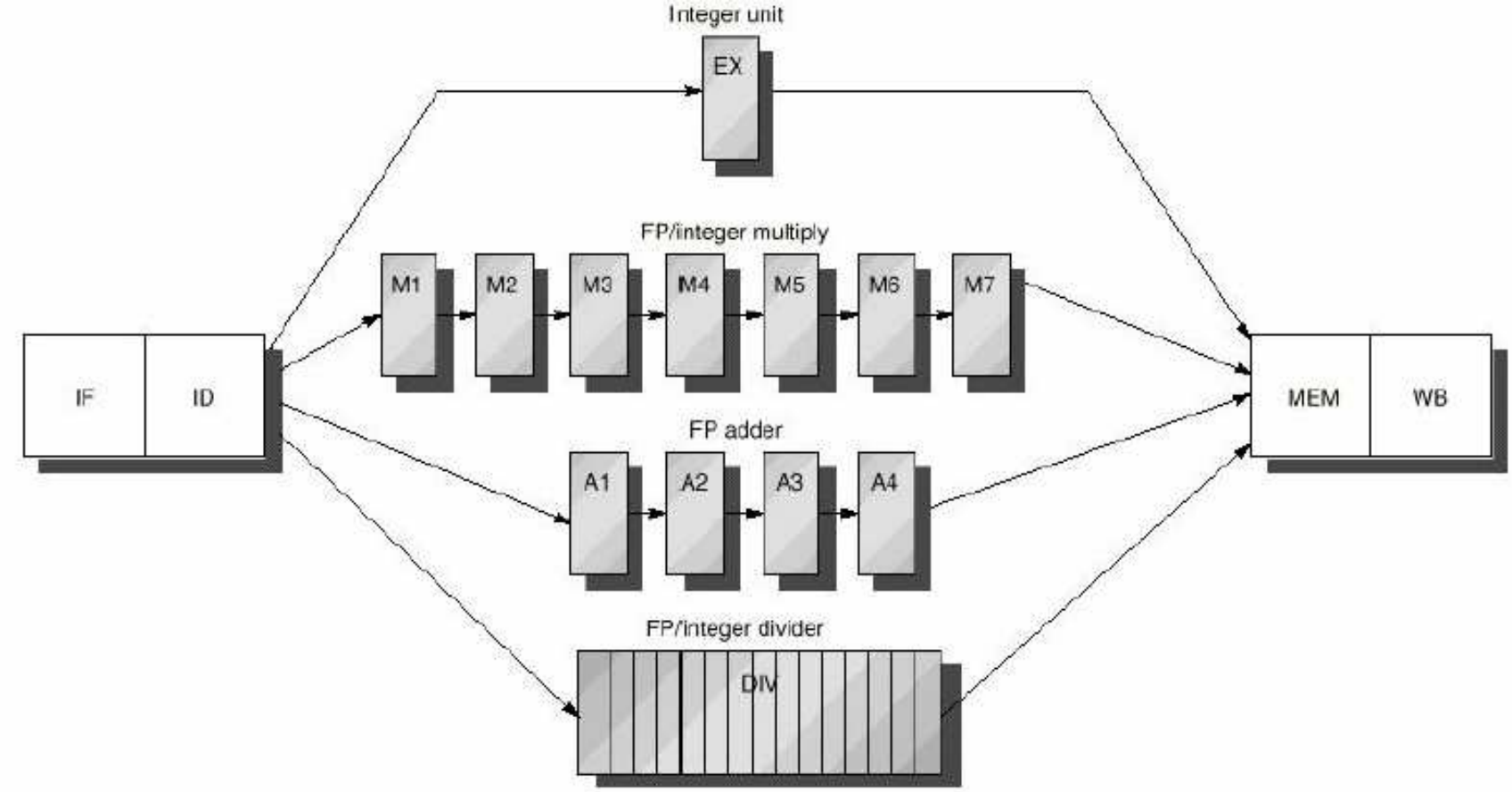
Komandu paralēla izpilde aparatūras līmenī un vairāku komandu paralēla izpilde

