Laboration nr 4 behandlar:

Processorn CPU12, Mikrodatorsystemet MC12 med debgugger dbg12 och laborationskortet ML4.

Följande uppgifter ur Arbetsbok för MC12 skall vara utförda och uppvisade för en handledare innan laboration 4 utförs. Visa listfilen för uppgift 31 och 33 för din handledare.

Uppg	**1-6**	**8-10**	**23-24**	**25-26**	**29-33**
(Signatur)					

Följande hemuppgifter i laboration 4 skall vara utförda och uppvisade för en handledare innan laboration 4 utförs

Hem uppg	**1**	**2**	**3**	**4**
(Signatur)				

Följande laborationsuppgifter demonstreras för en handledare innan dessa kopplas ner.

Labb uppg	**6**	**7**	**8**	**X**
(Signatur)				

Hemuppgift 1:

Lär dig hur Eterm och simulatorn fungerar genom att läsa och jobba med de 8 första sidorna av ”Arbetsbok för MC12”. Studera DBG12 DEBUGGER, bilaga till laboration 4, som du hittar via länken ”DBG12 monitor/debugger för MC12” på resurssidan under rubriken ”Handböcker och datablad”.

Hemuppgift 2:

Studera följande beskrivning av laborationsmiljön. Se figur på nästa sida.

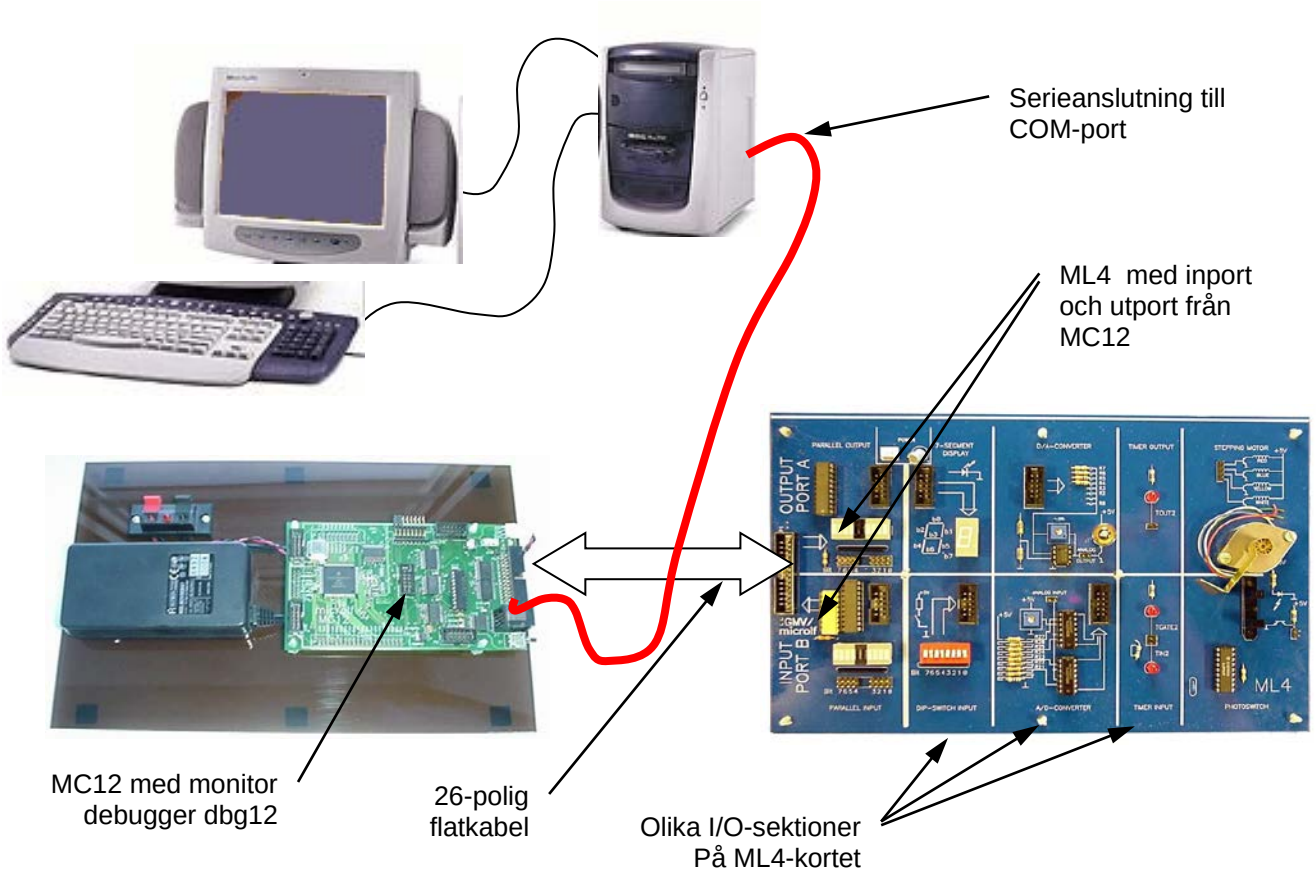
Följande enheter ingår i laborationsmiljön: En persondator (PC) med utvecklingsprogramvaran Eterm för MC12, laborationssystemet MC12 med monitor / debugger dbg12 och laborationskortet ML4.

