LMC (Little Man Computer) Notater

Motivasjon til LMC programmering:

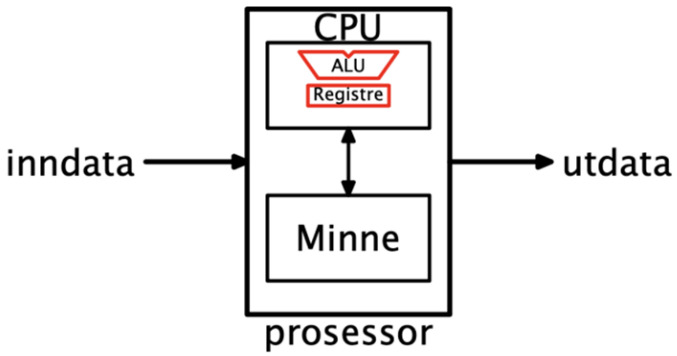
- LMC er en forenklet versjon av en datamaskin. Så for å forstå en datamaskin fungerer så er LMC en fin introduksjon uten å måtte forholde oss til alle detaljene.
- Datamaskinene bruker numerisk kode fordi CPUen bare forstår 0ere og 1ere. Men det er upraktisk for oss mennesker, så derfor øver vi på å kode assembler i LMC. Datamaskinen oversetter assemblerkoden for oss slik at den forstås av datamaskinen.
    - Numerisk kode: Kode hvor kodens «alfabet» bare inneholder tall. For eksempel binærkode.

OBS!:

- Husk å stoppe eksekveringen av programmet med HLT 000 på slutten.

Arkitekturen:

LCM har Von Neumann Arkitektur:



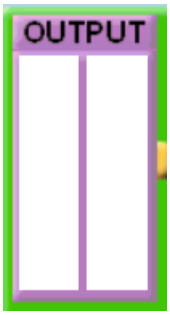
Inndata (Input)

- i dette kurset er inndataen ofte i form av et tall eller ascii verdi, men rent generelt kan det jo være tekst og bilder, museklikk etc.
- I LMC er dette der vi skriver input:

Utdata (Output)

- Det vi skriver ut. Enten tall, eller bokstaver og symboler(ASCII)

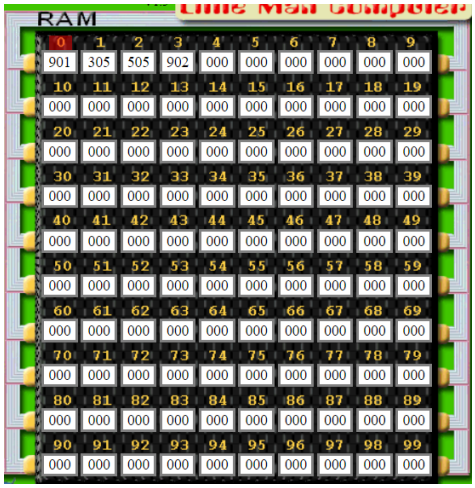
- I LMC er dette det som skrives ut med OUT (902) eller OTC (922) for ASCII-verdier. Det som skrives ut er det som er i akkumulatoren



Prosessoren

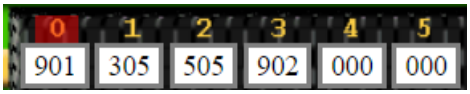
Minnet (RAM)

- Her lagres det data og programmer. Programmer er et sett med instruksjoner (instruksjonskoder) lagret i minnet.
- Minnet har 100 minneplasser/adresser. (0-99)
- Instruksjonene i minnet står som numerisk kode, men for oss mennesker er det lettere å forholde oss til assemblerkode. Vi kan skrive assemblerkode for eksempel i en tekstfil (filnavn.txt) og klikke på «load» og velge filen. Da lastes assemblerkoden inn i LMC og blri gjort om til instruksjoner som lagres i minnet.
    - Den første(øverste) instruksjonen i filen blir lagret i minneadresse 1 osv.



CPU (Central prosessing unit)

- CPUen henter en og en instruksjon eller data fra minnet.
- Instruksjonen inneholder instruksjonstallet, det første sifferet, og adressen til dataen den skal hente, de to siste sifrene.
    - Dette er adresse 1, 2, 3, 4 og 5 med noen instruksjonskoder.



Registre

- CPUen inneholder også register. Registrene er en liten mini lagring inni CPUen. Den lagrer de forskjellige komponentene ved instruksjonene og dataene:
  - Når CPUen henter en instruksjon fra minnet lagres komponentene av instruksjonen i disse registrene/mini minnene:
    - Instruksjonstallet, det første sifferet oppbevares i "Instruction register"

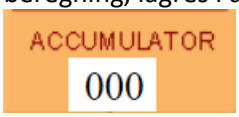
    - Adressen til data/instruksjon, de siste to sifferene lagres i "adress register».

