

# Introduktion til programmering (14)

## Introduktion til programmering (41)

Sidst opdateret 8. juni 2022

### Indhold

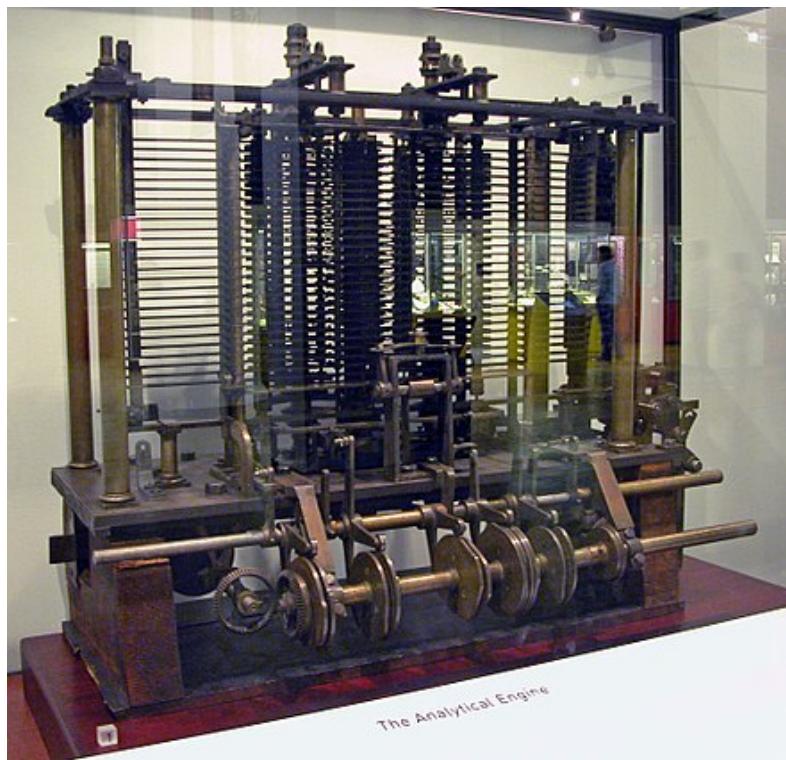
- 1 Programmering
  - 1.1 Den korte historie
- 2 Begreber inden for softwareudvikling
  - 2.1 Prøv det selv
- 3 Talsystemer
  - 3.1 Den lille tabel
  - 3.2 Lær det binære talsystem på
  - 3.3 Betegnelser
  - 3.4 Forkortelser
  - 3.5 Binær aritmetik
  - 3.6 Det hexadecimale talsystem
  - 3.7 Typer af tal
- 4 Kontakter
  - 4.1 Relæ
  - 4.2 Vakuumrør
  - 4.3 Transistor
- 5 Logiske kredsløb
  - 5.1 Sandhedstabeller
  - 5.2 Gates af elektronik
  - 5.3 Moores lov
  - 5.4 Lidt historie
  - 5.5 Beregning med logiske kredsløb
  - 5.6 ALU (Arithmetic logic unit)
- 6 Hukommelse
  - 6.1 OR Latch
  - 6.2 Set Reset AND-OR latch
  - 6.3 RAM intro
- 7 Hjertet i en CPU (clock)
  - 7.1 Hertz
- 8 En SAP (simple as possible) CPU
  - 8.1 Michells SAP1 simulering
  - 8.2 opkoder og maskinkode
  - 8.3 Altair
  - 8.4 Digrule2
  - 8.5 Moderne processorer
- 9 Kompilere og programmeringssprog
  - 9.1 Programmeringssprog
- 10 Syntaks og instruktioner
  - 10.1 Eksempler på syntaks
- 11 Grundlæggende programmering
- 12 Typer af applikationer
- 13 Udviklingsværktøjer
- 14 Version/source control

# Programmering

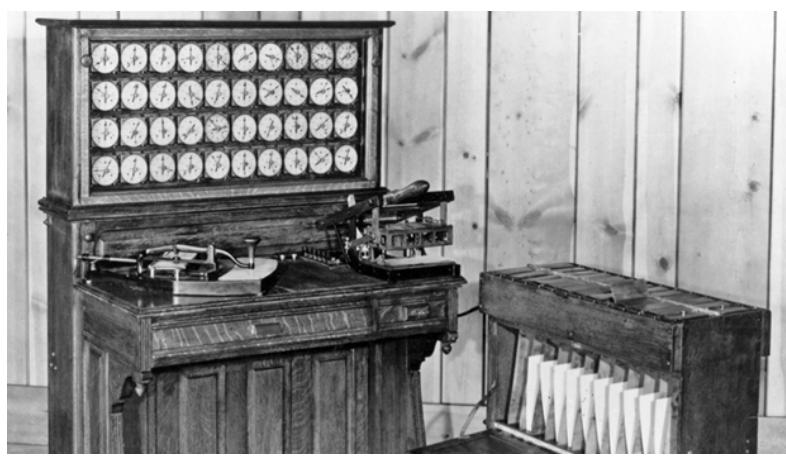
Formålet med programmering er, at skabe en samling instruktioner for at automatisere en process på en computer (typisk afviklet i et standardiseret operativsystem).

## Den korte historie

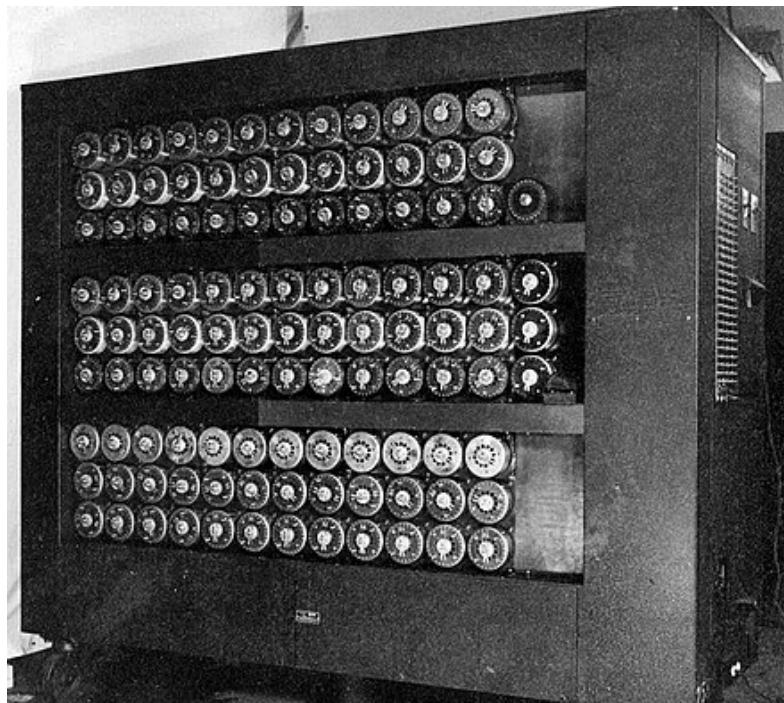
- Har rødder helt tilbage til Charles Babbage's mekaniske computer kaldet Analytical Engine fra omkring 1837 (183 år siden). Her skrev Ada Lovelace i 1843 det første (teoretiske) computerprogram



- Herman Hollerith skabte omkring 1885 (135 år siden) sin programmerbare "Electromechanical punched card tabulator" som blev brugt til folkeoptælling. Hans firma blev senere (1925) omdøbt til IBM.



