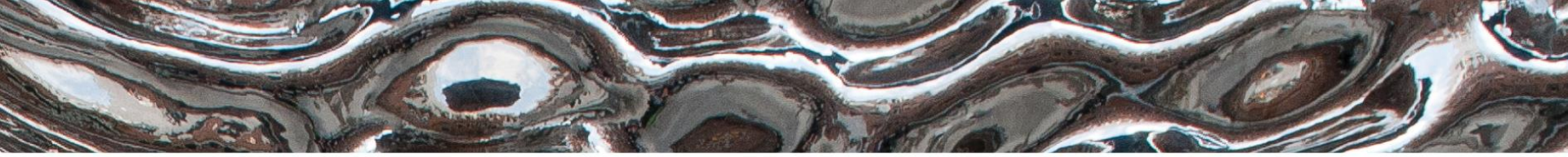


UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

Mer om prosessoren, og sikker programmering

I dag

- Generasjoner av prosessorer
- En moderne prosessor: x86-64
  - Likheter og forskjeller med LMC
- Sikker programmering

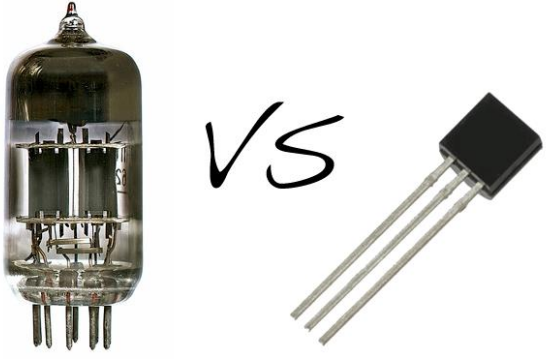
1. generasjon: Radiorør

År	Teknologi	Størrelse	Instr/sek	Pris (2019kr)
1945-60	Radiorør	$10m^3$	2000	60 mill

Maskinene var gigantiske og radiorørene måtte byttes ofte.

2. generasjon: Transistorer

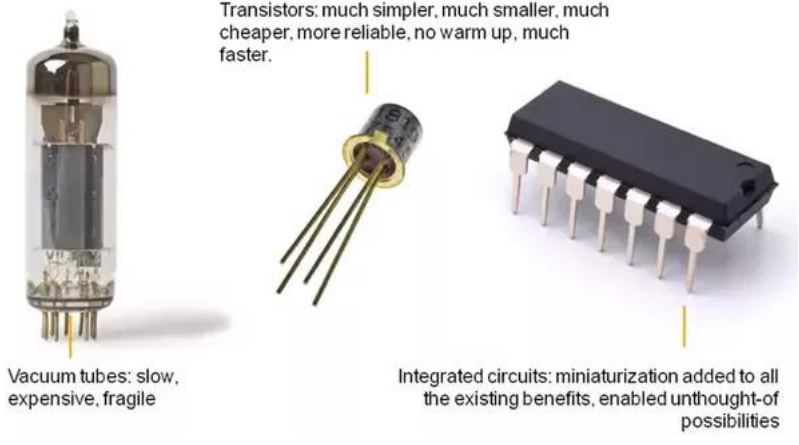
År	Teknologi	Størrelse	Instr/sek	Pris (2019kr)
1945-60	Radiorør	10m^3	2000	60 mill
1960-68	Transistorer	500dm^3	500 000	50 mill



Maskinene ble mindre, raskere og mye mer pålitelige

3. generasjon: Integrerte kretser

År	Teknologi	Størrelse	Instr/sek	Pris (2019kr)
1945-60	Radiorør	$10m^3$	2000	60 mill
1960-68	Transistorer	$500dm^3$	500 000	50 mill
1970-80	Integrerte kretser	$80dm^3$	300 000	200 000



