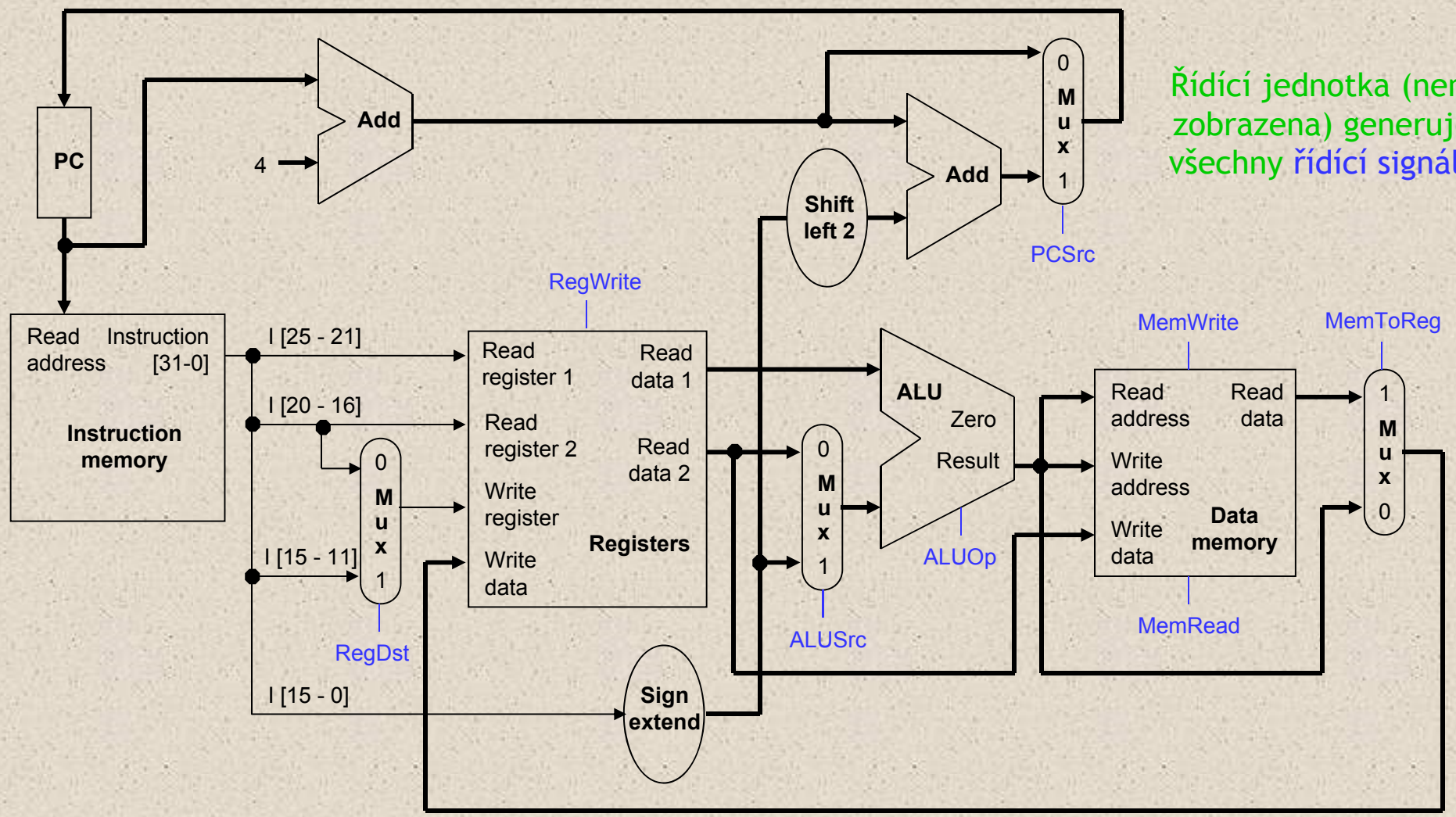
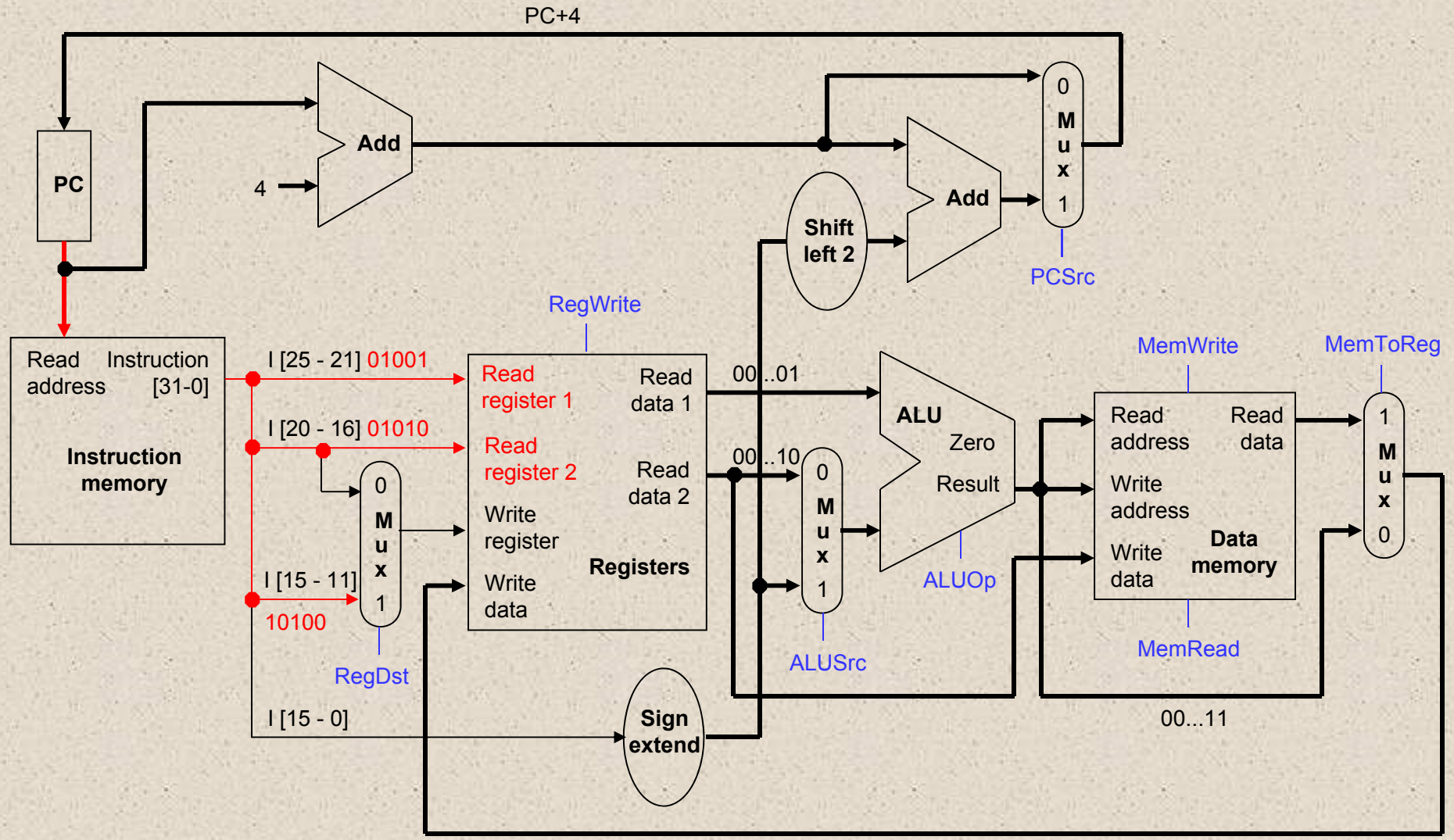
Úvod do organizace počítače