- Under 2. verdenskrig dukkede de første elektroniske computere op
  - Tyskernes Z3-computer baseret på relæer
  - Englændernes Bombe brugt til at bryde tyskernes Enigma kodemaskine. Skabt blandt andet af Alan Turing, som har haft kæmpe indflydelse på computer teknologi
    - Udgav en del afhandling, og beskrev blandt andet Turing Machine hvis teorier satte IT teknologien igang
    - Se filmen *The Imitation Game*



- 1952 blev den første (britiske) kommersielle computer udviklet - [Ferranti Mark I](#)
- I 1960'erne udviklere MIT og NASA operativsystemerne bag Apollo-missionerne
  - [Ikonisk billede](#) af Margaret Hamilton med source koden til Apollo skevet i Assembler

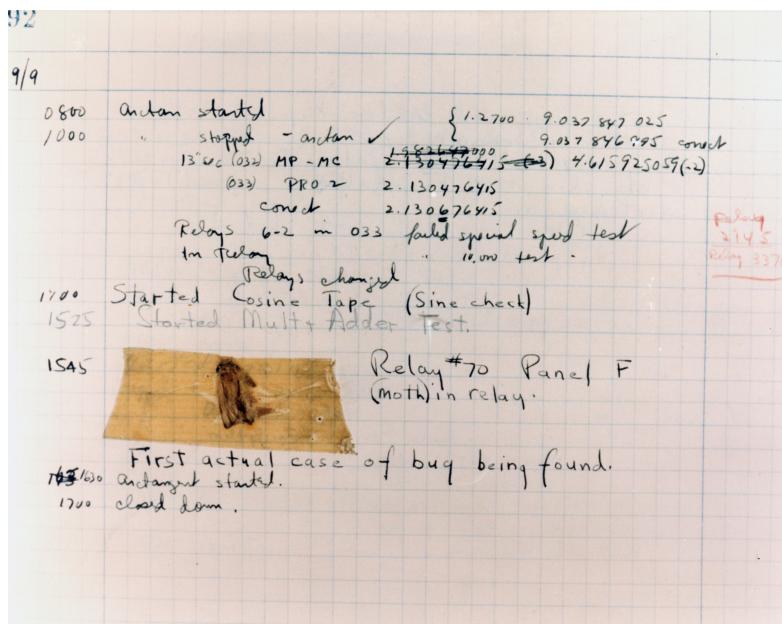


---

(Modul 145)

## Begreber inden for softwareudvikling

- Instruktioner til CPU skrives i **kildekode/source code**
  - Der findes et hav af forskellige programmeringssprog
  - Kildekode kompileres herefter til en for processoren forståelige instruktioner
  - Det er som alt muligt andet et håndværk at "skrive kode"
    - **Programmeringsparadigmer**
      - Iterativ
      - Procedural
      - Objektorientert
      - Funktionsorienteret
      - Opmærkningsbaseret
      - Visuelt
    - **IT Arkitektur**
- Programmering handler typisk om at styre programpointeren til at afvikle instruktioner, og alle moderne programmeringssprog indeholder typisk (i en eller anden form)
  - Variabler
  - Flow instruktioner
    - Løkker
    - Betingelser
  - Metoder/Funktioner
- Der kan opstå fejl i koden - også kaldet **bugs**
  - **Ikonisk billede** af "en bug" fra en relæ computer (MARK II) i 1946. Bug fundet af [Grace Hopper] - en af pionerne bag programmeringssprog (herunder COBOL)



- Fejlfinding / debugging er derfor en stor del af software udvikling.
- Applikationer testes på forskellig vis
  - Unit test
  - UI test
  - Integration test

## Prøv det selv

Software udvikling kan godt - i begrænset form - ske visuelt.

Prøv selv på [Hour of Code](#) - eksempelvis [blockly.games](#).



Alternativ AppInventor - Android (og iOS), eller MicroBit.

(Modul 146)

## Talsystemer

Den klassiske computer er baseret på det binære talsystem

- til/fra
- spænding/ikke spænding

Baseret på to værdier (typisk 0 og 1) kan man repræsentere alle tal (hvis man har plads nok)

### Den lille tabel

Værdi	Binært
0	00000000
1	00000001
2	00000010
3	00000011
4	00000100
5	00000101
6	00000110
7	00000111
8	00001000
9	00001001
10	00001010
11	00001011
12	00001100
13	00001101
14	00001110
15	00001111
16	00010000

### Lær det binære talsystem på

- csdemo om nummersystemer
- Quiz hos csdemo om konvertering til binære tal

### Betegnelser

- 1 = bit
- 4 = nibble
- 8 = byte
- 8/16/32/64 = word

## Forkortelser

### Bit

- 1 Kb = 1.024 bit (128 bytes)
- 1 Mb = 1.048.576 bit
- 1 Gb = 1.073.741.824 bit
- 1 Tb = 1.099.511.627.776 bit

### Bytes

- 1 KB = 1.024 bytes
- 1 MB = 1.048.576 bytes
- 1 GB = 1.073.741.824 bit
- 1 TB = 1.099.511.627.776 bit

## Binær aritmetik

Man kan foretage alle former for beregninger i det binære talsystem ligesom i det decimale talsystem - eksempelvis addition

$$\begin{array}{r}
 \text{(10)} \quad \text{(2)} \\
 \begin{array}{r} 1 \\ 5 \\ 7 \\ \hline 12 \end{array} \quad \begin{array}{r} 1 \ 1 \ 1 \\ 1 \ 0 \ 1 \\ 1 \ 1 \ 1 \\ \hline 1 \ 1 \ 0 \ 0 \end{array}
 \end{array}$$

$$\begin{array}{r}
 \text{(10)} \quad \text{(2)} \\
 \begin{array}{r} 1 \ 1 \\ 3 \ 5 \\ 1 \ 8 \ 9 \\ \hline 2 \ 2 \ 4 \end{array} \quad \begin{array}{r} 1 \ 1 \ 1 \ 1 \ 1 \ 1 \\ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \\ 1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \\ \hline 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \end{array}
 \end{array}$$

Se eksempler på [forskellige typer af beregninger](#) hos Wikipedia.

## Det hexadecimale talsystem

I IT teknologier vil du også møde det hexadecimale talsystem som tæller 0-15

Værdi	Hex
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

## Lær det hexadecimale talsystem på

- csdemo om nummersystemer
- Quiz hos csdemo om konvertering til hexadecimale tal

## Typer af tal

Generelt findes der heltal (signed (-)) og unsigned (+)) og kommatal.

