

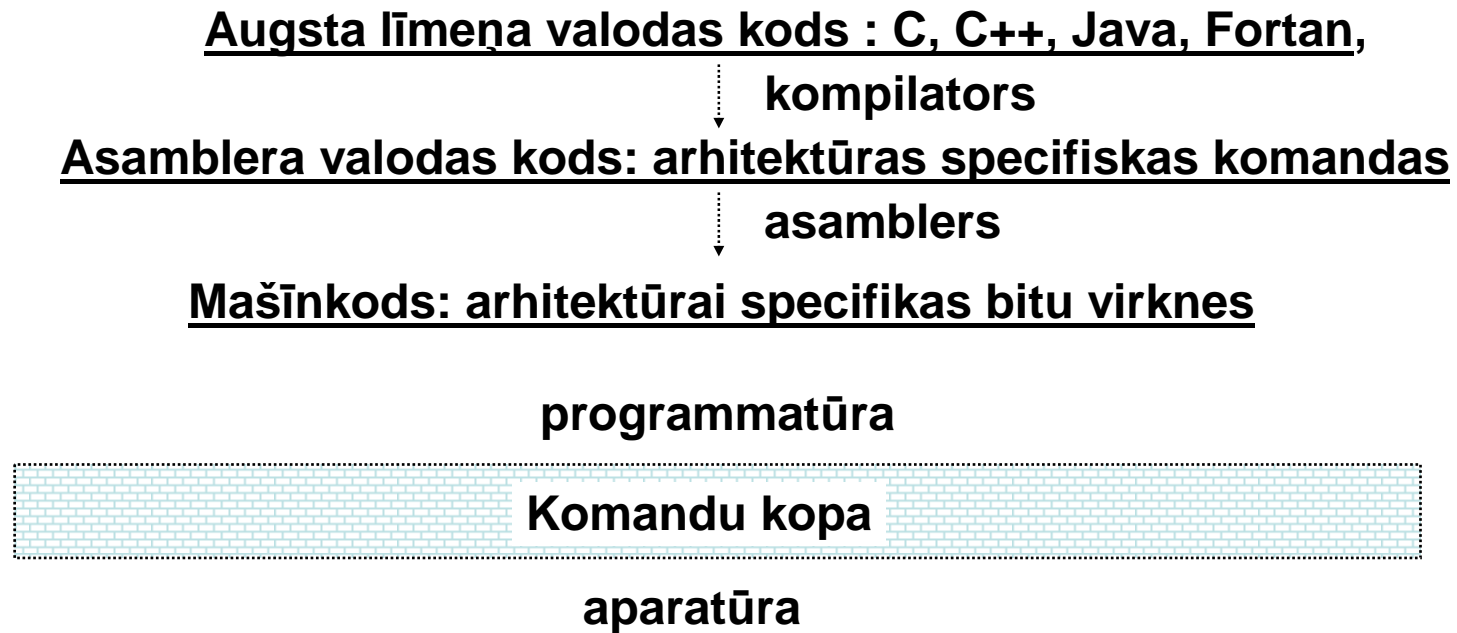
Komandu posmu paralēla izpilde

Komandu kopas ir...? Tās var iedalīt pēc...?
Starpeksāmens.