  - I tillegg lagres også:
    - Hvilken instruksjon vi er på i programmet, altså hvor vi er i minnet, lagres i "program counter»:

    - Data vi har hentet eller data vi har beregnet, svar på en beregning, lagres i akkumulatoren, "accumulator"

ALU (Aritmetisk logisk enhet)

- Det er den som gjør alle beregninger i prosessoren. Etter en beregning lagres svaret i akkumulatoren
- Kan bare addere og subtrahere

Eksekveringen av programmer i LMC

1. Bruk verdien i programdelen som adresse til minnet og hent neste instruksjon der.
   - Øk samtidig programtelleren med 1 (Dette gjøres i ALUen)
2. Splitt instruksjonen og legg delene i instruksjonsregisteret og adresseregisteret.
3. Utfør det som instruksjonsregisteret angir.
4. Gjenta fra punkt 1

Assemblerkode-kodene / Instruksjonene i LMC

- Første siffer angir instruksjonen. For eksempel om siffer en er 1 skal vi addere.
- Siffer 2 og 3 angir hvor i minnet. Ved 199, så skal vi addere det vi har i akkumulatoren med verdien som er lagret i minneadresse 99.

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)

ASCII-tabeller

- Skrive ut bokstaver og tegn.
- Istedenfor å skrive ut med OUT 902 bruker vi OTC 922. Det matchende tegnet til det som står i akkumulatoren skrives ut med OTC.

Mest brukte:

Enklere ASCII-tabell?

10	LF	39	'	47	/	62	>	93	]	124	|
32		40	(	48	0	63	?	94	^	125	}
33	!	41	)	:		64	@	95	_	126	~
34	"	42	*	57	9	65	A	96	‘		
35	#	43	+	58	:	:		97	a		
36	$	44	,	59	;	90	Z	:			
37	%	45	-	60	<	91	[	122	z		
38	&	46	.	61	=	92	\	123	{		

De mest nødvendige tegnene, i base 10.

Allt:

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]
1	1	1	1	[START OF HEADING]
2	2	10	2	[START OF TEXT]
3	3	11	3	[END OF TEXT]
4	4	100	4	[END OF TRANSMISSION]
5	5	101	5	[ENQUIRY]
6	6	110	6	[ACKNOWLEDGE]
7	7	111	7	[BELL]
8	8	1000	10	[BACKSPACE]
9	9	1001	11	[HORIZONTAL TAB]
10	A	1010	12	[LINE FEED]
11	B	1011	13	[VERTICAL TAB]
12	C	1100	14	[FORM FEED]
13	D	1101	15	[CARRIAGE RETURN]
14	E	1110	16	[SHIFT OUT]
15	F	1111	17	[SHIFT IN]
16	10	10000	20	[DATA LINK ESCAPE]
17	11	10001	21	[DEVICE CONTROL 1]
18	12	10010	22	[DEVICE CONTROL 2]
19	13	10011	23	[DEVICE CONTROL 3]
20	14	10100	24	[DEVICE CONTROL 4]
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]
22	16	10110	26	[SYNCHRONOUS IDLE]
23	17	10111	27	[ENG OF TRANS. BLOCK]
24	18	11000	30	[CANCEL]
25	19	11001	31	[END OF MEDIUM]
26	1A	11010	32	[SUBSTITUTE]
27	1B	11011	33	[ESCAPE]
28	1C	11100	34	[FILE SEPARATOR]
29	1D	11101	35	[GROUP SEPARATOR]
30	1E	11110	36	[RECORD SEPARATOR]
31	1F	11111	37	[UNIT SEPARATOR]
32	20	100000	40	[SPACE]
33	21	100001	41	!
34	22	100010	42	"
35	23	100011	43	#
36	24	100100	44	$
37	25	100101	45	%
38	26	100110	46	&
39	27	100111	47	'
40	28	101000	50	(
41	29	101001	51	)
42	2A	101010	52	*
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
99	63	1100011	143	c
100	64	1100100	144	d
101	65	1100101	145	e
102	66	1100110	146	f
103	67	1100111	147	g
104	68	1101000	150	h
105	69	1101001	151	i
106	6A	1101010	152	j
107	6B	1101011	153	k
108	6C	1101100	154	l
109	6D	1101101	155	m
110	6E	1101110	156	n
111	6F	1101111	157	o
112	70	1110000	160	p
113	71	1110001	161	q
114	72	1110010	162	r
115	73	1110011	163	s
116	74	1110100	164	t
117	75	1110101	165	u
118	76	1110110	166	v
119	77	1110111	167	w
120	78	1111000	170	x
121	79	1111001	171	y
122	7A	1111010	172	z
123	7B	1111011	173	{
124	7C	1111100	174	|
125	7D	1111101	175	}
126	7E	1111110	176	~
127	7F	1111111	177	[DEL]