Od jednocyklové implementace k pipelinningu

Jednocyklová jednotka

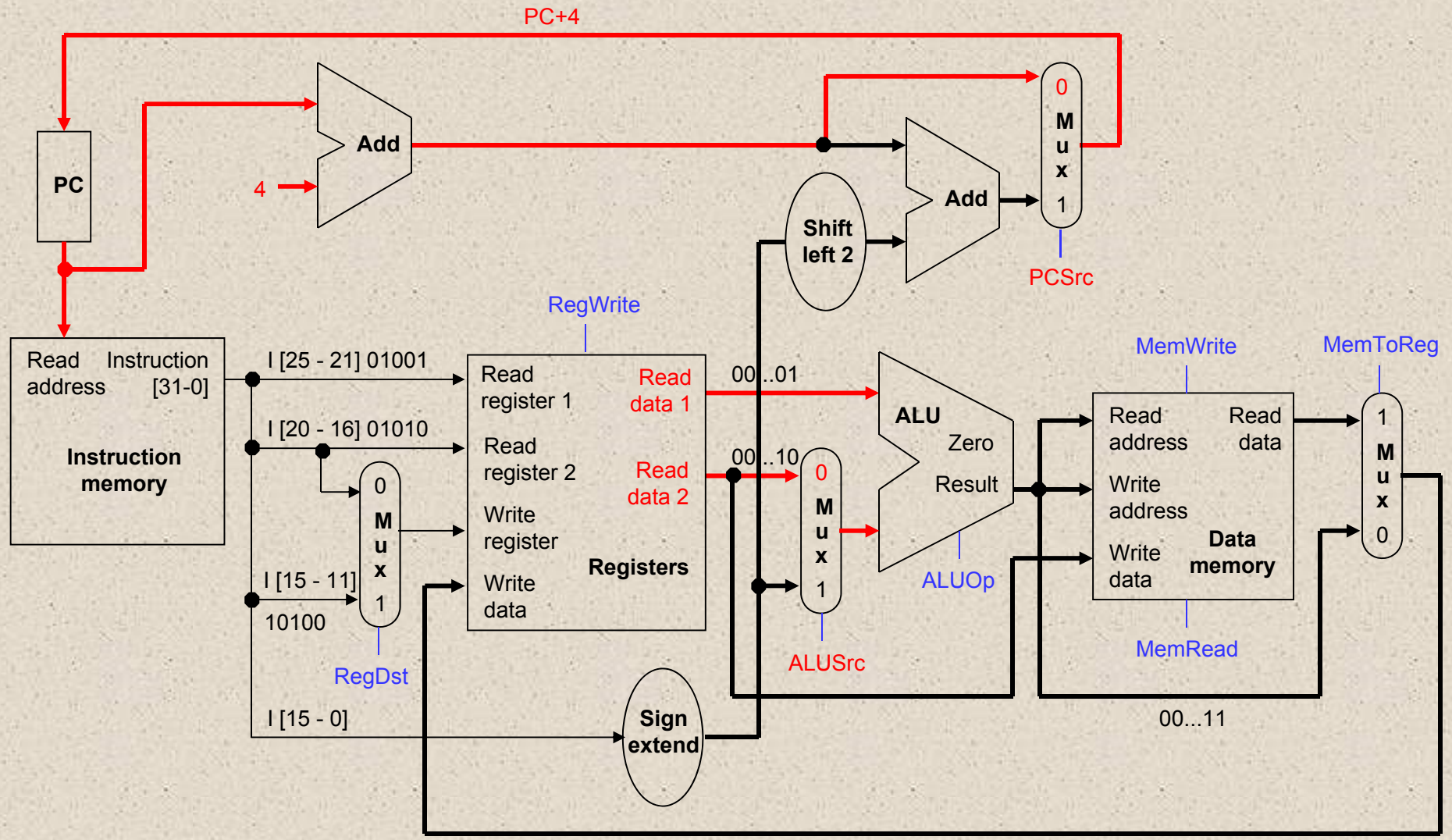


Provádění instrukce add $s4, $t1, $t2



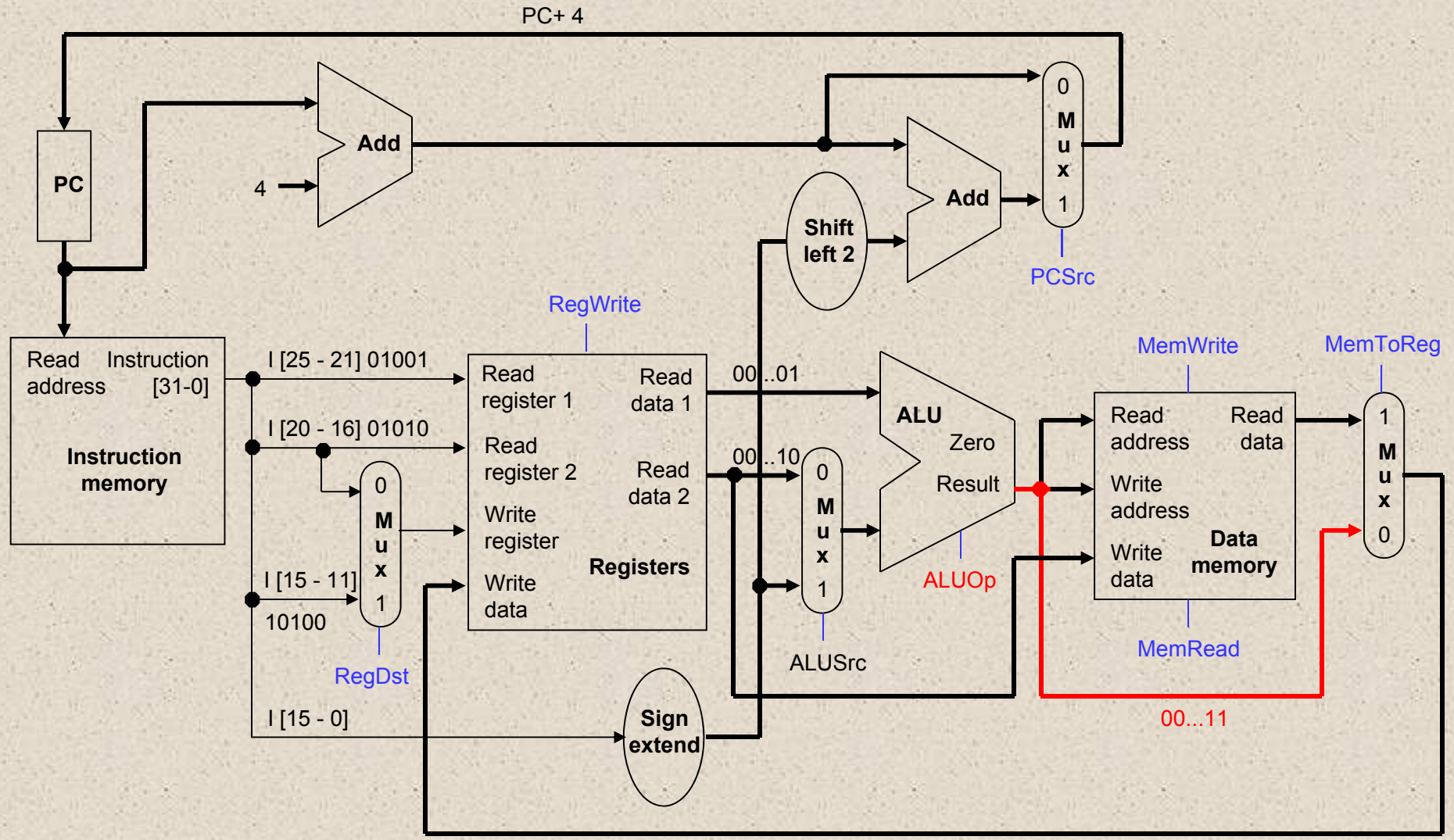
Čtení instrukce z paměti

Provádění instrukce add $s4, $t1, $t2



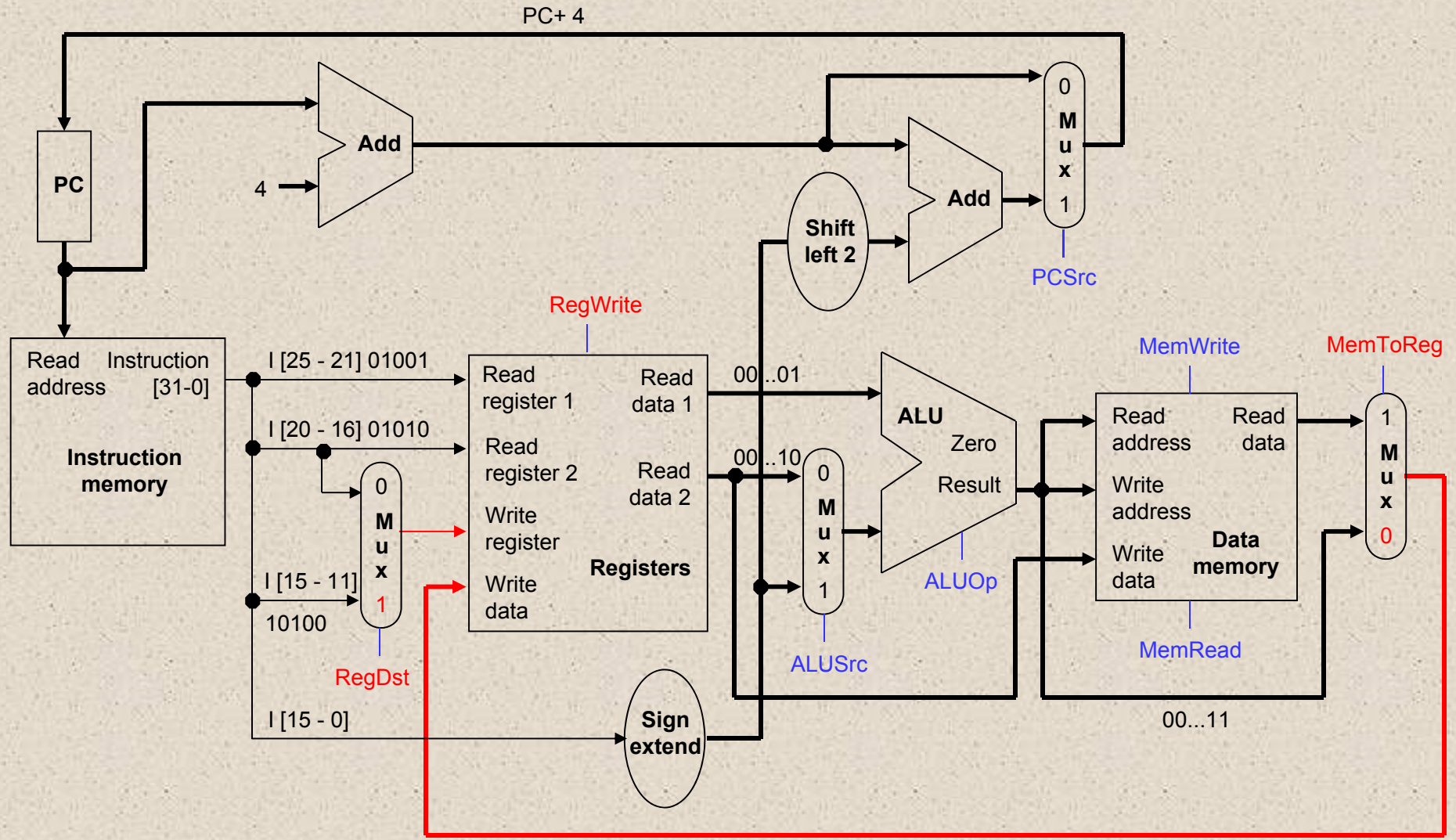
Čtení registrů, inkrement PC + 4, zápis nové hodnoty PC

