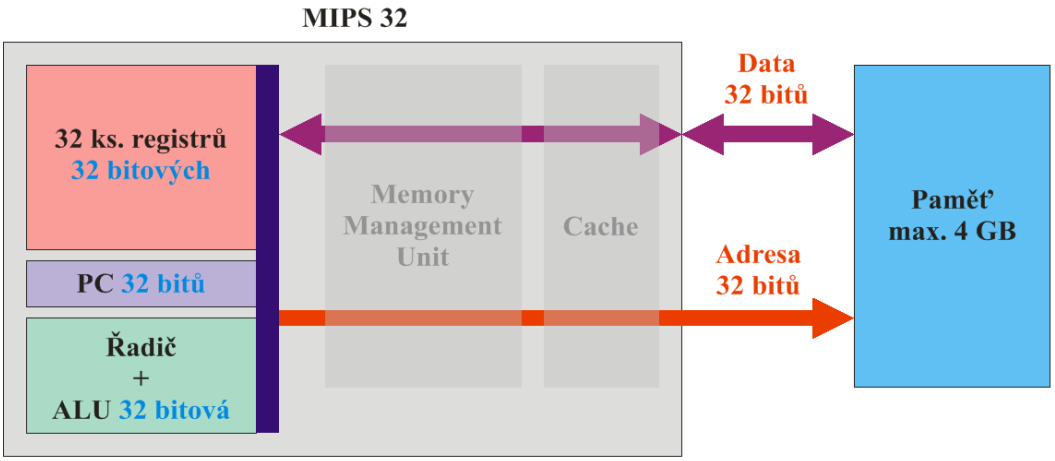
Strojový kód a asembler procesoru MIPS

Použití simulátoru SPIM

MIPS - prostředí

- 32 ks 32bitových registrů (⇒ adresa registru = 5 bitů).
  - Registr `$0` je „zero“ – čte se jako 0x0, zápis se neprovede.
- 32bitová adresa ⇒ adresní prostor 2^{32} = 4 GB.
- 32bitová datová sběrnice.

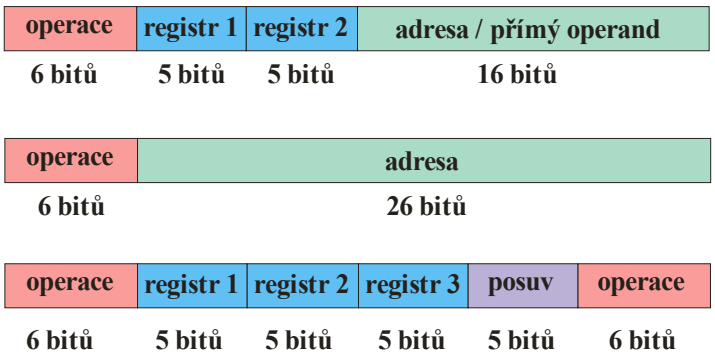


MIPS - charakteristika

- MIPS je procesor typu RISC:
  - Všechny instrukce mají délku 32 bitů.
  - Využívá se proudové zpracování (pipeline).
    - Počet stupňů závisí na implementaci
  - Architektura instrukcí Load – Store.
    - (ISA = Instruction Set Architecture)
  - Pro pohyblivou čárku se používá koprocesor.

MIPS - kódování instrukcí

- MIPS má 3 základní formáty instrukcí
  - Formát I (Immediate).
  - Formát J (Jump).
  - Formát R (Register).



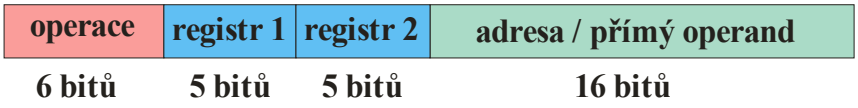
Formát I

- Obsahuje přímý operand o délce 16 bitů.
  - Offset pro výpočet adresy (signed, tj. < - 32768; + 32767 >), např. instrukce

```
LW $1,-0x0200($2)     # $1 ← [$2-200]
```

  - Konstanta pro přímé operace, např. instrukce

```
ADDIU $1,$2,0x1234    # $1 ← $2 + 1234
```

