# Dagens tema

- Rask-maskinen
  - Litt datamaskinhistorie
  - Registre og lagre
  - Instruksjoner
- Rasko-kode
- Raskas-kode





# Datamaskinenes historie

Menneskene har alltid prøvd å lage maskiner for å løse sine problemer.

Midt på 1800-tallet var problemet *tabeller* med feil.

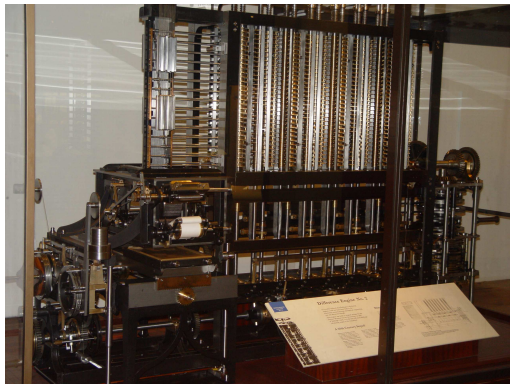
**Charles Babbage** konstruerte sin *Difference Engine* som kunne lage tabeller automatisk ved å regne ut polynomer. (Den ble først ferdig i 1991.)

Han arbeidet også med en *Analytical Engine* som skulle bli en generell beregningsmaskin.





*The difference engine* på *Science Museum* i London







# De første moderne datamaskiner

Problemet i 1930-årene var kanoner. Det er mulig å beregne en prosjektilbane, men det er mye arbeid for en matematiker.

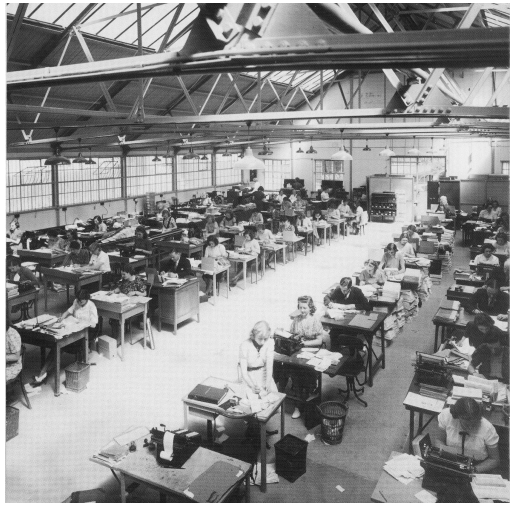
*U.S. Army Ordnance Department Ballistic Research Laboratory* trengte data for dusinvis av nye kanoner.

**Løsning**

Lag *arbeidsbeskrivelse*, og la egne «beregnere» gjøre jobben.

Fra en eldre utgave av *Webster's Dictionary*:

computer n, one that computes; *specif:* an automatic electronic machine for performing calculations







**Problem**

Hver bane tok opptil 20 timer å beregne (selv med elektrisk bordregnemaskin), og man trengte 2–4000 ulike baner for hver kanon.

**Løsning**

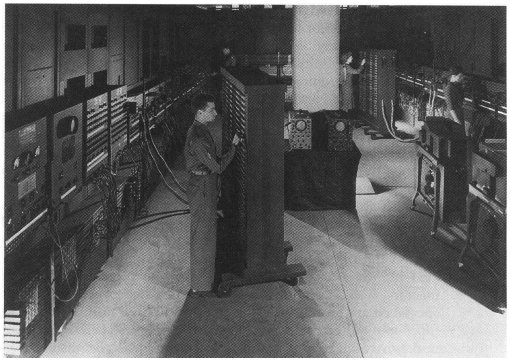
Lag en maskin som gjør dette automatisk.

*Moore School of Electrical Engineering* ved universitetet i Pennsylvania gjorde det med penger fra *Ballistic Research Laboratory*. Resultatet ble **Eniac** som ble ferdig i 1945. Den kunne beregne en kulebane på drøyt 10 s.





Eniac målte 2½×1×30 m, veide 30 tonn og inneholdt 18 000 radiorør.







## Oppbyggingen av Eniac

Tanken var å kopiere en menneskelig beregner. Den har

Aritmetisk enhet («ALU») tilsvarte regnemaskinen med de fire regneartene:

$$+ \quad - \quad \times \quad \div$$

Regnemaskinen har et tall for videre beregning; datamaskinen har et **register**.

Minnet tilsvarte et ark med mellomresultater. Datamaskinen kunne overføre innholdet av registeret til eller fra en celle i minnet.

