Prosessoren i LMC(CPUen) vs. Vanlig prosessor i dag (x86-64):

Ting	x86-64	CPUen i LMC
Minne	- RAM, 2^64 celler med minne - 16 registere - har cache. - har støtte for virtuelt minne - har sekundærminne	- RAM, 100 celler med minne - 4 registere - har ikke sekundærminne
Eksekvering (kjøring)	- Pipeline, kan kjøre instruksjoner parallelt (raskt)	- Sekvensielt, instruksjoner kjøres etter hverandre (treigt)
Arkitektur	Von Neumann	Von Neumann
ALU	- ALUen kan utføre flere operasjoner enn den i LMC. Bla. matematiske og logiske operasjoner.	- Veldig simpel ALU. Bla. Mangler logikkoperasjoner. Kan bare utføre addisjon og subtraksjon.
Maskinkode	- Maskinkoden har 64 plasser - binær	- Maskinkoden er på 3 siffer fra 0-9 - ikke binær.
Instruksjonskoder	- I maskinkoden angir de første 6-10 bitene instruksjonkoden. (varierer mellom 6-10 bit avhengig av type prosessor)	- Første siffer i maskinkoden angir instruksjonskode

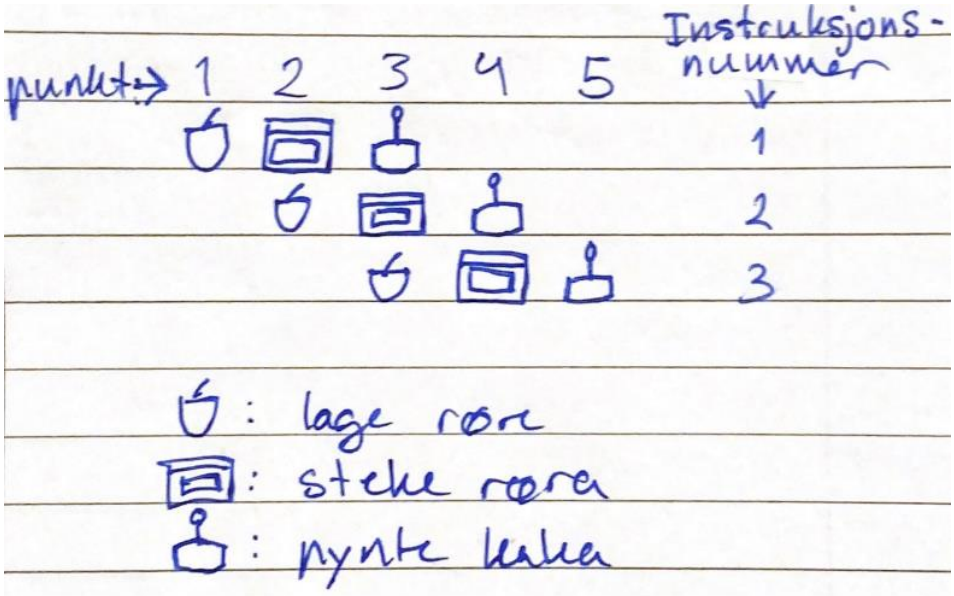
Pipeline:

- Både LMC og x86-64 har samme eksekvering: (når jeg snakker om LMC her mener jeg prosessoren til LMC, som er CPUen)

    1. Bruk verdien i programdelen som adresse til minnet og hent neste instruksjon der.
        - Øk samtidig programtelleren med 1 (Dette gjøres i ALUen)
    2. Splitt instruksjonen og legg delene i instruksjonsregisteret og adresseregisteret.
    3. Utfør det som instruksjonsregisteret angir.
    4. Gjenta fra punkt 1

- Men x86-64 har funksjonalitet for å eksekvere/kjøre flere instruksjoner samtidig. Kjøre instruksjoner parallelt.
    - Det har ikke LMC. Om man vil kjøre flere programmer, må man kjøre dem etter hverandre sekvensielt
- Dette gjør x86-64 mye mer effektiv og raskere enn LMC.
- Pipeline metafor:
    - Vi har fått instruksjon om å lage 3 kaker:
        - For å kunne lage en kake må vi lage røra, steke kaka og pynte kaka.
        - Og i vår fantasiverden tar dette i tid:
        - Lage røra = 5 min
        - Steke kaka = 5 min
        - Pynte kaka = 10 min
    - I LMC må vi:
        - Lage røre 1, steke røre 1 og pynte kake 1. 20 min
        - Lage røre 2, steke røre 2 og pynte kake 2. 20 min