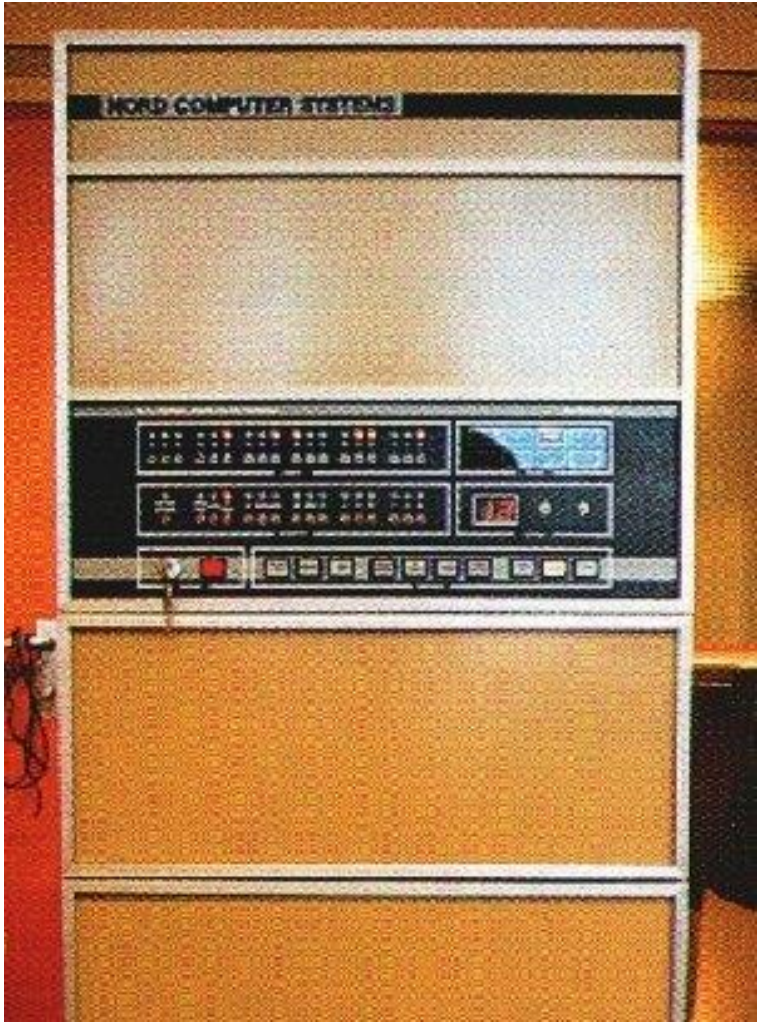
Maskinene ble mindre, og mye billigere

Norsk Data

Norsk datamaskinprodusent fra 1967

ND laget flere minimaskiner med stor suksess: Nord-1, Nord-10, Nord-100, etc. Det var på høyden i 1987 og med 4500 ansatte var det Norges nest største firma.

Selskapet kollapset i 1992.



4. generasjon: Mikroprosessorer

År	Teknologi	Størrelse	Instr/sek	Pris (2019kr)
1945-60	Radiorør	$10m^3$	2000	60 mill
1960-68	Transistorer	$500dm^3$	500 000	50 mill
1970-80	Integrerte kretser	$80dm^3$	300 000	200 000
1980-??	Mikroprosessorer	$0,1\text{-}20dm^3$	10^9	1-10 000

Mycron

Norsk datamaskin fra 1974.

Mycron laget noen av de første mikromaskinene i verden; de brukte Intel-CPUer og satset på operativsystemene CP/M og MP/M.

Selskapet sluttet å lage datamaskiner i 1990.

Tiki Data

Norsk datamaskinprodusent fra 1983.

Tiki Data satset på rimelige mikromaskiner, spesielt for norske skoler.

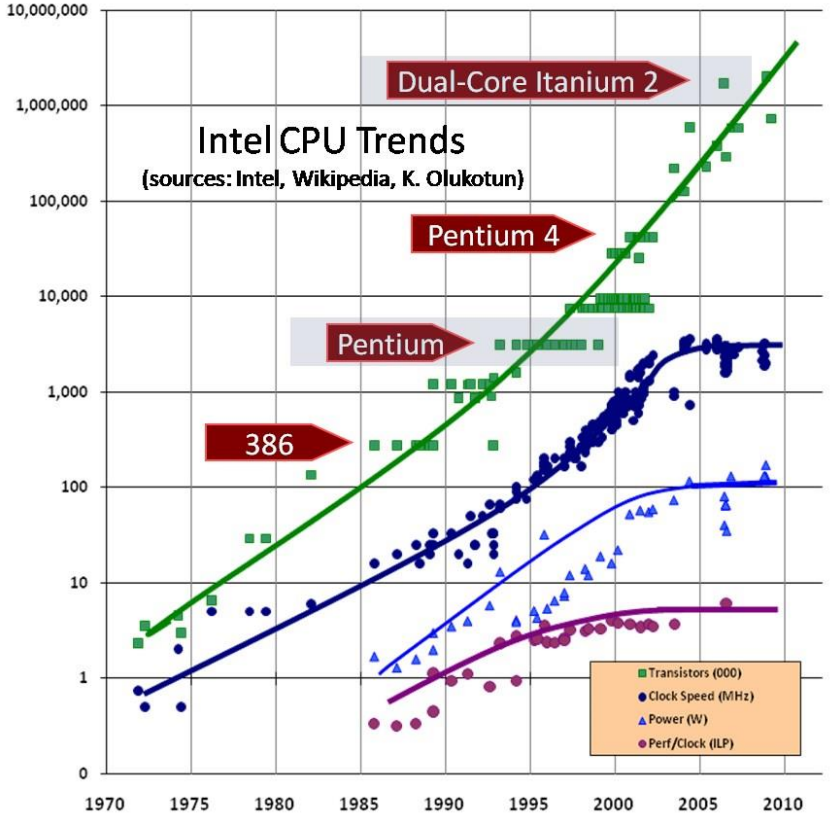
Firmaet opphørte i 1996.

