INFÖR LABORATION 3

Följande uppgifter ur kursboken ska vara utförda som förberedelse för laborationen. Du ska på begäran av laborationshandledare redogöra för dessa.

Uppgifter i kursboken		
8.1 8.2 8.3 8.7 8.9 8.11 8.17 8.21 (endast DECA)	(8.13) (8.18)	(9.3) (9.4) (9.5) (9.6) (9.7)

Uppgifter inom parentes är inte strikt obligatoriska, men om man gjort dem blir arbetet med de obligatoriska uppgifterna lättare.

Nedanstående hemuppgifter i detta PM som ska vara utförda innan laborationen påbörjas.

Hemuppgifter			
3.A	3.B	3.C	3.D

LABORATION 3

INLEDNING

Denna laboration består av fem deluppgifter. Under uppgifterna 3.1, 3.2 och 3.3 har du möjlighet att bekanta dig med laborationssystemet och lära dig att använda de grundläggande funktionerna för att därefter, under uppgifterna 3.4 och 3.5 självständigt implementera och testa två helt nya FLISP-instruktioner.

Laborationssystemet består av två delar:

- Dataväg med FLISP styrenhet, "DV-modul" eller kortare "dataväg" (LU3-FLISP-DV)
- Kopplingsplatta för att bilda styrsignaler externt (LU3-FLISP-SE)

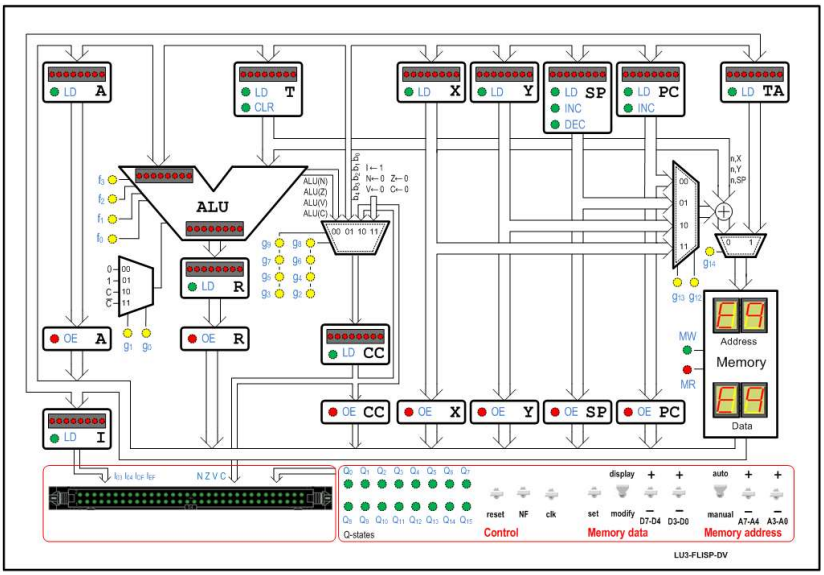
Med DV-modulen kan du detaljstudera hur styrsignalsekvenser sätts samman i maskininstruktioner för FLISP. Kopplingsplattan ansluts till DV-modulen via en 64-polig flatkabel. Med kopplingsplattan, som innehåller ett antal AND/OR-nät, kan du bilda styrsignaler och på så sätt skapa godtyckliga styrsignalsekvenser, dvs. nya instruktioner för FLISP.

Två nya FLISP-instruktioner ska konstrueras och testas. Det är helt nya instruktioner så de finns inte i den ordinarie instruktionslistan:

`MOVE`	`#Data,Adr`	*"move immediate data to memory"*
`CMJEQ`	`#Data,Adr`	*"compare register and data, jump if equal"*

BESKRIVNING AV LABORATIONSSYSTEMET

Följande bild visar LU3-FLISP-DV, eller kortare ”DV-modul” (datavägsmodul):



LU3-FLISP-DV, ”DV-modul”

OBS! På laborationssystemet har trycket av väljaren för CC-registret blivit fel. Ingångarna 3 och 4 är här växlade. Detta gäller dock INTE funktionen.

DV-modulen har dessutom en inbyggd styrenhet för att kunna utföra samtliga FLISP:s instruktioner, 256 bytes primärminne och funktioner för att kunna övervaka och modifiera minnesinnehållet.

De odefinierade operationskoderna $E0_{16}$ och FF_{16} hanterar styrenheten helt enligt FLISP-specifikationen, dvs. med undantagshantering.