        - Lage røre 3, steke røre 3 og pynte kake 3. 20 min
        - Så er vi ferdige. 60 min
    - I x86-64 kan vi:
        - Lage røre 1. 5 min
        - Steke røre 1, mens røre 1 steker kan vi lage røre 2. 5 min
        - Pynte kake 1, mens vi pynter kaka kan vi steke røre 2 og lage røre 3. 10 min
        - Pynte kake 2, mens vi pynter kaka kan vi steke røre 3. 10 min
        - Pynte kake 3. 10 min
        - Så er vi ferdige. 40 min



- Pipeline er mye raskere!
- Men ved pipeline kan vi støte på en data **hazard** som skjer vis data som bearbeides er avhengig av hverandre, slik at den første dataen ikke er bearbeidet ferdig før den skal brukes igjen. En strukturell hazard er når mer enn en instruks prøver å bruke en ressurs.
- **Basic five stage pipeline i x86-64:**
    - Her ser man forskjellige instruksjoner som eksekveres i pipeline.
    - Clock cycle(klokkesyklus) = punkt (Mer om klokkesykluser senere)
    - Instr. No. = instuksjonsnummer

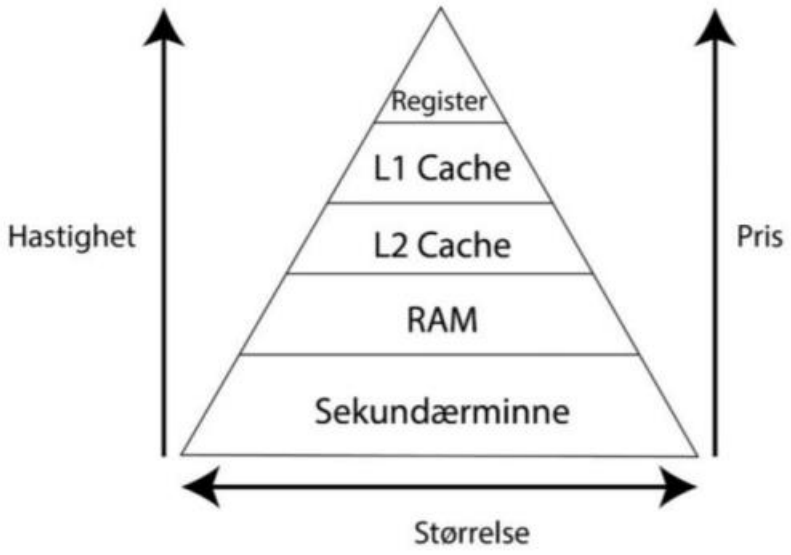
Basic five-stage pipeline

Instr. No. \ Clock cycle	1	2	3	4	5	6	7
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX

(IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back).

In the fourth clock cycle (the green column), the earliest instruction is in MEM stage, and the latest instruction has not yet entered the pipeline.

Minne:

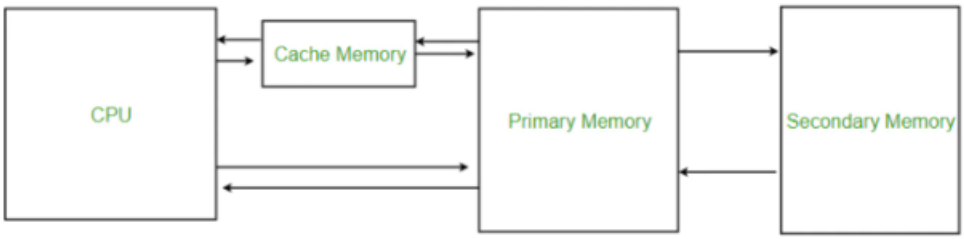


Registere:

- Både i LMC og x86-64
- Mini minne for å lagre instruksjonen, minneadresse (pasifisert av instruksjonen), lagring for en bit med data, osv.
    - I LMC: 4 registere: "Instruction register", "adress register", "program counter ", "accumulator"
- I de fleste moderne datamaskinarkitekturer følger man prinsippet om at data/instruksjoner leses fra minnet og inn i registere før bruk, og deretter lagre resultat tilbake til minnet fra et register.
- Registre er veldig raskt for CPUen å nå. Ligger jo inni CPUen. Veldig dyre.