## Heltal

Type	Bites	Spænd
Lille positivt heltal	8	0 til 255
Stort positivt heltal	16	0 til 65.536
Meget stort positivt heltal	64	0 til 18.446.744.073.709.551.616
Stort positivt og negativt heltal	32	-2.147.483.648 til +2.147.483.647

## Kommatal

Der er flere måder at beskrive et kommatal på. De fleste CPU'er har indbygget floating point repræsentation, som er en måde at beskrive et kommatal. Det er super hurtigt men ikke helt præcist på de yderste decimaler. Konvertering til floating point kan være ret kompleks men denne artikel forklarer det ret godt.

De fleste programmeringssprog tilbyder også andre muligheder for repræsentation af kommatal.

## Udfordringer

Hvis der vælges forkerte typer af heltal og kommatal kan det skabe fejl som kan være svære at finde. Se eksempelvis Engineering Disasters 13 - Software Flaws på YouTube. Her beskrives eksempelvis software fejl i amerikanernes Patriot missiler under Golfkrigen i starten af 90'erne.

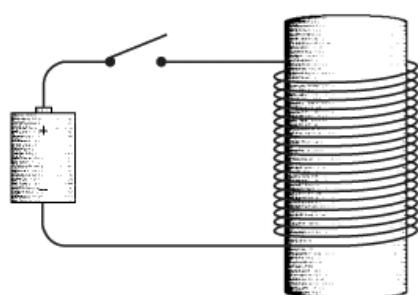
---

(Modul 147)

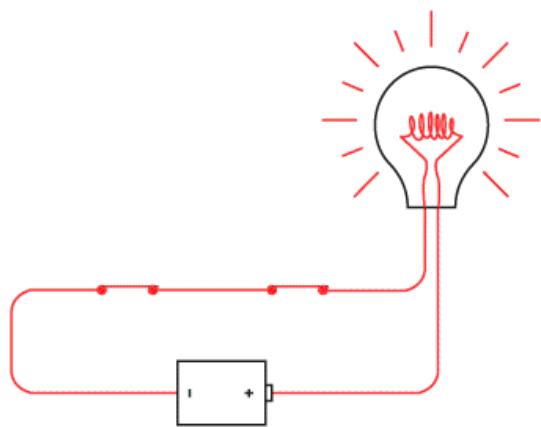
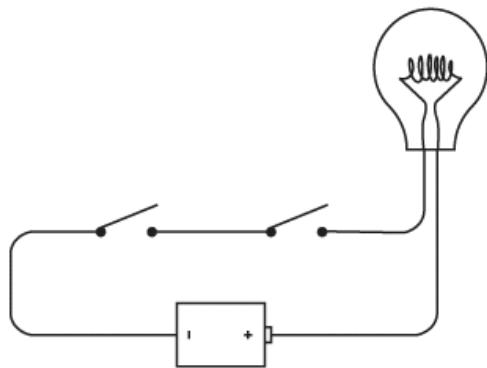
## Kontakter

Alle processorer er skabt af "kontakter".

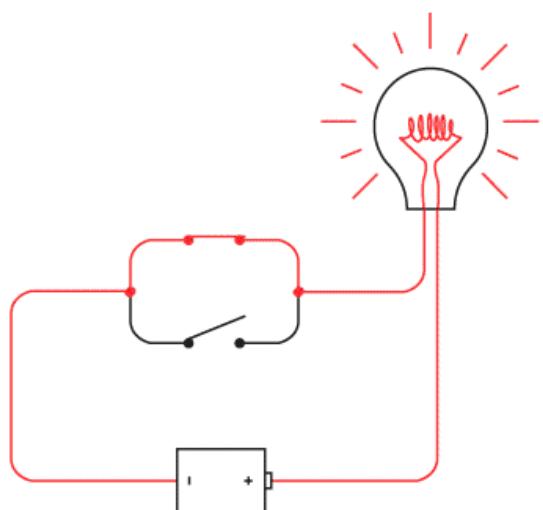
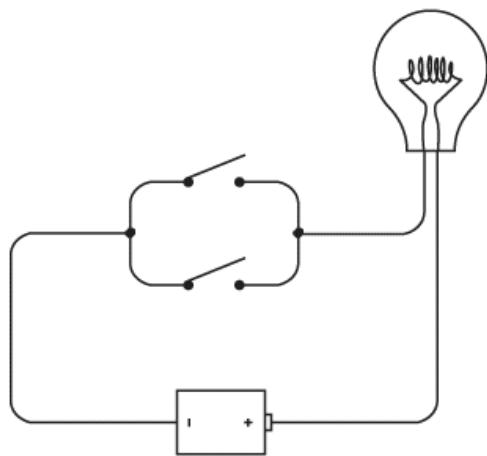
Her en simpel kontakt som skal være "til" for at aktivere en magnet.



Her et kredsløb med to kontakter hvor begge input skal være "til" for at lampen lyser.



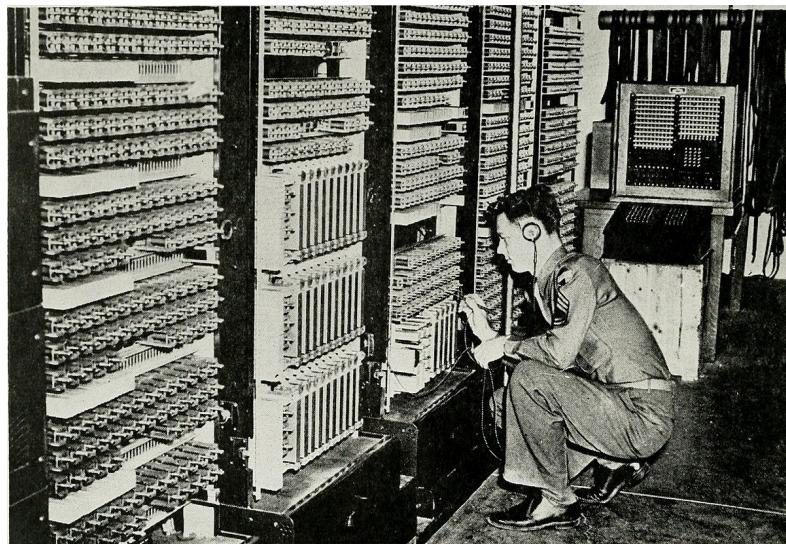
Og her et kredsløb med to kontakter hvor blot en af kontakterne skal være "til" for at lampen lyser.



(Alle tegninger er fra den forrygende bog **Code: The Hidden Language of Computer Hardware and Software** af Charles Petzold som i den grad kan anbefales hvis du vil vide mere om hvordan en computer/cpu fungerer.)

