

UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

Mer om Prosessoren:

Assemblerkode

IN1020 - Programmeringsdelen

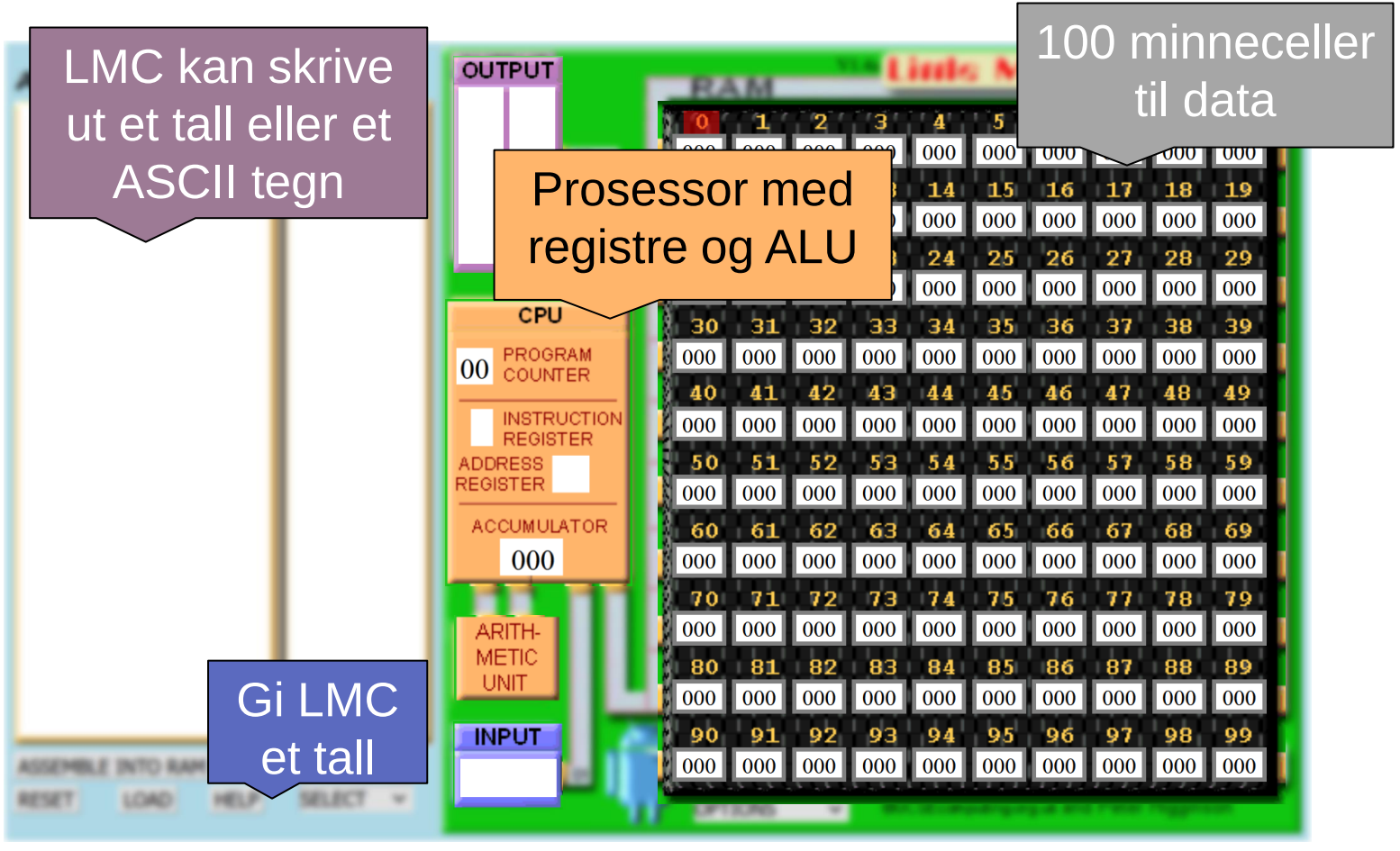
Mandag	Onsdag	Fredag	Ukesoppgaver	Oblig
	Datahistorie	~~Digital Representasjon~~		
Digital Representasjon	Prosessoren – en introduksjon	Prosessoren - Assemblerkode	Digital representasjon	Presenteres i dag
	Hvordan håndtere kompleksitet? Abstraksjonsnivå betydning	Hvordan oppnå høyere hastighet?	Intro til LMC LMC – 1	
		Avanserte prosessorer	LMC – 2	
	Operativsystemer	Datasikkerhet		Oblig 1 deadline





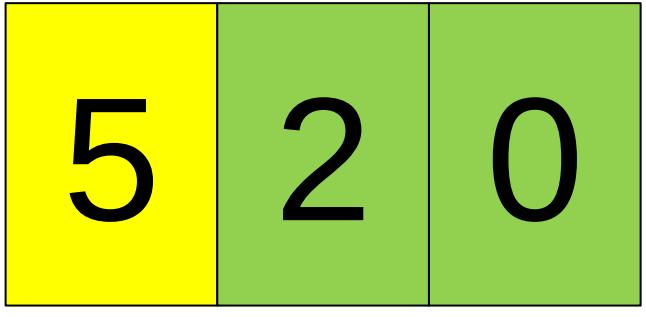


Dessverre ble opp[illegible]fra andre time ikke noe av.


LMC kan skrive ut et tall eller et ASCII tegn
Prosessor med registre og ALU
100 minneceller til data
Gi LMC et tall
OUTPUT
RAM
CPU
00
PROGRAM COUNTER
INSTRUCTION REGISTER
ADDRESS REGISTER
ACCUMULATOR
000
ARITH-METIC UNIT
INPUT
RESET
LOAD
HELP
SELECT

Instruksjoner i LMC

Adressedel



Instruksjonskode

Instruksjonskodene til LMC

Instruksjonskode	Beskrivelse
0	Stopper eksekveringen
1	Adderer verdien i angitt minnelokasjon med akkumulatoren
2	Subtraherer verdien i angitt minnelokasjon med akkumulatoren
3	Lagrer akkumulatoren i angitt minnelokasjon
4	Ikke i bruk
5	Henter verdi fra minnet til akkumulatoren
6	Hopper til angitt adresse
7	Hopper hvis akkumulatoren er 0
8	Hopper hvis akkumulatoren er ≥ 0
9	Input/Output

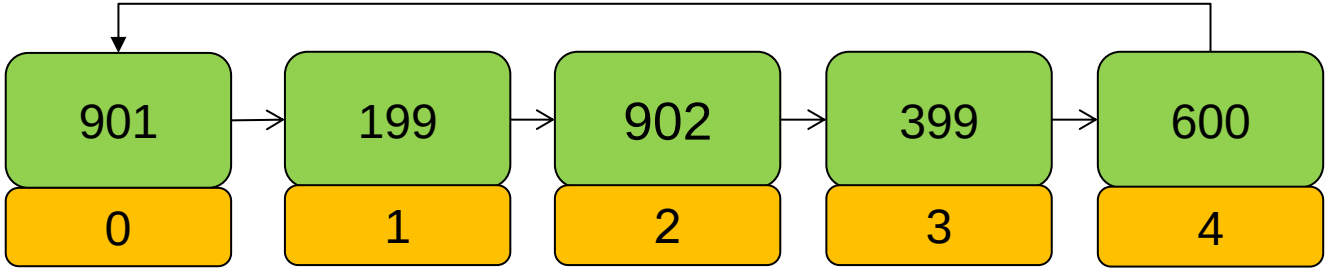
Noen som husker hva dette programmet gjør?

Adresse	Instruksjon
0	901
1	399
2	901
3	199
4	902
5	000

Hva med dette?

Adresse	Instruksjon
0	901
1	199
2	902
3	399
4	600
5	000

Ulemper med maskinkode



- Ikke lett å huske alle tallkodene, vanskelig å forstå ut fra kontekst
- Hvilke(n) minnelokasjoner (celler) inneholder data?
- Det er fort gjort å skrive en adresse feil
- Om vi endrer programmet, må vi også endre flere instruksjoner

Assemblernotasjon

Maskinkode	Assembler
0xx	HLT
1xx	ADD
2xx	SUB
3xx	STA
5xx	LDA

Aritmetikk

Lagring i RAM

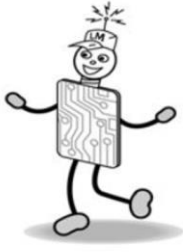
Maskinkode	Assembler
6xx	BRA
7xx	BRZ
8xx	BRP
901	INP
902	OUT
903	OTC

Programflyt

Input/Output

Assembler er et maskin-spesifikt språk.

9 instruksjoner.



ca. 256 instruksjoner.



Game boy
Sharp LR35902

> 3000 instruksjoner.

