1. Architektura superskalárních procesorů a algoritmy zpracování instrukcí mimo pořadí, predikce skoků.

Amdahlovo pravidlo

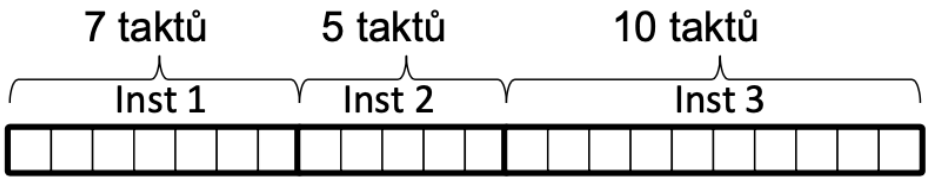
$$\text{Speedup}(N) = \frac{1}{(1-P)+\frac{P}{N}}$$

Serial part of job = 1 (100%) - Parallel part

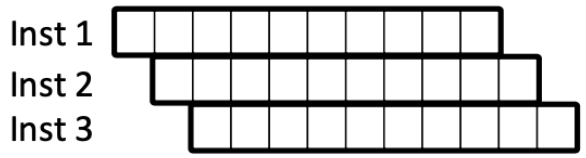
Parallel part is divided up by N workers

Architektura	Vydávání instrukcí	Provádění instrukcí
Subskalární von Neumann	Sekvenční 0 až 1	Sekvenční Několik taktů
Skalární (řetězené)	Sekvenční 0 až 1	Paralelní CPI >1
Superskalární a VLIW	Paralelní 0 až *m*	Paralelní IPC < *m*
Vícejádrové s časovým MT	1 jádro	Z více vláken
Vícejádrové s časovým a prostorovým MT	Více jader	Z více vláken na každém jádru

- **Neřetězená (sub-skalární, μ-programovaná) CPU**



- **Řetězená (skalární) CPU**



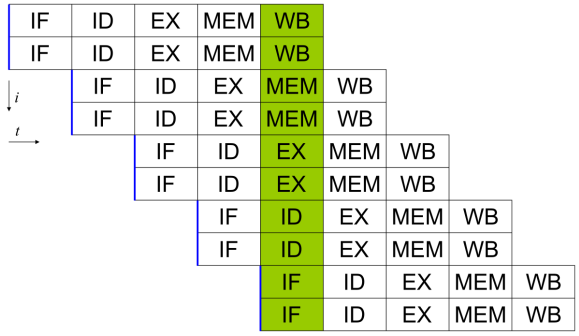
Zpracování instrukce se zde skládá z několika kroků:

- **Ifetch** - načtení instrukce
- **IDecode** - načtení registrových operandů, dekódování instrukce
- **Execute** - provedení instrukce nebo výpočet adresy
- **Memory access** - přístup do paměti cache (čtení/zápis)
- **Write Back** - zápis výsledku do cílového registru

Paměťové konflikty vznikající při řetězení instrukcí:

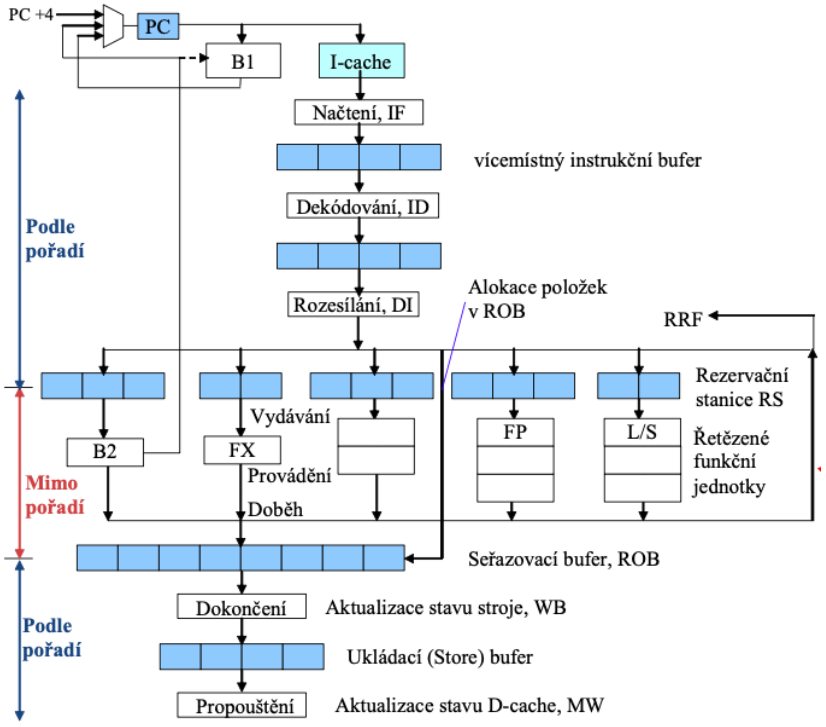
- datová závislost
  - RAW (ReadAfterWrite) - instrukce načítá nebo počítá s výsledkem generovaným dřívější instrukcí.
  - Nepravé konflikty - WAR, WAW, problém pouze s přejmenováním registrů.
- řídicí závislost
  - podmíněné a nepodmíněné skoky
- strukturní závislost
  - dvě instrukce potřebují stejný výsledek

Superskalární procesor



Rozdělení komponent superskalárního procesoru:

- **Front-end** (IF a ID)
  - Načítá a dekóduje několik instrukcí najednou, počet se mění dynamicky.
- **Back-end** (EX, MA, WB)
  - Provádí a ukládá výsledky několika instrukcí souběžně.
  - Některé stupně jsou rozděleny na podstupně.



