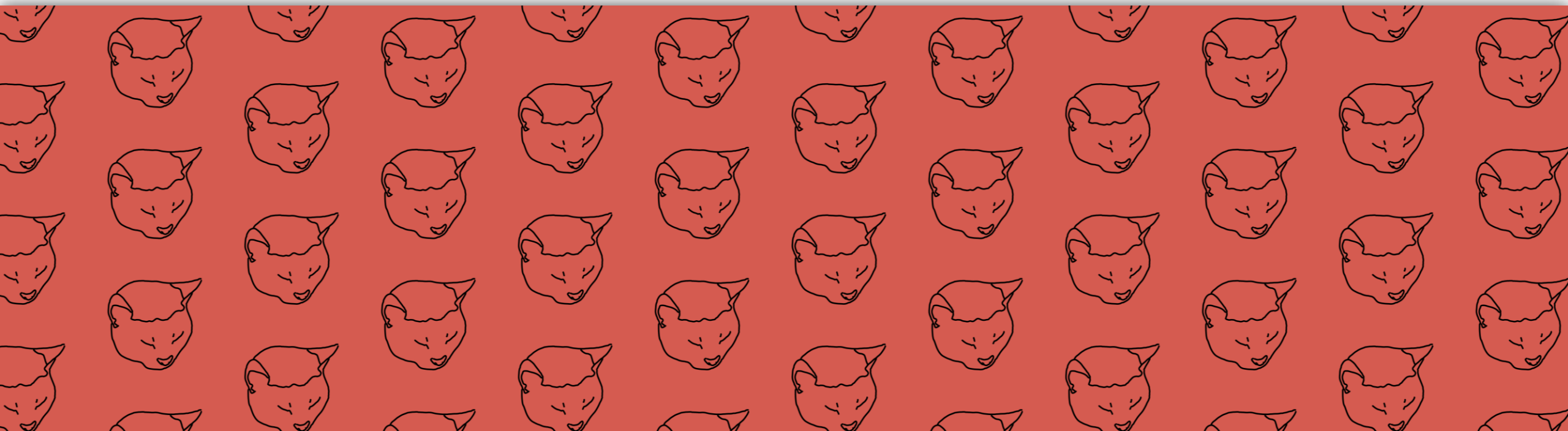
Prosessoren



Planen for i dag

Første time

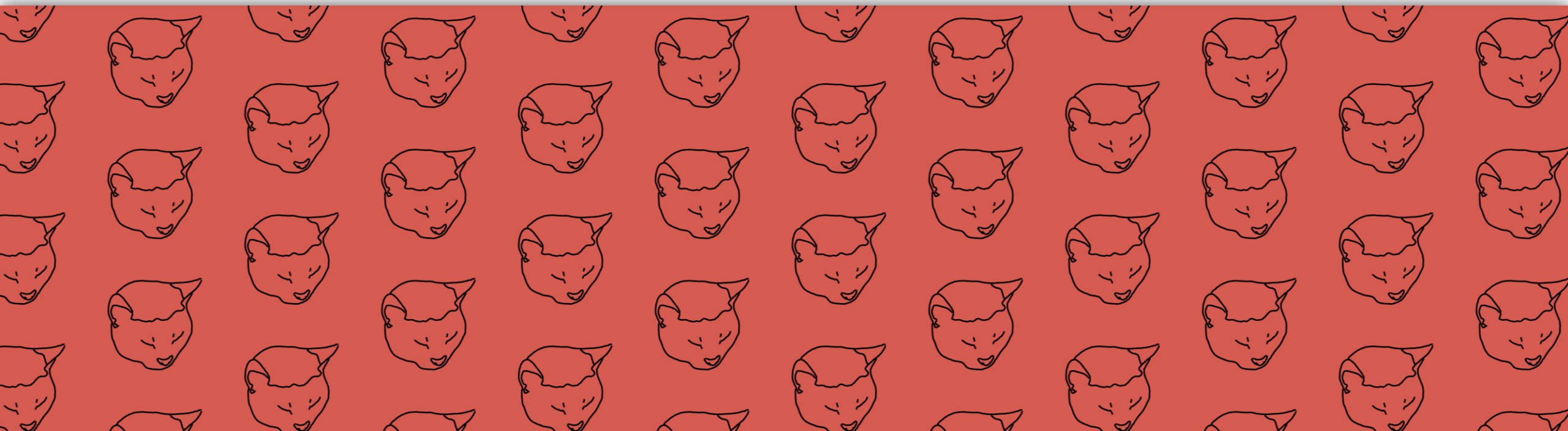
- Litt om datamaskinens arkitektur.
- Gjennomgang av LMC, maskinkode og assemblerkode.
- Gruppeoppgaver innimellom og rom for spørsmål.

