

Dynamické plánování instrukcí

AVS – Architektury výpočetních systémů

Týden 2, 2024/2025

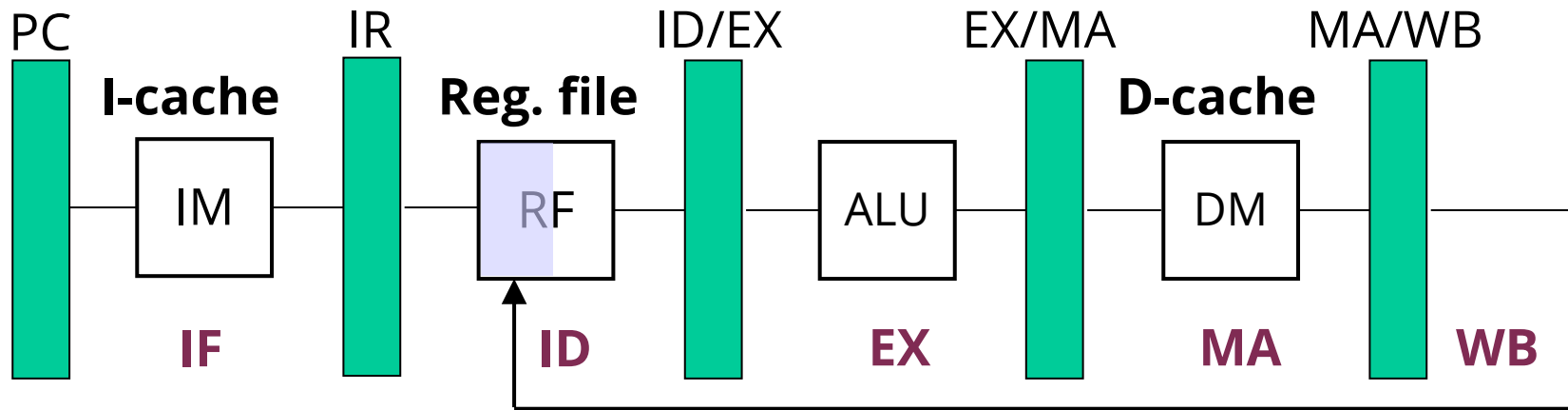
Jirka Jaroš

Vysoké učení technické v Brně, Fakulta informačních technologií
Božetěchova 1/2, 612 66 Brno - Královo Pole
jarosjir@fit.vutbr.cz



OPAKOVÁNÍ

Ref: [Henessy, str 147-162,](#)
[Appendix C](#)



Cena za řetězení

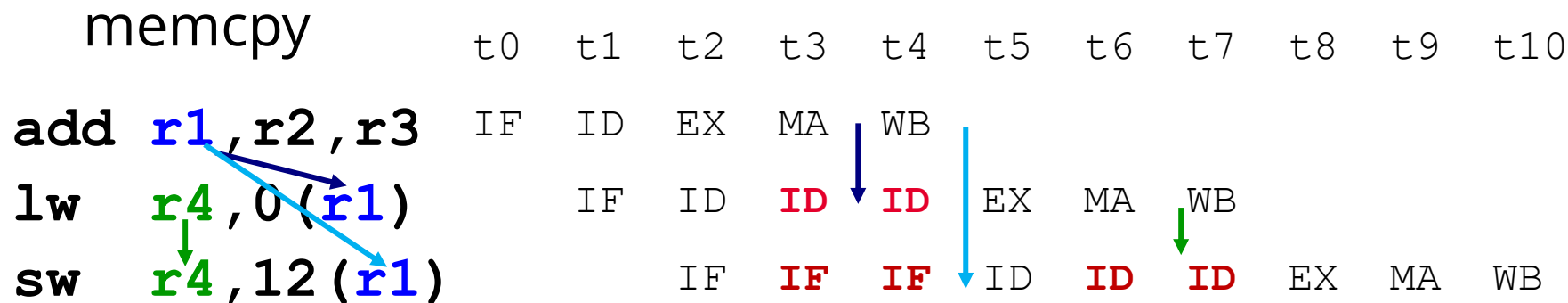
$$S_{\infty} = \frac{k\tau}{(\tau + t_d)}$$

Průměrná cena za konflikty

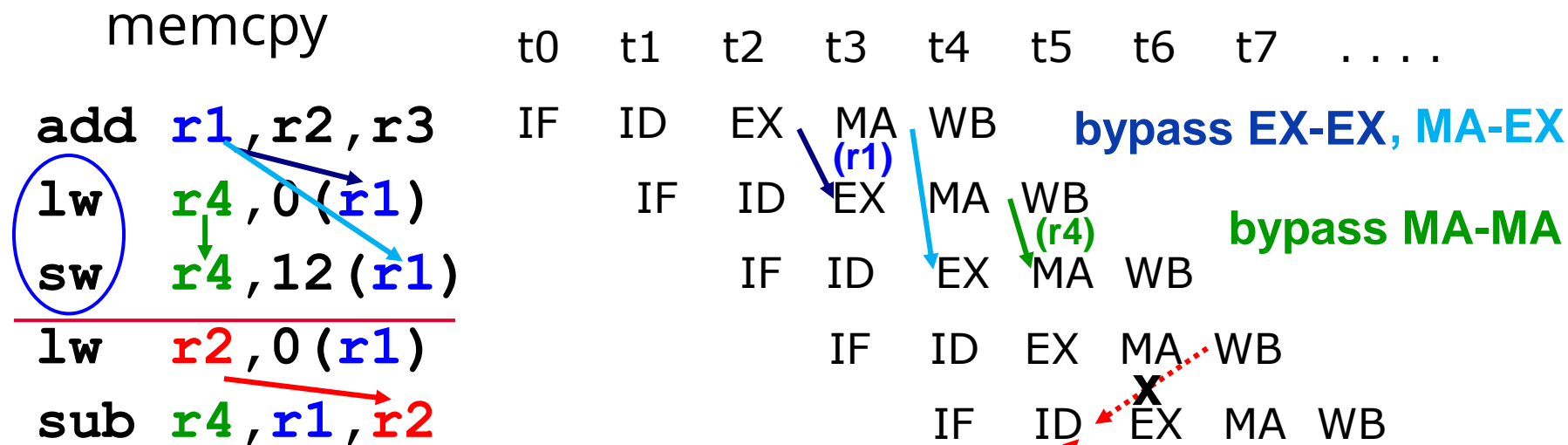
$$S_{N \rightarrow \infty} = \frac{k\tau}{(\tau + t_d)(1+q)} \rightarrow \frac{k}{1+q} = \frac{k}{CPI}$$

- **Pravé** (true dependencies), též postupové (flow):
 - **RAW** (Read After Write), instrukce načítá nebo počítá s výsledkem generovaným dřívější instrukcí.
 - **Řešení**: čekání na výsledek, předávání dat, přeuspořádání operací kompilátorem tak, aby byly operace dokončeny v předstihu.
 - **Dva typy**:
 - načtení – použití
 - výpočet – použití
- **Nepravé** (false/name dependencies) – vznikají změnou pořadí vykonání instr.
 - **WAR** – protiproudé (anti)
 - **WAW** – výstupní (output)
 - instrukce zapisuje tam, odkud předchozí instrukce četla nebo kam zapsala. Nejde o tok dat, ale o konflikt jmen.
 - **Řešení**: přejmenováním.

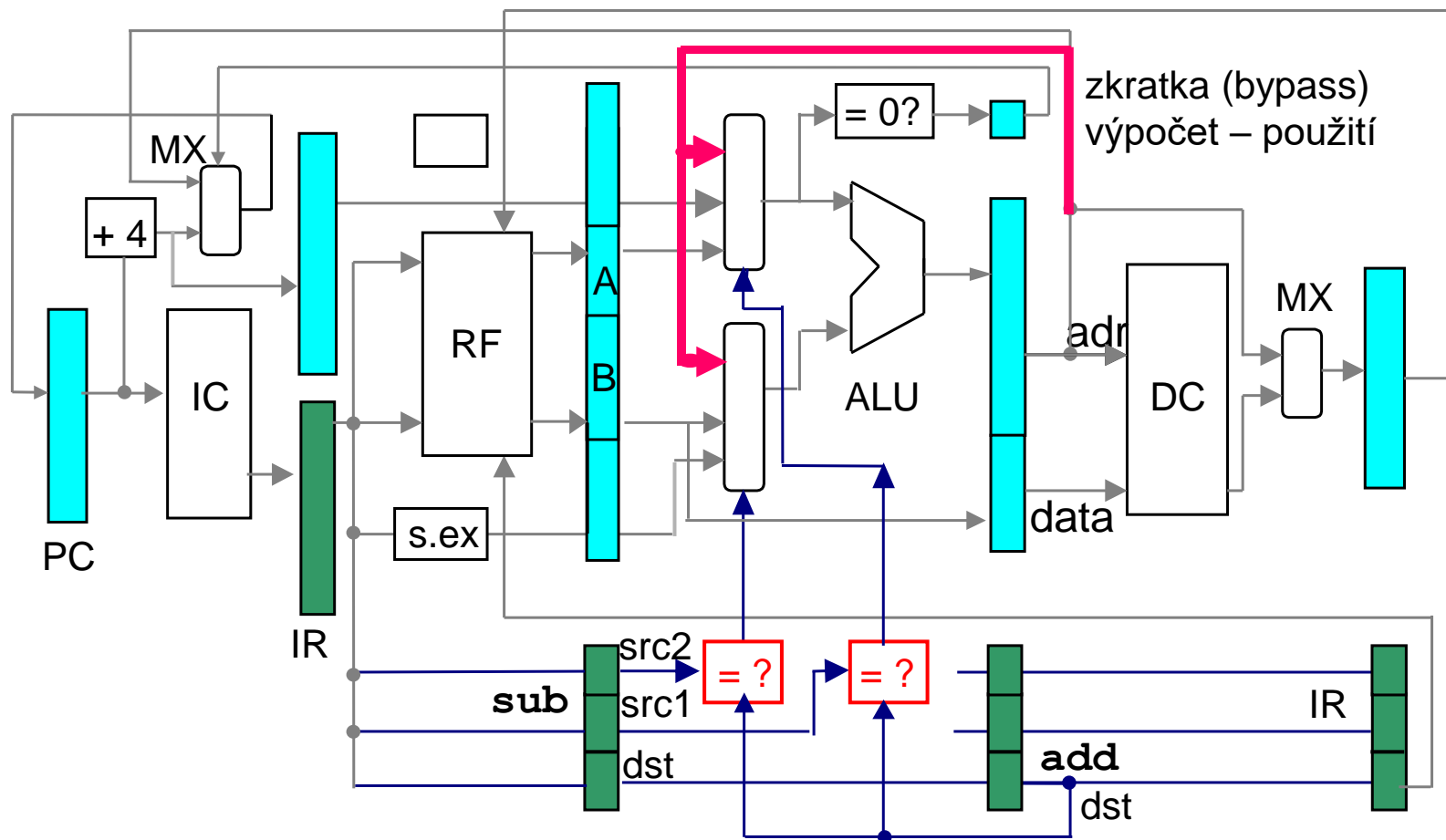
Konflikt vzniká, když pořadí RAW, WAR nebo WAW není dodrženo.