## Relæ

Bell Laboratories relay computer for U.S. Army (1946)



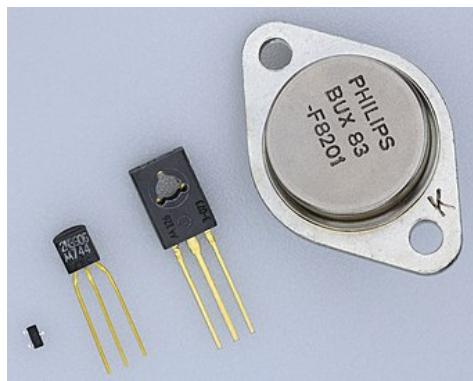
## Vakuumrør

Forskellige typer af vakuumrør



## Transistor

I 1950'erne blev "kontakterne" digitale. Se [denne gode dokumentar](#) om opfindelsen af transistoreren



---

(Modul 148)

## Logiske kredsløb

Logiske kredsløb er grundlaget for alt computer teknologi, og gør det muligt at regne og huske.

- NOT
- AND
- OR
- XOR
- NAND
- NOR
- XNOR

## Sandhedstabeller

AND			NAND		
A	B	Q	A	B	Q
0	0	0	0	0	1
1	0	0	1	0	1
0	1	0	0	1	1
1	1	1	1	1	0

A	B	Q	A	B	Q
0	0	0	0	0	1
1	0	1	1	0	0
0	1	1	0	1	0
1	1	1	1	1	0

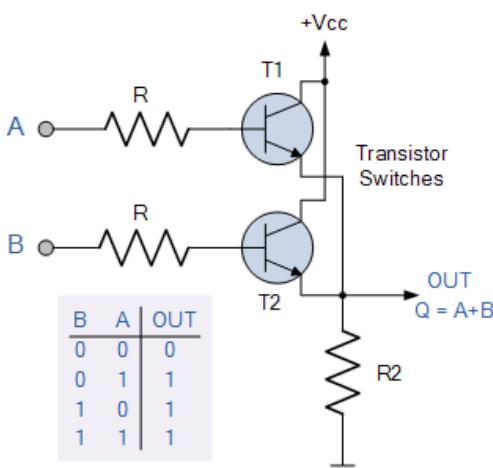
  

A	B	Q	A	B	Q
0	0	0	0	0	1
1	0	1	1	0	0
0	1	1	0	1	0
1	1	0	1	1	1

Check <https://csdemo.cronberg.dk/gates.html> så du kan se hvordan de fungerer.

## Gates af elektronik

Alle gates er i virkeligheden ret simple rent elektronisk. En OR gates eksempelvis kan skabes af to transistorer.

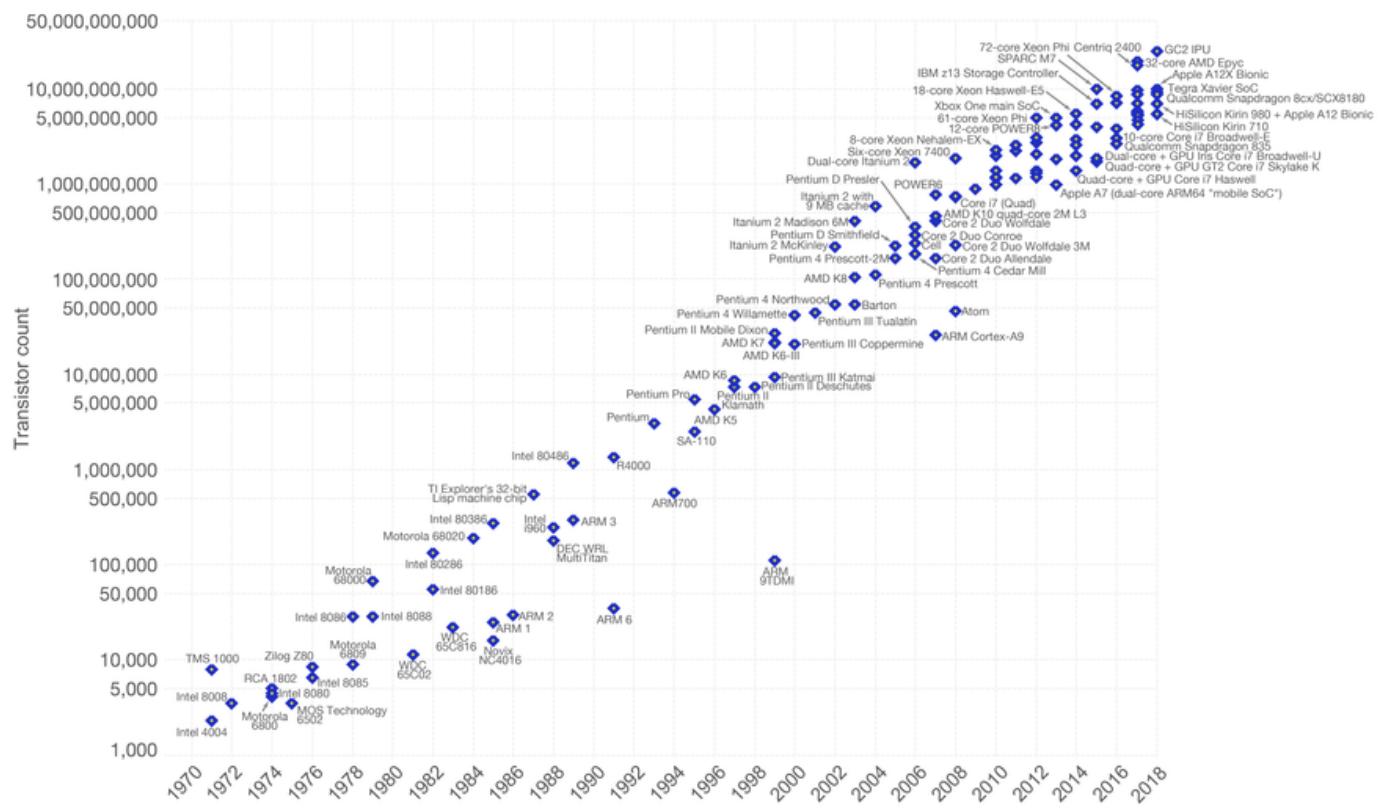


## Moore's lov

Gordon E. Moore (Intel) forudsagde i 1970, at antallet af transistorer i processorer ville blive fordoblet cirka hvert andet år. På det tidspunkt kunne man proppe omkring et par tusinde transistorer i en chip. Hans forudsigelse har holdt nogenlunde stik i halvtreds år, men nu begynder det at knibe. Komponenterne bliver så små at man har svært ved at holde styr på elektronerne.

## Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.



Data source: Wikipedia ([https://en.wikipedia.org/wiki/Transistor\\_count](https://en.wikipedia.org/wiki/Transistor_count))

The data visualization is available at [OurWorldInData.org](https://OurWorldInData.org). There you find more visualizations and research on this topic.

Licensed under CC-BY-SA by the author Max Roser.

Dog er der nye spændende materialer på vej som både vil øje antallet af transistorer igen, og samtidigt gøre kredsløb mere stabile og mindre varme. Især Grafen/Graphene er meget omtalt pt. Det er en form for carbon med kun et lag atomer, og er ufatteligt stærk. De første batterier er begyndt at dukke op i 2020 (<https://www.youtube.com/watch?v=dnE1nO6o-do>).

## Lidt historie

Check lige George Boole (1815-1864) - ham alle IT udviklere bør kende til.