Provádění instrukce add $s4, $t1, $t2



Provedení součtu v ALU

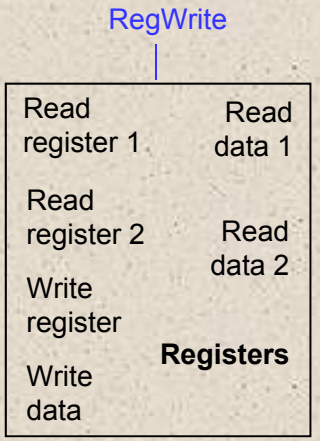
Provádění instrukce add $s4, $t1, $t2

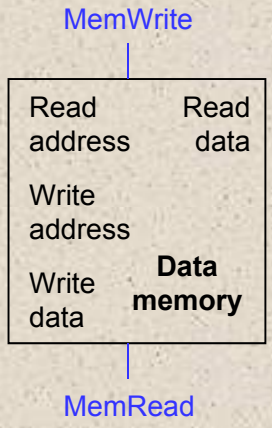


Zápis výsledku do registru

Časování: hrany hodinového signálu

- Jak zajistíme v instrukci jako add $t1, $t1, $t2, že do $t1 nebude zapsáno, ***dokud*** nebyla přečtena jeho originální hodnota?
- Budeme předpokládat, že paměťové prvky jsou taktovány pomocí náběžných hran hodinového signálu.
  - Registrový soubor a datová paměť mají explicitní řídící signály pro zápis, RegWrite a MemWrite. Do těchto jednotek se zapisuje v okamžiku, kdy je řídící signál aktivní a ***současně*** nastala náběžná hrana hodinového signálu.
  - V jednocyklovém stroji je do PC zapisováno v každém hodinovém cyklu, takže nepotřebujeme explicitní řídící signál.





Výkonnost jednocyklového procesoru

$$\text{CPU time}_{X,P} = \text{Instructions executed}_P * \text{CPI}_{X,P} * \text{Clock cycle time}_X$$

CPI = 1 pro jednocyklový návrh

S náběžnou hranou hodin je do PC zapsána nová adresa.

1. Je přečtena nová instrukce z paměti. Řídící jednotka aktivuje řídící signály tak, aby se provedlo:
   - čtení registrů,
   - generování výstupu ALU,
   - čtení datové paměti pro instrukci lw
   - výpočet cílové adresy skoku.
2. S ***další*** náběžnou hranou hodin se provede:
   - Zápis do registrového souboru pro aritmetické instrukce nebo pro instrukci lw.
   - Zápis do datové paměti pro instrukci sw.
   - Stanoven nový obsah PC – ukazuje na další instrukci.

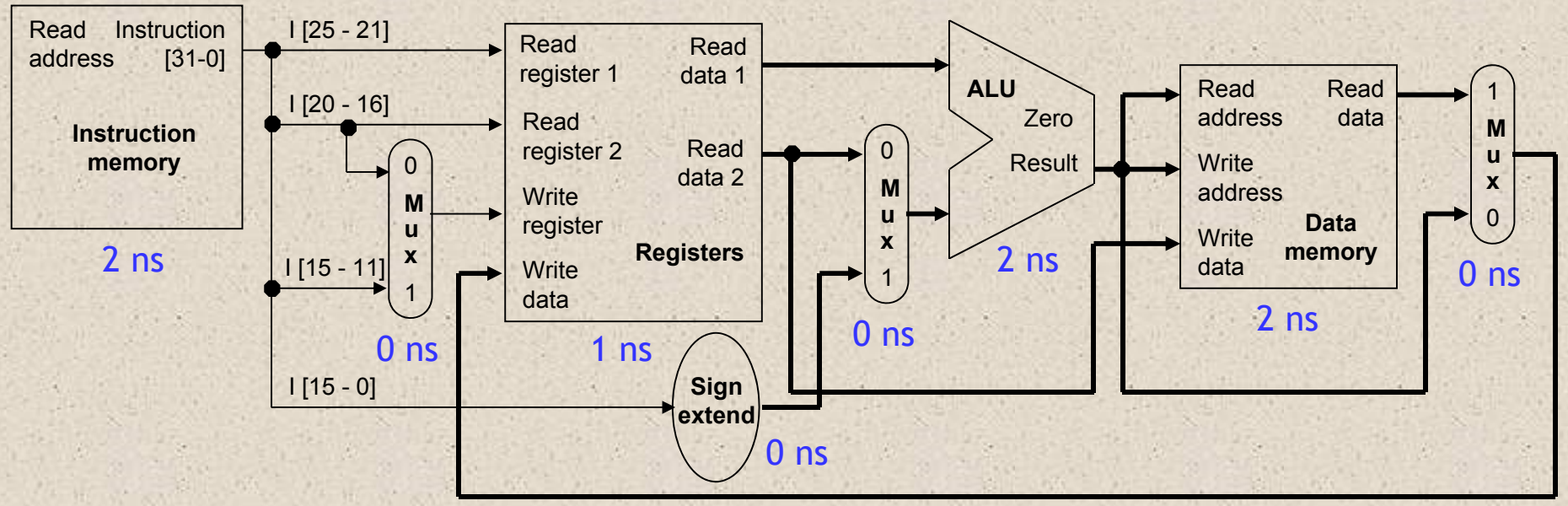
- U jednocyklové verze všechno v bodě 2 se musí odehrát během jednoho hodinového cyklu, před příchodem další náběžné hrany hodin.

Jak dlouho má trvat perioda hodin?

Prvky procesoru

- Jestliže se všechny instrukce musejí dokončit během jednoho cyklu, musí jeho délka vyhovět ***nejpomalejší*** instrukci.
- Každý prvek procesoru vykazuje určité zpoždění (latence)

čtení instrukční paměti	2ns	8ns
čtení registrového souboru	1ns	
výpočet v ALU	2ns	
přístup do datové paměti	2ns	
zápis do registrového souboru	1ns	

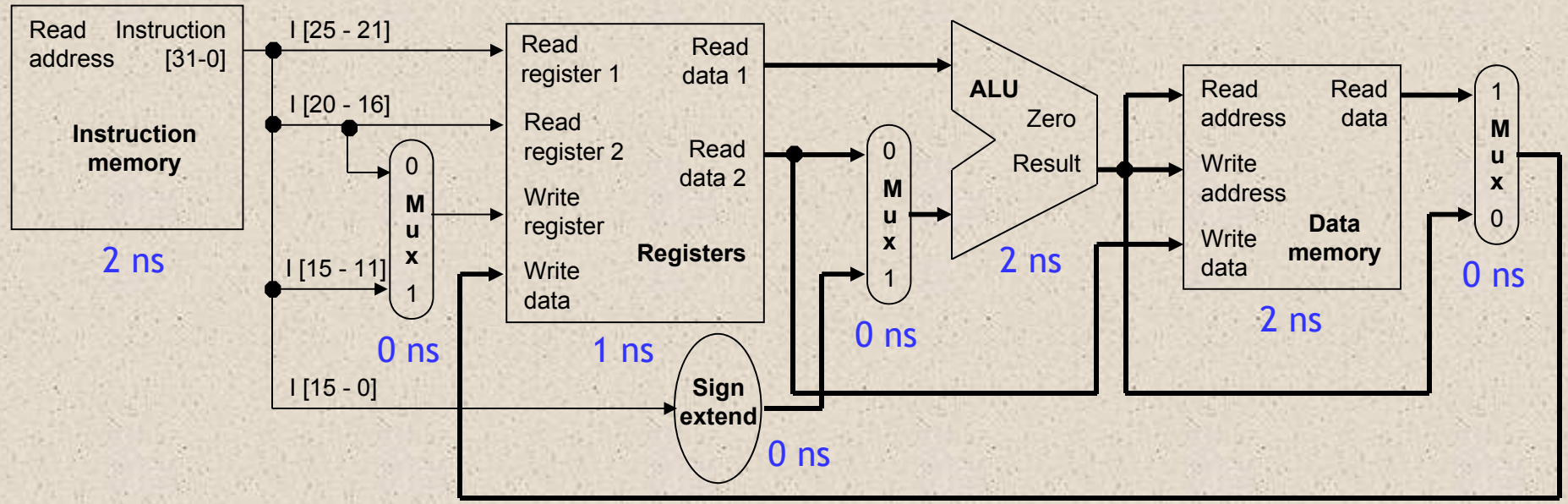


Nejpomalejší instrukce...

Instrukce lw používá ***všechny*** prvky procesoru

- Například, lw $t0, –4($sp) potřebuje 8 ns, použijeme-li následující odhad.