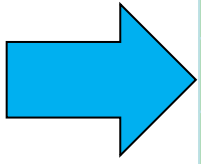


x86_64

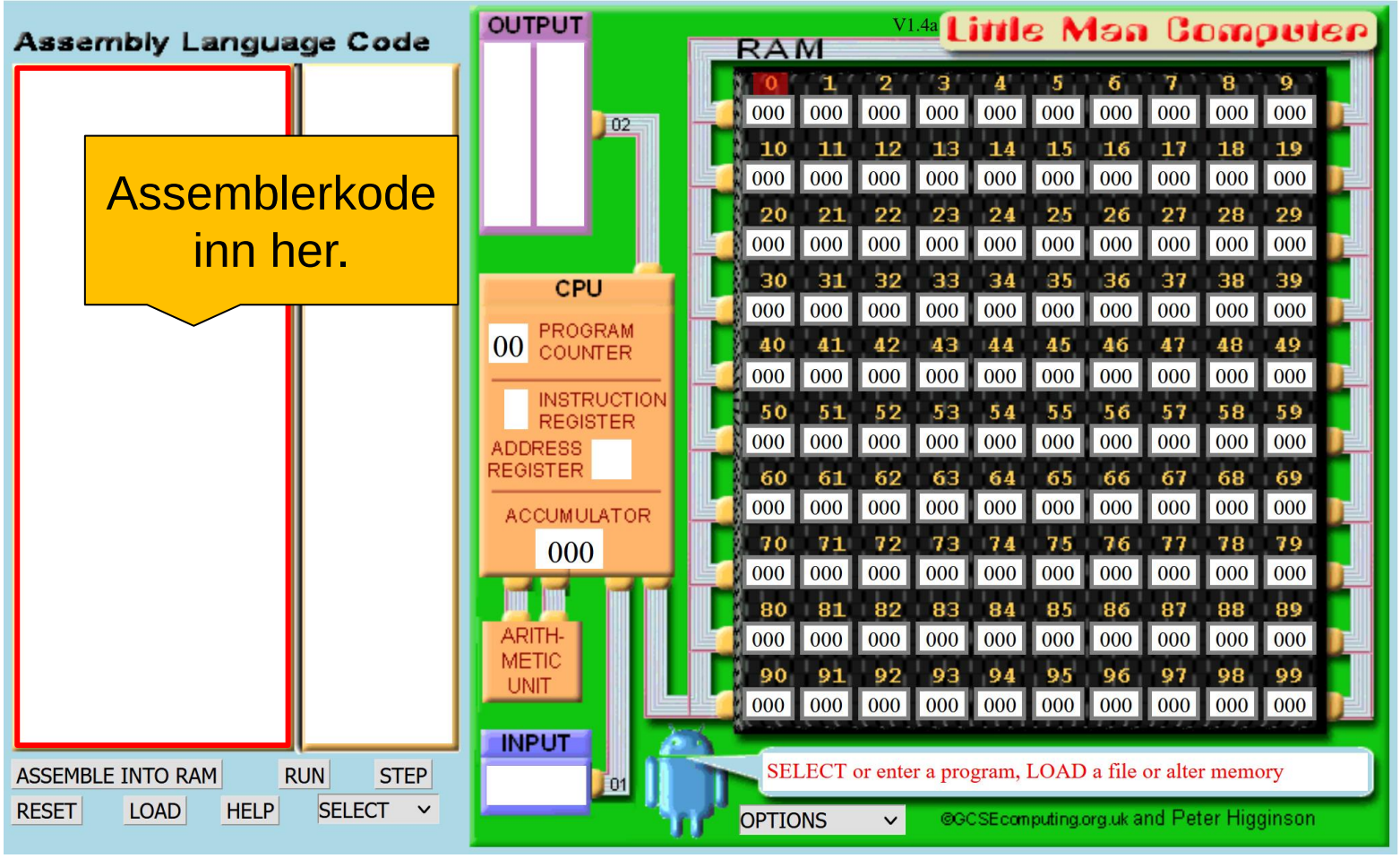
https://www.pastraiser.com/cpu/gameboy/gameboy_opcodes.html

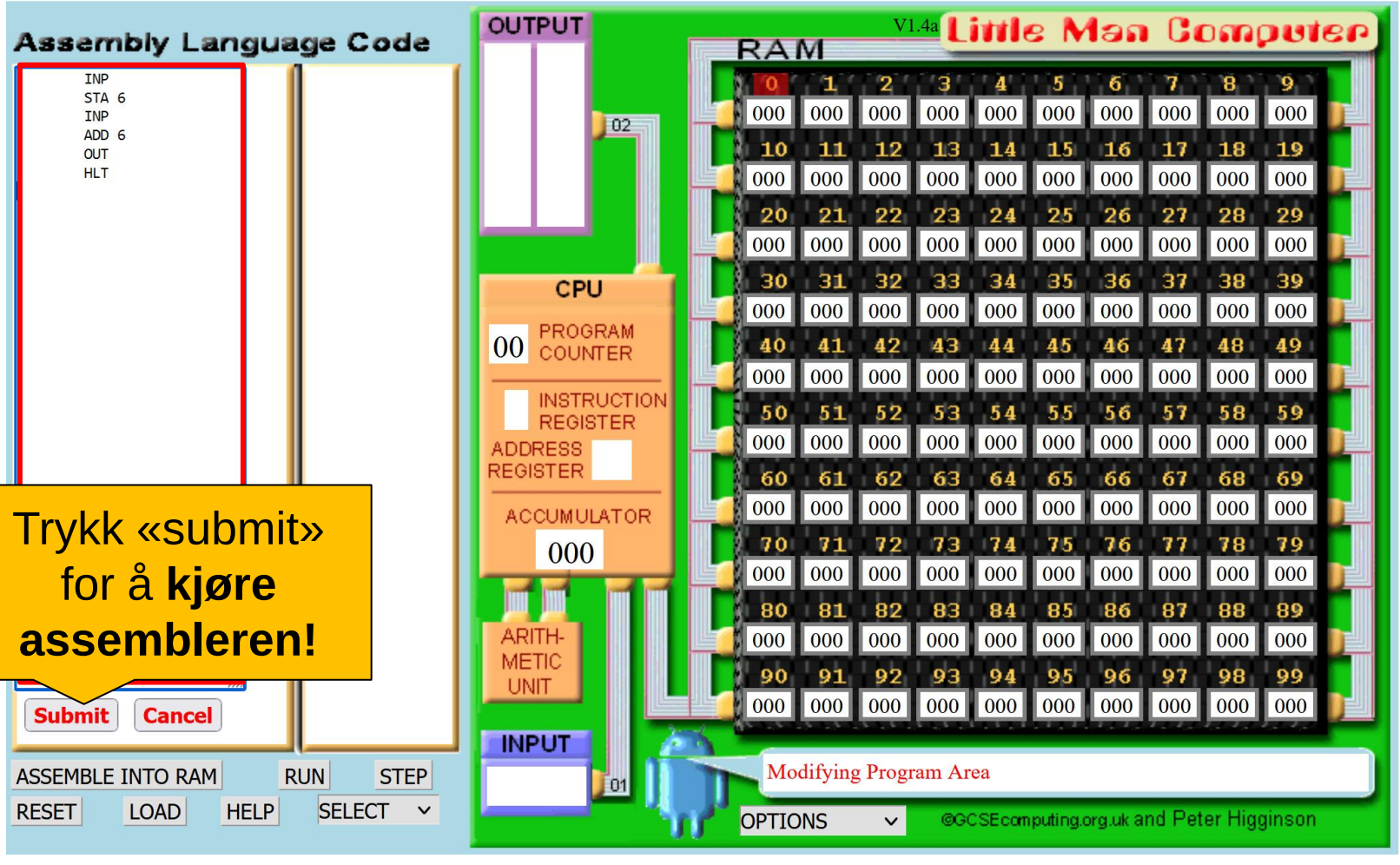
Fra Maskinkode til Assemblerkode

Adresse i minnet	Instruksjon
0	901
1	306
2	901
3	106
4	902
5	000

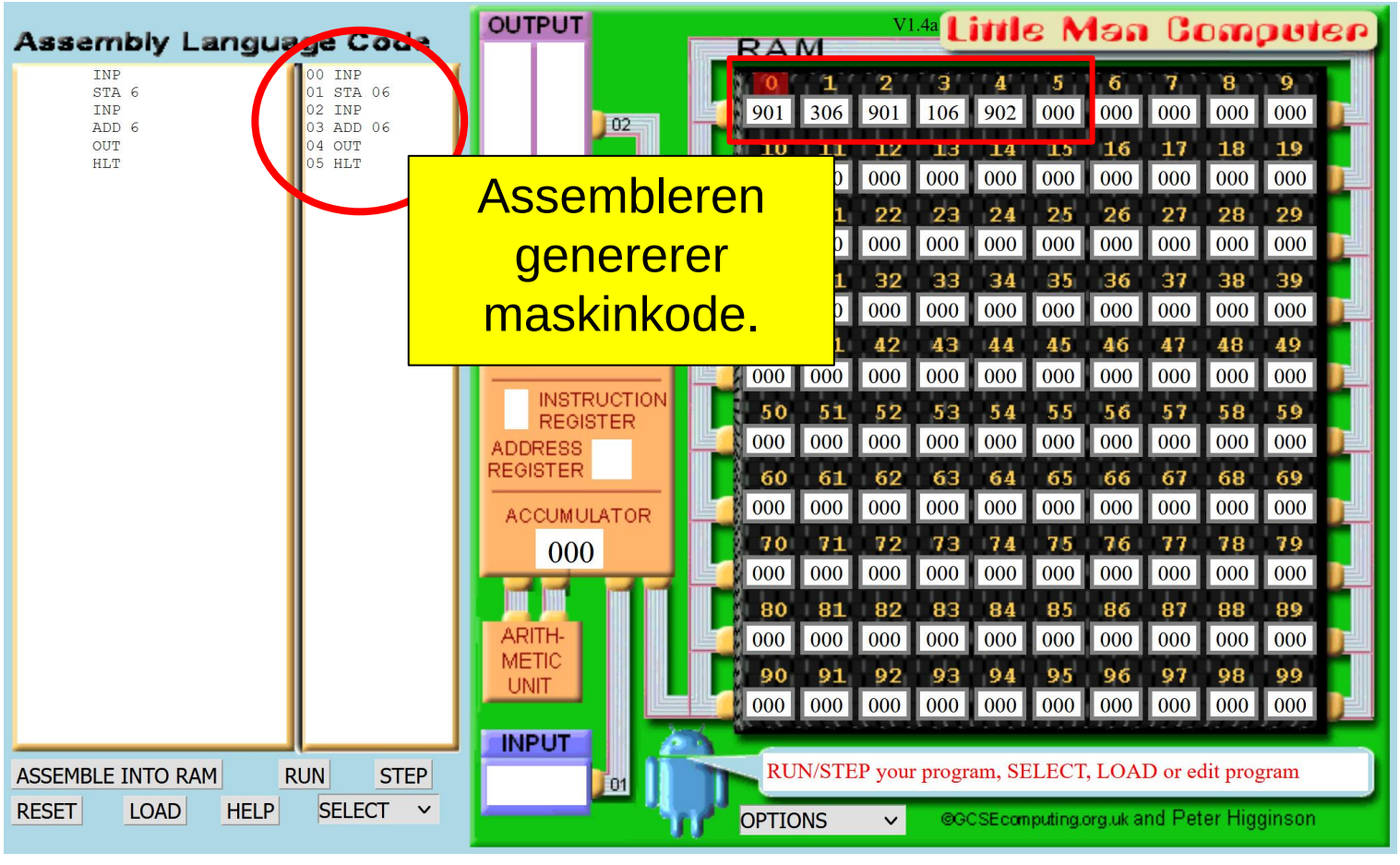


Instruksjon	Adressedel	Beskrivelse
INP		Les inn tall 1
STA	06	Lagre i minnet
INP		Les inn tall 2
ADD	06	Legg til fra minnet
OUT		Skriv ut svaret
HLT		Stopp!