Programmet tilsvarte beregnerens arbeidsbeskrivelse. Det skulle følges helt slavisk.

## Programmet

Et program for datamaskinen inneholdt de samme elementene som beregnerens arbeidsbeskrivelse:

**Aritmetiske operasjoner** var mulig i de fire regneartene; svaret kom i registeret.

**Mellomlagring av data** skjedde ved at registeret ble kopiert til en angitt celle i minnet.

**Hopp** til en angitt instruksjon var nødvendig for å kunne gå i løkker.

**Tester** i forbindelse med hopp var typisk på om registeret var $< 0$, $= 0$ eller $> 0$.

Programmene ble etter hvert kodet som tall (mens Eniac ble kodet med kabler).





## Rask-maskinen

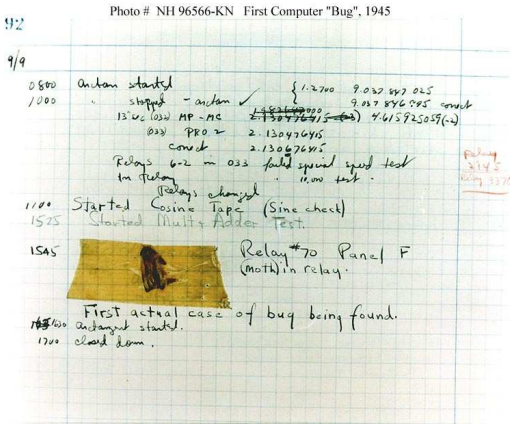
Rask-maskinen er en forenklet MIPS-prosessor.






## Den aller første «debugging»

Også de første datamaskinene hadde feil («bugs»). Den 9. september 1945 kl 15.45 fant man feilen i relé nr 70 i panel F i en Mark II Aiken elektromekanisk datamaskin og foretok den første «debugging».








### Minne:

Inneholder 10 000 celler som kan inneholde enten data eller instruksjoner.

### 33 registre:

$R_0$ inneholder alltid 0.

$R_1$–$R_{31}$ inneholder data.

PC gir adressen til neste instruksjon.

## Programutførelsen

Instruksjonene utføres alltid én og én:

```
PC = 0;
while (true) {
   if (PC == 9990) break;

   (CF,CA,CB,CC) = split Mem[PC];
   PC = PC + 1;
   utfør instruksjonen (CF,CA,CB,CC)
}
```





## Instruksjonene

| Nr | Navn | Operasjon |
|---|---|---|
| 1 | LOAD $R_A$, $R_B$, C | $R_A \leftarrow$ Mem[$R_B$+C] |
| 2 | SET $R_A$, $R_B$, C | $R_A \leftarrow R_B$+C |
| 3 | STORE $R_A$, $R_B$, C | Mem[$R_B$+C] $\leftarrow R_A$ |







| Nr | Navn | Operasjon |
|---|---|---|
| 4 | ADD $R_A$, $R_B$, $R_C$ | $R_A \leftarrow R_B+R_C$ |
| 5 | SUB $R_A$, $R_B$, $R_C$ | $R_A \leftarrow R_B-R_C$ |
| 6 | MUL $R_A$, $R_B$, $R_C$ | $R_A \leftarrow R_B \times R_C$ |
| 7 | DIV $R_A$, $R_B$, $R_C$ | $R_A \leftarrow R_B/R_C$ |





| Nr | Navn | Operasjon |
|---|---|---|
| 8 | EQ $R_A$, $R_B$, $R_C$ | $R_A \leftarrow$ 1 hvis $R_B$=$R_C$, ellers 0 |
| 9 | NEQ $R_A$, $R_B$, $R_C$ | $R_A \leftarrow$ 1 hvis $R_B \neq R_C$, ellers 0 |
| 10 | LESS $R_A$, $R_B$, $R_C$ | $R_A \leftarrow$ 1 hvis $R_B$<$R_C$, ellers 0 |
| 11 | LESSEQ $R_A$, $R_B$, $R_C$ | $R_A \leftarrow$ 1 hvis $R_B \leq R_C$, ellers 0 |
| 12 | GTR $R_A$, $R_B$, $R_C$ | $R_A \leftarrow$ 1 hvis $R_B$>$R_C$, ellers 0 |
| 13 | GTREQ $R_A$, $R_B$, $R_C$ | $R_A \leftarrow$ 1 hvis $R_B \geq R_C$, ellers 0 |
| 14 | JUMPEQ $R_A$, $R_B$, C | Hvis $R_A$=$R_B$, PC $\leftarrow$ C |
| 15 | JUMPNEQ $R_A$, $R_B$, C | Hvis $R_A \neq R_B$, PC $\leftarrow$ C |

