Ukesoppgaver - Innføring til LMC

Oppgave 1

I denne oppgaven skal vi se på hvordan vi går fra maskinkode (siffere mellom 000 og 999 som vi programmerer rett i minnet på LMC) til assemblerkode. Vi skal også se på bruk av **DAT** for mellomlagring av data. Dette vil hjelpe dere i arbeidet med LMC.

I første forelesning startet vi med å legge sammen to tall på denne måten:

Minne adresse	Instruksjon Data	Kommentar	Assemblernotasjon
0	901	Les første tall til akkumulator	INP
1	306	Akkumulator til adresse 6	STA 6
2	901	Les andre tall til akkumulator	INP
3	106	Beregn akkumulator + data i adresse 6	ADD 6
4	902	Skriv ut akkumulator	OUT
5	000	HALT	HLT
6	-	Mellomlagret tall	-
7-99	-	Ubrukt minne	

(a) Hvilke av adressene inneholder instruksjoner, og hvilke inneholder data?

(b) Legg inn instruksjonene i LMC. Ikke bruk assembler! Kjør programmet (**RUN**) og sjekk at koden kan legge sammen to tall. Prøv gjerne med forskjellige negative tall. Bruk **RESET** mellom hver kjøring for å reinitialisere programteller og andre registre.

(c) Hvorfor trenger vi et sted i minnet til å mellomlagre et av tallene vi skal legge sammen?

(d) Skriv om programmet til assembler. Dette gjør du ved å skrive inn programmet i tabellen over, men på assembler-form. Bruk gjerne assemblernotasjonen foreslått i tabellen. Skriv inn koden i det venstre tekstfeltet i LMC. Bruk så **SUBMIT** for å generere maskinkode. Sjekk at maskinkoden ser OK ut, før du bruker **ASSEMBLE TO RAM** for å legge inn maskinkoden i minnet til LMC. Start som normalt.

(e) Hva syntes du er enklest å forstå - assembler eller maskinkode?

(f) Utvid programmet til å legge sammen tre tall. Prøv deg fram i 10-15 minutter før du eventuelt ser på fasiten under, eller få hjelp av gruppelærer. *Ser du noe spesielt som gjør at denne "endringen" i programmet vårt er vanskelig?*

(g) Svaret på spørsmålet over er som følger: Når man utvider programmet til å legge sammen tre tall, må man passe på at adresse 6 brukes til mellomlagring. Legger man til instruksjoner må man altså flytte plasseringen til mellomlagret data. I tillegg må alle instruksjoner som refererer til adresse 6 endres. Under følger løsningsforslaget for å legge sammen tre tall. Legg merke til hvordan vi mellomlagrer summen i adresse 9 og hvordan dette er reflektert i **STA** og **ADD** instruksjonene.

Minne adresse	Instruksjon Data	Kommentar	Assemblernotasjon
0	901	Les første tall til akkumulator	INP
1	309	Akkumulator til adresse 9	STA 9
2	901	Les andre tall til akkumulator	INP
3	109	Beregn akkumulator + data i adresse 9	ADD 9
4	309	Akkumulator til adresse 9	STA 9
5	901	Les tredje tall til akkumulator	INP
6	109	Beregn akkumulator + data i adresse 9	ADD 9
7	902	Skriv ut akkumulator	OUT
8	000	HALT	HLT
9	-	Mellomlagret tall	-
10-99	-	Ubrukt minne	

(h) En mye bedre måte å organisere datalokasjoner på er med **DAT**. **DAT** er en pseudoinstruksjon som hjelper oss å se hvilke adresser som brukes til data. For å bruke dette i assembler gir vi datalokasjoner navn og en startverdi, som for eksempel:

Assembler notasjon	Kommentar
temp DAT 0	En vilkårlig adresse i minnet som heter **temp**. Initialiseres til '0'.

Assembler-instruksjoner som **STA** bruker nå *navnet* på lokasjonen i minnet i stedet for en hard-kodet adresse. For eksempel kan vi bruke følgende kode for å legge sammen to tall:

Assembler notasjon	Kommentar
INP	Les første tall til akkumulator
STA temp	Akkumulator til adresse **temp**
INP	Les andre tall til akkumulator
ADD temp	Beregn akkumulator + data i adresse **temp**
OUT	Skriv ut akkumulator
HLT	Stopp
temp DAT 0	Mellomlagret tall med startverdi 0

Legg merke til at *assembleren* finner ut en passende plass i minnet til å legge vår mellomverdi **temp** på når dere bruker **SUBMIT** med koden over. I maskinkoden kan dere se hvilken adresse **temp** har fått, men dere trenger ikke tenke så mye på dette lenger når dere utvider programmet.

Modifiser programmet for å legge sammen tre tall på samme måte.