- **r1** sice vznikne ve fázi EX, ale do registru se dostane až na začátku fáze WB
 - Následující instrukce čtou R1 ve fázi ID
 - **Pozor, nemohu mít dvě instrukce ve stejné fázi**
- **r4** se sice načte ve fázi MA, ale do registru se dostane až na začátku fáze WB
 - Následující instrukce čte již ve fázi ID
 - Jelikož je zpoždění na předchozí instrukci, stráví instrukce SW (stall) ve fázi IF 3 taky a ve fázi ID také 3 takty



- (r1) = výsledek add, který teprve bude zapsán do reg. r1
- Kopírování dat lw – sw bez pokuty.
- **Předávat data do minulého taktu nelze !!**
sub musí počkat další tak ve fázi EX!



WAR

i0: fdiv f0, f2, f4
i1: fadd f6, f0, f8
i2: fsub f8, f10, f14

Může vzniknout jen při změně pořadí provedení instrukcí, **i2 před i1**: i2 změní chybně hodnotu f8 pro i1, čekající na f0

WAW

i1: fmul f1, f2, f3 ; IF ID EX EX EX EX CA **WB**
i2: fadd f1, f4, f5 ; IF ID EX EX CA **WB**

pořadí zápisů změněno

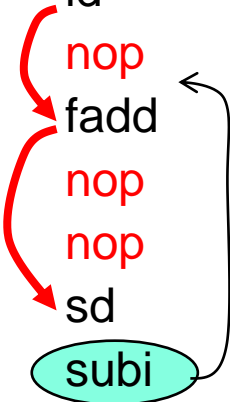


i1: fmul f1, f2, f3
i2: fadd f6, f4, f5

odstranění konfliktu přejmenováním cílového registru

V klasické 5 stupňové lince FX konflikty WAR a WAW vzniknout nemohou.


```
loop:  ld      f0, 0(r1)      ; f0 = prvek vektoru
      nop
      fadd    f4, f0, f2     ; přičti skalár z f2
      nop
      nop
      sd      f4, 0(r1)     ; ulož výsledek zpět
      subi   r1, r1, #8     ; dekrementuj ukazovátka o 8 bajtů
      nop
      bnez   r1, loop       ; skoč když r1 ≠ 0
```



**Naivní, jen
vkládání NOP,
9 taktů/prvek**

Kompilátor
přehodí subi
na lepší
místo

Producent	Konzument	Meziop. latence
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

Ref: [Hennessy, str 147-162](#)

```

loop:  ld      f0, 0(r1)
       fadd    f4, f0, f2
       sd      0(r1), f4
-----
       ld      f0, -8(r1)
       fadd    f4, f0, f2
       sd      -8(r1), f4
-----
       ld      f0, -16(r1)
       fadd    f4, f0, f2
       sd      -16(r1), f4
-----
       ld      f0, -24(r1)
       fadd    f4, f0, f2
       sd      -24(r1), f4
-----
       subi    r1, r1, #32
       bnez    r1, loop
    
```

27 taktů včetně prázdných
6,8 taktu/element



```

loop:  ld      f0, 0(r1)
       ld      f6, -8(r1)
       ld      f10, -16(r1)
       ld      f14, -24(r1)
       fadd    f4, f0, f2
       fadd    f8, f6, f2
       fadd    f12, f10, f2
       fadd    f16, f14, f2
       sd      0(r1), f4
       sd      -8(r1), f8
       sd      -16(r1), f12
       subi    r1, r1, #32
       sd      8(r1), f16
       bnez    r1, loop
    
```

14 taktů, žádný prázdný
3,5 taktu/element

**Změna
pořadí
zpracování
instrukcí**

**4 iterace
prolnuty**

**Odstraněny
všechny
RAW
konflikty**

SUPERSKALÁRNÍ PROCESORY

Ref: [Henessy, str 167-193, 233-247, Appendix C](#)

Jak zrychlit dobu výpočtu?

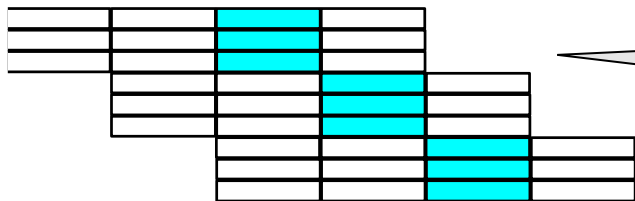
$$\text{doba výpočtu} = IC * CPI * T$$

- **IC** = Instruction Count, počet provedených instrukcí – optimalizace kódu.
- Snížení doby taktu **T**
 - větší počet stupňů linky, až 30 (**hlubší linka**, superřetězení)
 - ale velké pokuty, vyšší příkon
 - dolní hranice: doba zpoždění hradel a příkon
- Snížení **CPI**, tj. zvýšení **IPC**:
 - více instrukcí (až **m**) v jednom stupni
 - **m**-cestný **superskalární procesor** (**tedy širší linka**)
 - složitější, nižší možný kmitočet
 - **reálně dosažitelná hodnota IPC** (kolik instrukcí průměrně končí v jednom taktu) **je vždy značně nižší než m**

Moderní mikroprocesory používají oba přístupy.

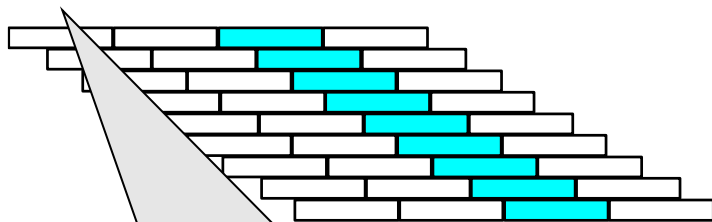
Zpoždění 1 stupně skalárního procesoru τ je rozděleno na n kratších taktů, počet cest navýšen na m nebo obojí.

Superskalární CPU, $m = 3$

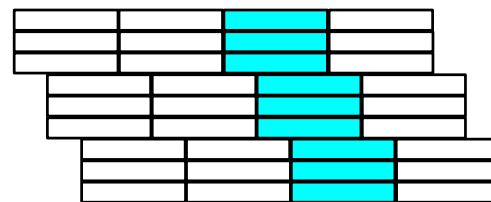


$$t_1 = (\tau + t_d) / m$$

Superřetězená CPU, $n = 3$



$$t_1 = [\tau / n + t_d]$$



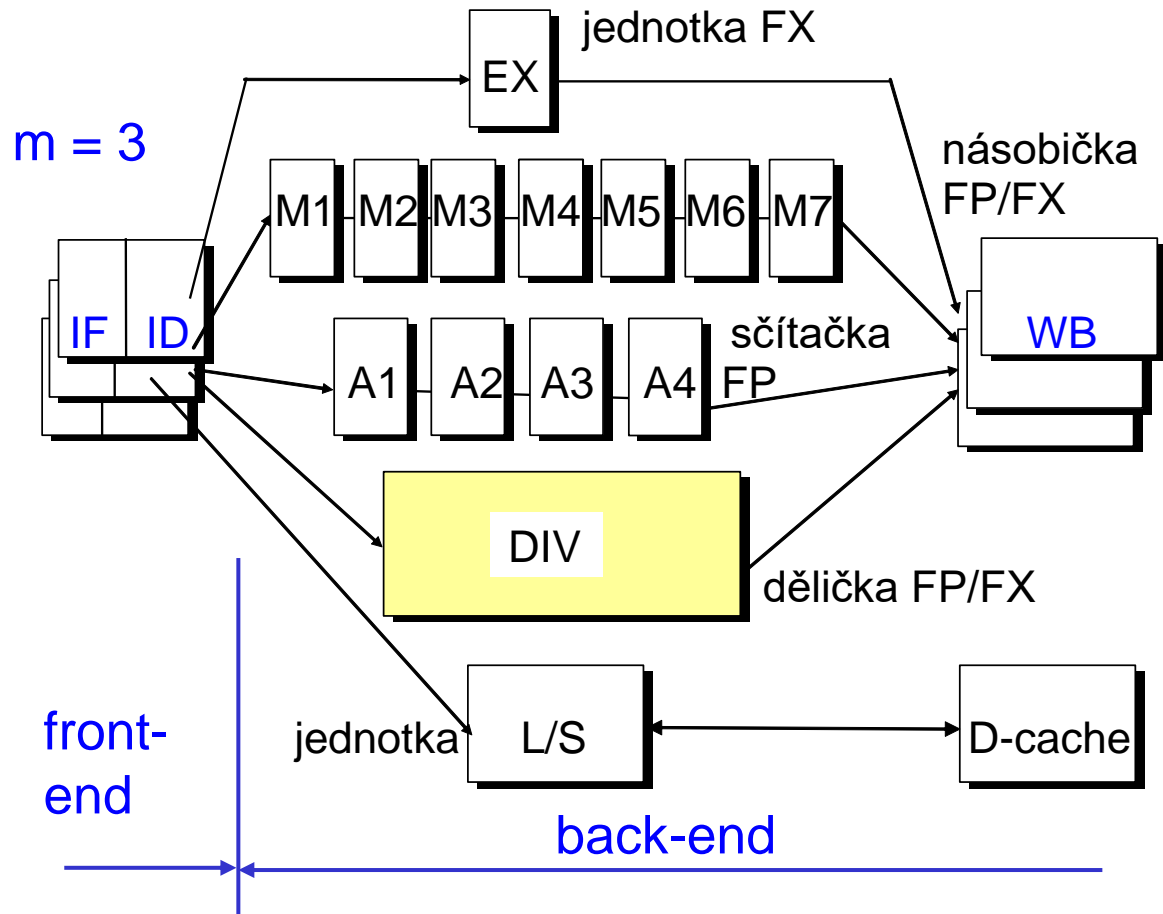
Superskalární i superřetězená CPU, $m = n = 3$

$$t_1 = [\tau / n + t_d] / m$$

- **Front-end** odpovídá stupňům **IF** a **ID**:
 - Načítá a dekoduje několik instrukcí najednou, počet se mění dynamicky.
 - *m*-cestný superskalár vydává až *m* instrukcí do funkčních jednotek v jednom taktu.
- **Back-end** odpovídá stupňům **EX**, **MA**, **WB**:
 - Provádí a ukládá výsledky několika instrukcí souběžně.
 - Některé stupně jsou rozděleny na pod stupně.
 - Počet (skupin) funkčních jednotek je minimálně *m*, ale počet propojovacích cest roste jako m^2 .