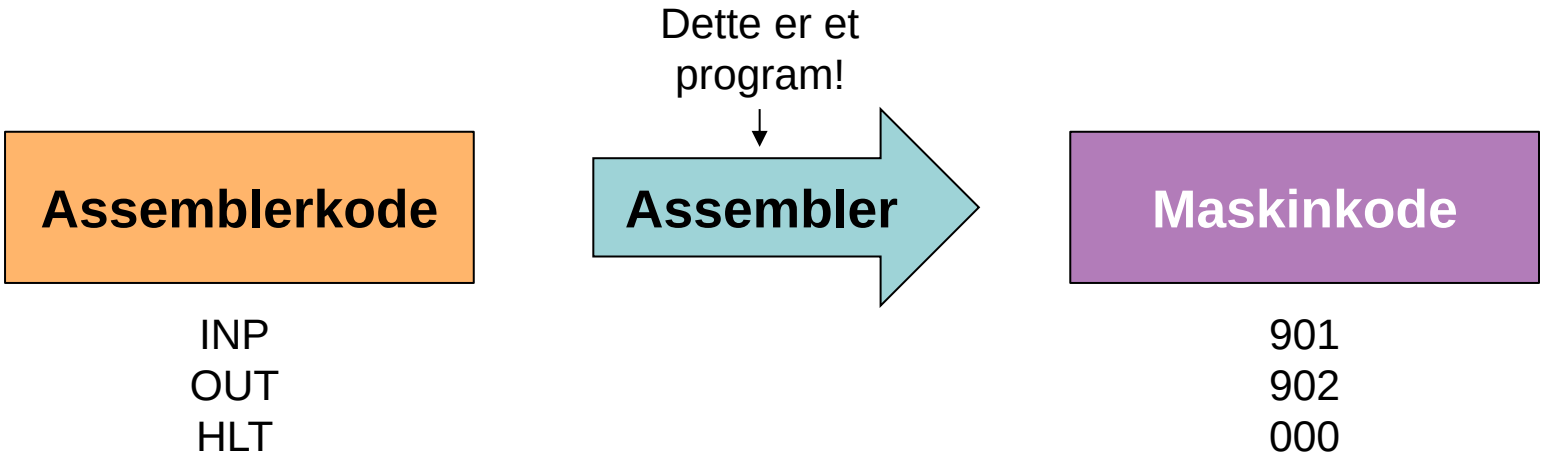

Assembly Language Code
Assemblerkode inn her.
ASSEMBLE INTO RAM
RUN
STEP
RESET
LOAD
HELP
SELECT
OUTPUT
02
CPU
00
PROGRAM COUNTER
INSTRUCTION REGISTER
ADDRESS REGISTER
ACCUMULATOR
000
ARITH-METIC UNIT
INPUT
01
V1.4a
Little Man Computer
RAM
SELECT or enter a program, LOAD a file or alter memory
OPTIONS
©GCSEcomputing.org.uk and Peter Higginson



Trykk «submit» for å **kjøre assembleren!**

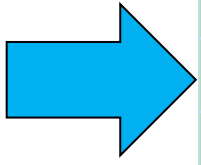

Assembly Language Code
INP
STA 6
INP
ADD 6
OUT
HLT
00 INP
01 STA 06
02 INP
03 ADD 06
04 OUT
05 HLT
Assembleren genererer maskinkode.
OUTPUT
02
V1.4a
Little Man Computer
RAM
0 1 2 3 4 5 6 7 8 9
901 306 901 106 902 000 000 000 000 000
INSTRUCTION REGISTER
ADDRESS REGISTER
ACCUMULATOR
000
ARITH-METIC UNIT
INPUT
01
ASSEMBLE INTO RAM
RUN
STEP
RESET
LOAD
HELP
SELECT
RUN/STEP your program, SELECT, LOAD or edit program
OPTIONS
©GCSEcomputing.org.uk and Peter Higginson

Dette er slik det fungerer i virkeligheten også.

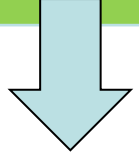


Fra Maskinkode til Assemblerkode

Adresse i minnet	Instruksjon
0	901
1	306
2	901
3	106
4	902
5	000



Instruksjon	Adressedel	Beskrivelse
INP		Les inn tall 1
STA	06	Lagre i minnet
INP		Les inn tall 2
ADD	06	Legg til fra minnet
OUT		Skriv ut svaret
HLT		Stopp!



Hva med vårt mellomlagrede tall på adresse 6?

Pseudoinstruksjonen DAT

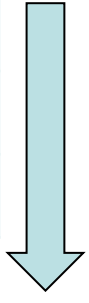
- Hvordan sier vi:
  - «Adresse 6 brukes til DATA»
- **DAT <verdi>**
  - Sier at «denne minnecellen er til data»
  - Initialiseres med **verdi**

Addresse	Instruksjon	Adressedel
0	INP	
1	STA	6
2	INP	
3	ADD	6
4	OUT	
5	HLT	
6	**DAT**	0

- Dette er fremdeles rotete
  - Hva hvis vi legger til flere instruksjoner?



Addresse	Instruksjon	Adresse del
0	INP	
1	STA	6
2	INP	
3	ADD	6
5	**OUT**	
6	HLT	
7	DAT	0



Alle disse instruksjonene sin adressedel må nå endres til «7»

- Dette er fremdeles rotete
  - Hva hvis vi legger til flere instruksjoner?

Addresse	Instruksjon	Adresse del
0	INP	
1	STA	7
2	INP	
3	ADD	7
4	ADD	7
5	OUT	
6	HLT	
7	DAT	0

En ekstra ADD

Data i adresse 6 flytter seg til adresse 7

Bruk av etiketter «labels»

- Etiketten erstatter adressen
  - Feks. **tall** til høyre ☺
- Bruk etiketten i adressedelen
- Assembleren holder styr på hvilken adresse etiketten refererer til
  - E.g. **tall** er adresse **7**.

Etikett ☺ ☺	Instruksjon	Adresse del
	INP	
	STA	tall
	INP	
	ADD	tall
	ADD	tall
	OUT	
	HLT	
tall	DAT	0

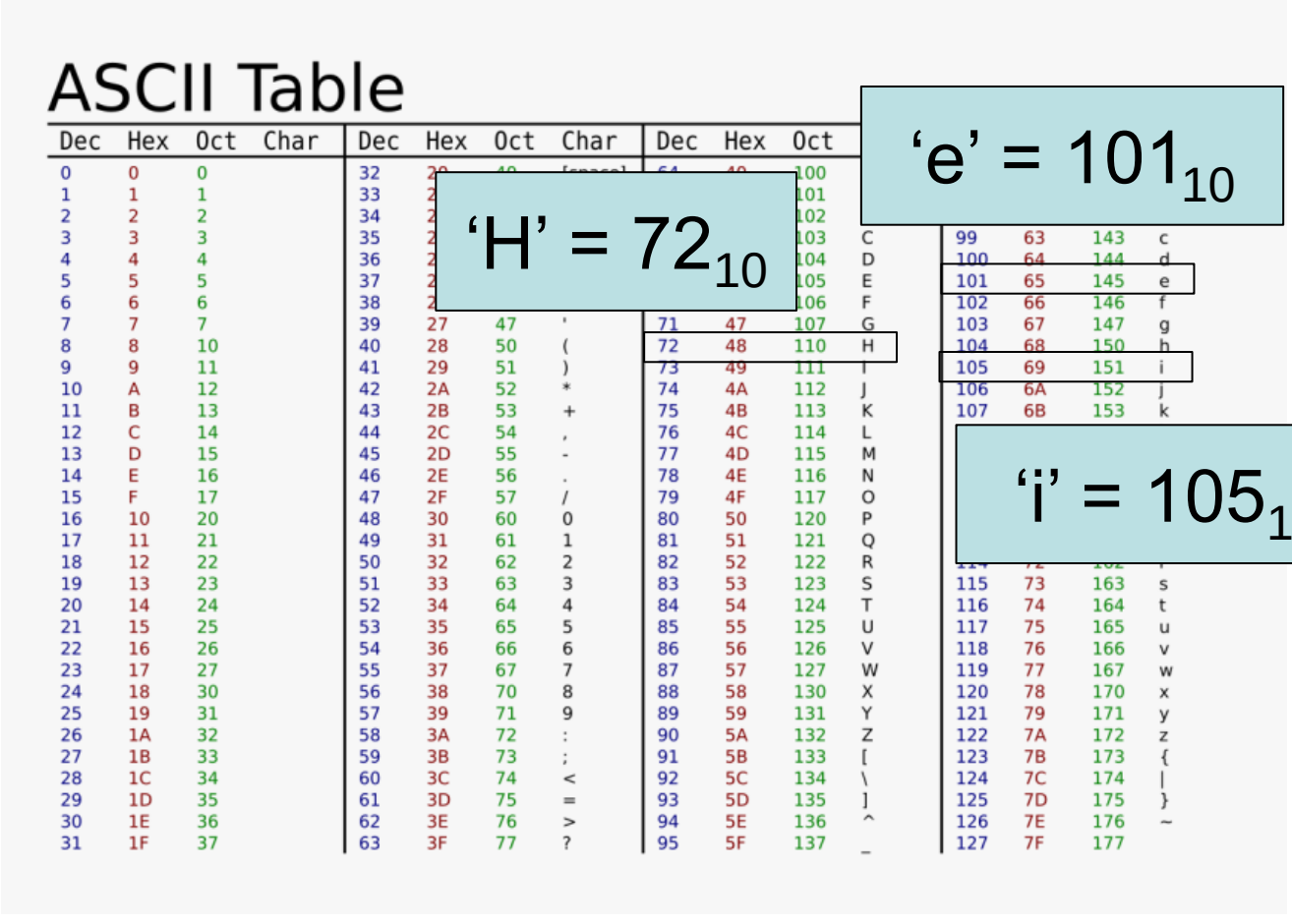
Bruk av etiketter «labels»

