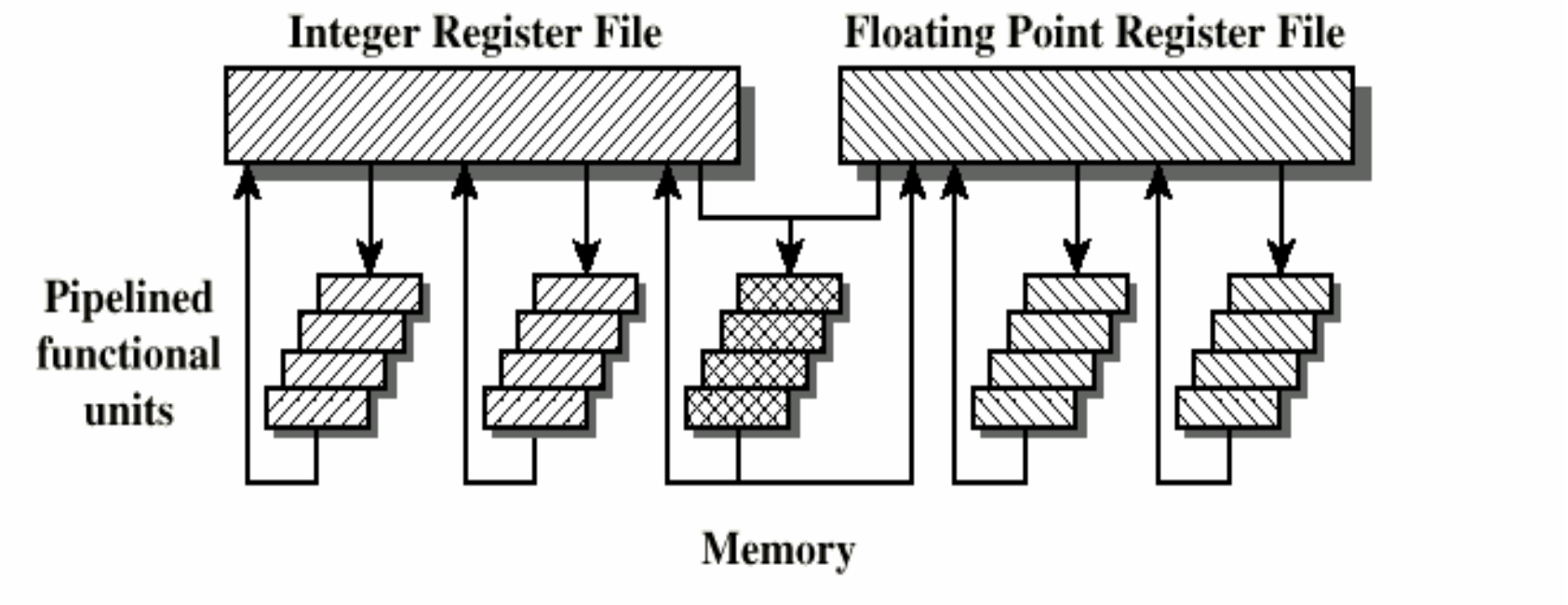
Architektura Pentia – úvod

Co je to superskalární architektura?

- Minimálně dvě fronty instrukcí.
- Provádění instrukcí je možné iniciovat současně, instrukce se pak provádějí paralelně.
- Realizovatelné jak v CISC tak i RISC architekturách.

Superskalární architektura - obecně



Několik funkčních jednotek - každá z nich realizovaná jako fronta – podpora paralelního provádění instrukcí

Dva typy instrukcí (pro operace s čísly typu integer a operace s čísly v pohyblivé řádové čárce).

Superzřetězené architektury (superpipelined)

