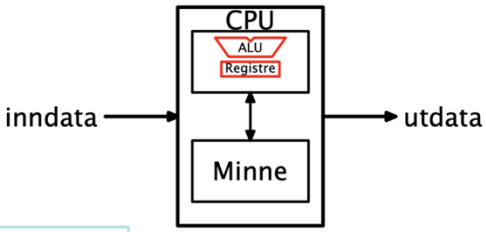
Notater fra IN1020 gruppetime uke 36

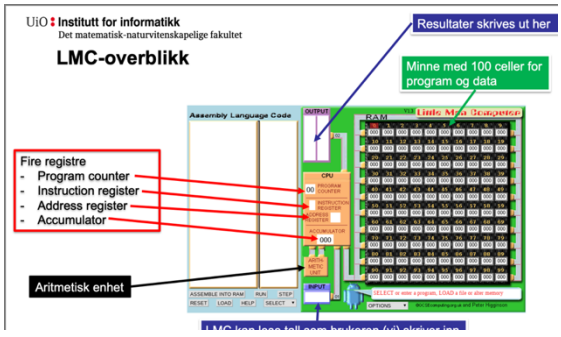
Oppbygging av vanlig datamaskin:
Dette er en enkel modell for hvordan en datamaskin er bygd opp.
Den består av følgende komponenter:



- **Input**: Det vi gjør på maskinen, f.eks. tastetrykk, museklikk, touch etc.
- **Output**: Maskinen «reagerer» på vår input og gir ut et resultat, f.eks. vise noe på skjermen.
- **CPU**: Hjernen til datamaskinen – den gjør alt, men på det mest basic nivået, så utfører den enkle instruksjoner. Den består av 2 komponenter:
    - **ALU**:
        - Den aritmetiske/logiske delen av CPUEN
        - Utfører matematiske og logiske beregninger
        - I en vanlig datamaskin kan den gjøre alle vanlige beregninger, både for flyttall og for heltall.
    - **Registre**:
        - Små minnelager for å kunne fortere hente informasjon lagret i minnet.
        - Det tar egentlig «lang» tid (in computer terms) å hente data fra minnet, men registre gjør de mer tilgjengelig og har data som skal brukes lett tilgjengelig.
- **Minne**: Plass for å lagre både instruksjoner og data.

LMC:
LMC er en forenklet versjon av en datamaskin. Den inneholder de samme komponentene, men disse er svært enkle i forhold til en ekte datamaskin. Disse er:

- **Input**: Tall
- **Output**: Tall/bokstaver
- **CPU**:
    - **ALU:** Utfører beregninger på inputtallene, men kun + og – og ingen logikk.
    - **Registre**: Har 4 registre, der hver av dem holder på en spesifikk verdi.
        - **Program counter**: hvilken instruksjon i minnet som skal utføres neste gang
        - **Instruction register**: koden til instruksjonen som skal utføres (5 – LDA)
        - **Address register**: adressedelen til koden til instruksjonen (99 – minne 99)
        - **Accumulator**: svaret kommer hit



Hvorfor driver vi med LMC programmering?

- LMC gir oss en fin introduksjon til assemblerkoding og hjelper oss med å forstå hvordan datamaskinen funker uten å måtte forholde oss til alle detaljene.
- LMC ≈ assembler
- En datamaskin bruker numerisk kode – CPUEN forstår bare en sekvens av 0ere og 1ere. Men å programmere med numerisk kode er ekstremt upraktisk for oss mennesker, så vi bruker assemblerkode istedenfor.
- I assembler spesifiserer vi både operasjonen som skal gjøres (f.eks. subtraksjon) og de registrene vi vil utføre operasjonen på.
- Datamaskinen oversetter fra assemblerkode til maskinkode for at den skal forstå instruksjonene som blir gitt i koden.
- Eksempel (ikke pensum i IN1020, dette er bare for å illustrere)
    - Vi vil subtrahere 1 fra et annet tall som er lagret i et register R3.
    - Assemblerkoden blir: SUB R3, R3, #1
    - Maskinkode: 1110 0010 0100 0011 0011 0000 0000 0001
        - Vanlig å bruke heksadesimal også for å unngå lange tallrekker.