Problēmas ar dažādu komandu izpildes ilgumu



Problēmas ar dažādu komandu izpildes ilgumu

1. Struktūras konflikti ja iekārtas nav dublētas / konveijerizētas
2. Rakstīšanas darbības var būt vairāk par vienu gab. vienā taktī
3. WAW
4. Komandas beidz darbu nepareizā secībā
5. RAW

Problēmas nedaudz samazinās nodalot FP un veselo skaitļu mezglus (reģistrus) jo izpildes laiki nedaudz izlīdzinās

Konveijera problēmas

- Konflikti ierobežo veiktspēju
  - Struktūras - prasa papildus aparatūru
  - Datu – prasa pārsūtīšanas loģiku un kompilatora plānošanu
  - Vadības – prasa ātrāku izvērtējumu, ***aizkavētās pārejas***, ***pāreju paredzēšanu***
- Palielinot konveijera dziļumu palielinās zaudējumu apjoms un komandu latentums
- ***Izņēmumi*** un FP aritmētika sarežģī konveijerizāciju vēl vairāk
- Kompilatori var samazināt datu un vadības konfliktu zaudējumus
  - Koda pārplānošana
  - Aizkaves zonas (datu pārvietošanas un vadības komandām)
  - Statiskā zarojumu paredzēšana

Dinamiska plānošana konveijeros

- Statiskā plānošana
  - Apturēt komandu un neņemt jaunas komandas kamēr nav novērts konflikts
  - Lietot SW konveijera plānošanu lai samazinātu aizkaves
- Dinamiskā plānošana
  - Izmanto HW lai pārplānotu (pārvietotu) komandas ar mērķi samazināt aizkaves
  - Var izmantot arī gadījumos kad atkarības nav zināmas kompilācijas laikā
  - Ļoti sarežģī aparatūru
  - Rada ārpus secības izpildi kas savukārt noved pie WAW un WAR konfliktiem

Scoreboarding

- Lieto ***centralizētu*** vadības mezglu (scoreboard) kas vada un kontrolē vairākus konveijera izpildes mezglus.
- ***Scoreboard mezgls*** ir atbildīgs par komandu izsniegšanu un konfliktu noteikšanu
- **Izsniegšana** — izsniedz komandu un atjauno vadības tabulu tikai tad ja nepieciešamais mezgls ir brīvs un neviena cita aktīva komanda negrib rakstīt tajā pat reģistrā (WAW). Citos gadījumos aptur izpildi kamēr tiek novērsti struktūras konflikti.
- **Operandu lasīšana** — Scoreboard pārbauda operandu pieejamību un kad tie ir pieejami nodod komandu funkcionālajam blokam tos nolasīt un sākt izpildi. Operands ir pieejams ja neviena aktīva komanda tajā nevēlas rakstīt (vai jau neraksta) (**RAW**)
- **Izpilde** — mezgli paziņo par uzdotās darbības izpildi scoreboard iekārtai
- **Rezultāta pieraksts** — komanda nevar pierakstīt rezultātu kamēr kāda cita aktīva komanda nav nolasījusi operandus (**WAR**)

Reģistru pārsaukšana

- Kompilatori var likvidēt tikai dalu no WAW un WAR konfliktiem jo:
  - Nav pietiekoši daudz reģistru
  - Konflikti var būt pāri cikliem
- Atbilde ir dinamiski mainīt reģistru vārdus