- Far til boolsk algebra
  - Logiske beregninger/formler med klasser/set af elementer (ikke tal) og operatorer som eksempelvis + (or), x (and) og 1 (not)

Og check Claude Shannon (1916-2001), som videre udviklede George Boole's tanker til elektroniske kredsløb omkring 1937.

Og medens du er igang - John Bardeen (1908-1991), William Bradford Shockley (1910-1989) and Walter Houser Brattain (1902-1987) - som fik Nobel prisen i Fysik i 1956 for deres arbejde indenfor halvledere og transitorer.

Og sluttelig Jack Kilby (1923-2005) som opfandt IC kredse i 1950'erne (Nobel prisen i fysik i 2000).

- George Boole
- Claude Shannon

## Beregning med logiske kredsløb

Når først man har adgang til forskellige logiske kredsløb kan man foretage alle typer af beregninger - startende med at lægge to bit sammen uden mente

<https://csdemo.cronberg.dk/halfadder.html>

og to bit sammen med mente

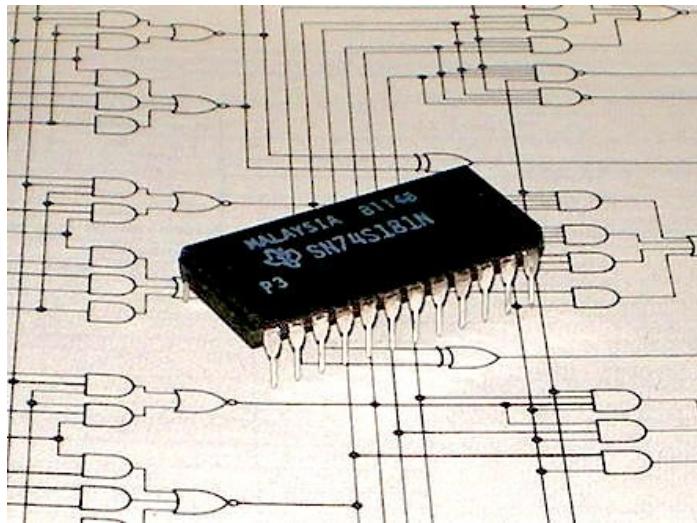
<https://csdemo.cronberg.dk/fulladder.html>

Og hvis man kan lægge to bit sammen kan man lægge tre bit sammen, og fire bit sammen, og ...

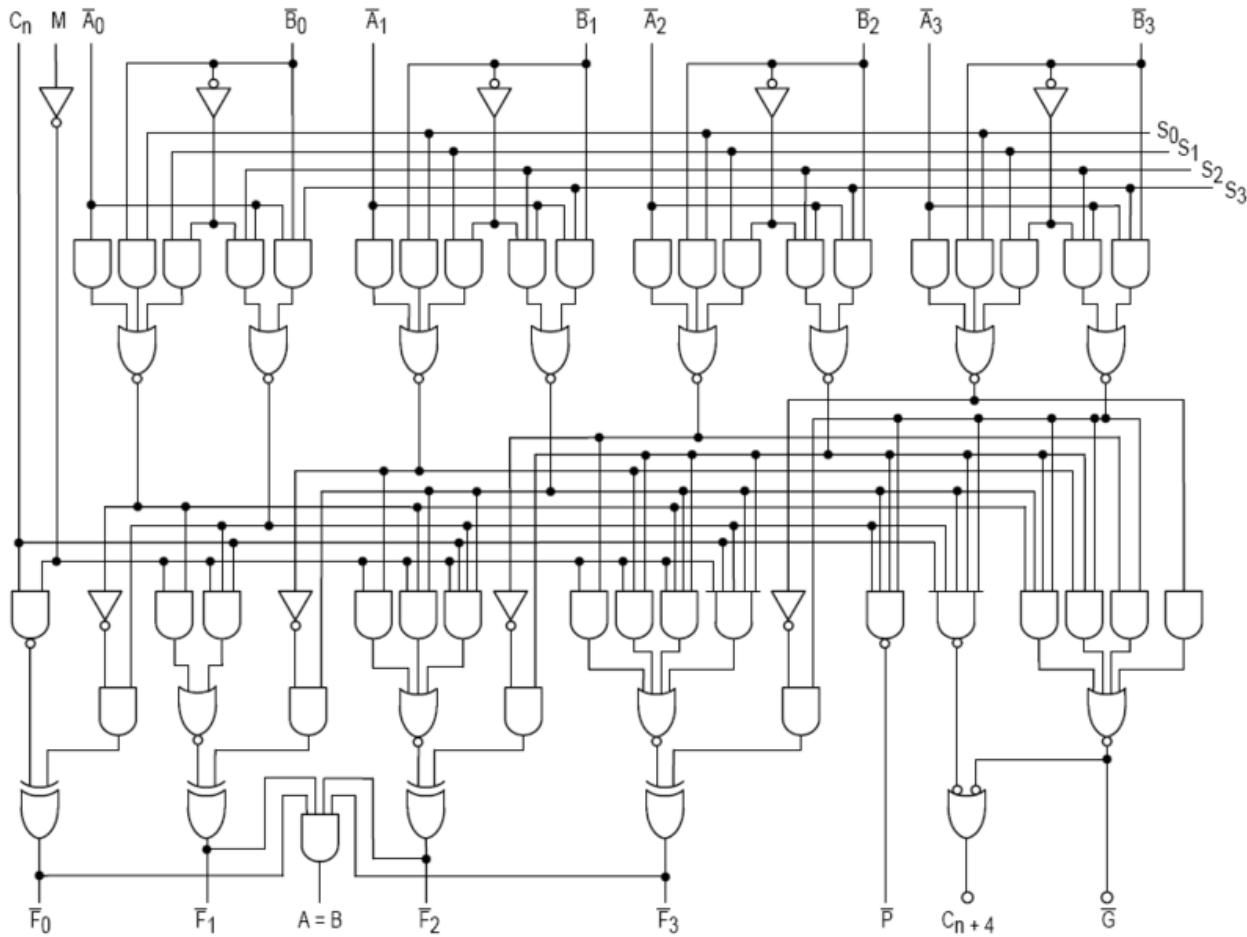
## ALU (Arithmetic logic unit)

Er komponenten i en processor som typisk foretager beregninger, logiske operationer og sammenligninger

Her en SN74A181N.



Her er [datasheet](#) for kredsen, og her skemaet for samme - bemærk de mange gates.



(Modul 149)

## Hukommelse

Kombinationen af logiske kredsløb (og typisk en clock) kan skabe hukommelse - der er mange forskellige typer.