Prosessoren x86

Intel's **x86** har gått fra å være én av mange prosessorer til å bli den dominerende

Noen årsaker:

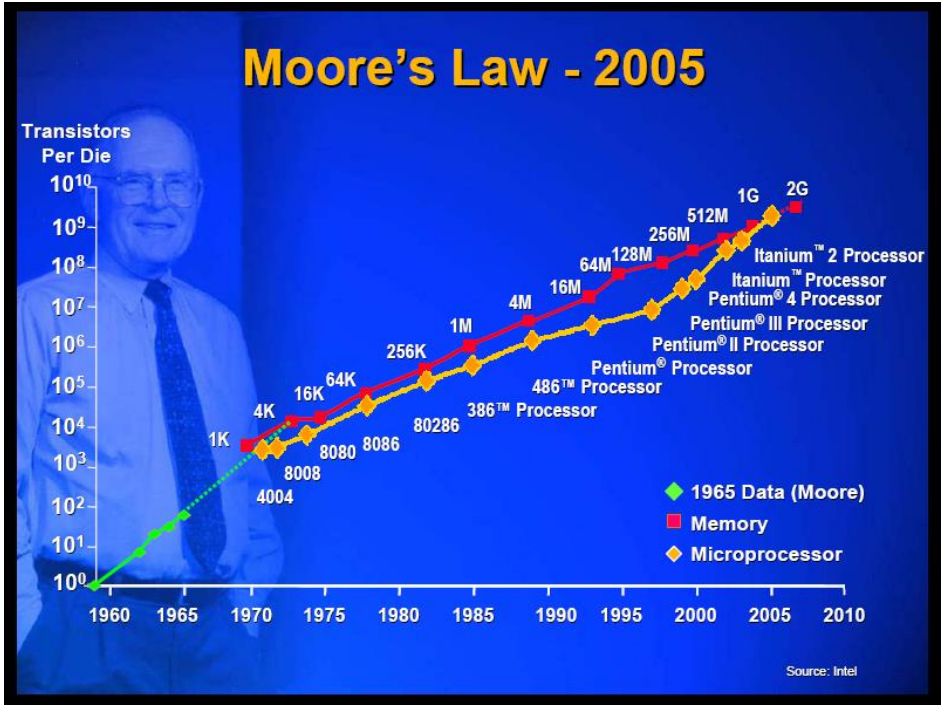
- Tidlig ute med nyvinninger
- Bakoverkompatibel



Moore's «lov»

Antall transistorer i en krets dobler seg annethvert år.

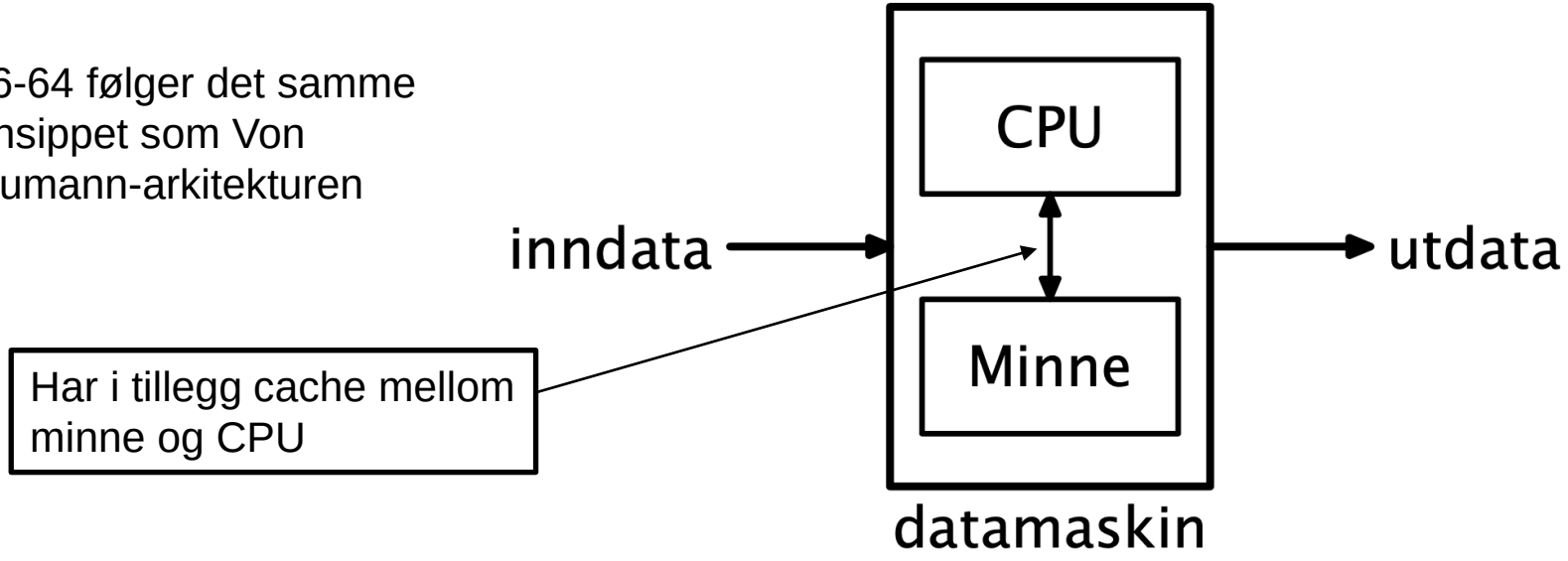
Ble observert av Gordon Moore ved Intel allerede i 1965.