Du har använt Eterm för FLEX tidigare. Eterm för MC12 fungerar på samma sätt så att du kan editera, assemblera och testa dina program i simulatorn för den nya datorn ”MC12”. Utöver detta kan du ansluta laborationssystemet MC12 till PC:n (till Eterm).

Laborationssystemet består av ett mikrodatorkort, ett nätaggregat och en klämkontakt för 5V’s strömförsörjning till laborationskortet ML4.

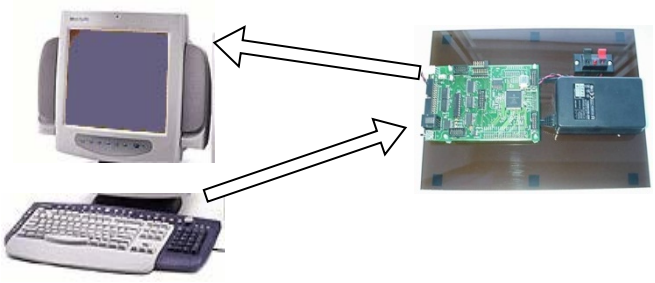
För att ladda ner och starta program i MC12 krävs det några enkla programrutiner i MC12 för att ta emot maskinprogrammet från utvecklingsmiljön Eterm och placera detta på önskad adress i minnet på MC12. Dessa rutiner kallas monitor debugger dbg12 och finns permanent lagrade (i PROM) i MC12.

Laborationskortet ML4 ansluts till MC12's inport och utport med en 26-polig flatkabel. Kortet har en sektion för IN och en sektion för UT. Vidare kan ett antal olika I/O-sektioner på kortet anslutas till IN och UT med 10-poliga flatkablar.



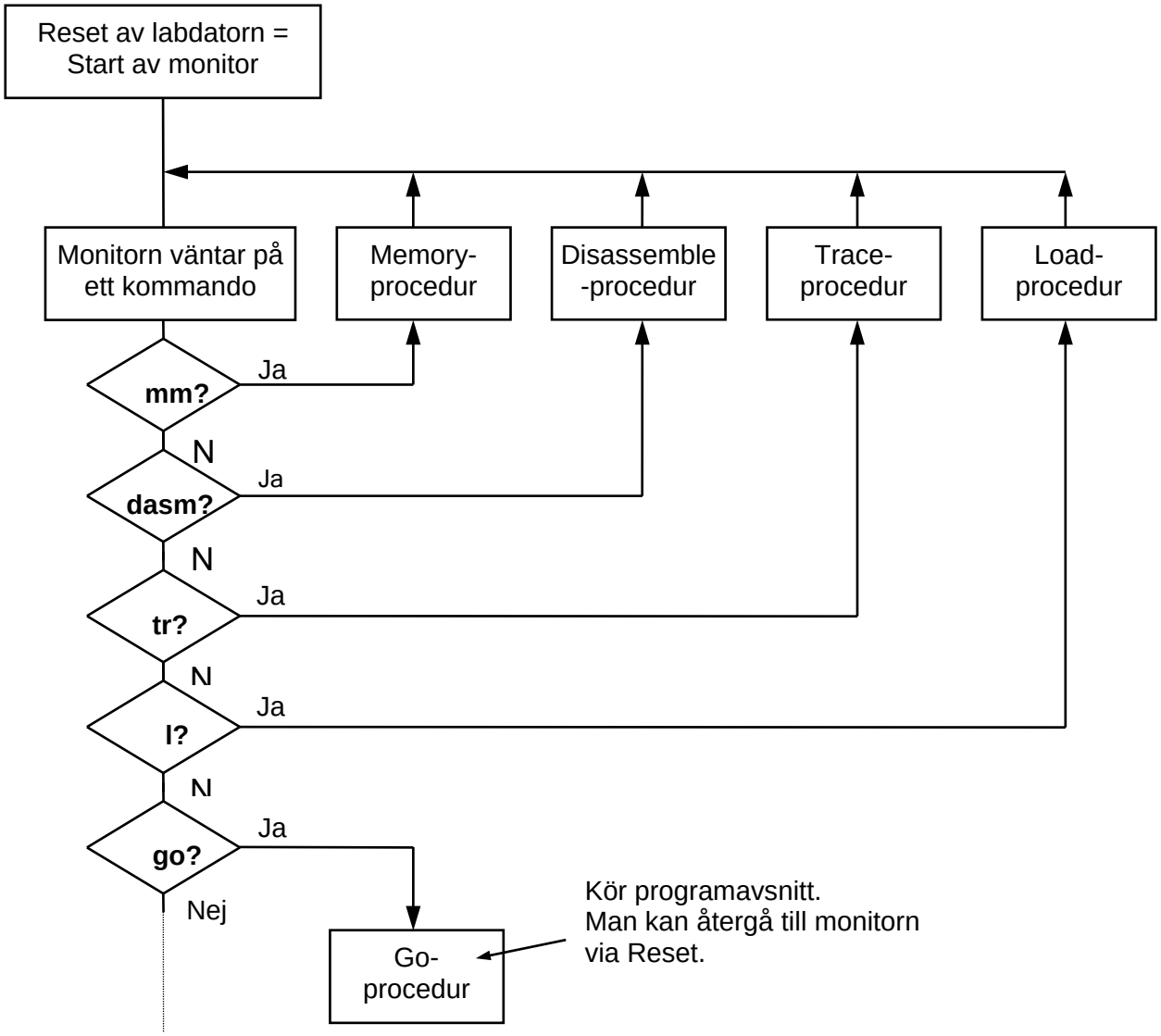
När vi nu ansluter labsystemet MC12 till en COM-port på PC:n och startar ett terminalfönster (Fliken Debug/Terminal) fungerar PC:n som en ”dum terminal” bestående av tangentbord och skärm. (Eterm = Emulera terminal = *härma terminal*).