Dnes *m* nejvýš 6 až 9, ale důležité je **IPC**: kolik instrukcí průměrně končí za 1 takt a to je o dost méně než *m*.

- **Dělení dle způsobu jakým instrukce opouštějí front-end:**
 - **Podle pořadí** v programu, po vyřešení konfliktů (in-order, INO), jednoduchý HW.
 - **Mimo pořadí** (out-of-order, OOO). Nepravé konflikty vyřešeny **přejmenováním v HW**, RAW řešeny **čekáním** rozpracovaných instrukcí. Zápis výsledků v původním pořadí zajištěn **seřazovací pamětí (ROB)**.
- **Příklady:**
 - **INO**: první superskalární procesory (Pentium, DEC Alpha 21164), ale i nejnovější (IBM Power6, Intel Atom, ARM nebo Intel Xeon Phi).
 - **OOO**: Intel P6, Pentium4, Intel Core, ..., až Rocket Lake či Zen



- Paralelní řetězené linky
- Jednotlivé stupně mohou mít různé zpoždění nebo propustnost instrukcí za takt
- Front end opouští až m instrukcí v jednom taktu
- Samostatná funkční jednotka L/S místo stupně MA

- **Paralelní řetězené linky** (INO i OOO)
 - Časový i prostorový paralelismus (paralelní načítání, dekodování, vydávání instrukcí do FJ, jejich paralelní provádění a dokončování).
- **Přejmenování registrů v HW** (OOO)
 - Odstraní konflikty WAR a WAW
- **Dynamické plánování instrukcí out-of-order** (OOO)
 - Po dekodování čekají instrukce na své operandy, které se tvoří. Jakmile jsou operandy připraveny, spustí se operace.
 - Instrukce, včetně přístupů do paměti, jsou zpracovávány v jiném pořadí oproti pořadí v programu (OOO).
- **Seřazovací paměť** (OOO)
 - Stupeň WB pomocí ní zajišťuje ukládání výsledků v pořadí určeném zdrojovým kódem.
- **Spekulativní zpracování instrukcí** (OOO)
 - Spekulace, že skok dopadne podle predikce nebo že dopředu načtená data se již nezmění.

	max. vydaných instr./takt
• Intel Pentium Pro, Pentium II, III (P6)	2/3
• Pentium 4 (NetBurst)	3/6
• Compaq Alpha EV7	6
• AMD Opteron	9
• IBM PowerPC 7400 (G4)	6
• HP PA 8700	4
• MIPS R14000	4
• Intel Core	4/5
• Intel Nehalem	5/6
• Intel Sandy Bridge – Cascade Lake	6/6
• AMD Zen	4/6FX, 4FP
• ARM Cortex A15	3
• Apple M1	8

dekódovaných instrukcí x86 (CISC) za takt

vydaných µoperací (RISC) za takt

http://www.agner.org/optimize/instruction_tables.pdf

DYNAMICKÉ PLÁNOVÁNÍ INSTRUKCÍ

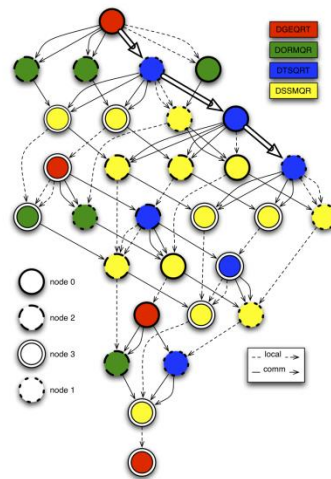
Instrukce jsou vydávány do FJ a prováděny **mimo pořadí** v programu, pokud mezi nimi **nejsou konflikty** a **FJ** jsou **volné**.

1. ScoreBoarding (Thorntonův algoritmus, 1964)

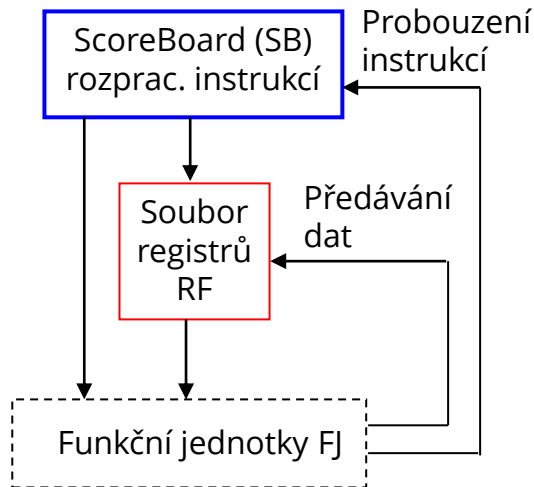
- Registruje všechny **konflikty** (RAW, WAW, WAR) v **tabulce rozpracovaných instrukcí** a udržuje jejich skóre (SB).
- SB vydá instrukce dál jen když nejsou v konfliktu s ostatními instrukcemi v SB. **Přejmenování registrů neprobíhá**.

2. Rezervační stanice (Tomasulův algoritmus, 1967)

- Konflikty WAW a WAR se řeší přejmenováním
- **Rezervační stanice RS** (bufery) umožňují odložit čekající instrukce a pracovat dopředu na dalších – tím řeší RAW.
- Rezervační stanice centrální (instruction window) nebo individuální u FJ či skupinové pro skupiny FJ.



http://users.utcluj.ro/~sebestyen/_Word_docs/Cursuri/SSC_course_5_Scoreboard_ex.pdf



Formát jedné položky **ScoreBoard (SB)**:

- **stav** instrukce (vydána do FJ, operandy načteny, hotová)
- funkční jednotka **FJ busy?**
- **operace** FJ
- **dst** (**adresa** cílového registru)
- **src1** (**adresa** zdrojového reg. 1)
- bit **V1** (operand 1 platný?)
- **src2** (**adresa** zdrojového reg. 2)
- bit **V2** (operand 2 platný?)

Formát **registrů v RF(dst)**:

Rezervační bit **V** | value

0 – neplatný (**rezervovaný**)
1 – platný (někdo, ale ještě může potřebovat)

Valid bit V1 a V2

0 – neplatný **nebo** použitý
1 – platný, ale ještě nepoužitý

- **Vydání nové instrukce – Fáze ID**

- **Rezervace cílového registru v RF** (Serializace zápisů – WAW)

- Registr **platný (V = 1)**, zapíše V = 0 a vloží položku do SB.
Zde neřeším, že hodnotu ještě někdo může potřebovat.
 - Registr již **neplatný (V = 0)** – nějaká instrukce do něj právě generuje výsledek – **čekáme** až dokončí předchozí instrukce.

- **Vydání instrukce** (Eliminace RAW)

- Jsou-li oba zdrojové operandy **platné (V1 a V2 = 1)**,
 - odešli op-kód instrukce do FJ
 - odešli adresy src1 a src2 do RF a odtud jejich hodnoty do FJ
 - **vynuluj příznaky V1 a V2 ve SB a změň stav instrukce** (operandy načteny)
 - Pokud není některý operand platný, **čekej**

- **Vykonání instrukce – Fáze EX**

- Výsledek a adresa dst registru se objeví na výstupu FJ

- **Zápis výsledků do registru – Fáze WB** (Eliminace WAR)
 - Dokud se **shoduje dst** výsledku se **src1** nebo **src2** v nějaké instrukci v SB s bitem **V1** nebo **V2 = 1** („platný a ještě nepoužitý“)
 - musí **se zápis výsledku do registru RF(dst)** pozdržet i když je registr pro tento výsledek již rezervován ($V = 0$).
 - stávající obsah RF(dst) totiž ještě nepřečetla nějaká čekající instrukce.
 - Jakmile jsou relevantní bity **V1** a **V2** v SB **vynulovány**
 - zapíšeme **výsledek** a **V = 1** do RF(dst), tedy data platná.
 - **prohledáme SB** a změníme bit **V1** nebo **V2** na 1 ve všech položkách SB, které čekaly na výsledek ($\text{src1} \mid \text{src2} = \text{dst}$)
- Jakmile je instrukce dokončena, její záznam je smazán z tabulky score.

Závěr: konflikty RAW, WAR a WAW se u této varianty řeší čekáním.