Prosessoren x86-64

x86-64 fra Intel og AMD finnes i de aller fleste «vanlige» datamaskiner i dag.

x86-64 følger det samme prinsippet som Von Neumann-arkitekturen

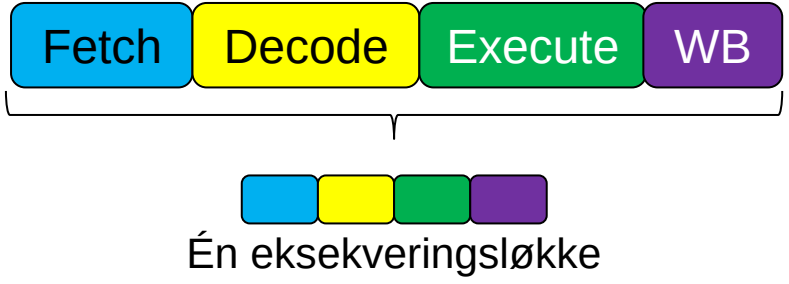


Eksekveringsløkken

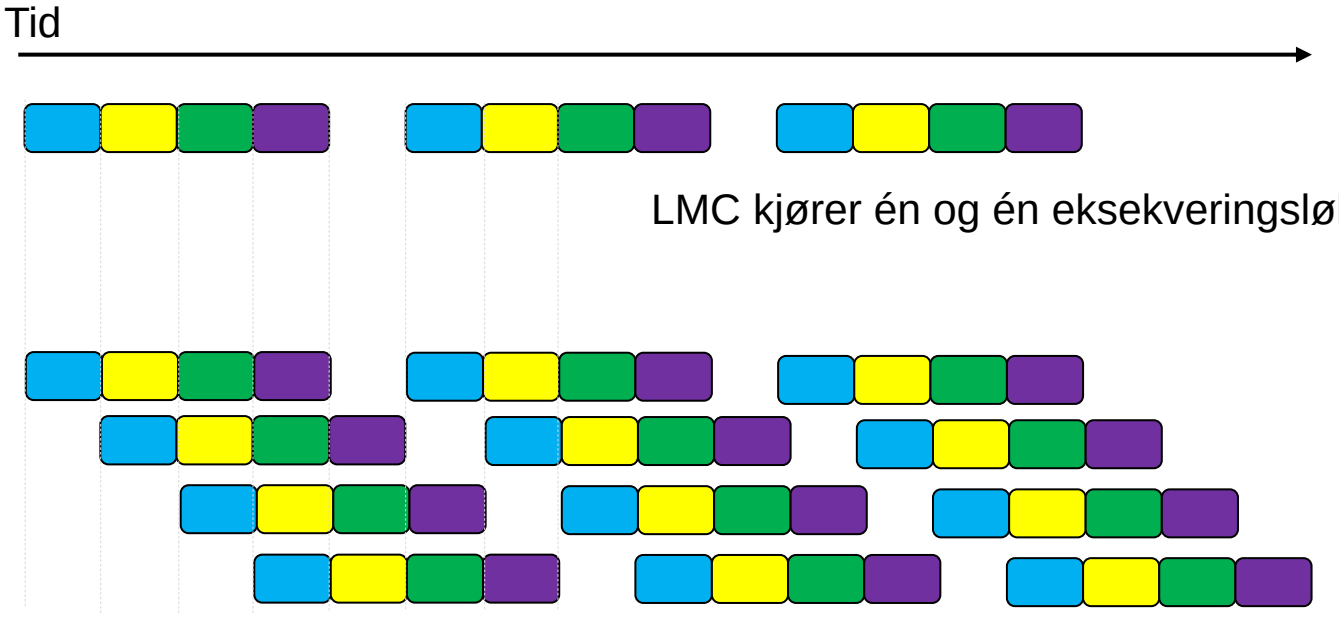
Både LMC og x86-64 har samme eksekveringsløkke:

1. Hent instruksjonen i minnet; programtelleren angir hvor.
2. Dekod instruksjonen i operasjon og adresse; de legges i instruksjonsregisteret og adresseregisteret.
3. Utfør instruksjonen.
4. Om aktuelt, skriv svaret i minnet/register.

x86-64 har i tillegg pipeline for å kunne jobbe raskere.