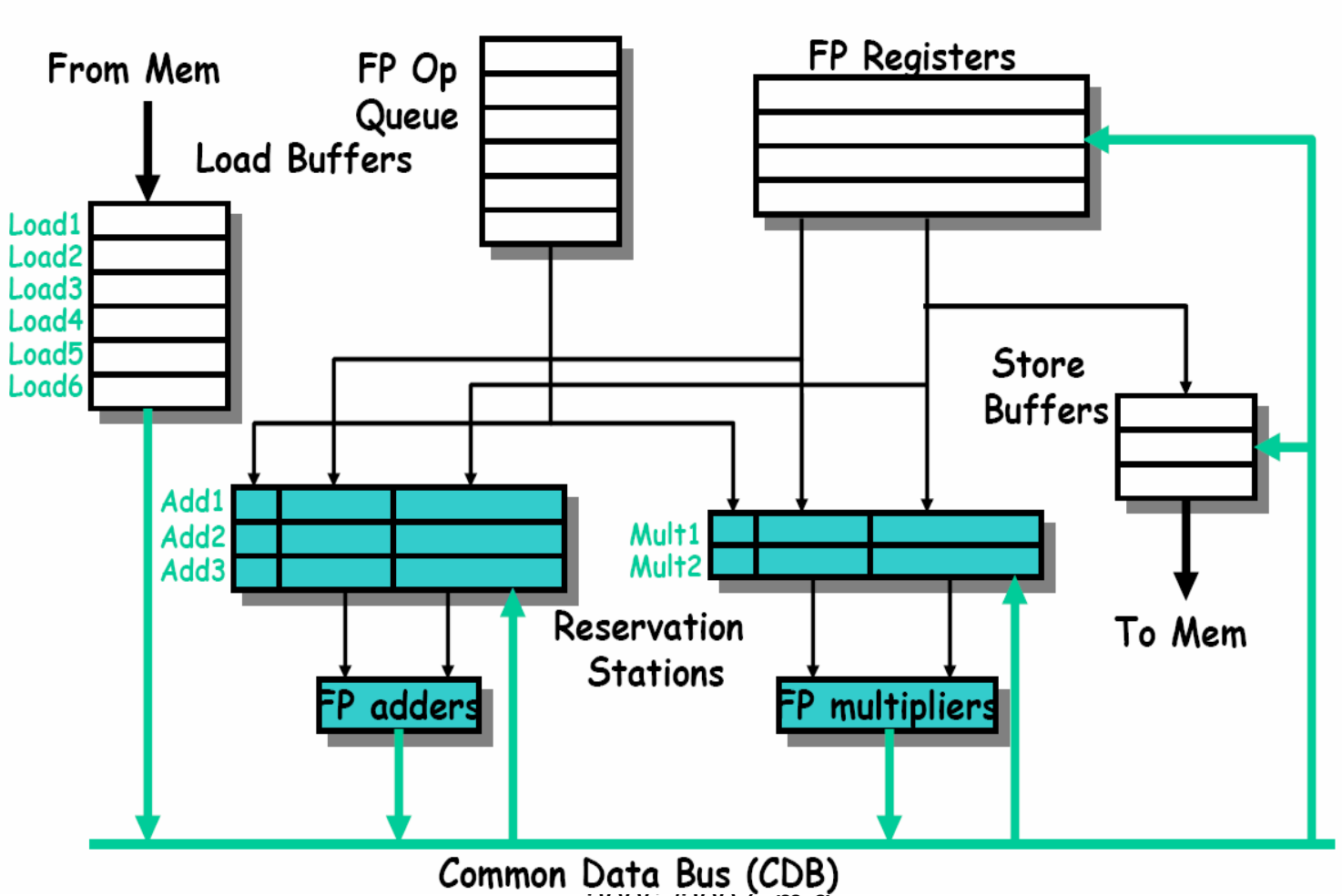
```
DIV   F0, F2, F4          DIV   F0, F2, F4
ADD   F6, F0, F8          ADD   S, F0, F8
SW    F6, 0(R1)     →     SW    S, 0(R1)
SUB   F8, F10, F14        SUB   T, F10, F14
MUL   F6, F10, F8         MUL   F6, F10, T
```

- nevis ***vietas***
  - Ir daudz vairāk vietu (“reservation stations” vai

Tomasulo algoritms

- ***Sadalīta dinamiskās plānošanas*** shēma ko atklāja R.Tomasulo (ieviesa IBM 360).
  - Izejas operandi tiek *iekopēti* pagaidu vietās (reservation station) ja tādas un tie ir pieejami (WAR)
  - Rezultāti tiek noraidīti plašapraides kopnē (CDB) novēršot gaidīšanu tajā
  - Noraidītā informācija satur gan datus gan izejas mezgla ***tagu***.
  - Visi gaidošie mezgli klausās kopni un analizē vai tas ir gaidītais rezultāts.
  - Lieto dinamisku reģistru pārsaukšanu lai novērstu WAW un WAR konfliktus.

