Gruppetime 8
16.10.2025



Sørg for at alle sitter ved siden av noen

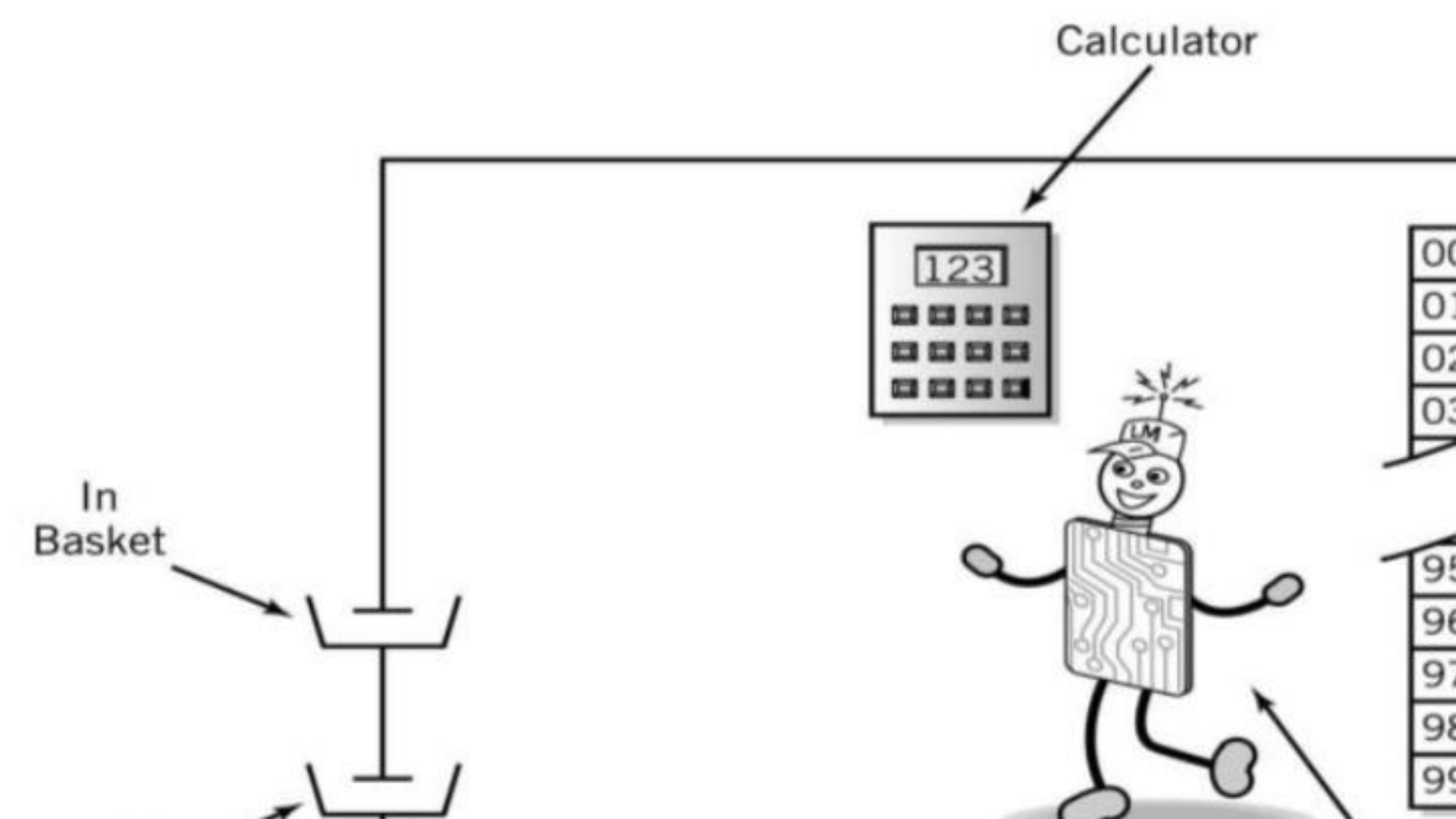


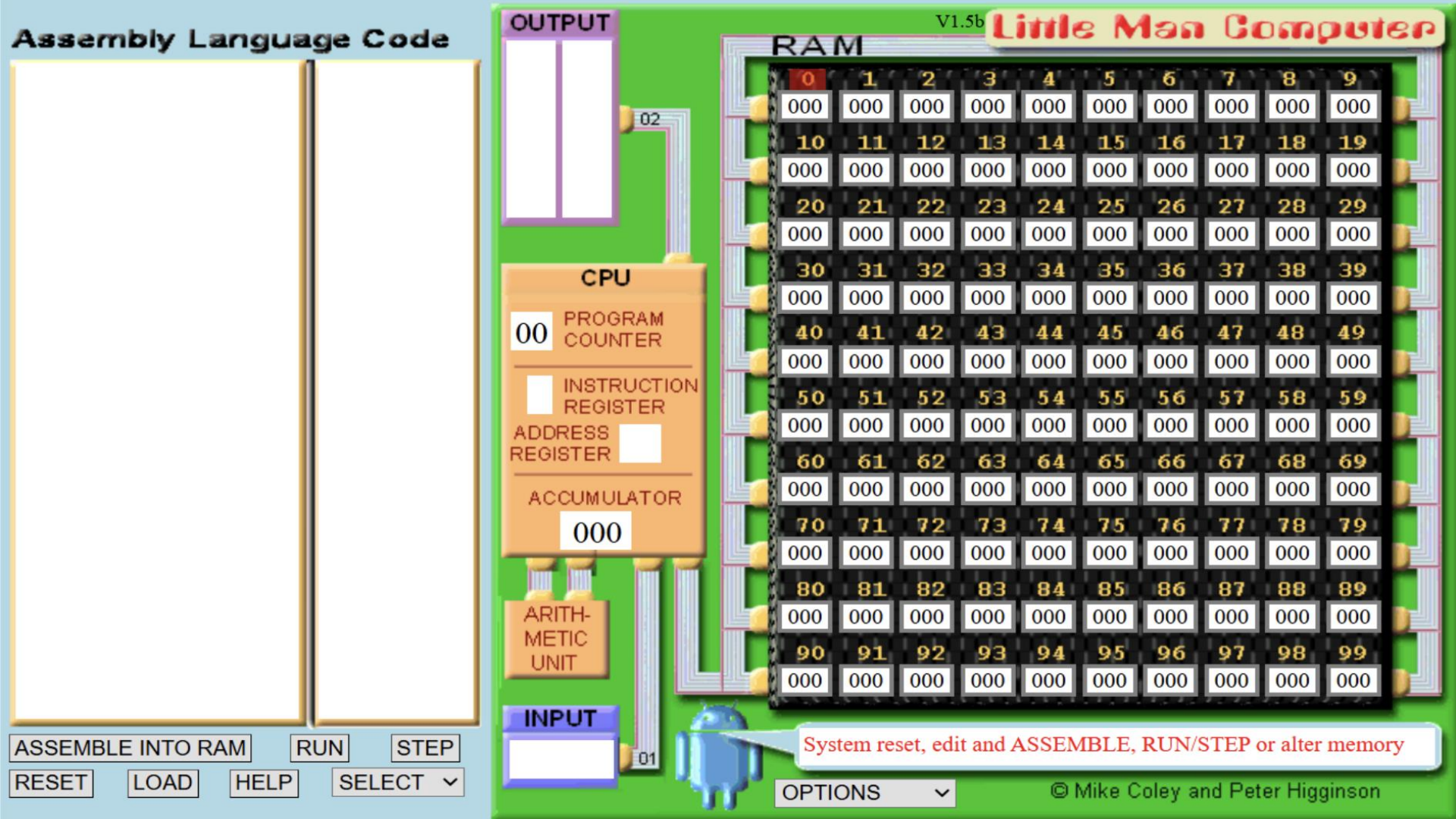
Velkommen til gruppetimen! ☺


LMC
intel
Core i7

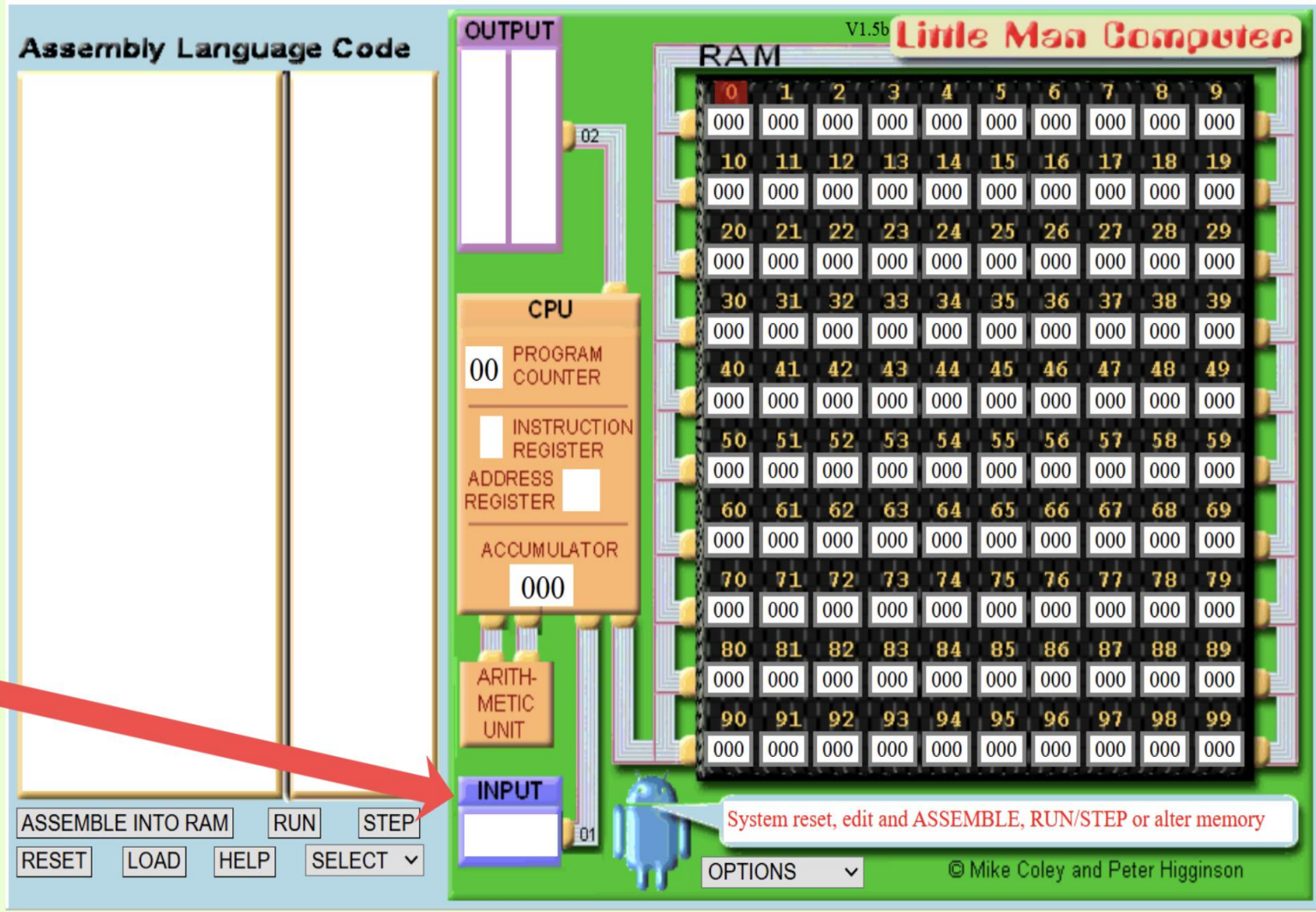
Praktisk info

- IN1020 universet: programmeringsoppgaven ligger ute på satelitten, lurt å komme i gang allerede nå!