De odefinierade operationskoderna 03_{16}, 04_{16}, DF_{16} och EF_{16} hanteras på följande sätt av laborationsenheten:

Då någon av dessa operationskoder finns i instruktionsregistret:

1. aktiveras respektive signal I_{03}, I_{04}, I_{DF} eller I_{EF} till kopplingsplattan
2. alla interna styrsignaler till datavägen inaktiveras, styrsignalerna hämtas nu i stället från kopplingsplattan.

Datavägen och kopplingsplattan kan därför användas för att skapa nya instruktioner för FLISP genom att styrsignalsekvenser bildas med hjälp av Q- och I-signaler som via AND/OR näten återkopplas från kopplingsplattan till datavägen i form av styrsignaler.

För ytterligare beskrivningar av datavägens indikatorer för register, styrsignaler och tillståndssignaler hänvisas till kursboken och annan dokumentation av FLISP.

Laborationsenheten kan styras med ett antal strömställare:

Control	
reset	DV-modulen försätts i återställningstillstånd Q0
NF	DV-modulen klockas fram till nästa FETCH-fas (stega en instruktion)
clk	en klockpuls ges till DV-modulen
Memory address	
auto	innehållet i minnesenhetens adressindikator är det värde som kopplats till minnesenheten med styrsignaler g12, g13 och g14
manual	minnesenhetens adressindikator sätts med hjälp av omkopplarna A7-A4, de fyra mest signifikanta adressbitarna och A3-A0, de fyra minst signifikanta adressbitarna.
Memory data	
display	innehållet i minnesenhetens dataindikator ges av innehållet på adressen som finns i addressindikatorn
modify	minnesenhetens dataindikator sätts med hjälp av omkopplarna D7-D4/D3-D0
set	om omkopplaren står i modify-läge förs innehållet i dataindikatorn in i DV-modulens minne på den adress som anges i adressindikatorn.

Följande bild visar LU3-FLISP-SE, eller kortare "kopplingsplattan":

LU3-FLISP-SE, ”KOPPLINGSPLATTA”

Kopplingsplattan har tre sektioner med ingångar, dvs. signaler som kommer *från* DV-modulen, dessa är:

OP code, då någon av de odefinierade operationskoderna 03_{16},04_{16},DF_{16} eller EF_{16} finns i DV-modulens instruktionsregister aktiveras också motsvarande signal I_{03}, I_{04}, I_{DF} eller I_{EF} till kopplingsplattan. Det finns 16 stiftlist för varje signal.

State, anger vilket exekveringstillstånd Q_4-Q_{15} som DV-modulen är i. Även här finns 16 stiftlist för varje signal.

Flags, (N, Z, V, C) från DV-modulens CC-register. Dessa kan användas för att bilda enkla flaggvillkor. Varje signal kan tas ut från något av de fyra stiften på den intilliggande stiftlisten.

Insignalerna kopplas med hjälp av 24 AND/OR-grindnät för att bilda styrsignaler för DV-modulen. Ingångarna på AND-grindarna har en så kallad *weak pull-down*, så logiknivån är noll på en ingång som inte är ansluten. Varje utgång kan kopplas till maximalt tre olika styrsignaler om så skulle krävas. Det är dock rekommenderat att endast en styrsignal kopplas per ett AND/OR-nät. Använd alltså endast ett av de tre stiften på utgången från varje AND/OR-nät.

Styrsignalerna kopplas tillbaks till DV-modulen via sektionen *Control signals* som också har ljusdiodindikatorer för att indikera styrsignalernas nivå.

LABORATIONSUPPGIFT 3.1

I denna uppgift ska du manuellt lägga in en instruktionssekvens och resetvektor i DV-modulens minne och därefter kontrollera sekvensens funktion genom att utföra programmet cykelvis (klockcykel för klockcykel).

HEMUPPGIFT 3.A

Disassemblera, dvs. tolka minnesinnehållet och använd FLISP:s instruktionslista för att komplettera tabellen med mnemonics. Ange också instruktionens sista tillstånd i exekveringsfasen (det tillstånd i vilket signalen NF aktiveras).

Adress	Maskin-kod	Assemblerkod	Instruktionens NF-tillstånd
20	F0	LDA #7E	
21	7E		Q5
22	07	INCA	Q6
23	33	JMP 22	
24	22		Q5
FF	20		

VID LABORATIONSPLATSEN

Så här skriver du in data manuellt till FLISP:s primärminne:

- Ställ Memory address omkopplare i läge manual, och ställ in adressen 20 med omkopplarna A7-A4/A3-A0.
- Sätt Memory data omkopplaren i läge modify och ställ in data F0 med omkopplarna D7-D4/D3-D0. Tryck på set-omkopplaren för att skriva in värdet F0 på adress 20 i minnet.