Tomasulo algoritms

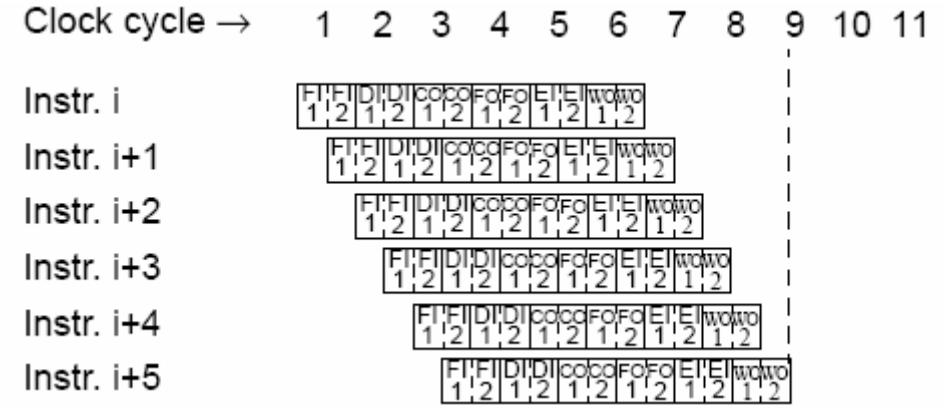


Mērķis = CPI >1

- Superskalārās ir tādas arhitektūras kuras ļauj ***palaist*** vairākas komandas vienlaicīgi un ***izpildīt*** *tās* neatkarīgi
  - Konveijerizācija ļāva izpildīt vairākas komandas vienlaicīgi bet tām bija jāatrodas dažādos konveijera posmos
- Uz superskalārām arhitektūrām var attiecināt visu līdz šim apskatīto konveijerizācija kontekstā (tikai pieļaujot iespēju izpildīt vairākas komandas vienā konveijera posmā vienlaicīgi)
- Šodien ir divi pamata veidi kā to panākt:
  1. Superpipelining
  2. Superscalar

Superpipelining

- Superpipelining pamatdoma ir sadalīt konveijera taktis apakštaktīs tādejādi palielinot katrā laika momentā konveijerā esošo komandu skaitu
  - Sadalot takti divās daļās rezultāts no konveijera tiks iegūts katras t/2 sekundēs
  - Dotajai arhitektūrai un ISA ir kāds noteikts konveijera posmu skaits virs kura palielinot to kopējā veiktspēja kritīsies (P4 vs P3)