Rozdělení superskalárních procesorů:

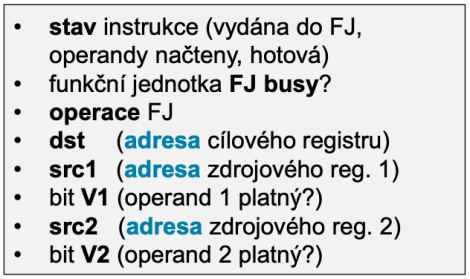
- **In-order**
  - Instrukce opouštějí frontend podle pořadí v programu.
  - Jednoduchý HW.
- **Out-of-order**
  - Instrukce zpracovány mimo pořadí
  - Nepravé konflikty vyřešeny přejmenováním v HW.
  - RAW vyřešeny čekáním rozpracovaných instrukcí.
  - Zápis výsledků v původním pořadí zajištěn pomocí seřazovací paměti (ROB)

Rysy superskalárních procesorů

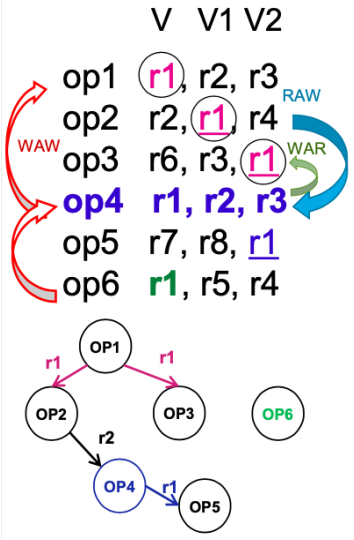
- **Paralelní řetězené linky** - Časový i prostorový paralelismus (paralelní načítání, dekódování, vydávání instrukcí do FJ, jejich paralelní provádění a dokončování).
- **Přejmenování registrů v HW** - Odstraní konflikty WAR a WAW
- **Dynamické plánování instrukcí out-of-order** - Po dekódování čekají 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** - Stupeň WB pomocí ní zajišťuje ukládání výsledků v pořadí určeném zdrojovým kódem.
- **Spekulativní zpracování instrukcí** - Spekulace, že skok dopadne podle predikce nebo že dopředu načtená data se již nezmění.

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) 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.



Příklad 1: Scoreboarding



Rezervace r1 pro op4:

- když RF(r1).V = 1, rezervuj r1 pro op4 zápisem 0→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) = free, pošli r2,r3→RF, [RF(r2).value, RF(r3).value, op4] → FJ, (elim. RAW),
- nastav 0 → SB(op4).V1 a 0 → 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→ RF(r1).V a také 1→ SB(op5).V2 (platný a ještě nepřečtený)

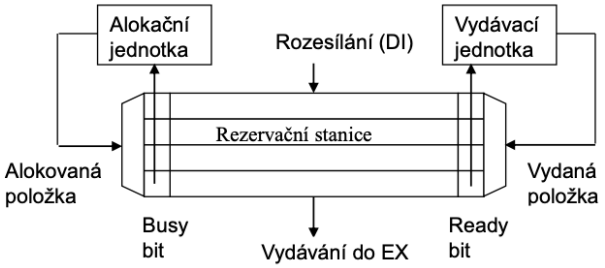
2. **Reservační stanice** (Tomasulův algoritmus) Konflikty WAW a WAR se řeší přejmenováním. Rezervační stanice RS (bufery) umožňují odložit čekající instrukce a pracovat dopředu na dalších - tím řeší RAW. Rezervační stanice centrální (instruction window) nebo individuální u FJ či skupinové pro skupiny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

Formát instrukcí v RS(i):

- busy bit
- operace
- src1 (hodnota, tag1, valid bit V1)
- src2 (hodnota, tag2, valid bit V2)
- dst (adresa cíl. reg., tag)
- ready bit

Formát **registrů v RF**(dst):

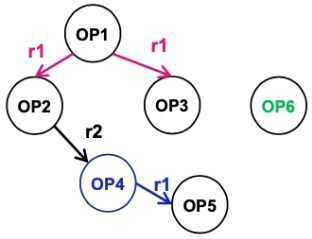
RF(dst) | V | tag | hodnota |



Příklad 2: Tomasulo algoritmus

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

i1 r1, r2, r3	tag1 → RS(i1), tag1 → r1	i1 t1, r2, r3
i2 r2, r1, r4	tag2 → RS(i2), tag2 → r2	i2 t2, t1, r4
i3 r6, r3, r1	tag3 ….	i3 t3, r3, t1
i4 r1, r2, r3	tag4 → RS(i4), tag4 přepíše tag1 v r1	**i4 t4, t2, r3**
i5 r7, r8, r1	tag5…	i5 t5, r8, t4
i6 r1, r5, r4	tag6→ RS(i6), tag6 přepíše tag4 v r1	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).

Predikce skoků

Predikce podmínky

- Statická - lze určit za kompilace programu.
- Dynamická - závisí na aktuálním běhu programu.

Dynamická predikce skoků

- **Predikce podmínky skoku** (jen u podmíněných skoků) - 1-bitový prediktor, AI prediktory.
- **Predikce cílové adresy** - Branch target buffer (BTB), Return Stack Buffer (RSB).