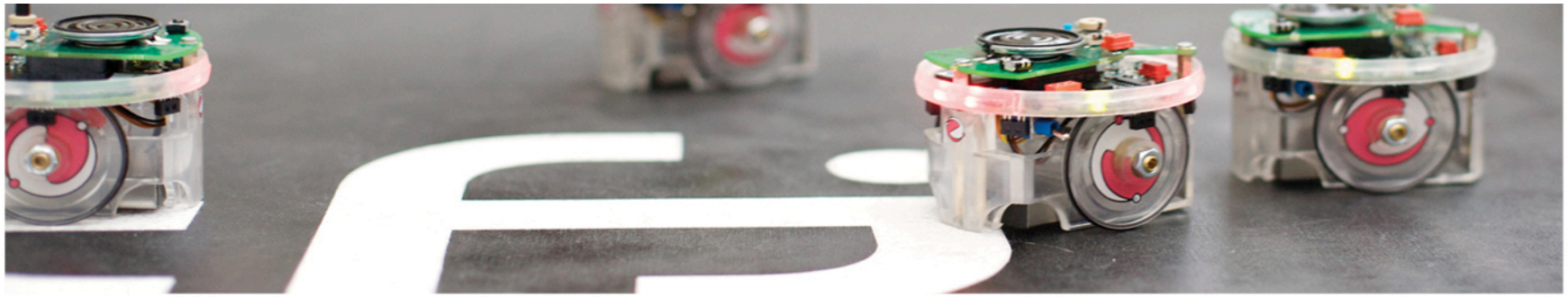


UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

INF2270

Input / Output (I/O)

Hovedpunkter

- Innledning til Input / Output
- Ulike typer I/O
- I/O internt i datamaskinen
- I/O eksternt

Input / Output

- En datamaskin kommuniserer med omverdenen gjennom mange ulike enheter:
  - Harddisk
  - Ram
  - Mus
  - Tastatur
  - Skjerm
  - Nettverk
- Kan dele kommunikasjon mellom to typer:
  - Kommunikasjon mellom enheter internt i maskinen og mellom en datamaskin og direkte koble utstyr.
  - Kommunikasjon mellom ulike datamaskiner knytte sammen i nettverk

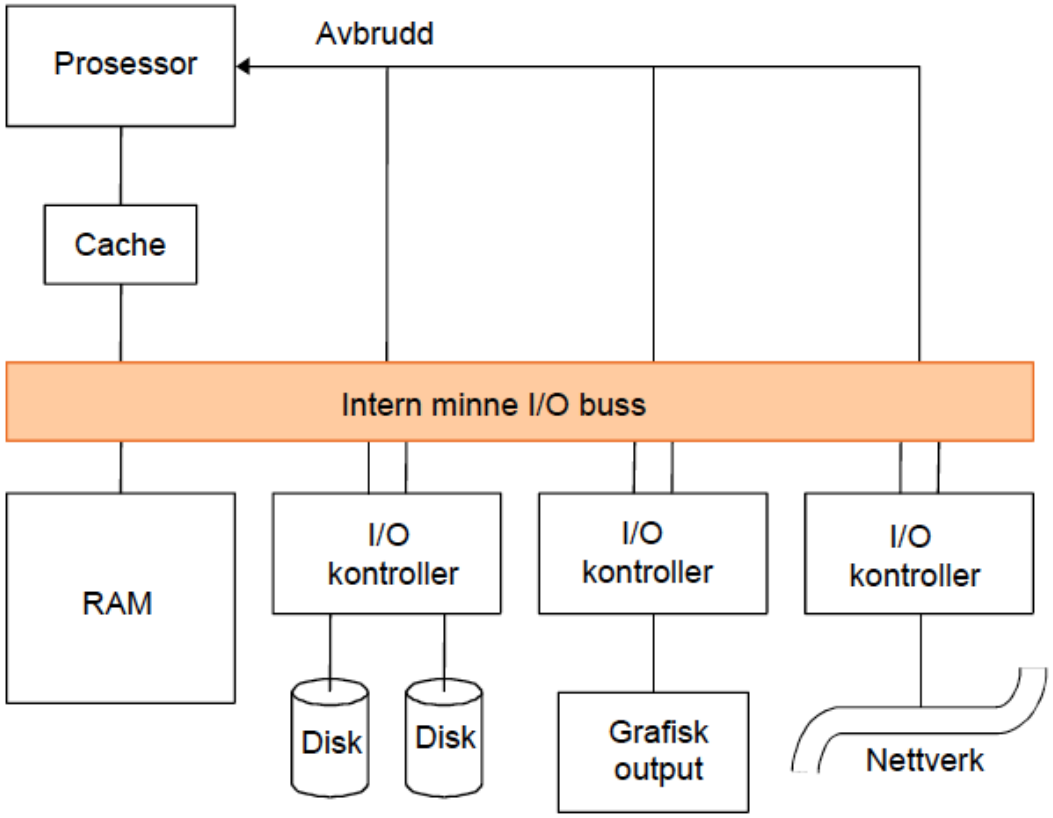
- Ytelsen til I/O-systemer avhenger av flere faktorer:
  - Prosessoren
  - Hukommelseshierarkiet
  - busen(e) som kobler sammen maskinen
  - Kontrollenheter for I/O og enhetene som er tilknyttet bus´en
  - Hastigheten til operativsystemet
  - Programvarens bruk av I/O

- To vanlige målestokker for ytelse til I/O er:
  - **Throughput**: Båndbredde eller gjennomstrømning av data per tidsenhet.
  - **Responstid**: Forsinkelse fra start til svar