Pipelining



LMC kjører én og én eksekveringsløkke

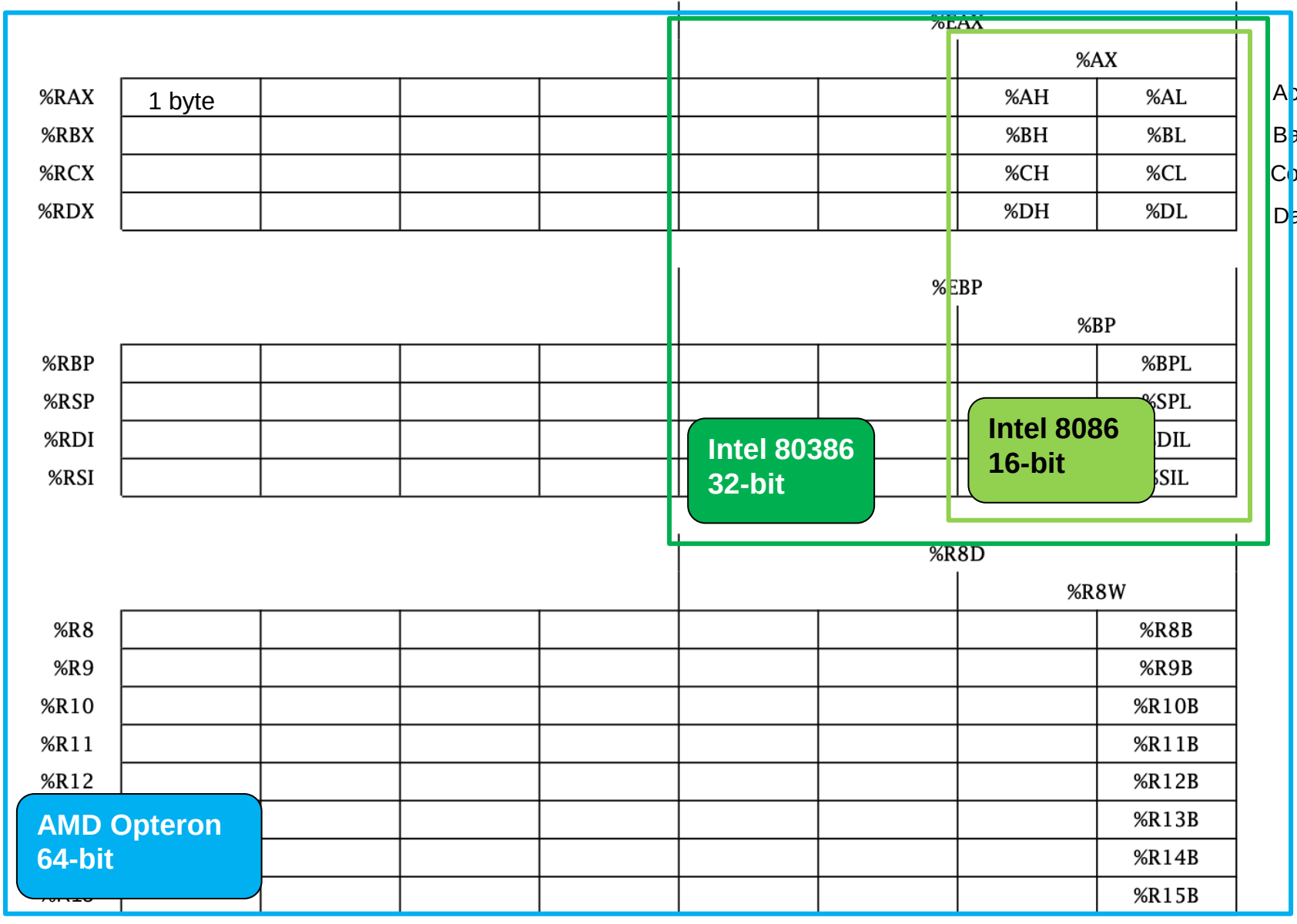
x86-64 kjører flere eksekveringsløkker i parallell «pipelining»

Registre

LMC har ett register som programmereren kan bruke (akkumulator-registeret)

Det kan lagre verdier –999 til +999.

x86-64 har 16 registre. De kan brukes som 1-, 2-, 4- og 8-bytes registre.