- Hvis vanlig obliggløp: kan leveres i grupper! Registrer gruppa i devilry. 1-4 personer.
- Kan ikke levere blankt denne gangen. Kriterie for godkjenning: gjøre en ordentlig innsats, men trenger ikke å få alt riktig.
- Alle i gruppa skal kunne forstå/forklare hele innleveringen.


Calculator
123
In
Basket

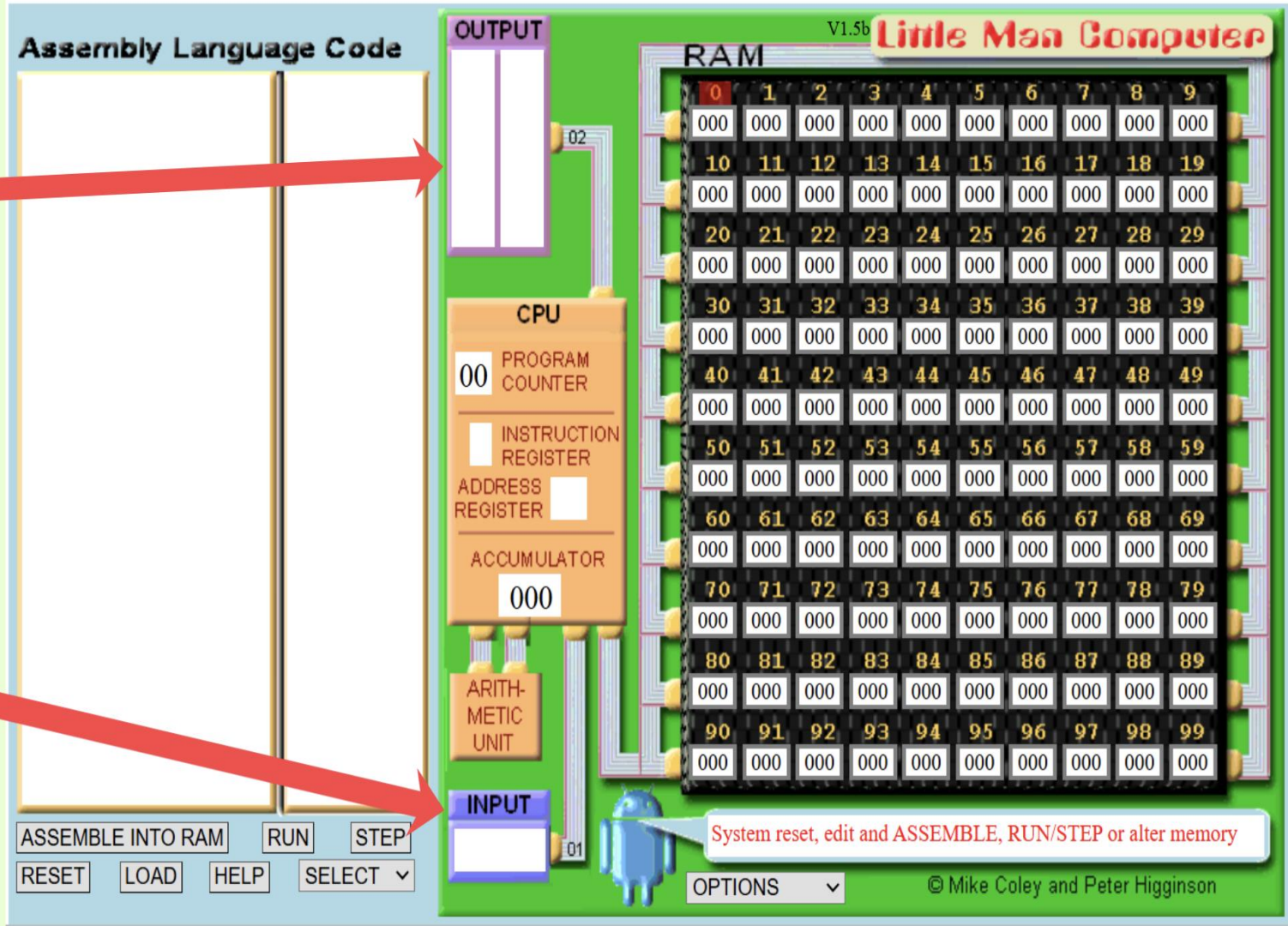

Assembly Language Code
OUTPUT
V1.5b
Little Man Computer
RAM
0 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99
000
02
CPU
00
PROGRAM COUNTER
INSTRUCTION REGISTER
ADDRESS REGISTER
ACCUMULATOR
000
ARITH-METIC UNIT
INPUT
01
System reset, edit and ASSEMBLE, RUN/STEP or alter memory
ASSEMBLE INTO RAM
RUN
STEP
RESET
LOAD
HELP
SELECT
OPTIONS
© Mike Coley and Peter Higginson

