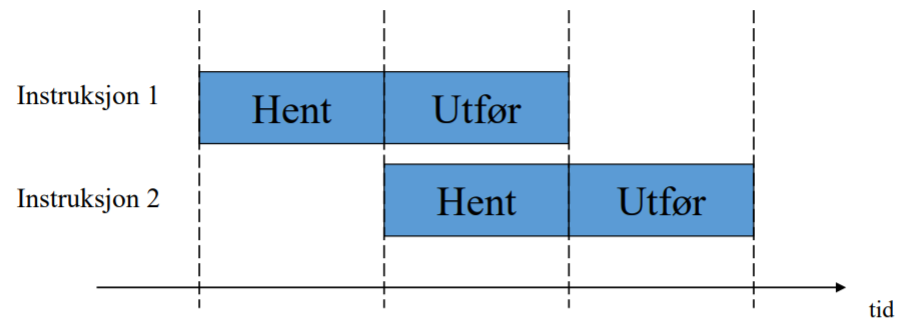
Moderne prosessarkitektur

Instruksjonssyklusen består av hentefasen, utføringsfasen og avbruddsfasen. Disse har så langt blitt utført sekvensielt, men man kan øke hastigheten på prosessoren ved å utføre disse mer parallelt.

**Pipeline**: utføre instruksjoner parallelt. Man kan trenger egen elektronikk for hent og utfør instruksjoner. Problemet med dette er at utføringssyklusen tar mye lenger tid enn hentesyklusen. Utføresyklusen krever flere klokkesykluser enn hentesyklusen.

Kan sammenlignes med et samlebånd på en bilfabrikk. Fra utsiden ser det ut som produksjonstiden er veldig kort, til tross for at samlebåndet er langt. Altså det jobbes med bilene parallelt.



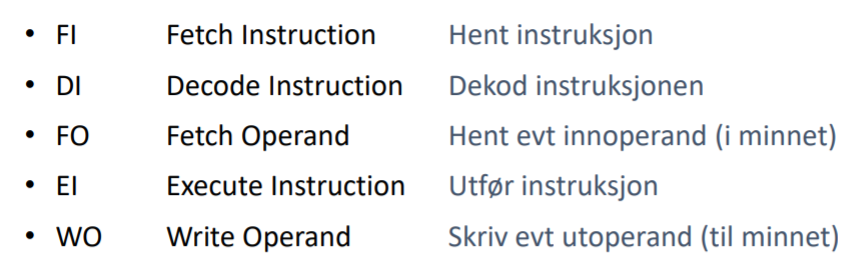
**Hentefasen**: gå dit programtelleren peker og hente data som ligger der.

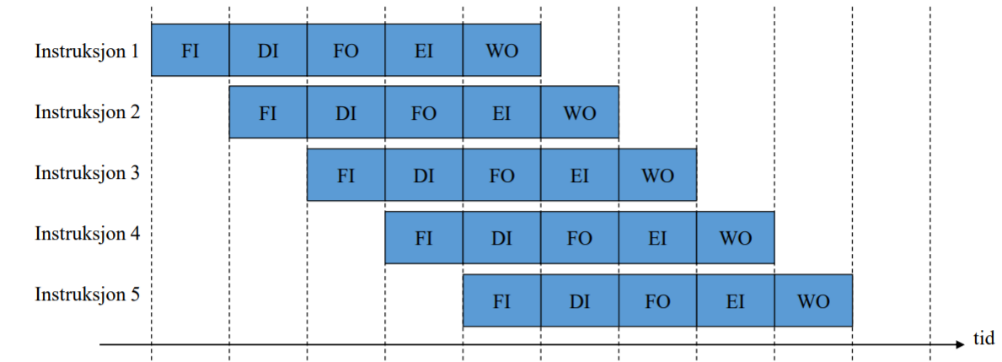
**Utføringsfasen**: tolke instruksjonen, eventuelt hente data fra minnet om de trengs, utføre instruksjonen, eventuelt skrive til minnet dersom et resultat skal skrives dit.

*Man ser at det er mer som foregår i utføringssyklusen i forhold til hentesyklusen.*

**Finere oppdeling**

Ideen er at man kan dele opp hentesyklusen på samme måte som vi delte hent og utføringssyklusen som kjører parallelt. I vanlig prosessor design er det 4-8 ulike typer mikrooperasjoner.



