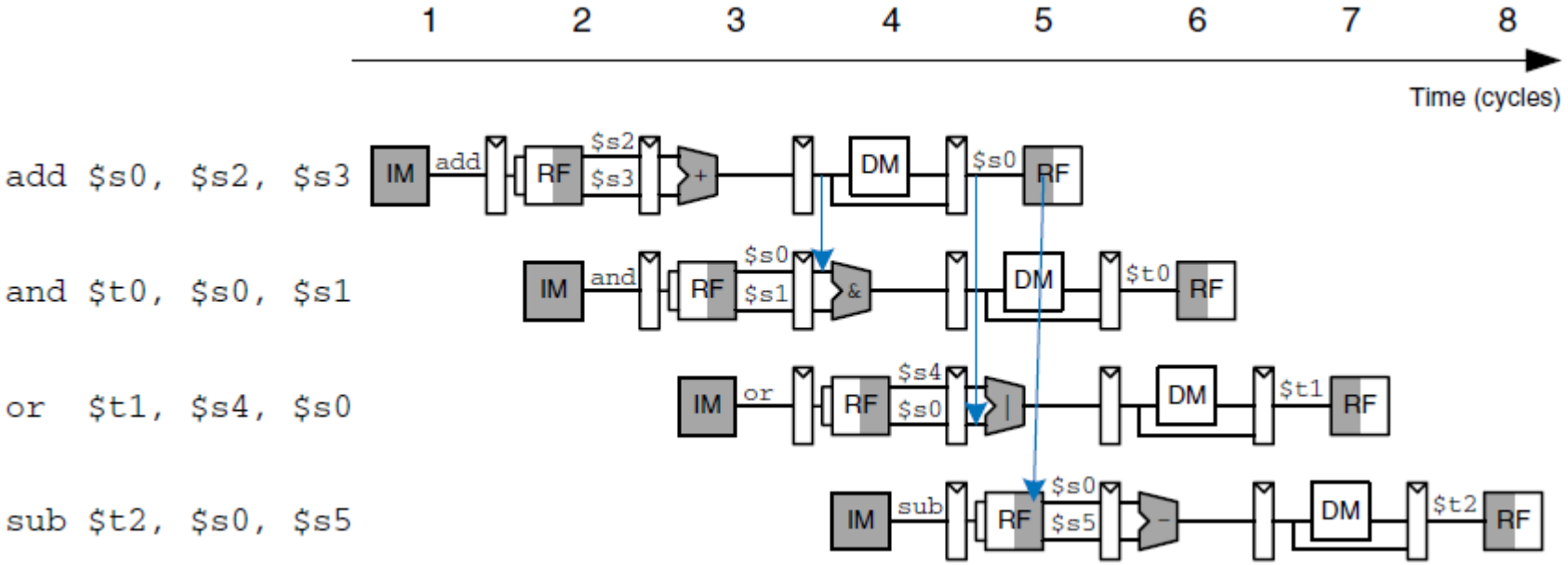


Vznik datových hazardů



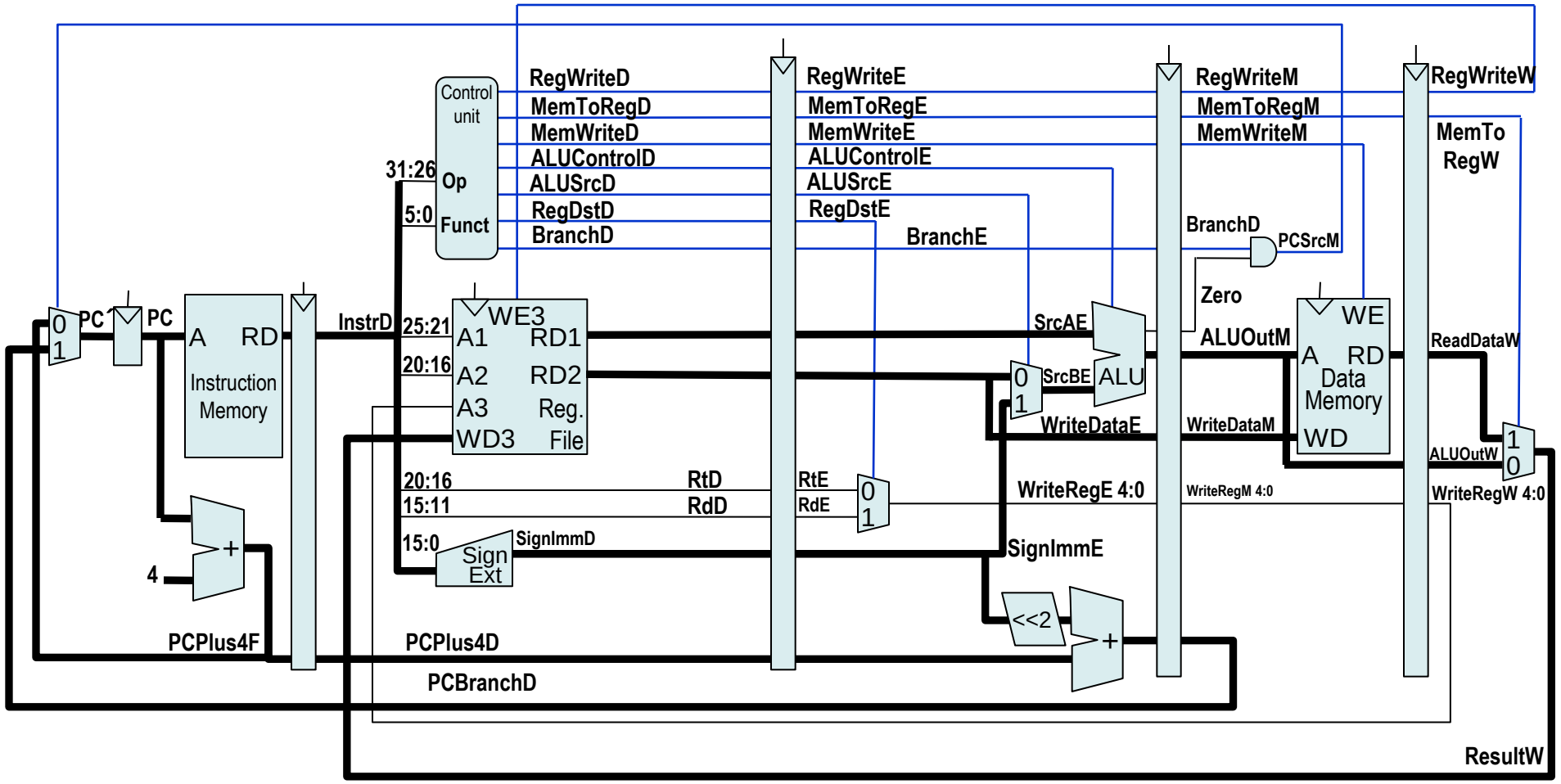
- Pracovní registry (Register File) – přístup v dvou fázích (Decode, WriteBack) – zápis v první polovině cyklu, čtení ve druhé..
- RAW hazard…
- Jak je možné řešit tento hazard a nedegradovat výkon pipeline?

Řešení datových hazardů přeposíláním (forwarding)