Upprepa förfarandet för varje adress tills hela instruktionssekvensen och RESET-vektorn lagts in i minnet.

Du kontrollerar instruktionssekvensen på följande sätt:

- Ställ Memory address-omkopplaren i läge auto, Memory data-omkopplaren i läge display.
- Kontrollera att DV-modulen är i tillstånd Q_0, tryck reset annars.
- Utför instruktionssekvensen cykelvis genom att ge klocksignaler, dvs. tryck in clk-omkopplaren.

LABORATIONSUPPGIFT 3.2

I denna uppgift får du goda tips om hur du testar ett program i DV-modulen.

- Utför programmet instruktionsvis ("stega" igenom programmet)
- Utför programmet utan uppehåll (exekvera programmet)

HEMUPPGIFT 3.B

Använd FLISP:s instruktionslista och komplettera följande tabell genom att översätta sekvensen av mnemonics till maskinkod, dvs. assemblera programmet.

Adress	Maskin-kod	Assemblerkod		
20	F0	L1:	LDA	#3
21	03			
22	08	L2	DECA	
23	E1		STA	$10
24	10			
25	24		BEQ	L1
26	FA			
27	33		JMP	L2
28	22			
FF	20			

VID LABORATIONSPLATSEN

Programmet DigiFlisp har en inbyggd terminalfunktion som kan användas för att kommunicera med DV-modulen via en USB-anslutning.

- Starta DigiFlisp och välj Console-fliken längst ner i fönstret. Ange den port som DV-modulen är ansluten till, i textfältet till höger om Connect. Porten anges på formatet '**\\.\COMxx**', där '**xx**' är numret på enheten 'USB Serial Port' (enligt Windows-programmet *Device Manager*).
- Klicka på Connect för att ansluta till DV-modulen. Console-delen av fönstret byter färg till blått när anslutningen lyckats. Ge kommandot **'i'** i det blå terminalfönstret (*interactive mode*) från tangentbordet.
- Skriv manuellt in maskinprogrammet i DV-modulens minne på samma sätt som i föregående laborationsuppgift, eller använd kommandot **'wmYYXX'** i terminalfönstret. Kommandot skriver det hexadecimala värdet **XX** till hexadecimala adressen **YY**. (Man behöver inte trycka Enter efter kommandot.)
- Ställ DV-modulens Memory address omkopplare i läge manual, och ställ in adressen 10 med omkopplarna A7-A4/A3-A0. Genom att observera innehållet på denna adress kan du senare se att `DECA`-instruktionen i programmet utförs korrekt.
- Gör reset på DV-modulen.

Ge kommandot **'n'** i terminalfönstret (*step instruction*) från tangentbordet. Om kommandot **'n'** inte fungerar, använd istället omkopplarna på DV-modulen som i förra uppgiften**.** För varje gång du ger detta kommando utförs en hel instruktion, på detta sätt blir det enklare att följa programutförande genom instruktioner som är kända att fungera korrekt.

- Stega instruktionsvis några varv i programslingan, observera innehållet på adress 10_{16}.
- Placera markören i terminalfönstret och ge kommandot **'e'** (execute). Programmet utförs nu utan att stanna. Studera speciellt innehållet på adress 10_{16} i minnet. För att avbryta skriv **'e'** igen.

LABORATIONSUPPGIFT 3.3

I den här uppgiften ska du köra samma program som i laborationsuppgift 3.2, fast den här gången ska du använda dina egna implementationer av instruktionerna JMP och DECA.