- Før sa vi
  - «STA – lagre akkumulatoren på minneadresse **xx**»
- Nå sier vi
  - «STA – lagre akkumulatoren på etikett **tall**»

Etikett ☺ ☺	Instruksjon	Adresse del
	INP	
	STA	tall
	INP	
	ADD	tall
	ADD	tall
	OUT	
	HLT	
tall	DAT	0

Veien til «Hello world»

```
>>> print("Hello World!")
Hello World!
>>>
```



Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@
1	1	1		33	21	41	!	65	41	101	A
2	2	2		34	22	42	"	66	42	102	B
3	3	3		35	23	43	#	67	43	103	C
4	4	4		36	24	44	$	68	44	104	D
5	5	5		37	25	45	%	69	45	105	E
6	6	6		38	26	46	&	70	46	106	F
7	7	7		39	27	47	'	71	47	107	G
8	8	10		40	28	50	(	72	48	110	H
9	9	11		41	29	51	)	73	49	111	I
10	A	12		42	2A	52	*	74	4A	112	J
11	B	13		43	2B	53	+	75	4B	113	K
12	C	14		44	2C	54	,	76	4C	114	L
13	D	15		45	2D	55	-	77	4D	115	M
14	E	16		46	2E	56	.	78	4E	116	N
15	F	17		47	2F	57	/	79	4F	117	O
16	10	20		48	30	60	0	80	50	120	P
17	11	21		49	31	61	1	81	51	121	Q
18	12	22		50	32	62	2	82	52	122	R
19	13	23		51	33	63	3	83	53	123	S
20	14	24		52	34	64	4	84	54	124	T
21	15	25		53	35	65	5	85	55	125	U
22	16	26		54	36	66	6	86	56	126	V
23	17	27		55	37	67	7	87	57	127	W
24	18	30		56	38	70	8	88	58	130	X
25	19	31		57	39	71	9	89	59	131	Y
26	1A	32		58	3A	72	:	90	5A	132	Z
27	1B	33		59	3B	73	;	91	5B	133	[
28	1C	34		60	3C	74	<	92	5C	134	\
29	1D	35		61	3D	75	=	93	5D	135	]
30	1E	36		62	3E	76	>	94	5E	136	^
31	1F	37		63	3F	77	?	95	5F	137	_

Dec	Hex	Oct	Char
99	63	143	c
100	64	144	d
101	65	145	e
102	66	146	f
103	67	147	g
104	68	150	h
105	69	151	i
106	6A	152	j
107	6B	153	k
115	73	163	s
116	74	164	t
117	75	165	u
118	76	166	v
119	77	167	w
120	78	170	x
121	79	171	y
122	7A	172	z
123	7B	173	{
124	7C	174	|
125	7D	175	}
126	7E	176	~
127	7F	177	

Hei, verden!

- Bruk DAT for å «forhåndslagre» ascii-koder for tall
  - Ikke glem at DAT <verdi> initialiserer minnet til <verdi> !
- Bruk etiketter for å laste inn ascii-kodene i akkumulator

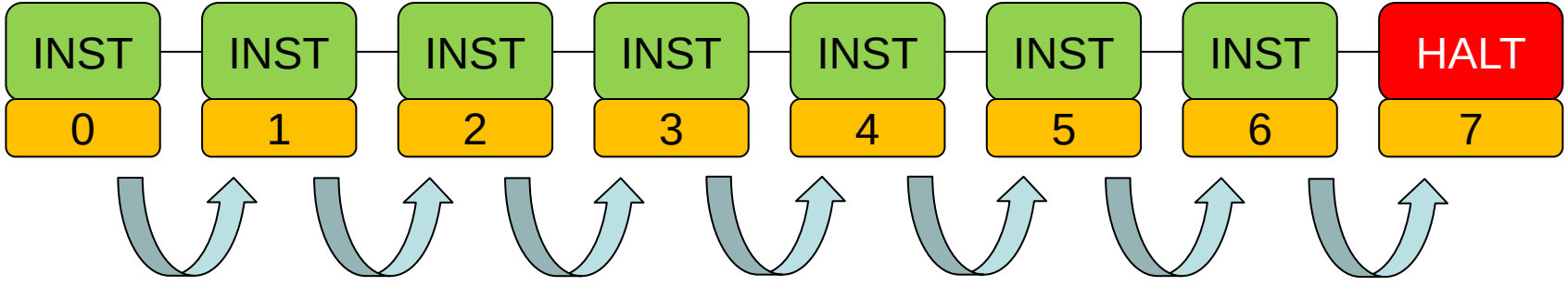
Adresse	Mnemonic	Adressedel
0	LDA	7
1	OTC	
2	LDA	8
3	OTC	
4	LDA	9
5	OTC	
6	HLT	
7	DAT	72
8	DAT	101
9	DAT	105

Et spørsmål til dere.

- Er en etikett det samme som en variabel?

Programflyt

Repetisjon: Programflyten i LMC er Sekvensiell.



Programtelleren starter på 0, og øker med én for hver eneste instruksjon.

Vanlig instruksjon

Adresse i minnet

Instruksjoner for Programflyt

Instruksjon
BRA xx
BRZ xx
BRP xx

- Disse instruksjonene kan endre programtelleren til LMC.
- Dette betyr at de kan **påvirke hvilken instruksjon LMC skal kjøre i neste instruksjonssykel.**
- Uformelt sier vi gjerne at vi «hopper» (engelsk: «brancher») til en adresse i minnet, og kjører neste instruksjon derfra.

Instruksjoner for Programflyt

Instruksjon	Beskrivelse
BRA xx	LMC sin neste instruksjon er på adresse **xx**.

BRZ xx	if akkumulator == 0:
	LMC sin neste instruksjon er på adresse **xx**.
	else:
	Ingen endring.

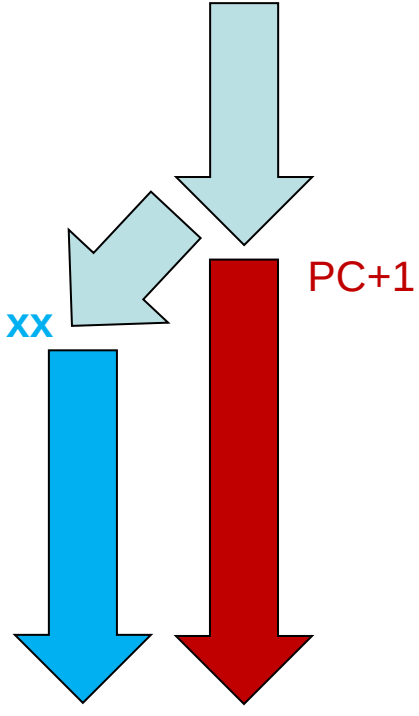
BRP xx	if akkumulator >= 0:
	LMC sin neste instruksjon er på adresse **xx**.
	else:
	Ingen endring.

BRP/BRZ xx

- I praksis kan LMC «gå to veier» fra en BRP (eller BRZ) instruksjon.

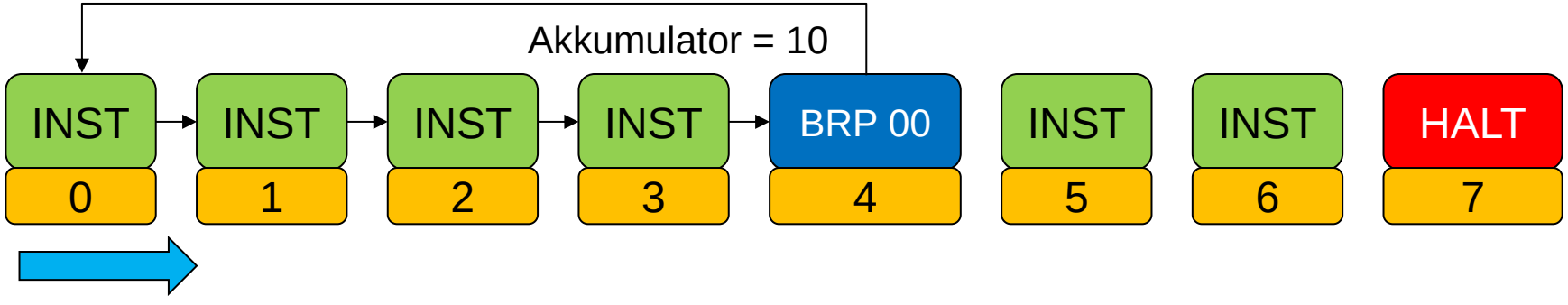
- Hvis akkumulator >= 0 (BRP):
  - Sett programteller til adresse **xx**.
  - Neste instruksjon kjøres fra adresse **xx**.