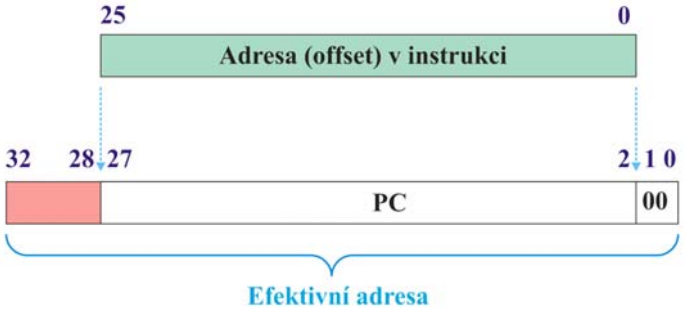


Formát J (1)

- Formát pro skokové instrukce. Obsahuje 26 bitový offset pro cílovou adresu.
  - Cílová adresa = PC[32..27] || offset.
  - Offset se posouvá o 2 bity doleva, tj. lze adresovat v rozsahu segmentu o velikosti 2^{28} (= 256 MB).
  - Příklad: instrukce `J lab_1` .
  - Pro skok na absolutní adresu se musí použít nepřímé adresování registrem (např. `JR $1`).

Formát J (2)

- Formát pro skokové instrukce. Obsahuje 26 bitový offset pro cílovou adresu.
  - Cílová adresa = PC[32..27] || offset.
  - Offset se posouvá o 2 bity doleva, tj. lze adresovat v rozsahu segmentu o velikosti 2^{28} (= 256 MB).
  - Příklad: instrukce `J lab_1` .
  - Pro skok na absolutní adresu se musí použít nepřímé adresování registrem (např. `JR $1`).



Formát R

- Obsahuje pole pro určení 3 registrů (source, target, destination), pole pro délku posuvu a rozšířený operační kód.
  - Příklad:
    instrukce `ADD $1,$2,$3  # $1 ← $2 + $3` .

operace	registr 1	registr 2	registr 3	posuv	operace
6 bitů	5 bitů	5 bitů	5 bitů	5 bitů	6 bitů

Pseudoinstrukce (1)

- Rozšiřují instrukční soubor, dostupný programátorovi.
- Nahrazují se jednou nebo více instrukcemi ve strojovém kódu ⇒ překládají se **1:n** .

– Příklad:

```
MOVE   $2,$1 # $2 ← $1
```

přeloží se jako

```
ADDU $2,$0,$1 .
```

Poznámka: Registr **$0** je „zero“ – čte se jako 0x0, zápis se neprovede.

Pseudoinstrukce (2)

- Rozšiřují instrukční soubor, dostupný programátorovi.
- Nahrazují se jednou nebo více instrukcemi ve strojovém kódu ⇒ překládají se **1:n** .

– Příklad:

```
LW   $10,var_1
```

přeloží se jako

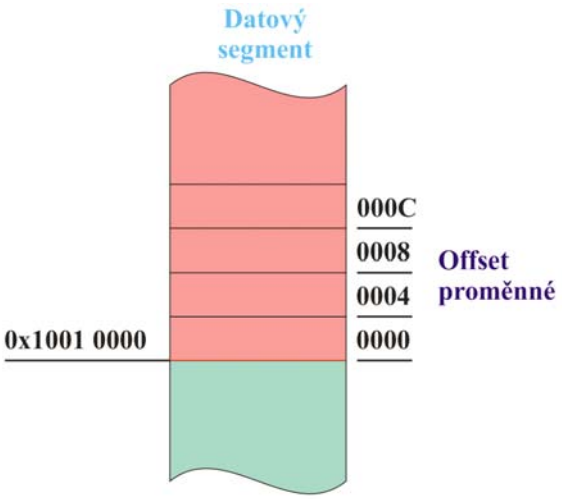
```
LUI $1,0x1001
LW   $10,8($1)   .
```

Do registru $1 se uloží báze datového segmentu (0x1001 0000).