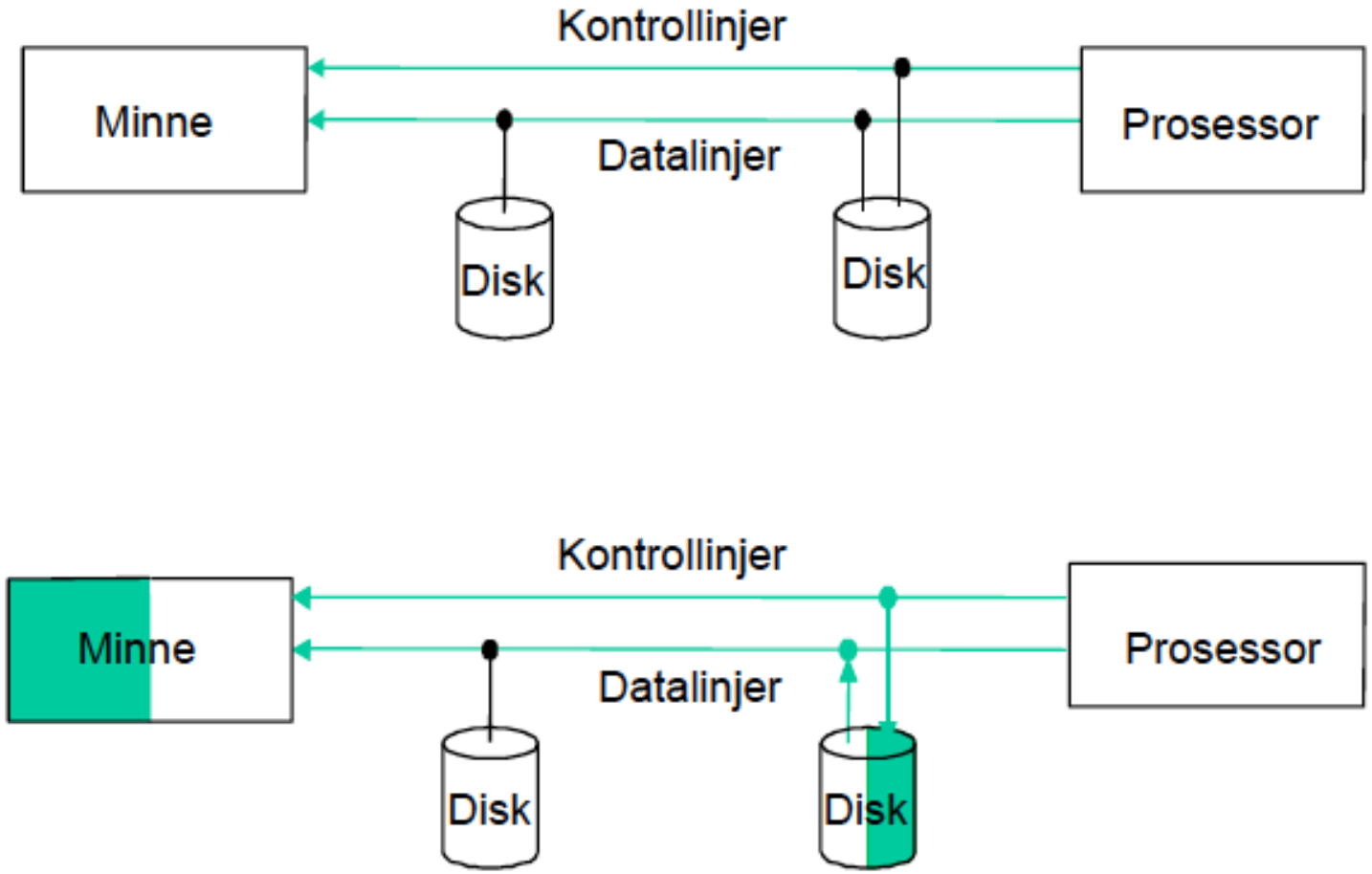
- En maskin har som regel flere uavhengige bus´er som er spesialiserte.

- En **bus** knytter sammen mange ulike enheter
- bus´en er ofte en flaskehals i systemet, fordi mange enheter konkurrerer om å få bruke den
- Siden de ulike enhetene deler samme fysiske bus, trengs regler for hvilke enhet som kan bruke bus´en til hvilket tidspunkt.
- En **protokoll** spesifiserer kjørereglene som gjelder for bruk av bus´en
- Det finnes mange ulike protokoller for ulike bus-typer:
  - ISA
  - PCI
  - TCP/IP
  - ATM
  - +++++

Intern kommunikasjon



- En bus består av **datalinjer** og **kontrollinjer**
- Kontrollinjene regulerer bruken av bus´en, spesifiserer hva bus´en inneholder, synkroniserer overføring osv
- Datalinjene inneholder de data som sendes over busen, både adresser og faktiske data.