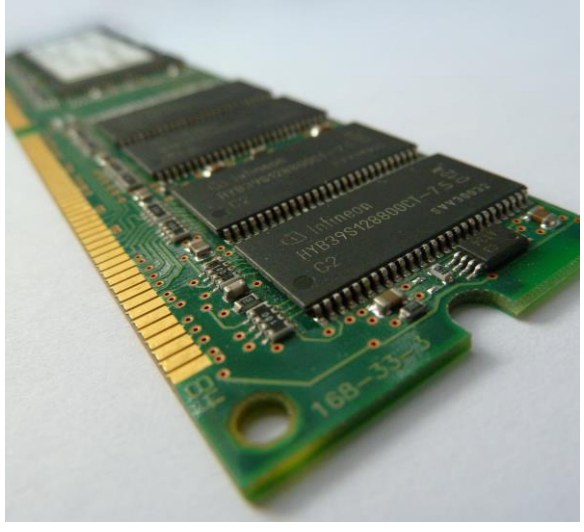

%RAX
%RBX
%RCX
%RDX
1 byte
%AX
%AH
%AL
%BH
%BL
%CH
%CL
%DH
%DL
Accumulator
Base
Counter
Data
%EBP
%BP
%RBP
%RSP
%RDI
%RSI
%BPL
%SPL
%DIL
%SIL
Intel 80386
32-bit
Intel 8086
16-bit
%R8D
%R8W
%R8
%R9
%R10
%R11
%R12
%R8B
%R9B
%R10B
%R11B
%R12B
%R13B
%R14B
%R15B
AMD Opteron
64-bit

Minne (RAM)

LMC har 100 celler i minnet, og hver celle kan lagre verdier –999–+999.

x86-64 kan ha inntil $2^{64} \approx 1,8 \cdot 10^{19}$ celler; dvs ca 17 000 000 TB («terabyte») eller 16 EB («exabyte»), men ingen maskiner i dag har så mye installert. Hver celle inneholder én byte (= 8 bit).

I tillegg har x86-64 støtte for **virtuelt minne.**



CPU-arkitektur

LMC er en RISC-maskin mens x86-64 er en CISC-maskin.

CISC («Complex instruction set computer») har noen enkle og noen ganske kompliserte instruksjoner, som:

REP MOVSB kan flytte et vilkårlig antall byte fra ett sted i minnet til et annet.

FYL2XP1 beregner $x \cdot (\log_2 y + 1)$.

RISC («Reduced instruction set computer») har et begrenset antall enkle instruksjoner, men kan utføre disse ekstremt raskt og effektivt.

CISC vs RICS, et eksempel

Anta at vi skal lage kode tilsvarende Python-setningen

```
a[i] += 1
```

x86-64:

```
leaq    a, %r8
movq    i, %r9
incq    (%r8, %r9, 8)
```

LMC:

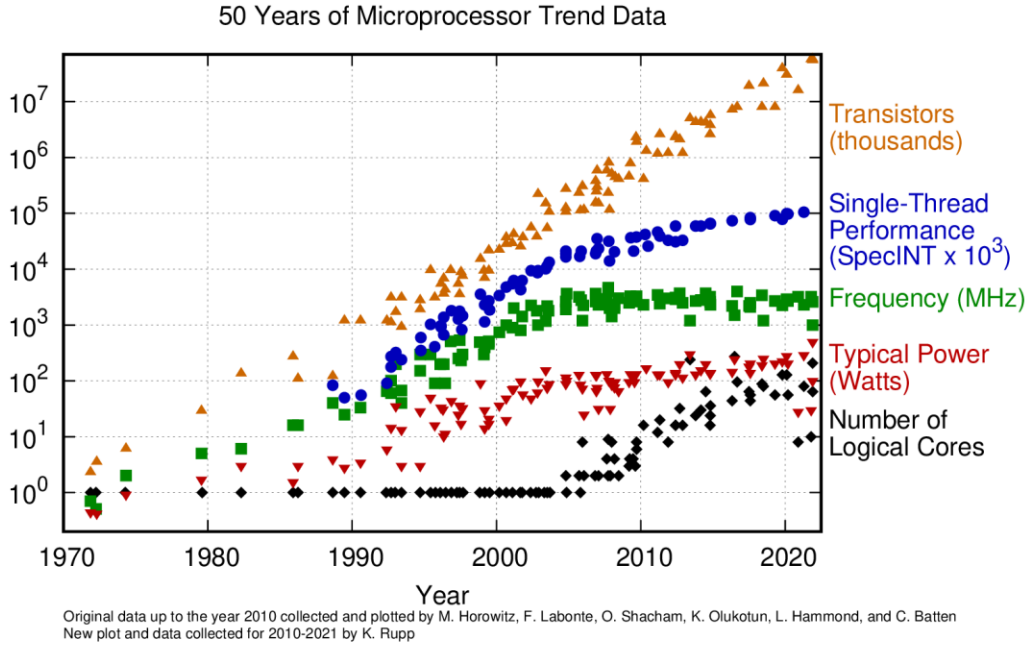
```
          LDA     LDA_A_i
          ADD     i
          STA     INCR_i
          LDA     STA_A_i
          ADD     i
          STA     INCR_i2
INCR_i    DAT     0
          ADD     v1
INCR_i2   DAT     0
          :
LDA_A_i   LDA     A
STA_A_i   STA     A
v1        DAT     1
i         DAT     0
```

