



ETERM8 och MD407 inledande övning 1

Under denna övning kan du lära dig att använda utvecklingssystemet *ETERM8* tillsammans med simulatören *SimServer* som ger dig en virtuell miljö för bland andra måldatorn *MD407*.

Vi behandlar den grundläggande programutvecklingsprocessen dvs. hur man redigerar en källtext i assemblerspråk, därefter översätter källtexten till ett maskinprogram, och slutligen hur programmet testas och undersöks i simulatören.

Anmärkningar:

- Instruktioner i detta häfte förutsätter att du arbetar med en fungerande installation av *ETERM8* och *SimServer*.
- *ETERM8* har utformats för användning med såväl verktygskedjan *binutils* (GNU-projektet), som GMV's assemblerer. För *MD407* finns dock ingen GMV-assembler och här beskrivs därför bara användningen av *binutils* för ARM. Det förutsätts att du sedan tidigare är bekant med något assemblerspråk och har tillgång till lämpliga läromedel, instruktionslista etc. för ARM (v6).

INNEHÅLL

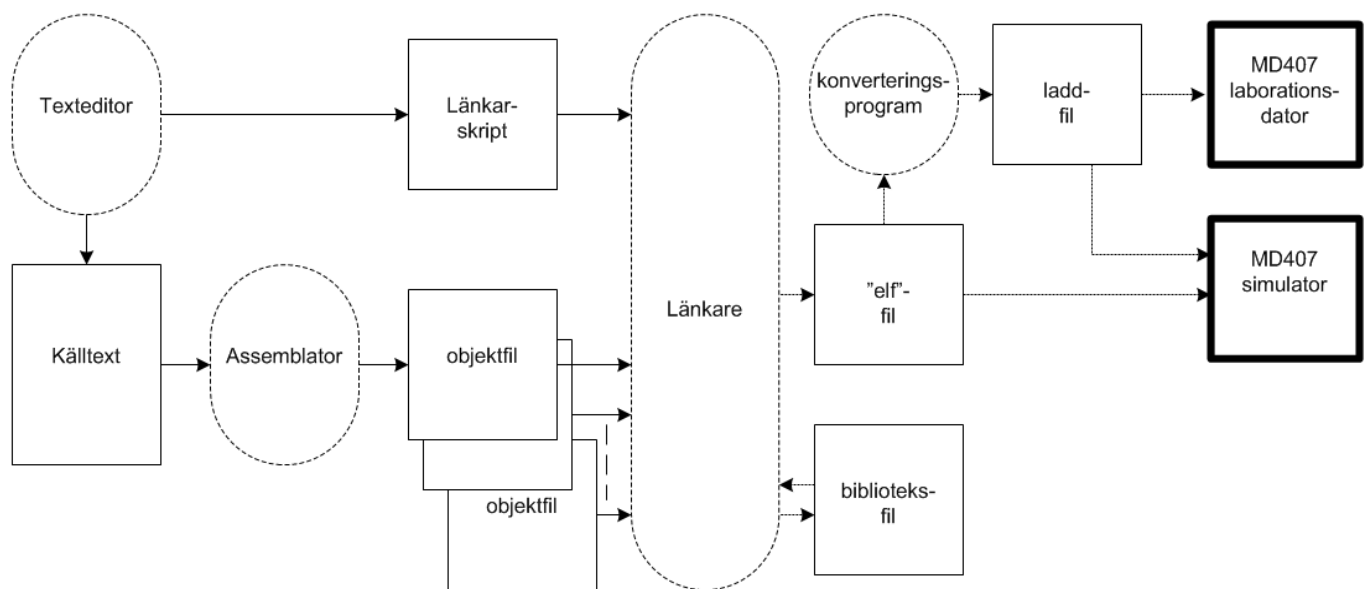
ETERM8 och programutvecklingen	2
Skapa ett assemblerprogram	5
Assemblering	6
Simulatorfunktioner	7
Processorns användarregister	7
Simulering av programexekvering.....	8
Brytpunkter	11
Återstart av program	12
Simulatorns hantering av minnet	13

ETERM8 och programutvecklingen

ETERM8 är avsett för programutveckling i assemblerspråk och har anpassats för undervisningsändamål. *ETERM8* omfattar funktioner för:

- *Textredigering*, källtexten skrivs/redigeras med hjälp av en *Editor*, färgad syntax används för att hjälpa dig upptäcka enklare stavningsfel.
- *Assembling*, källtexten översätts till en laddfil som innehåller programmets maskinkod och data.
- *Test*, laddfilen överförs till den inbyggda *simulatore*n eller till laborationsdatorn *MD407* via *ETERM8*:s terminalfunktion.