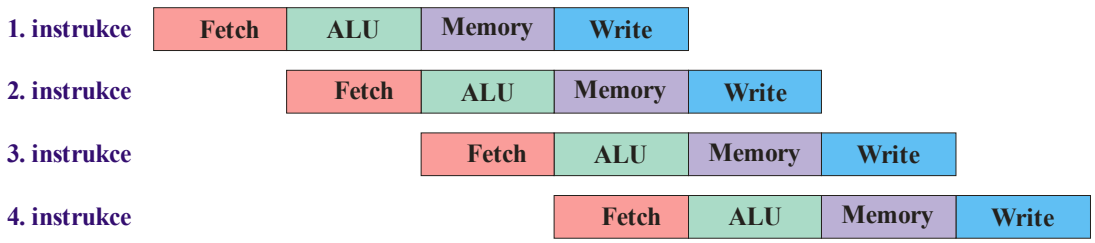
Proměnná **var_1** leží na adrese 8 vzhledem k začátku segmentu.

Datový segment

- Registr $1 (alias $at) se nastaví na začátek datového segmentu.
- V instrukci **`LW`** je offset proměnné vzhledem k začátku segmentu.



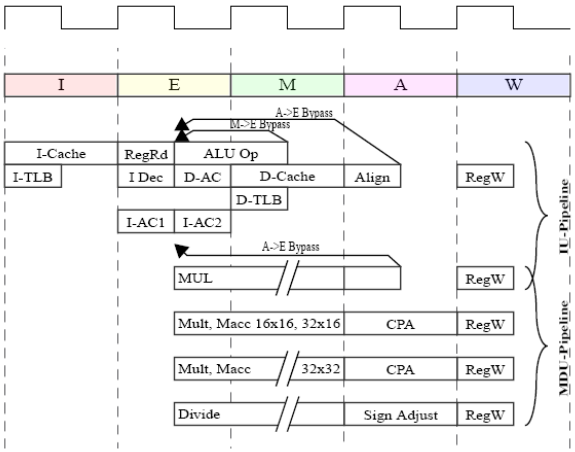
Důsledky proudového zpracování



- Výsledky operací ALU jsou k dispozici až po několika taktech CLK.

Pipeline u MIPS

5 fází: **I**nstruction, **E**xecution, **M**emory, **A**lign/Accumulate, **W**riteback



I-Cache	: I$ Tag and Data read
I-TLB	: I-TLB Look-up
I Dec	: Instruction Decode
RegRd	: Register file read
I-AC1 I-AC2	: Instruction Address Calculation stage 1 and 2
ALU Op	: Arithmetic Logic and Shift operations
D-AC	: Data Address Calculation
D-Cache	: D$ Tag and Data read
D-TLB	: D-TLB Look-up
Align	: Load data aligner
RegW	: Register file write or HI/LO write
MUL	: The MUL instruction. Uses MDU-Pipeline write Reg file
CPA	: Carry Propagate Adder
Mult, Macc	: Multiply and Multiply Accumulate instructions
Divide	: Divide instructions
Sign Adjust	: Last stage of Divide is a sign adjustment
//	: One or more stall cycles.

Zpoždění skokových instrukcí (1)

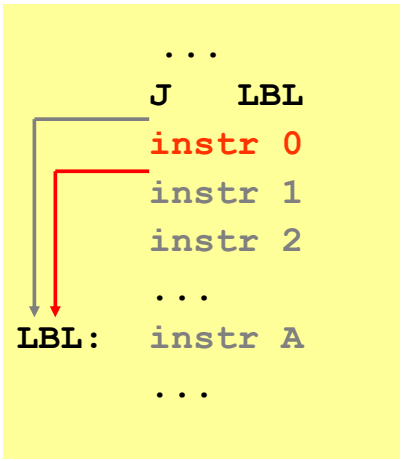
- MIPS simuluje zpoždění skoků o 1 instrukci (provede se 1 instrukce za **`J LBL`**).

```
        ...
        instr 0
        J   LBL
        instr 1
        instr 2

        ...
LBL:    instr A
        ...
```

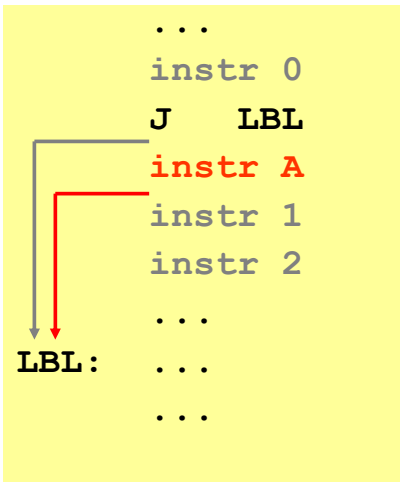
Zpoždění skokových instrukcí (2)

- Řešení 1: za **`J LBL`** se přesune předchozí instrukce.