Komandu struktūra

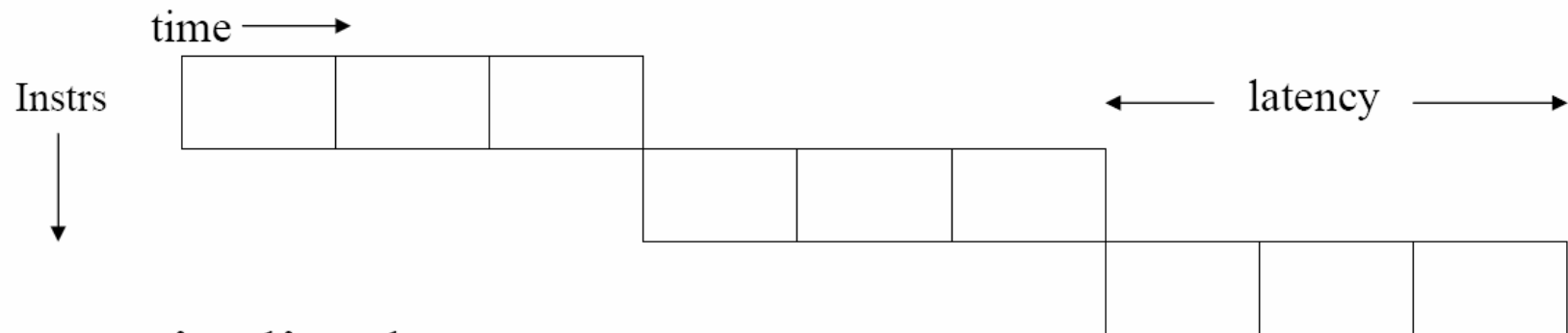
- Mainīga
 - Komandu garums mainās atkarībā no operācijas koda un adrešu specifikātoriem
 - Piemēram VAX komandas var mainīties garuma ziņā no 1 līdz 53 baitiem bet x86 no 1 līdz 17 baitiem.
 - Veidojas kompakts kods bet to ir grūti dekodēt un konveijerizēt
- Fiksēta
 - Vienāda garuma komandas
 - Piemēram MIPS, Power PC, Sparc
 - Ne tik kompakts kods bet to vieglāk dekodēt un konveijerizēt
- Jaukta
 - Var būt vairāki garuma formāti ko nosaka operācijas kods
 - piemēram IBM 360/370
 - Kompromiss