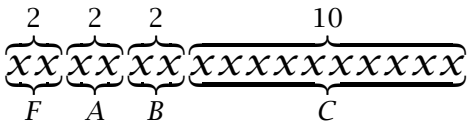
| Nr | Navn | Operasjon |
|---|---|---|
| 16 | CALL $R_A$, $R_B$, C | $R_{31} \leftarrow$ PC og PC $\leftarrow$ C |
| 17 | RET | PC $\leftarrow R_{31}$ |





## Representasjon av instruksjoner

En Rask-instruksjon lagres slik:








## Spesielle adresser

Fem standardfunksjonen utgjør «operativsystemet» og ligger i faste adresser:

| Adresse | Funksjon |
|---|---|
| 9990 | exit($R_{11}$) |
| 9991 | getchar() |
| 9992 | getint() |
| 9993 | putchar($R_{11}$) |
| 9994 | putint($R_{11}$) |





## Et eksempel

Dette programmet skriver ut et utropstegn:

```
 211000000000033
1600000000009993
 211000000000001
1600000000009990
```

## Rasko-koden

For å få Rask-maskinen til å utføre koden, må vi «pakke den tekstlig inn» i såkalt Rasko-kode:

```
#! /store/bin/rask

 211000000000033
1600000000009993
 211000000000001
1600000000009990
```





```
> rask utrop.rask
!
DUMP
====
PC=9990
Last 6 instructions:
     0     1  9993     2     3  9990
Registers
 R0=   0  R1=   0  R2=   0  R3=   0  R4=   0  R5=   0  R6=   0  R7=   0
 R8=   0  R9=   0 R10=   0 R11=   1 R12=   0 R13=   0 R14=   0 R15=   0
R16=   0 R17=   0 R18=   0 R19=   0 R20=   0 R21=   0 R22=   0 R23=   0
R24=   0 R25=   0 R26=   0 R27=   0 R28=   0 R29=   0 R30=   0 R31=   4
Memory
   0: 211000000000033 1600000000009993 211000000000001 1600000000009990
<skipped 0-lines>
```





## Raskas-kode

For å gjøre det enklere å skrive Rasko-kode, har man funnet på Raskas. Da kan man

- bruke navn istedenfor tallkoder
- bruke navn på datalokasjoner
- sette navn på instruksjoner (for å hoppe dit)





## Eksempel: nexta.raskas

```
# Et minimalt testprogram:
# Det ber om et tall v og skriver så ut v+1.

main:   SET     R11,'?'     #          '?'
        CALL    putchar     # putchar(   );

        CALL    getint      # R1 = getint();
        SET     R3,1        #           1
        ADD     R11,R1,R3   # R11 = R1+ ;
        CALL    putint      # putint(R11);

        SET     R11,0       #       0
        CALL    exit        # exit( );
```

## Kodefilen

Den genererte Rasko-filen ser slik ut:

```
#! /store/bin/rask

211000000000063
1600000000009993
1600000000009992
203000000000001
411010000000003
1600000000009994
211000000000000
1600000000009990
```





Kjøringen går slik:

```
> raskas nexta.raskas
> rask nexta.rask
?322
323
```





## Oversikt laget av programmet raskas

```
Raskas assembler version 1.0
Source file: nexta.raskas
                        # Et minimalt testprogram:
                        # Det ber om et tall v og skriver så ut v+1.
   0:  2 11  0      63  main:   SET     R11,'?'     #         '?'
   1: 16  0  0    9993          CALL    putchar     # putchar(   );
   2: 16  0  0    9992          CALL    getint      # R1 = getint();
   3:  2  3  0       1          SET     R3,1        #          1
   4:  4 11  1       3          ADD     R11,R1,R3   # R11 = R1+ ;
   5: 16  0  0    9994          CALL    putint      # putint(R11);
   6:  2 11  0       0          SET     R11,0       #      0
   7: 16  0  0    9990          CALL    exit        # exit( );
Program labels:
exit       9990
getchar    9991
getint     9992
main          0
putchar    9993
putint     9994
```





## Oppsummering

- Vi har sett på instruksjonene og registrene i Rask.
- Rask-maskinen leser filer med Rasko-kode.
- Det er lettere å skrive *assemblerkoden* Raskas enn *maskinkoden* Rasko.