Andre time

- Mer LMC-gjennomgang og oppgaver.
- Egenarbeid med ukeoppgaver og, eksamensoppgaver eller Oblig 2.
- Hvis det er behov: En mini-gjennomgang av tallsystemer.

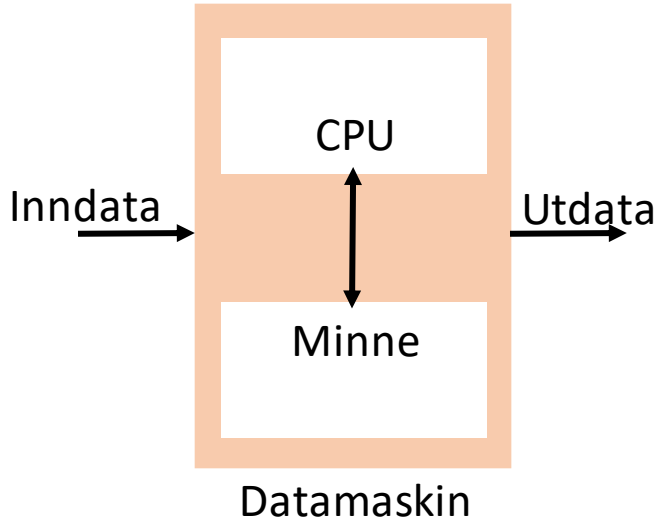
Repetisjon

Prosessoren, LMC og assemblerkode



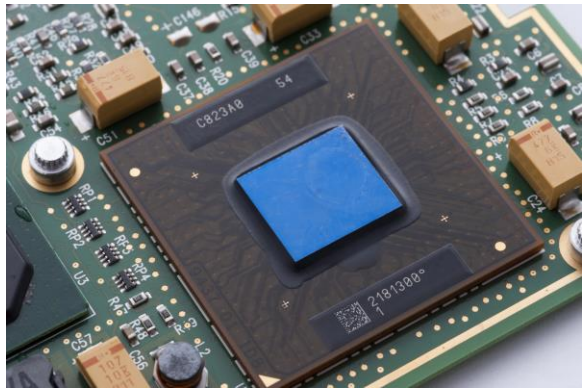
Datamaskinens arkitektur

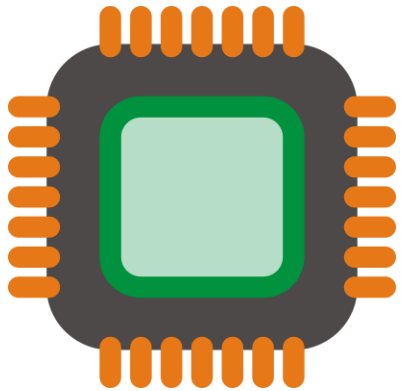
- Von Neumann-arkitektur (oppkalt etter John von Neumann)
- CPU
- Minnet: data OG instruksjoner (programmet)



CPU-en

- Central Processing Unit
- "The brain of the computer"
- ALU: arithmetic logic unit.
- Registere: et slags lite minne som holder på data som maskinen trenger rask tilgang til.
- Kontrollogikk: For å utføre operasjoner.





LMC (little man computer)