Kontrollinjer
Minne
Datalinjer
Prosessor
Disk
Disk
Kontrollinjer
Minne
Datalinjer
Prosessor
Disk
Disk

- Bus´er er enten **synkrone** eller **asynkrone**
  - **Synkron**: Endringer på bus´en skjer etter en fast protokoll, relativt til et klokkesignal i kontrollinjene
    - » Raskere enn asynkrone
    - » Knytter sammen enheter med samme klokkehastighet
    - » Enhetene må ligge fysiskt nær hverandre
  - **Asynkrone**: Ingen klokkesignal blant kontrollinjene. Overføring av data skjer etter regler avtalt mellom enhetene ("handshaking")
    - » Knytter sammen enheter med ulik hastighet
    - » Gir færre begrensninger i bus-lengde
    - » Mer komplisert protokoll for synkronisering

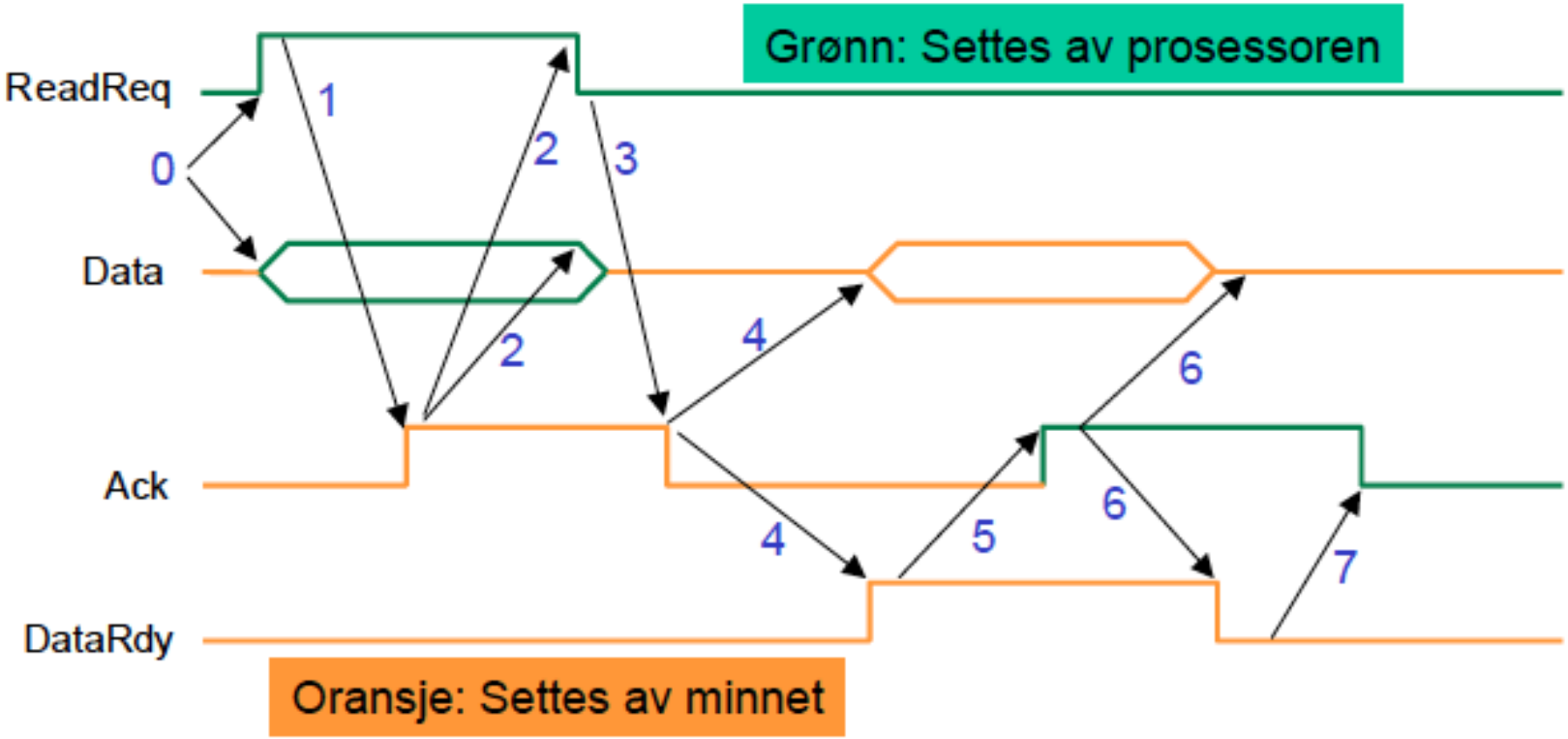
Asynkrone bus´er og handshaking

- Handshaking brukes for å koordinere transmisjon av data mellom sender og mottager

- Gitt et enkelt system med tre kontrollinjer:
  1) **ReadReq**: Brukes for å indikere en forespørsel om lesing fra minne. Adressen legges på datalinjene samtidig.
  2) **DataRdy**: Indikerer at data er klare på datalinjene
  3) **Ack**: Brukes for å bekrefte at ReadReq eller DataRdy er mottat fra den andre enheten

- De tre kontrollinjene brukes for å utveksle informasjon om hvor langt de to enhetene har kommet.

Lesing fra minne til prosessor



(0) Prosessoren setter ReadReq = ‘1’, og legger adressen ut på datalinjen.