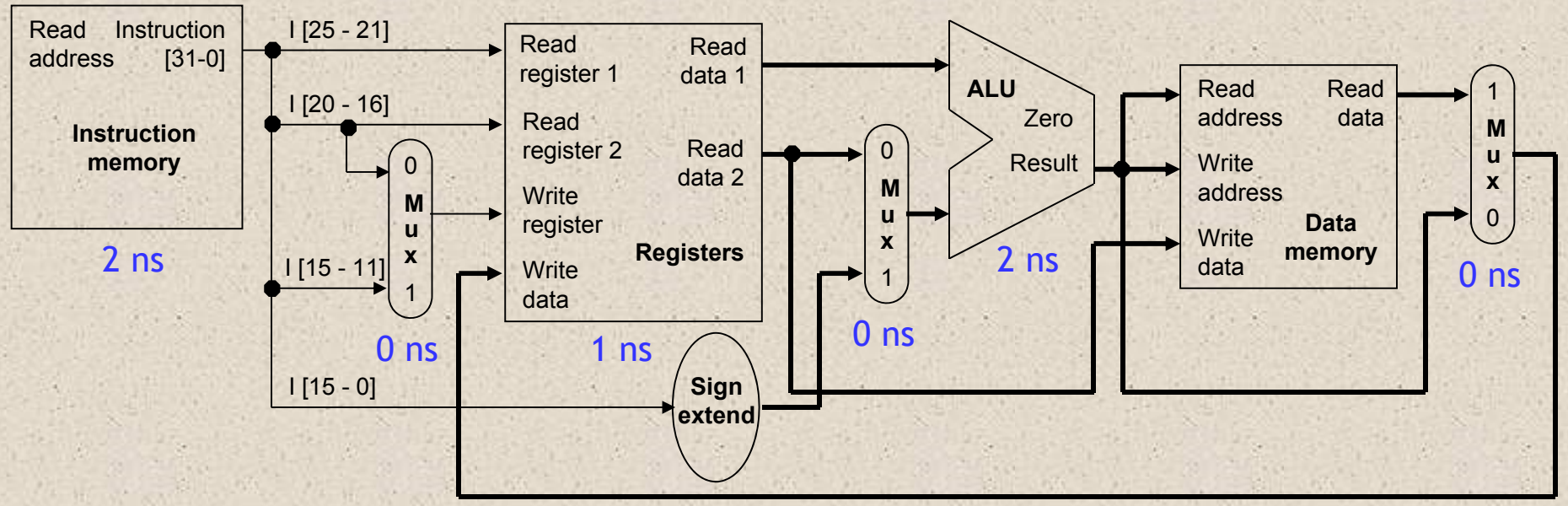
čtení instrukční paměti	2ns	
čtení bázového registru $sp	1ns	
výpočet adresy paměti $sp-4	2ns	8ns
čtení datové paměti	2ns	
uložení dat zpět do $t0	1ns	



...určuje dobu periody hodin

- Zvolíme-li dobu cyklu 8 ns, potom *každá* instrukce bude trvat 8 ns, i když by tolik času nepotřebovala.
- Například instrukci add $s4, $t1, $t2 ve skutečnosti postačí jen 6 ns.

čtení instrukční paměti	2ns	6ns
čtení registrů $t1 a $t2	1ns	
výpočet $t1 + $t2	2ns	
uložení výsledku do $s0	1ns	



Lze to akceptovat?

- Při stejných zpožděních by instrukce sw potřebovala 7 ns a beq jen 5 ns.
- Předpokládejme instrukční mix **gcc** z učebnice.

Instrukce	Četnost
Aritmetické	48%
Čtení (lw)	22%
Uložení (sw)	11%
Větvení	19%

- V případě jednocyklového návrhu by každá instrukce potřebovala 8 ns.
- Jestliže budeme provádět výpočet jak nejrychleji je možné, bude střední doba výpočtu instrukce pro gcc:

(48% x 6ns) + (22% x 8ns) + (11% x 7ns) + (19% x 5ns) = 6.36 ns

- Jednocyklový návrh je přibližně 1.26 krát pomalejší!

Může být hůř...

- Použili jsme příliš optimistický předpoklad o době cyklu paměti:
  - Přístup do hlavní paměti moderních strojů je >50 ns.
  - Pro srovnání, operace ALU v Pentiu4 trvá ~0.3 ns.

- Náš nejhorší případ cyklu (load/store) zahrnuje 2 přístupy do paměti
  - U moderních jednocyklových implementací bychom skončili pod <10Mhz.
  - Cache paměti zlepšují střední přístupovou dobu, nikoliv nejhorší případ.

Zlepšení výkonu

- Dvě možnosti jak zlepšit výkon:

  1. Rozdělit každou instrukci do více kroků, každý bude trvat 1 cykl
     - kroky: IF (instruction fetch), ID (instruction decode), EX (execute ALU operation), MEM (memory access), WB (register write-back)
     - pomalé instrukce zaberou více cyklů, než ty rychlejší
     - = *vícecyklová* implementace

  2. Důležité „zjištění“: každá instrukce využívá pouze část hardware v každém kroku
     - instrukce lze ve zpracování *překrývat*; každá využívá jen část hw
     - = implementace s režimem *pipeline*

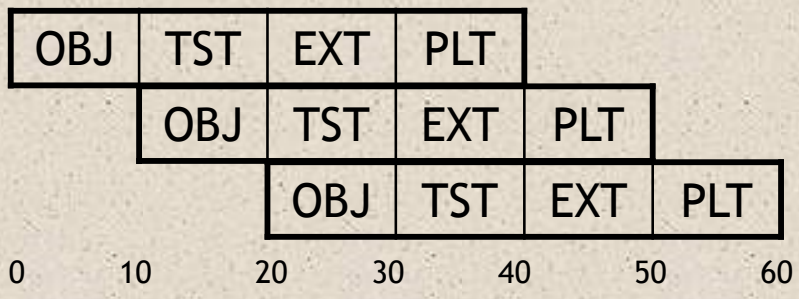
- Příklady pipeliningu: jakýkoliv proces montáže (auta, bagety), prádelna (pračka + sušička – lze provádět v režimu pipeline), atd.

Příklad

- Sestavení bagety:
  - OBJ (8 sekund) Objednávka
  - TST (0 nebo 10 sekund) Nahřátí
  - EXT (0 nebo 10 sekund) Případná výplň
  - PLT (5 sekund) Placení

Jednoduchá bageta trvá 13 až 33 sekund

- Použijeme-li pipelining můžeme sestavit bagetu každých 10 sekund:



Pipelining

- **Pipelining může *zvýšit propustnost* (#baget za hodinu), ale …**

1. Každá bageta musí využívat *všechny* stupně
   — brání konfliktům v pipeline
2. každý stupeň musí zabírat stejné množství času
   — limitováno *nejpomalejším* stupněm (v tomto případě 10 sekund)

- Tyto dva faktory *snižují latenci* (doba/1bagetu)!

- V optimální k-stupňové pipeline struktuře:
  1. každý stupeň koná smysluplnou činnost
  2. délka stupňů je vyvážená

- Za těchto podmínek lze dosáhnout *téměř* optimální zrychlení: *k*
  — “téměř” protože stále tu existuje doba *plnění* a *vyprázdnění*

Pipelining není „multiprocessing“

- Pipelining je vlastně určitý druh paralelního zpracování.

- Jak multiprocessing, tak pipelining znamenají aktivaci více procesů ve více „funkčních jednotkách“
  - **Multiprocessing – každý proces běží celý v samostatné funkční jednotce**
    - např. pokladny v supermarketu
  - **Pipelining** – každý proces je rozčleněn na drobné části a každá je prováděna ve specializované funkční jednotce.
- Pipelining a multiprocessing se navzájem nevylučují
  - Moderní procesory uplatňují oba principy, násobné pipeline struktury (superskalární procesory)
- Pipelining je obecný princip zvyšování efektivity, používaný v počítačových systémech:
  - Sítě, I/O zařízení, softwarová architektura serverů

Pipelining u MIPSu

- Provedení instrukce MIPS může zabrat 5 kroků

Krok	Jméno	Popis
Instruction Fetch	IF	Čtení instrukce z paměti
Instruction Decode	ID	Čtení zdrojových registrů a generace řídících signálů
Execute	EX	Výpočet výsledku R-typu popř. větvení
Memory	MEM	Čtení nebo zápis do datové paměti
Writeback	WB	Uložení výsledku do cílového registru

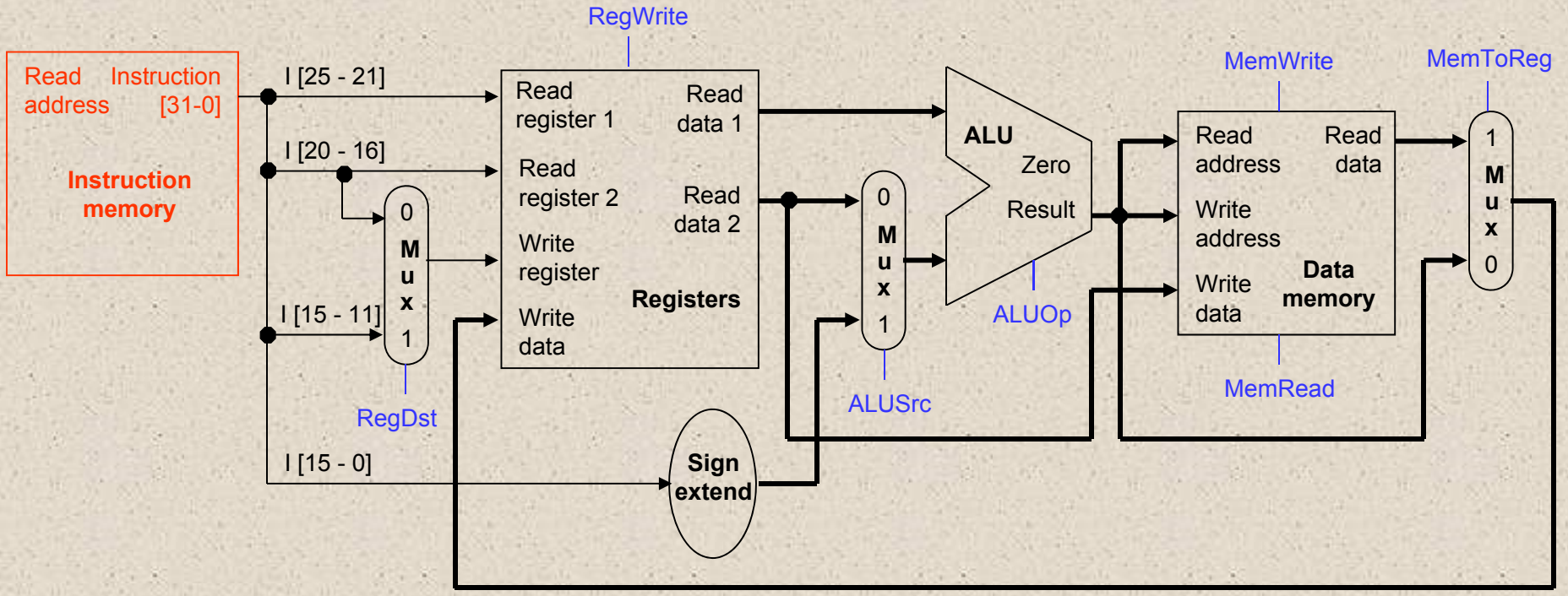
- Všechny instrukce nepotřebují všech 5 stupnů…