Registry

Name	Valid	Value
R1	0	?
R2	1	50
R3	1	30
R4	1	40
R5		
R6		

op1 **r1**, r2, r3

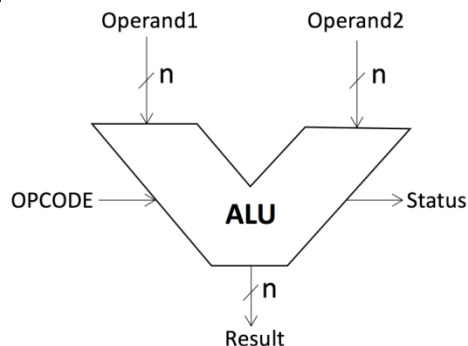
op2 r2, **r1**, r4

op3 r6, **r3**, **r1**

op4 **r1**, **r2**, **r3**

op5 r7, r8, **r1**

op6 **r1**, r5, r4



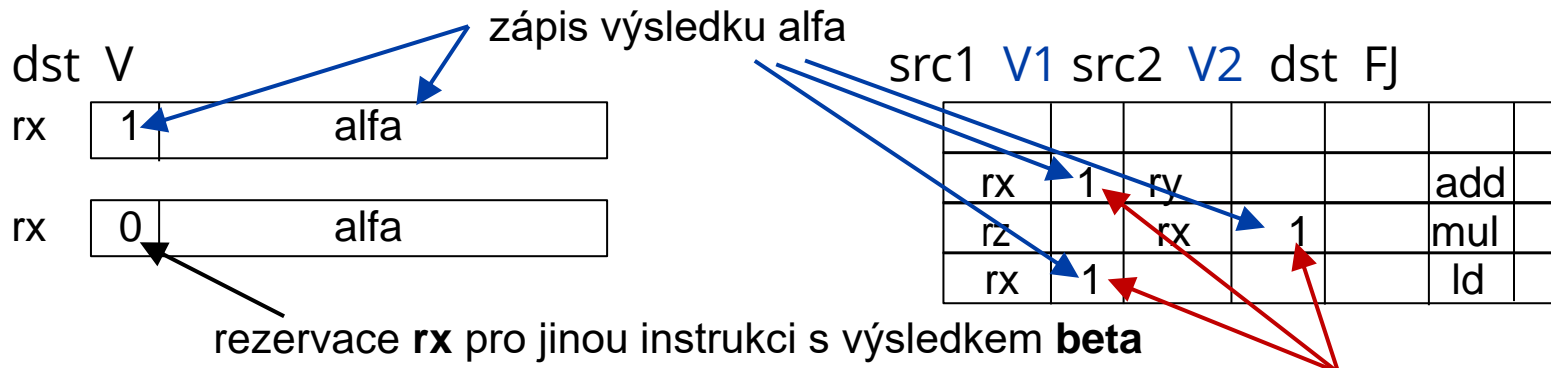
Score Board

Dst	S1	V1	S2	V2

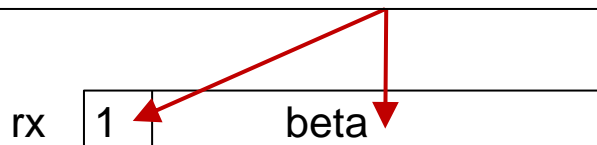
ScoreBoard – bez přejmenování registrů: je třeba signalizovat

- **okamžik čtení** src operandů čekajícími instrukcemi opx
 $0 \rightarrow \text{SB}(\text{opx}).V1/V2$ pro **WAR**
- a **zápisu** cílového registru $1 \rightarrow \text{RF}(\text{dst}).V$ pro **WAW**.

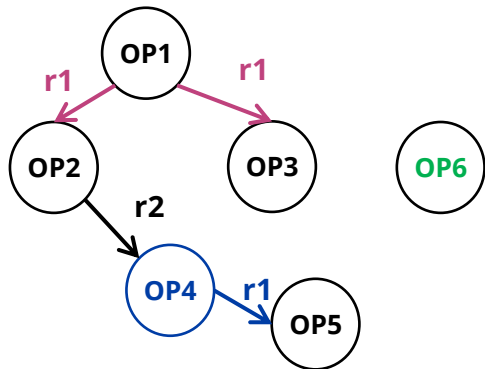
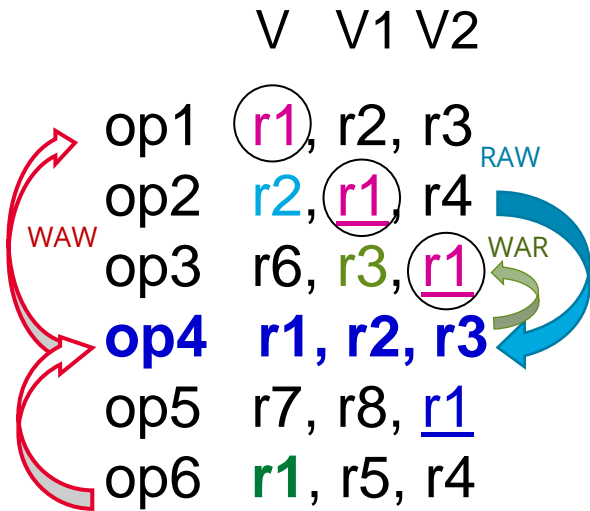
SB



výsledek beta lze zapsat do rx až bylo **alfa** přečteno všemi čekajícími instrukcemi ($1 \rightarrow 0$ ve V1/V2)



nelze rezervovat pro gama, dokud je $V = 0$,
 tj. dokud je rx rezervován pro **beta**



Rezervace r1 pro op4:

- když $RF(r1).V = 1$, rezervuj r1 pro op4 zápisem 0 $\rightarrow RF(r1).V$, tj. tvoří se nový obsah r1, ale starý r1 platí!

Rezervace r1 pro op6:

- když bit $RF(r1).V = 0$, čekej až bude tento bit 1 (eliminace WAW).

Čtení operandů r2, r3 a instrukce op4 do FJ: Když

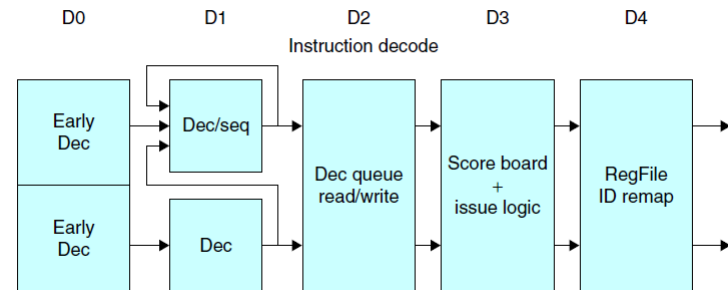
- $SB(op4).V1 = 1$ & $SB(op4).V2 = 1$ a $FJ(op4) = \text{free}$, pošli r2, r3 $\rightarrow RF$, $[RF(r2).value, RF(r3).value, op4] \rightarrow FJ$, (elim. RAW),
- nastav 0 $\rightarrow SB(op4).V1$ a 0 $\rightarrow SB(op4).V2$ (tj. r2 a r3 přečteno).

Zápis výsledku op4 do r1:

- čekej, až op2 a op3 přečtou r1 vytvořený op1, tj. až bude $SB(op2).V1 = 0$ & $SB(op3).V2 = 0$ (eliminace WAR).
- pak zapiš výsledek op4 do $RF(r1).value$ a nastav 1 $\rightarrow RF(r1).V$ a také 1 $\rightarrow SB(op5).V2$ (platný a ještě nepřečtený).

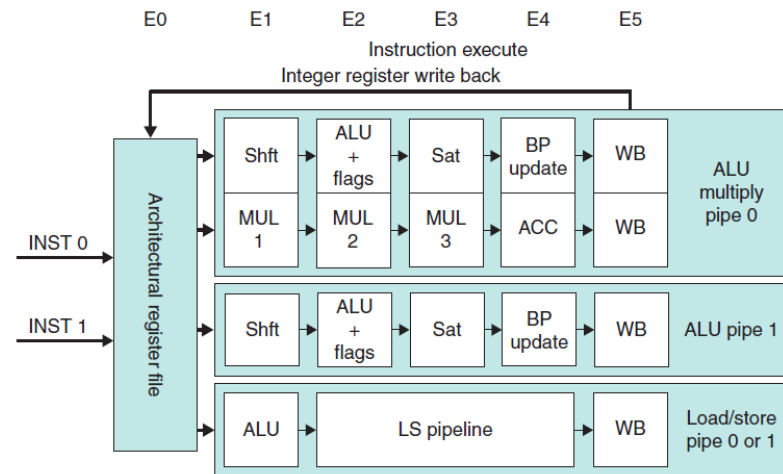
• Dekódovací a vydávací jednotka

- Dekódují se až 2 instrukce za takt
- Pokud žádná z nich není skok, PC se posune o dvě instrukce dále
- ScoreBoard logika se stará o plánování instrukcí



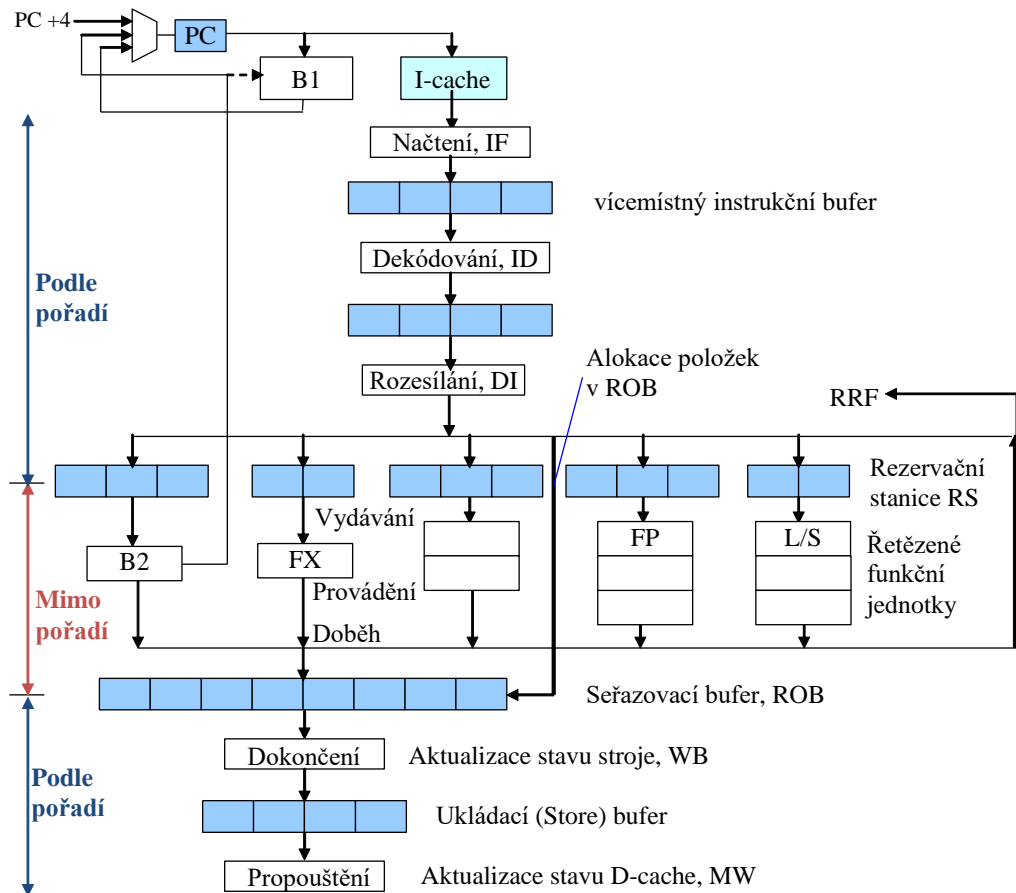
• Exekuční jednotka

- Pokud nedokáže pomoci forwarding budou se závislé instrukce vykonávat sériově
- Forwarding je implementován mezi všemi linkami
- Pokud dojde k závislosti jsou buď obě nebo ta druhá pozastaveny