① Minne ser ReadReq = ‘1’, leser adressen og setter Ack = ‘1’ for å indikere at adressen er lest

② Prosessoren ser Ack = ‘1’ og setter ReadReq = ‘0’ og frigir datalinjen

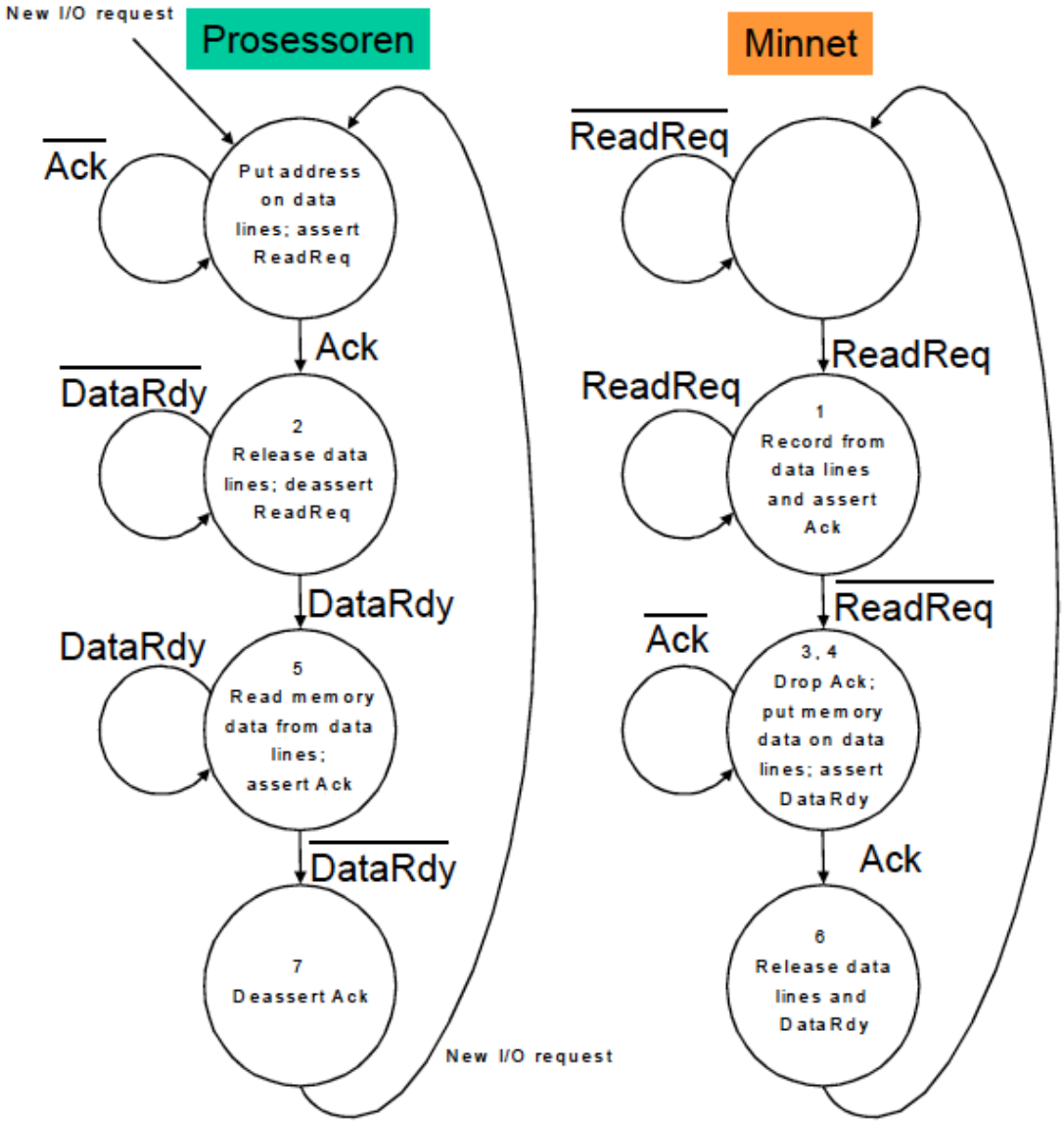
③ Minne ser ReadReq = ‘0’ og setter Ack = ‘0’ for å bekrefte at ReadReq-signalet er mottatt

④ Når minet har data klart for overføring, plasseres data på datalinjen, minne setter DataRdy = ‘1’ for å indikere at det er gyldige data på bus´en

⑤ Prosessoren ser at DataRdy = ‘1’, leser data fra bus´en og indikerer at den har lest ferdig ved å sette Ack = ‘1’

⑥ Minne ser at Ack = ‘1’, setter DataRdy = ‘0’ og frigir datalinjene

⑦ Prosessoren ser at DataRdy = ‘0’ og setter Ack =‘0’ for å indikere at transmisjone er ferdig.


New I/O request
Prosessoren
Minnet
Ack
Put address on data lines; assert ReadReq
ReadReq
ReadReq
Ack
DataRdy
2 Release data lines; deassert ReadReq
ReadReq
1 Record from data lines and assert Ack
DataRdy
DataRdy
5 Read memory data from data lines; assert Ack
Ack
3, 4 Drop Ack; put memory data on data lines; assert DataRdy
ReadReq
DataRdy
7 Deassert Ack
Ack
6 Release data lines and DataRdy
New I/O request

Ekstern kommunikasjon

- Datamaskiner er ofte knyttet sammen med andre maskiner i nettverk av ulik størrelse:
  - » LAN (Local Area Network): Innenfor begrenset område, feks innenfor en bygning
  - » WAN (Wide Area Network): Innenfor større geografisk område, feks, hele blindern
  - » Internet: Hele verden