Instrukce	Vyžadované kroky				
beq	IF	ID	EX		
R-type	IF	ID	EX		WB
sw	IF	ID	EX	MEM	
lw	IF	ID	EX	MEM	WB

- …jednocyklová verze ale musí zvládnout všech 5 stupňů během jednoho taktu

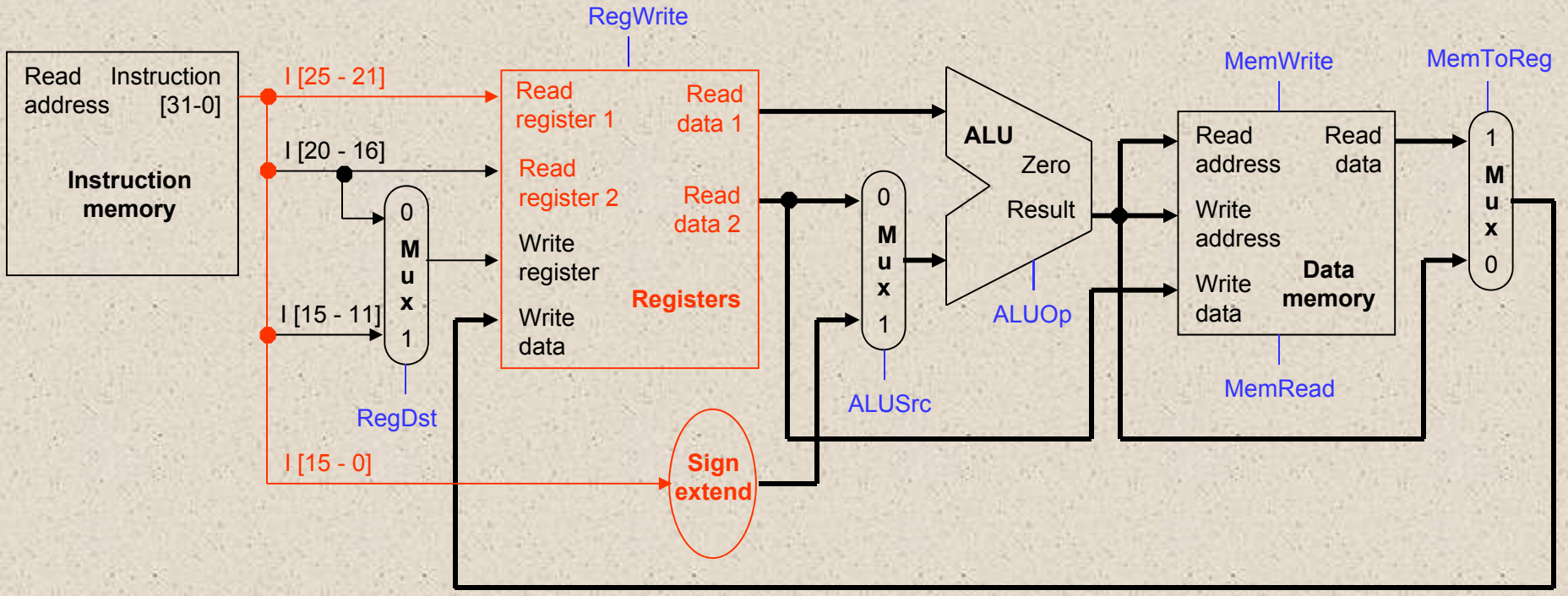
Čtení instrukce (IF)

- Zatímco se provádí IF, zbytek jednotky je nečinný …



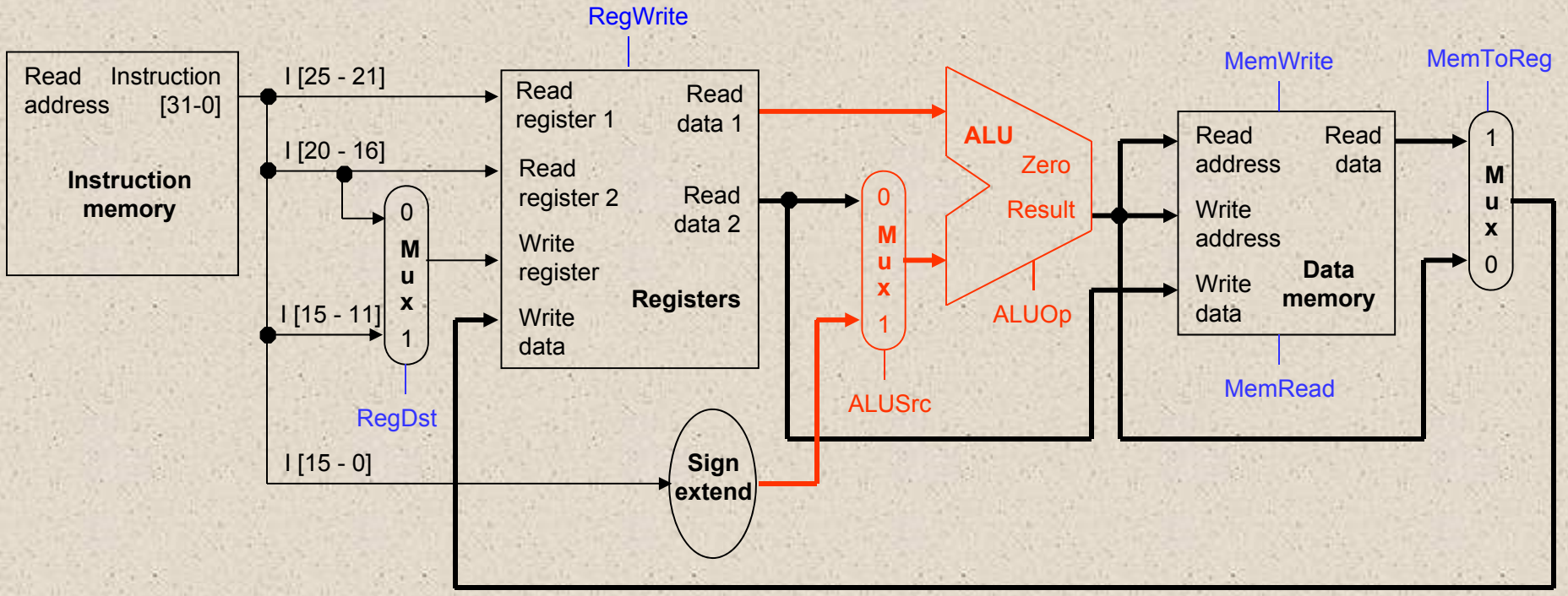
Dekódování instrukce (ID)

- Při provádění ID je část, odpovídající IF nečinná …



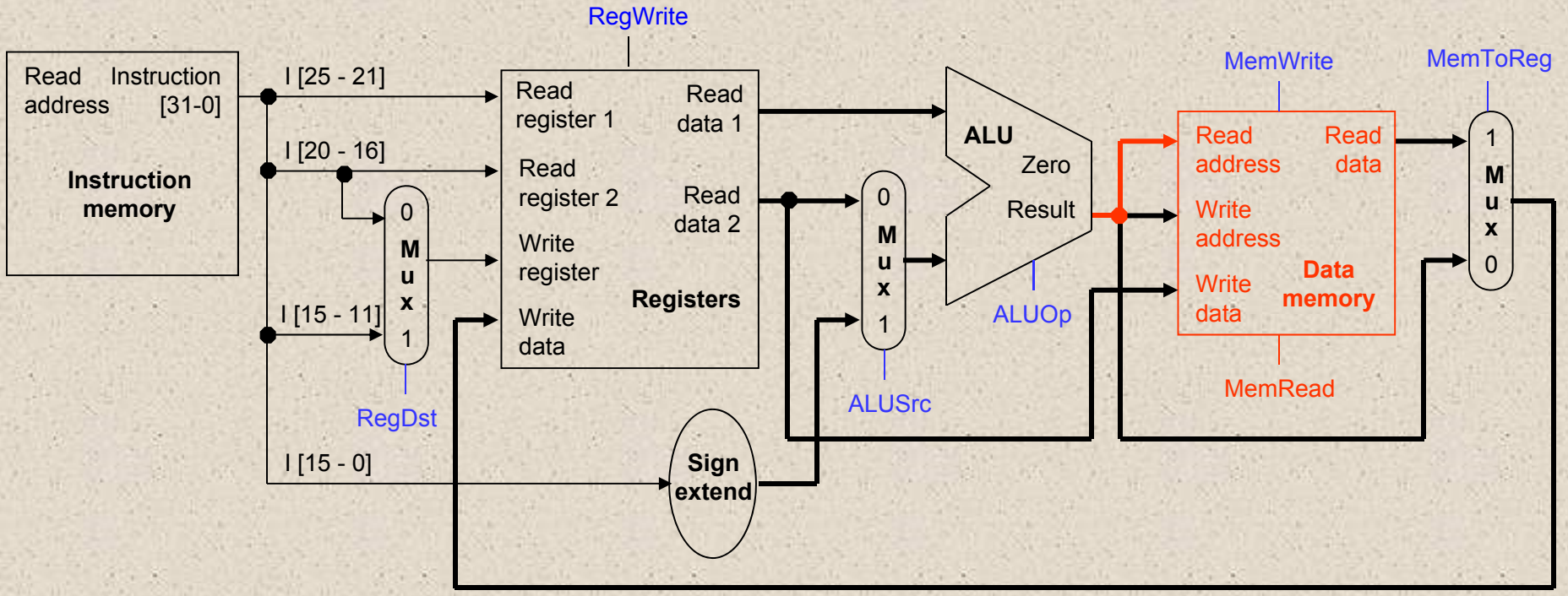
Prováděcí fáze (EX)