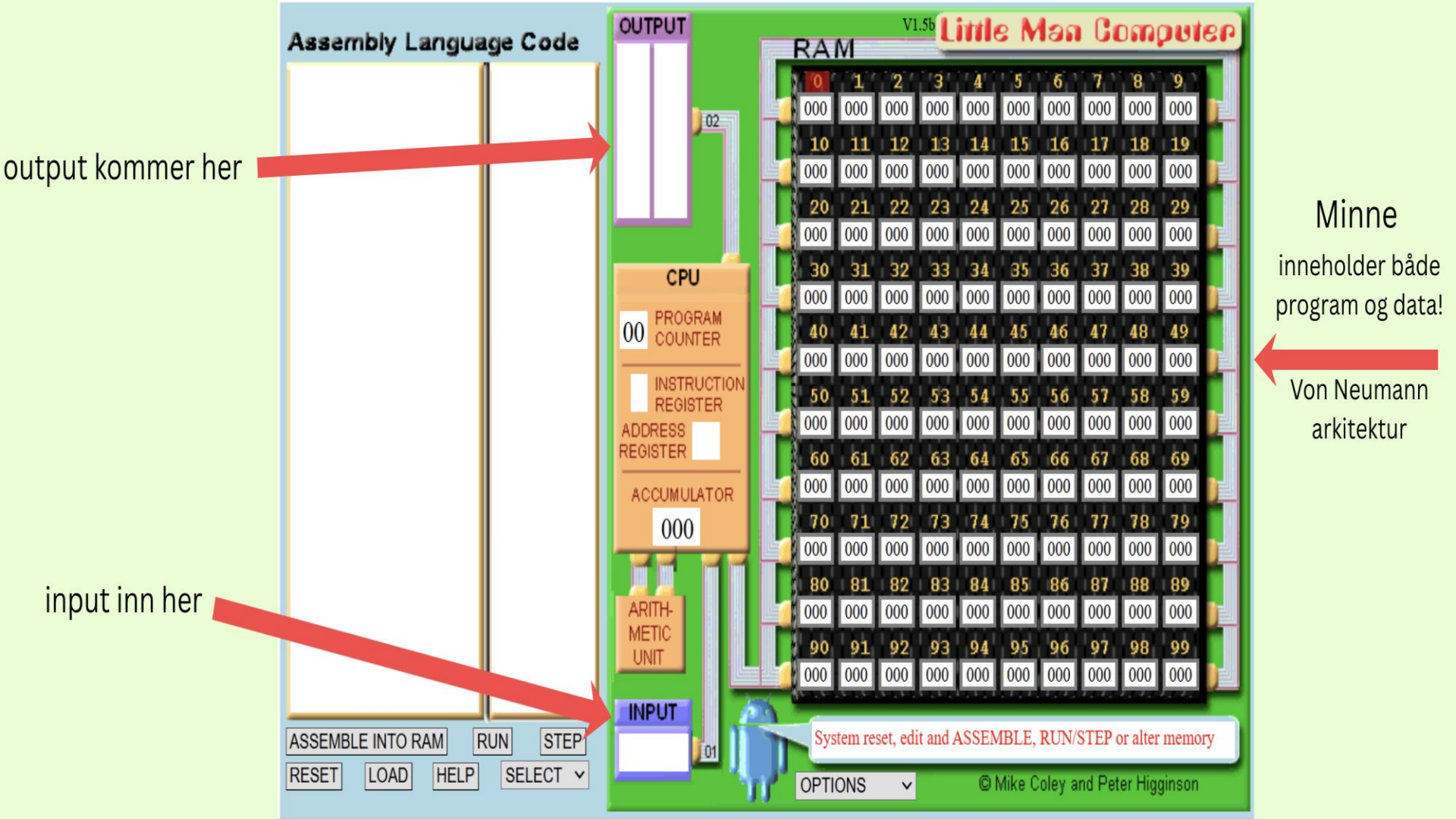
input inn her

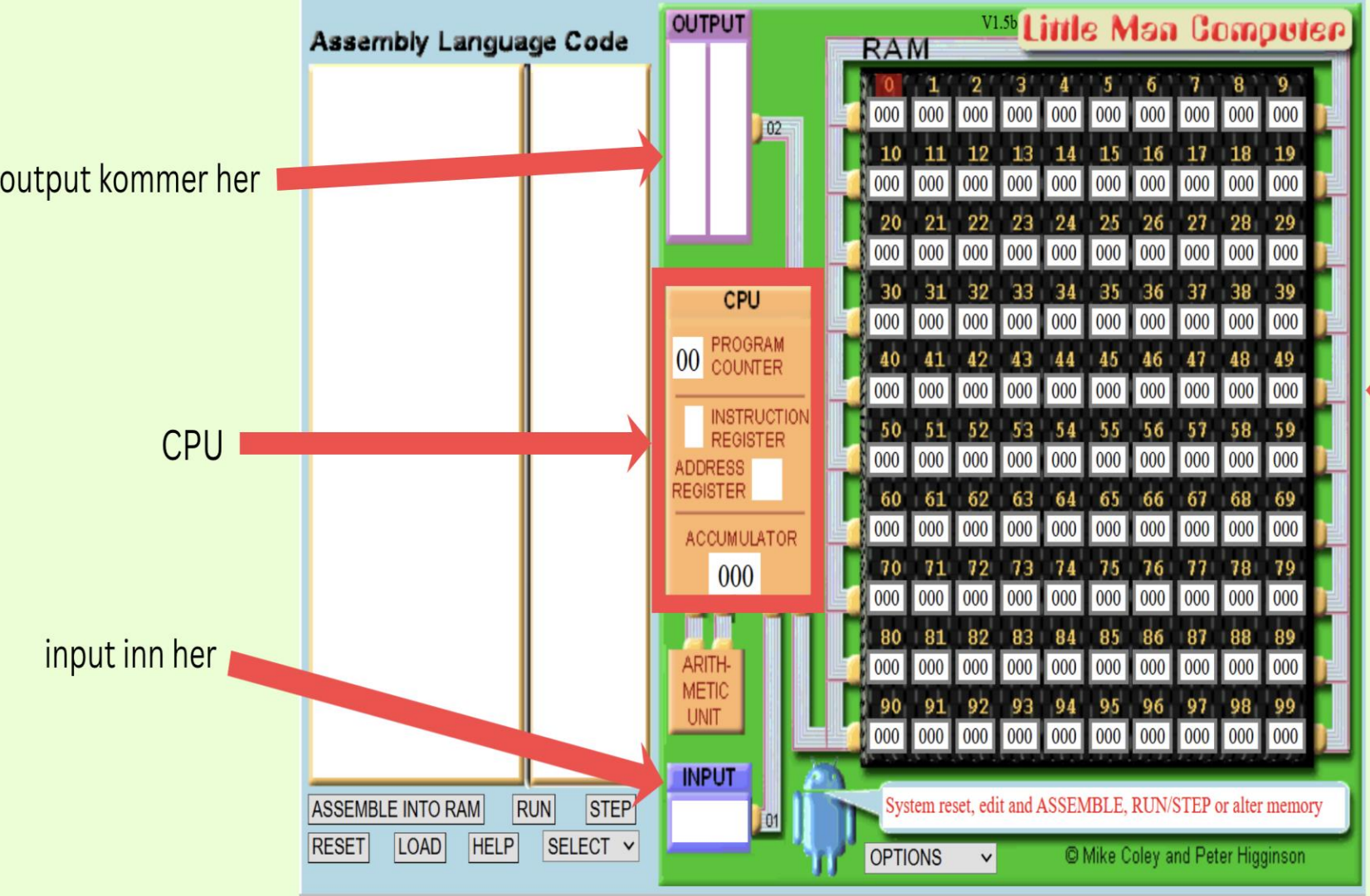


output kommer her

input inn her




output kommer her
input inn her
Assembly Language Code
OUTPUT
V1.5b
Little Man Computer
RAM
CPU
00
PROGRAM COUNTER
INSTRUCTION REGISTER
ADDRESS REGISTER
ACCUMULATOR
000
ARITH-METIC UNIT
INPUT
02
01
ASSEMBLE INTO RAM
RUN
STEP
RESET
LOAD
HELP
SELECT
OPTIONS
System reset, edit and ASSEMBLE, RUN/STEP or alter memory
© Mike Coley and Peter Higginson
Minne
inneholder både program og data!
Von Neumann arkitektur


Assembly Language Code
OUTPUT
V1.5b
Little Man Computer
RAM
CPU
00 PROGRAM COUNTER
INSTRUCTION REGISTER
ADDRESS REGISTER
ACCUMULATOR
000
ARITH-METIC UNIT
INPUT
ASSEMBLE INTO RAM
RUN
STEP
RESET
LOAD
HELP
SELECT
System reset, edit and ASSEMBLE, RUN/STEP or alter memory
OPTIONS
© Mike Coley and Peter Higginson
output kommer her
CPU
input inn her
Minnet
inneholder både program og data!
Von Neumann arkitektur