Orientieri

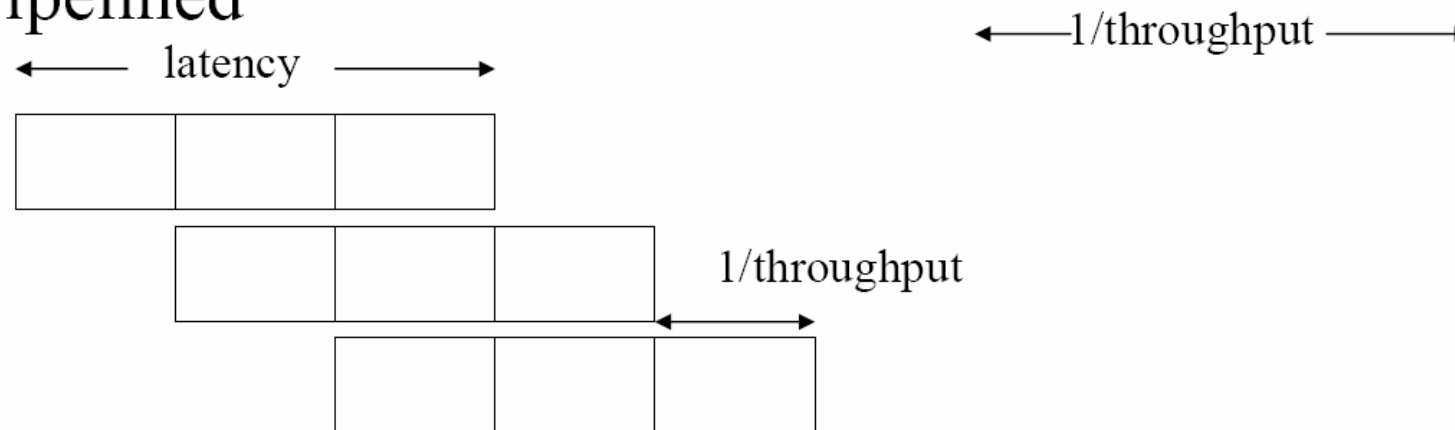


Konveijerizācija aparātūras līmenī

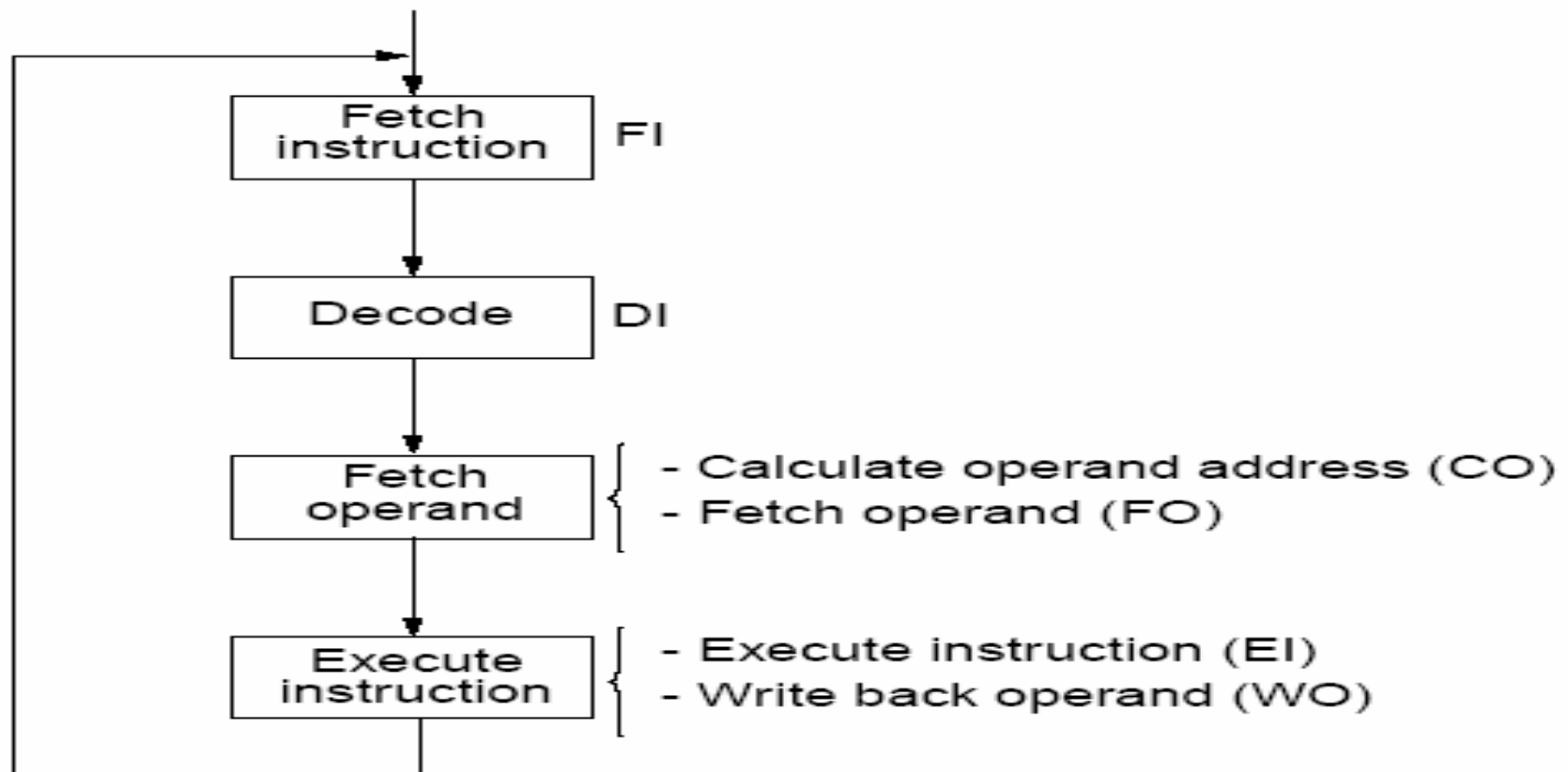
- Unpipelined



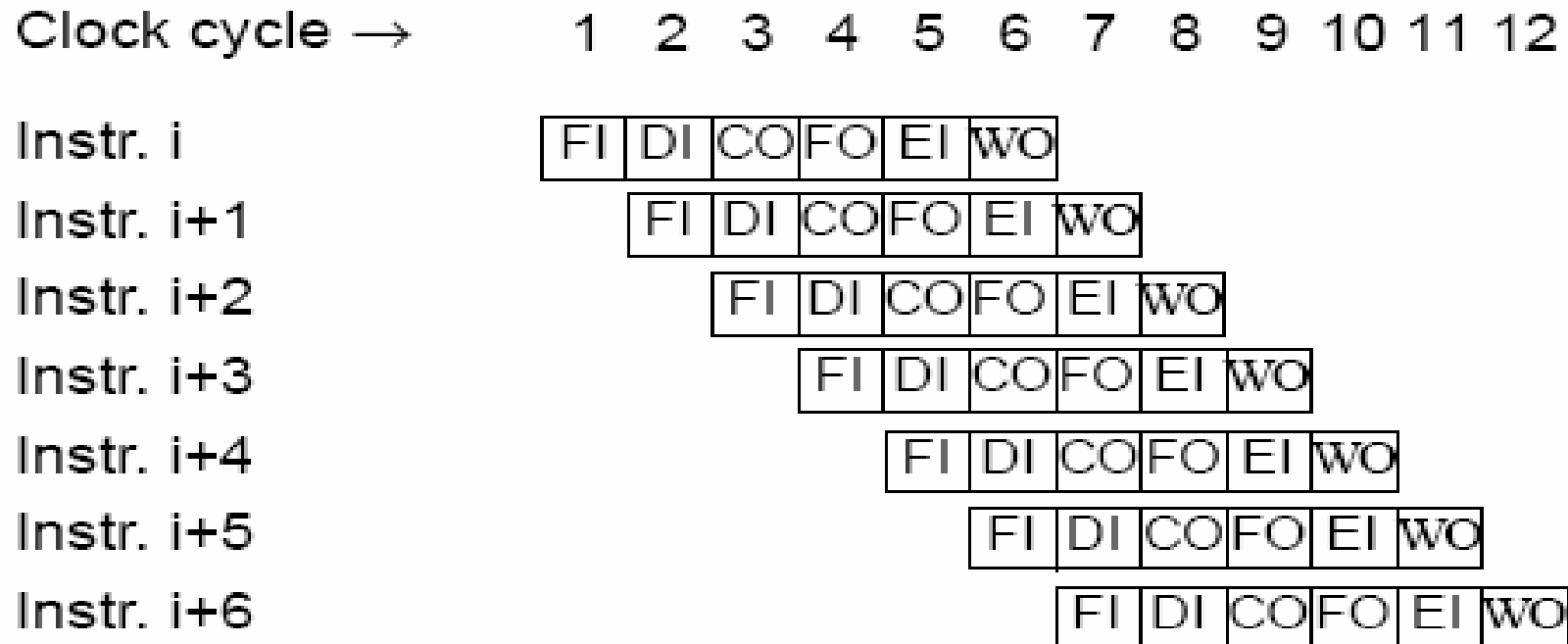