TOMASULŮV ALGORITMUS REZERVAČNÍ STANICE

PC	programový čítač, instrukční pointer
FX	integer funkční jednotka, integer instrukce
FP	jednotka/instrukce s plovoucí desetinnou čárkou
L/S	jednotka/instrukce přístupu do paměti load/store
B	jednotka zpracování skoků, má 2 části: B1 a B2
RF	soubor registrů (ARF architekturních v instrukčním souboru, RRF přejmenovaných – rename reg. file)
RS	rezervační stanice
IF	načtení skupiny instrukcí, predikce skoků
ID	dekódování instrukcí a přejmenování registrů
DI	rozeslání instrukcí (Dispatch) do RS a ROB
EX	vydání instrukcí do FJ a jejich provedení
WB	aktualizace ARF podle pořadí v ROB
MW	aktualizace stavu D-cache (Memory Write)



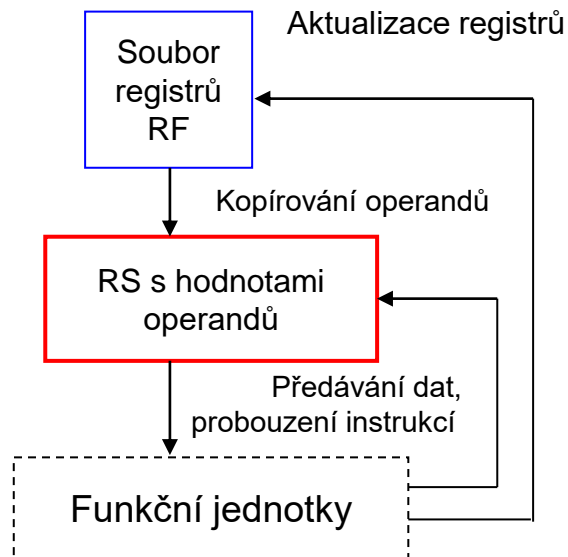
B, Branch unit,
jednotka pro zpracování skoků
B1 – spekulativní, B2 – reálné

RRF – registry pro
přejmenování (cca 256)

RS – zde individuální rezervační
stanice

Předávání výsledků do RS a RF
přes **CDB** (společná datová
sběrnice)

- Intel **P6**: centrální RS s 5 bránami (5 skupinami FJ), **20** položek; obsahuje μ ops s kopiemi operandů.
- Intel **Pentium 4** (NetBurst): 5 skupinových RS, každá s frontou 8–12 položek, obsluhují 5 skupin FJ: (MEM, ALU0, ALU1, FP/move, FP/MMX/SSE)
- Intel **Core i7** (Nehalem): centrální RS, **36** položek
- Intel **Sandy Bridge**: centrální RS, **54** položek, 6 bran
- Intel **Haswell**: centrální RS, **60** položek, 8 bran
- Intel **Cascade Lake** : centrální RS, **97** položek, 8 bran
- **Opteron** (AMD, 64 bitová CPU kompatibilní s IA-32): ALU + AGU: 3 krát skupinová RS s 8 položkami, FP operace: skupinová RS s 36 položkami. Celkem **60** položek v RS
- AMD **Ryzen**: Skupinová RS, **6 x 14** položek, 6 bran

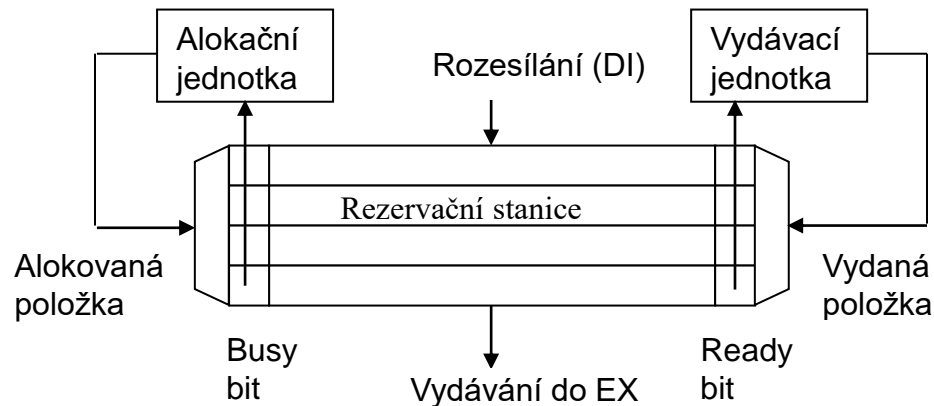


Formát **registrů v RF(dst)**:

RF(dst) **V** | **tag** | **hodnota**

Formát instrukcí v RS(i):

- busy bit // Je položka v tabulce volná
- operace
- src1 (**hodnota**, tag1, valid bit V1)
- src2 (**hodnota**, tag2, valid bit V2)
- dst (**adresa**, tag)
- ready bit // Je instrukce připravena



• Cílový operand

- Dynamicky vytvoříme **příznak** (přejmenujeme)
 - Zapišeme do RS (**tag**)
 - Zapišeme do RF (**tag**), lze přepsat i předchozí příznak
- Zatím neplatný obsah RF(dst) se označí zápisem $V = 0$.
- Jen naposled přejmenovaný dst reg bude mít kopie v RS i RF.
- Dříve přejmenované instance téhož registru existují jen v RS.

• Zdrojové operandy

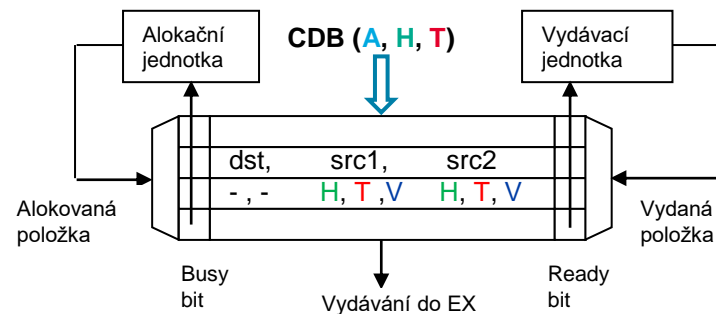
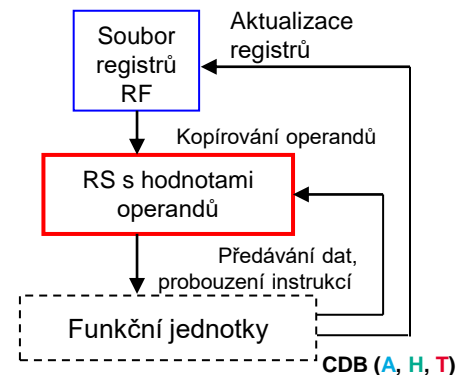
- **platné** hodnoty ($V = 1$) zdrojových operandů z RF včetně tagu se načtou do RS současně s instrukcí; $V1/V2 = 1$.
- zatím ještě **neplatné** operandy ($V = 0$): načte se pouze tag s $V1/V2 = 0$.

Formát položky RS:

- busy bit
- operace
- zdrojový operand1 (hodnota, tag1, valid bit V1)
- zdrojový operand2 (hodnota, tag2, valid bit V2)
- cílový operand (dst reg., **tag** – příznak)
- ready bit

RF(dst): **V** | **tag** | hodnota

- Pokud **src** operandy instrukce v RS **ready** (V1 & V2 = 1) & FJ volná
 - Odešle se instrukce z RS (včetně dat, dst addr, tag) do FJ
 - Jinak čekej až budou ready (eliminace RAW)
- FJ vytvoří výsledek
 - Rozhlásí hodnotu výsledku na CDB (pokud je volný) spolu s adresou dst i tagem dst registru
- RS monitoruje CDB
 - Instrukce čekající v RS monitorují CDB, porovnávají tagy operandů s tagem na CDB
 - V případě shody zachytí hodnotu z CDB do příslušného políčka RS, případně do store buferu (instrukce store) (eliminace WAR, WAW);
- RF monitoruje CDB
 - Při shodě tagů na CDB i v reg. RF(dst) se hodnota zapíše paralelně i do reg. RF(dst) a nastaví se v něm V = 1.
 - Při neshodě tagů (tj. RF(dst) byl již znovu přejmenován) se hodnota v RF(dst) vůbec neobjeví, je zachycena pouze v polích RS, které tak slouží jako tagem přejmenované registry.



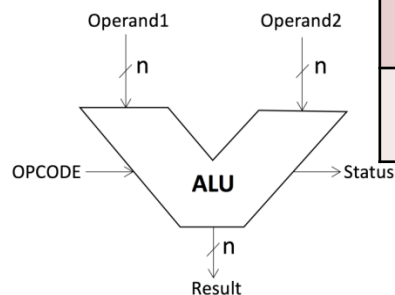
Registry

Name	Valid	Tag	Value
R1			
R2			
R3			
R4			
R5			
R6			

op1 r1, r2, r3
 op2 r2, r1, r4
 op3 r6, r3, r1
op4 r1, r2, r3
 op5 r7, r8, r1
 op6 r1, r5, r4

Rezervační stanice

DST	Tag	V	Tag	Val	V	Tag	Val



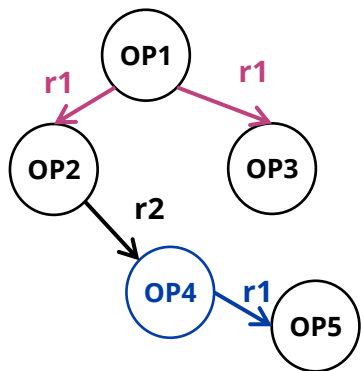
CDB

Rezervační stanice – přejmenování registrů příznakem (tag) odstraní nepravé konflikty

i1 **r1**, r2, r3
i2 r2, **r1**, r4
i3 r6, r3, **r1**
i4 **r1**, **r2**, **r3**
i5 r7, r8, **r1**
i6 **r1**, r5, r4

tag1 → RS(i1), **tag1** → r1
tag2 → RS(i2), tag2 → r2
tag3
tag4 → RS(i4), **tag4** přepíše **tag1** v r1
tag5...
tag6 → RS(i6), **tag6** přepíše **tag4** v r1

i1 **t1**, r2, r3
i2 t2, **t1**, r4
i3 t3, r3, **t1**
i4 **t4**, **t2**, **r3**
i5 t5, r8, **t4**
i6 **t6**, r5, r4

