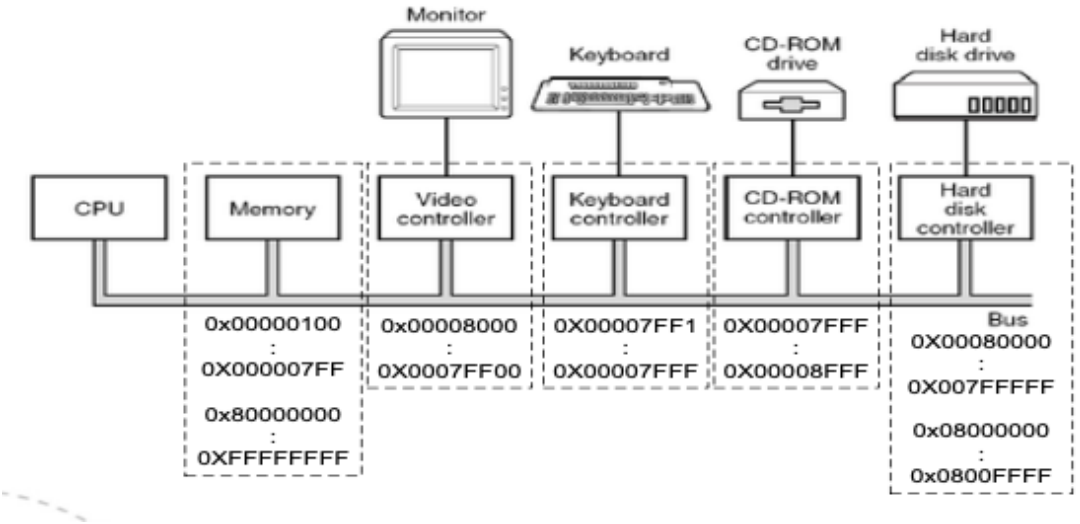
Week 4

Minnesystem



Alt er minnemaped i systemet. Vi får tak i data ved å peke på dem. Vi kan skrive til eller fra.
Vi har en "Main Memory". Dette minnet er tilgjengelig for program og som operativsystemet også bruker

Latency hiding techniques

- Teknikker for å "skjule" forsinkelse ved minneaksess
- Program "ser" gjennomsnittlig lav aksesstid
- Minnegenerasjonene henger etter for prosessor
- Processor venter 1000 runder før den kan skrive til minnet. Vi må gjøre noe