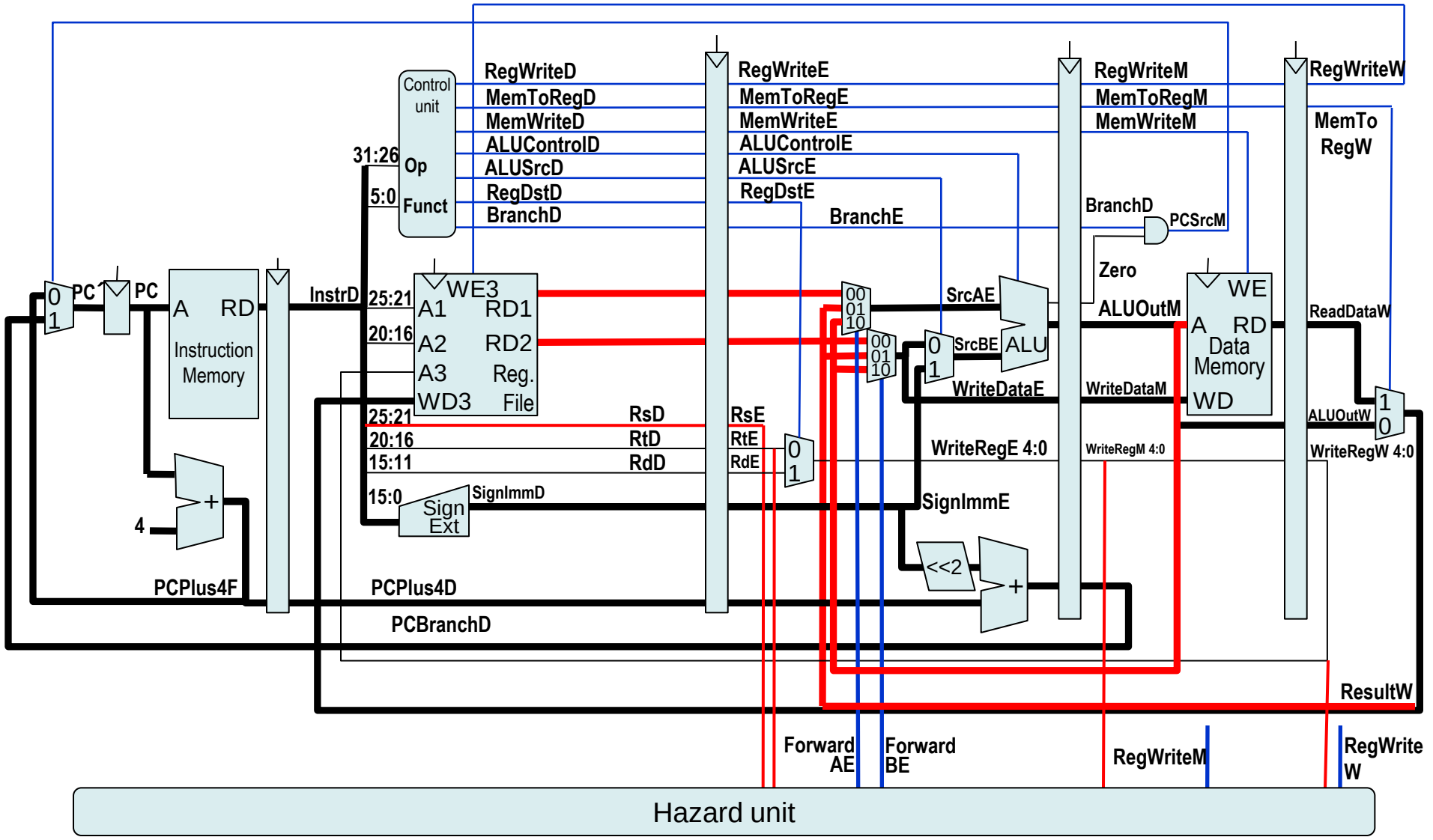


- Pokud výsledek vzniká dřív, než jej následující instrukce skutečně potřebují, pak lze tento hazard vyřešit přeposíláním (forwarding)
- Dochází k němu, když se použité zdrojové registry instrukce ve stupni E shodují s cílovým registrem ve stupni M nebo WB,
- přičemž se vyhodnotí, zda se skutečně jedná o cílový registr – viz instr. lw vs. sw
- Obsah registrů se z uvedených stupňů posílá do Hazard Unit,