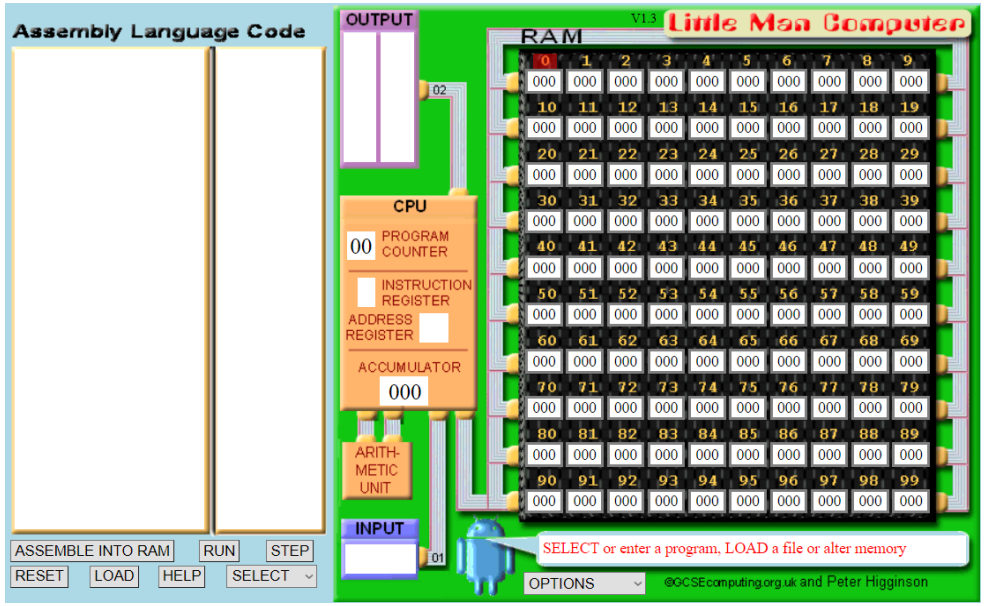
- En enkel datamaskin

De ulike delene (bortsett fra CPU):

- Assembly language code: et felt du kan programmere i eller laste opp kode til.
- RAM: Her kan programmet ditt og data lagres.
- Input: her kan du skrive input når du har programmert dette.
- Output: her kommer output når du har programmert dette.

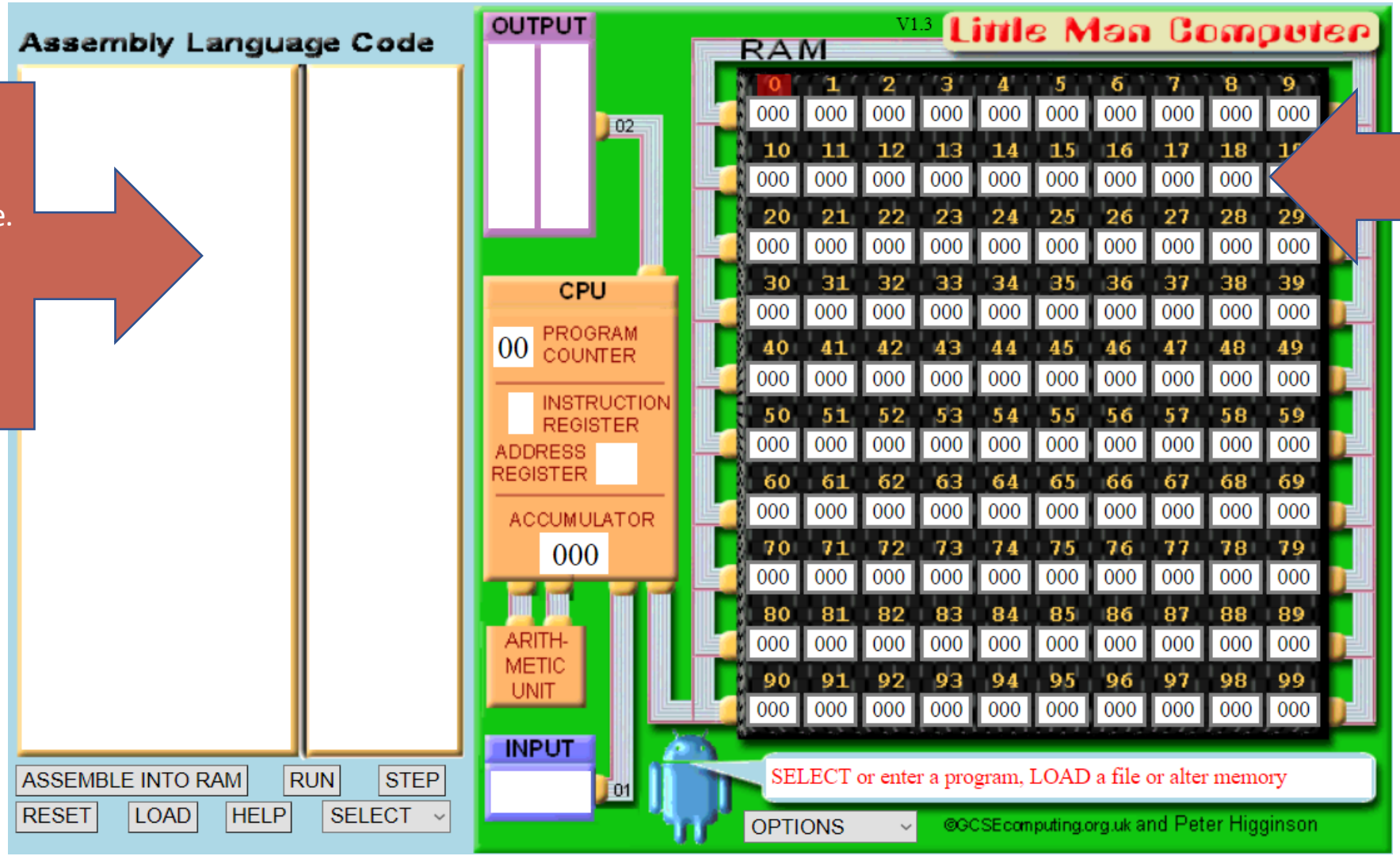
CPU-ens registere:

- Program counter viser hvilken plass i RAM vi er på.
- Instruction og address register fanger opp instrukser og adresser som gis.
- Accumulator samler på verdier i eventuelle regneoperasjoner. Dette gjøres via den aritmetiske enheten.

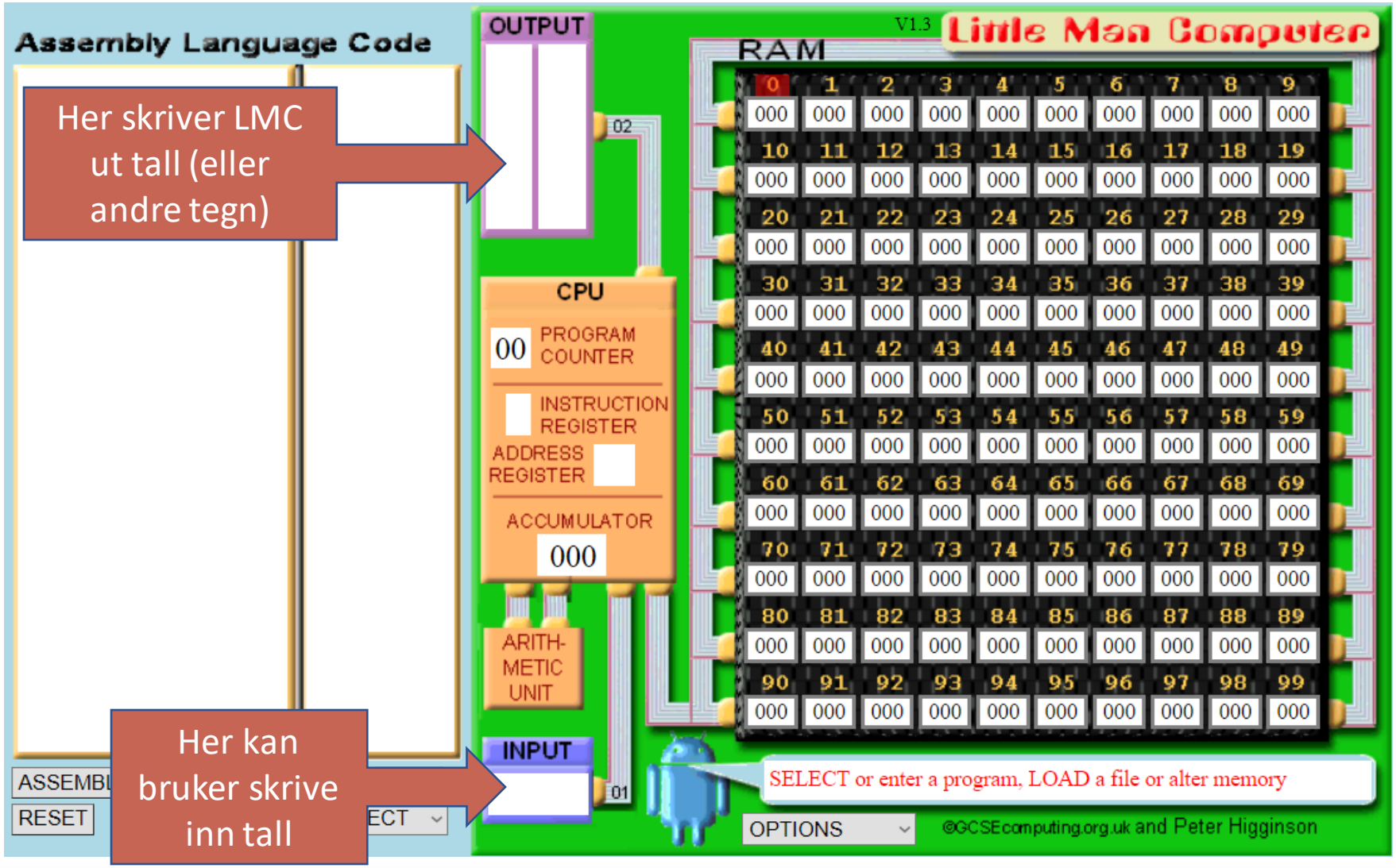


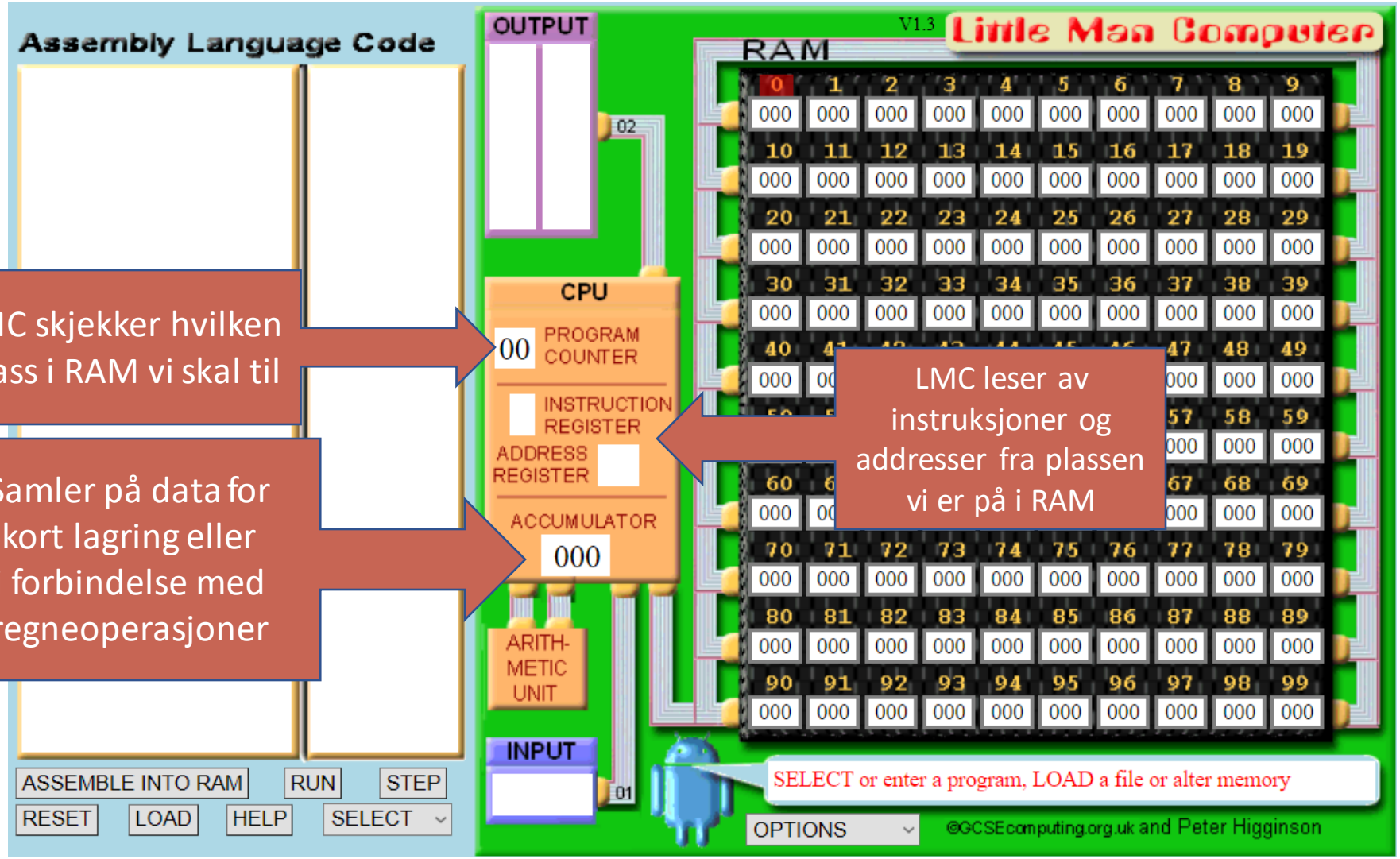
https://peterhigginson.co.uk/lmc/

Her kan du skrive/laste opp assemblerkode. Koden blir så lagt inn I RAM når du trykker "assemble into ram"