Cache:

- Mellomlager mellom cpuen(prosessoren) og Minne(RAM).
- Er for å gjøre prosessoren raskere.
    - I LMC når vi henter data og instruksjoner heter vi det en og en fra RAM. Dette bruker mye tid
    - Da blir løsningen å sette litt ekstra minne nær CPUen, mellom CPUen og Minnet(RAM)
    - Dette heter cache
    - Når vi da skal for eksempel utføre noen beregninger, hentes det ikke bare en instruksjon fra Minnet(RAM), men vi henter alle instruksjonene og dataene fra Minnet(RAM) og legger det i cache.



RAM:

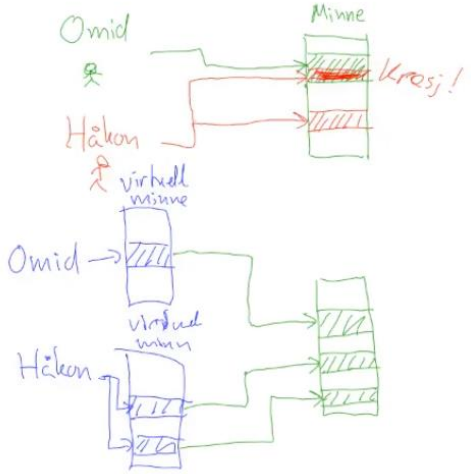
- Radom access memory
- I x86-64 har RAM 2^64 celler med minne. MEN i realiteten har man (og trenger ikke) ikke så mange celler med minne.
- Ting lagret i RAM forsvinner når vi skrur av datamaskinen.

Sekundærminne:

- Hard disk drive (HDD) og Solid State Drive (SDD) er eksempler på sekundærminne.
- På for eksempel harddisken på pcen, lagres hele programmer og data, som ikke forsvinner om pcen skrus av, i motsetning til RAM
- Har stor kapasitet til lagring.

Virtuelt minne:

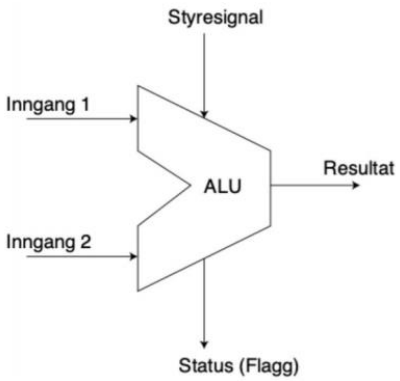
- Ikke et eget «minne-sted»
- To brukere kan ha hver sin virtuelle minneblokk. De to kan lagre noe på samme plass i hver sin virtuelle minneblokk. Men når disse to brukerne skal lagre dette fysisk på for eksempel en PC, så oversetter operativsystemet den virtuelle minneadressen(der de har lagret noe begge to) til to forskjellige fysiske minneadresser. Dette gjør at når vi skal lagre noe fysisk på for eksempel PCen forstyrrer ingen minneblokker hverandre.



- (Fra IN2110 Informasjonsikkerhet)(ikke IN1020 pensum, bare her for forståelse). X86-64 har støtte for virtuelt minne; for overføring av data mellom Minnet(RAM) og sekundærminne. Virtuelle minneadresser oversettes av operativsystemet til fysiske minneadresser. Dette gjør at om vi kjører en prosess som inneholder skadevare(for eksempel virus), har den ikke har tilgang til fysiske adresser for andre data og prosesser, bare sit eget virtuelle minneområde.

Forklaring på andre ting:

- **Maskinkode** er et sett med instruksjoner som blir utført direkte i datamaskinens prosessor (CPU).
- **ALU, Aritmetisk Logisk Enhet** (arithmetic logic unit, forkortet ALU) er en elektronisk krets som utfører aritmetiske og logiske operasjoner.
    - Styresignalet i ALU bestemmer operasjonen den skal utføre.
        - I LMC er dette instruksjonskoden.
    - Flagg indikerer statusen til prosessoren.
        - Et eksempel er «owerflow flag» for å si at vi fikk overflyt ved for eksempel en addering av to binærtall.
        - Mer om flagg: https://onlineclassnotes.com/purpose-of-using-flag-registers/