- Eftersom kopplingsplattan endast kan användas för att ange styrsignaler för OP-koderna 03_{16}, 04_{16}, DF_{16} och EF_{16}, kan du låta JMP använda OP-koden 03_{16}. Byt därför ut motsvarande OP-kod i programmet från laborationsuppgift 3.2.
- Implementera din lösning för uppgiften 8.11 (JMP) genom att koppla upp den på kopplingsplattan.
- Bekräfta att programmet får samma beteende som det hade när du körde det i föregående laborationsuppgift.
    - Utför instruktionssekvensen för programmet cykelvis genom att ge klocksignaler, tills du ser operationskoden 03_{16} i instruktionsregistret.
    - Kontrollera nu, för exekveringsfasen dvs. Q_4, att styrsignalerna aktiveras korrekt.
- Byt ut OP-koden för DECA i testprogrammet mot 04_{16}.
- Koppla upp din implementation av instruktionen DECA (uppgift 8.21) på kopplingsplattan. Både DECA och JMP ska alltså vara uppkopplade samtidigt.
- Bekräfta att programmet fortfarande har samma beteende som tidigare.
    - Utför instruktionssekvensen för programmet cykelvis genom att ge klocksignaler, tills du ser operationskoden 04_{16} i instruktionsregistret.
    - Kontrollera nu, för varje cykel i exekveringsfasen dvs. Q_4 och uppåt, att styrsignalerna aktiveras korrekt.
- Visa upp din koppling för en handledare.

LABORATIONSUPPGIFT 3.4

Anta att vi vill skriva värdet 2 till minnet, adress 10_{16}. Om vi inte vill ändra något av de ordinarie "synliga" registren hos FLISP, dvs. A, X, Y, SP eller CC för att dessa redan innehåller värden vi inte vill skriva över, skulle vi behöva skriva en programsekvens som den följande:

```
PSHCC
PSHA
LDA   #2
STA   $10
PULA
PULCC
```

Den här uppgiften handlar om att implementera en instruktion som förenklar en sådan kopiering, så att vi *inte* behöver använda stacken som i ovanstående sekvens. Den önskade kopieringen kan med denna nya instruktion, MOVE, utföras på följande sätt, vilket sparar exekveringstid och gör att koden får plats i minnet på bara tre byte:

```
MOVE  #2,$10
```

MOVE-instruktionen specificeras enligt följande:

MOVE #Data,Adr

RTN	Data → M(Adr)
Flaggor	Påverkas ej.
Beskrivning	Initierar en minnescell med en konstant.

Detaljer:

Instruktion **MOVE**	Adressering				Operation	Flaggor			
	metod	OP	#	~		N	Z	V	C
`MOVE #Data,Adr`	Imm/ Absolute	DF	3		Data → M(Adr)	-	-	-	-

Instruktionsformat (lägg märke till att adressen kommer *före* datan!):

DF	Adr	Data

HEMUPPGIFT 3.C

- Konstruera MOVE-instruktionen med hjälp av DigiFlisp:s "Instruction builder". Lägg till din implementering av instruktionen till filen ”`flisp_ins.fcs`”.
- Skapa ett testprogram enligt nästa sida (komplettera med saknad maskinkod), och för in maskinkoden som ”setMemory”-direktiv i en ny fil "`test_move.fmem`".
- Kontrollera `MOVE`-instruktionens funktion med hjälp av simulatorn i DigiFlisp. Rätta eventuella fel, och fyll därefter i tabellen med styrsignalsekvenser för `MOVE`-instruktionen som finns två sidor längre fram i detta PM.

Adress	Maskin-kod		Assemblerkod		Direktiv för att initiera minne
20	DF	L1	MOVE	#2,$10	#setMemory 20=DF
21	10				#setMemory 21=10
22	02				#setMemory 22=02
23	38	L2	DEC	$10	#setMemory 23=38
24	10				#setMemory 24=10
25	24		BEQ	L1	#setMemory 25=24
26	FA				#setMemory 26=20
27	21		BRA	L2	#setMemory 27=21
28	23				#setMemory 28=21
29					
FF	20				#setMemory FF=20

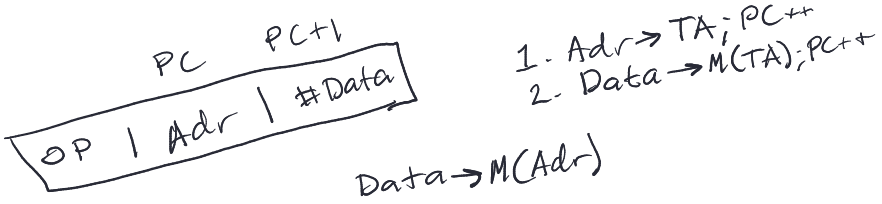
VID LABORATIONSPLATSEN

Om tiden tillåter, Du ska nu verifiera att din MOVE-instruktion fungerar även i laborationshårdvaran.