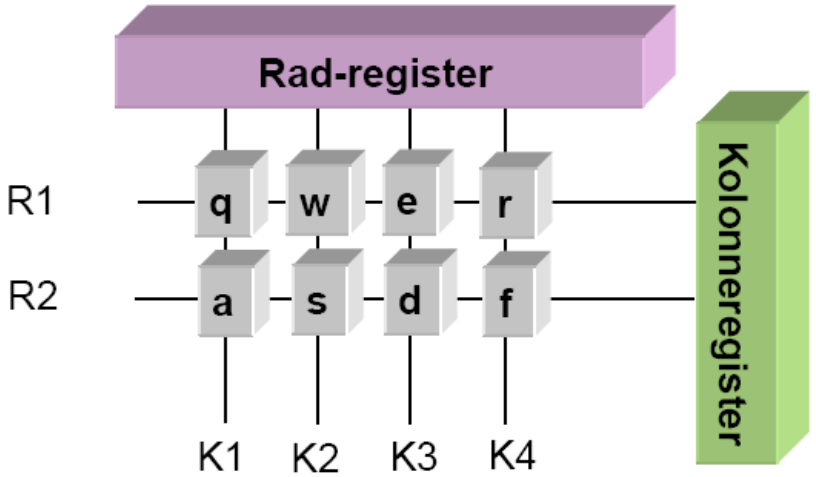
- Slike sammenkoblinger skiller seg fra den interne sammenkoblingen i datamaskiner på flere måter:
  - Knytter sammen utstyr fra ulike produsenter med ulike egenskaper
  - Store forskjeller i hastighet
  - Må tåle feil ved utstyr som er tilkoblet
  - Må være skalerbart, dvs lett å koblie til flere enheter.

Eksterne hendelser

- I noen tilfeller krever en ekstern hendelse eller begivenhet at prosessoren foretar seg noe bestemt (dvs. eksekverer en bestemt subrutine eller funksjon).
- For at prosessoren skal finne ut at noe har skjedd kreves det signalering mellom den ytre enheten og prosessoren.

Eksempel: *Avlesning av tastetrykk på tastatur*

- Når en tast trykkes ned, blir det kontakt mellom en rad og en kolonne. Trykkes tasten merket 'e' ned, blir det kontakt mellom R1 og K3. Dette registreres i Rad og Kolonneregisteret



Eksterne hendelser

Prosessoren kan lese innholdet av rad- og kolonnergisteret for å finne ut hvilken tast som er trykket ned.

Problem: Hvordan finne ut når en tast er trykket ned?

Dette kan løses på to måter:

1. Polling
2. Avbrudd

Polling

Prosessoren kan med jevne mellomrom lese av innholdet av Rad- og Kolonne-registrene (f.eks hvert 10. millisekund) og sjekke om det er en *endring* fra forrige gang.

Fordel: Enkelt å implementere (gå i evig løkke og les av registrene og sjekk mot forrige verdi).

Ulempe: Prosessoren får ikke gjort så mye annet enn å sjekke disse registrene hele tiden!

Avbrudd

I stedet for å sjekke jevnlig, kan tastaturet selv si fra at en tast er trykket ned.
Prosessoren finner ut hvilken tast ved å sammenligne gammelt og nytt innhold i Rad- og Kolonneregistrene

Fordel: Prosessoren kan løse andre oppgaver enn bare å sitte og vente på at en tast skal trykkes ned

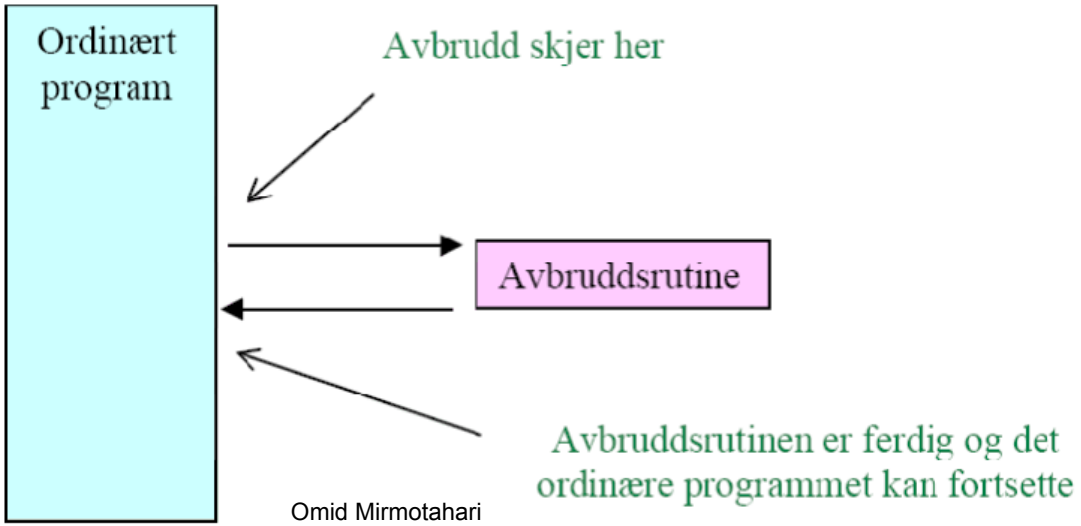
Ulempe: Det kreves mer av hardware; må ha egne signaler inn til prosessoren som kan brukes til f.eks. å si fra et en tast er trykket ned.

Avbrudd er en generell mekanisme som finnes i alle maskiner og brukes til bl.a.