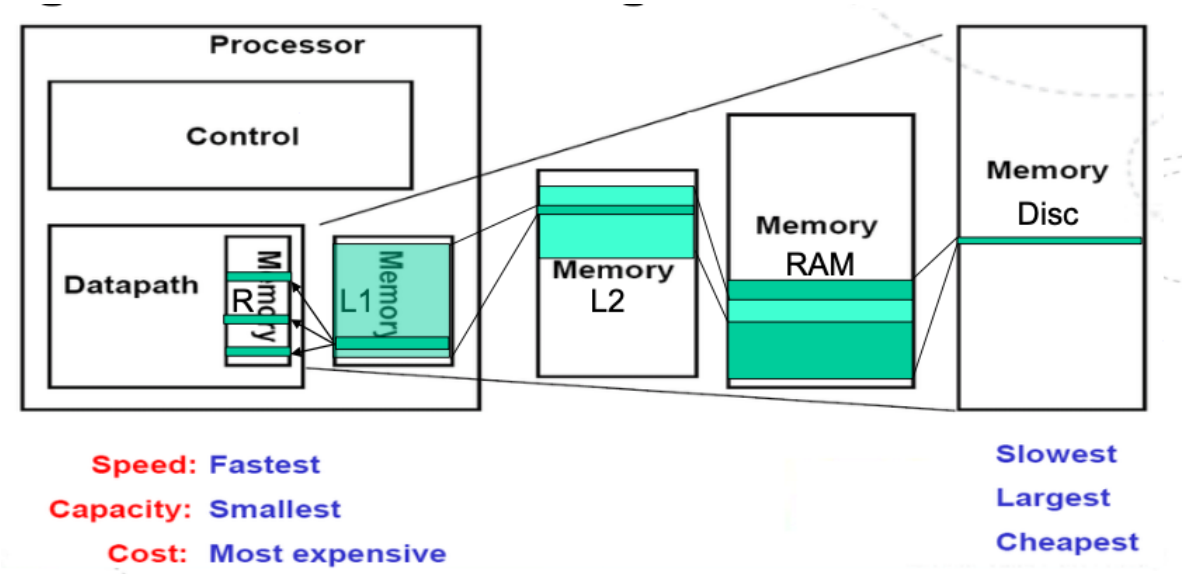
Lagerhierarki

- Ofte bygd opp av:
  - Fysisk plassering
  - Aksesstid
  - Størrelse

- Hierarkiet er som følger:
  - Register (som er i processoren)
  - Cache
  - Main memory
  - Magnetic disk

- Tape/Optical disk

Avbilding



En bit av programmet ligger som en liten del av discen. Dette blir kopiert til RAMen. RAM er tregt så vi tar en bit av instruksjonene og kopierer det videre til et hurtigbuffer. Dette igjen kan vi kopiere en bit av til Level 1 (L1) cache. Her kan vi f.eks kopiere tre variabler og få det i de nødvendige registre.

Vi skiller ofte instruksjoner og data i L1 slik at vi kan lese instruksjoner samtidig som man kan oppdatere data i registrene

Vi kan late som at aksesstid blir mye bedre enn hva det egentlig er basert på dette.

Register

- Register i processor (PC, IR, osv)
- All data må inn i register

Hurtigbuffet ("cache")

- Lokalitet
  - Rom (data nær (minnelokasjon), sannsynlig aksess)
  - Tid (data brukt "før", sannsynlig aksess)
- Aksesstid
  - Gjennomsnittlig
  - **Mean access time = c +(1-h) * m**
    - c: cache access time
    - m: main memory access time

        - H: hit ratio
- Øker ytelsen
    - Latency hiding

CPU organisering, yting

En enkelt kjerne har flatet ut mpt ytelse, men vi legger til flere kjerner i CPUen for å fremheve totale ytelsen. Dette er altså parallell ytelse. Å kjøre ting parallelt er vanskelig.

En annen løsning på dette er å ofre generalitet. CPUen er altså spesialisert i enkelte oppgaver.

Parallelitet

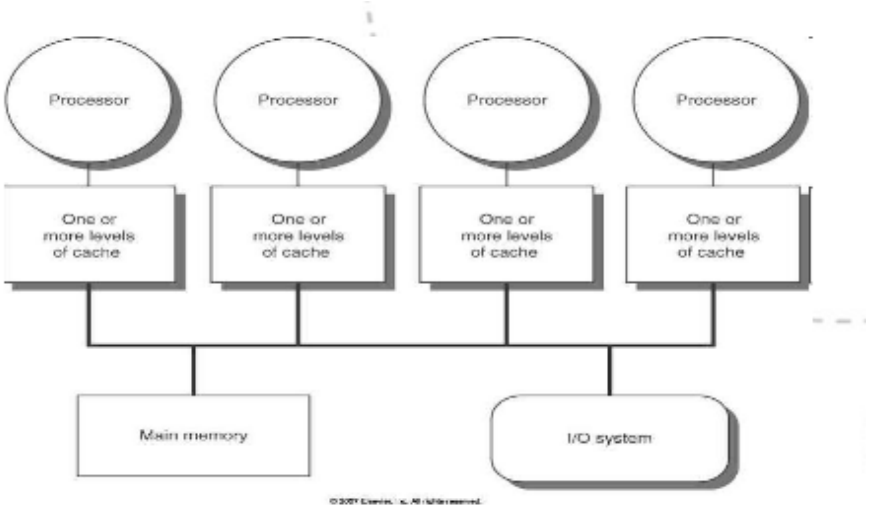
- Essensielt for å øke ytelse
- To typer
    - Instruksjonsnivåparrallelitet (ILP)
        - Flere instruksjoner uføres samtidig (1 prosessor)
        - Samlebånd (pipelining=
            - Eks: unødvendig at ALU er ledig mens CPU leser instruksjon
        - Superskalaritet: Duplisering av CPU-komponenter
    - Prosesspornivåparallelitet
        - En prosessor er bra, flere er bedre
        - SIMD enkle prosessorer, lokal kommunikasjon
        - MIMD mer komplisert prosessorer, ingen - til global kommunikasjon