- Hvis ikke:
  - Ikke gjør noe..
  - Dvs. programteller økes bare med èn som vanlig.



BRP

if accumulator >= 0:
    Sett programteller til 00.

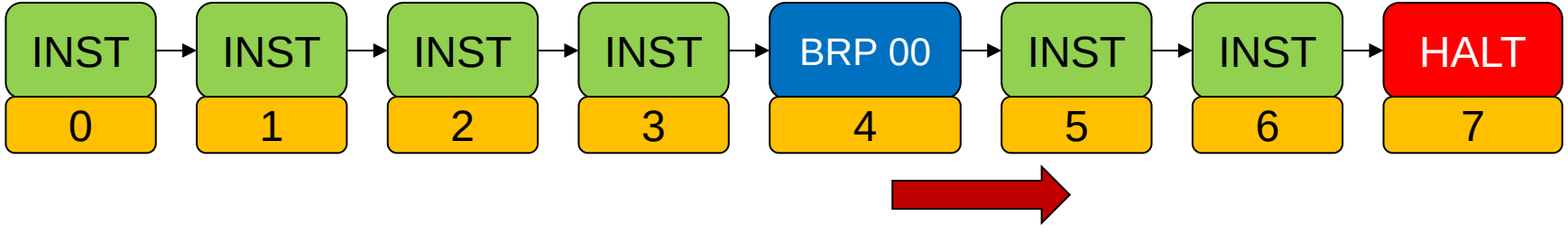


Neste instruksjon henter
LMC fra adresse 0.

BRP

if accumulator >= 0:
    Sett programteller til 00.

Akkumulator = -1

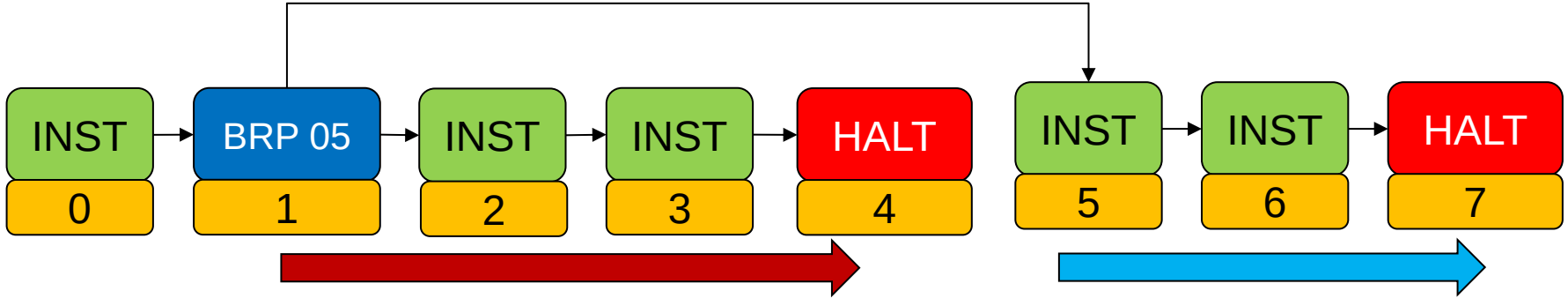


Neste instruksjon henter LMC fra adresse 5.

else:
    Ingen endring.

BRP – En annen variant

if accumulator >= 0:
    Sett programteller til 05.

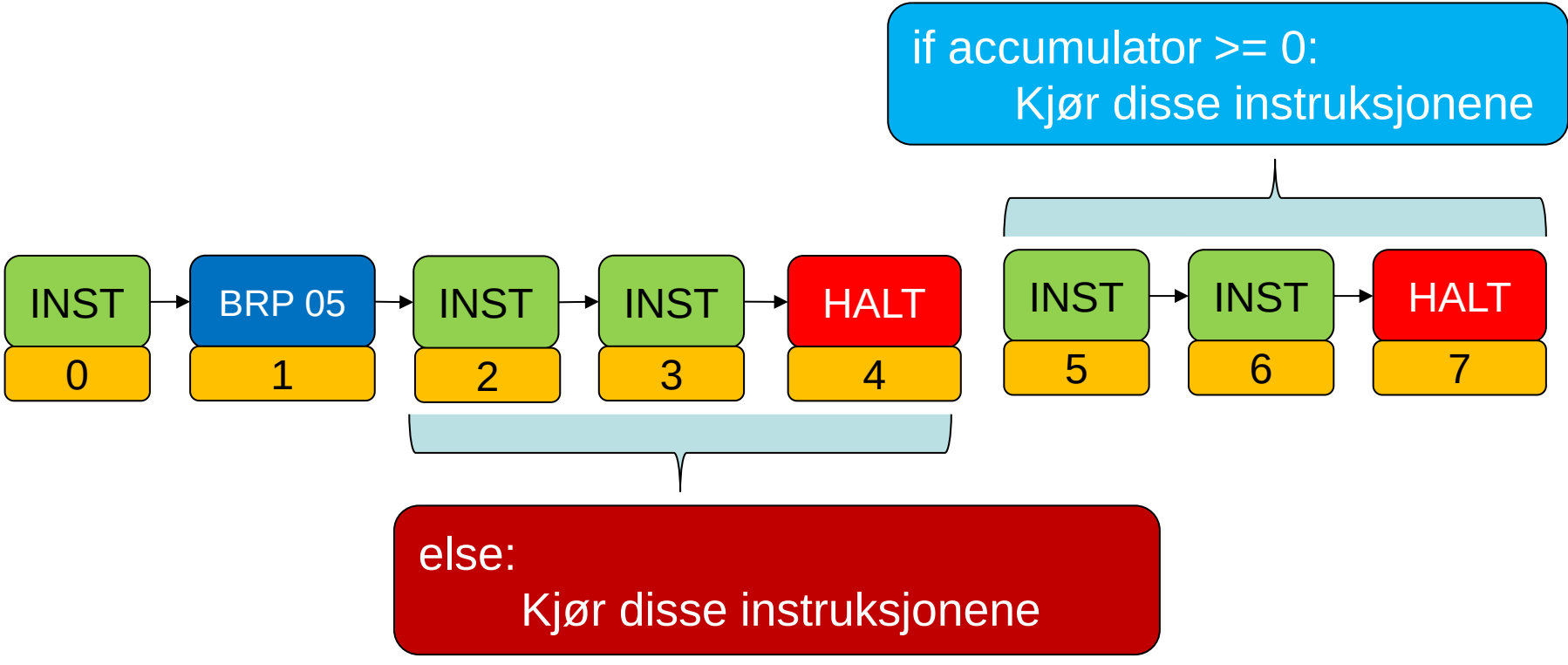


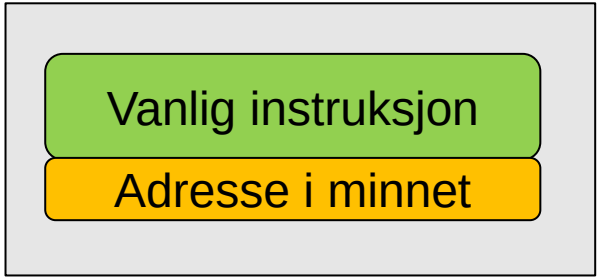
Neste instruksjon henter LMC fra adresse 02.

Neste instruksjon henter LMC fra adresse 05.

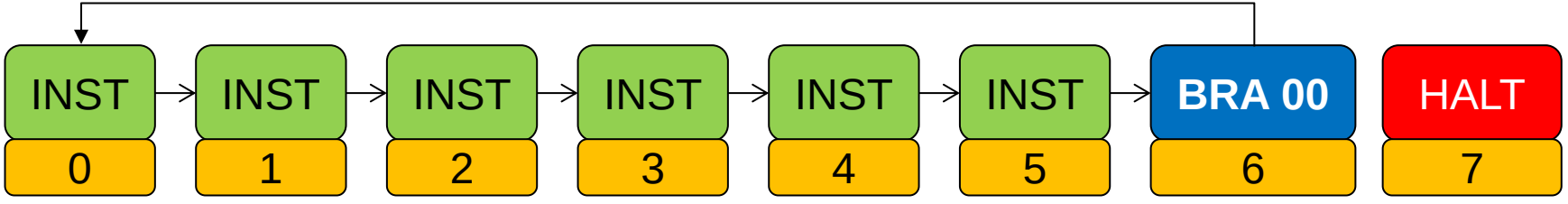
else:
    Ingen endring.

En annen variant.