Allt som nu skrivs på tangentbordet skickas direkt ner till labsystemet. Om monitor:n dbg12 är startad i MC12 (sker vid reset av MC12) kommer denna att ”eka” alla tecken från tangentbordet och skicka dessa direkt till skärmen. Vidare, när vi trycker på ”Enter” (krävs inte alltid) avkodas tecknen och monitorn startar upp tillhörande rutin. Rutinen avslutas oftast med att skriva ut någon text på skärmen.



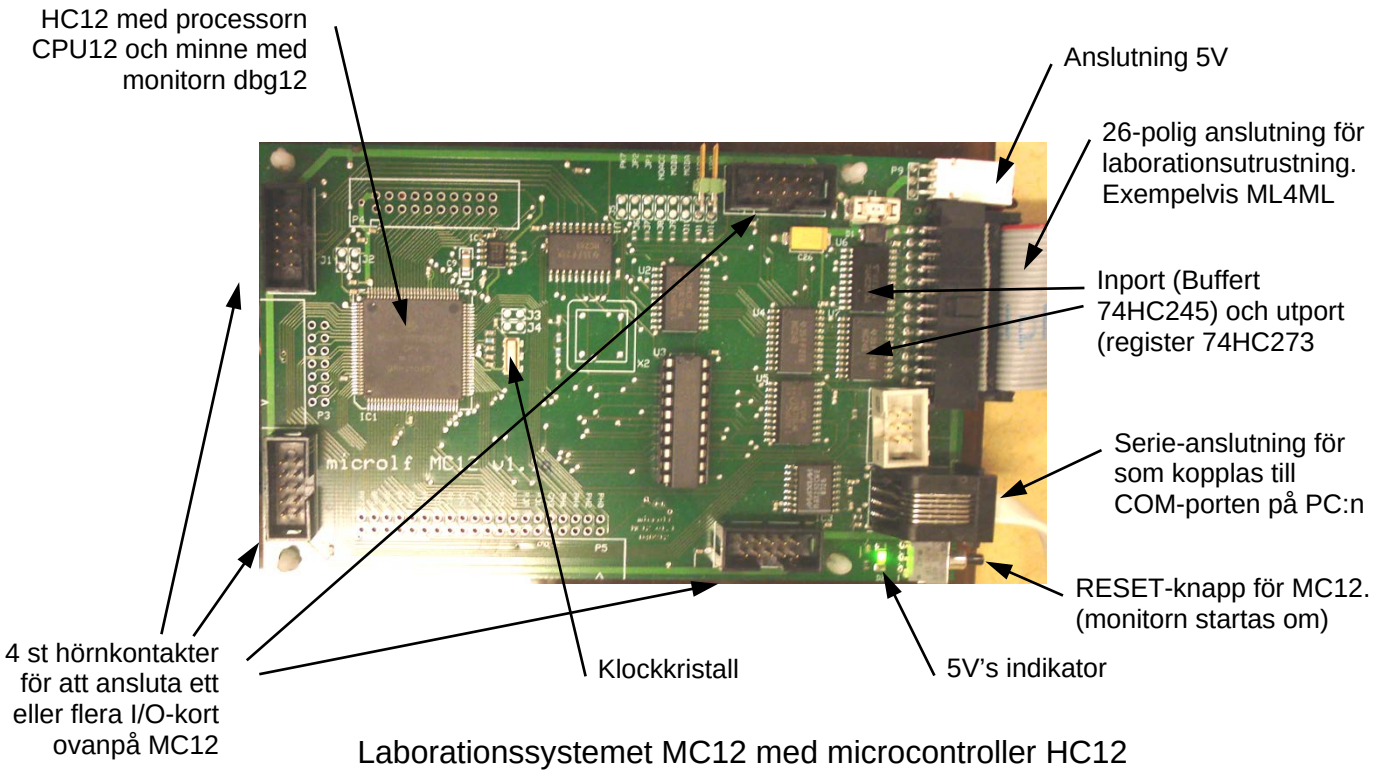
Monitorn dbg12 ”lyssnar” hela tiden på kommandon från tangentbordet bortsett från när monitorn är i läge ”go” dvs MC12 kör ”ditt” program. För att återgå till monitorn i detta fall krävs reset av MC12 för att återstarta monitorn.

Att undersöka minnesinnehåll och att disassemblera minnesinnehåll är exempel på programrutiner som finns i dbg12. Studera figuren på nedan och bekanta dig med MC12’s monitor/debugger dbg12 som du hittar på resurssidan.