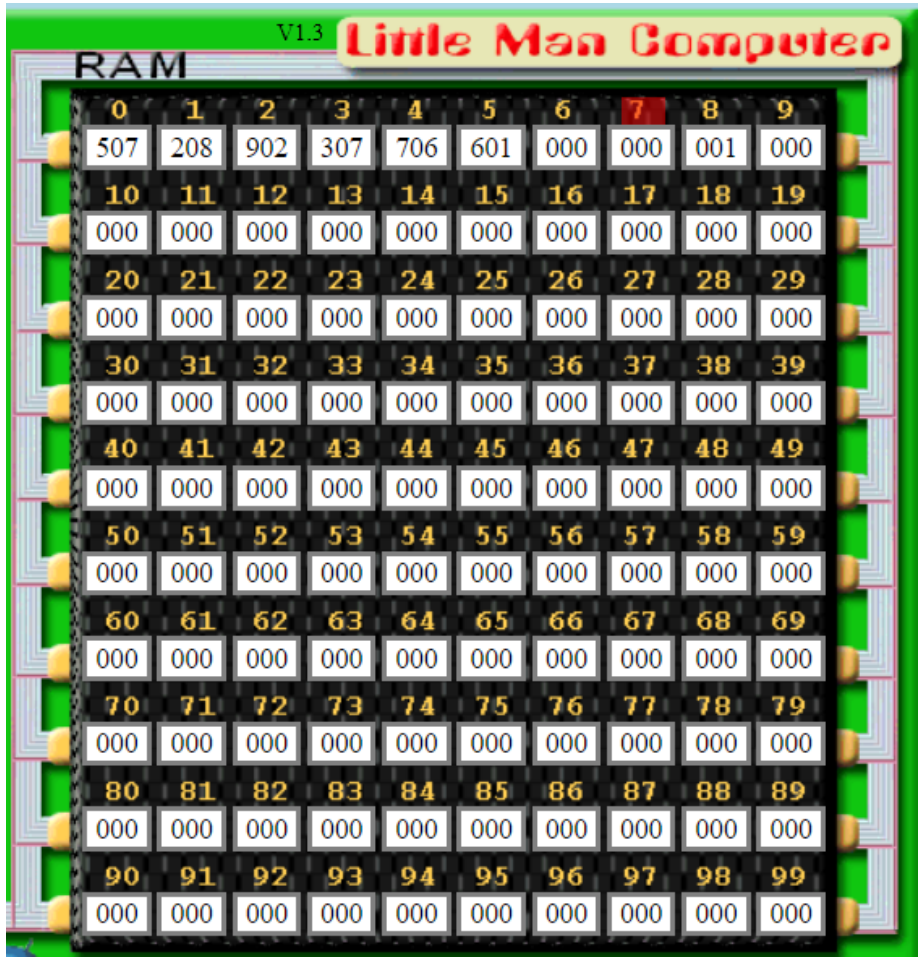
Her kan du også programmere. Det er her LMC leser instruksjoner og data


Assembly Language Code
Her skriver LMC ut tall (eller andre tegn)
OUTPUT
02
CPU
00 PROGRAM COUNTER
INSTRUCTION REGISTER
ADDRESS REGISTER
ACCUMULATOR
000
ARITH-METIC UNIT
INPUT
01
Her kan bruker skrive inn tall
ASSEMBL
RESET
ECT
V1.3
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
SELECT or enter a program, LOAD a file or alter memory
OPTIONS
©GCSEcomputing.org.uk and Peter Higginson


Assembly Language Code
OUTPUT
V1.3
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
SELECT or enter a program, LOAD a file or alter memory
OPTIONS
©GCSEcomputing.org.uk and Peter Higginson
LMC skjekker hvilken plass i RAM vi skal til
Samler på data for kort lagring eller i forbindelse med regneoperasjoner
LMC leser av instruksjoner og addresser fra plassen vi er på i RAM

Hvordan vet LMC hva den skal gjøre?