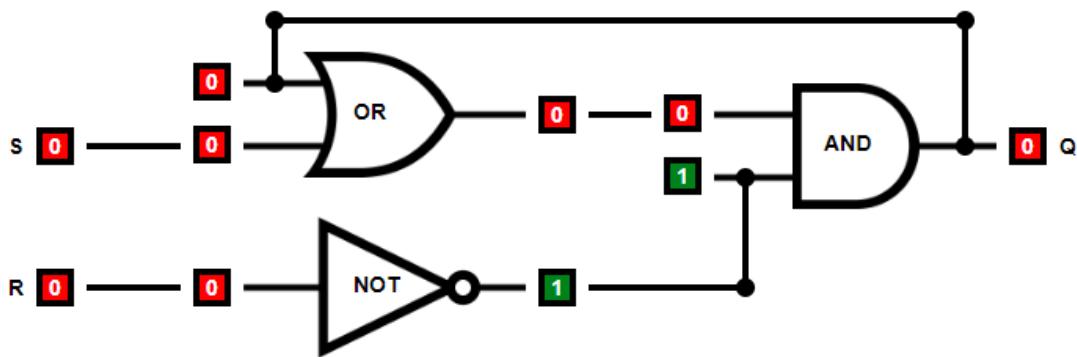
Kredsløb bruges til små registre, RAM og ROM mv

## OR Latch



## Set Reset AND-OR latch

En af de mere simple Set Reset latch'er er [SR AND-OR latch'en](#)



Den kan du lege med på [CS Demo](#)

## RAM intro

RAM er opbygget således, at en konkret adresse indeholder en værdi - så en 8 bit RAM kunne se således ud.

Adresse			Værdi		
2	10	16	2	10	16
00000000	0	0	00000000	0	00
00000001	1	1	00001100	12	0C
00000010	2	2	00000010	2	02
00000011	3	3	00011001	25	19
00000100	4	4	00000011	3	03
00000101	5	5	11001000	200	C8
00000110	6	6	00001010	10	0A
...	...	...	...	...	...
11111010	250	FA	00000000	0	00
11111011	251	FB	00000001	1	01
11111100	252	FC	01100011	99	63
11111101	253	FD	00010111	23	17
11111110	254	FE	00100001	33	21
11111111	255	FF	01100100	100	64

Når man benytter variabler i programmer er navnene på variablerne i virkeligheden blot en reference til en adresse i hukommelsen.

Prøv selv på <https://csdemo.cronberg.dk/ram.html>.

## Hjertet i en CPU (clock)

En CPU har en eller flere clock-kredsløb som sørger for at styre og time afvikling af operationer.

I sin helt simple form bruges en timer-chip eller en krystal.

## Hertz

Hertz mäter frekvens - 1 Hz = 1 slag i sekundet

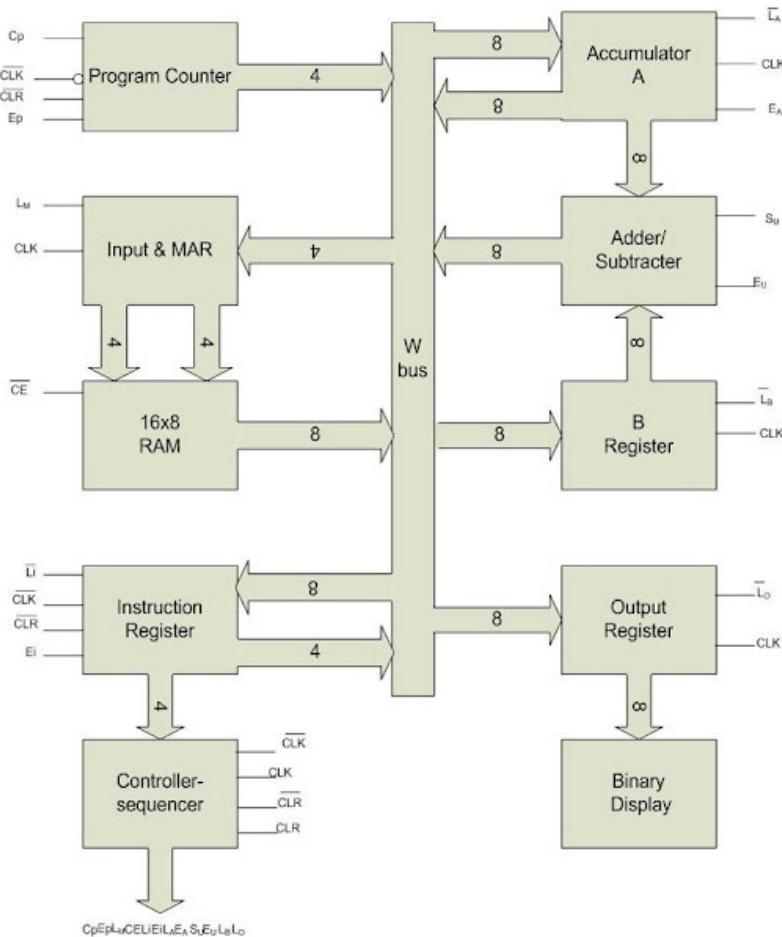
Forkortelse	Beskrivelse	hertz
Hz	hertz	1
kHz	kilohertz	1.000
mHz	megahertz	1.000.000
gHz	gigahertz	1.000.000.000

(Modul 151)

## En SAP (simple as possible) CPU

Skabt af Albert Paul Malvino i bogen Digital computer electronics

**Block Diagram of SAP-1**

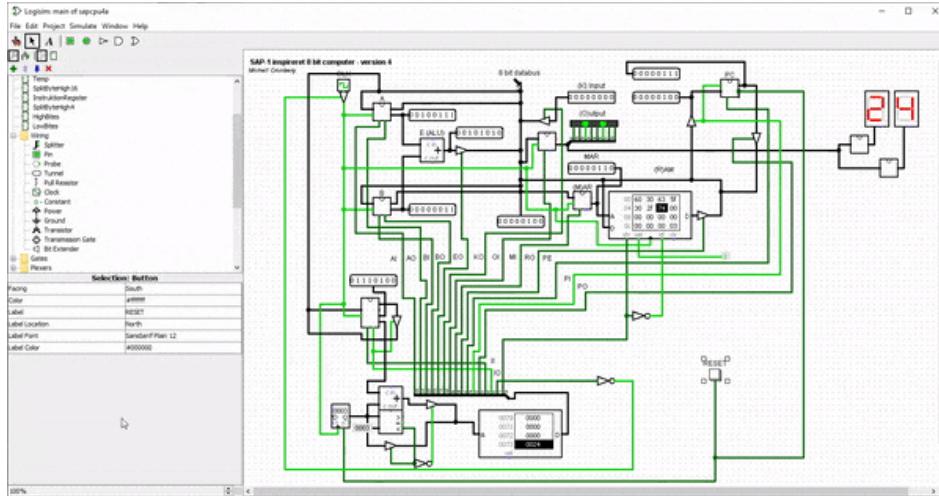


## Ref

- Ben Eater's 8 bit computer

## Michells SAP1 simulering

Du kan finde en simulering af SAP1 på <https://github.com/devcronberg/sap-cpu>



Det som er allervigtigst at forstå, er hvordan registrer og ALU kan påvirkes via kontrollinjer, og at disse kan påvirkes både manuelt samt gennem binære værdier.