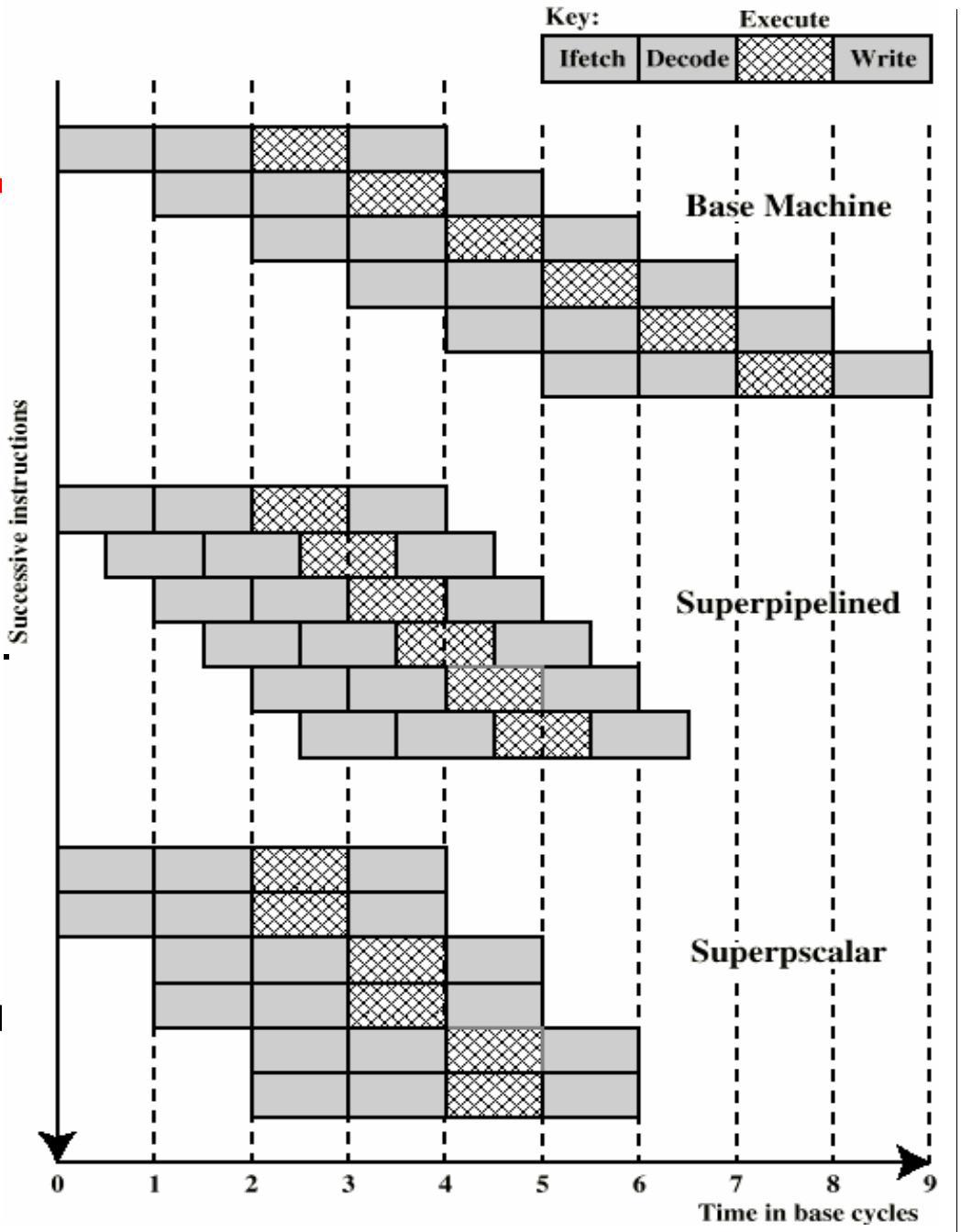
- Existují situace, kdy je možné instrukci zpracovat v kratším čase než polovina cyklu nebo zahájit zpracování instrukce uprostřed vnějšího cyklu.
- Jeden vnější cyklus – dva interní cykly.
- Ve vztahu k vnějšímu cyklu – provádějí se dvě úlohy v jednom vnějším cyklu.

Superzřetězené architektury

Klasická zřetězená architektura – jedna fronta instrukcí

Superzřetězené architektury – vnitřní kmitočet 2x vyšší než vnější.

Superskalární architektura – v obou frontách se provádění zahajuje ve stejném okamžiku.



Omezení superskalárních architektur

- Pojem „paralelismus na úrovni provádění instrukcí“ (instruction level parallelism) - ILP
- Snaha o maximální využití těchto technik:

  na úrovni překladu

  na úrovni hardware
- Omezení využití:
  - datová závislost,
  - procedurální závislost,
  - konflikt zdrojů,
  - výstupní závislost,
  - antidatová závislost.

Datová závislost

- ADD r1, r2 (r1 := r1+r2;)
- MOVE r3,r1 (r3 := r1;)
- Druhou instrukci je **možné** paralelně vložit do fronty instrukcí a dekódovat, není možné ji začít provádět.
- **Není možné** zahájit provádění další instrukce.
- Závislost, kdy následná instrukce nemůže být provedena dříve než je ukončena instrukce předcházející.