- Mata in testprogrammet i minnet i hårdvaru-FLISP.
- Koppla upp MOVE-instruktionens styrsignaler på kopplingsplattan. Gör **reset** på DV-modulen.
- Utför nu instruktionssekvensen cykelvis genom att ge klocksignaler, tills du ser operationskoden DF_{16} i instruktionsregistret.
- Kontrollera nu, för varje cykel i exekveringsfasen dvs. Q_4 och uppåt, att styrsignalerna aktiveras korrekt. Säkerställ att värdet på minnesadress 10_{16} utvecklas enligt 02_{16}, 01_{16}, 00_{16} för att sedan börja om på 02_{16}.

Då MOVE-instruktionen fungerar som den ska tillkallar du en handledare och redovisar laborationsuppgiften.

Därefter kopplar du ner denna instruktion och fortsätter med nästa uppgift.



MOVE #Data,Adr

Tillstånd	Summa-term	RTN-beskrivning	Styrsignaler =1
Q4		$M(PC) \rightarrow TA$; $PC+1 \rightarrow PC$,	LD_{TA} ; INC_{PC} ; MR,
Q5		$M(PC) \rightarrow R$; $PC+1 \rightarrow PC$,	LD_R ; OE_{PC} ; INC_{PC} ; MR,
Q6		$R \rightarrow M(TA)$	OE_R ; g_{14} ; MW,

Denna uppgift ger exempel på en mer komplex instruktion än den föregående. Jämförelse, test och villkorlig flödesändring kan utföras med en enda instruktion:

```
CMJEQ       #Data,Adress
```

Liknande funktion fås med instruktionsföljden

```
CMPA  #Data

BEQ   Adress
```

med skillnaden att CMJEQ implementerar ett absolut hopp istället för som i BEQ där hoppet är relativt. I vår nya instruktion anger vi alltså destinationsadressen som en absolut adress, vilket förenklar implementeringen av styrsignalsekvensen något.

Instruktionen specificeras av följande:

CMJEQ Compare register A with data, jump if equal

RTN	A – Data, If Z = 1: Adr → PC	
Flaggor	N:	Får värdet hos skillnadens teckenbit (bit 7).
	Z:	Ettställs om skillnaden blir noll.
	V:	Ettställs om 2-komplementoverflow uppstår vid subtraktionen
	C:	Ettställs om borrow uppstår vid subtraktionen.
Beskrivning	Data subtraheras från innehållet i register A. Skillnaden lagras ej, utan påverkar endast flaggorna. Därefter testas Z-flaggans värde. Om Z=1 utförs ett hopp till adressen Adr. Om Z=0 utförs inget hopp. Nästa instruktion blir i så fall den direkt efter CMJEQ-instruktionen i minnet.	

Detaljer:

Instruktion **CMJEQ** Variant	Adressering metod	OP	#	~	Operation	Flaggor N	Z	V	C
`CMJEQ #Data,Adr`	Immediate/Absolute	EF	3		A – Data, If (Z = 1) Adr → PC	Δ	Δ	Δ	Δ

Instruktionsformat (lägg märke till att addressen kommer före datan!):

EF	Adr	Data

Du får konstruera styrsignalsekvensen hur du vill; det finns inga "prestandakrav". Du kan exempelvis använda följande tips:

Gör en lösning som liknar dem för BEQ- och CMPA-instruktionerna, observera dock att `Adr` här är kodad som absolut adress, och inte relativ som i fallet med BEQ.

1. Läs in `Adr`, dvs. destinationsadress för uppfyllt villkor, placera i register R
2. Läs in `Data` till register T, jämför med register A och ladda CC-registret med flaggor från ALU:n
3. Överför adressen i R till PC (jfr BEQ) om Z=1; avsluta därefter styrsignalsekvensen.

HEMUPPGIFT 3.D

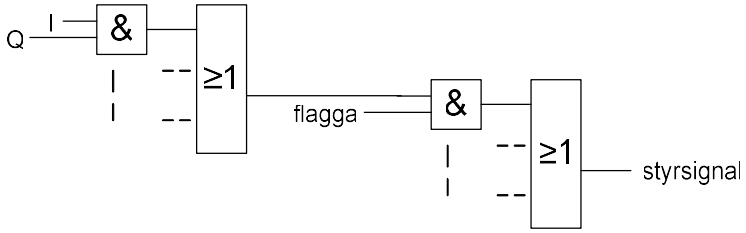
- Konstruera CMJEQ-instruktionen med hjälp av DigiFlisp:s "Instruction builder". Lägg till din implementering av instruktionen till filen ”`flisp_ins.fcs`”.
- Skapa ett testprogram enligt nedan. och för in maskinkoden som ”setMemory”-direktiv i en ny fil "`test_cmjeq.fmem`".
- Tala med en handledare om du är osäker på hur testprogrammet skall fungera.
- Kontrollera CMJEQ-instruktionens funktion med hjälp av simulatorn i DigiFlisp. Rätta eventuella fel, och fyll därefter i tabellen med styrsignalsekvenser för CMJEQ-instruktionen på nästa sida.