Fremtiden

5. generasjons maskiner vil sannsynligvis

- Ha samme ytelse for hver enkeltprosessor som i dag
- Ha høy grad av parallellisering
- "Hardkodet elektronikk"
  - Kunstig intelligens, råbildebehandling, nettverk-på-chip..

Spåmenn har blitt overrasket før.



https://github.com/karlrupp/microprocessor-trend-data

Sikker programmering

Som programmerere er det viktig å skrive programmer som er

trygge («safe»), dvs vil alltid oppføre seg fornuftig

sikre («secure»), dvs tåler bevisste angrep

De første datamaskinene (1945 – 1955)

De første datamaskinene ble programmert med numeriske koder (som LMC); dette kalles gjerne **1. generasjonsspråk**.


Effektivt (for maskinen)


Veldig lett å gjøre ulike feil:

- bruke gal instruksjonskode
- skrive gal adresse
- ødelegge instruksjoner ved å overskrive med data
- prøve å utføre data som instruksjoner


Veldig vanskelig å feilsøke

Assemblerkode

Programmeringen ble enklere med assemblerkode, ofte kalt **2. generasjonsspråk**:

- 👍 Man slipper å huske numeriske koder.
- 👍 Adresser håndteres stort sett automatisk.
- 👎 Stadig ingen feilsjekking og ingen begrensninger på hva man får lov til.
- 👎 Vanskelig å feilsøke

```
INP             // Les inn
STA x           // verdien x.
ADD x           // Doble den.
OUT             // Skriv den ut.

x DAT 0         // Variabel x
```

De første programmeringsspråkene (1955 – 1965)

Man ønsket å

- programmere raskere
- gjøre programmene mer lesbare
- gjøre færre feil

Resultatet ble **3. generasjonsspråk**, som f.eks. FORTRAN, COBOL, BASIC.


Vi fikk array-er, setninger og funksjoner; mer strukturert programmering


Stadig ingen sjekk av grenser, antall parametre etc


Ingen krav om å deklarere variabler


Noen i ettertid «dumme» eller «rare» ideer

Dårlige Idéer

- Sikkerhet
  - Bedre enn for assembler, men stadig en lang vei å gå

- FORTRAN

```
   22 IF (E) 17, 24, 5

   24 DO 5 K = 1,3
```

- COBOL

```
        ALTER stmt1 TO PROCEED TO stmt2.

   stmt1.
        GO TO stmt3
```

https://en.wikibooks.org/wiki/Fortran/Fortran_examples

Nye programmeringsparadigmer (1965–1980)

Nå er datamaskinene blitt større og raskere, og tiden er moden for mer avanserte programmeringsspråk.

- 👍 Vi får nye paradigmer som objektorientert programmering, funksjonell programmering, logisk programmering etc
- 👍 Vi får spesialiserte språk for systemprogrammering, databasespørring etc
- 👍 Man finner ut at goto ikke er så lurt.
- 👍 De fleste språkene får i det minste noe sjekking av grenser
- 👎 De fleste nye språkene lager tregere kode

Eksempler: C, C++, Scheme, Pascal, Simula, SQL

Interpreterte programmeringsspråk (1980–1990)

Mikromaskinene kommer, og det blir laget mange typisk interpreterte språk: **4. generasjonsspråk**.

Interpreterte språk som Eiffel, Perl, Python, Ruby, . .

Interpreterte programmeringsspråk (1980–1990)

Mikromaskinene kommer, og det blir laget mange typisk interpreterte språk: **4. generasjonsspråk**.

Interpreterte språk som Eiffel, Perl, Python, Ruby, . .

- 👍 Mer fleksible, krever mindre av programmereren
- 👍 Kjapt å lage småprogrammer
- 👍 Veldig god sjekking av ulovligheter (under kjøring 👎)
- 👎 Enda tregere eksekvering

Programmeringsspråk for Internettet (1990-)

- 👍 Noen har gode mekanismer for distribuert eksekvering
- 👍 Noen er laget for integrering i nettlesere

Emerald, Java, Javascript, PHP, ...