Procedurální závislost

- Není možné provádět instrukce, které jsou za instrukcí skoku paralelně s instrukcemi před instrukcí skoku.
- Obdobně: instrukce je dlouhá, je nutno ji nejdříve dekódovat (zjistí se délka instrukce), pak se teprve přečte zbytek instrukce – není možné zahájit čtení další instrukce.
- Procedurální závislost postihuje situace, kdy se pracuje s pamětí a je nutné rozhodnout, kdy nude možné zahájit další činnost s pamětí.

Konflikt zdrojů

- Dvě nebo více instrukcí potřebuje přístup ke stejnému zdroji.
  - např. dvě aritmetické instrukce
- Řešení: zdvojení zdrojů
  - např. dvě aritmetické jednotky

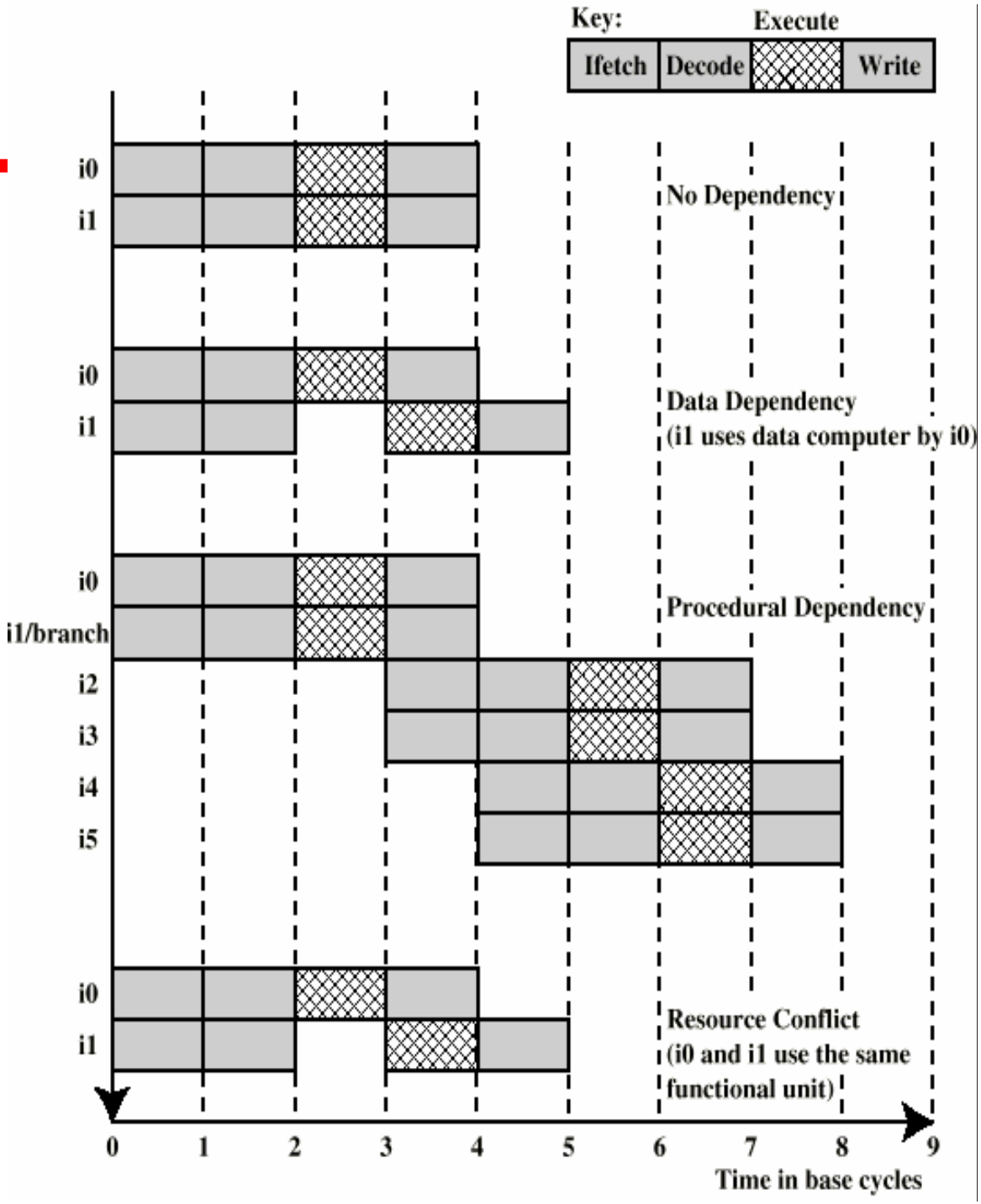
Vliv závislostí

Bez závislosti

Datová závislost – (instrukce i1 potřebuje data vypočtená instrukcí i0)