Programutveckling i assembler skiljer sig inte särskilt mycket från programutveckling i något högnivåspråk. Programmet skrivs i form av källtext, dvs. en textfil som innehåller instruktioner och direktiv till assemblern. Då programmet, eller en lämplig del av det, är färdigt måste det översättas till maskinkod innan programmet kan testas i en måldator eller simulator. Översättningen av programmet sker i flera steg med hjälp av olika verktyg. Processen illustreras i figur 1.



FIGUR 1: ÖVERSÄTTNING AV ASSEMBLERPROGRAM TILL EXEKVERBAR KOD

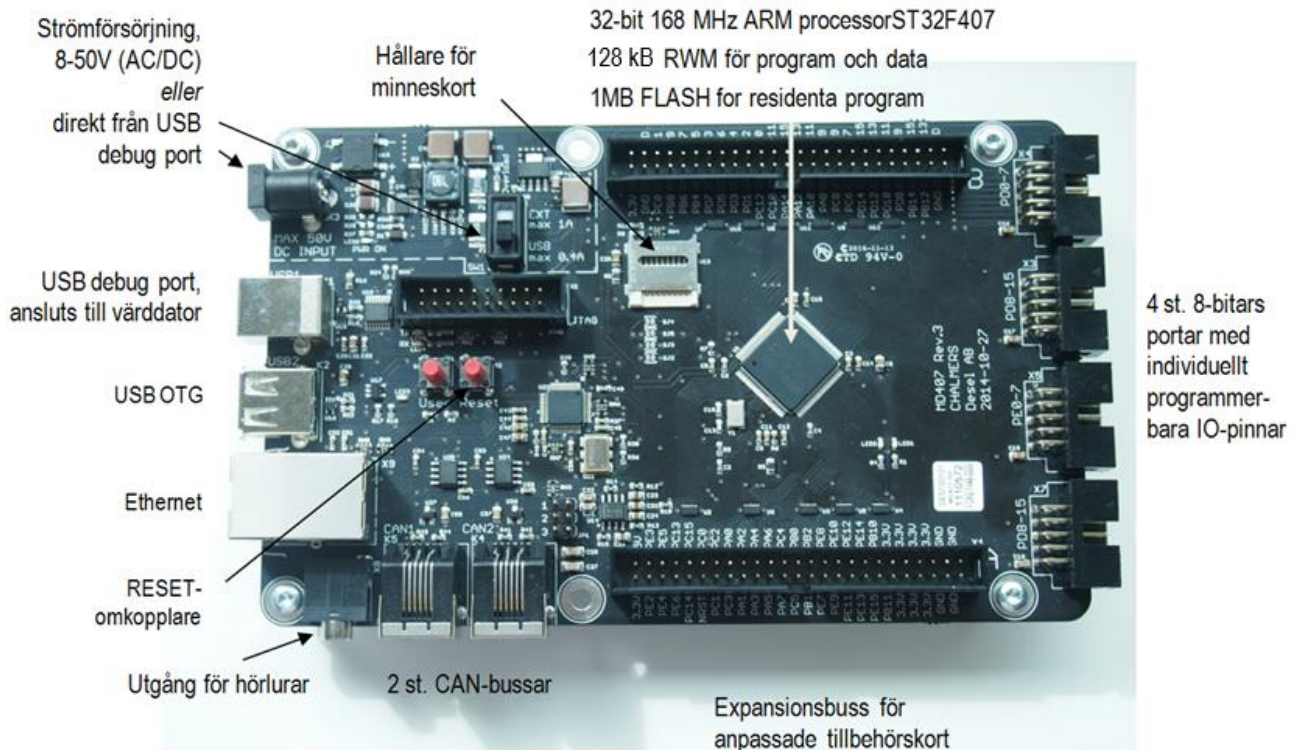
Det första steget kallas assembling (ung. "sätta samman") och utförs av assemblern. Assemblern översätter programmets källtext till objektкод, till skillnad från källtextens textformat är objektkoden i binärformat.

En, eller eventuellt flera objektfiler, kombineras samman av länkaren till en exekverbar fil med *elf-format* (*executable linkable format*). Man kan också välja att låta länkaren skapa bibliotek med program som senare kan kombineras på nytt med ytterligare objektкод.

För att skapa en exekverbar fil krävs också ett så kallat *länkar-skript*, en textfil där man bland annat samlat information om var maskinkoden ska placeras i måldatorns minne, var i minnet utrymme reserverats för variabler etc.