- Signalisere at en ekstern hendelse har skjedd
- Markere avslutningen på en operasjon
- Allokere CPU-tid (context switching)
- Signalisere at en uventet eller ulovlig situasjon har oppstått (exception)

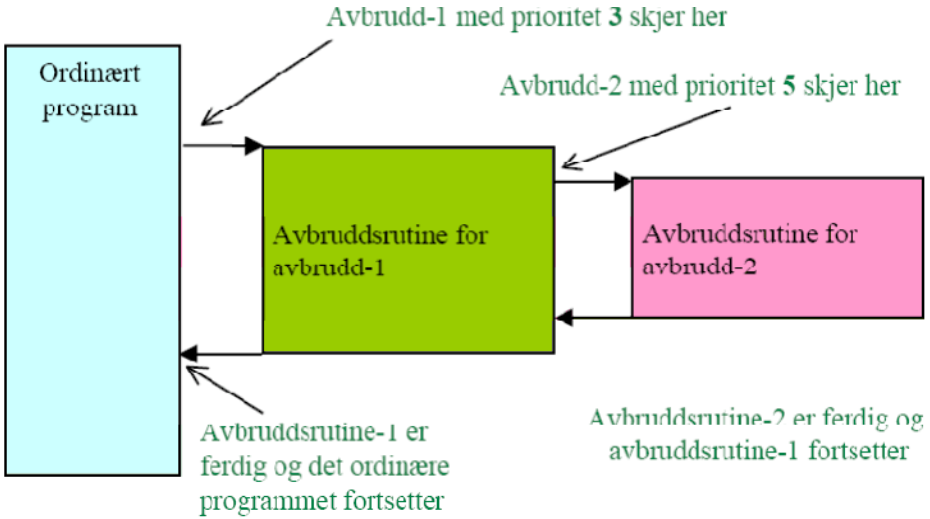
Avbrudd – Hva skjer?

- Prosessoren avslutter den instruksjonen den holder på med å eksekvere
- Alle registre som er i bruk må lagres unna
- Avhengig av hvilken kilde som genererte avbruddet vil det bli startet opp en avbruddsrutine som prosesserer avbruddet.
- Når avbruddsrutinen er ferdig, gjenopprettes registrene som ble lagret unna, og prosessoren fortsetter å eksekvere det programmet den kjørte før avbruddet skjedde.



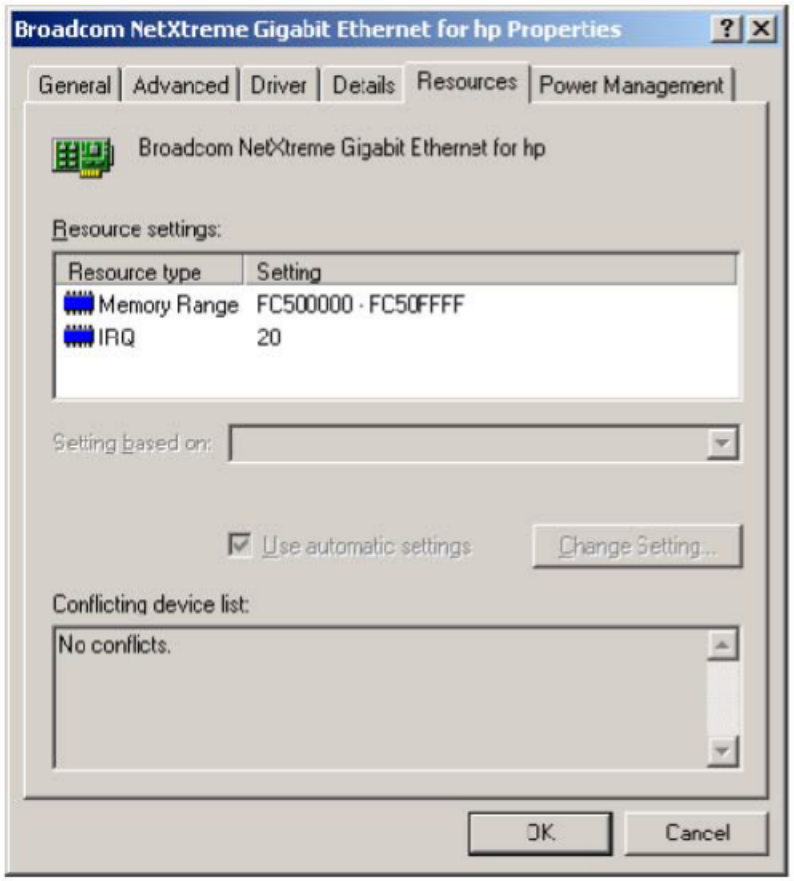
Avbrudd – Hva skjer?

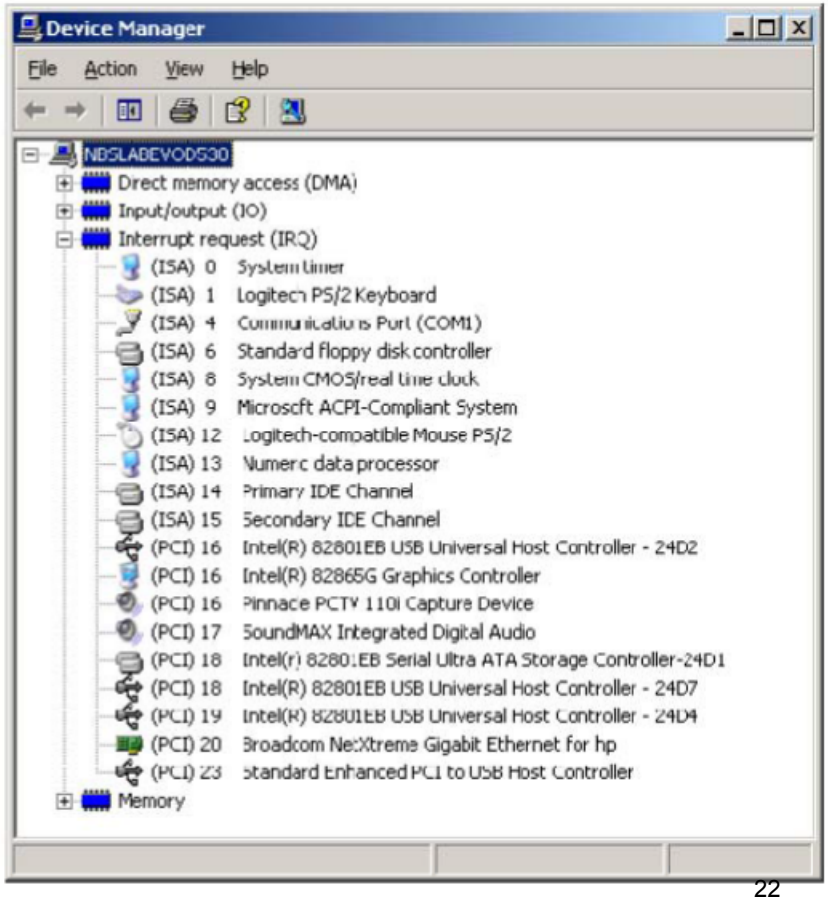
- Fordi hendelser og begivenheter kan ha varierende betydning og viktighet, tilbyr prosessorer flere avbruddsnivåer med ulik prioritet.
- Et avbrudd med høy prioritet kan avbryte behandlingen av (dvs. avbruddsrutinen til) et avbrudd av lavere prioritet.
- Hvis avbrudd fra to ulike kilder har samme prioritet behandles de ferdig i den rekkefølge de kom, og informasjon om andre avbrudd (med samme eller lavere prioritet) blir lagt i en kø