Vanlig instruksjon
Adresse i minnet

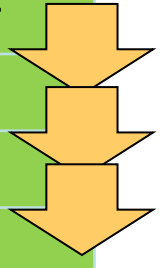
BRA


INST
0
INST
1
INST
2
INST
3
INST
4
INST
5
BRA 00
6
HALT
7

BRA – Å telle ned fra 10 til 0

Etikett	Instruksjon	Adressedel
	LDA	teller
print	OUT	
	SUB	en
	BRA	print
	HLT	
teller	DAT	10
en	DAT	1



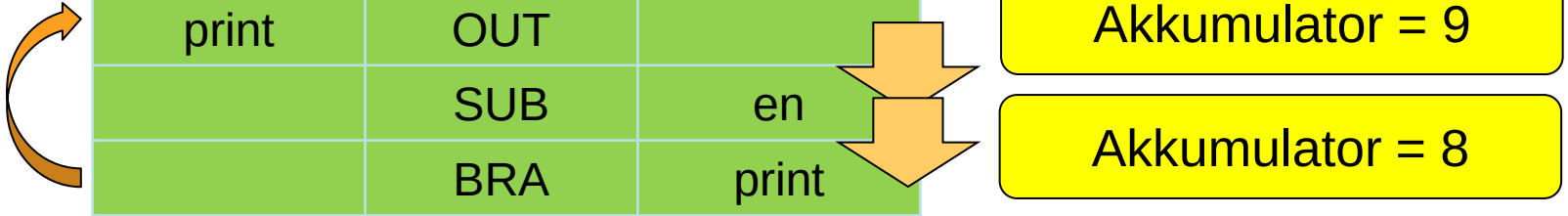


Akkumulator = 10

Akkumulator = 9

BRA – Å telle ned fra 10 til 0

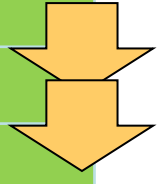
Etikett	Instruksjon	Adressedel
	LDA	teller
print	OUT	
	SUB	en
	BRA	print
	HLT	
teller	DAT	10
en	DAT	1



BRA – Å telle ned fra 10 til 0

Etikett	Instruksjon	Adressedel
	LDA	teller
print	OUT	
	SUB	en
	BRA	print
	HLT	
teller	DAT	10
en	DAT	1



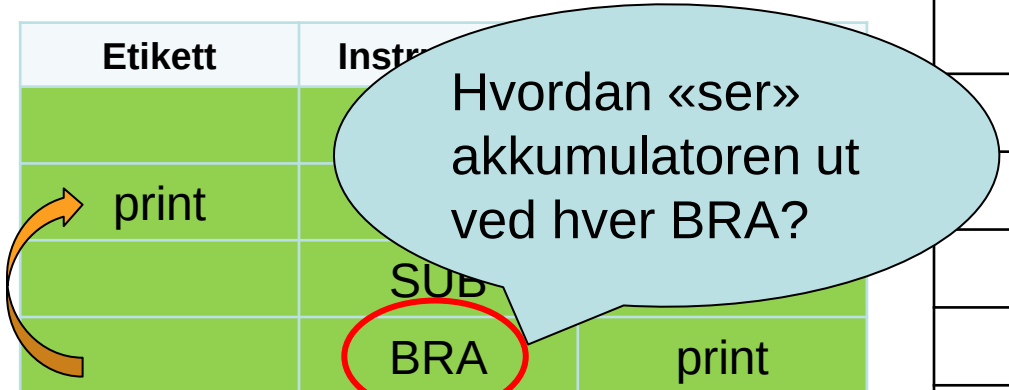


Akkumulator = 8

Akkumulator = 7

Å tenke iterativt.

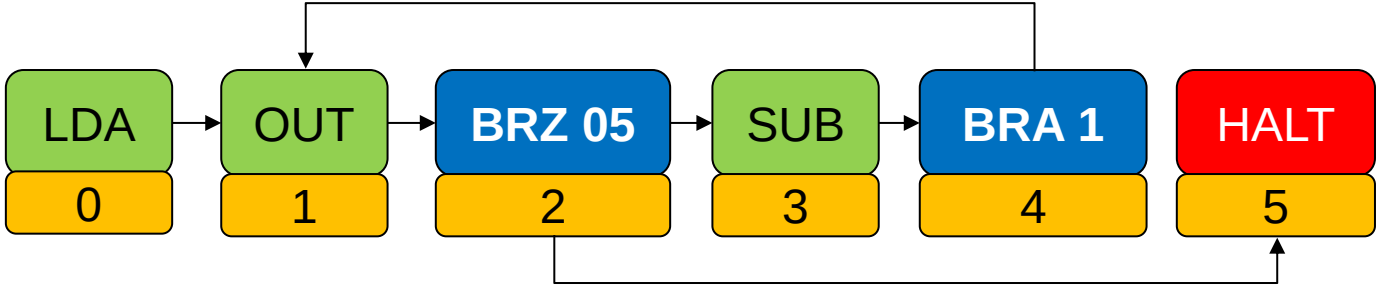
Etikett	Instr	
print		
	SUB	
	BRA	print
	HLT	
teller	DAT	10
en	DAT	1



Iterasjon	Akkumulator
1	9
2	8
3	7
4	6
5	5
6	4
7	3
8	2
9	1
10	0
11	-1

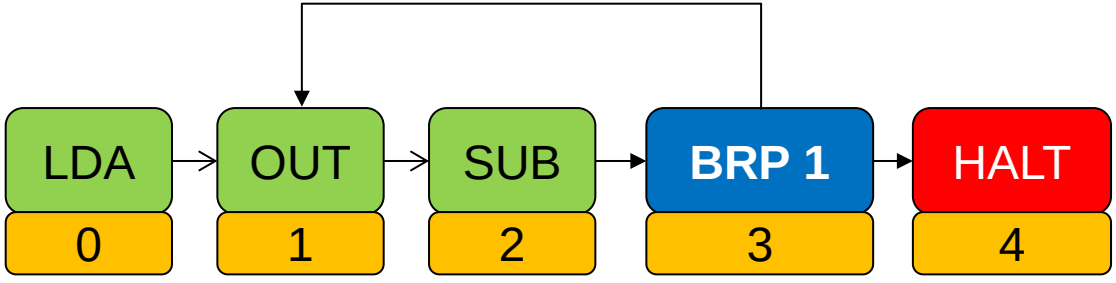
Vanlig instruksjon
Adresse i minnet

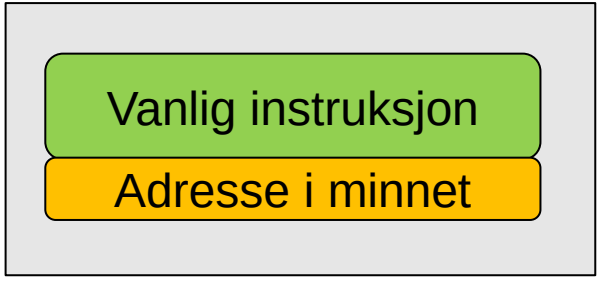
Hopp ut av sirkelen.



Den iterasjonen akkumulator er 0, hopper vi ut.

Bryt sirkelen i stedet?





If sjekking med BRZ & BRP

- BRZ sjekker: $akkumulator == 0$
  - Eller mer formelt om $a == 0$

```
if a == 0:
    print(“Sant”)
else:
    print(“Usant”)
```

- Hva med: If $akkumulator == 2$?
  - Kan dette løses med BRZ?

```
if a == 2:
    print(“Sant”)
else:
    print(“Usant”)
```

If sjekking med BRZ & BRP

$a == 0$ LMC kan sjekke dette ganske enkelt med BRZ

$a == 2$ Vi ønsker å gjøre dette. Kan vi stokke om litt her?

$a - 2 == 0$

Regn ut dette,
legg det i
akkumulatoren,
og kjør BRZ!

If sjekking med BRZ & BRP

$a \geq 0$ LMC kan sjekke dette ganske enkelt med BRP

$a \geq 2$ Vi ønsker å gjøre dette. Kan vi stokke om litt her?

$a - 2 \geq 0$

Regn ut dette, legg det i akkumulatoren, og kjør BRP!

Klarer dere disse da?

$a > 2$

$a \leq 2$

Sjekk ukesoppgavene!

Avanserte programmeringseksempler i LMC

Eksempel: Multiplikasjon

Oblig Tips!

Vi ønsker å multiplisere to tall.

1. Har vi en instruksjon som gjør det?
2. Kan vi oppnå samme resultat med én eller flere andre instruksjoner?

Vi vet at

```
a * b = b + b + ... + b,
```

a ganger.

Å multiplisere med to.

Etikett	Instruksjon	Adressedel	Akkumulator	temp
	INP		4	
	STA	temp		4
	ADD	temp	8	
	OUT			
	HLT			
temp	DAT			

Å multiplisere med tre.

Etikett	Instruksjon	Adressedel	Akkumulator	temp
	INP		4	
	STA	temp		4
	ADD	temp	4+4=8	
	ADD	temp	8+4=12	
	OUT			
	HLT			
temp	DAT			

$= 4 * 3$

Å multiplisere med fire.

Etikett	Instruksjon	Adressedel	Akkumulator	temp
	INP		4	
	STA	temp		4
	ADD	temp	4+4=8	
	ADD	temp	8+4=12	
	ADD	temp	12+4=16	
	OUT			
	HLT			
null	DAT	0		

$= 4 * 4$

Observasjonen..

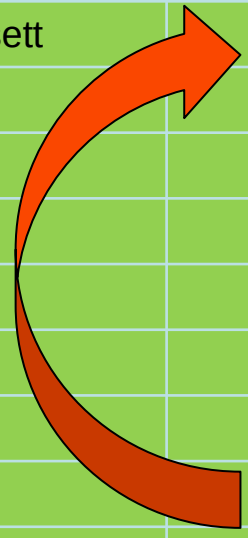
- En ekstra **ADD** for hver gang vi ganger.
- Dette må vi da kunne gjøre med kontrollert programflyt?
- BRA, BRP eller BRZ?