MC12’s minne är konfigurerat enligt följande:

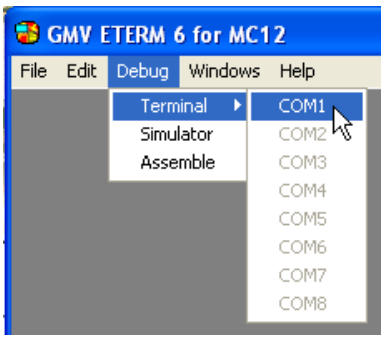
I/O	Adress $0000-$0FFF	Med plats för I/O-kort och interna (HC12-) portar
RWM	Adress $1000-$3BFF	Med plats för dina program och data
RWM	Adress $3C00-$3FFF	Variabelarea för Monitorn dbg12
ROM	Adress $4000-$FFFF	Med plats för bland annat Monitorn dbg12



Laborationssystemet MC12 med microcontroller HC12

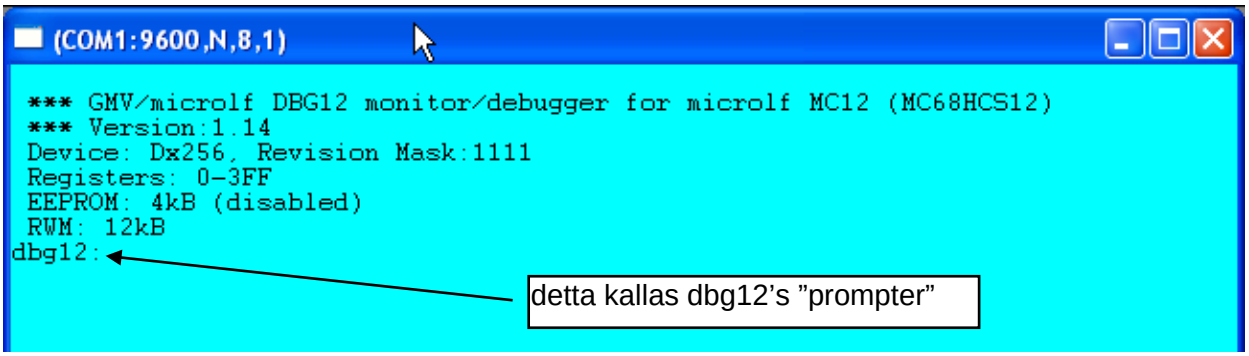
Laborationsuppgift 1:

Studera MC12 och ML4 som skall vara uppkopplade vid på din laborationsplats. Starta Eterm och anslut en terminal (COM1) under fliken Debug.



Tryck på RESET på laborationssystemet MC12 och observera att text skrivs till terminalfönstret. Kontakta en handledare om ingen text skrivs till skärmen eller om texten är oläslig.

Terminalfönstret bör se ut som figuren nedan visar. Du har nu genom att trycka på RESET startat monitorn som skriver ut en välkomsttext till dig.



Den sista raden kallas för monitorns prompter och innebär att monitorn är redo att ta emot kommandon (order) från dig. Tryck Enter (<Enter>) ett par gånger på tangentbordet. Det du skriver på tangentbordet skickas direkt till MC12 för att avkodas och eventuellt utföras.

Skriv `Help<Enter>`

```
dbg12:
dbg12:help
 *** GMV/microlf DBG12 monitor/debugger HELP
Type help 'command' for any of:
 tr    - Trace instruction
 go    - Run program
 reg   - Display/Modify registers
 dm    - Display memory
 mm    - Modify memory
 dasm  - Disassemble
 bp    - Breakpoints
 l     - Load
 fload- Program Banked FLASH
 feras- Erase Banked FLASH
dbg12:help mm
 *** GMV/microlf DBG12 monitor/debugger HELP
 mm    - Modify memory
 Forms:
```

Monitorn gör vad du begär och skriver ut en hjälptext. Se figuren till höger.

Undersök de olika kommandon genom att skriva exempelvis ”`help mm`” om du vill undersöka mm-kommandot närmare.

Undersök nu minnesinnehållen i de olika minnesområden vi har i systemet.

Skriv: `dm 0<Enter>` för att studera adressområdet där vi har in- och utenheter.
Skriv: `dm 1000<Enter>` för att studera adressområdet där du kan placera dina program.
Skriv: `dm e000<Enter>` för att studera adressområdet där vi har PROM (FLASH).

```
(COM1:9600,N,8,1)
00E0 00 00 00 80 00 00 00 00 C0 00 40 00 47 00 00 00  ..........@.G...
00F0 04 00 00 20 00 00 00 00 04 00 00 20 00 00 00 00  ... ........ ....
dbg12:dm 1000
1000 33 89 A1 CD 47 99 3F 87 D9 A7 B5 96 A9 EC 05 EE  3...G.?.........
1010 05 1A C2 67 BC 9B 92 24 15 5E D2 28 E5 F5 DC 7F  ...g...$.^.(....
1020 62 A8 DD C2 F4 59 0B 75 D9 3F D6 4E 1B 12 34 CC  b....Y.u.?.N..4.
1030 30 AA 0A E8 AE BD C1 41 47 86 E5 8B 45 57 A8 67  0......AG...EW.g
1040 49 65 E1 14 C5 A4 36 DB 38 02 9D 8B E8 57 A2 72  Ie....6.8....W.r
```

Adressangivelsen (hexadecimal) anges till vänster, minnesinnehåll (hexadecimalt) i mitten och till höger tolkas minnesinnehållet som ASCII-tecken.