Procedurální závislost

Konflikt zdrojů



Paralelismus na úrovni instrukcí a počítače

- Na úrovni instrukcí (ILP)
  - Nezávislost instrukcí ve frontě.
  - Provádění instrukcí se může překrývat.
  - Souvislost s datovou a procedurální závislostí.
- Na úrovni počítače (Machine Level Parallelism – MLP)
  - Počítač musí být vybaven tak, aby bylo možné využít ILP.
  - Závislost na počtu paralelně pracujících front.

Souvislost s prováděním instrukcí

- **Paralelní provádění instrukcí** – kvalitní procesor musí být vybaven tak, aby se při plánování činností nemusel držet striktně pořadí instrukcí, ale mohl provádět změny v posloupnosti provádění instrukcí – reordering.
- **Možnosti přeuspořádání**:

  - Pořadí, v němž jsou instrukce řazeny do fronty.

  - Pořadí provádění instrukcí.

  - Pořadí, podle něhož instrukce mění obsahy registrů a pamětí.

Vstup instrukcí podle pořadí, dokončení instrukcí podle pořadí

- Instrukce jsou čteny podle pořadí v programu.
- Nepříliš efektivní.
- Do vstupní fronty je čtena více jak jedna instrukce.
- V případě potřeby se provádění instrukcí pozastaví.

Vstup instrukcí podle pořadí, dokončení instrukcí podle pořadí (diagram)

Decode		Execute			Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
I3	I4	I1					3
	I4			I3	I1	I2	4
I5	I6			I4			5
	I6		I5		I3	I4	6
			I6				7
					I5	I6	8

2 vstupní fronty, 3 funkční jednotky, 2 výstupní fronty.
Na provedení instrukce I1 jsou potřeba 2 cykly.
I3 a I4 potřebují při provádění stejné funkční jednotky.
I5 čeká na výsledek I4.
I5 a I6 potřebují stejnou funkční jednotku.
Doba trvání – 8 cyklů.

Vstup instrukcí podle pořadí, změna pořadí výstupu

- Výstupní závislost
  - R3:= R3 + R5; (I1)
  - R4:= R3 + 1; (I2)
  - R3:= R5 + 1; (I3)
  - Výsledek I2 závisí na výsledku I1 – **datová závislost**.
  - Pokud se provedení I3 ukončí před ukončením I1, bude výsledek I1 nesprávný – výstupní závislost → takovou změnu pořadí nemůžeme provést.
  - Tyto aspekty je třeba reflektovat při reorganizaci posloupnosti provádění instrukcí.

Vstup instrukcí podle pořadí, změna pořadí výstupu (diagram)

Decode		Execute			Write		Cycle
							1
		I1	I2				2
		I1		I3	I2		3
				I4	I1	I3	4
			I5		I4		5
			I6		I5		6
					I6		7
I1	I2						
I3	I4						
	I4						
I5	I6						
	I6						

Doba provedení – 7 cyklů.