Processor (CPU) ILP

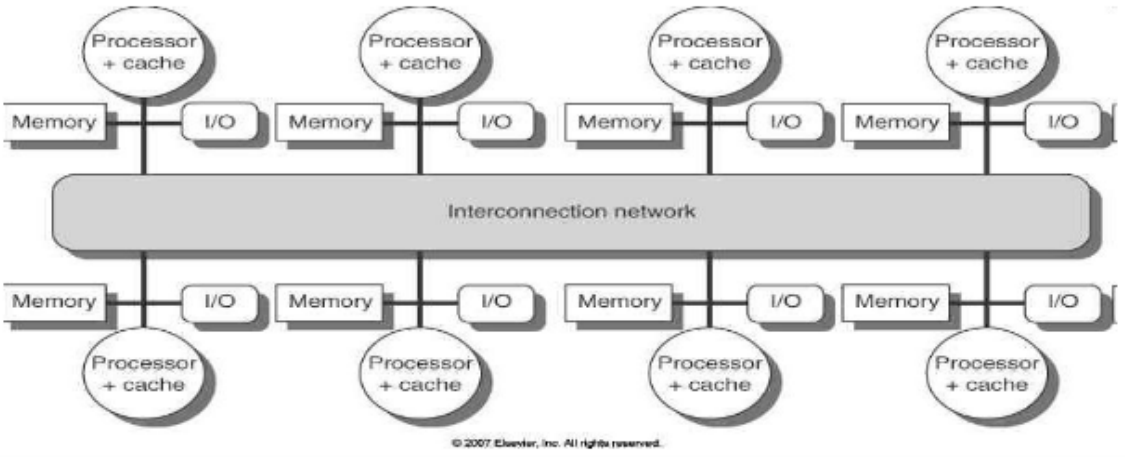
- Organisering som er fokus

System (processor-level parallelism)

- System design
    - organisering av alle "fysiske maskiner"
- datamaskin arkitektur
    - Organisering av resurser for samtidighet

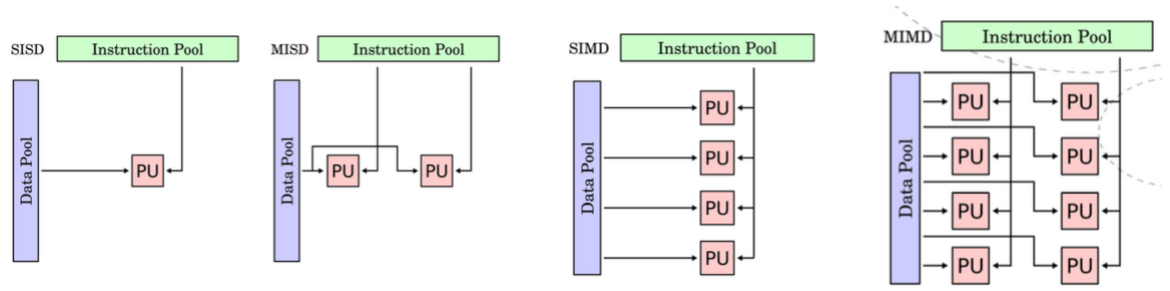


Her er prosessorene hver for seg og alle har sin egen cache. Dersom prosessorene ønsker å snakke med hverandre så må de skrive dette til main memory som da vil bli lest av en annen prosessor. Bussen og minnet er altså en felles ressurs. Dette vil kanskje bli en flaskehals.