Zpoždění skokových instrukcí (3)

- Řešení 2: za **`J LBL`** se přesune instrukce z cíle skoku.
  - Může vést k chybě, pokud se na **`LBL`** skáče z různých míst.



Zpoždění čtení dat z paměti

- SPIM simuluje zpoždění čtení dat z paměti o 2 instrukce ⇒ data v registru jsou k dispozici po provedení 2 instrukcí za **`LW $n,var`**.

```
        ...
LW      $10,var_1
        nop
        nop
ADDI    $11,$10,0xF0
        ...
```

Dva vložené `nop` zpomalují výpočet. Vhodnější je použít na jejich místě instrukce z jiného místa programu.

Assembler MIPS (1)

- Používají se „alias“ názvy registrů.

Číslo registru	Alias název	Použití
0	$zero	dummy registr
1	$at	rezervováno pro assembler
2	$v0	návratové hodnoty funkcí
3	$v1	
4	$a0	předávání argumentů
...	...	
7	$a3	
8	$t0	neukládané registry
...	...	
15	$t7	

Číslo registru	Alias název	Použití
16	$s0	ukládané registry
...	...	
23	$s7	
24	$t8	neukládané registry
25	$t9	
26	$k0	rezervováno pro O.S.
27	$k1	
28	$gp	global pointer
29	$sp	stack pointer
30	$fp	frame pointer
31	$ra	return address

Assembler MIPS (2)

- Datová sekce `.data`
- Kódová sekce `.text`

- Program se spouští skokem do procedury instrukcí `JAL main` ⇒ v $ra je návratová adresa do systému.
- Ukončení výpočtu: Instrukce `J $ra` vrací řízení do O.S.

```
        .data
var1:   .word  0x12345678
        ...

        .text
        .globl main
main:   ...
        ...
        j      $ra
        nop
```


Vstupy a výstupy

- SPIM simuluje několik základních systémových služeb pro IO operace.
- Volání služeb instrukcí `syscall`.
  - V registru $v0 musí být číslo služby.
  - V registrech $a0 - $a3 případné další argumenty.

Příklad IO

- print_string
  - Číslo služby = 4.
  - Adresa textu = $a0.
  - Text končí 0x00.

```
        .data
t_nazd: .asciiz "Nazdar\n"

        .text
        .globl main
main:   li  $v0,4       #číslo služby
        la  $a0,t_nazd  #adresa textu
        syscall         #volání služby
        nop
        j   $ra         #návrat
        nop
```

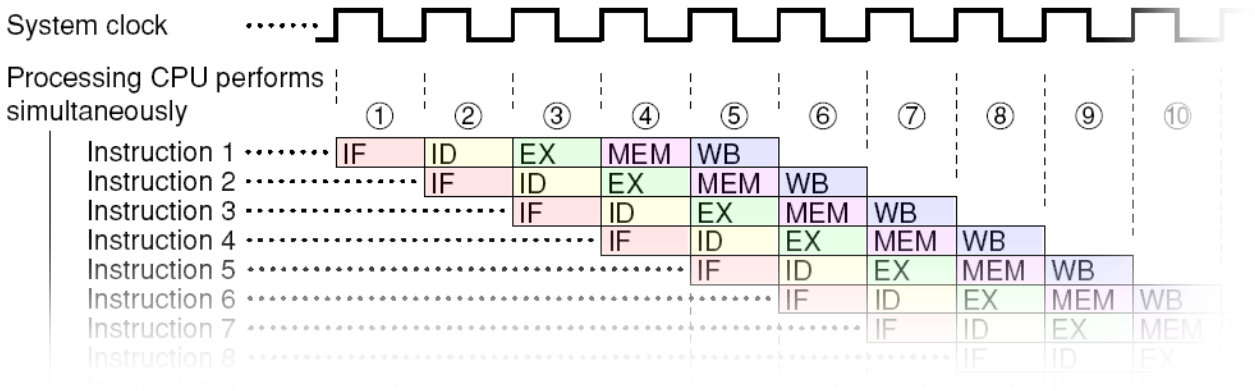

Řazení fází u procesoru NEC V850

- Procesor NEC V850 má 32bitovou architekturu RISC.
- Datasheet podrobně dokumentuje způsob řazení jednotlivých fází instrukčního cyklu. Je proto vhodný jako doplňující materiál pro studium vlastností procesorů tohoto typu.
- Podrobnosti viz `www.renesas.eu/products/mpumcu/v850/index.jsp` .