- ..totéž platí pro EX …



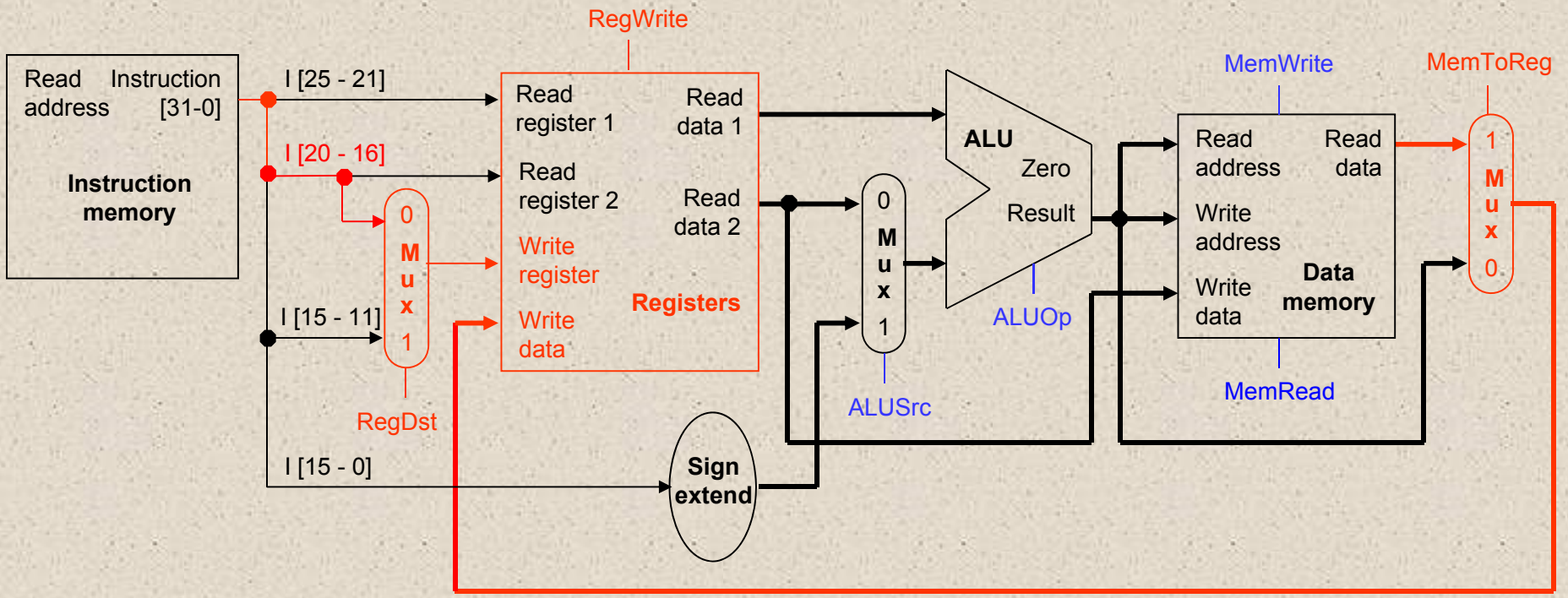
Paměť (MEM)

- … část MEM …



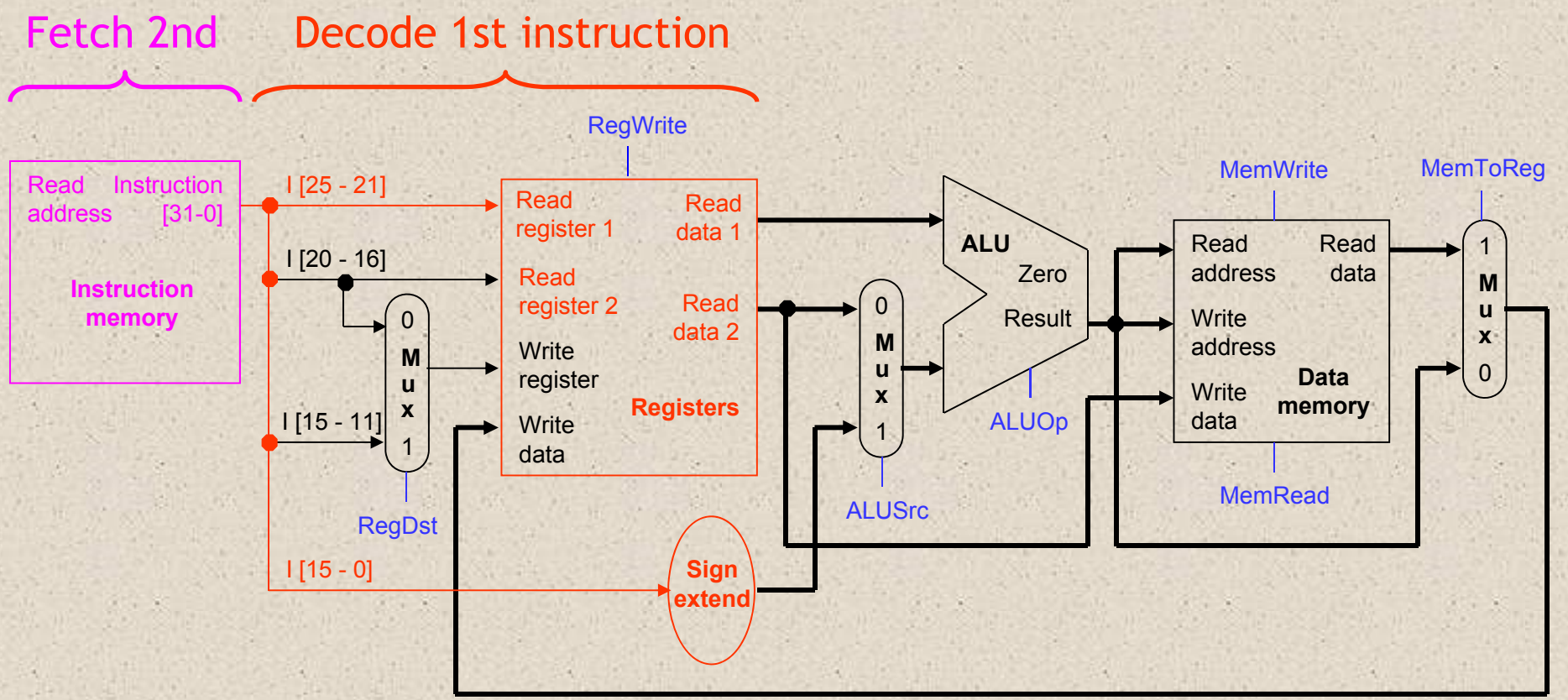
Zápis výsledku (WB)

- … totéž pro část, odpovídající WB.
- „konflikt“ ve stupni IF v registrovém souboru?
- Odpověď: Do registrového souboru se zapisuje s náběžnou hranou, čtení probíhá v další části hodinového cyklu. Konflikt tedy nenastává.