Eksempler / Oppgaver:

DAT og ASCII:

- Skriv ut «hei» i LMC
    - Skrive ut tegn i LMC, ASCII
    - DAT

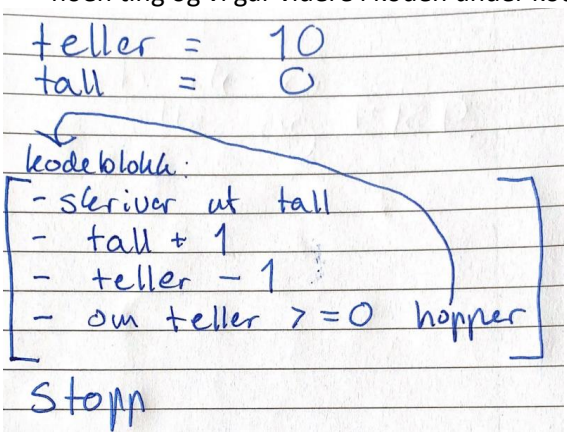
```
            INP             //Input fra oss
            STA tall        //Lagrer input i minneadressen til tall
            LDA tall        //Henter innholdet i minneadressen til tall
            OUT

skrivu      LDA H           //Henter innholdet i minneadressen til H
            OTC             //Skriver ut akkumulatoren som ASCII-tegn
            LDA e
            OTC
            LDA i
            OTC
            HLT             //stopp


tall        DAT             //minneadresse med navn tall
H           DAT 72          //minneadresse med navn H og innhold 72. 72 er ASCII-verdien til symbolet H.
e           DAT 101
i           DAT 105
```

Løkker:
Løkke: Løkker er når vi vil kjøre en kodeblokk flere ganger til vi når en viss tilstand hvor vi ikke vil at den skal kjøre mer.

- BRA, Hopper til angitt minneadresse
    - (ingen tilstand)
- BRP, Hopper til angitt minneadresse om akkumulatoren >= 0
    - Tilstand: om akk < 0 vil ikke løkken kjøre mer
- BRZ, Hopper til angitt minneadresse om akkumulatoren = 0
    - Tilstand: om akk ≠ 0 vil ikke løkken kjøre mer

1. **Løkke med bruk av teller og BRP**
   - Oppgaven: Vi vil skrive ett program som teller for oss fra 0 til 10.
   - Dette kan vi gjøre ved å bruke en teller (DAT) som teller ned fra 10 til -1 og BRP som hopper tilbake til starten av kodeblokken så lenge telleren(akkumulatoren) >= 0, og når telleren(akkumulatoren) er et negativt tall så fortsetter vi videre i koden.
     - BRP = Hopper(brancher) tilbake opp til øverst i kodeblokken om akkumulatoren >= 0
   - Løsning:
     - Vi har en teller = 10, fordi vi vil telle 10 ganger.
     - Vi har et tall = 0, som er tallet vi vil skrive ut for hver gang vi kjører kodeblokken.
     - Når vi har tellet 11 ganger(0-10) vil vi ikke at løkken skal kjøre mer, så tilstanden vår blir når teller er negativ skal vi gå videre i koden.
     - **Kodeblokken:**
       - I kodeblokken henter vi ut tall som vi skriver ut.
       - Så tar vi tall + 1 siden neste gang vi kjører kodeblokken vil vi skrive ut 2, (eller det som er en større enn forrige gang vi kjørte kodeblokken.)
       - Så tar vi telleren – 1, fordi da vet vi at vi skal telle 9 ganger til, (eller skrive ut en gang mindre enn forrige gang vi kjørte kodeblokken.)
       - Så sjekker vi telleren med BRP, er telleren større eller lik 0? Ja, den er 9, så da hopper vi opp igjen til starten av kodeblokken.

     - Når telleren når -1, så er den ikke større eller lik 0, så dermed vil BRP ikke gjøre noen ting og vi går videre i koden under kodeblokken. I dette tilfellet stopp.