Dersom det kreves mye kommunikasjon mellom prosessorene så vil denne kommunikasjonen bli gjort via et spesialisert nettverk.

Parallellitet typer, definisjon



Helt til venstre ser vi en SISD (Single Instruction Single Datastream). Den har et prosesseringselement.

MISD (Multipel Instruction Singel Datastream). Vi har flere prosesseringselementer. Hver av dem har tilgang til sine egne instruksjoner, men de jobber med de samme dataene.

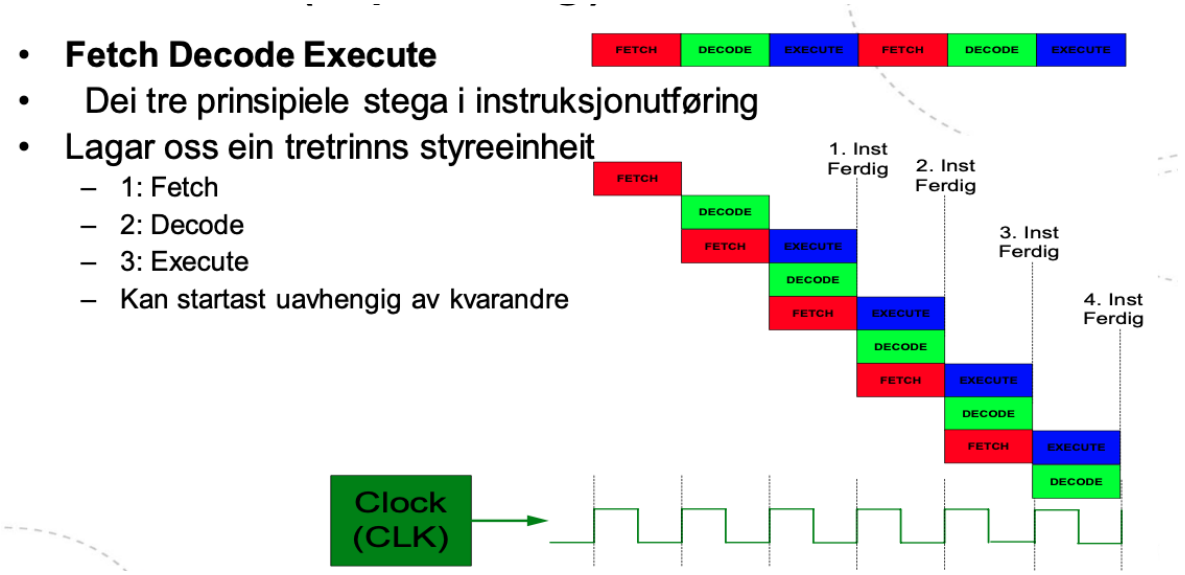
SIMD (Single Instruction Multiple Datamaskin) typisk GPU. Vi har mange prossesseringselement. De har samme instruksjoner, men har forskjellige dataset.

MINM Da kan vi få mange PU som kjører det samme programmet, men kan kjøre forskjellige deler av programmet.

Prosessor for å få høy ytelse

Samlebånd (pipelineing) (ILP)

- **Fetch Decode Execute**
- Dei tre prinsipiele stega i instruksjonutføring
- Lagar oss ein tretrinns styreeinheit
  - 1: Fetch
  - 2: Decode
  - 3: Execute
  - Kan startast uavhengig av kvarandre