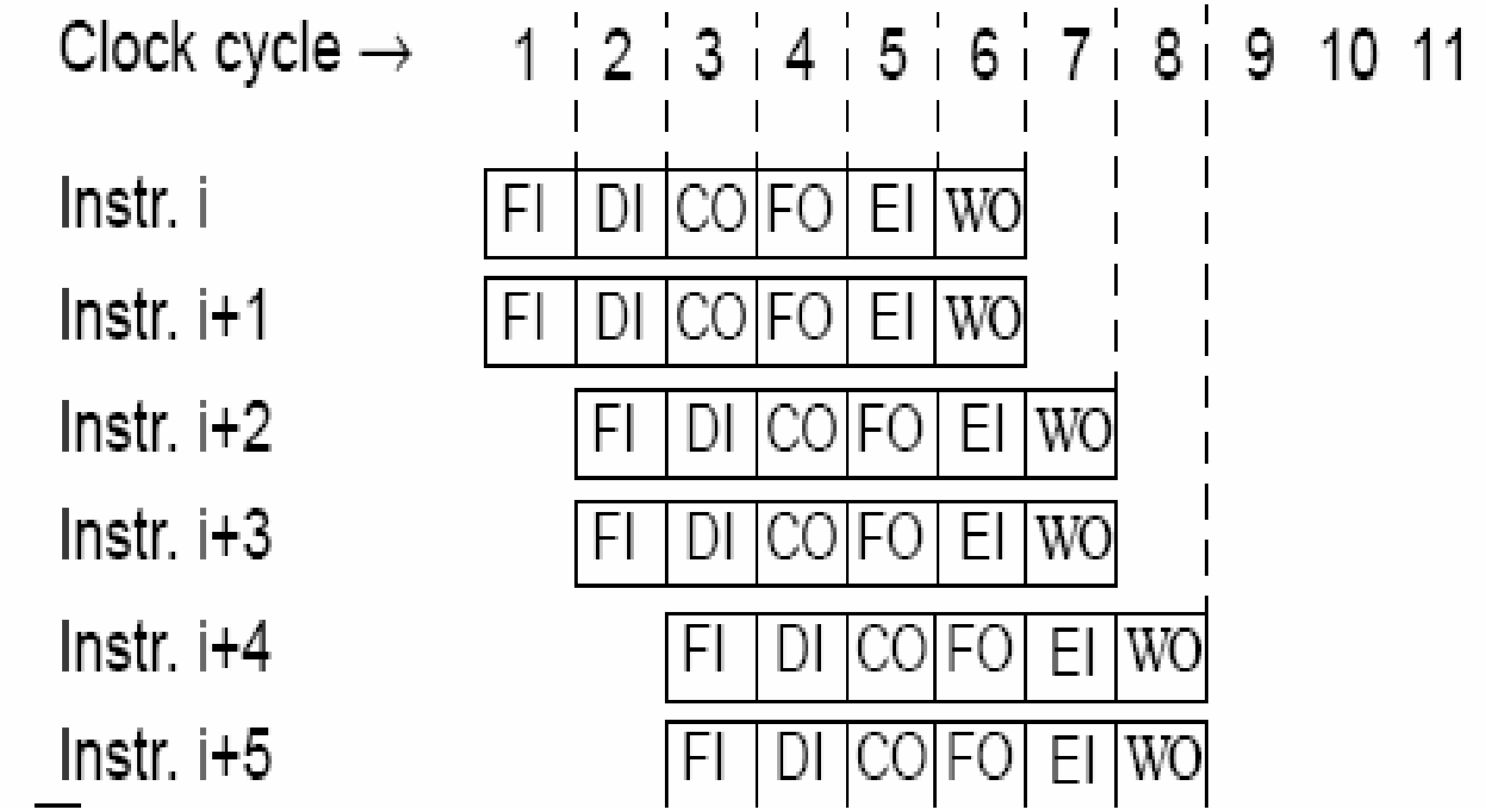


- Lai vēl palielinātu ātrdarbību tiek ieviestas superskalārās arhitektūras.