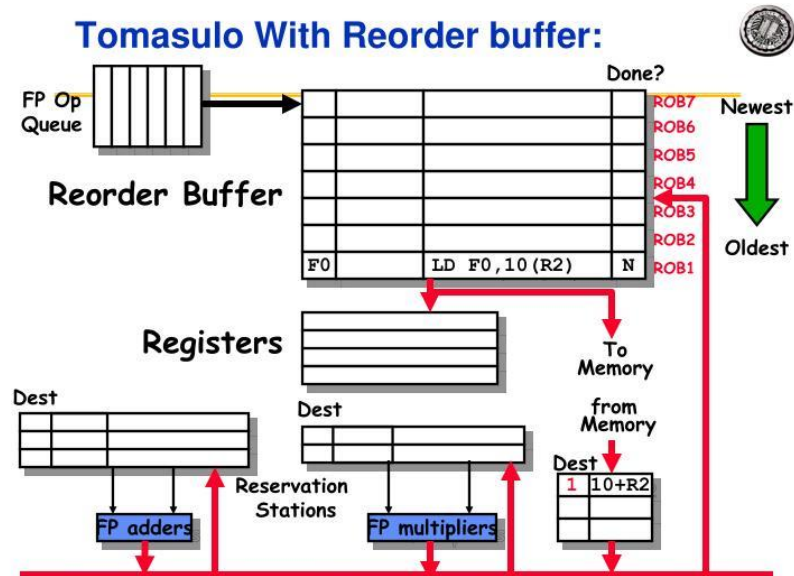


- Do RS(i) se načítá tag cílového reg. a operandy nebo tagy zdrojových reg.
- Instance registru r1 existují v src políčkách v RS označených **tagem1**, **tagem4** a **tagem6**,
- V RF existuje jen instance označená **tagem6** (poslední přejmenování).

- Instrukční závislosti **RAW** se detekují a řeší čekáním v RS
 - u skalárních CPU to bylo čekání/storno instrukce po zjištění problému během fáze ID,
 - podobně u superskalárních INO CPU).
- **Instrukce způsobující konflikty WAR a WAW:**
 - registry viditelné programátorovi (architekturní, ARF) jsou mapovány (**přejmenovány příznakem**) na položky RS (15–60 celkem).
 - Konflikty WAR a WAW je ale lépe řešit explicitním přejmenováním (viz příští přednáška).
- U superskalárních procesorů INO (in-order) přejmenování registrů neprobíhá.

SEŘAZOVACÍ PAMĚŤ REORDER (RETIRE) BUFFER

- ROB je kruhová vyrovnávací paměť **rozpracovaných instrukcí obsahující**
 - informace o stavu instrukce
 - předběžné/spekulativní výsledky.
- Instrukce jsou vloženy do ROB při vydání do RS
- Všechny rozpracované instrukce jsou zde udržovány ve frontě FIFO v programovém pořadí.
- **Propouštění instrukcí pouze z čela ROB**, po propuštění všech předchozích instrukcí.
- **Zápis do arch registrů/paměti při OPUŠTĚNÍ ROB**
- ROB může být použit i pro přejmenování.
- **Současné procesory**: Intel Cascade Lake: 224 položek, AMD ZEN2 : 224 položek Retire Queue.

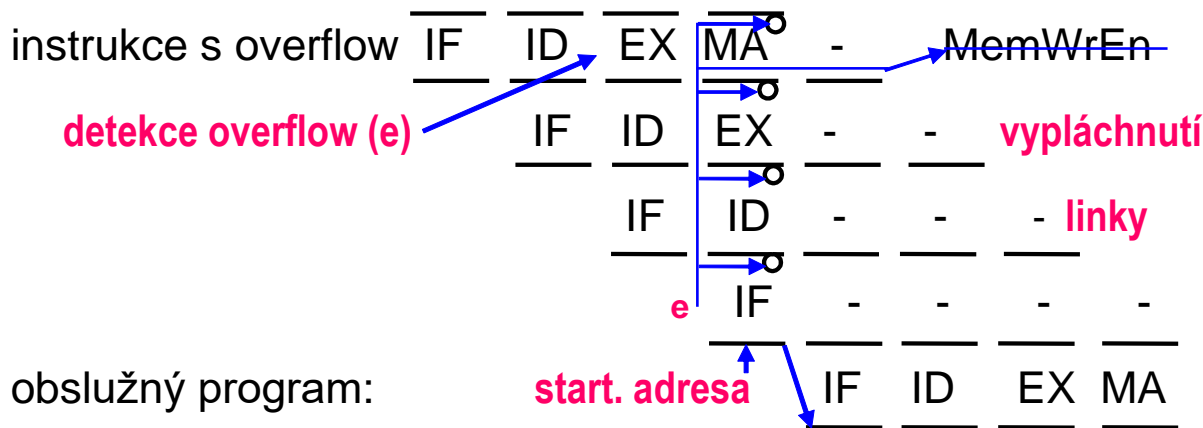
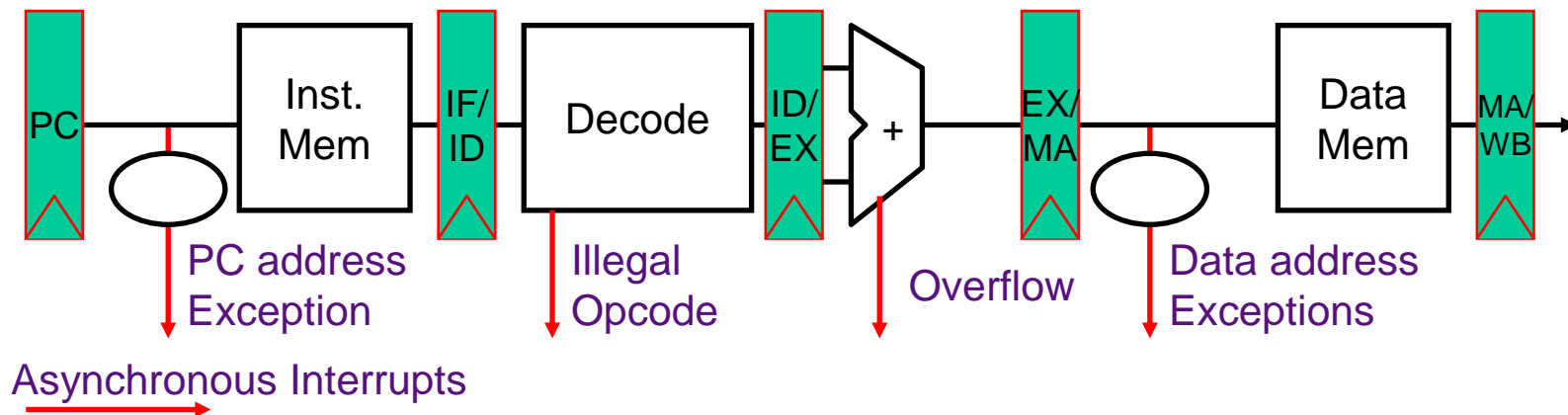


- **Typ instrukce** (aritmetická, L/S, branch,...)
- **Destinace** – adresa arch. registru pro uložení výsledku.
- **Flag** – stav instrukce
 - binární flag ukazuje jestli instrukce doběhla ve FJ (done),
 - vícebitový flag může udávat i jiný stav (dispatched v RS, exec, ready)
- **Hodnota** spočtená instrukcí, zatím ale **nezávazná**
 - tato pole představují sadu registrů RenameRF (jedna možnost přejmenování).
 - závazná hodnota je v arch. registru.
- Při vydávání instrukcí je pro každou instrukci podle jejího pořadí **rezervována první volná položka na konci aktivního úseku ROB**.
- **ROB poskytuje historii více přejmenování téhož registru**: nejstarší verze registru již může být v ARF, předchozí uvnitř fronty v ROB a nejmladší na jejím konci.

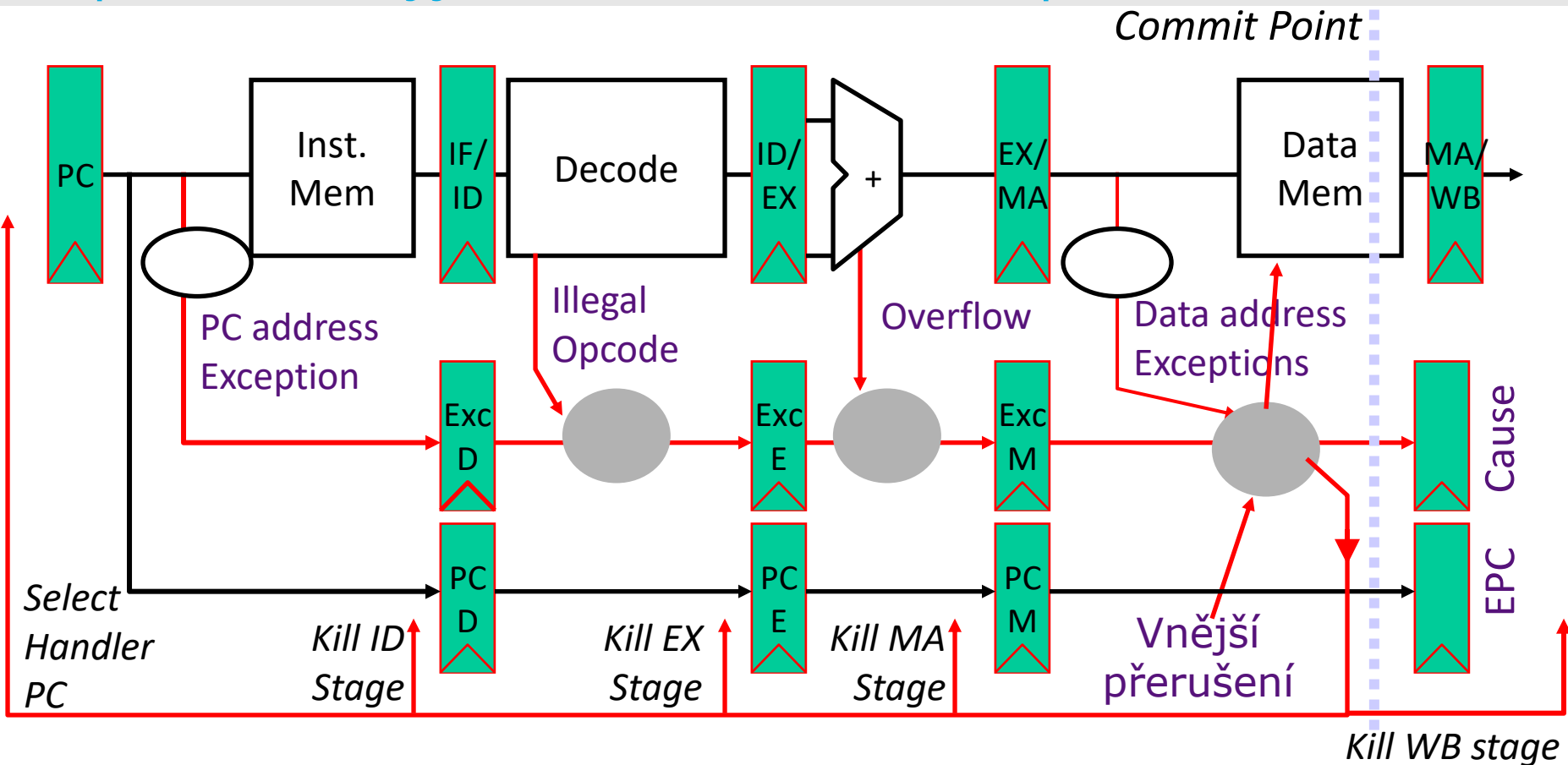
Jde o události, které vyžadují zpracování systémovým programem.