- Pipelined



Komandas dzīves cikls



Uzlabojumi no konveijerizācijas



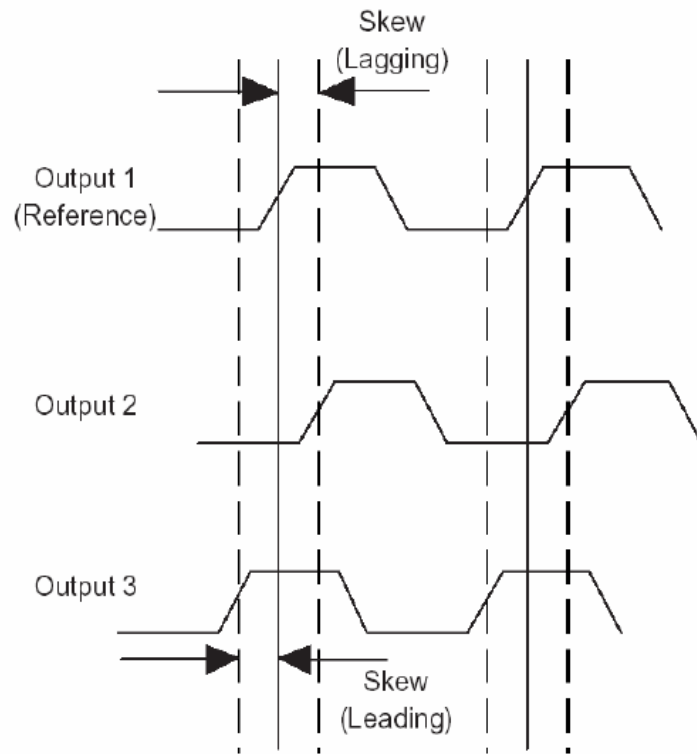
Execution time for the 7 instructions, with pipelining:

$$(T_{ex}/6) * 12 = 2 * T_{ex}$$

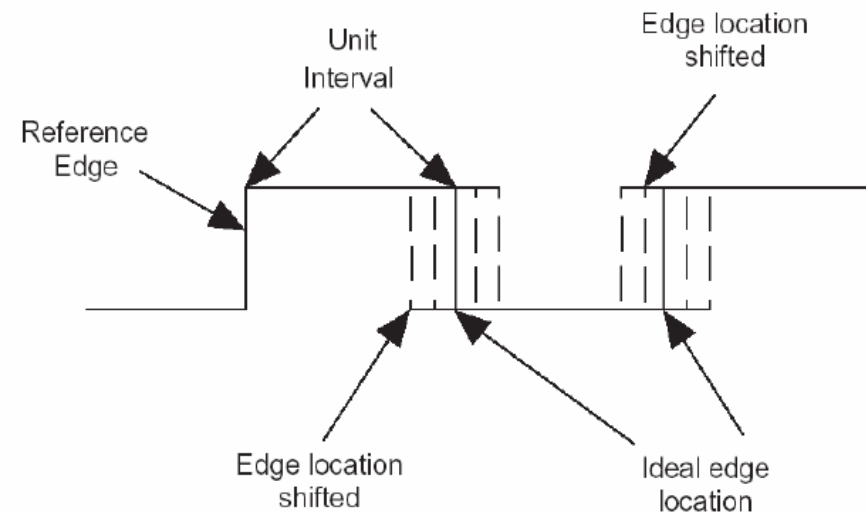
Uzlabojumi

- Pēc zināma laika (N-1 taktīm) visas N posmi strādā un konveijers ir pilns
- Teorētiski konveijers var darboties nodrošinot maksimālu paralēlismu (N komanda izpildās vienlaicīgi) Liekās - jo vairāk posmu jo labāka veikspēja
- Diemžēl jāņem vērā tas ka:
 - Lielāks posmu skaits nozīmē arī lielākus virstēriņus uz datu pārvietošanu un sinhronizāciju starp posmiem (āiztures (latches), takts nobīdes (skew), takts trīces (jitter) $20 \text{ ps} = 1/20 \cdot 10^{-12} = ?$)
 - Diemžēl palielinoties konveijera posmu skaitam aug arī CPU sarežģītība
 - Nav iespējams turēt konveijeru pilnu dēļ konveijeru konfliktiem (hazards)
 - CPI vairs nav konstyants bet gan sastāv no divām daļām (ideālais un aizkaves)
- Piemēri:
 - Pentium: 5. posmu konveijers veseleim skaitļiem un 8. posmu fp komandām
 - PowerPC: 4. posmu konveijers veselo skaitļu komandām un 6. posmu fp komandām

Skew un jitter



Clock Skew



Clock Jitter

Pamati

- Datori izpilda miljoniem darbību sekundē tāpēc mums ir svarīga to caurlaides spēja nevis darbības ārtums
- Sadalām komandas izpildi daļās IF DI CO FO EI WR
- Katrā laika momentā **dažādas komandas būs dažādos izpildes posmos**
- **Ilgākais** posma izpildes laiks noteiks takts frekvenci
- **CPI=1** (ideālā gadījumā)
- Uzlabojums = posmu skaits (ideālā gadījumā)
- Diemžēl vairāki faktori neļauj sasniegt ideālus:
 - Nevar sadalīt vienādos posmos
 - Jāpatērē laiks lai aizpildītu un iztukšotu konveijeru
 - Virstēriņi darbību sinhronizācijai, datu pārraidei utml
 - Konflikti (Structural, data, control hazards)