CPUen

“Hjernen”. Sørger for utføringen av alle operasjonene. Den består av:

- Program counter - forteller den lille mannen hvilken postkasse hans neste instruksjon er i. Øker for hver instruksjon utført.

- Kontrollogikk (instruction & adress *register*)

- *Instruksjonsregisteret* inneholder den gjeldende instruksjonen som utføres.
- *Adresseregisteret* inneholder adressen til postkassen som inneholder dataene eller instruksjonen som skal behandles.

- Accumulator - Den holder verdier midlertidig under programkjøring.

- Arithmetic Unit - en simpel kalkulator som kan håntere 3 sifrer

Minnet

En “mail box”

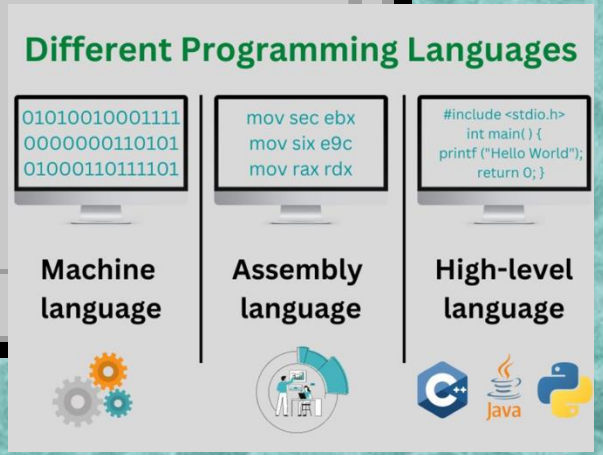
- Vi kan hente og lagre ting i minnet, både instrukser og data.
- Minnet i LMC er veldig liten og enkel. 100 minneplasser.
  - ikke forveksle minnelokasjon med minneinnhold
- Innholdet i hver minnelokasjon er begrenset til maks 3 siffre. Ingen negative verdier eller tall over 3 siffrer.