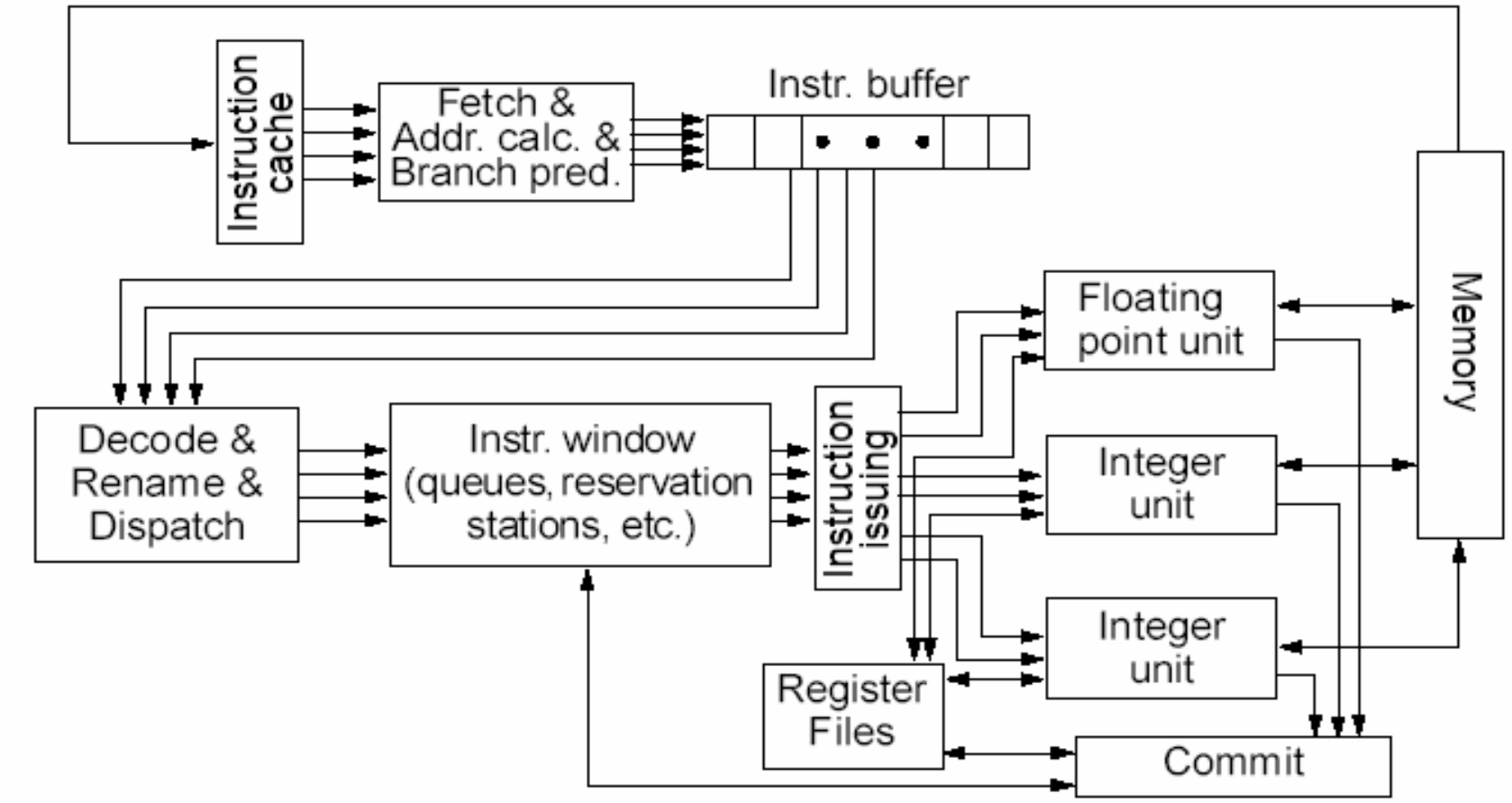
Superskalārās arh.

- Superskalarās arhitektūras ļauj katrā laika taktī palaist un izpildīt vairākas komandas.
- Superskalārās arhitektūras pamatā ir vairāki konveijeri kas strādā paralēli (1 - 8 gab. kurus plāno SW vai HW)
- Atkarībā no tā cik un kādi mezgli ir izveidoti paralēlai izpildei ir atkarīgs kādas un cik komandas varēs izpildīt paralēli (vienkāršākais variants ir FP un veselo skaitļu iekārtu nodalīšana un to izveide ar saviem neatkarīgiem konveijeriem)
- Superskalārās arhitektūras var būt statiski vai dinamiski plānotas
  - izsniedz mainīgu daudzumu komandu katrā taktī
- VLIW (EPIC)
  - Izsniedz nemainīgu daudzumu komandu katrā taktī kas noformētas kā viena gara komanda (komandu pakete)
  - Līdzība ar statiski plānotām superskalārajām arhitektūrām

Superskalārās arh.