Avbrudd i Windows

- Interrupt Request (IRQ) på Windows XP. Lavere tall = høyere prioritet





Bruk av avbrudd

Signalisering av ekstern hendelse

- I kontrollsystemer overvåker og styrer datamaskiner temperatur, trykk, strålingsnivå etc. Avbrudd kan brukes til å signalisere at et kritisk nivå eller en grense er nådd som krever spesiell handling, f.eks iverksetting av alarm
- Prosessering av tastetrykk er også eksempel på slike hendelser som krever spesiell prosessering (f.eks Ctrl-C som betyr at et program skal avsluttes)

Synkroniserings-/avslutningssignal

- Avbrudd kan brukes av f.eks printere for å be en prosessor om å få sendt over mer data hvis et internt buffer er tomt.
- Avbrudd kan generelt brukes til flytkontroll for å signalisere start/stopp av transaksjoner, overføringer osv. (“send mer data”, “stopp å sende data”…)

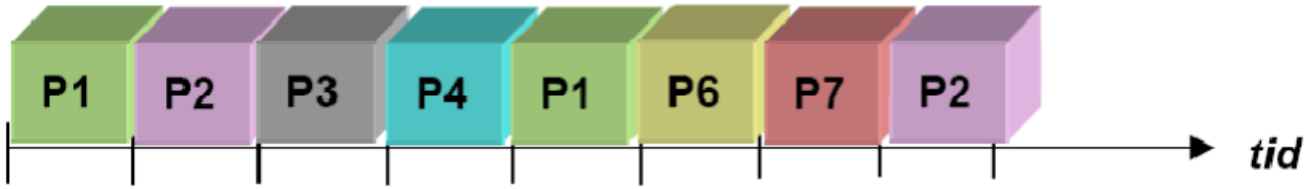
Bruk av avbrudd

Signalisering av unormal hendelse

- Dette er en viktig mekanisme og brukes både av hardware og software for å signalisere at en gitt unormal hendelse har inntruffet.
- Hvis avbruddet genereres av software kalles det “execption” (unntak) og brukes enten for å gi beskjed om en ulovlig operasjon som (f.eks divisjon med null), eller for å angi at en instruksjon må behandles av en ekstern hardware-enhet (f.eks egen flyttalls-prosessor)

Tidsdeling i operativsystemer

- Operativsystemer simulerer parallellitet ved å dele prosessor-tiden opp i små tidsintervall, og lar hver prosess får bruke prosessoren i minst ett tidsintervall.