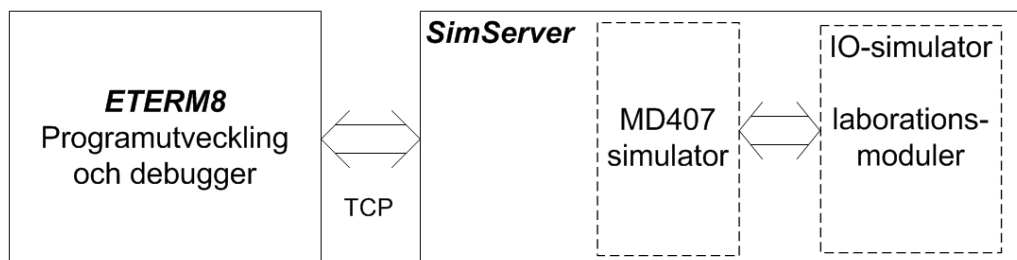
För att kunna överföra en exekverbar fil till laborationsdatorn måste den konverteras till en så kallad *laddfil*, med ett textbaserat format (*Motorola S-format*) detta görs med ett konverteringsprogram. I laddfilen finns programmet representerat på en form som kan överföras till laborationsdatorn och där tolkas som instruktioner och data.

Då programmet, i form av laddfil, överförs till laborationsdatorn *MD407*, se figur 2 nedan, kan det exekveras (utföras) och man kan då kontrollera programmets funktion men man kan också använda *ETERM8*:s simulator för test av program.



FIGUR 2: LABORATIONSATOR MD407

Funktionerna för utveckling av assemblerprogram har integrerats i programmet *ETERM8*, medan simuleringsfunktioner utförs i ett annat program: *SimServer*. *ETERM8* och *SimServer* kommunicerar via *Transmission Control Protocol (TCP)*, se figur 3



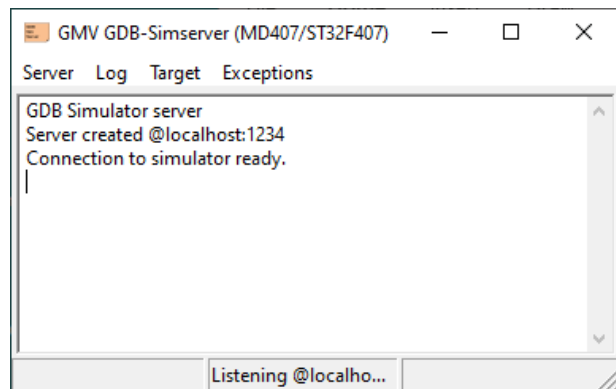
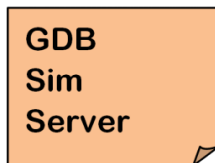
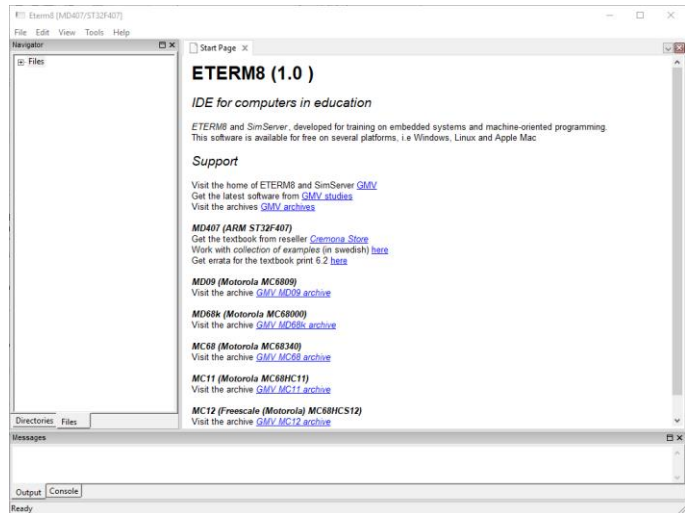
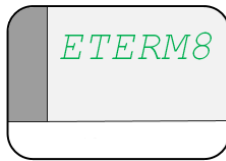
FIGUR 3: ETERM8 OCH SIMSERVER

SimServer motsvarar den fysiska laborationsdatorn (bland andra MD407) tänk dock på att det alltid finns skillnader mellan simulatorer och hårdvara som gör att likheten inte är fullständig.

ETERM8 kommunicerar alltså med *SimServer* via *TCP* där *ETERM8* är klienten medan *SimServer* utgör serverfunktionen. Kommunikationsporten kan väljas fritt och standardvärdet är port 1234. Normalt behöver inte detta ändras men det kan vara praktiskt om man vill utföra flera samtidiga instanser av *SimServer*, dessa måste då tilldelas olika portar.

I *SimServer*, finns utöver simulatorer för laborationsdatorer också simulering av laborationsenheter. Detta gör att du kan utföra uppgifter och laborationsövningar utan att ha tillgång till den fysiska hårdvaran. Syftet med detta är att du ska kunna förbereda dig grundligt innan du börjar arbeta med laborationsutrustningarna.

Om du inte redan gjort det så starta nu *ETERM8* och *SimServer*.