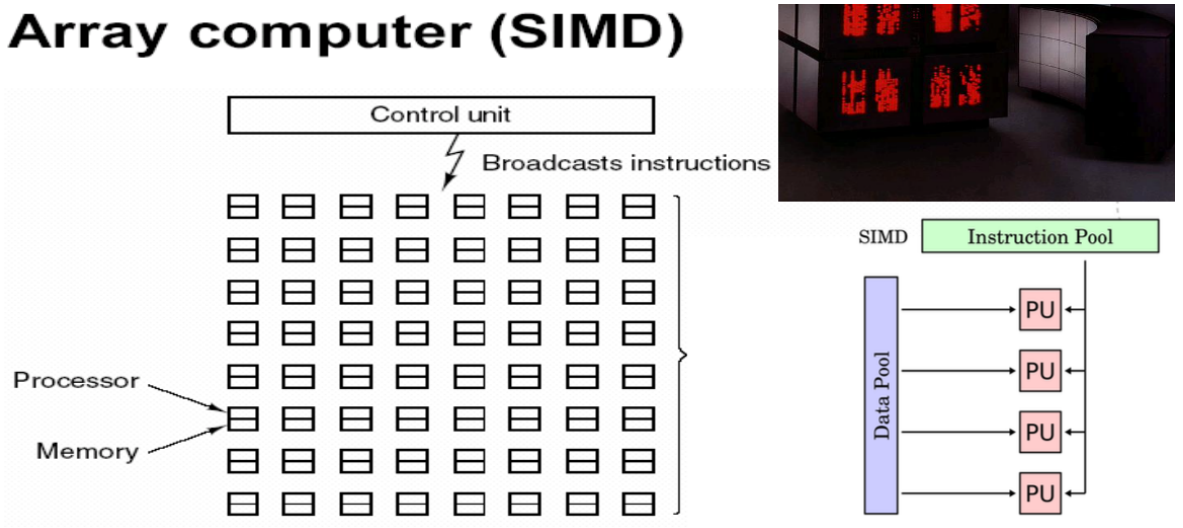


Vi kan dele opp instruksjoner i tre deler: fetch, decode og execute. Vi kan altså hente en instruksjon, når vi decoder denne så henter vi inn neste samtidig. Deretter når vi utfører instruksjonen så decoder vi forrige og henter neste samtidig. Dermed vil vi få en instruksjon ferdig for hvert klokkesyklus.

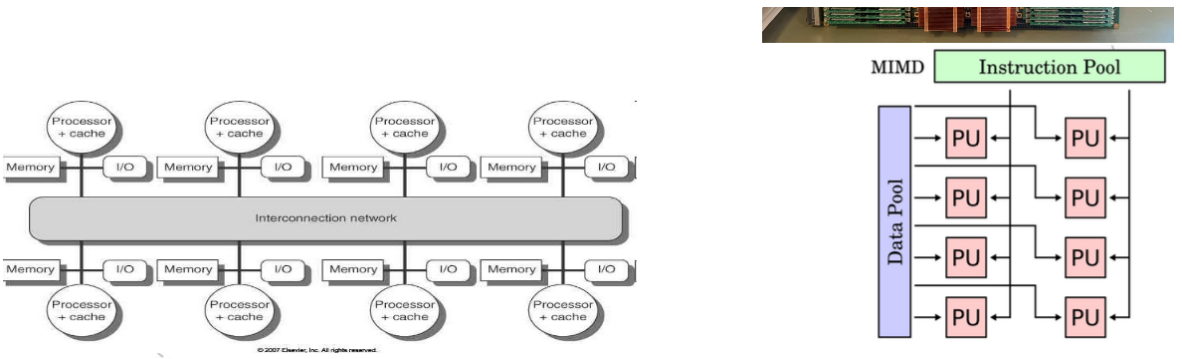
Vi kan gjøre dette raskere ved å:

- Få enklere trinn
  - Mindre logikk, korte vei
- Flere trinn
  - Kortere tid
- Høyere klokkefrekvens

Mer parallelitet



Alle prosessorene har litt minne. Har 10 000er prosessorer. All kommunikasjon var gjort kun til de nærmeste naboene.
Dette ligner veldig en GPU.



Dette er fra VILJE. Dette er en MIND maskin. Den har PU som har sine egne instruksjoner og egen data. Alle prosessorene er koblet sammen i et raskt nettverk for å kommunisere med hverandre.