- **Výjimka (exception):** neobvyklá interní událost při zpracování konkrétní instrukce (např. dělení nulou, nedefinovaný op-kód, přetečení, výpadek stránky)
Obecně instrukce nemůže být dokončena a musí být restartována po zpracování výjimky
 - to vyžaduje anulaci účinků jedné nebo více částečně provedených instrukcí (zotavení).
 - Trap: speciální instrukce volání systému – přechod do privilegovaného módu kernelu (SW přerušování).
- **Přerušování:** HW signál přepínající procesor na nový proud instrukcí při výskytu nějaké *externí události* (požadavek na obsluhu zařízení I/O, signál časovače, porucha napájení, HW porucha)

- Když se procesor rozhodne zpracovat přerušení
 - zastaví běžící program u instrukce I_i , a dokončí všechny instrukce až do I_{i-1}
 - uloží PC instrukce I_i do speciálního registru (EPC, Extra PC),
 - zablokuje další přerušení a předá řízení určenému programu obsluhy přerušení běžícímu v kernelu OS.
- **Obslužný program:** přečte registr *Cause* (indikuje příčinu)
 - Buď ukončí program nebo restartuje u instrukce I_i
 - Používá zvláštní instrukci nepřímého skoku RFE (*return-from-exception*), která
 - obnoví uživatelský režim CPU, EPC \rightarrow PC
 - obnoví stav hardwaru a stav řízení
 - povolí přerušení.



NOP do všech
4 registrů podél linky
= kill 4 rozpracované
instrukce

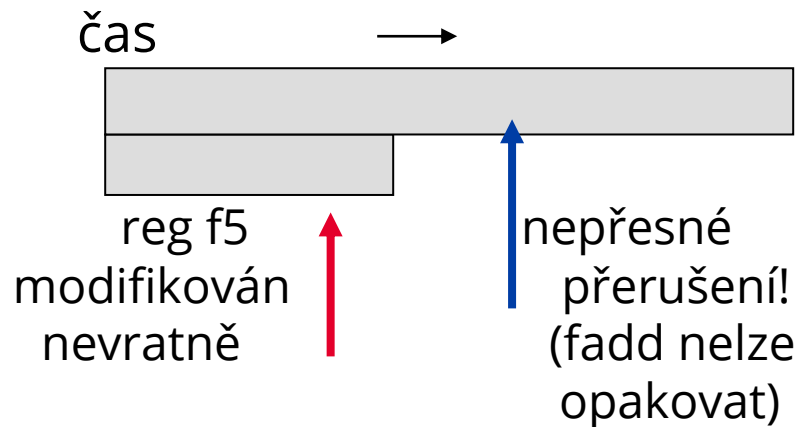


Paralelně prováděné operace mohou skončit v jiném pořadí než v programu:

i-1

i	fdiv	f2, f1, f0
i+1	fadd	f5, f4, f5

i+2



Odstranění: Výsledek operace je třeba uchovat jako prozatímní v dočasné paměti a přepsat do cílového architekturního registru (f5) až po skončení všech předchozích instrukcí – **Propouštění z čela ROB.**

- Ošetření přerušení v ROB

- Instrukce jsou propouštěny v programovém pořadí, přerušení je poznačeno u instrukce v ROB **speciálním bitem**.
- CPU vyřizuje přerušení jen od instrukce na čele fronty v ROB.
- Následná instrukce ještě nezapsala prozatímní výsledek z ROB do ARF a dá se opakovat po ošetření přerušení, tedy jde o **přesné přerušení**.

- Některé procesory dovolují práci ve vybraném režimu:

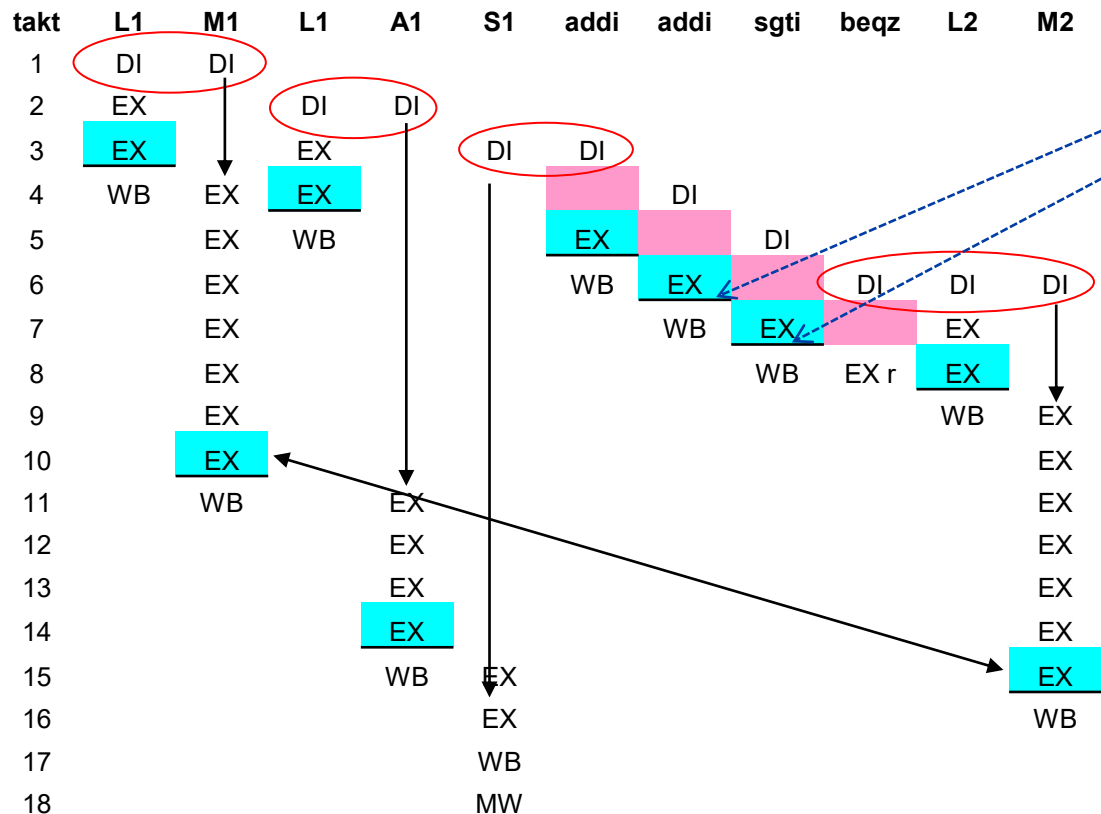
- Pomalejší zpracování s přesnými přerušeními (ladění).
- Rychlé zpracování s nepřesnými přerušeními.

FP násobení 7 taktů, FP sčítání 4 takty, L/S 2 takty.

- Fáze IF, ID nejsou v diagramu níže zobrazeny. Začíná se rozesíláním instrukcí do RS (fáze DI).
- Výsledek poslední fáze EX na CDB podtržen, poslední takt EX zvýrazněn.
- **V diagramu je vidět (ověřte sami):**
 - rozesílání instrukcí do RS po skupinách 1 až 3,
 - čekání instrukcí v RS,
 - provádění instrukcí OOO (dokonce L2 předbíhá S1),
 - výsledek na CDB (na konci EX),
 - čekání instrukcí na volný CDB (strukturní konflikt),
 - překrytí první a druhé iterace (tj. dynamické rozbalování smyčky),
 - rozestup prvních mezivýsledků (i výsledků) 5 taktů.

$y[i] = \text{alpha} * x[i] + y[i]$

```
loop:ld      f2, 0(r1)
      fmul    f4, f2, f0
      ld      f6, 0(r2)
      fadd    f6, f4, f6
      sd      0(r2), f6
      addi    r1, r1, #8
      addi    r2, r2, #8
      sgti    r3, r1, done
      beqz    r3, loop
```



na CDB
lze vložit
jen jeden
výsledek

$$y[i] = \alpha * x[i] + y[i]$$

```

loop: ld      f2, 0(r1)
      fmul    f4, f2, f0
      ld      f6, 0(r2)
      fadd    f6, f4, f6
      sd      0(r2), f6
      addi    r1, r1, #8
      addi    r2, r2, #8
      sgti    r3, r1, done
      beqz    r3, loop
    
```

3 cestný procesor

- FP násobení 7 taktů,
- FP sčítání 4 takty,
- L/S 2 takty.

OPTIMALIZACE TOKU DAT PŘES REGISTRY

- **Přejmenování registrů** v jednoduché formě již v rezervačních stanicích, ale
 - počet položek **RS** omezen
 - počet přejmenování registrů za 1 takt omezen.
- Při zpracování 4 a více instrukcí/takt může být potřeba přejmenovat několik registrů v jednom taktu, případně jeden a tentýž registr vícekrát v jednom taktu:

```
mul r3, r0, r2      ... 4 instrukce postupující současně
add r2, r1, r4
sw  r2, 0(r5)        r2, r2 ; r2, r2 ... WAR
sub r2, r3, r1        r2, r2 ... WAW
```

- Stačí přejmenovat 2x cílový registr r2 v 1 taktu:
r2 → r6, **r2** → r7.