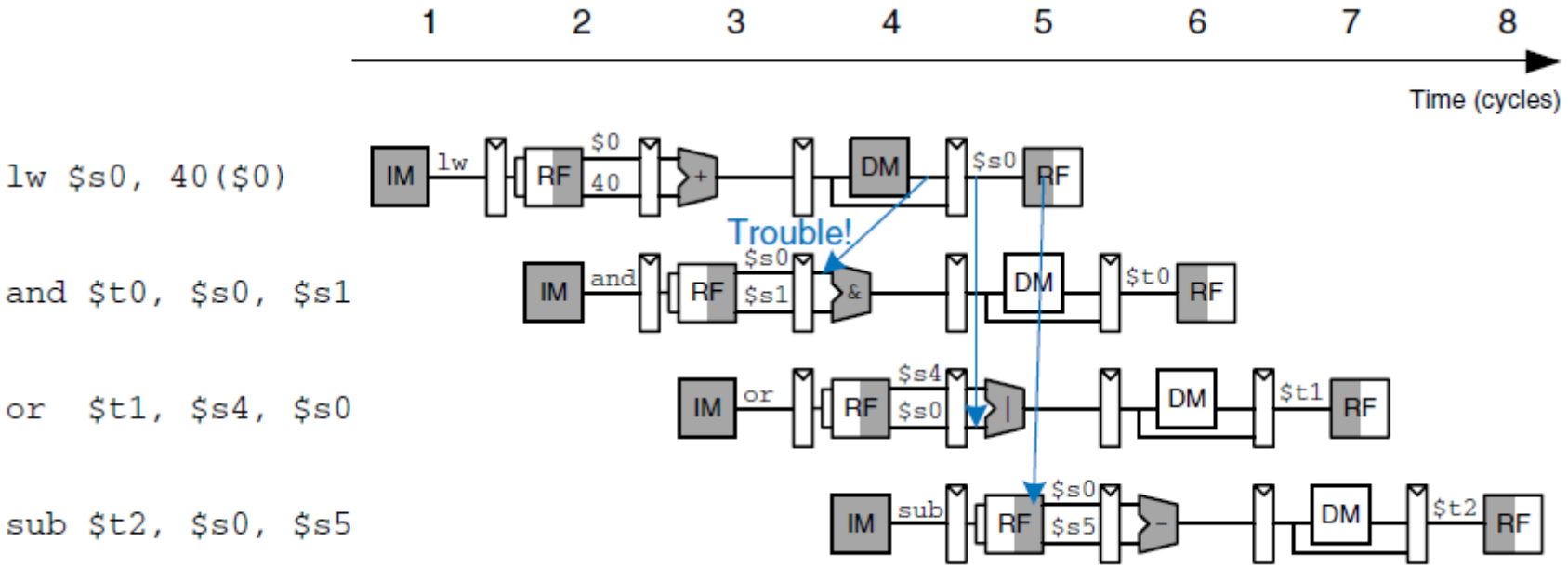
Stávající procesor



Řešení datových hazardů přeposíláním (forwarding)

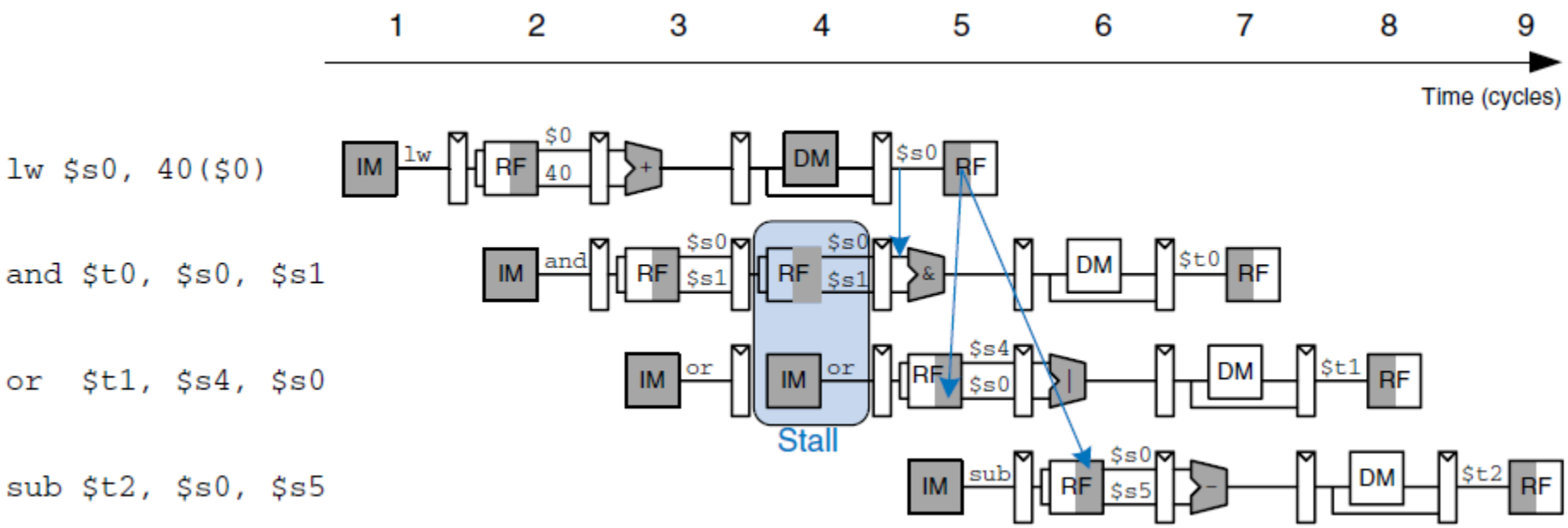


Řešení datových hazardů pozastavením (stall)



