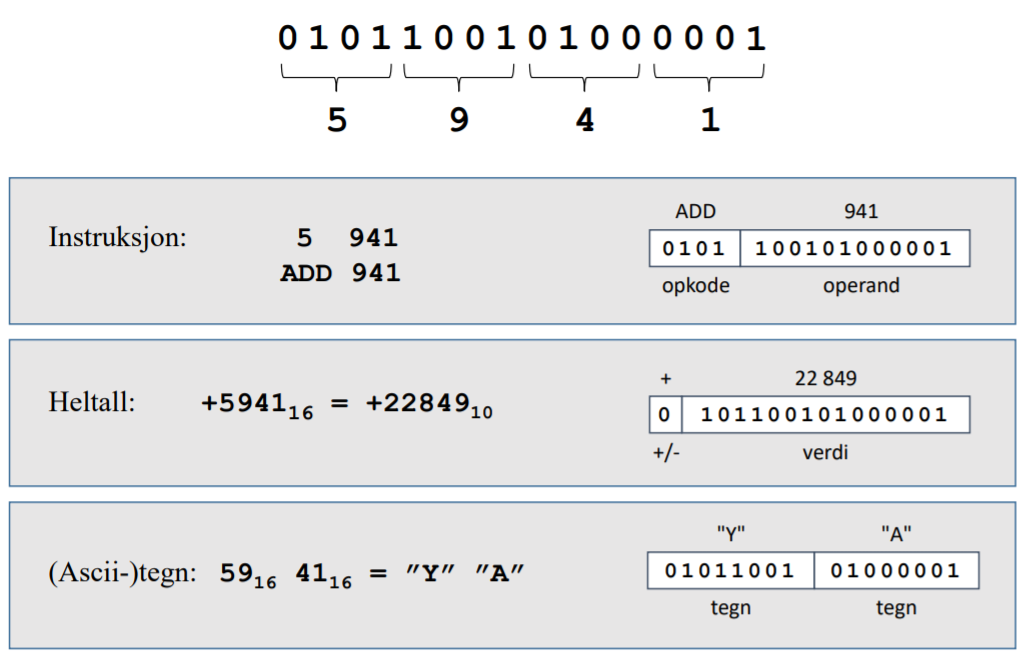
Datarepresentasjon og aritmetikk

**Bitmønster**



**Viktige datatyper**

Logiske (boolske) data og operasjoner

* Representasjon
* Operasjoner og operatorer: AND, OR, XOR, NOT …
* Shift og rotasjon
* Maskering (Finn ut hva det er)

Tegn og strenger

* ASCII og Unicode
* Lengde
* Operasjoner

Heltall og flyttall

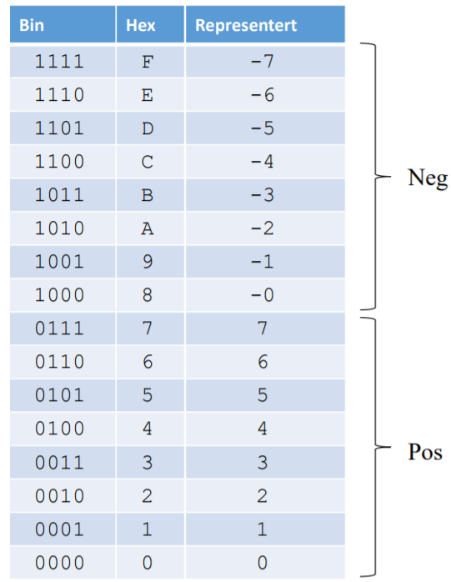
# Heltall

**Heltall uten fortegn**

Vi bruker bit-verdien direkte. Størrelsen på tallet avhenger av antall bits. F.eks. 16 bits 0 til 65.535.