Etikett	Instruksjon	Adressedel
fortsett	LDA	ganger
	BRZ	exit
	SUB	en
	STA	ganger
	LDA	sum
	ADD	input
	STA	sum
	BRA	fortsett
exit	LDA sum	
	OUT	
	HLT	
sum	DAT	0
en	DAT	1
ganger	DAT	2
input	DAT	5

Dette programmet regner ut:

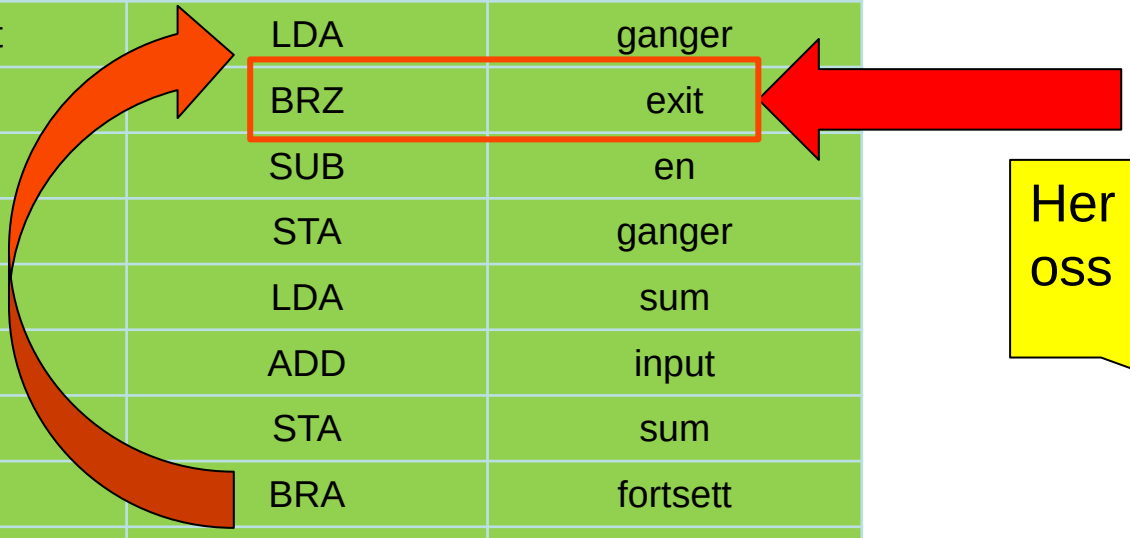
$$sum = input * ganger$$

Etikett	Instruksjon	Adressedel
fortsett	LDA	ganger
	BRZ	exit
	SUB	en
	STA	ganger
	LDA	sum
	ADD	input
	STA	sum
	BRA	fortsett
exit	LDA sum	
	OUT	
	HLT	
sum	DAT	0
en	DAT	1
ganger	DAT	2
input	DAT	5



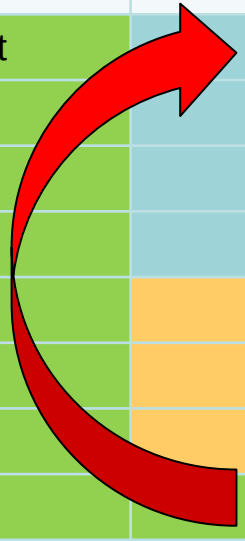
Dette er **hovedløkka** som går om og om igjen til vi er ferdige.

Etikett	Instruksjon	Adressedel
fortsett	LDA	ganger
	BRZ	exit
	SUB	en
	STA	ganger
	LDA	sum
	ADD	input
	STA	sum
	BRA	fortsett
exit	LDA sum	
	OUT	
	HLT	
sum	DAT	0
en	DAT	1
ganger	DAT	2
input	DAT	5



Her er linja som får oss ut av løkka.

Etikett	Instruksjon	Adressedel
fortsett	LDA	ganger
	BRZ	exit
	SUB	en
	STA	ganger
	LDA	sum
	ADD	input
	STA	sum
	BRA	fortsett
exit	LDA sum	
	OUT	
	HLT	
sum	DAT	0
en	DAT	1
ganger	DAT	2
input	DAT	5

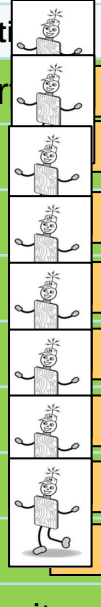


Denne delen bestemmer antall **ganger** vi legger sammen **input**.

Her legger vi til **input** i **sum** for hver gang vi er igjennom.

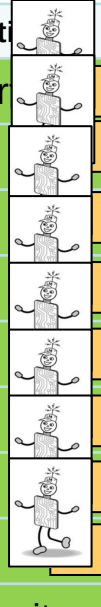
Her skriver vi ut svaret.

Eti	Instruksjon	Adressedel	Akkumulator	input	ganger	sum
for	LDA	ganger	2	5	2	0
	BRZ	exit				
	SUB	en	1			
	STA	ganger			1	
	LDA	sum	0			
	ADD	input	5			
	STA	sum				5
	BRA	fortsett				
exit	LDA sum					
	OUT					
	HLT					
sum	DAT	0				
en	DAT	1				
ganger	DAT	2				
input	DAT	5				



Første iteratsjon:
sum = 1 * 5

Eti	Instruksjon	Adressedel	Akkumulator	input	ganger	sum
for	LDA	ganger	1	5	1	5
	BRZ	exit				
	SUB	en	0			
	STA	ganger			0	
	LDA	sum	5			
	ADD	input	10			
	STA	sum				10
	BRA	fortsett				
exit	LDA sum					
	OUT					
	HLT					
sum	DAT	0				
en	DAT	1				
ganger	DAT	2				
input	DAT	5				



Andre iteratsjon:
sum = 2 * 5 = 10

Et	Instruksjon	Adressedel	Akkumulator	input	ganger	sum
for	LDA	ganger	0	5	0	10
	BRZ	exit				
	SUB	en				
	STA	ganger				
	LDA	sum				
	ADD	input				
	STA	sum				
	BRA	fortsett				
	LDA sum		10			
	OUT					
	HLT					
sum	DAT	0				
en	DAT	1				
ganger	DAT	2				
input	DAT	5				

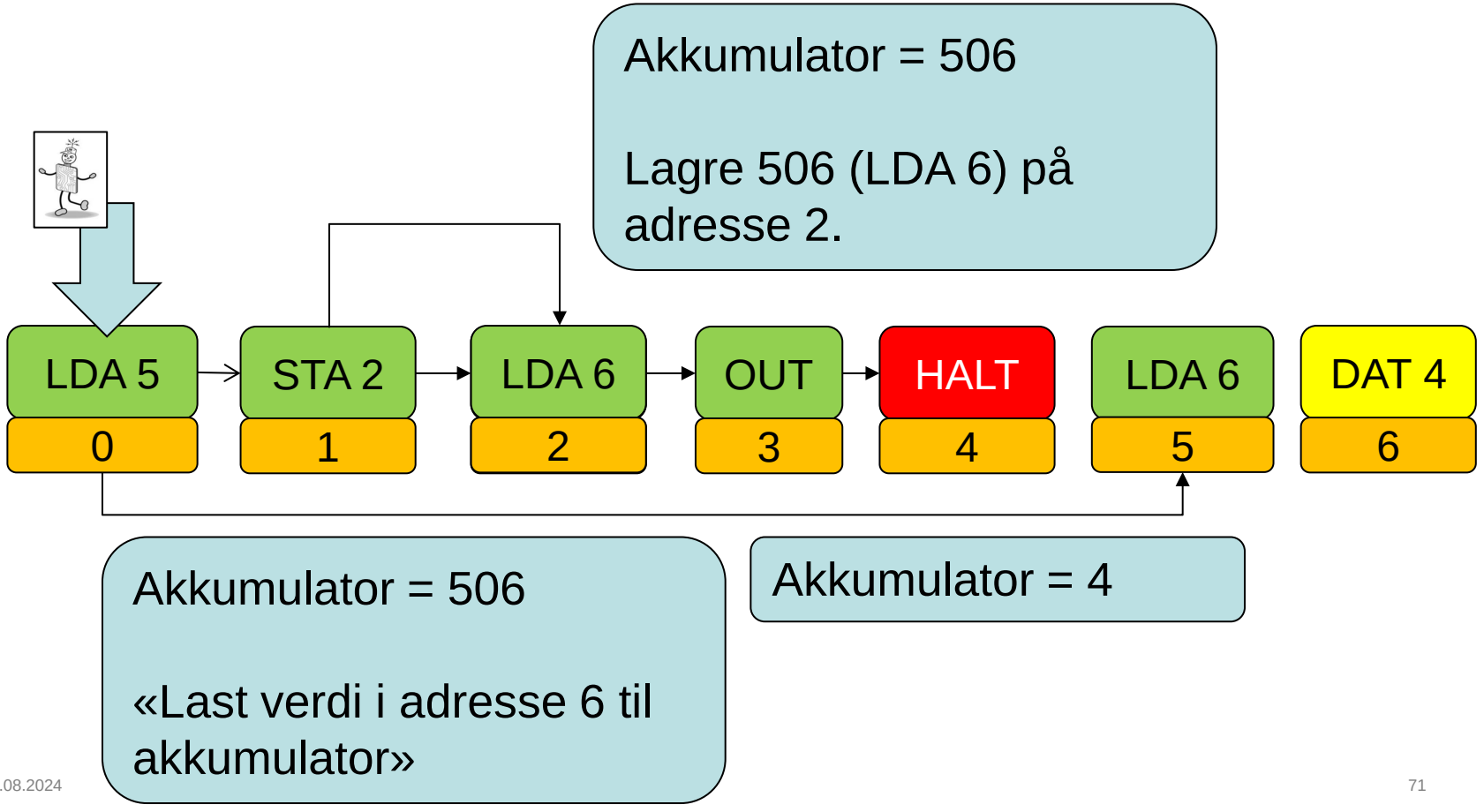
Kort forklart, hver gang LMC «ser» BRZ..