- Pokud následující instrukce potřebují výsledek dřív, než skutečně vzniká, lze hazard řešit také jinou metodou, a to pozastavením (stall)
- Pozastavení (stall) 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)

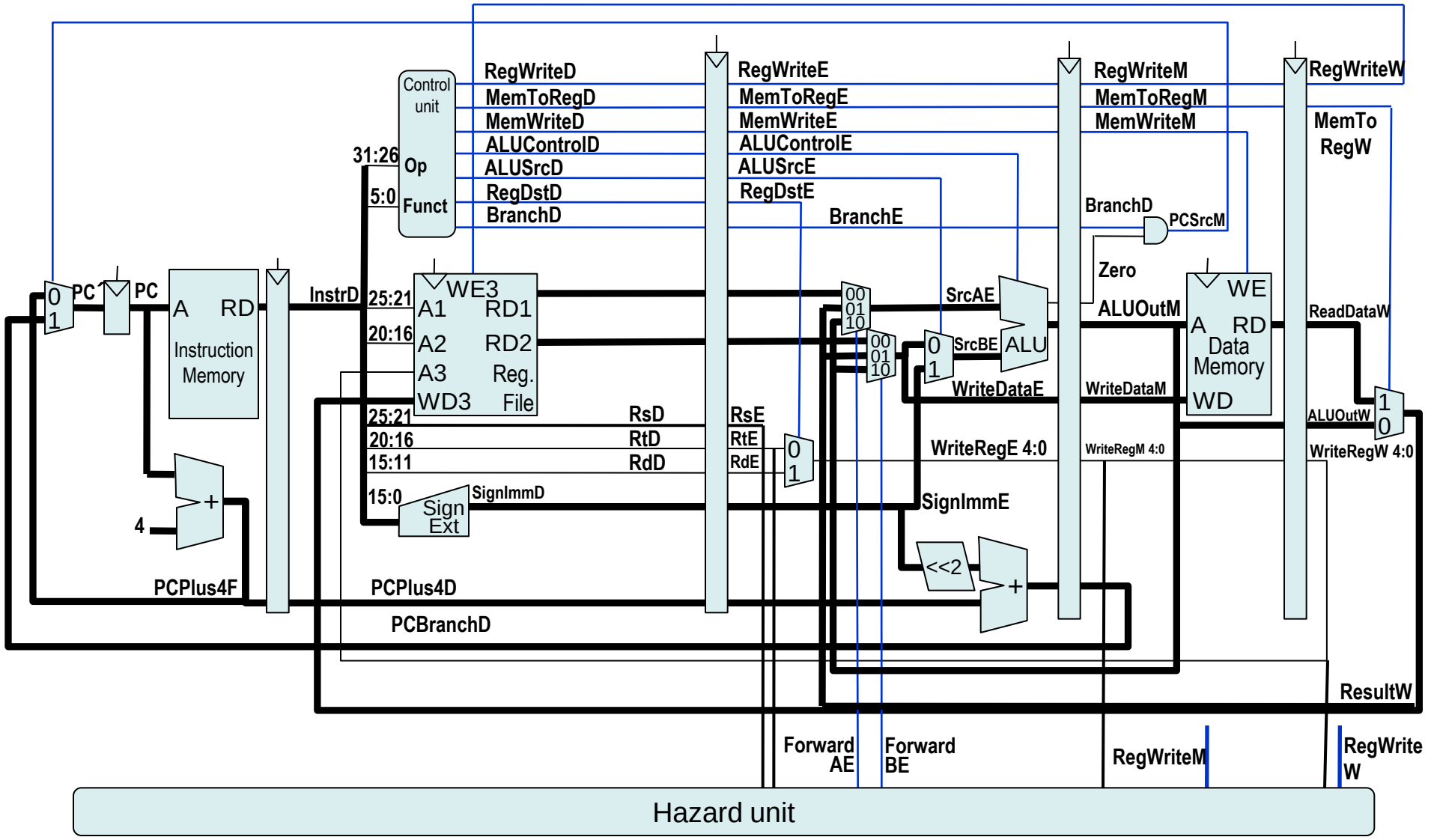
Řešení datových hazardů pozastavením (stall)



- 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)