Adress	Maskin-kod		Assemblerkod		Direktiv för att initiera minne
20	05	L1	CLRA		#setMemory 20=05
21	07	L2	INCA		#setMemory 21=07
22	EF		CMJEQ	#2,L1	#setMemory 22=EF
23	20				#setMemory 23=20
24	02				#setMemory 24=02
25	21		BRA	L2	#setMemory 25=21
26	21				#setMemory 26=21
27					
FF	20				#setMemory FF=20

VID LABORATIONSPLATSEN

Om tiden tillåter, Verifiera att din CMJEQ-instruktion fungerar även i laborationshårdvaran:

- Mata in testprogrammet i minnet i hårdvaru-FLISP.
- Koppla upp CMJEQ-instruktionens styrsignaler på kopplingsplattan. För att skapa villkor för programflöde krävs här en AND-grind med tre ingångar, men någon sådan finns inte på laborationsplatsen, du kan i stället använda två AND/OR-nät enligt följande:



- Gör **reset** på DV-modulen.
- Utför nu instruktionssekvensen cykelvis genom att ge klocksignaler, tills du ser operationskoden EF_{16} i instruktionsregistret.
- Kontrollera nu, för varje cykel i exekveringsfasen dvs. Q_4 och uppåt, att styrsignalerna aktiveras korrekt.
- Se till att programmet hoppar till L1 och L2 vid rätt tillfälle.

Då CMJEQ-instruktionen fungerar som den ska, tillkallar du en handledare och redovisar laborationsuppgiften.

Därefter kopplar du ner och snyggar till din laborationsplats, laborationen är klar.

OP | Adr | Data

STYRSIGNALSEKVENS, HEMUPPGIFT 3.D

CMJEQ #Data,Adr

Tillstånd	Summa-term	RTN-beskrivning	Styrsignaler =1
Q4	$(Q4 \cdot I_{EF})$	$M(PC) \rightarrow R$ (M(Adr)); $PC+1 \rightarrow PC$,	LD_R; INC_{PC}; f_3; f_0; MR,
Q5	$(Q5 \cdot I_{EF})$	$M(PC) \rightarrow T$ (#Data); $PC+1 \rightarrow PC$,	MR; LD_T; INC_{PC},
Q6	$(Q6 \cdot I_{EF})$	$A-T$; $ALU(Z) \rightarrow CC$	f_3; f_2; OE_A; g_0; LD_{CC}; g_1; g_8; g_5; g_4; g_3; g_2
Q7	$(Q7 \cdot I_{EF})$	if (Z = 1) $M(R) \rightarrow PC$	$(OE_R$; $LD_{SP}) \cdot Z$

TILLÄGG TILL PM LABORATION 3

Du kan ge kommandon till DV-modulen genom att klicka på terminalfönstret och skriva in något av följande

Kommando	Betydelse
i	Interaktiv mode, alla tecken skickas tillbaka och syns i terminalfönstret
n	Utför hel instruktion (till nästa NF)
e	Utför program utan uppehåll, exekvera, avbryt exekvering genom att ge ytterligare ett ’e’-kommando.
wrZXX	Skriv värdet XX till register Z. Värdet XX anges på hexadecimal form med precis två siffror. Registret, Z, kan vara något av datavägens register enligt: a,t,x,y,s=sp,p=pc,u=ta,r,c=cc.
wmYYXX	Skriv värdet XX till minnesadress YY. Såväl värdet XX som adressen YY anges på hexadecimal form med precis två siffror.

INFÖR LABORATION 4

Följande uppgifter ur kursboken ska vara utförda som förberedelse för laborationen. Du ska på begäran av laborationshandledare redogöra för dessa.

Uppgifter i kursboken	
(9.1) (9.2) (9.13) (9.15) (9.16) 9.18	9.20

Uppgifter inom parentes är inte strikt obligatoriska, men om man gjort dem blir arbetet med de obligatoriska uppgifterna lättare.

Nedanstående hemuppgifter i detta PM som ska vara utförda innan laborationen påbörjas.

Hemuppgifter		
4.A	4.B	4.C