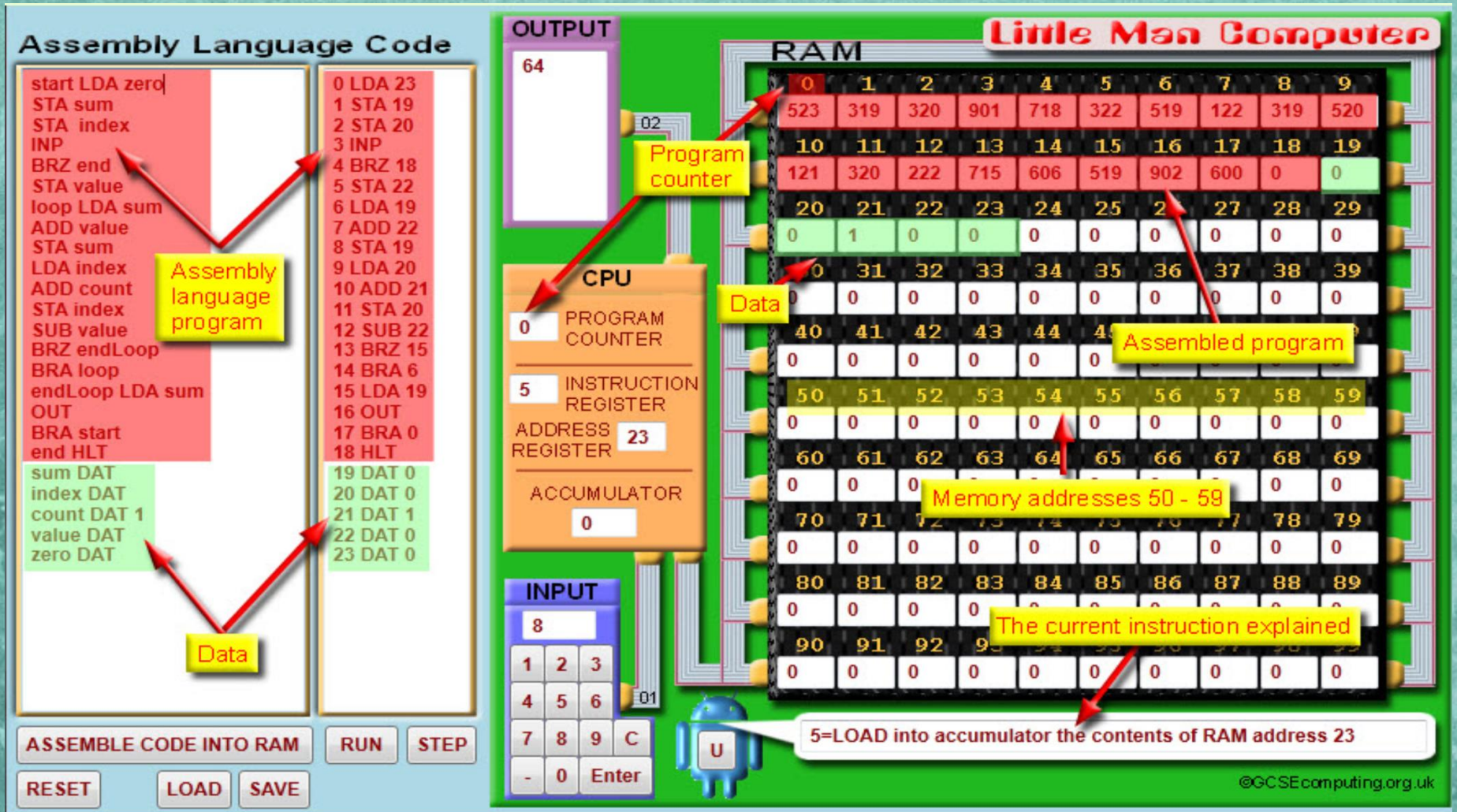
- Første siffer: instruksjonskoden, sier hvilken operasjon LMC må gjøre.
- Andre og tredje siffer: adressen i minnet, hvor LMC skal operere (minnelokasjon).