Försök att ändra minnesinnehåll genom att använda kommandot mm.
Skriv först: `help mm<Enter>` för att studera en hjälptext.
Skriv sedan `mm 1000<Enter>` för att studera och ha möjlighet att ändra minnesinnehållet på adress 1000 där du har RWM. Tryck Enter ett par gånger. I figuren intill finns det tydligtvis 33 på denna adress.
Tryck på + (plus) för att studera minnesinnehållet på nästa adress. Tryck på + på nytt och ändra minnesinnehållet på adress 1002 till 55 genom att skriva `55<Enter>.` Studera figuren intill. Testa även – (minus) tangenten.

```
dbg12:mm 1000
1000 33 :
1000 33 :
1000 33 :+
1001 89 :+
1002 A1 :55
1002 55 :
1002 55 :-
1001 89 :-
1000 33 :-
```

Vilka två sätt kan du använda för att avsluta mm-kommandot och återgå till monitorns prompter?

_____

_____

Försök ändra minnesinnehållet på adress 4000, 5D16 och E01F. Använd mm-kommandot som ovan. Vad händer och varför?

_____

_____

Försök nu att ändra innehållet på adress 00CA till 3. Skriv `mm ca<Enter>` och ändra till 3.

Här skriver du direkt till ett internt styrregister i MC12 som sköter om kommunikationen mellan PC:n (COM-porten) och mikrodatorsystemet. Troligen blir det skärptecken på skärmen, eller ingen text alls när du fortsätter att ge kommandon till MC12.

Ett felaktigt program kan alltså ”skriva sönder” initieringar som monitorn (dbg12) utför vid RESET. Tryck på RESET för att starta om monitorn.

Undersök RESET-vektorn. Var i minnet är denna placerad?
(Tips: Se sid 2 i instruktionslistan för CPU12): _____

Vilken startadress laddas till PC vid RESET? _____

För att starta ett program i MC12 används go-kommandot. Skriv `go` (startadressen du hittade ovan`)<Enter>.` Du bör nu observera att monitorn startas på nytt, på samma sätt som när du trycker på RESET-knappen på MC12.

Se till att laborationskortet ML4 är anslutet till MC12. Utporten är placerad på adress 400 och inporten på adress 600. Använd monitorn och mm-kommandot för att kontrollera att anslutningen till ML4 fungerar korrekt.

Skriv: `mm 400<Enter>` för att ”öppna” denna minnesadress och skriv sedan en etta till utporten. Observera att lysdioden för bit 0 tänds upp på ML4. Bra värden att skriva ut på parallellportar är att växelvis skriva ut 55 och AA (hexadecimalt). Dessa mönster består av växelvis ettor och nollor. Testa dessa värden.

```
dbg12:
dbg12:
dbg12:mm 400
0400 04 :
0400 04 :1
0400 04 :
0400 04 :55
0400 04 :aa
0400 04 :55
0400 04 :aa
0400 04 :
```

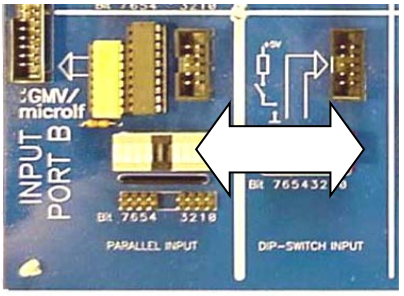
Försök förklara varför monitorn inte kan läsa det som är skrivet till utporten. (Jfr Inport och Utport på FLEX)

_____

_____

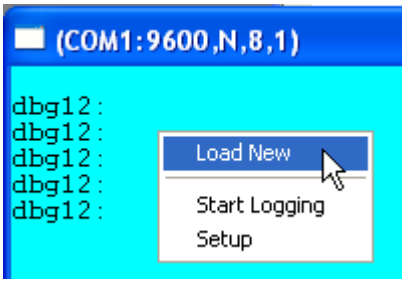
_____

Undersök nu inporten genom att skriva `mm 600<Enter>.`
Observera att strömbrytarna måste anslutas med en 10-polig flatkabel till inporten. Se figur.
Ändra på strömbrytarna och begär att monitorn skall läsa adress 600 på nytt genom att trycka på `<Enter>.` Ändra strömbrytarna på nytt och ge ännu ett `<Enter>-`kommando.



Laborationsuppgift 2:

Vi skall nu studera hur du laddar ner och kör ditt program som du redan testat i simulatorn.



Se till att monitorn är redo och har skrivit ut sin prompter. Högerklicka sedan i terminalfönstret och välj Load New för att ladda ner programmet du använde i uppgift 5 i arbetsboken. Programmet är även avbildat längst ner på sidan 9 i arbetsboken. (*Observera att programmet nu skall ha startadressen 1000*)

Eterm skickar först ett l-kommando (Load, se figuren till höger) ner till monitorn som startar sin Load-rutin. Eterm skickar därefter maskinprogrammet (Laddfilen) ner till MC12. Monitorn läser in maskinprogrammet och placerar detta på adress 1000 och framåt i minnet. När det är klart skrivs texten Load Complete ut.

```
dbg12:
dbg12:l
Loading ..
Load Complete


dbg12:dasm 1000
1000 LDAA  $0600
1003 COMA
1004 STAA  $0400
1007 JMP   $1000
100A ORB   <$E3
100C LDD   $BCD6
100F LDAA  -10,Y
1011 ANDB  $CCD5
1014 CLRA
1015 ADDA  $48B7
dbg12:
```

För att undersöka om korrekt program är placerat på rätt minnesadress kan vi undersöka detta med dasm-kommandot (dasm dissassemble) programmet. Skriv `dasm 1000<Enter>` och studera skärmutskriften.