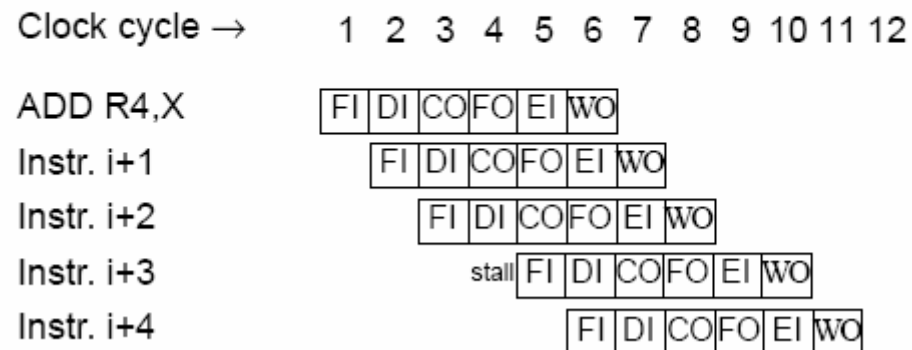
Konflikti

- Konveijera riski ir situācijas kad kādas problēmas neļauj izpildīt komandu tai atvēlētajā taktī.
- Šādu komandu sauc par aizkavētu (stalled)
- Kad komanda tiek aizkavēta visas pārējās komandas kas atrodas konveijerā aiz aizkavētās komandas arī tiek aizkavētas
- Komandas kas atrodas pirms aizkavētās komandas var tikt izpildītas
- Aizkaves gadījumā jaunas komandas netiek ielādētas.
- Konfliktu veidi:
 - Strukturālie (Structural hazards) Aparatūra nespēj izpildīt komandu secību ja piem. divas komandas grib izmantot vienu resursu
 - Datu (Data hazards) Komanda ir atkarīga no iepriekš esošas komandas rezultāta
 - Vadības (Control hazards) Zarošanās un citas komandas kas maina PC

Struktūras konflikti

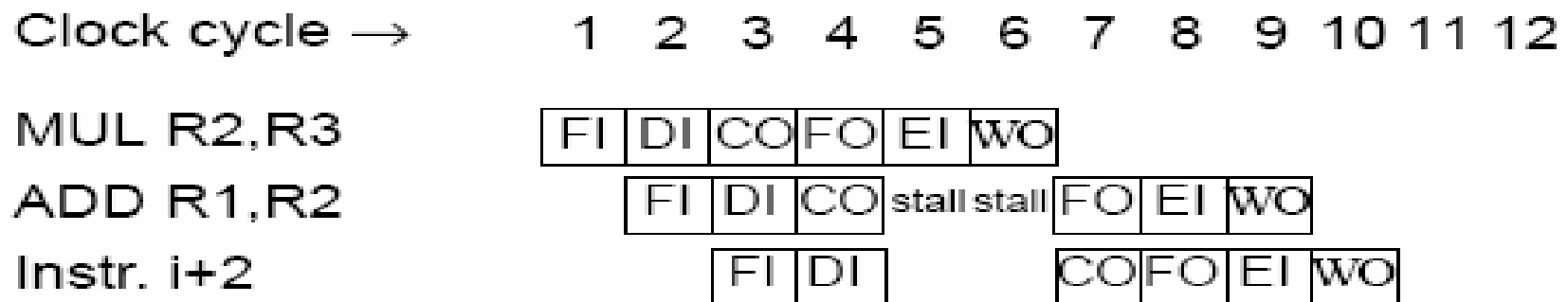
- Struktūras konflikti rodas tad kad divas vai vairāk komandas pieprasa vienu un to pašu resursu
- Parasti lai izvairītos:
 - Dublē resursus
 - Konveijerizē resursus
 - Pārkārto komandas
- Ir gadījumi kad šos konfliktus nevar novērst un tad ir iāaizkavē konveijers

Instruction ADD R4,X fetches in the FO stage operand X from memory. The memory doesn't accept another access during that cycle.



Datu konflikti

I1: MUL R2,R3 $R2 \leftarrow R2 * R3$
 I2: ADD R1,R2 $R1 \leftarrow R1 + R2$




Before executing its FO stage, the ADD instruction is stalled until the MUL instruction has written the result into R2.

Penalty: 2 cycles

3. Veidu datu konflikti

- Read After Write (RAW)

Komanda_J cenšas nolasīt operandu pirms Komanda_I pieraksta to

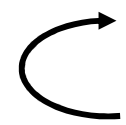
 I: add r1,r2,r3
J: sub r4,r1,r3

- To izraisa atkarība ("dependance") ko izveido programmētājs vai kompilators un kura pieprasa šo datu komunikāciju

3. Veidu datu konflikti

- Write After Read (WAR)

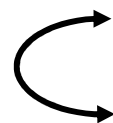
Komanda_J pieraksta operandu pirms Komanda_I to nolasa