## opkoder og maskinkode

På sitet findes andre versioner af CPU'en så det er nemmere at forstå

- Registrer
- ALU
- RAM
- Kontrollinjer
- Mikroinstruktioner
- Opkoder

Den sidste version af computeren introducerer Assembler som en direkte oversættelse af opkoder til assembler.

## Altair



## Ref

- Altair 8800
- Altair 8800 simulator
- Instruktion

## Læg 2 to tal sammen fra RAM

nr	kommando	værdi	kommentar
00000000	LDA	00111010	Load til A fra 128
00000001	-	10000000	
00000010	-	00000000	
00000011	MOV	01000111	Flyt A -> B
00000100	LDA	00111010	Load til A fra 129
00000101	-	10000001	
00000110	-	00000000	
00000111	ADD	10000000	Læg A og B sammen
00001000	STA	00110010	Flyt A til 130
00001001	-	10000010	
00001010	-	00000000	
00001011	JMP	11000011	Jump til 0
00001100	-	00000000	
00001101	-	00000000	

## Digirule2

Oprindeligt fra [KickStarter](#) - nu direkte hos Bradley Slattery



## Læg 2 to tal sammen fra RAM

nr	kommando	værdi	kommentar

nr	kommando	værdi	kommentar
00000000	COPYLR	00000011	copy 2 til ram pos 20
00000001	2	00000010	
00000010	20	00010100	
00000011	COPYLR	00000011	copy 2 til ram pos 21
00000100	2	00000010	
00000101	21	00010101	
00000110	COPYRA	00000110	copy ram pos 20 til a
00000111	20	00010100	
00001000	ADDRA	00001001	add ram pos 21 til a
00001001	21	00010101	
00001010	COPYAR	00000101	copy a til 255 (out)
00001011	255	11111111	
00001100	jump	00011100	uendeligt loop
00001101	12	00001100	

## Moderne processorer

- X86/X64 arkitektur
  - X86 instruktioner
- ARM arkitektur

(Modul 152)

## Kompilere og programmeringssprog

CPU niveau

- Maskinkode

Lavniveau

- Maskinsprog Assembler

Højniveau

- COBOL
- C-sprog
- .NET / JAVA
- Høj niveau sprog er typisk abstraktion på abstraktion på abstraktion

- Sprog der kompilere til et andet sprog der kompileres til et tredje sprog
- Kompilering til eksekvebar kode kaldes "kompilering" (compile) eller "byggesekvens" (build)
  - Kompilering til andre sprog kaldes "transpilering"

Prøv det selv på <https://csdemo.cronberg.dk/compile.html>.

## Programmeringssprog

- 1940: Assembly
- 1950-60: COBOL, FORTRAN, Basic
- 1970: Logo, Pascal, C, Smalltalk
- 1980: C++, Eiffel, Perl
- 1990: Python, Java, Delphi, JavaScript, Ruby
- 2000: C#, F#, Ruby on Rails
- 2010: Dart
- 2012: TypeScript

De mest populære programmeringssprog - se [StackOverflow](#)

En del danskere har bidraget med programmeringssprog

- c++ (Bjarne Stroustrup)
- c#, delphi, pascal (Anders Hejlsberg)
- php (Rasmus Lerdorf)
- dart (Lars Bak og Kasper Lund)

---

(Modul 153)

## Syntaks og instruktioner

- De fleste sprog er forskellige i syntaks
- Alle består af instruktioner af en eller anden art
  - Kildekode
  - Opmærkninger
  - Grafisk
- Instruktioner er defineret på flere måder
  - Linjeskift
  - Semikolon
  - Tabulering
- Programpointer
  - Flow i koden
  - Instruktion til instruktion
  - Linje til linje
  - Funktion til funktion

## Eksempler på syntaks

### Maskinkode

```
00000010 10000000 00000000 00000110
10000010 10000000 01111111 11111100
10001000 10000000 01000000 00000010
11001010 00000001 00000000 00000000
00010000 10111111 11111111 11111011
10000110 10000000 11000000 00000101
10000001 11000011 11100000 00000100
00000000 00000000 00000000 00010100
00000000 00000000 00001011 10111000
```

### Assembler

```

MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER PAGE  2

C000 8E 00 70  START  ORG    ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START  LDS    #STACK

*****+
* FUNCTION: INITA - Initialize ACIA
* INPUT: none
* OUTPUT: none
* CALLS: none
* DESTROYS: acc A
*****+
0013  RESETA EQU    $00010011
0011  CTLREG EQU    $00010001

C003 86 13  INITA  LDA A  #RESETA  RESET ACIA
C005 B7 80 04  STA A  ACIA
C008 86 11  LDA A  #CTLREG  SET 8 BITS AND 2 STOP
C00A B7 80 04  STA A  ACIA

C00D 7E C0 F1  JMP    SIGNON  GO TO START OF MONITOR

*****+
* FUNCTION: INCH - Input character
* INPUT: none
* OUTPUT: char in acc A
* DESTROYS: acc A
* CALLS: none
* DESCRIPTION: Gets 1 character from terminal
*****+
C010 B6 80 04  INCH   LDA A  ACIA    GET STATUS
C013 47          ASR A
C014 24 FA          BCC  INCH    SHIFT RDRF FLAG INTO CARRY
C016 B6 80 05  LDA A  ACIA+1  RECEIVE NOT READY
C019 84 7F  AND A  #87F    GET CHAR
C01B 7E C0 79  JMP   OUTCH   MASK PARITY
C01B 7E C0 79  RTS

*****+
* FUNCTION: INHEX - INPUT HEX DIGIT
* INPUT: none
* OUTPUT: digit in acc A
* CALLS: INITA
* DESTROYS: acc A
* Returns to monitor if not HEX input
*****+
C01E 8D F0  INHEX  BSR   INCH    GET A CHAR
C020 81 30  CMP A  #0    ZERO
C021 81 31  BSR   HEXERR  NOT HEX
C024 81 39  CMP A  #19   NINE
C026 2F 0A  BLE   HEXRTS  GOOD HEX
C028 81 41  CMP A  #'A
C02A 2B 09  BMI   HEXERR  NOT HEX
C02C 81 46  CMP A  #'F
C02E 2E 05  BGT   HEXERR
C030 80 07  SUB A  #7    FIX A-F
C032 80 0F  AND A  #$0F   CONVERT ASCII TO DIGIT
C034 39  RTS

C035 7E C0 AF  HEXERR  JMP    CTRL   RETURN TO CONTROL LOOP

```

## Basic

```

5 LET S = 0
10 MAT INPUT V
20 LET N = NUM
30 IF N = 0 THEN 99
40 FOR I = 1 TO N
45 LET S = S + V(I)
50 NEXT I
60 PRINT S/N
70 GO TO 5
99 END

```

## Cobol

```

OPEN INPUT sales, OUTPUT report-out
INITIATE sales-report

PERFORM UNTIL 1 <> 1
  READ sales
    AT END
      EXIT PERFORM
  END-READ

  VALIDATE sales-record
  IF valid-record
    GENERATE sales-on-day
  ELSE
    GENERATE invalid-sales
  END-IF
END-PERFORM

TERMINATE sales-report
CLOSE sales report-out
.