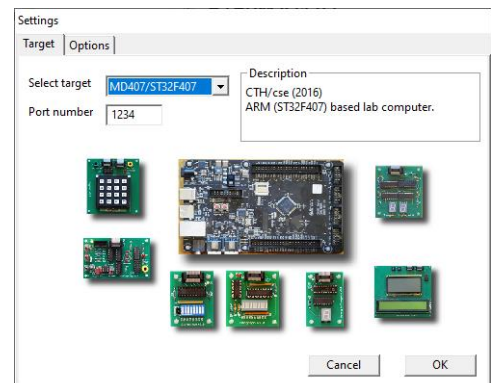
Börja med att kontrollera att *ETERM8* är inställd för rätt laborationsdator och att rätt kommunikationsport används.

Välj Tools | Settings:

I listboxen Select Target, välj MD407/ST32F407.

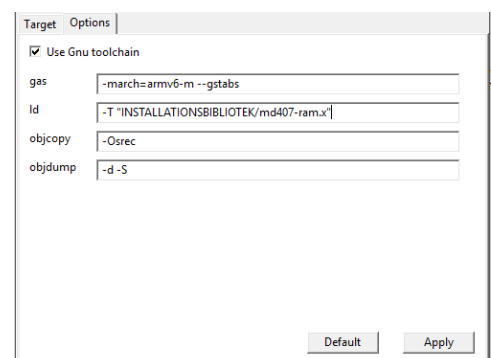
Kontrollera Port number (TCP-port) detta ska vara samma som i *SimServer*

Klicka OK för att spara eventuella ändringar, eller Cancel för att återgå utan att spara ändringar.



Under fliken Options kan du ge speciella flaggor för de använda verktygen.

Texten INSTALLATIONSBIBLIOTEK anger här den plats där *ETERM8* och *SimServer* är installerade. Den kan variera beroende på just din installaton.



Kontrollera att *SimServer* ställts in för samma kommunikationsport, det gör du med menyvalet Server | Set Target.

Tänk också fortsättningsvis på att bilderna här kommer att kunna avvika mer eller mindre beroende på vilken plattform (Windows, MacOS eller Linux) du använder. På samma sätt kan utseendet avvika beroende på det valda temat, i operativsystemet.

ETERM8 har initialt tre olika arbetsytor med olika flikar:

- Navigator - fliken Directories används för att ange ett arbetsbibliotek, under fliken Files visas alla filer i arbetsbiblioteket.
- Messages - i fliken Output visas meddelanden från assemblern, fliken Console rymmer ett terminalfönster som används för att kommunicera med laborationsdatorn MD407 via någon USB-port.
- Den tredje arbetsytan som initialt visar en startsida, används också som redigeringsfönster för textfiler.

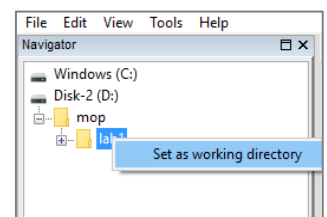


Ytterligare flikar skapas av ETERM8:s debugger, vi återkommer med detaljer efterhand som dessa introduceras.

Börja nu med att skapa ett arbetsbibliotek (här D:\moplabb1) välj själv lämplig plats och namn.

Anm. För ditt arbetsbibliotek bör du undvika sökvägar med svenska tecken eller "mellanslag" eftersom vissa programverktyg som ETERM8 använder inte hanterar dessa korrekt.

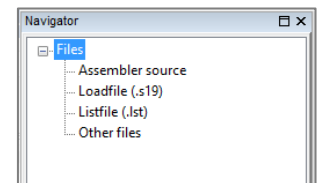
Använd Navigator-Directories för att välja arbetsbiblioteket, välj (vänsterklicka) på arbetsbibliotekets namn, högerklicka därefter, en popup med namnet Set as working directory visas, klicka på denna.



Växla därefter till fliken Files.

Under Files sorteras filerna i arbetsbiblioteket i fyra olika grupper:

- Assembler source listar filer med ändelse som är typisk för en källtext.
- Loadfile (ändelse .s19), laddfil med binär kod/data som kan överföras till laborationsdator eller simulator
- Listfile (ändelse .lst) adressinformation tillsammans med assemblerkod, kan ibland vara användbart vid test och felsökning.
- Filer som inte har någon speciell betydelse för ETERM8 sorteras in under Other files.

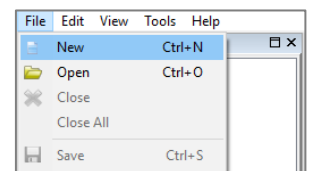


Skapa ett assemblerprogram

Du skapar en ny källtextfil genom att välja

File | New från menyn.