RENESAS
Everywhere you imagine

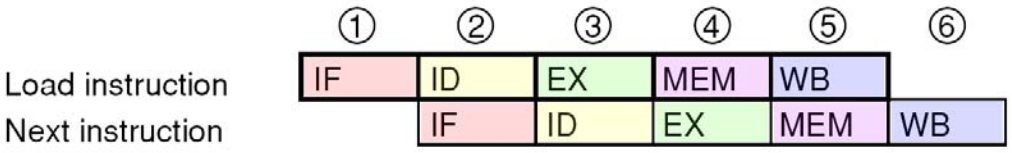
Řazení fází u procesoru NEC V850 (1)

- Obecně má zpracování instrukce 5 fází (stupňů pipeline):
  - Fetch čtení kódu instrukce,
  - Decode dekódování,
  - Execute provedení,
  - Mem práce s pamětí,
  - Write Back zápis do registrů.

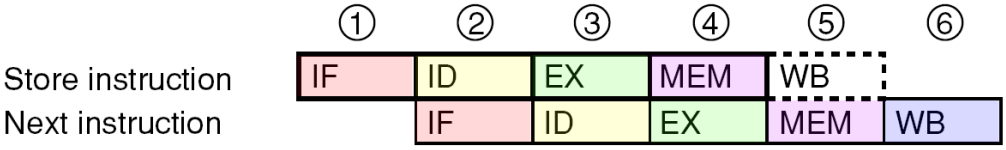


Řazení fází u procesoru NEC V850 (2)

- Instrukce typu Load (čtení z paměti do registru):
  - Provádí se ve všech 5 fázích (taktech).

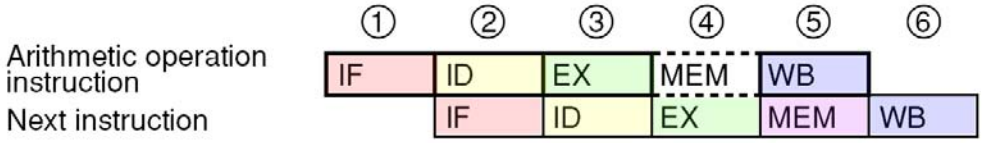


- Instrukce typu Store (zápis dat do paměti):
  - Nemá fázi WB.



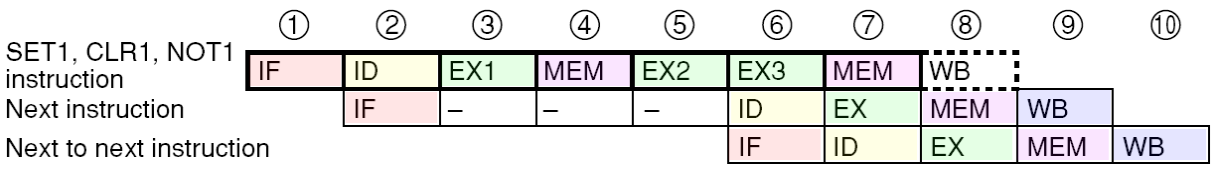
Řazení fází u procesoru NEC V850 (3)

- Instrukce typu Add (arimetika registr → registr):
  - Nemá fázi MEM (operandu se čtou a zapisují do registrů).



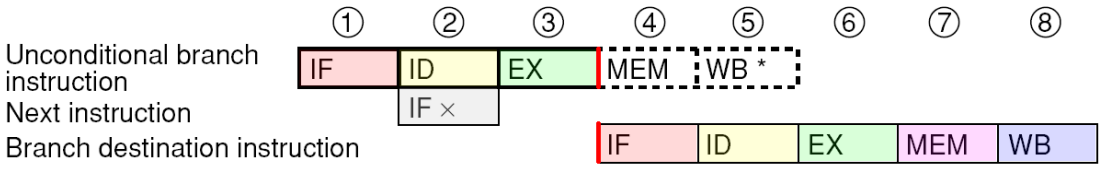
Řazení fází u procesoru NEC V850 (4)

- Bitové operace s daty v paměti (určené pro práci se SFR):
  - Jsou typu Read – Modify – Write. Operand se čte z paměti, modifikuje a zapíše do paměti.
  - Nemá fázi WB.
  - Provádí se v 7 fázích → následující instrukce vkládá 3 čekací takty (Stall).