Assembler

- Maskinkode er kløynete, lettere å jobbe med assembler. Instruksjonene er mer intuitive :)
- Det er en **lavnivåprogrammeringsspråk** som direkte korresponderer med maskinspråksinstruksjonene en datamaskin kan utføre.
- Direkte oversettelse numerisk → tekstlig representasjon




Assembly Language Code
start LDA zero
STA sum
STA index
INP
BRZ end
STA value
loop LDA sum
ADD value
STA sum
LDA index
ADD count
STA index
SUB value
BRZ endLoop
BRA loop
endLoop LDA sum
OUT
BRA start
end HLT
sum DAT
index DAT
count DAT 1
value DAT
zero DAT
0 LDA 23
1 STA 19
2 STA 20
3 INP
4 BRZ 18
5 STA 22
6 LDA 19
7 ADD 22
8 STA 19
9 LDA 20
10 ADD 21
11 STA 20
12 SUB 22
13 BRZ 15
14 BRA 6
15 LDA 19
16 OUT
17 BRA 0
18 HLT
19 DAT 0
20 DAT 0
21 DAT 1
22 DAT 0
23 DAT 0
Assembly language program
Data
ASSEMBLE CODE INTO RAM
RUN
STEP
RESET
LOAD
SAVE
OUTPUT
64
02
Program counter
CPU
0
PROGRAM COUNTER
5
INSTRUCTION REGISTER
ADDRESS REGISTER
23
ACCUMULATOR
0
INPUT
8
1 2 3
4 5 6
7 8 9 C
- 0 Enter
01
U
RAM
Little Man Computer
0 1 2 3 4 5 6 7 8 9
523 319 320 901 718 322 519 122 319 520
10 11 12 13 14 15 16 17 18 19
121 320 222 715 606 519 902 600 0 0
20 21 22 23 24 25 26 27 28 29
0 1 0 0 0 0 0 0 0 0
30 31 32 33 34 35 36 37 38 39
0 0 0 0 0 0 0 0 0 0
40 41 42 43 44 45
0 0 0 0 0 0
50 51 52 53 54 55 56 57 58 59
0 0 0 0 0 0 0 0 0 0
60 61 62 63 64 65 66 67 68 69
0 0 0 0 0 0
70 71 72 73 74 75 76 77 78 79
0 0 0 0 0 0 0 0 0 0
80 81 82 83 84 85 86 87 88 89
0 0 0 0
90 91 92 93
0 0 0 0 0 0 0 0 0 0
Assembled program
Memory addresses 50 - 59
The current instruction explained
5=LOAD into accumulator the contents of RAM address 23
©GCSEcomputing.org.uk

Alle instruksjonene i LMC med assemblernotasjon

Kode	Navn	Beskrivelse
0	HLT	Stopper eksekveringen
1	ADD	Adderer verdien i angitt minnelokasjon med akkumulatoren
2	SUB	Subtraherer verdien i angitt minnelokasjon med akkumulatoren
3	STA	Lagrer akkumulatoren i angitt minnelokasjon
4	-	Ikke i bruk
5	LDA	Henter verdi fra minnet til akkumulatoren
6	BRA	Hopper til angitt adresse
7	BRZ	Hopper hvis akkumulatoren er 0
8	BRP	Hopper hvis akkumulatoren er ≥ 0
901	INP	Leser verdi fra input, og legger svaret i akkumulatoren
902	OUT	Skriver ut verdien i akkumulatoren
922	OTC	Skriver ut ASCII-tegn (ikke i boka)

Instruksjoner i LMC

• Programmet lagres pa numerisk form i minnet, som en sekvens av instruksjoner.

Hver instruksjon er et tall 0 - 999 der

- 1. siffer angir instruksjons koden og
- 2. 0g 3. siffer angir en adresse i minnet (om det er aktuelt)

• Eksempel:

- 542 betyr: Hent verdien i adresse 42 i minnet og legg den i akkumulator registeret. (Kode 5 angir at prosessoren skal hente noe i minnet)
- LMC har 9 forskjellige instruksjoner

Instruksjoner i LMC

- **HLT:**
  - Instruksjonen HLT ("Halt" også kalt COB = "Coffee Break" i boka) har kode 000. Instruksen stopper kjøring
- **INP:**
  - Instruksjonen INP ("Input", kalt IN i boka) har kode 901. Den ber brukeren skrive et tall, og dette tallet legges i akkumulator registeret
- **OUT:**
  - Instruksjonen OUT ("Output") har kode 902. Den skriver ut verdien i akkumulatoren

Instruksjoner i LMC

• STA:

• Instruksjonen STA ("Store accumulator", kalt STO i boka) har kode 3xx, og den kopierer verdien i akkumulatoren til adresse xx i minnet.

• LDA:

• Instruksjonen LDA ("Load accumulator") har kode 5xx, og den kopierer verdien i lokasjon xx til akkumulatoren.

Instruksjoner i LMC

- **ADD:**
  - Instruksjonen ADD har kode 1xx og adderer verdien i adresse x i minnet til akkumulatoren
- **SUB:**
  - Instruksjonen SUB ("Subtract") har kode 2xx, og den trekker verdien i xx i minnet fra akkumulatoren

Instruksjoner i LMC

• BRA:

• Instruksjonen BRA ("Branch") med kode 6xx gjor at programmet hopper til et annet sted i minnet. Dette lar oss blant annet lage lokker.