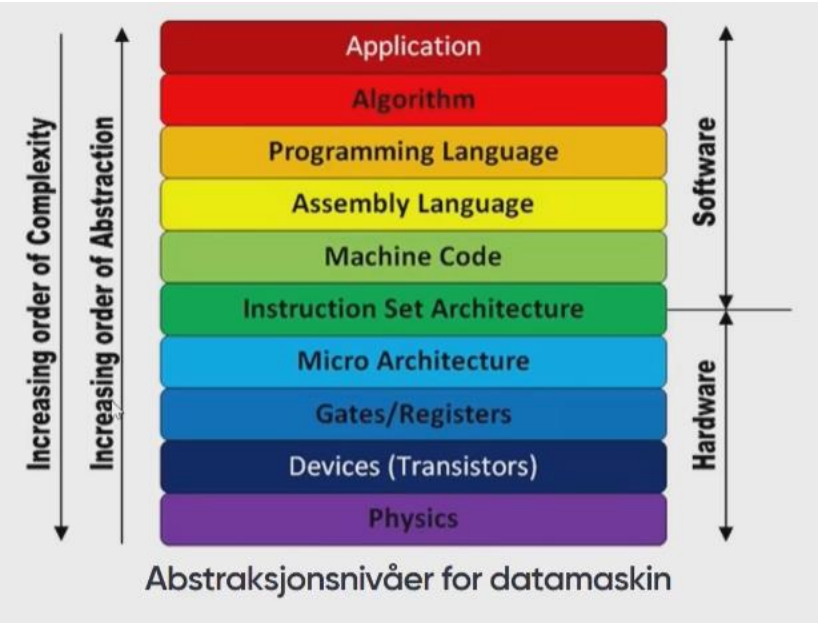
- Mikroarkitektur i abstraksjonsnivået

- **Logisk port**, grunnleggende byggestein i digital elektronikk, utfører logiske operasjoner med bits. Base for de logiske funksjonene i ALUen.
- **Transistorer**, en elektronisk komponent, en bryter for elektrisitet

Abstraksjon:

- Abstraksjon handler om å skjule det som «faktisk» ligger under. Man skjuler det som i virkeligheten skjer. Gjemmer kompleksitet.
- Bil: Får å vite hvordan du skal kjøre fra A til B, trenger du ikke å forstå hvordan motoren i bilen fungerer.

Abstraksjonsnivåer for datamaskiner



Abstraksjonsnivåer for datamaskiner: Eksempler / forklaring

- **Applikasjon**: Spill
- **Algoritmer**: metoder/funksjoner skrevet i programmeringsspråket
- **Programmeringsspråk**: for eksempel Python

- **Assemblerkode**: for eksempel LMC assemblerkode
- **Maskinkode**: Maskinkode er et sett med instruksjoner som blir utført direkte i datamaskinens prosessor (CPU). Binære tall.
- **Instruksjonssettarkitektur**: CPU. Tar in maskinkode og tolker instruksjonene.
- **Mikroarkitektur**: ALU. Utfører logiske operasjoner basert på instruksjon fra CPUen.
- **Porter/registere**: Porter er basen for logiske operasjoner i ALUen
- **Transistorer**: en elektronisk komponent, en bryter for elektrisitet
- **Fysikk**: strøm.

Applikasjoner	3D-Rendering program, Word, Zoom, etc. Operativsystem er det nederste «laget» av applikasjons-laget.	Software
Algoritmer	Hvordan et applikasjon skal kjøres elns.	
Programmeringsspråk	Java, Python, etc.	
Assembly kode	Simpleste koden som ikke består av kun binære tall. Varierer basert på hvilken maskin som brukes.	
Maskin kode	Binære tall som maskinen kan «forstå». Antall binære tall er avhengig av antall bits som er avhengig av maskin-arkitektur.	Fra SW Til HW
Instruction Set Architecture	En abstrakt modell av en datamaskin som beskriver hva maskinen består av. data typer, registre, etc.	
Micro Architecture	Kretsløpet i en datamaskin	Hardware
Gates/Registers	Enheter som utfører boolske funksjoner. AND, NOT, OR, XOR,	
Devices (Transistors)	Spenning, ikke spenning	
Fysikk	HvA eR sPeNiNg	