Femtrinns pipeline *(kan finnes fler enn fem)*

Denne typen parallellisering kalles «instruction-level parallelism».

Problemer med dette:

* Flere instruksjoner kan trenge samme ressurs.
* Resultater som trengs i en instruksjon er ikke ferdig beregnet.
* Slike situasjoner kalles ***Hasard***

# **Hasard**

*Man kan f.eks. ikke lese(FI) og skrive(WO) til minnet samtidig. Når dette skjer kalles det en hasard. Vanligvis har ikke alle instruksjoner de samme mikrooperasjonene, derfor skjer dette ikke hver gang slik som i eksempelet på forrige side.*

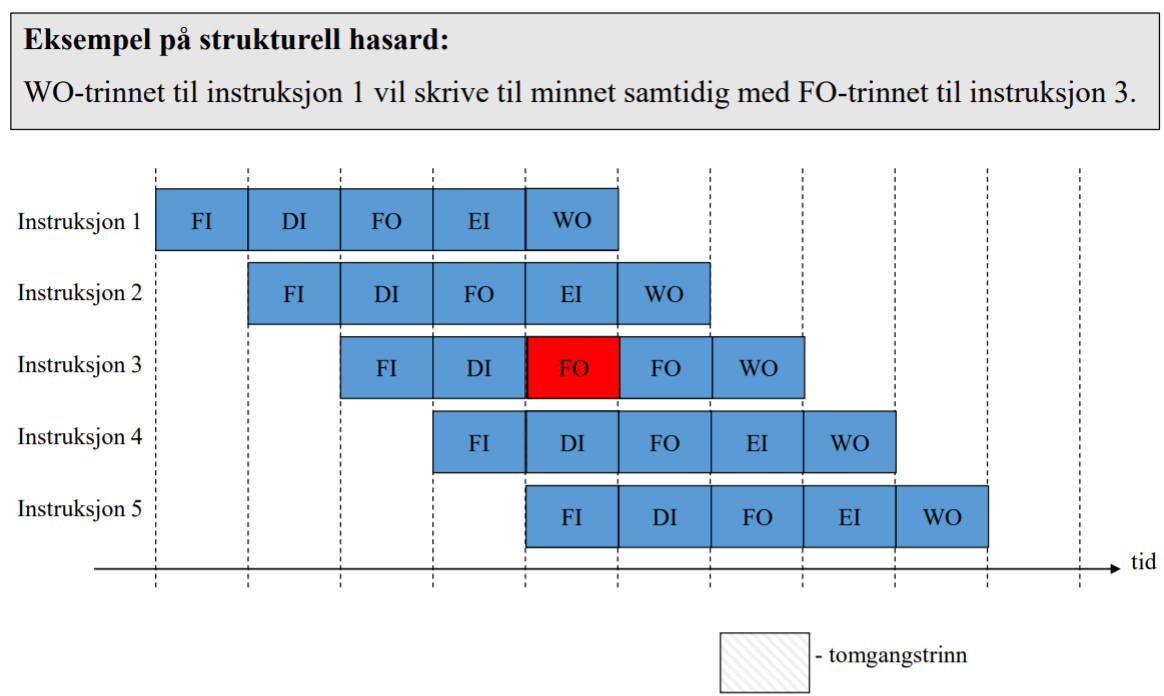
**Tre ulike typer:**

1. **Strukturell hasard**: Ikke nok maskinvare til å gjøre noe samtidig. Typisk er minnet, når to instruksjoner trenger samme ressurs samtidig.
2. **Data-hasard**: En instruksjon trenger resultatet fra en annen instruksjon som ikke har resultatet klart enda.
3. **Kontroll**-hasard: Hopp i programmert gjør at vi ikke vet hvilke instruksjoner som skal utføres før hoppinstruksjonen er utført.