- **Přejmenování** pomocí registrů **v ROB**:
 - **1 i více výsledků** dokončených instrukcí z čela ROB se přesunuje v každém taktu najednou do ARF (souboru architekturních, tj. viditelných registrů).
 - Intel P6, Pentium M: 40 položek, Nehalem 128 položek
- **Přejmenování v HW** pomocí sady (128 a více) registrů pro přejmenování (**Rename Register File RRF**) s více branami
 - RRF obsahuje potvrzené i spekulativní výsledky.
 - Soubor registrů pro přejmenování RRF může být:
 - oddělen od architekturních (ARF)
 - monolitický, integrován s ARF (Intel Pentium 4, Sandy Bridge)

- Pro novou instrukci se **vytvoří položka na konci ROB** {**typ instrukce**, **dst reg.**, **flag**, **hodnota vypočtená instrukcí**}.
- **Do RAT (Register Alias Table)** se zapíše (podle pořadí v programu poslední) přejmenování
arch.reg. → tag = adresa ROB.
- Tabulka přejmenovaných registrů RAT udává pro každý arch. registr adresu ROB nebo adresu registru v ARF.
- **Ready operand** se hledá nejdříve v ARF, pak v ROB(tag).
 - ARF obsahuje již potvrzené výsledky.
 - ROB udržuje nepotvrzené a spekulativní výsledky.
- **Po provedení instrukce** je
 - **výsledek a flag = ready** zapsán do ROB
 - **Přepis** (WB) do ARF se provede až bude instrukce na čele ROB!

Před : add r3, r3, 4
 add r4, r7, r3
 add r3, r2, r7

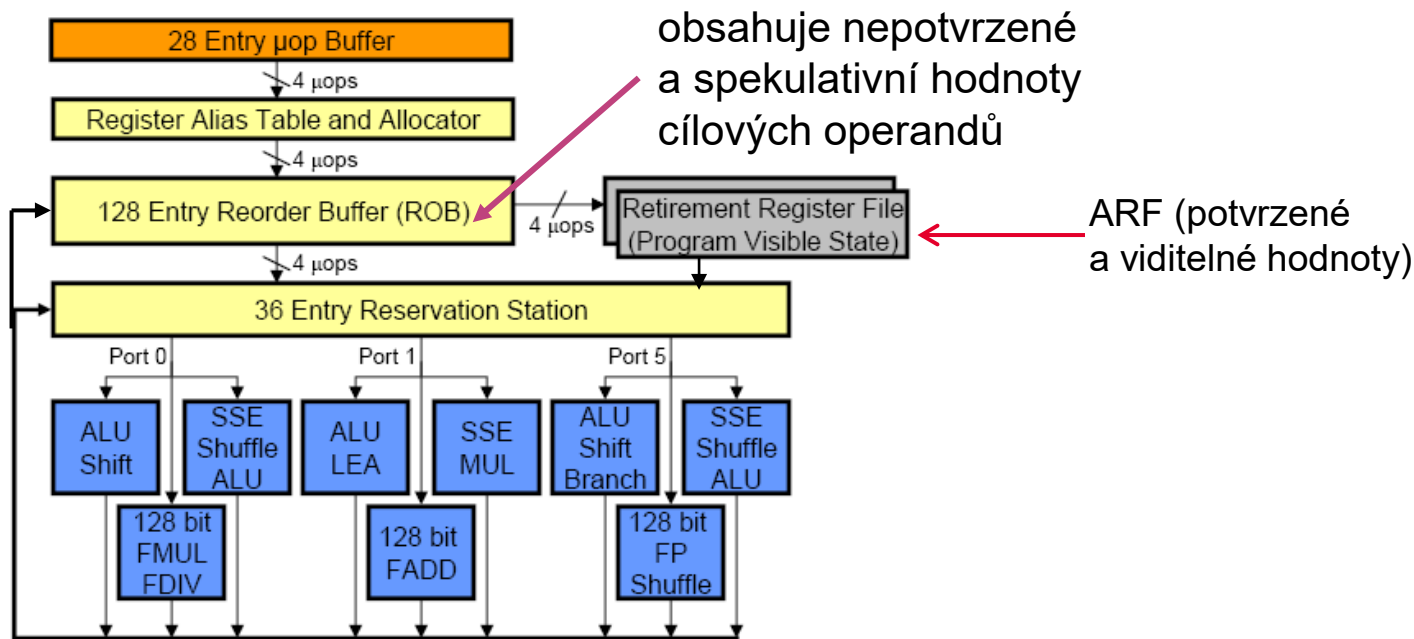
Po: add rob6, r3, 4
 add rob7, r7, rob6
 add rob8, r2, r7

RAT	
...	poslední přejmenování
r2	r2
r3	rob8
r4	rob7
...	tag

ROB	
...	
→6	add r3 ready 204
7	add r4 exec
8	add r3 exec
→9	...

Je-li jeden dst reg. přejmenován 2x a 2 výsledky by se měly zapsat v jednom taktu do téhož registru ARF, zapíše se tam jen **poslední výsledek** v program. pořadí.

Intel Nehalem: RAT může přejmenovat až 4 μ ops v každém taktu a každé přidělí cílový registr v ROB. Přepis až 4 výsledků hotových instrukcí z čela ROB do ARF v programovém pořadí.



- **Každý fyzický registr** z Physical Register File (PRF) může být použit jako **architekturní** (viditelný) nebo **přejmenovaný**.
- **Při dekódování se**
 - načte **tag** z FIFO volných tagů
 - přejmenuje se dst registr $ARF(x) \rightarrow PRF(tag)$
 - do **mapovací tabulky RAT** (Register Alias Table) se uloží nové mapování $x \rightarrow tag$, **unready**. (tag = adresa registru v PRF)
 - Register je **unready**, když registr čeká na zápis výsledku.
 - Ke stejnému ARF registru může být přiřazeno několik tagů, $PRF(tag)$ je ale určen jednoznačně.
 - Do RAT se dá jen **poslední přejmenování**.
- **Po provedení instrukce**
 - cílový operand s tagem rozhlášen do PRF, RS a do RAT.
 - V RS se operand načte do políček operandů se shodným tagem.
 - V RAT se u tagu zapíše **ready**.

free list: r37, r38, r39

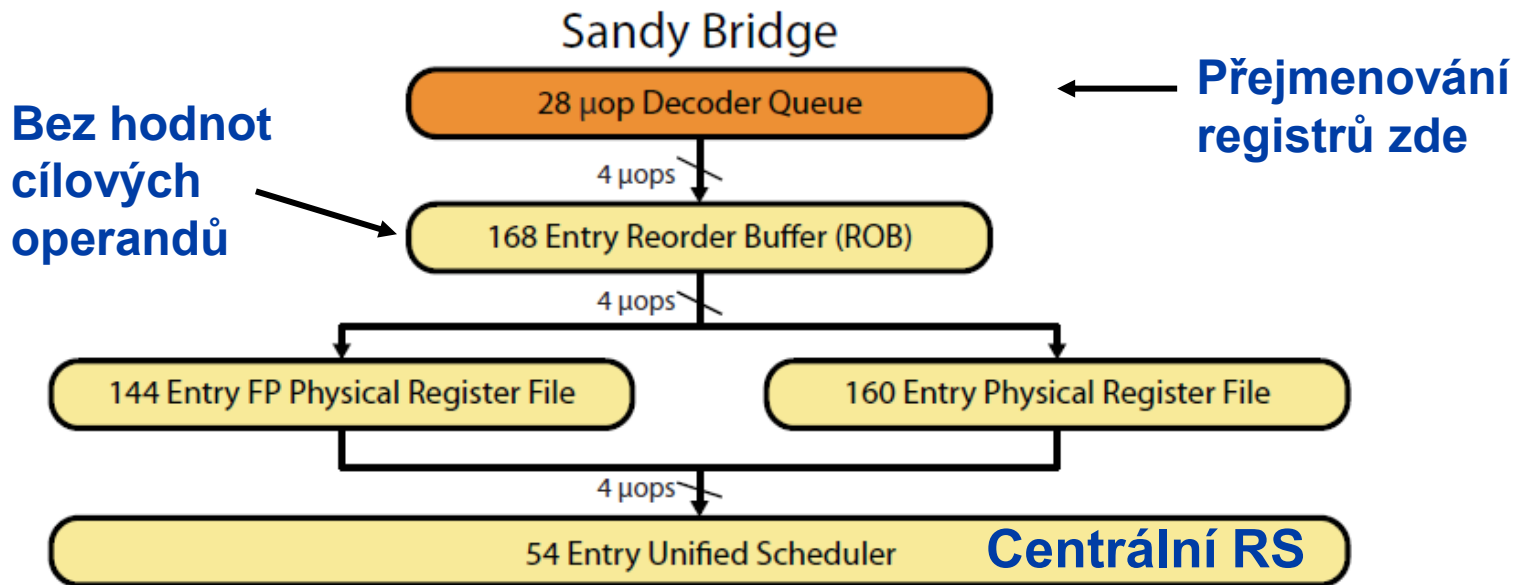
Před : add r3, r3, 4
 add r4, r7, r3
 add r3, r2, r7

Po: add r37, r3, 4
 add r38, r7, r37
 add r39, r2, r7

← přejmenován podruhé →

Při **propouštění instrukce** se

- mapování dst registru uloží do další tzv. **propouštěcí RAT**, která definuje **okamžitý soubor ARF**.
- Do propouštěcí RAT se uloží jen poslední mapování v programovém pořadí (**r3 → r39**).
- Ostatní volné tagy → FIFO (**r37 → free list**).
- **U každého skoku** je pořízen snímek propouštěcí RAT kvůli zotavení ze špatné predikce. Při špatné predikci nebo přepnutí kontextu je třeba se vrátit k poslednímu platnému mapování v propouštěcí RAT.



- ROB **neobsahuje hodnoty** operandů. Odpadá tak kopírování výsledků z ROB do ARF při dokončování (propouštění) instrukcí. 😊
- **Write-back** aktualizuje jen stav (flag) instrukcí v ROB.

Pokračování příště