# Oblig 1 - LMC

## Forklaring på kode:

Jeg startet enkelt, slik det var forklart i obligen. Først kodet jeg det slik at den hentet n inputs, deretter bygget jeg mer og mer på koden, helt til den gjorde alt obligen ba om.

**FORKLARING LINJE FOR LINJE:**

(Jeg følger linjer slik som i innlevert kode, der linje 1-6 er kommentarer, og første kode kommer på linje 7: loop INP)

I linje 54-64 lagres forskjellige verdier i minnet som utgjør enten tall eller som senere skal skrives ut som ASCII tegn. Her er det brukt DAT for å enkelt lagre verdiene, slik at det enkelt kan hentes ut / lagres som nye verdier, på en enkel og effektiv måte. Ved bruk av etiketter (slik vi ser i linje 54: «input») trenger jeg heller ikke velge adresse selv, koden vil plassere verdien der det i minnet er plass. Om dermed verdien er lagret i adresse 10 i minnet, og koden blir lenger, vil LMC automatisk flytte verdien til neste ledige plass i minnet og også endre adressenummeret over alt i koden der denne verdien skal brukes. Dette gjør det enklere å kode, og minimerer feil.
Etiketter vil altså oppdatere alle plassene i koden som bruker etikettet, dersom minne-adressen skulle endres.
Linje 54-64 gjør dette:

```
input   <— 0
sum     <— 0
n       <— 0
quot    <— 0 //quotient
one     <— 1
=       <— 61
        osv…
```

På linje 59-64 plasseres verdier i minnet som senere skal printes som ASCCII tegn.

Linje 7-17 er en while-loop, BRZ hopper til etikett «forts» dersom INP = 0:

```
while INP>0 do:
        input   <— INP
        sum     <— INP + sum
        n       <— n + 1
```

Så fort inputet fra bruker er 0, vil BRZ på linje 10 hoppe til etikett «forts». BRZ vil altså få LMC til å hoppe ut av loopen, til linje 20.
Hittil har koden tatt inn n inputs, og lagt de sammen som sum. Både sum og n er lagret på hver sin adresse i minnet.
Nå gjenstår det kun å skrive ut summen, antall inputs n, og gjennomsnittet d.
For å gjøre det ryddig, startet jeg med å skrive ut verdiene vi allerede har: summen (a=…) og n (n=…), og også «d=«, dette skjer i linje 20-39.
Her hentes (med LDA) altså først ut verdien i minnet som med kommandoen OTC vil skrive ut ASCII-tegnet «a». Det samme gjøres for «=«. Så brukes LDA og OUT til å skrive ut tallet lagret i adressen «sum» i minnet. Samme gjøres med n=…
Samme med «d=«.
Før vi kan skrive ut verdien d, må denne regnes ut, det skjer på linje 40-50.

Her er det en loop til. BRA startL2, hopper direkte til linje 46, og loopen starter herifra.
Det starter med at vi trekker fra n fra summen.
«BRP loop2» i linje 49, vil hoppe tilbake til linje 42 så lenge verdien i Accumulatoren (altså svaret fra sum-n) er større eller lik 0.
Er svaret større eller lik 0 vil vi starte fra linje 42, og nå øke quot (kvotienten) med en, før vi igjen trekker n fra sum og sjekker om det nye svaret igjen er større eller lik 0. Fortsetter slik fram til svaret er negativt.

```
sum <— sum-n
while sum >= 0:
        sum <— sum - n
        quot <— quot + 1
```

Så fort sum < 0 vil vi altså hoppe ut av while-løkken.
Vi printer så ut quot. Quot er kvotienten, og verdien som viser hvor ofte n går i summen. Det er dermed gjennomsnittet til summen av alle inputs brukeren har gitt.
Siste instruksjon som LCM gjennomfører (linje 53) er HLT, denne stopper maskinen.

For å lage fin utskrift i koden har jeg brukt ASCII-tegnet for \n (10) for å skrive ut neste output på ny linje. Dette er kun brukt bak ASCII-tegn, fordi det etter OUT print allerede automatisk skrives på ny linje ved neste output uavhengig om det er OUT eller OTC. Mens det etter OTC trengs ASCII-tegnet \n for å starte ny linje ved neste output.