- Før man lærer assemblerkode er det lurt å vite hva koden representerer.
- Hver plass i RAM har rom for å lagre data eller en instruksjon.
- Disse lagres som et tall mellom 000-999.
- Eks: tall som starter med 5 tolkes som instruksjonen "hent". 507 tolkes da som "hent fra plass 7".
- Med mindre noe annet er spesifisert kommer alltid LMC til å tolke tall i RAM som instruksjoner.



Assemblerkode

- Numeriske koder er ikke så lette å skrive eller lese. Lett å gjøre feil.
- Assemblerkode er litt mindre abstrakt, men mer abstrakt enn f.eks. Python.
- Noen av kommandoene kan utføres uten ekstra info, slik som for eksempel HLT, INP og OUT.
- Andre må man legge til adresser for at skal funke. For eksempel LDA (load from address) gir ikke mening uten en adresse. Load from where?
- For å holde styr på data kan du også sette navn på celler. Ligner litt på variabler i python.

Kode	Navn	Beskrivelse
0xx	HLT	Stopper eksekveringen
1xx	ADD	Adderer verdien i angitt minnelokasjon med akkumulatoren
2xx	SUB	Subtraherer verdien i angitt minnelokasjon med akkumulatoren
3xx	STA	Lagrer akkumulatoren i angitt minnelokasjon
4xx	-	Ikke i bruk
5xx	LDA	Henter verdi fra minnet til akkumulatoren
6xx	BRA	Hopper til angitt adresse
7xx	BRZ	Hopper hvis akkumulatoren er 0
8xx	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)



Skrive ut bokstaver og tegn

- Da bruker vi ASCII-tabellen.
- Husk at tabellen er skrevet på heksadesimal. Så det må vi gjøre om til desimal.
- Jeg vil skrive "Erika" i LMC. Da må jeg først se hvilke verdier "E", "r", "i", "k" og "a" er.

$E = 45_{16} = 5x1 + 4x16 = 69_{10}$
$r = 72_{16} = 2x1 + 7x16 = 114_{10}$
$i = 69_{16} = 9x1 + 6x16 = 105_{10}$
$k = 6B_{16} = 11x1 + 6x16 = 107_{10}$
$a = 61_{16} = 1x1 + 6x16 = 97_{10}$

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1	DCE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Eksempel på kode

- Et program som skriver ut "Erika"
- Alle tegnene er lagret som variabler, som jeg henter med LDA og skriver ut med OTC.
- Det er lurt å huske å avslutte programmet med HLT før LMC prøver å kjøre dataene dine.

LMC eksempel_ASCII.txt - Notepad

File Edit Format View Help

```
LDA E
OTC
LDA r
OTC
LDA i
OTC
LDA k
OTC
LDA a
OTC
HLT
E DAT 69
r DAT 114
i DAT 105
k DAT 107
a DAT 97
```

IN1020 eksamen høst 2018

2(c) Maskinkode 1

```
start   INP
        STA   50
        ADD   50
        ADD   50
        OUT
slutt   HLT
```

Hva skrives ut når denne koden kjøres og brukeren gir **7** som inndata?

Velg ett eller flere alternativer (men bare ett er korrekt):

- Det skrives ikke ut noe.
- 50
- 21
- 7
- 0



IN1020 eksamen høst 2018

2(c) Maskinkode 1

```
start   INP
        STA   50
        ADD   50
        ADD   50
        OUT
slutt   HLT
```

Hva skrives ut når denne koden kjøres og brukeren gir **7** som inndata?

Velg ett eller flere alternativer (men bare ett er korrekt):

- [ ] Det skrives ikke ut noe.
- [ ] 50
- [ ] 21
- [ ] 7
- [ ] 0



Branch og løkker

- Kommandoene BRA, BRZ og BRP kan brukes til å lage løkker.
- BRA: Hopp til adresse uansett.
- BRZ: Hopp til adresse hvis akkumulatoren er 0.
- BRP: Hopp til adresse hvis akkumulatoren ≥ 0.
- Ved å lage variabler, som vi trekker fra eller legger til kan vi kontrollere hvor mange ganger løkken kjører.

```
LDA CURR
LOOP SUB ONE
OUT
STA CURR
BRZ STOPP
BRA LOOP
STOPP HLT
CURR DAT 9
ONE DAT 1
```