**Strukturell hasard**

**Løsning**: Det settes inn tomgangstrinn i den siste instruksjonen.

*Legg merke til at et tomrom legges til den røde boksen og videre nedover. Alle instruksjoner etter dette punktet blir forflyttet. Animasjoner ikke på pdf.*

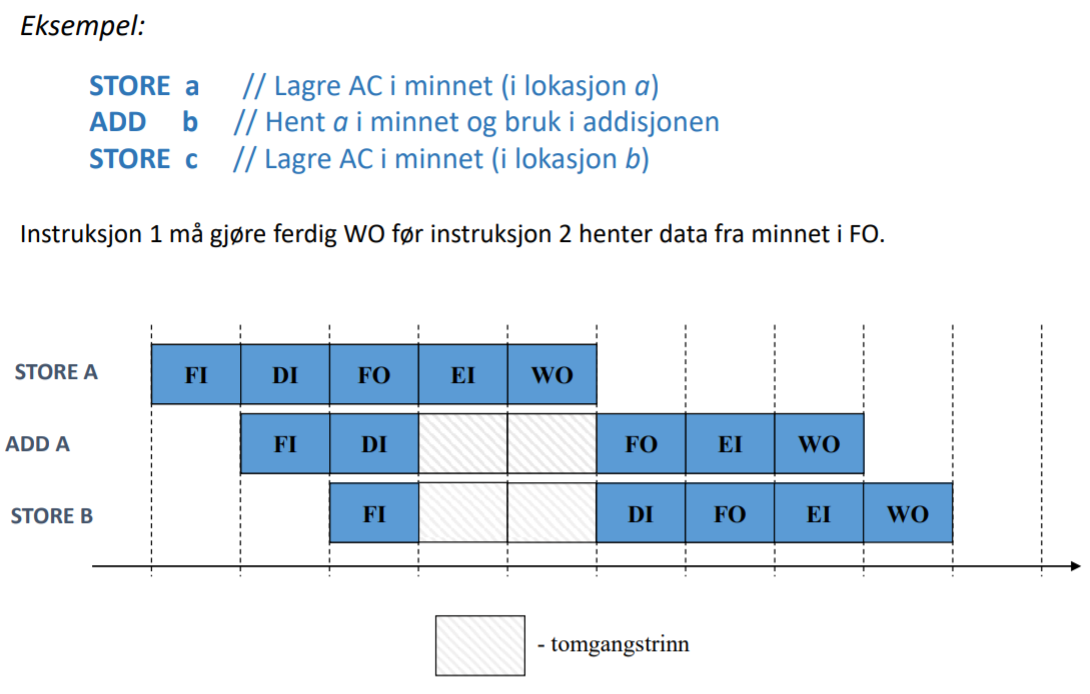


Fordelen med å dele data- og instruksjons cache er at man kan hente instruksjonen samtidig som man henter dataen. Det er hovedgrunnen til at L1-cachen er delt opp slik. Spørsmål på eksamen 2018 høst.

**Data-hasard**

**Problem**: En instruksjon trenger data som ennå ikke ferdig (fordi instruksjonen som produserer data ikke har kommet langt nok i pipen)