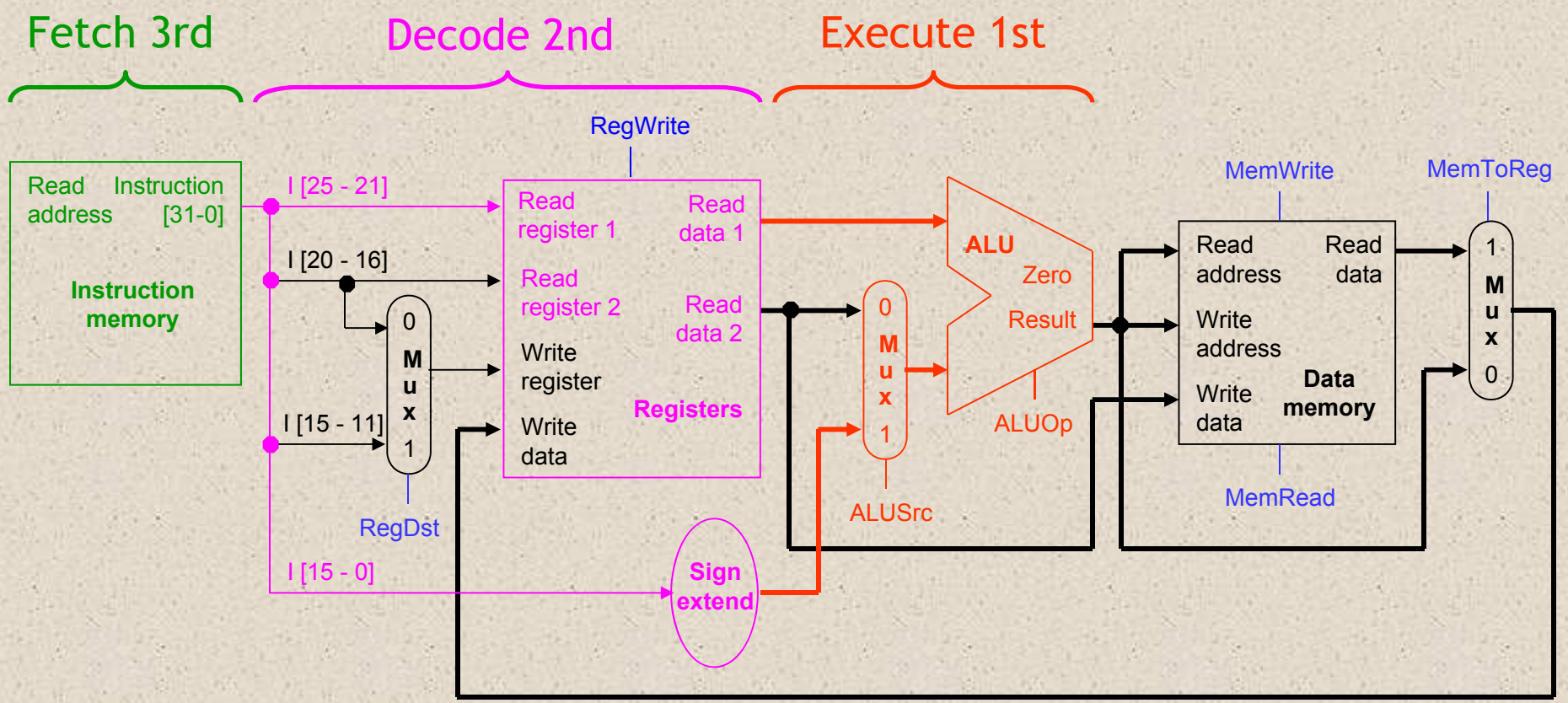
Dekódování a čtení instrukce současně

- Můžeme jít dále a číst další instrukci, zatímco se přečtená dekóduje?



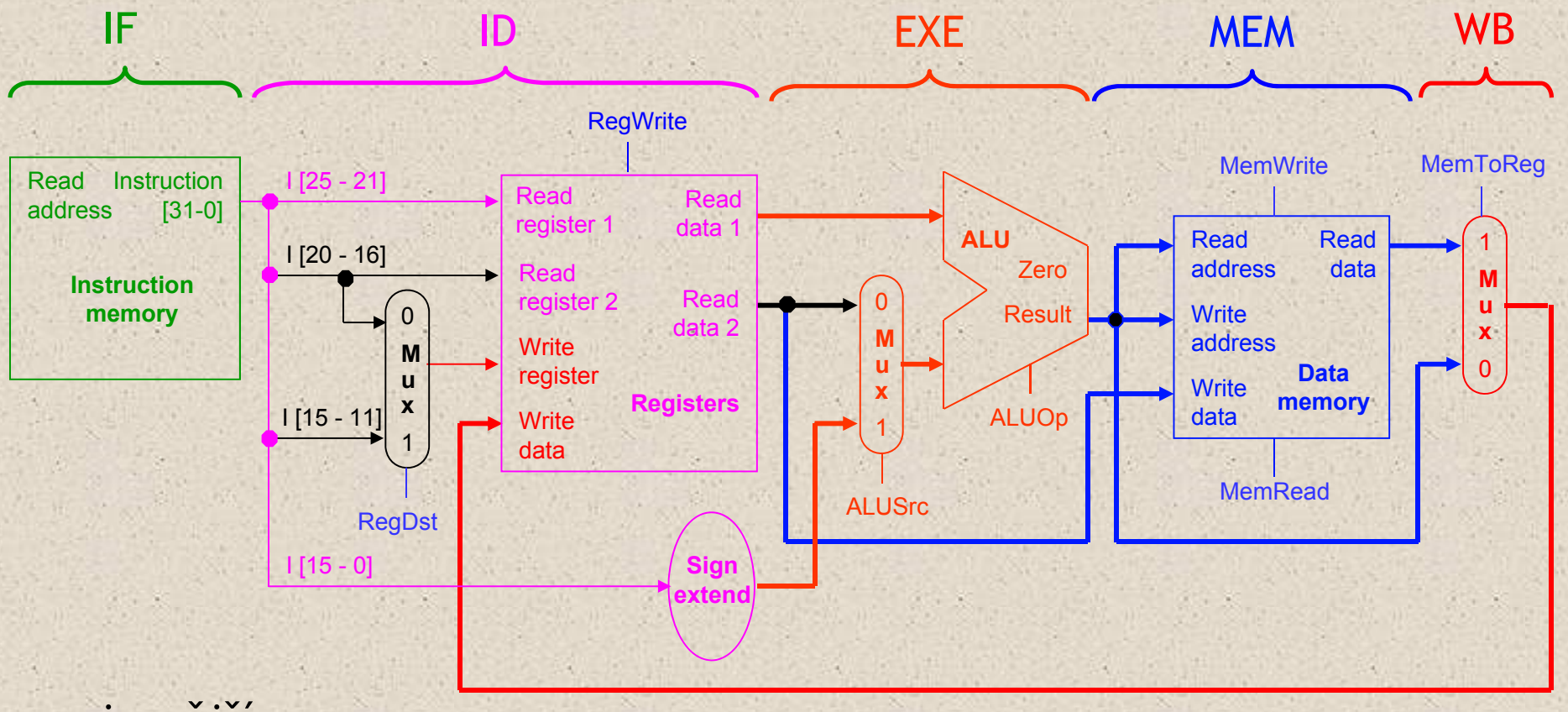
Provádění, dekódování a čtení instrukce

- Podobně, jakmile vstoupí instrukce do prováděcího stupně, lze zároveň dekódovat druhou instrukci.
- Instrukční paměť je ale teď opět volná a proto můžeme načíst třetí instrukci!



Rozdělení jednotky do 5 stupňů

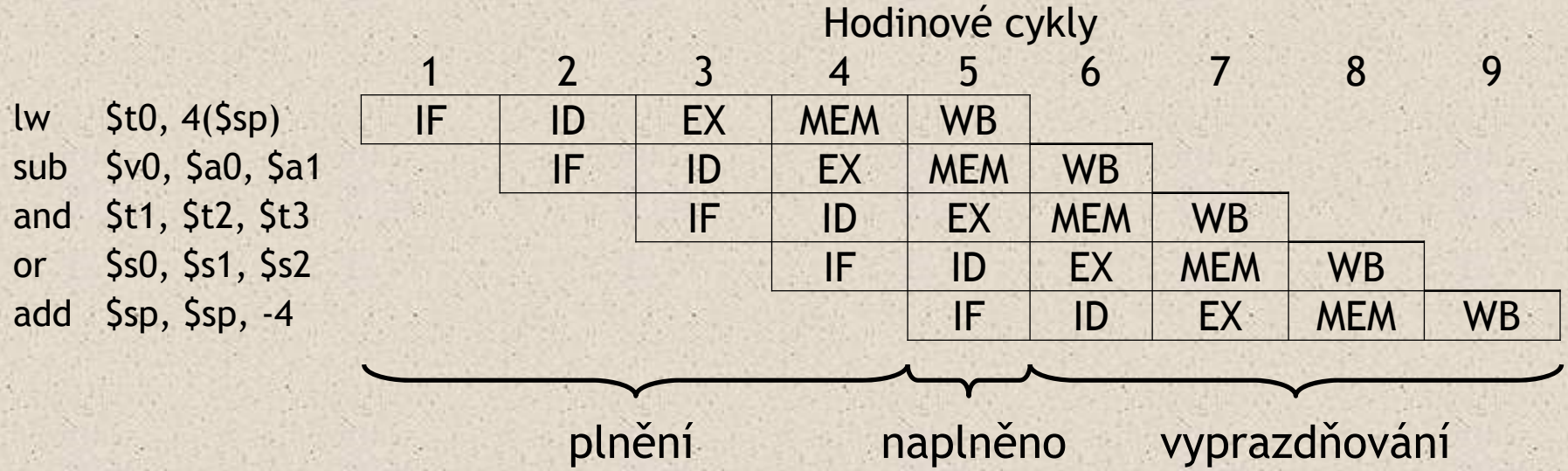
- Každý stupeň má svoji vlastní funkční jednotku
- Plný režim pipeline ⇒ procesor současně zpracovává 5 instrukcí!



Pipeline diagram

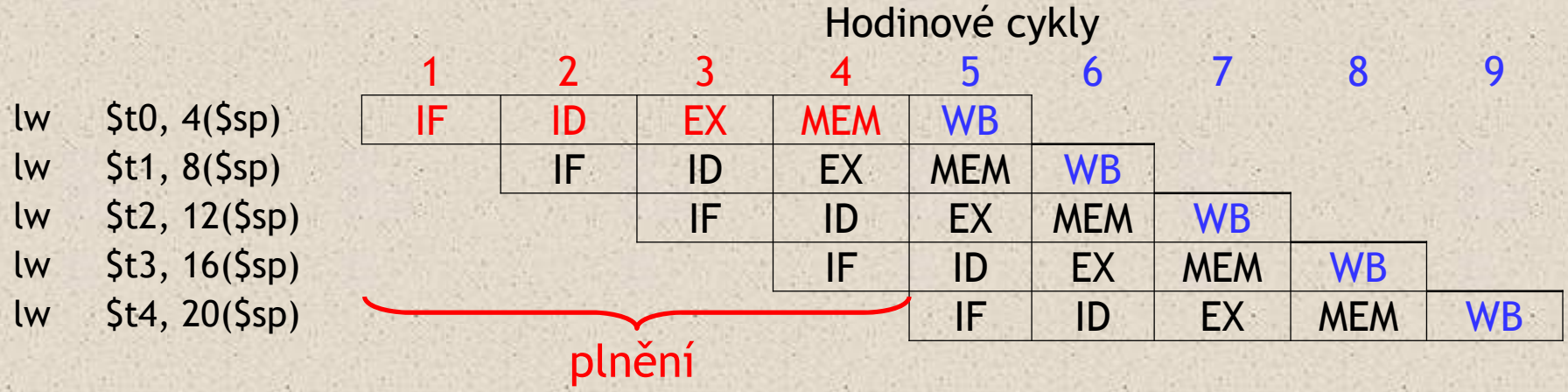
- Pipeline diagram ukazuje provádění série instrukcí
- Posloupnost instrukcí se zobrazuje vertikálně shora dolů
- Hodinové cykly ze zobrazují horizontálně zleva doprava
- Každá instrukce je rozdělena do stupnů
- Názorně ukazuje překrývání instrukcí. Například ve třetím cyklu jsou aktivní tři instrukce.
- Instrukce “lw” je ve stupni Execute.
- Současně “sub” je ve stupni Decode.
- Konečně instrukce “and” se právě načítá.