```

## Fortran

```

C AREA OF A TRIANGLE WITH A STANDARD SQUARE ROOT FUNCTION
C INPUT - CARD READER UNIT 5, INTEGER INPUT
C OUTPUT - LINE PRINTER UNIT 6, REAL OUTPUT
C INPUT ERROR DISPLAY ERROR OUTPUT CODE 3 IN JOB CONTROL LISTING
    READ INPUT TAPE 5, 601, IA, IB, IC
501 FORMAT (3IS)
C IA, IB, AND IC MAY NOT BE NEGATIVE
C FURTHERMORE, THE SUM OF TWO SIDES OF A TRIANGLE
C IS GREATER THAN THE THIRD SIDE, SO WE CHECK FOR THAT, TOO
    IF (IA) 777, 777, 781
781 IF (IB) 777, 777, 782
782 IF (IC) 777, 777, 783
783 IF (IA+IB-IC) 777,777,784
784 IF (IA+IC-IB) 777,777,785
785 IF (IB+IC-IA) 777,777,799
777 STOP 1
C USING HERON'S FORMULA WE CALCULATE THE
C AREA OF THE TRIANGLE
799 S = FLOATF (IA + IB + IC) / 2.0
    AREA = SQRT( S * (S - FLOATF(IA)) * (S - FLOATF(IB)) *
    + (S - FLOATF(IC)))
    WRITE OUTPUT TAPE 6, 601, IA, IB, IC, AREA
601 FORMAT (4H A= ,15H B= ,15H C= ,15,8H AREA= ,F10.2,
    + 13H SQUARE UNITS)
    STOP
    END

```

## Pascal

```

type
  a = array[1..10] of integer;
  b = record
    x : integer;
    y : char
  end;
  c = file of a;

```

## Python

```

prices = {'apple': 0.40, 'banana': 0.50}
my_purchase = {
    'apple': 1,
    'banana': 6}
grocery_bill = sum(prices[fruit] * my_purchase[fruit]
                   for fruit in my_purchase)
print 'I owe the grocer $%.2f' % grocery_bill

```

## Smalltalk

```

| rectangles aPoint collisions |
rectangles := OrderedCollection
  with: (Rectangle left: 0 right: 10 top: 100 bottom: 200)
  with: (Rectangle left: 10 right: 10 top: 110 bottom: 210).
aPoint := Point x: 20 y: 20.
collisions := rectangles select: [:aRect | aRect containsPoint: aPoint].

```

## C++

```

#include<iostream>
#include<vector>

int main()
try {
    std::vector<int> vec{3,4,3,1};
    int i{vec.at(4)};
}
//An exception handler, catches std::out_of_range
catch(std::out_of_range& e) {
    std::cerr<<"Accessing a non-existent element: "<<e.what()<<'\n';
}
catch(std::exception& e) {
    std::cerr<<"Exception thrown: "<<e.what()<<'\n';
}

```

## Java

```

package fibsandies;
import java.util.HashMap;

/**
 * This is an example of a Javadoc comment; Javadoc can compile documentation
 * from this text. Javadoc comments must immediately precede the thing being documented.
 */
public class FibCalculator extends Fibonacci implements Calculator {
    private static HashMap<Integer, Integer> memoized = new HashMap<Integer, Integer>();
    static {
        memoized.put(0, 0);
        memoized.put(1, 1);
    }
    /**
     * An example of a method written in Java, wrapped in a class.
     * Given a non-negative number FIBINDEX, returns
     * the Nth Fibonacci number, where N equals FIBINDEX.
     * @param fibIndex the index of the Fibonacci number
     * @return The Fibonacci number itself
     */
    public static int fibonacci(int fibIndex) {
        if (memoized.containsKey(fibIndex)) {
            return memoized.get(fibIndex);
        } else {
            int answer = fibonacci(fibIndex - 1) + fibonacci(fibIndex - 2);
            memoized.put(fibIndex, answer);
            return answer;
        }
    }
}

```

## JavaScript

```
var sum = function() {
  var i, x = 0;
  for (i = 0; i < arguments.length; ++i) {
    x += arguments[i];
  }
  return x;
}
sum(1, 2, 3); // returns 6
```

## F#

```
/// Iteration using a 'for' Loop
let printList lst =
  for x in lst do
    printfn "%d" x

/// Iteration using a higher-order function
let printList2 lst =
  List.iter (printfn "%d") lst

/// Iteration using a recursive function and pattern matching
let rec printList3 lst =
  match lst with
  | [] -> ()
  | h :: t ->
    printfn "%d" h
    printList3 t
```

## Ruby

```
#!/usr/bin/ruby

$i = 0
$num = 5

while $i < $num do
  puts("Inside the loop i = #$i" )
  $i +=1
end
```

---

(Modul 154)

## Grundlæggende programmering

De fleste programmeringssprog har features til fælles

- Variabler
  - Opbevaring af midlertidige værdier
    - Jf tidlige sektion om RAM
- Flow instruktioner
  - løkker
    - loop et givet antal gange
  - betingelser
    - hvis et eller andet er sandt så... ellers....
- Funktioner
  - Samling af instruktioner der nemt kan afvikles
    - med/uden argumenter
    - med/uden returværdi

---

(Modul 155)

## Typer af applikationer

- Konsol
- Lokale GUI applikationer

- Web applikationer
  - Web/Desktop
    - Electron
    - SOA
  - Service-Oriented Architecture
  - Mobile applikationer
  - Embedded
    - PLC
    - Espruino, Tessel.IO
    - Raspberry Pi, Netduino
  - API / SDK
    - application programming interface
  - Test
    - Unit test
    - UI Test
    - Integration test
- 

(Modul 156)

## Udviklingsværktøjer

- I de fleste programmeringssprog er kildekoden ren tekst
    - Notepad
  - Udviklingsmiljøer giver den bedste produktivitet
    - "Visuel" kode
    - Løbende baggrundskompilering
    - Hjælp til kodeskrivning
    - Integration af forskellige værktøjer
    - Test
    - Deployment
  - Nogle af de kendte
    - Visual Studio (.NET og meget meget)
    - Visual Studio Code
    - Eclipse (java +)
    - Netbeans (java, c++, mv)
    - gEdit (linux)
- 

(Modul 157)

## Version/source control

Source control - også kaldet version styring - benyttes primært til styring af rettelser i kode. I dag kan de fleste systemer håndterer

- Rettelser
  - Hvad er rettet
  - Hvem har rettet
  - Hvornår er der rettet
  - Hvorfor er der rettet
- Versionstyring
  - Branching
- Projektstyring
- Fejlrapportering
- Styring og automatisering af build

Af kendte systemer findes

- Git
- Subversion

- Mercurial

og de fleste vælge at have sin "kode i skyen" - eksempelvis hos

- GitHub
- BitBucket
- AWS CodeCommit
- Microsoft Team Foundation Server

Dette materiale er udviklet af Michell Cronberg (michell@cronberg.dk) og må ikke benyttes uden tilladelse.