• BRP - Betinget hopp:

• Ofte onsker vi à hoppe noen ganger, men ikke alltid

• Instruksjonen BRP ("Branch if positive") har kode 8xx. Den hopper til adressen xx hvis akkumulatoren inneholder en verdi >= 0; ellers gjor den ingenting

Instruksjoner i LMC

- BRZ:
  - Instruksjonen BRZ ("Branch if zero") har kode 7xx. Den utforer et hopp til adresse xx dersom akkumulatoren inneholder verdien 0; eller gjor den ingenting

Alle instruksjonene i LMC med assemblernotasjon

Kode	Navn	Beskrivelse
0	HLT	Stopper eksekveringen
1	ADD	Adderer verdien i angitt minnelokasjon med akkumulatoren
2	SUB	Subtraherer verdien i angitt minnelokasjon med akkumulatoren
3	STA	Lagrer akkumulatoren i angitt minnelokasjon
4	-	Ikke i bruk
5	LDA	Henter verdi fra minnet til akkumulatoren
6	BRA	Hopper til angitt adresse
7	BRZ	Hopper hvis akkumulatoren er 0
8	BRP	Hopper hvis akkumulatoren er ≥ 0
901	INP	Leser verdi fra input, og legger svaret i akkumulatoren
902	OUT	Skriver ut verdien i akkumulatoren
922	OTC	Skriver ut ASCII-tegn (ikke i boka)

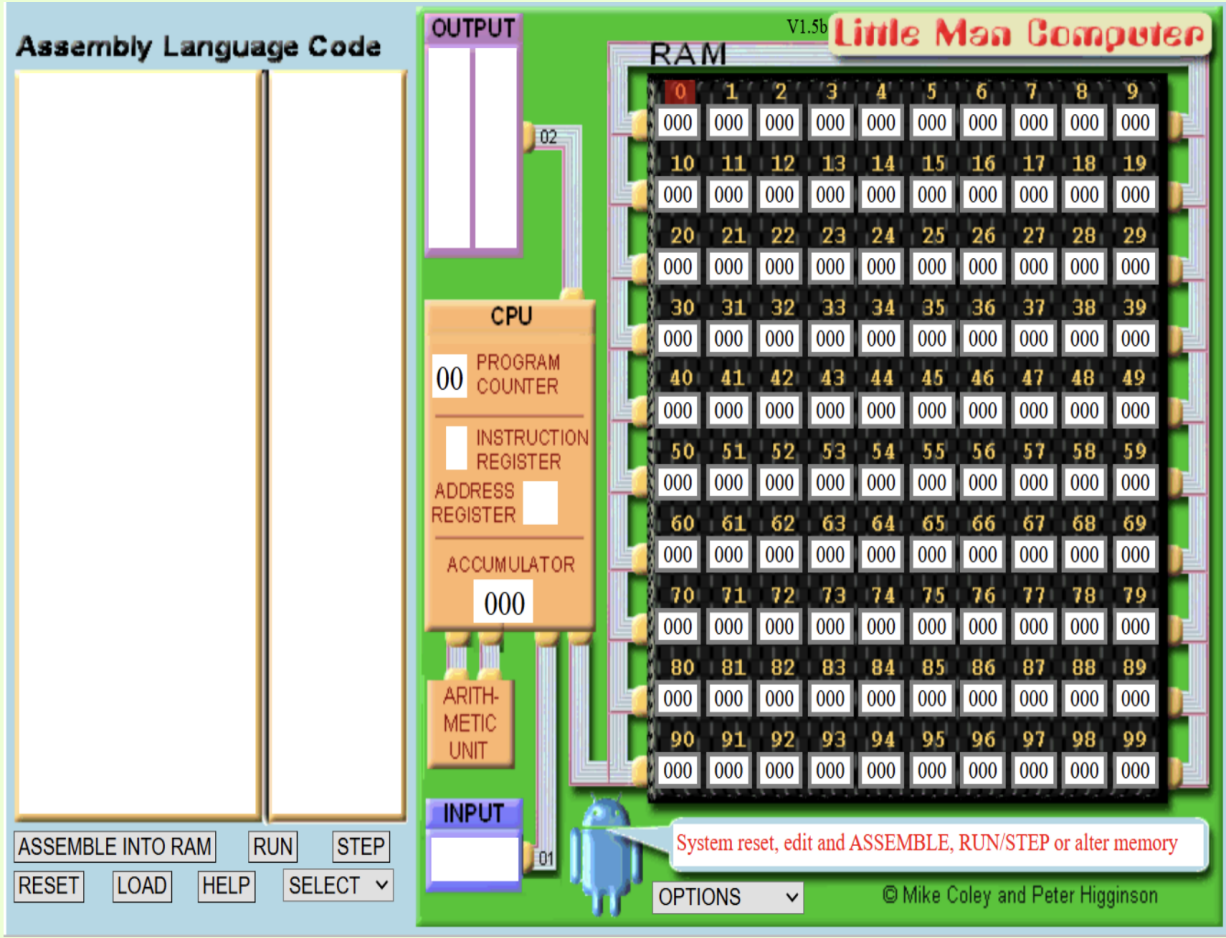
Eksekvering av LMC

1. Bruk verdien i **programtelleren** som adresse til minnet og hent neste instruksjon der. Øk samtidig programtelleren med 1

2. Splitt instruksjonen og legg delene i instruksjonsregisteret og adresseregisteret.

3. Utfør det som instruksjonsregisteret angir.

4. Gjenta fra punkt 1!



Multiplikasjon

- Ingen instruksjon for multiplikasjon i LMC.
- Vi må bruke andre instruksjoner: addisjon og løkker.
- Vi må plusse et tall med seg selv det antall ganger du vil multiplisere
- a * b er det samme som b + b + b…. a ganger

1. Leser inn **a** og lagrer det i minnet.
2. Leser inn **b** og lagrer det i minne.
3. Trenger en **variabel** for resultatet. Kan starte som 0.
4. Vi lager en løkke der vi øker resultatverdien med a.
5. Og minke **b** med 1 for hver iterasjon.
6. Skriver ut resultatet.
7. Halt

Spørsmål?

Jobbe med oblig? ☺