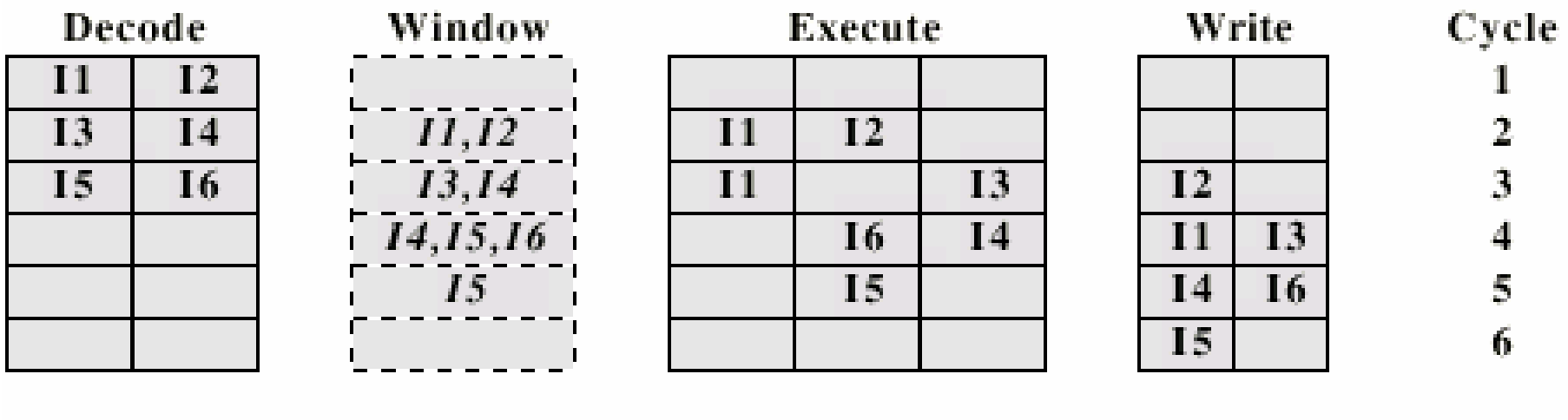
Ve všech doposud diskutovaných případech se organizace pořadí instrukcí mění ve vstupní frontě.

Následující případ – jiná koncepce.

Změna pořadí na vstupu, změna pořadí na výstupu

- Na výstupu jsou instrukce párovány jinak než ve vstupní frontě.
- Do vstupní fronty se přivádějí instrukce a rozdekódují se.
- Jakmile se uvolní funkční jednotka, instrukce se provede.
- Instrukce byly rozdekódovány, procesor se může „dívat dopředu".
- Pokud při rozdekódování instrukce vznikne konflikt, tak se řeší a nejsou dekódovány další instrukce, dokud se tento konflikt nevyřeší – procesor se přestane zabývat instrukcemi, které následují za touto konfliktní instrukcí (chybí „pohled dopředu" – lookahead).
- Řešení – instruction window.

Změna pořadí na vstupu, změna pořadí na výstupu (diagram)



Do fronty se plynule čte (na rozdíl od předcházejících architektur) – konflikty se řeší až ve „window“.

Instrukční okno není další frontou. O instrukci, která je v okně platí, že procesor má dostatek informace o tom, jaké mohou při realizaci instrukce vzniknout konflikty, takže může být rozhodnuto o o tom, že provádění instrukce bude zahájeno.

Antizávislost (antidatová závislost)

- Předcházející situace datové závislosti: při párování (současném zpracování dvou instrukcí ve dvou frontách) nesmí být instrukce n+1 provedena dříve než instrukce n, pokud obě instrukce pracují se stejným operandem (instrukce n přiřazoval hodnotu parametru, s nímž se pracuje jako s operandem v instrukci n+1).
- Write-write dependency
  - R3:=R3 + R5; (I1)
  - R4:=R3 + 1; (I2)
  - R3:=R5 + 1; (I3)
  - R7:=R3 + R4; (I4)
  - Pokud by se I3 a I2 prováděli paralelně, nesmí se I3 dokončit dříve než I2, protože I2 potřebuje hodnotu uloženou v R3 a I3 mění hodnotu v R3 (nesmí se stát, aby I3 změnila hodnotu v R3 předtím, než I2 „vezme“ hodnotu z R3).

Přejmenování registrů (Register Renaming)

- Pokud se instrukce provádějí beze změny pořadí, pak hodnoty v registrech jsou „identifikovatelné“.
- Pokud se mění pořadí instrukcí, pak informace v registrech nebudou reflektovat původní pořadí instrukcí.
- Může se stát, že různé instrukce „bojují“ o tytéž registry – problém – tento problém je ještě výraznější, pokud se pro přidělování používají optimalizační postupy - registry jsou alokovány dynamicky.
- Problém „konflikt zdrojů“ je řešitelný duplikací zdrojů – přejmenování.

Příklad techniky přejmenování registrů

- R3b:=R3a + R5a (I1)
- R4b:=R3b + 1 (I2)
- R3c:=R5a + 1 (I3)
- R7b:=R3c + R4b (I4)
- Označení bez indexů – registr podle instrukce.
- Označení s indexem – hardwarově přidělený registr.

Podpora MLP (Machine Level Parallelism)

- Duplikace zdrojů.
- Změna pořadí provádění instrukcí.
- Přejmenování.
- Propojení technik duplikace zdrojů s technikou přejmenování registrů.
- Instrukční okno dostatečné velikosti.