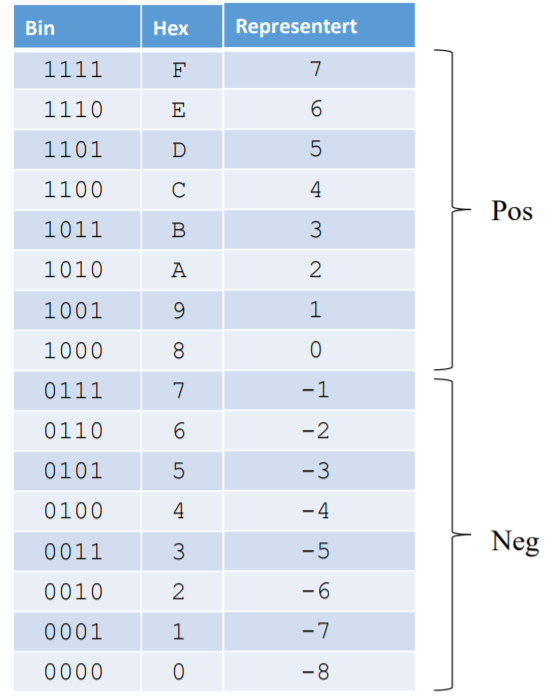
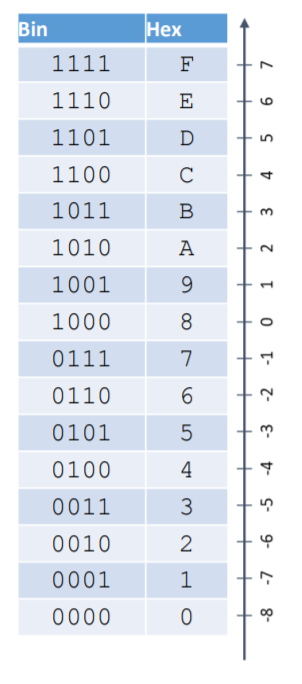
**Heltall med fortegn**

Vi bruker halvparten av bitmønstrene til positive tall og resten til negative. Finnes flere muligheter for hvordan.

*Metode 1. fortegnsbit*

0 er +, og 1 er -. Resten av bitmønsteret angir verdi. Ulemper er at to verdier for 0, matematikken stemmer ikke. Derfor brukes ikke denne metoden så mye.

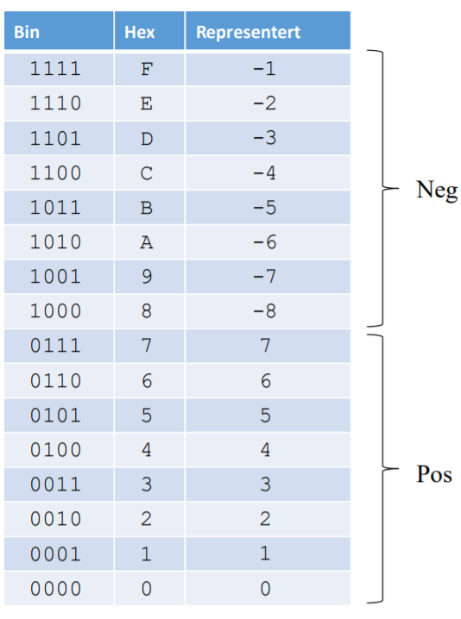
*Metode 2. forskyving (bias)*



Nederste halvdelen til negative tall og den øverste halvdelen til positive tall. Vi forskyver verdiene en halv skala.

Ulemper er rar verdi for null, ingen verdi stemmer med eget bitmønster.

*Metode 3. toer-komplement*



Den vanligste metoden som blir brukt. Kan virke tungvint ved første øyekast.

-1 er alltid representert med bare 1-ere. Det minste negative tall starter med 1 og resten er 0-ere.

*MSB = 0 for positive tall.*

*MSB = 1 for negative tall.*

MSB kalles ofte fortegnsbiten, eller «mest signifikante bit».

**Hvordan finne representasjonene til negative tall?**

For negative tall må vi regne for å finne sammenhengen mellom representasjon og verdi.

Tre metoder:

1. Komplementere og addere 1 (enkelt for prosessor)
2. Bruk aritmetikk (relativt enkelt for mennesker) ***LES SELV***
3. Regne-trick for hurtig omgjøring (veldig enkelt for mennesker)

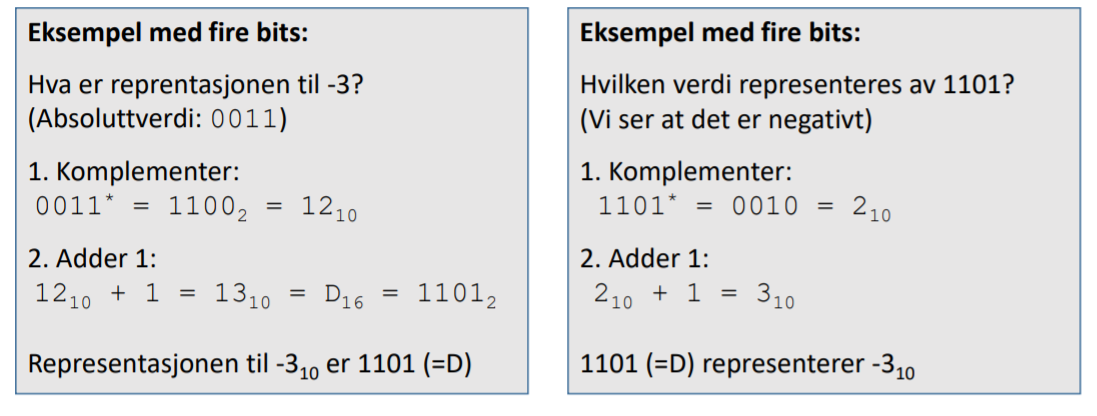
*Metode 1. Komplementer og adder*

Fremgangsmåte:

1. Komplementerer bitmønsteret. Det vil si at 0-er blir 1-er og 1-er blir 0-er.
2. Legg til 1.

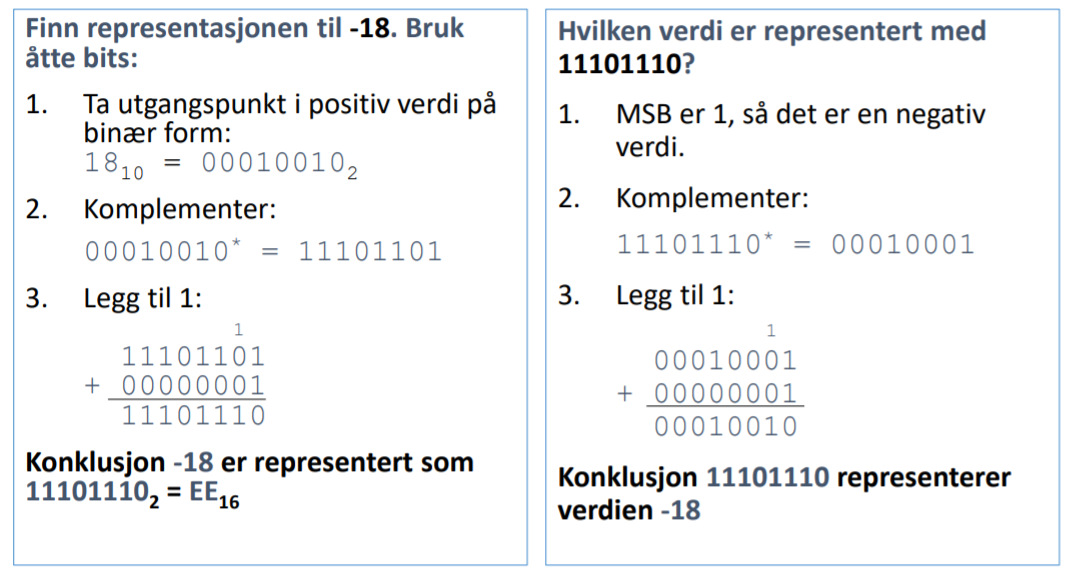
Metoden virker likedan begge veier fra verdi til representasjon og fra representasjon til verdi.

**Eksempel 1**



**Eksempel 2**

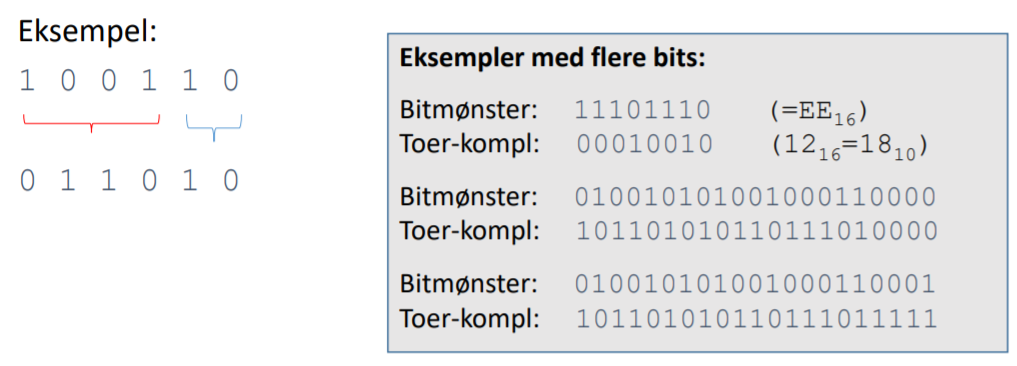
Hvilket tallsystem man bruker når man legger til 1 har ikke noe å si. Svaret skal bli det samme.