iterasjon	1	2	3
ganger	2	1	0
sum	0	5	5+5

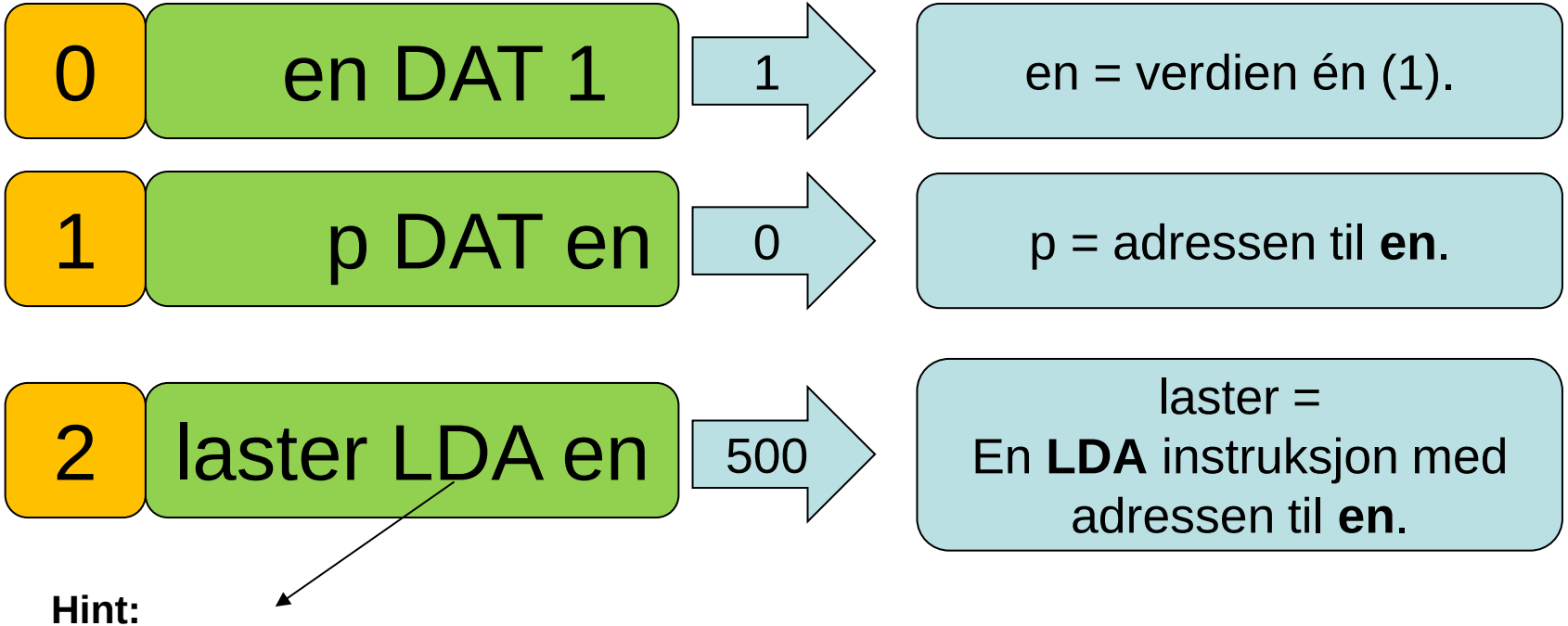
Selvmodifiserende kode

- LMC kan ikke skille på kode og data.
- LMC kan endre sin egen kode mens programmet kjører.

Selvmodifiserende Kode



Hvilket tall ligger i minnecellen?



Hint:
Instruksjonskoden til «LDA» er 5!

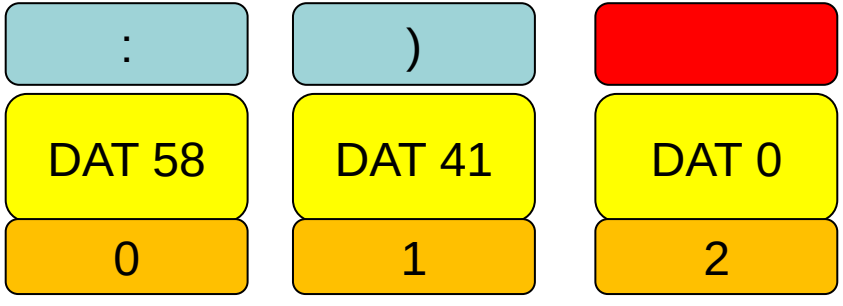
Adresse i minnet

Kan vi bruke dette til noe nyttig?

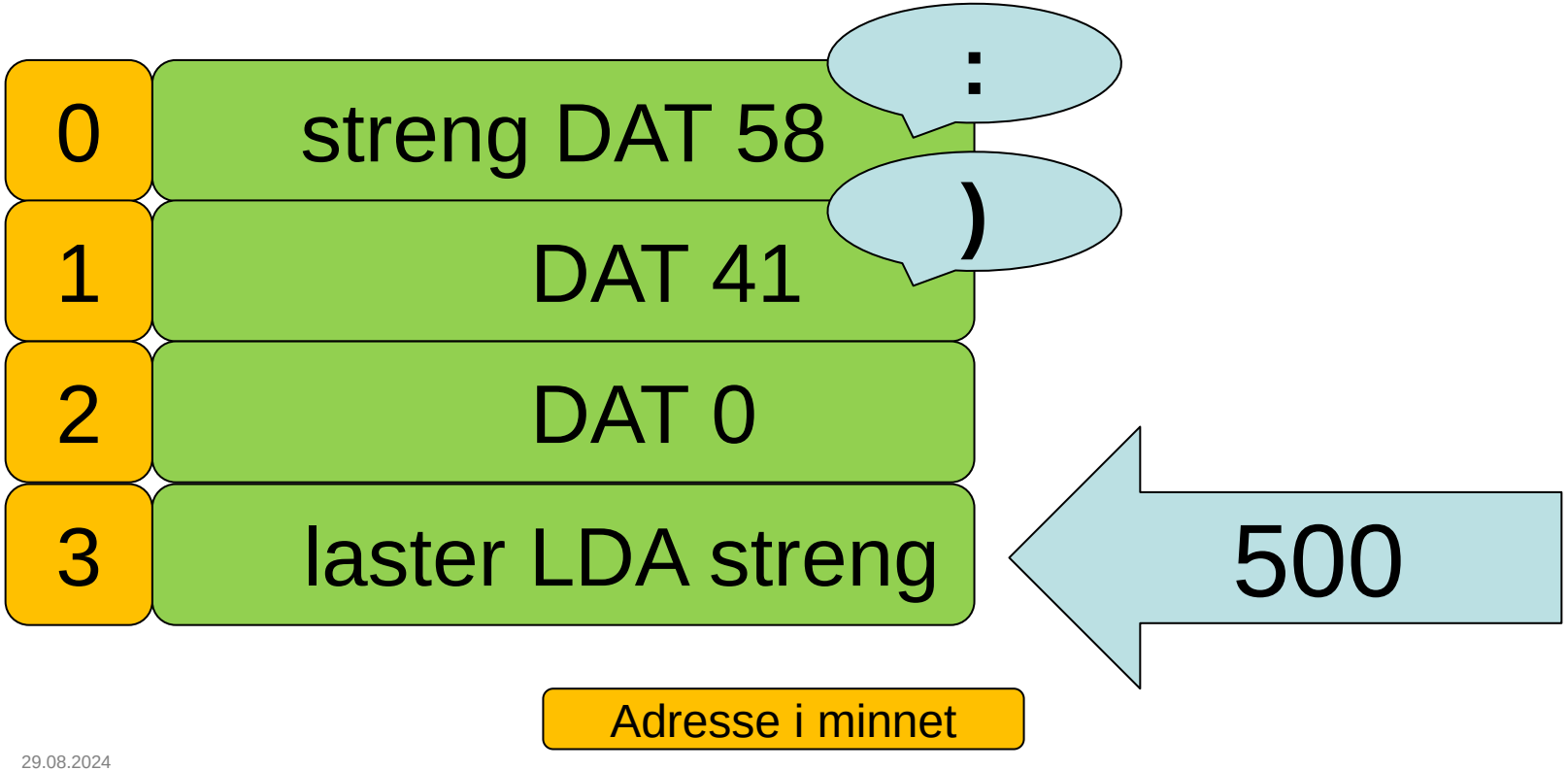
- Hva med å skrive ut litt tekst?

En ASCII Streng

- En streng er en rekke med ASCII tegn som ligger i minnet
- For å markere slutten på strengen bruker vi “0”
- Dette er strengen “:)”



Vi lager oss en instruksjon for å laste første tegn.



	Etikett	Instruksjon	Adressedel	Akkumulator
0		LDA	laster	508 (LDA 8)
1	print	STA last		
2	last	LDA 9		58
3		BRZ	exit	
4		OTC		
		LDA last		508 (LDA 8)
5		ADD en		509 (LDA 9)
		STA last		
6		BRA print		
7	exit	HLT		
8	streng	DAT	58	
9		DAT	41	
10		DAT	0	
11	laster	LDA 508	streng	
12	en	DAT	1	



	Etikett	Instruksjon	Adressedel	Akkumulator
0		LDA	laster	
1	print	STA last		
2	last	LDA 10		41
3		BRZ	exit	
4		OTC		
		LDA last		509 (LDA 9)
5		ADD en		510 (LDA 10)
		STA last		
6		BRA print		
7	exit	HLT		
8	streng	DAT	58	
9		DAT	41	
10		DAT	0	
11	laster	LDA	streng	
12	en	DAT	1	

	Etikett	Instruksjon	Adressedel	Akkumulator
0		LDA	laster	
1	print	STA last		
2	last	LDA 10		0
3		BRZ	exit	
4		OTC		
		LDA last		
5		ADD en		
		STA last		
6		BRA print		
7	exit	HLT		
8	streng	DAT	58	
9		DAT	41	
10		DAT	0	
11	laster	LDA	streng	
12	en	DAT	1	

Array'er

- Dette eksempelet er det samme som å lage seg et array med ASCII tegn i python, og skrive ut et og et til man ser en NULL.

```
1  streng = [":", ")", 0]
2
3  i = 0
4  while True:
5      if streng[i] == 0:
6          break
7
8      print(streng[i])
9
10     i = i + 1
```

Selvmodifiserende kode

- I moderne maskinarkitekturer er det typisk en beskyttelse som forhindrer at man modifiserer kode som data

Hva kan vi nå?

- Assembleren
  - Bruk av **DAT** og **etiketter**
- Assembler
  - Minnehåndtering (LDA & STA)
  - Aritmetikk (ADD & SUB)
  - Programflyt, løkker (BRA & BRZ & BRP)
- Eksempler
  - Telling, multiplikasjon, arrayer, selvmodifserende koder
- Strenger

Spørsmål?