«5. Generasjons Programmeringsspråk»

Dette er språk der programmereren angir hva han/hun ønsker å få gjort uten å spesifisere i detalj *hvordan* dette gjøres.

Eksempel: Sortering av tall

Inn: A

Ut: $\forall\{i \in [1;n]: A[i] < A[i-1]\}$

5. Generasjon: PROLOG


Bør gjøre det enkelt å programmere

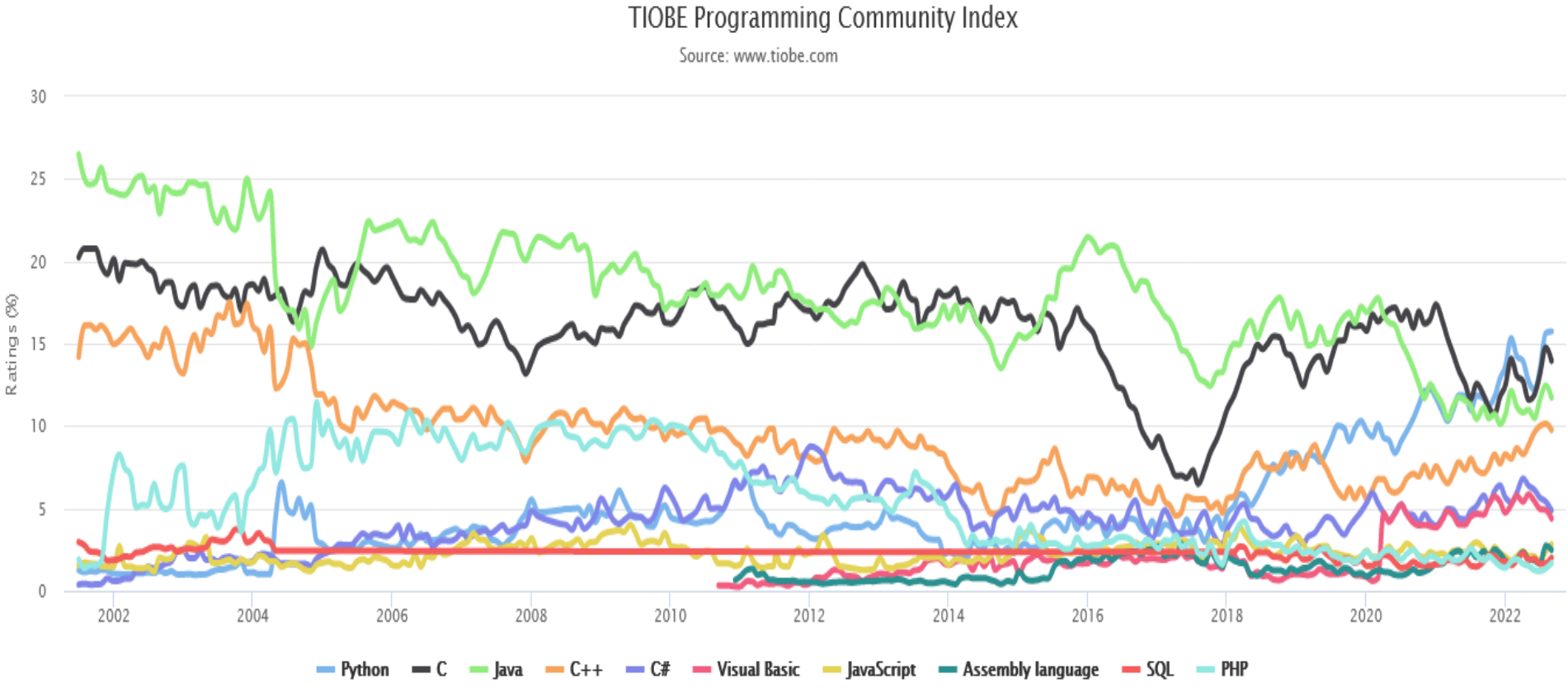

Programmene blir feilfrie (om formlene er korrekte)


Vanskelig (umulig?) å bruke på annet enn enkle demonstrasjoner

..så det er kanskje ikke fremtiden likevel?

Trøsten

- Det er fremdeles mulig for enkeltpersoner å gjøre en innsats
- Etter et masterstudium på IFI vil dere ha nok kunnskap til å designe deres eget programmeringsspråk
  - Med fornuftig valg av emner
- Flere av dagens mest populære språk er laget av enkeltpersoner..
  - Perl, Python, Ruby--


TIOBE Programming Community Index
Source: www.tiobe.com
Ratings (%)
30
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
2016
2018
2020
2022
Python
C
Java
C++
C#
Visual Basic
JavaScript
Assembly language
SQL
PHP