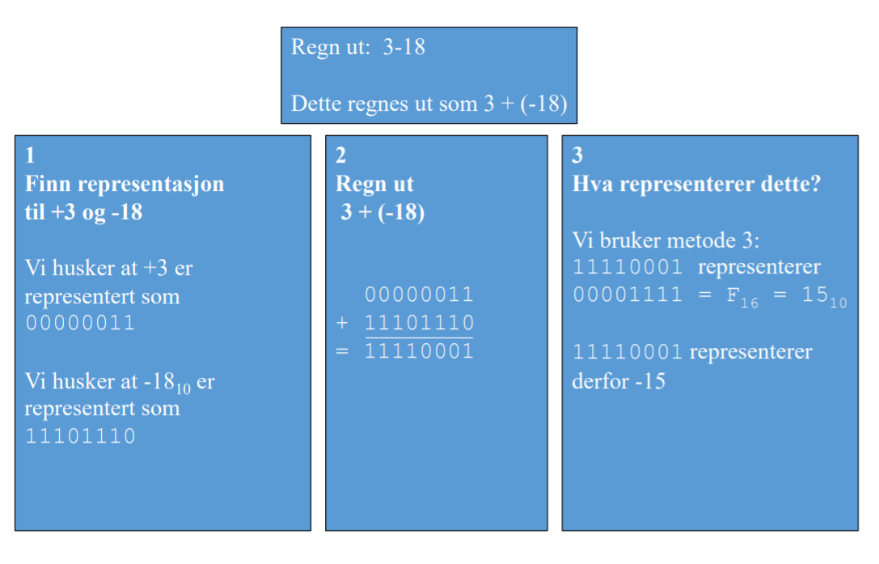
*Metode 3. Hurtig trick for toer-komplement*

1. Start bakfra og behold alle bits uendret til og med første ener.
2. Komplementer resten av bitene.

*Bruk på eksamen? Si ifra at denne metoden brukes!*



Addisjon og subtraksjon med denne metoden:



**Overflow**

Resultater kan bli større (eller mindre) enn det største (minste) tallet som kan representeres med det antll bits vi bruker. Dette kalles «overflow».

Eksempel:

8 bits aritmetikk: Resultatet må være mellom -128 ..127

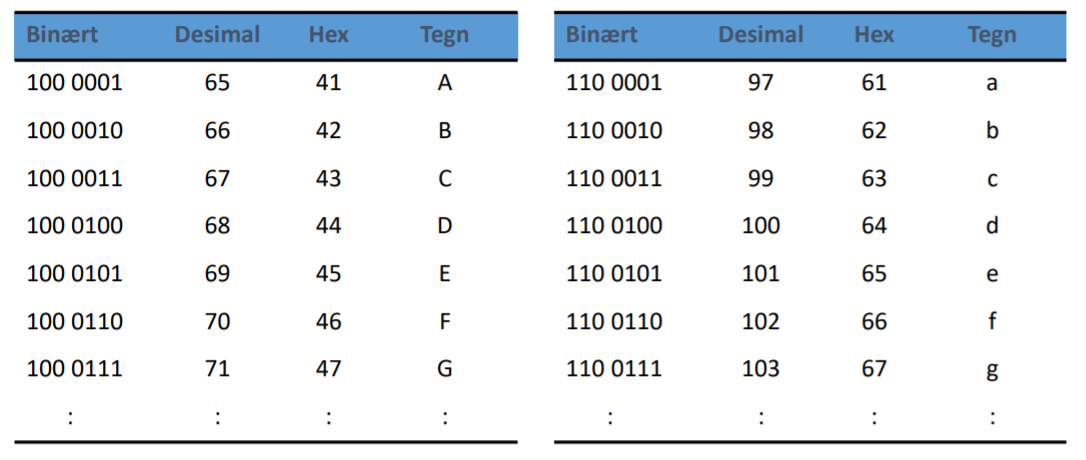
100 + 100 gir «overflow».

# Tegn og strenger

**Tegn**

**ASCII**: American Standard Code for Information Interchange. Den opprinnelige ASCII-standarden er 7-bits kode. Utvidet ASCII er 8-bits.

**UNICODE:** 16-bits, hvert tegn er 2 byte. Tar derfor mer plass i minnet. Kan representere 65536 ulike tegn. Blant disse er for eksempel hebraiske og japanske (og norske) tegn. UNICODE brukes blant annet i Java.



**Strenger**

Strenger varierer i lengden. Vi må lagre info om lengden sammen med strengen.

Dette kan gjøres på ulike måter.

* Java: strenger av 16-bits UNICODE-teng.
  + Lengden som lagres som et 32 bits heltall (med fortegn)
* C og C++: strenger av ASCII-tegn
  + En streng avsluttes med ASCII-verdien 0 0.