 I: sub r4, **r1**, r3
J: add **r1**, r2, r3
K: mul r6, r1, r7

- Sauc par “anti-dependence” un to izraisa vārda “r1” atkārtota izmantošana
- Vienkāršā konveijerā neiespējami jo:
 - Visas komandas izpildās vienādā taktu skaitā
 - Nolasa vienmēr 3. taktī bet pieraksta vienmēr 6. taktī
- WAR konflikti rodas ja komandas tiek izpildītas ārpus secības vai tās pieklūst datiem vēlāk nekā parasti

3. Veidu datu konflikti

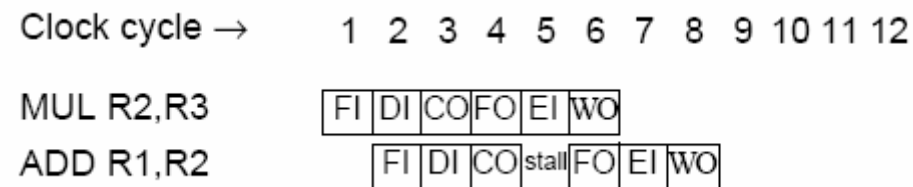
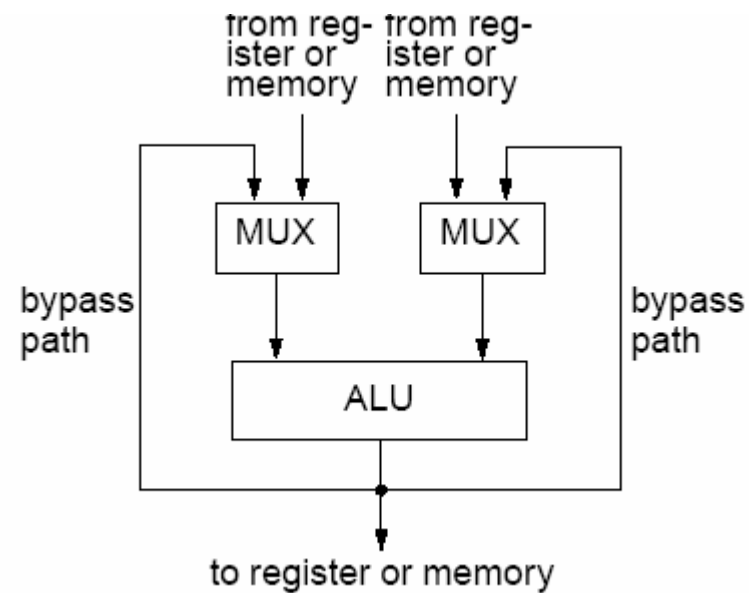
- Write After Write (WAW)
Komanda_j pieraksta operandu pirms Komanda_i pieraksta to pašu operandu

 I: sub **r1**, r4, r3
J: add **r1**, r2, r3
K: mul r6, r1, r7

- Tiek saukta par “output dependence” un to arī izraisa vārda “r1” atkārtota izmantošana
- Vienkāršā konveijerā neiespējami jo:
 - Visas komandas izpildās vienādā taktu skaitā
 - Pieraksta vienmēr 6. taktī
- WAR un WAW ir iespējami tālāk apskatītos sarežģītajos konveijeros

Datu konfliktu novēršana

- Dažus datu konfliktus var novērst ar datu pārsūtīšanu (forwarding, bypassing)



Datu konfliktu novēršana

- Dažus datu konfliktus nevar novērst ar aparatūras palīdzību (piem. load)
- Bet tos var novērst ar programmatūras palīdzību.

Datu konfliktu novēršana

Piemēram izveidojiet pēc iespējas ātrāku dotā koda izpildi

$$a = b + c$$

$$d = e - f$$

Pieņemot to ka a, b, c, d, e , un f atrodas atmiņā

Lēni:

Ātri:

LW	Rb,b
LW	Rc,c
ADD	Ra,Rb,Rc
SW	a,Ra
LW	Re,e
LW	Rf,f
SUB	Rd,Re,Rf
SW	d,Rd

LW	Rb,b
LW	Rc,c
LW	Re,e
ADD	Ra,Rb,Rc
LW	Rf,f
SW	a,Ra
SUB	Rd,Re,Rf
SW	d,Rd

2006/2007 m.g.

Mājas darbi

- http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/pipe_title.html
- Vai var vēl uzlabot 18. slaida piemēru?

BUJ