Řazení fází u procesoru NEC V850 (5)

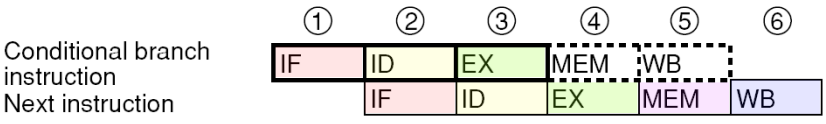
- Skoková instrukce (nepodmíněný skok):
  - Nemá Mem a WB.
  - Z paměti se přečte instrukce za JMP a zruší se.
  - Některé procesory (též MIPS) instrukci bezprostředně za JMP **nezruší**, nýbrž ji normálně provedou **(!)** (zpožděný skok).
  - Instrukce z cílové adresy se čte se zpožděním 2 taktů.



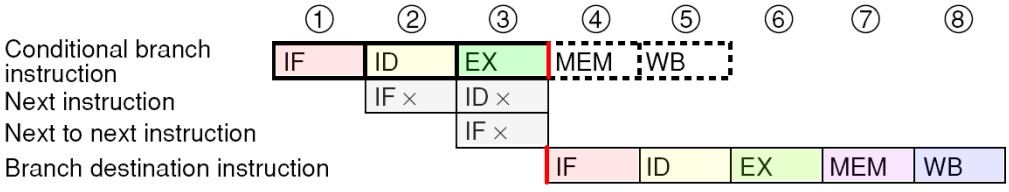
Řazení fází u procesoru NEC V850 (6)

- Skoková instrukce (podmíněný skok):
  - Není-li podmínka splněna, nenaruší se pipeline.
  - Je-li podmínka splněna, čtou se dvě instrukce před provedením skoku ⇒ ztrácí se 2 takty.

When the condition is not realized

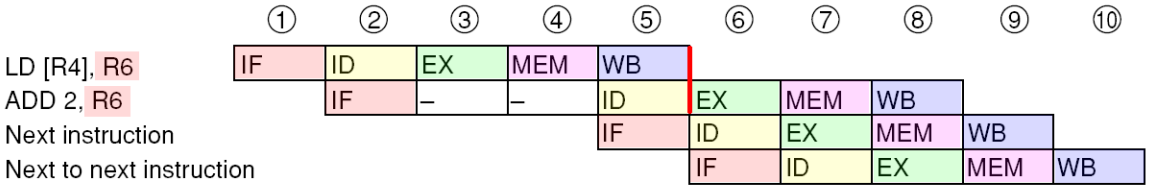


When the condition is realized



Řazení fází u procesoru NEC V850 (7)

- Použití operandu čteného předchozí instrukcí z paměti:
  - Následující instrukce musí počkat na dokončení fáze WB.
  - Některé procesory (také MIPS) **nečekají** automaticky – instrukce potom dává nesprávný výsledek **(!)**.

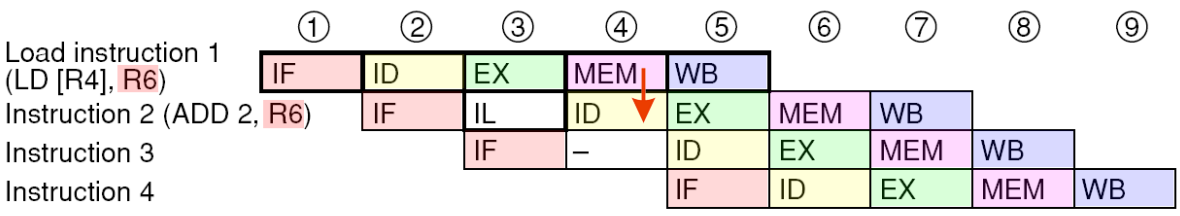


IL: Idle inserted for data wait by interlock function

–: Idle inserted for wait

Řazení fází u procesoru NEC V850 (8)

- Použití operandu čteného předchozí instrukcí z paměti - **forwarding**:
  - Data z paměti jsou zavedena přímo na vstup operační jednotky.
  - Nemusí se čekat na dokončení fáze WB.