Vilka instruktioner hittar du efter instruktionen JMP $1000? Och var kommer dessa ifrån?

_____

_____

Vilket kommando skall du ge för att enbart skriva ut dina fyra instruktioner?: (Tips: Använd help-kommandot) _____

För att köra programmet används kommandot go.
Skriv: `go 1000<Enter>`
Vad händer? Det verkar som systemet hänger sig. Varför skrivs inget till skärmen?

```
1011 ANDB  $CCD5
1014 CLRA
1015 ADDA  $48B7
dbg12:dasm 1000 4
1000 LDAA  $0600
1003 COMA
1004 STAA  $0400
1007 JMP   $1000
dbg12:go 1000
```

_____

_____

Testa att ändra strömbrytarna på ML4 och studera ändringarna på utporten. Testa lite olika värden.

Stanna ditt program! Hur gör du? _____

Ge RESET till MC12 och ändra sedan innehållet på minnesadress 1008 från 10 till C0. Använd mm-kommandot. Skriv `go 1000<Enter>` ett antal gånger när du ändrar strömbrytarna för varje go-kommando du ger. Vad händer? Använd dasm-kommandot för att undersöka ditt program. Vad har du ändrat?

_____

_____

Ändra tillbaka så att du hoppar till adress 1000 med JMP-instruktionen.

Vid felsökning är det lämpligt att använda trace-kommandot för att kunna stega igenom sitt program. Ställ först in värdet 81 (1000 0001) på strömbrytarna. Skriv `tr 1000<Enter>` och studera utskriften.

Den första raden anger registerinnehållen. Den andra raden anger vilken instruktion som står i tur att utföras. Adressangivelsen framför COMA-instruktionen är samma värde som finns i PC.

```
dbg12:
dbg12:tr 1000
S=3C80 PC=1003 Y=0000 X=0000 A=81 B=00 CC=D9
1003 COMA
dbg12:
```

Tydligen är LDAA-instruktionen redan utförd. Register A har redan värdet 81. Trace-kommandot fungerar så. När du skriver `tr 1000<Enter>` så utförs instruktionen på denna adress. När du nu enbart ger kommandot `tr<Enter>` fortsättningsvis utförs nästa instruktion som står i tur att utföras. Testa detta.

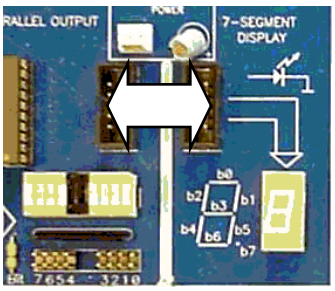
Editera om din källfil och lägg till en NOP-instruktion direkt efter ORG $1000 i ditt program. Assemblera och ladda ner till MC12 och testa trace-kommandot på nytt. Du bör se din LDAA-instruktion nu om du skriver `tr 1000<Enter>.`

Laborationsuppgift 3:

Verifiera att dina tabellvärden du angav i uppgift 8, 9 och 10 stämmer när du testar dina program i MC12 och ML4. Använd dasm-kommandot för att kontrollera att du har korrekt program du testar. Testa med trace-kommandot i uppgift 8 och go-kommandot i uppgift 9 och 10.

Laborationsuppgift 4:

Verifiera att programmet i uppgift 31 stämmer när du testar dina program i MC12 med ML4. Anslut därför en 10-polig flatkabel till sifferindikatorn enligt figuren om detta inte är gjort tidigare.



Ändra ditt program och lägg till en NOP-instruktion allra först i programmet för att få bättre skärmutskrifter vid trace. Assemblera och ladda till MC12.

Undersök om programmet ser korrekt ut i minnet och att dataarean på adress 3000 är korrekt. Starta sedan programmet med go-kommandot.

Ställ in olika värden [00,09] på strömbrytarna och kontrollera att korrekt siffra visas på sifferindikatorn.

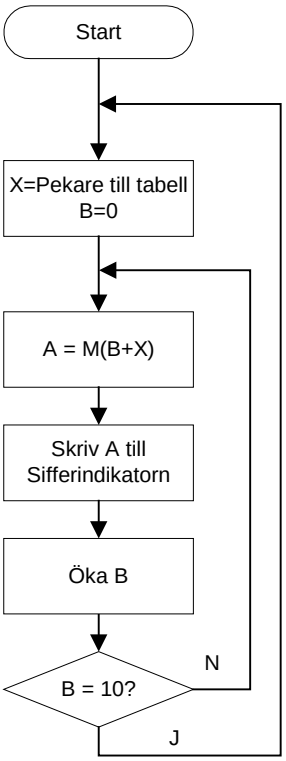
Ställ nu in några andra värden i intervallet [0A,FF] på strömbrytarna och observera vad som skrivs till sifferindikatorn. Försök förklara vad som händer och varför resultatet ser konstigt ut ibland.

_____

_____

Hemuppgift 3:

Ändra ditt program från uppgift 31 så att du kontinuerligt skriver ut siffrorna [0,9] enligt flödesplanen till höger. Testa detta i simulatorn och förvissa dig om att det fungerar korrekt. Kör Run Slow i simulatorn. Spara ditt program med filnamnet Hem3.