Med kommentarer

*LMC eksempel_enkel_lokke.txt - Notepad

File Edit Format View Help

```
LDA CURR //hent current
LOOP SUB ONE //start løkke. Trekk fra 1
OUT //print current
STA CURR //lagre current
BRZ STOPP //hvis current er 0 stopper vi
BRA LOOP //hvis ikke kjører vi løkken igjen
STOPP HLT //hit kommer vi når current er 0
CURR DAT 9 //løkken skal kjøre 10 ganger
ONE DAT 1  //hver gang trekker vi fra 1
```



```
LDA E
OTC
LDA r
OTC
LDA i
OTC
LDA k
OTC
LDA a
OTC
HLT
E DAT 69
r DAT 114
i DAT 105
k DAT 107
a DAT 97
```

Tidligere viste jeg denne koden. Kunne jeg skrevet koden annerledes/bedre?



Forslag: Forbedring via selvmodifiserende kode og en løkke

- Gjennomgått på forelesning.
- I steden for å repetere de samme kommandoene (LDA og OTC) lager vi en løkke.
- I tillegg implementerer vi selvmodifiserende kode. Det blir færre variabler. (txt er en array)
- Fint hvis vi skal skrive tekster lenger enn et par bokstaver. Eller lagre data som passer til å være en liste. F.eks ulike mengder av en ting.

```
Skriv LDA txt
BRZ Ferdig
OTC
LDA Skriv
ADD v1
STA Skriv
BRA Skriv
Ferdig HLT
v1 DAT 1
txt DAT 69
DAT 114
DAT 105
DAT 107
DAT 97
DAT 0
```

Oppgave 6

Oppgave [Englander 6.12]:

Dette programmet skal lese tre tall IN1, IN2 og IN3 og beregne

IN1 + IN2 – IN3

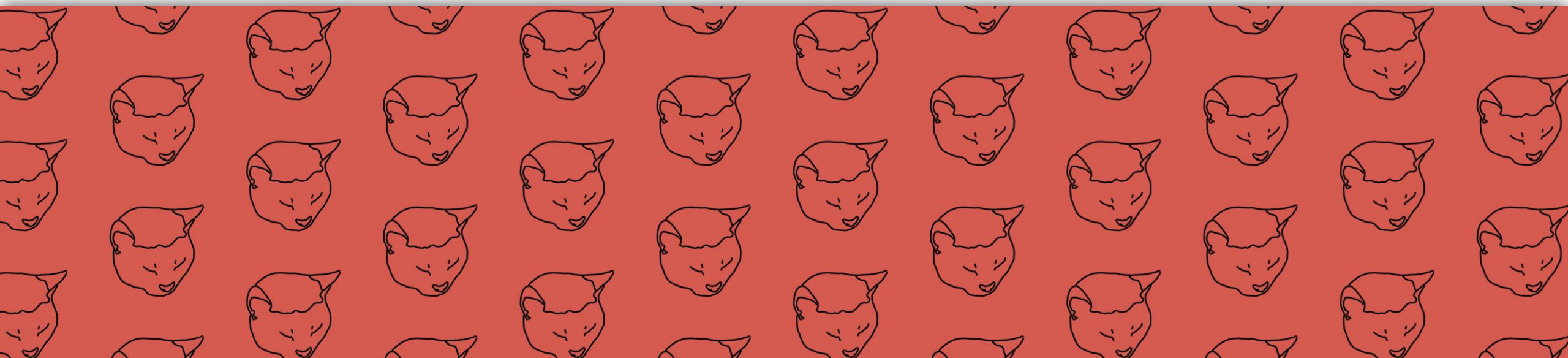
men det inneholder en feil. Hva er galt?

```
// Regn ut IN1 + IN2 - IN3

        INP
        STA     99
        INP
        ADD     99
        STA     99
        INP
        SUB     99
        OUT
        HLT
```



Diskuter:
På hvilke måter er assemberkode likt eller ulikt python (evt. andre kodespråk dere er kjent med)



Tips til oblig 2 og eksamen

- Som vanlig: gjør ukesoppgavene
- Tidligere eksamensoppgaver:

  2019: 3.3 , 3.4 , 3.5 , 3.6

  2018: 2c , 2d , 2e , 2f

  2017: 3d , 3e , 3f , 3g
- Oblig 2 har både tallsystemer og LMC.
- Se på slidene fra forje gruppetime, hvis tallsystemer er vanskelig. Gjør ukeoppgavene fra uke 3.