Superskalārās arh.



Superskalāro arh. limiti

- Nodalīt ALM / FP iekš HW ir viegli bet ieguvums ir tikai programmām ar:
  - Tieši 50% FP darbībām &
  - Kurās nav konfliktu
- Ja tiek izsniegtas vairāk par divām komandām vienā taktī tad ir daudz komplicētāka dekodēšanas un izsniegšanas loģika
- Pat divu komandu gadījumā ir jāapskata divi operācijas kodi, seši adrešu specifikatori un jāizlemj vai abas komandas var izsniegt
- Modernos CPU tiek izsniegtas no 2 līdz 8 komandām vienā taktī

VLIW

- Very Long Instruction Word (VLIW) CPU
  - Vienkāršojas dekodēšana bet ir mazāk komandu
  - Garais komandas vārds satur vairākas darbības
  - Pēc noklusējuma visas darbības ko kompilators ievieto garajā komandas vārdā var izpildīt paralēli
  - Piemēram, 2 ALM operācijas, 2 FP operācijas, 2 atmiņas piekļuves, 1 zarojums
    - 16 līdz 24 biti katram laukam => 7*16 (112) līdz 7*24 (168) bitu vārds
  - Pāri zarojumiem kodu plāno *kompilators*
- Explicitly Parallel Instruction Computer (ia64)
  - Komandas tiek noformētas kā komandu grupa kuras garums var būt mainīgs (pat visa programma) bet kurā nav reģistru datu atkarības

Vairāku komandu izsniegšanas ierobežojumi

- Ierobežojumi kas ir specifiski superskalārām un VLIW realizācijām
  - Superskalārajās arhitektūrās ir problēmas ar komandu dekodēšanu un izsniegšanu jo tās vajag *n* reizes vairāk
  - VLIW koda apjoms ir daudz lielāks jo ***jāatver cikli*** krietni vairāk un parādās neizmantotie lauki VLIW vārdā
  - VLIW gadījumā viena aizkave aptur visas komandas
  - VLIW nav binārās atpakaļsaderības ar x86 u.c. arhitektūrām
- Ierobežojumi ILP kopumā
  - Ja ir 1 zarojums katrās piecās komandās - kā turēt pilnu konveijeru?
  - Mezglu latentumi nav vienādi un tāpēc ir grūtības plānot komandas
  - Aptuveni nepieciešams iegūt (Konveijera dziļums*Funkcionālo bloku skaits) neatkarīgu komandu lai visi mezgli būtu noslogoti

Apkopojums

Common Name	Issue Structure	Hazard Detection	Scheduling	Examples
Superscalar (static)	Dynamic	Hardware	Static	Sun UltraSPARC II/III
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	IBM POWER2
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	Pentium III/4, MIPS R10K, Alpha 21264, IBM POWER4, HP PA8500
VLIW	Static	Software	Static	Trimedia, i860
EPIC	"mostly" static	mostly software	mostly static	Itanium (IA64)

Kopumā

- Superskalārās un VLIW arhitektūras
  - CPI > 1
  - Superskalārās arhitektūras ir vairāk atkarīgas no HW (dinamiski plānotas)
  - VLIW arhitektūras ir vairāk atkarīgas no SW (statiski plānotas)
  - Jo vairāk komandu izsniedz vienā laikā jo ***lielāki zaudējumi konfliktu gadījumos***!!

Ko tālāk

- Spekulatīva ielāde & poison biti
- “Trace scheduling” jeb viena programmas zara izpilde ***paredzot to kompilācijas laikā*** (jāpievieno papildus kods ja minējums nav bijis pareizs)
- “Branch Predication” un izpilde no abām zarojuma pusēm
- ILP ziņā kopš 2004 situācija ir bēdīga ;(
- Varbūt TLP (hyperthreading) un SMP/multicore...?
- Mājās:
  - http://arstechnica.com/articles/paedia/cpu/hyperthreading.ars/1
  - http://dt.cs.rtu.lv/direct.php/18/465