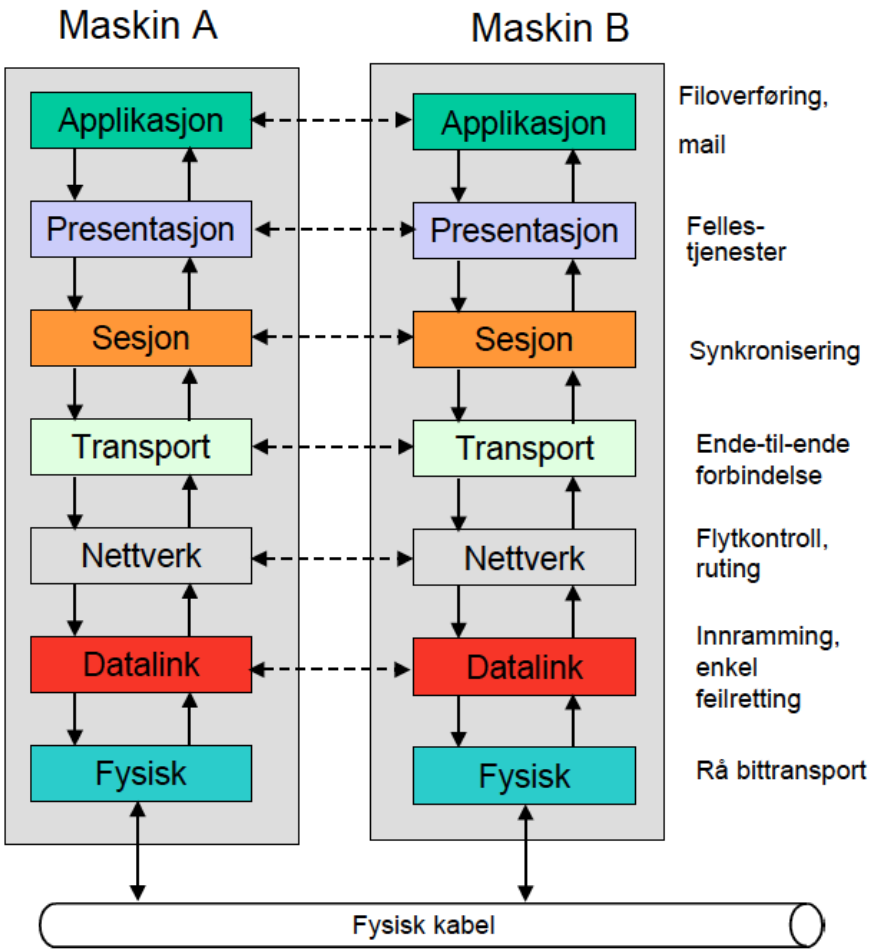
Implementasjon av tidsdeling

Kan implementeres slik:

- Med faste intervaller genereres et avbrudd som signaliserer at operativsystemet skal suspendere prosessen som kjører i øyeblikket. Registre, peker til stakkområdet og statusregistre som prosessen brukte blir lagret
- Operativsystemet tar over kontrollen og plukker ut hvilken prosess som skal få kjøre (skedulering).
- Registre, stakkområde osv. til prosessen som skal startes, lastes inn i CPU'en av operativsystemet.
- Operativsystemet gir kontrollen til neste prosess som kan fortsette å eksekveres

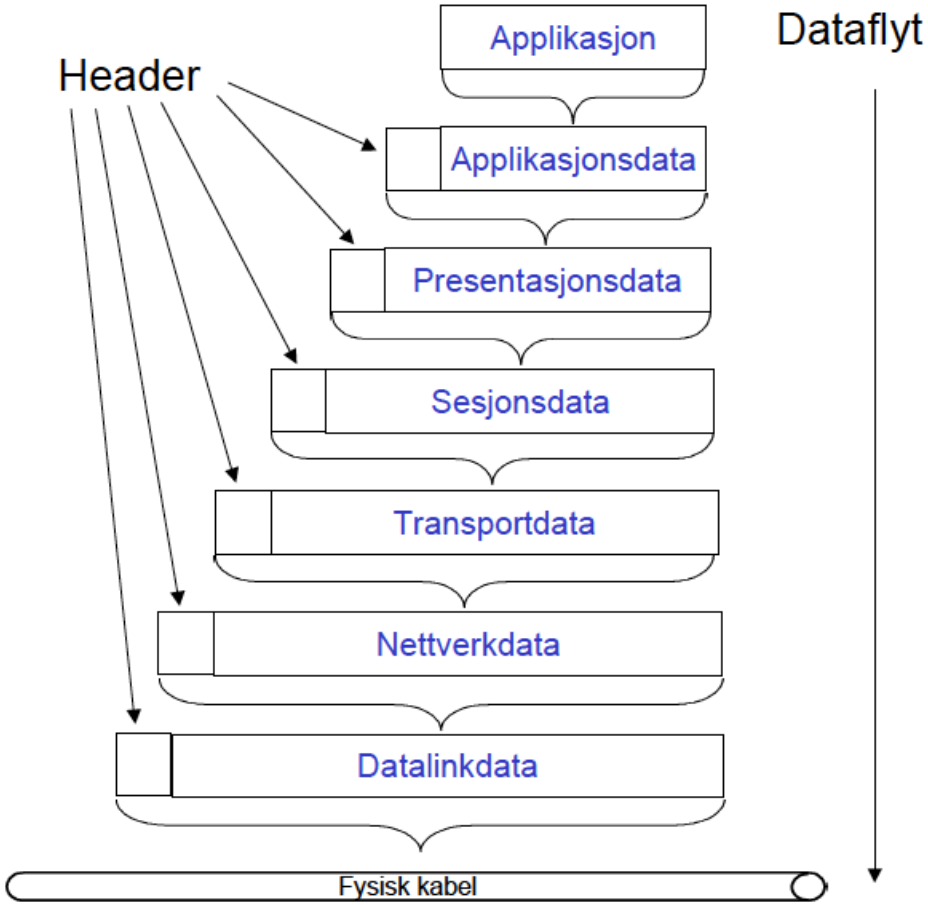
På samme måte som avbrudd fra ulike kilder kan ha ulik prioritet, vil også prosesser ha ulik prioitet. Typisk vil operativsystemet ha høyere prioritet og få tilgang til CPU'en før et brukerprogram

OSI - modellen



- Data grupperes sammen i enheter som kalles pakker eller rammer. Hver pakke består av **header** og **data.**
- Header´en inneholder informasjon om pakken og innholdet som trengs for å behandle den riktig
- Kommunikasjon skjer gjennom **protokollstakkene** på hver maskin
- Logisk sett kommuniserer lag X på maksin A med lag X på maskin B.
- Fysisk sett kommuniserer lag X på maskin A med lag X-1 og lag X+1 på maskin A og tilsvarende på maskin B.
- Data sendes ovenfra og nedover til det fysiske laget og så motsatt vei på mottager-maskinen.

På vei ned gjennom protokollstakken legges på kontrollinformasjon for hvert lag:



På den andre maskinen fjernes headerene i motsatt rekkefølge.