- Her gir jeg starten av kodeblokken navnet(etiketten) lok, for løkke, og slutten av kodeblokken er der det står BRP lok.
- BRP lok, betyr at om teller(akkumulatoren) >= 0 så hopper vi opp til lok.

```
lok         LDA   tall    //henter ut verdien i minneadressen teller til akkumulatoren
            OUT           //Skriver ut det i akkumulatoren (altså telleren)
            LDA   tall    //henter ut verdien i minneadressen teller til akkumulatoren
            ADD   en      //Vi adderer det i akkumulatoren med det i minneadressen en(1)
            STA   tall    //Vi lagrer svaret i minneadresse teller
            LDA   teller  //Vi henter it verdien i minneadresse tall til akkumulatoren
            SUB   en      //Vi substraherer det i akkumlulatoren med det i minneadresse en(1)
            STA   teller  //Vi lagrer svaret i minneadresse tall
            LDA   teller  //henter ut verdien i minneadressen tall til akkumulatoren
            BRP   lok     //om akkumulatoren er >= 0 nå så hopper vi ikke tilbake opp til lok.

            HLT           //programmet stopper

teller      DAT   10
tall        DAT   00
en          DAT   01
```

2. **Løkke med bruk av Input og BRZ.**
   - Ved å bruke denne type løkke med BRZ kan vi/bruker bestemme om vi vil at kodeblokken skal kjøre flere ganger eller ikke
     - BRZ = Hopper(brancher) tilbake opp til øverst i kodeblokken om akkumulatoren er 0
   - Da kan vi bruke BRZ. Om vi/bruker skriver inn 0 og så sjekker vi med BRZ, så vil programmet hoppe tilbake opp, om vi skriver inn noe annet, for eksempel 1, så gjør BRZ ingenting og vi går videre i programmet. I dette tilfellet stopper programmet.

```
lok         LDA   tall    //henter ut verdien i minneadressen teller til akkumulatoren
            OUT           //Skriver ut det i akkumulatoren (altså telleren)
            LDA   tall    //henter ut verdien i minneadressen teller til akkumulatoren
            ADD   en      //Vi adderer det i akkumulatoren med det i minneadressen en(1)
            STA   tall    //Vi lagrer svaret i minneadresse teller
            INP           //Om bruker skriver inn 0 forsetter vi om de skriver inn noe annet feks 1, forsetter vi ikke
            BRZ   lok     //om akkumulatoren er =0 nå så hopper vi ikke tilbake opp til lok.

            HLT           //programmet stopper

teller      DAT   10
tall        DAT   00
en          DAT   01
```

3. **Løkke med BRA**

```
      LDA   tall //henter verdi fra minneadresse tall til akkumulatoren

lok   SUB   en   //trekker 1 fra det i akkumulatoren
      OUT        //skriver ut akkumulatoren
      STA   tall  //lagrer akkumulatoren i minneadresse tall
      BRZ   stopp //om akkumulatoren(tall), nå er 0 hopper vi til etiketten stopp
      BRA   lok   //hopper til etiketten lok

stopp HLT

tall  DAT   20
en    DAT   01
```

4. Multiplisere to tall i LMC:

```
//Tar inn to tall og multipliserer dem sammen
//Siden vi bare kan plusse må vi plusse og minuse, må vi plusse tall1 på
//tall1, tall2 ganger.
//Si at tall1=2 og tall2=3. Da skal vi regne 2 + 2 + 2.
//Dette kan vi gjøre i en løkke, samtidig som vi trekker fra 1 på tall2
//helt til tall2 = 0.
//OBS! Fjern disse kommentarene øverst om du skal kjøre koden.

      INP              //tar input fra bruker: tall1
      STA   tall1      //Lagrer inputen i tall1(der tall1 har adresse)
      INP              //tar input fra bruker: tall2
      STA   tall2      //lagrer inputer i tall2(der tall2 har adresse)


lok   LDA   tot        //henter totalen fra minnet
      ADD   tall1      //plusser totalen og tall1
      STA   tot        //lagrer resultatet i tot(adressen til tot)
      LDA   tall2      //henter tall2 fra minnet
      SUB   en         //tall2 - 1
      STA   tall2      //lagrer svaret i tall2(adressen til tall2)
      BRP   lok        //Så lenge tall2 >= 0 hopper vi tilbake opp til lok Når tall2 = 0, fortsetter koden under

      LDA   tot        //henter totalen fra minnet
      SUB   tall1      //vi har gjort tall1 + tall2 en gang for mye over pga. den hoppet opp til lok også når tall2 = 0
      STA   tot        //lagrer totalen i minnet
      OUT              //skriver ut det i akkumulatoren
      HLT

tall1 DAT   0
tall2 DAT   0
tot   DAT   0
en    DAT   1
```