Tips og anna til LMC:

- Etter du har skrevet kode, se inn i RAM og se om minneadressene har blitt fylt ut. Hvis du ser at kun noen må minneadresser har blitt fylt ut og koden din er lang, så har du skrivefeil i koden. Du kan og se om koden blir "assemblet" i den boksen til høyre for kodefeltet.

- Husk å RESET etter hver RUN (etter du har kjørt programmet) slik at CPUen ikke husker verdien fra forrige kjøring og forstyrrer det nye resultatet.

- Husk å ha med HLT (stopp) i koden slik at vi signaliserer at koden skal være ferdig å kjøre.

- Dere skal både klare å kode med «tall» og «bokstaver» (sånn som i tabellen.)

Enkelt kodeeksempel 1:

```
    INP // tar input
    STA a // lagrer input i a
    INP // ny input
    ADD a // akkumulator + a
    SUB en // resultatet - 1
    OUT // printer ut
    HLT
a   DAT 0
b   DAT 0
en  DAT 1
```

Enkelt kodeeksempel 2:

```
    INP // tar input
    STA 99 // lagrer input i celle 99
    BRZ done // hopper til done om 0
her INP // ny input
    SUB 99 // input 99 - input 98
    OUT
    BRP her // ny input om positivt resultat
    OUT
done HLT
```

Kodeeksempel:

Oppgave: Skal lese inn 2 tall og multiplisere dem sammen.

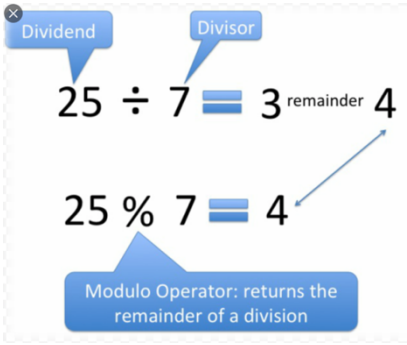
	INP		Leser input
	STA a		Lagrer inputen som a
	INP		Ny input
	STA b		Lagrer inputen i variabelen b – fortsatt i akkumulatoren
loekke	LDA	tot	Starter løkke og henter tallet i tot fra minnet
	ADD	a	Add a til a b ganger = multiplikasjon med addisjon
	STA	tot	Lagrer tot+a i totalen
	LDA	b	Henter b
	SUB	en	Har nå adda sammen a en gang
	STA	b	Lagrer b sin nye verdi til neste runde
	BRP	loekke	Hvis b >= 0 så er vi fortsetter vi løkka
	LDA	tot	Henter ut totalen
	SUB	a	Minus b fordi vi har gjort de en gang for mye ovenfor (b=0 nono)
	STA	tot	Lagrer totalen
	OUT		Printer
	LDA	d	
	OTC		
	LDA	o	
	OTC		
	LDA	n	
	OTC		
	LDA	e	Stopper
	HLT		
			Huske å definere variabelen a med startverdi 0
a	DAT	0	Definere variabel
b	DAT	0	Definere konstant 1
tot	DAT	0	
en	DAT	1	
d	DAT	68	
o	DAT	111	
n	DAT	110	
e	DAT	101	

Kodeeksempel:

Lag et program for LMC som leser inn to positive tall a og b og beregner a mod b (resten vi får når vi deler a med b). Print ut «Done».

```
INP
STA dividend
INP
STA divisor

LDA dividend
positiv SUB divisor
BRP positiv

ADD divisor
OUT

loekke LDA tekst
BRZ ferdig
OTC
LDA loekke
ADD en
STA loekke
BRA loekke
Ferdig HLT

a DAT 0
b DAT 0
en DAT 1
tekst DAT 10
DAT 68
DAT 111
DAT 110
DAT 101
DAT 0
```

Logikken bak koden:



Eksempel på selvmodifiserende kode

Selvmodifiserende kode: Når vi bruker tall som instruksjoner (istedenfor bokstaver) slik at vi kan endre instruksjonen for å lese av ny minneboks mens programmet kjører:

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)

[1]

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com