Därefter skriver du in namnet på den fil du vill skapa, skriv nu mom1.asm och klicka på Save. Om du inte anger något filnamnstillägg (eller anger ett annorlunda filnamnstillägg) lägger ETERM8 automatiskt till .asm. Nu skapas ett nytt fönster:



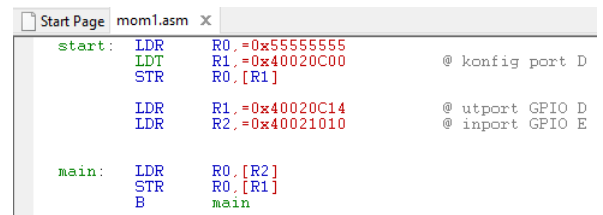
UPPGIFT: TEXTREDIGERING

Skriv nu in källtext enligt följande:

```
start:  LDR    R0,=0x55555555
        LDT    R1,=0x40020C00 @ konfig port D
        STR    R0,[R1]
        LDR    R1,=0x40020C14 @ utport GPIO D
        LDR    R2,=0x40021010 @ inport GPIO E
main:   LDR    R0,[R2]
        STR    R0,[R1]
        B      main
```

Observera hur texteditorn färglägger din text ("färgad syntax").

- Tecknet '@' används för att ange att all påföljande text är kommentarer. Kommentarer ignoreras av assemblern, tecknet kan förekomma var som helst på en textrad och kommentarer färgas grå.
- Ett giltigt symbolnamn färgas grönt. Med "giltigt symbolnamn" menas alla kombinationer av teckensträngar där de ingående tecknen är tillåtna för en symbol. Detta innebär självfallet inte att symbolen är korrekt definierad eller refererad.
- Giltiga instruktioner, namn på ARM-register och assemblerdirektiv färgas blå
- Konstanter och operatorer färgas röda. Teckenkombinationen '0x' (noll-x) anger att påföljande talvärde ska tolkas på hexadecimal form.



Notera speciellt hur instruktionen:

```
LDT R1,=0x40020C00
```

färgas grön, dvs. tolkas som en symbol. Detta beror på att vi (avsiktligt) stavat instruktionen fel, rätt instruktion ska här vara LDR. Låt felet vara kvar, vi ska strax rätta till det. För att spara filen använder du nu File | Save.

Tänk på att korrekt "färgad syntax" inte nödvändigtvis innebär att ditt assemblerprogram är korrekt, det är snarare till för att göra dig uppmärksam på enklare stavfel, syntaxfel etc. Därför kan det hända att du får felmeddelanden även om du stavat såväl instruktioner som operander riktigt.

Assemblering

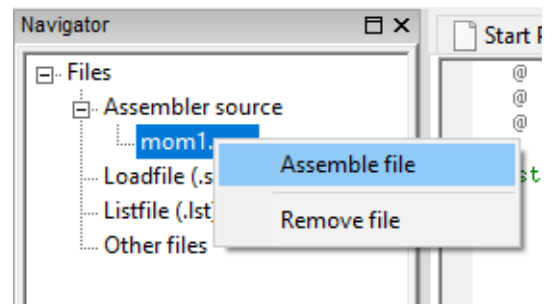
Du assemblerar källtexten genom att:

1. Växla till fliken Files i Navigator fönstret.
2. Öppna Assembler source
3. Välj (vänsterklicka) på filnamnet mom1.asm högerklicka därefter, en popup knapp visas...
4. välj Assemble File

alternativt:

Från Menyn väljer du Tools|Assemble, ett dialogfönster låter dig nu välja den fil du vill assemblera.

Utskrifter från assemblern visas i sektionen Messages under fliken Output.



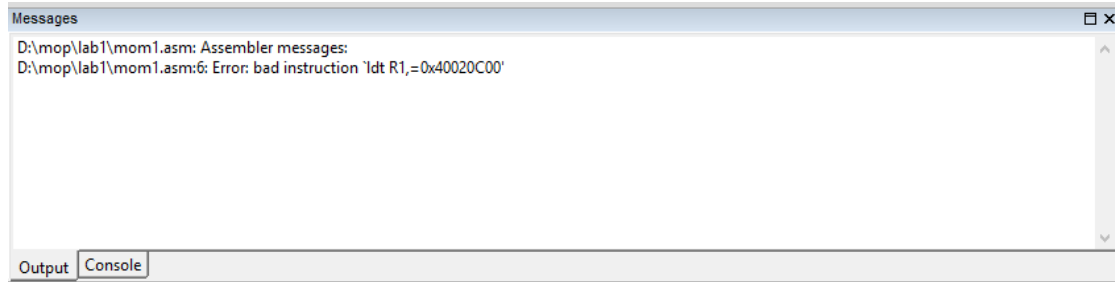
UPPGIFT: ASSEMBLERING

1. Assemblera nu filen mom1.asm

Assembleren kommer att klaga på den felstavade instruktionen. Efter filnamnet, med fullständig sökväg (som kan se annorlunda ut i ditt exempel) skrivs radnummer, därefter typ av fel. Utskriften

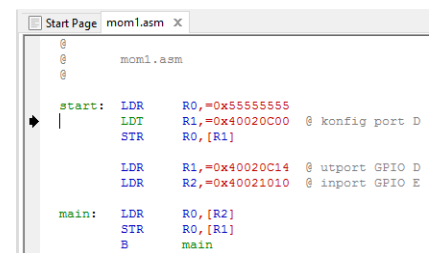
"Error: bad instruction 'ldt R1,=0x40020C00'"

berättar vad vi redan misstänkte, dvs. att instruktionens namn är felstavat.