Laborationsuppgift 5:

Testa programmet från hemuppgift 3 i MC12 och visa siffrorna på ML4. Ladda ner programmet, verifiera att du laddat korrekt program och starta med go-kommandot.

Troligen lyser enbart en 8’a. Varför?

Gör RESET på MC12 och kör trace istället. Observera att programmet som visas inleds med en NOP-instruktion.

```
dbg12:
dbg12:tr 1000
S=3C80 PC=1001 Y=0000 X=3000 A=6B B=04 CC=D9
1001 LDX   #$3000
dbg12:tr +
dbg12:
S=3C80 PC=1004 Y=0000 X=3000 A=6B B=04 CC=D1
1004 CLRB
dbg12:
```

LDX-instruktionen står på tur att utföras. Skriv `tr +<Enter>` för att starta ”hot trace”. Använd help för tr-kommandot för att studera detta mer ingående.
Du kan nu fortsättningsvis endast trycka . (punkt) för att utföra en instruktion. Testa detta och följ förloppet i ditt program.

Du skall nu lära dig att utnyttja brytpunkter i monitorn. Ge kommandot `help bp<Enter>` för att studera en hjälptext.

Vi väljer att sätta en brytpunkt på instruktionen som ökar register B. Studera listfilen och ange vilken adress denna instruktion är placerad på. Adress: _____

Sätt brytpunkt nummer noll på denna adress genom att skriva: `bp 0 set xxxx<Enter>`. Där xxxx är adressen du hittade ovan.

```
dbg12:
dbg12:
dbg12:bp 0 set 100a
dbg12:bp
    BREAKPOINT TABLE:
    NO: ADDRESS  STATUS
    0   100A     (Active)
    1            (Not Used)
    2            (Not Used)
    3            (Not Used)
    4            (Not Used)
    5            (Not Used)
    6            (Not Used)
    7            (Not Used)
    8            (Not Used)
    9            (Not Used)
dbg12:
```

Välj sedan att ge kommandot `bp<Enter>` för att studera brytpunktstabellen. Du bör se något liknande som figuren visar. Observera att det måste finnas en operationskod på en brytpunktsadress som du anger.

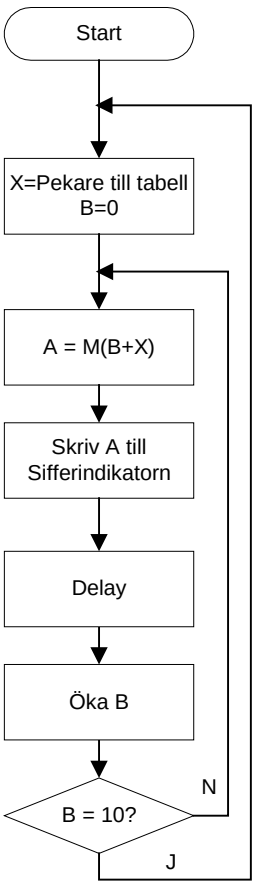
Du kan nu skriva `go 1000<Enter>` för att starta ditt program. Observera att du stannar på brytpunkten och att monitorn skriver ut sin prompter.

Vi borde nu förvänta oss att programmet har skrivit ut en nolla till sifferindikatorn. (Om vi följt flödesplanen). Kontrollera detta. Om vi nu enbart skriver `go<Enter>` borde vi förvänta oss att vi återstartar programmet från brytpunktsadressen och skriver ut nästa siffra till sifferindikatorn. Testa detta. Ge ett antal go-kommandon.

Monitorn försöker att köra instruktionen på din brytpunktsadress, men då detta ***är*** en brytpunkt stannas programmet. För att stanna på brytpunkten nästa gång den krävs det att du först ger ett tr-kommando för att stega en instruktion. Sedan kan du ge ett go-kommando. Vill du nästa gång du passerar brytpunkten ignorera denna, skriver du endast go-kommando. Testa genom att växelvis ge go- och tr-kommandon och observera utskriften till sifferindikatorn.

Hemuppgift 4:

Ändra ditt program från hemuppgift 3 så att du kontinuerligt skriver ut siffrorna [0,9] med den ändringen att du anropar en fördröjningsrutin varje varv i snurran. På så sätt kommer varje siffra att visas en längre stund på sifferindikatorn. Studera flödesplanen till höger. Spara ditt program med filnamnet Hem4.



Du bör använda dig av exempelvis JSR DELAY i huvudprogrammet och skriva in subrutinen längre ner. (Före eller efter din tabell spelar ingen roll). Tänk på att din DELAY-rutin inte får förstöra registerinnehåll för huvudprogrammet.

(Tips: Använd din DELAY-rutin från laboration 4 och ändra till ett 16-bitars register ifall du behöver längre fördröjning).

Testa detta i simulatorn och förvissa dig om att det fungerar korrekt. Kör Run Fast i simulatorn.

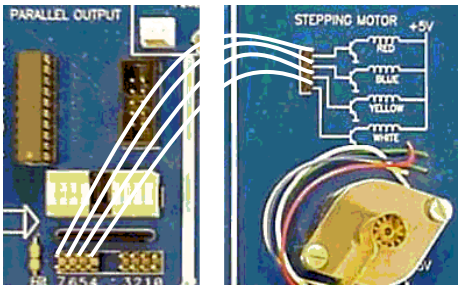
Laborationsuppgift 6:

Testa programmet från hemuppgift 4 i MC12 och visa siffrorna på ML4. Ladda ner programmet, verifiera att du laddat korrekt program och starta med go-kommandot.