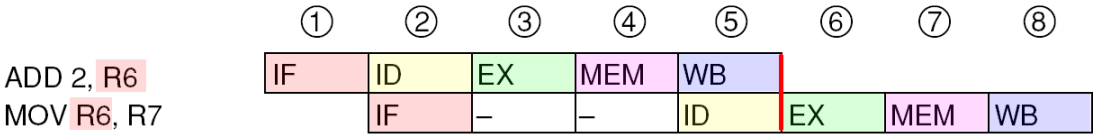
IL: Idle inserted for data wait by interlock function

–: Idle inserted for wait

↓: Short path

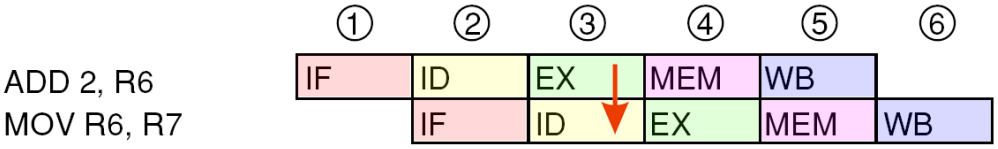
Řazení fází u procesoru NEC V850 (9)

- Instrukce pracuje s výsledkem předchozí operace:
  - Musí se počkat na dokončení fáze WB.



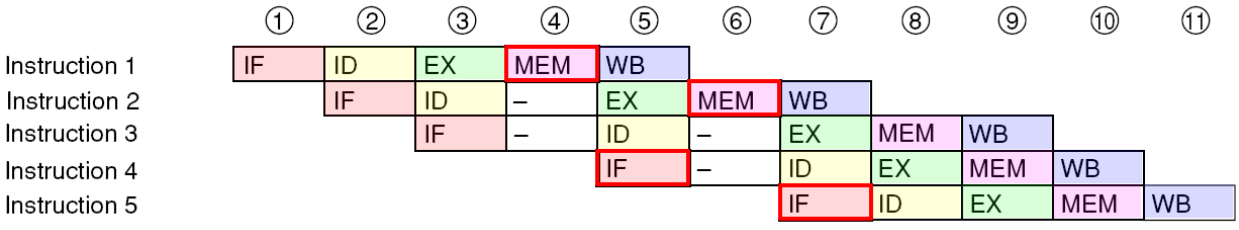
–: Idle inserted for wait

- **Short Pass**:
  - Výsledek předchozí operace je zaveden přímo na vstup operační jednotky.
  - Nemusí se čekat na dokončení fáze WB.



Řazení fází u procesoru NEC V850 (10)

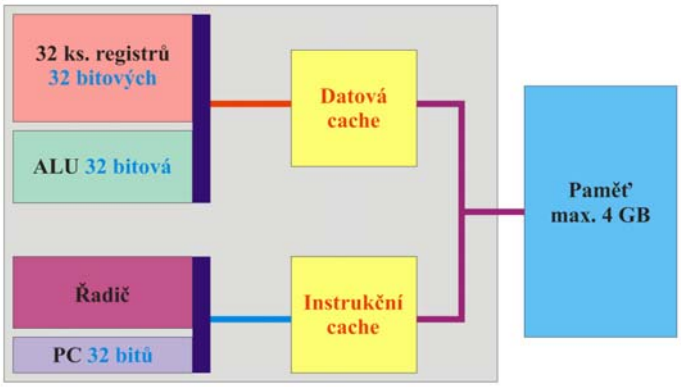
- Kolize na sběrnici:
  - Fáze MEM a IF různých instrukcí potřebují současně přístup do paměti.



–: Idle inserted for wait

Řazení fází u procesoru NEC V850 (11)

- Řešení kolize na sběrnici – Harwardská architektura:
  - Datová a instrukční cache jsou samostatné – lze současně číst/zapisovat data a číst instrukce.
  - Některé počítače (mikrokontroléry) mají oddělenou i datovou a kódovou paměť.



Řazení fází u procesoru NEC V850 (12)

RENESAS Everywhere you imagine

- Řešení kolize na sběrnici – Harwardská architektura:
  - Datová a instrukční cache jsou samostatné – lze současně číst/zapisovat data a číst instrukce.