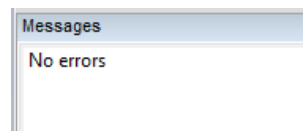


Felutskrift från assembler

2. Dubbelklicka nu (vänster knapp) på felutskriften. Markören i marginalen pekar ut raden i källtextfilen som genererat felet.



3. Rätta felet (ändra den felaktiga instruktionen till LDR) och assemblera på nytt. Meddelandet No Errors ska nu visas i Output-fliken.



Simulatorfunktioner

Med ETERM8:s debugger och MD407-simulator kan du simulera instruktionsutförandet i laborationsdatorn MD407. Du kan också övervaka och manuellt ändra såväl register som minnesinnehåll.

Processorns användarregister

Simulatorns registersektion visar registeruppsättningen hos MD407.

Registers							
R0	00000000	R8	00000000	PSR	01000000	S0	00000000
R1	00000000	R9	00000000	FPSR	00000000	S1	00000000
R2	00000000	R10	00000000	MSP	2001C000	S2	00000000
R3	00000000	R11	00000000	PSP	00000000	S3	00000000
R4	00000000	R12	00000000	PRIMASK	00000000	S4	00000000
R5	00000000	SP	2001C000	FAULTMAS	00000000	S5	00000000
R6	00000000	LR	FFFFFFFF	BASEPRI	00000000	S6	00000000
R7	00000000	PC	20000000	CONTROL	00000000	S7	00000000
						S8	00000000
						S9	00000000
						S10	00000000
						S11	00000000
						S12	00000000
						S13	00000000
						S14	00000000
						S15	00000000

FIGUR 4: SIMULATORNS REGISTERSEKTION

Du kan ändra registrens innehåll manuellt genom att klicka på fönstret för det register du vill ändra och skriva in ett nytt värde, värdet anges på hexadecimal form. Då du ändrat ett registerinnehåll måste du också trycka Enter, för att ändringarna ska registreras i simulatorn.

Simulering av programexekvering

Med debuggerns hjälp kan du få en första inblick i hur assemblerinstruktioner fungerar. Du kan utföra ett assemblerprogram instruktionsvis och i lugn och ro studera effekterna. Glöm inte att spara dina källtextfiler, många moment utgår från att återanvända kod.

Under detta moment kommer vi att introducera användningen av debuggern. Först visar vi hur du kan ladda program till simulatören och utföra programmet instruktionsvis. Du kommer också att se hur man kan koppla *kringenheter* till simulatören.

Kontrollera först att du startat *SimServer*.

STARTA DEBUGGERN

Vi ska nu prova programmet `mom1.asm`. Debuggern kan startas på olika sätt,

1. I fliken Navigator|Files, välj gruppen Loadfile(.s19)
2. Högerklicka på `mom1.s19`, en popup med tre alternativ visas:
Open file - om du vill granska laddfilen med en editor
Debug in simulator - för att starta simulatören med denna laddfil.
Remove file för att ta bort filen.

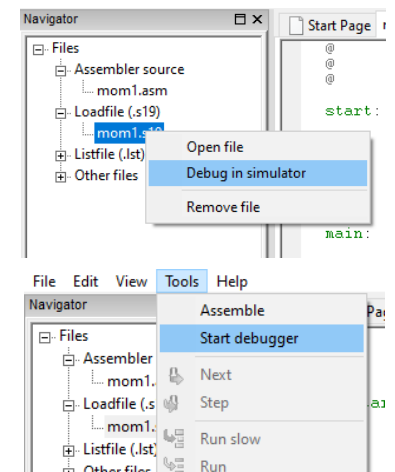
alternativt:

Välj från menyn

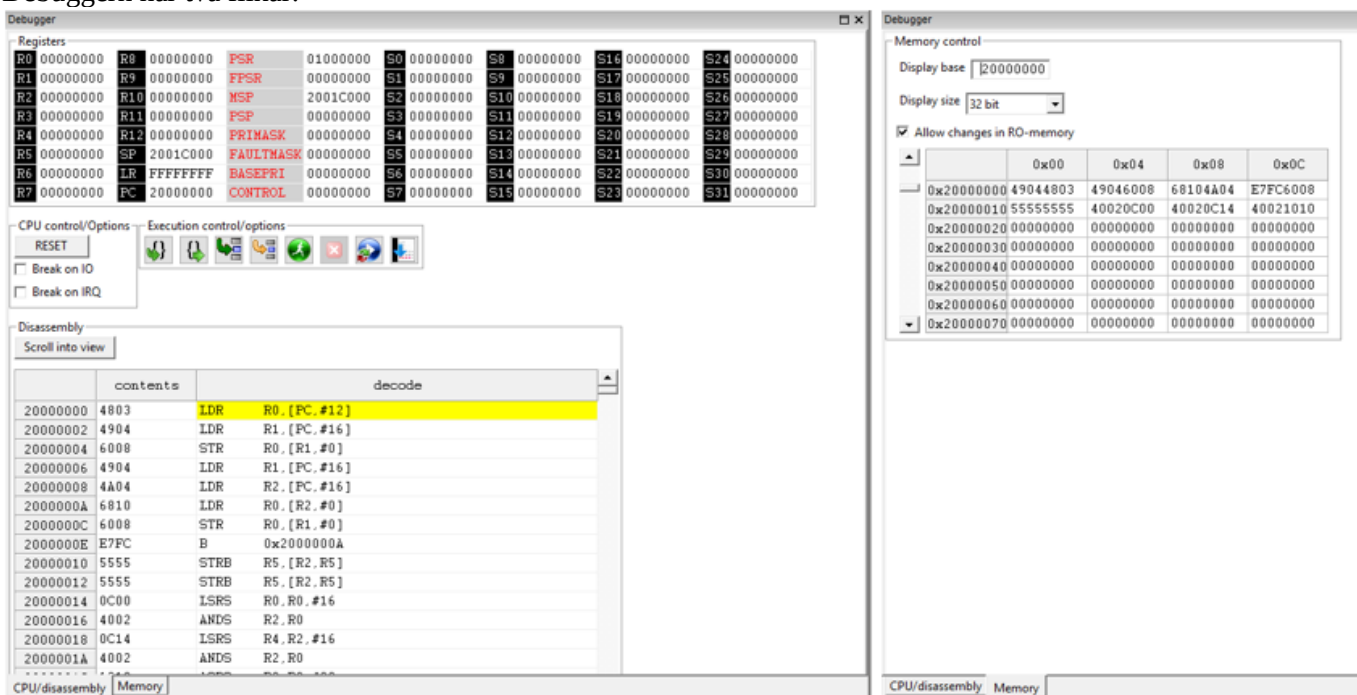
Tools | Start debugger

en dialog låter dig nu välja laddfil.

1. Välj från menyn Tools | Start debugger
2. Välj i dialogrutan: `mom1.s19`



Debuggern har två flikar:



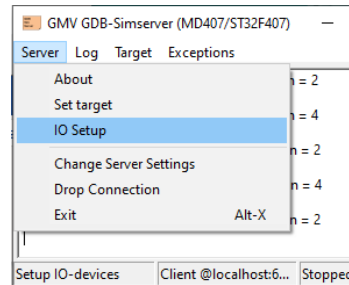
- *CPU/disassembly*: MD407-simulatörens olika ARM-register och en bild av det disassemblerade minnesinnehållet dvs. programmet och debuggers kontrollfunktioner.
- *Memory*: MD407-simulatörens minne

Eftersom programmet utför in- och ut-matning måste vi dock göra vissa förberedelser innan vi provar programmet.

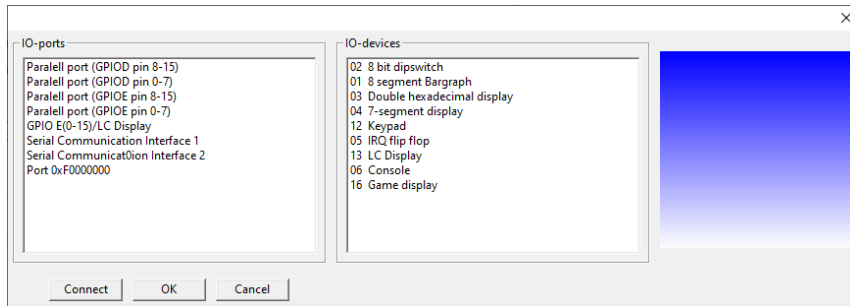
Till simulatören finns också en fristående del som kallas *IO-simulator* (Input/Output-simulator). Dess uppgift är att simulera olika omgivningar till laborationsdatoren, dvs. de enheter som inmatning sker från och utmatning sker till. Eftersom IO-simulatorn innehåller olika typer av kringenheter och dessa kan kopplas till olika anslutningar hos MD407 och vi måste göra dessa inställningar manuellt.

I SimServer,

välj Server | IO Setup:



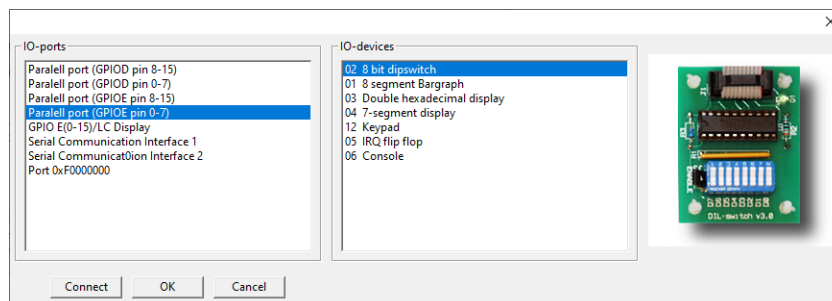
Följande dialogruta öppnas:



Till vänster, *IO-ports*, visas de anslutningar (*portar*) hos *MD407* som kan användas i simulatören. I mitten, *IO-devices*, visas de kringenheter som kan anslutas. Till höger visas en bild av den valda kringenheten.