Ändra nu din delayrutin så att varje siffra lyser ungefär i 200 ms. På så sätt tar det ca 2s att visa alla siffrorna [0,9]. Spara denna Delay-rutin till senare.

Laborationsuppgift 7:

ML4 innehåller en stegmotor som du skall styra. Kontrollera eller om nödvändigt koppla anslutningarna mellan utporten och stegmotorn enligt anvisningarna.



Anslutning av faser:	
RÖD	bit 7
BLÅ	bit 6
GUL	bit 5
VIT	bit 4

Stegmotorns axel fås att rotera genom att de olika faserna (RED, BLUE, YELLOW och WHITE) styrs ut. Betrakta figuren ovan. Observera att dessa faser är anslutna till +5V. För att få stegmotorn att rotera skall 0V kopplas till två av faserna medan de två andra faserna skall kopplas till +5V.

Stegmotorn har fyra olika tillstånd, se tabellen nedan. Genom att i sekvens skriva dessa tillstånd till stegmotorn kommer denna att rotera.

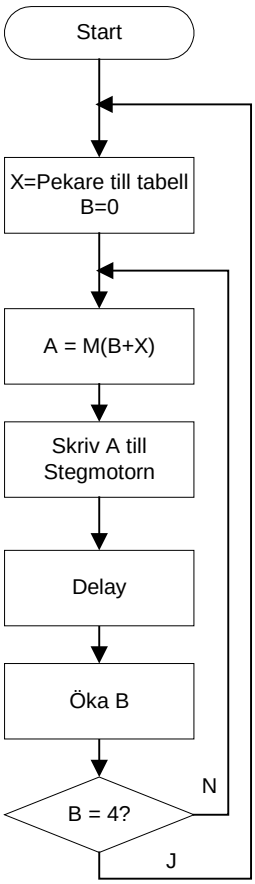
	b7	b6	b5	b4	HEX
Tillstånd 1	1	0	0	1	
Tillstånd 2	1	0	1	0	
Tillstånd 3	0	1	1	0	
Tillstånd 4	0	1	0	1	

Använd mm-kommandot och skriv ut tillstånd 1,2,3,4,1,2,3…. med monitorns hjälp. Observera att stegmotorn roterar. Försök att skriva tillstånden i motsatt följd 4,3,2,1,4,3…

Skriv nu ett program som får motorn att rotera med en jämn hastighet enligt flödesplanen till höger.

Utgå från programmet i laborationsuppgift 6. Spara som ny fil (Lab7) och ändra i denna fil.

Editera klart, assemblera ladda ner och testa ditt program för stegmotorn.



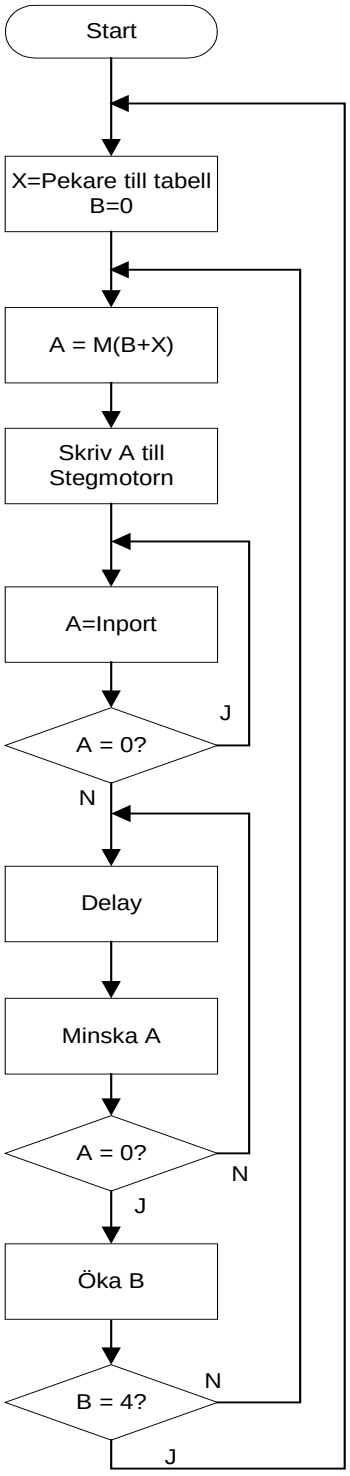
Laborationsuppgift 8:

För att kunna köra stegmotorn med variabel rotationshastighet ändrar du programmet du använde i laborationsuppgift 7. Spara med nytt filnamn Lab8.

Det förra programmet är utökat med en läsning av strömbrytarna anslutna till inporten. Läses 3 från strömbrytarna skall delay-rutinen utföras tre gånger. Se flödesplanen.

Att fortlöpande skriva olika tillstånd till stegmotorn utan en nödvändig fördröjning efter varje nytt tillstånd är meningslöst. Detta upplevs som om stegmotorn står still då den inte hinner reagera innan nästa stillstånd skrivs.

Välj därför att göra en kontroll på att du anropar Delay med ett fördröjningsvärde ≠0



Nu e du hemma! Snyggt jobbat!