Terminologie



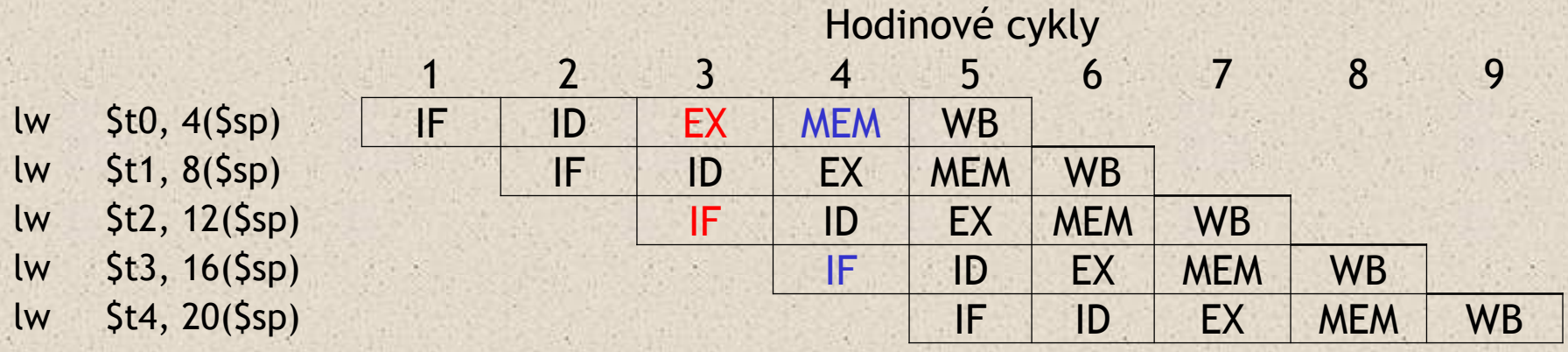
- Hloubka pipeline je počet stupňů, zde pět
- V prvních čtyřech probíhá plnění, jsou zde nevyužité funkční jednotky
- V cyklu 5 je pipeline plná. Probíhá zpracování pěti instrukcí současně, jsou využity všechny hardwarové jednotky
- V cyklech 6-9 se pipeline vyprazdňuje

Výkon pipeline struktury



- Doba výpočtu v ideální pipeline:
  - —doba pro naplnění pipeline + jeden cykl na instrukci
  - —Kolik času pro N instrukcí? $k - 1 + N$, kde k = hloubka pipeline
- Jiný způsob odvození: k cyklů pro první instrukci, plus 1 pro každou ze zbývajících $N - 1$ instrukcí.
- Porovnejte tuto pipeline implementaci (2 ns perioda hodin) s jednocyklovou implementací (8 ns perioda hodin). Kolikrát bude rychlejší pro N=1000 ?

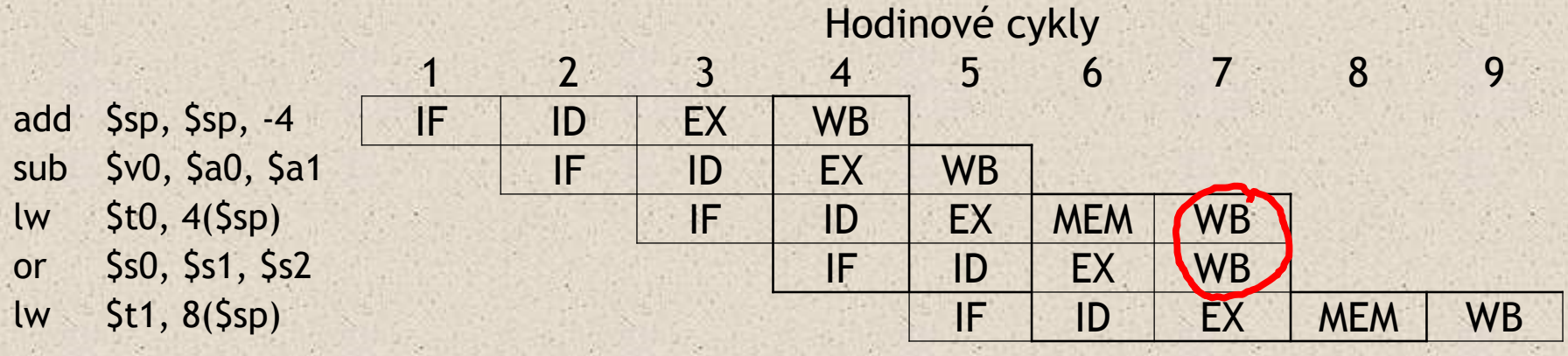
Datové cesty pro pipelinning: Požadavky



- Musíme provádět více operací v každém cyklu.
  - Inkrementace PC a sečtení registrů ve stejnou dobu.
  - Čtení instrukce v době, kdy jiná instrukce čte nebo zapisuje data.
- Jaké změny to přináší pro hardware?
  - Oddělená sčítačka (ADDER) a ALU
  - Dvě paměti (instruční paměť a datová paměť)

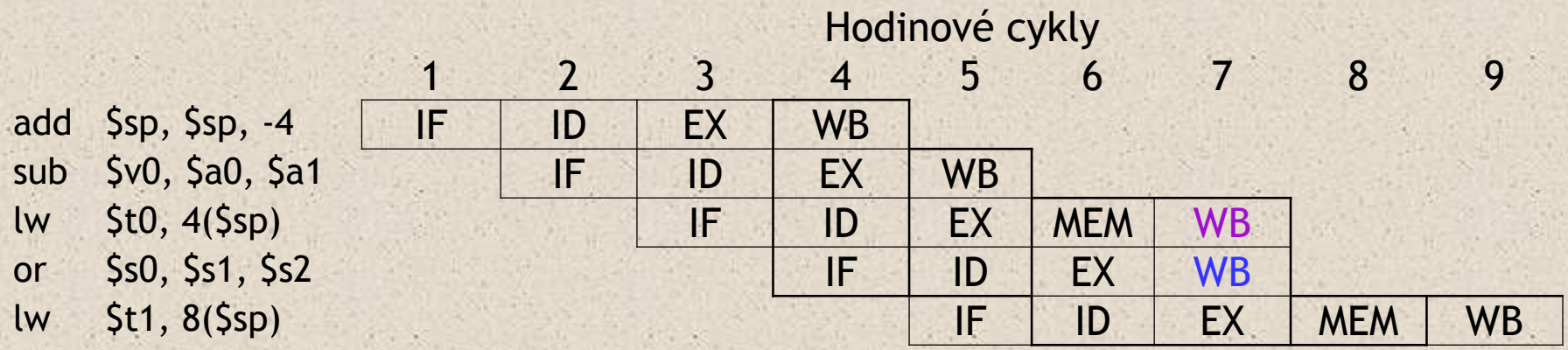
Pipelining u dalších instrukcí

- Instrukce typu-R vyžadují pouze 4 stupně: IF, ID, EX a WB
  —Nepotřebujeme stupeň MEM
- Co se stane, načteme-li do pipeline instrukce typu-R?



Konstatování

- Každá funkční jednotka smí být použita jen jednou na instrukci
- Každá funkční jednotka musí být použita ve stejném stupni pro všechny instrukce:
  - Load používá zápisový port registrového souboru v 5. stupni
  - R-typ používá zápisový port registrového souboru ve 4. stupni



Řešení: Vložení NOP

- Vynucení uniformity
  - —Navrhnout tak, aby všechny instrukce trvaly 5 cyklů.
  - —Navrhnout se stejným počtem stupňů, ve stejném pořadí
    - některé stupně budou neaktivní pro některé instrukce

R-type	IF	ID	EX	NOP	WB

Hodinové cykly

		1	2	3	4	5	6	7	8	9
add	$sp, $sp, -4	IF	ID	EX	NOP	WB				
sub	$v0, $a0, $a1		IF	ID	EX	NOP	WB			
lw	$t0, 4($sp)			IF	ID	EX	MEM	WB		
or	$s0, $s1, $s2				IF	ID	EX	NOP	WB	
lw	$t1, 8($sp)					IF	ID	EX	MEM	WB

- Zápisy a větvení mají NOPy …

store	IF	ID	EX	MEM	NOP
branch	IF	ID	EX	NOP	NOP

Závěr

- Pipelining maximalizuje propustnost překrývaným zpracováním instrukcí.
- Pipelining poskytuje značné urychlení.
  - V optimálním případě se v každém cyklu dokončuje jedna instrukce a zrychlení je rovno hloubce pipeline.
- Datové cesty se podobají cestám pro jednocyklové zpracování, navíc jsou doplněny pipeline registry
  - Každý stupeň potřebuje svoji funkční jednotku