Eksamensbesvarelser

# Parallellitet på instruksjonsnivå

## Hva menes med begrepet?

At en og samme prosessor kan utføre flere instruksjoner parallelt.

## Mekanismer for å implementere dette?

* Pipeline
  + **Hentesyklusen opp i flere trinn slik at flere instruksjoner kan utføres parallelt.**
    - Økt effektivitet
  + Fare for *hasarder*
* Superskalar prosessor
  + **Flere parallelle pipelines**
  + Grad av superskalaritet = antall parallelle pipelines
* Dynamisk utføring
  + **prosessoren utføre hopp ved best mulig flyt ved hjelp av algoritmer som gjetter hopp.**
    - Dynamisk forutsigelse
      * Lagrer info om tidligere hopp utført og tar avgjørelser om opp på bakgrunn av denne
    - Statisk forutsigelse
      * Gjetter likt for alle hopp

## Hasarder

* Hvem er ansvarlig for å løse disse?
  + Prosessoren
  + Nevn at programmereren og kompilatoren kan hjelpe prosessoren
* Hvilken typer hasarder har vi?
  + Strukturellhasard
    - **Flere instruksjoner trenger samme ressurs samtidig**
    - Løsning: sette inn et ventetrinn
      * Ulempe er at dette forplanter seg videre i andre instruksjoner slik at en liten forsinkelse 🡪 ikke lenger en instruksjon ferdigstilt pr klokkeperiode
  + Datahasard
    - **Om to instruksjoner som er *dataavhengig* kjøres parallelt**
    - Løsning: sette inn et ventetrinn
      * Vi får samme ulempe som ved strukturellhasard
  + Kontrollhasard
    - **Når en hoppinstruksjon gjør at neste instruksjon ikke ligger i neste sekvensielle minnelokasjon**
    - Ulemper med denne typen hasard er muligheten for *pipeflush* ved feil gjetting.
      * Betinget hopp
      * Ubetinget hopp

# Pipeline og cache

## Hvorfor benyttes en delt cache gjerne i en prosessor med pipelinearkitetktur?

Ved bruk av pipelinearkitetkur kan det oppstå strukturelle hasarder om flere instruksjoner vil bruke bussen samtidig. Om vi bruker en delt cache vil vi kunne hente data og instruksjoner parallelt uten at strukturelle hasarder er et problem, siden hver cache har sin egen buss. Dette eliminerer ikke strukturelle hasarder men er med på å forebygge flere. Med dette får vi økt effektivitet.

# Vranglås

## Betingelsen for vranglås

* **No preemtions**
  + Ressursen kan kun frigis av prosessen som bruker den. Ingen kan kreve denne ressursen som prosessen har tak i
* **Sirkulær venting**
  + Dette oppstår når hver prosess holder på en ressurs som er ønsket av en annen
* **Hold på ressursen og vent**
  + Prosessen holder på ressurser mens den spør om nye
* **Gjensidig utelukking**
  + En ressurs er enten ledig eller allokert til en prosess. Prosessen som er allokert til ressursen har eksklusiv tilgang og alle andre prosesser som prøver å få tak i ressursen blir utestengt.

## Hvordan løses vranglås

* Med jevne mellomrom må operativsystemet gå gjennom alle ressursene og prosessene for å se om det kan ha oppstått en vranglås.
* OS kan løse vranglåsen:
  + **Preemptions**
    - Ta fra prosessorer ressurser og gi den til andre prosessorer
  + **Rollbacks**
    - Man har jevne sjekkpunkter, her blir statusen til prosessen skrevet til fil så dersom en vranglås har oppstått kan prosessen gi fra seg ressursene og når de har fått tilbake ressursene kan prosessen fortsette ved siste sjekkpunkt.

# Sidedelt minne

## Hvilke problemer har sidedelt minne løst?

Siden store programmer ikke alltid har plass i RAM-en får vi mer effektive bruk av minne ved bruk av sidedelt minne.

## Hva brukes et baseregister til i forbindelse med minneadministrasjon? En sidetabell kan sees på som en tabell med mange base-registre. Forklar.

*Et baseregister sier noe om hvor en prosess starter i det fysiske minnet.* En sidetabell er en datastruktur som inneholder to kolonner, «page» nummer og «frame» nummer. «Page» nummer oppbevarer navnet på de forskjellige del-prosessene mens «frame» nummeret forteller oss start adressen til del-prosessen i det fysiske minnet. På denne måten kan man se på sidetabellen som en liste med start adresser til forskjellige del-prosesser i det fysiske minnet. Siden et baseregister forteller noe om hvor en prosess starter i det fysiske minnet kan man se på sidetabellen som en tabell med base-registre.

# Sikkerhet

## Angi de tre OS sikkerhetsmålene og beskriv kort hensikten med hver. Hvilke trusler prøver hver enkelt å unngå?

**Konfidensialitet:**

Brukeren av OS vil ***opprettholde konfidensialitet av infoen som er lagret i datasystemet***. Det vil si at informasjon ikke bør være tilgjengelig til noen som ikke er autorisert for å mota det.

**Integritet:**

Brukeren vil ***opprettholde integriteten av informasjonen lagret på datasystemet***. Det betyr at informasjonen ikke bør endres eller slettes av personer som ikke er autorisert til å gjøre slike endringer.

**Tilgjengelighet**:

Brukeren vil ***ivareta tilgjengeligheten av tjenesten som er tilbydd av datasystemet***. Det betyr at personene som er autorisert til å bruke systemet burde få gjøre det uten nekting av tjeneste.

**Ansvarlighet**:

Brukeren ***ønsker å sikre ansvarlighet***. Det betyr at det burde være mulig å bestemme hvordan brukere har valgt å bruke deres autoritet, så de kan holdes ansvarlig for valg de har tatt innenfor grensene som er satt av sikkerhet kontroller.

# Schedueling

* Preemptiv cpu-scheduling
  + Ikke-preemptiv scheduling tillater at prosessen kjører på cpu-en helt til den er ferdig. Det finnes ingen vei fra cpu-en og tilbake til cpu-køen.
  + Eks:
    - FCFS (first come first serve)
    - SJN (Shortes job first)
* Ikke preemptiv cpu-scheduling
  + kjørende prosess kan bli avbrutt og ny prosess allokert til cpu-en.
  + Prosesser med høyest prioritering får kjøre på CPU-en resten blir organisert i en CPU-kø
  + Eks:
    - Round Robin
      * Hver prosess har en tidskvant, om prosessen overskredere denne vil den fjernes fra CPU-en og legges tilbake i CPU-køen
      * Små tidskvanter 🡪 kortere responstid (prosessene står kortere i kø, men tiden går mest til kontektsskifte og ikke til å gjøre noe nyttig på cpuen)
      * Store tidskvanter 🡪 lengre responstid (prosessne står lengre i kø, men tiden som brukes blir godt brukt på cpuen)