**Løsning**: Det settes inn tomgangstrinn i den siste instruksjonen



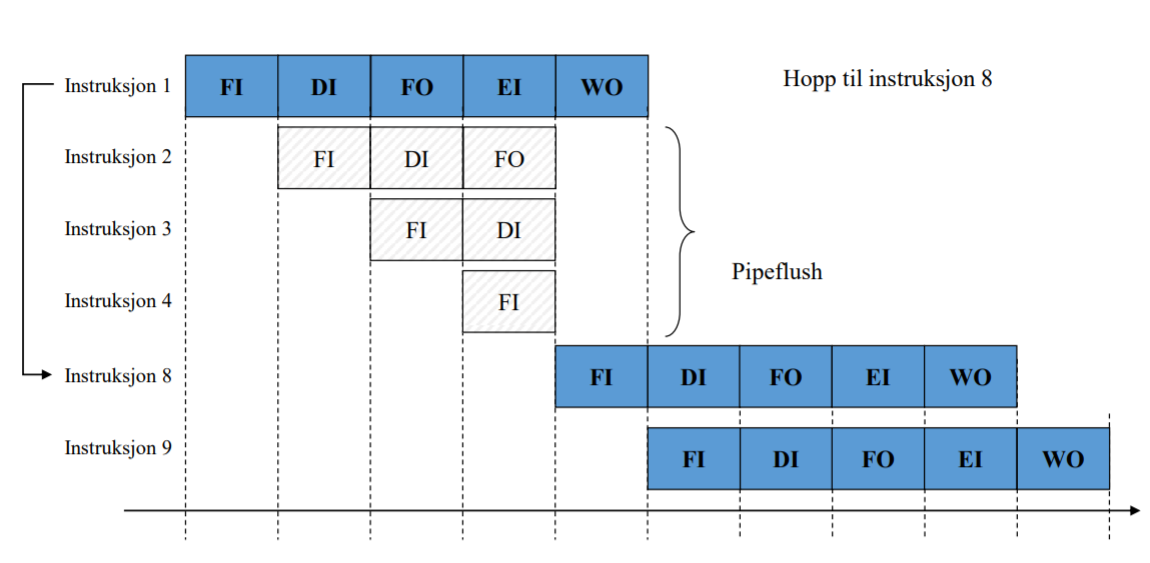
a\*

**Kontroll-hasard**

**Problem**: En hoppinstruksjon gjør at neste instruksjon ikke ligger i neste sekvensielle minnelokasjon

**Løsning**: CPU må gjette hvor det skal hoppes: dette er enkelt for ubetingede hopp, men vanskelig for betingede hopp.

* Konsekvens ved feilgjetting: pipeflush (fjerne instruksjoner hvis hopp skal foregå).

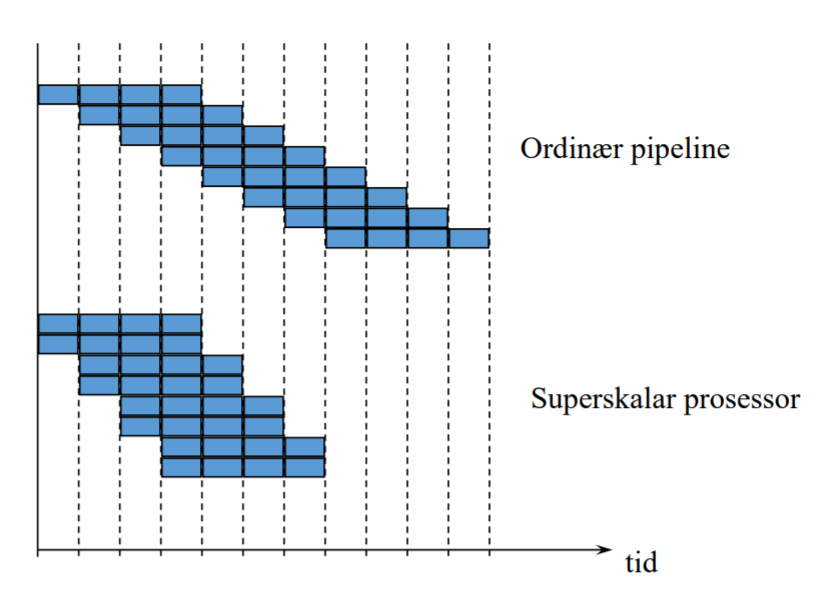


# Superskalare struktur

***Superskalare prosessorer***: Flere parallelle pipelines, eller flere samlebånd hvis man bruker bilfabrikk eksempelet.

Antall parallelle pipelines kalles prosessorens **grad** av superskalar. F.eks. grad to har to parallelle pipelines. Instruksjonene utføres parvis.

Det er derimot ikke fornuftig å ha flere pipelines.



**Dynamisk utføring**

Dersom en instruksjon må vente på minnet eller L2-cache vil CPU begynne å jobbe på instruksjoner videre i programmet selv om den ikke vet helt sikkert at instruksjonene virkelig skal utføres.

*Viktige begrep:*

* **Dataflytanalyse:** Finn ut hvilke instruksjoner som er avhengige av hverandre
* **Gjetting ved hopp:** Effektiviteten blir svært avhengig av det gjettes riktig ved betingede hopp. Feil gjetting gjør at det utføres feil instruksjoner
* **Spekulativ utføring**: Instruksjoner utføres gjerne i en helt annen rekkefølge enn den sekvensielle. Resultatene legges i “foreløpige registre”. Deretter frigis de (retired) i samme rekkefølge som de er programmert.