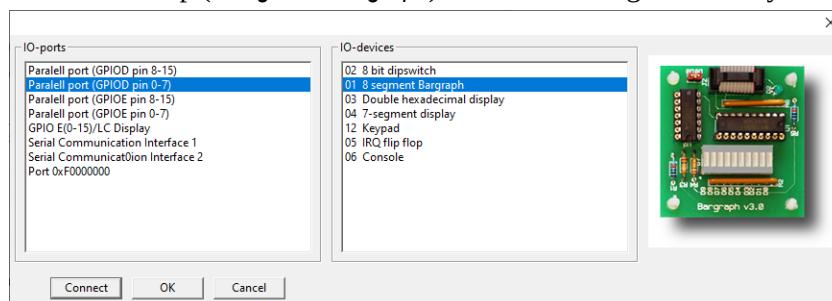
För att prova vårt första program, `mom1`, kopplar vi nu en 8 bitars strömställare (8 bit dipswitch) till den minst signifikanta byten av port E (GPIOE pin 0-7).

Märk porten och kringenheten:



Klicka därefter på **Connect**. Strömställarmodulen öppnas. Den är försedd med 8 st. enpoliga knappar som kan ställas i läge on eller off genom att man klickar på dem. Initialt står knapparna i läge off, vilket innebär att motsvarande logiknivå är 0. Genom att klicka en gång ändras läget till on, logiknivån är då 1, klicka en gång till på knappen för att få den att återgå till off, osv. I fönstret skrivs också den aktuella anslutningen ut, i detta fall PE0-7.

Fortsätt nu med att ansluta en diodramp (8 segment Bargraph) till den minst signifikanta byten hos port D, GPIOD pin 0-7.



Diodrampsmodulen öppnas, den innehåller 8 st. dioder som kopplats till porten.

Klicka slutligen **OK** för att behålla anslutningarna.

TEST AV PROGRAM

Då du anslutit enheter för såväl in- som utmatning är du redo att testa ditt program. Följande knappar används i *ETERM8* för att kontrollera programexekveringen:

	Step instruction: Utför en maskininstruktion, kallas också <i>trace</i> , exakt en maskininstruktion utförs.
	Next instruction: Utför nästa instruktion, skillnaden mot Step instruction är att en tillfällig brytpunkt placeras på nästa adress i det sekventiella programflödet. Detta innebär exempelvis att hela subrutiner kan utföras utan att man behöver stega igenom varje instruktion i subrutinen.
	Step 5 ins/sec: Stega automatiskt, 5 instruktioner per sekund
	Run, stop at breakpoint: Utför programmet snabbt, respektera brytpunkter
	Run, ignore breakpoints: Utför programmet snabbt, ignorera brytpunkter

Observera nu hur debuggern har märkt den instruktion som står i tur att utföras med en gul bakgrund.

Klicka på Step instruction för att utföra denna instruktion.

	contents	decode
20000000	4803	LDR R0, [PC, #12]
20000002	4904	LDR R1, [PC, #16]
20000004	6008	STR R0, [R1, #0]
20000006	4904	LDR R1, [PC, #16]
20000008	4A04	LDR R2, [PC, #16]
2000000A	6810	LDR R0, [R2, #0]
2000000C	6008	STR R0, [R1, #0]
2000000E	E7FC	B 0x2000000A

Instruktionen utförs, nästa instruktion märks med gul bakgrund. Notera effekten av instruktionen, register R0 har fått nytt värde.

Fortsätt stega tills instruktionen på adress 2000000A märks med gul bakgrund. Notera hur värdet i processorns register R1 och R2 ändrats i registersektionen. Adressen till GPIOD:s indataregister finns nu i R2, adressen till GPIOE:s utdataregister finns i R1.

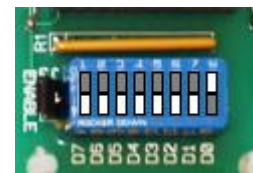
Registers			
R0	55555555	R8	
R1	00000000	R9	
R2	00000000	R10	
R3	00000000	R11	

Ställ nu in värdet 1 på strömställarmodulen genom att klicka på den minst signifikanta biten.

Utför nu även nästa instruktion för att läsa ett värde från inporten:

```
LDR R0, [R2, #0]
```

Observera hur register R0 laddas med strömställarmodulens värde

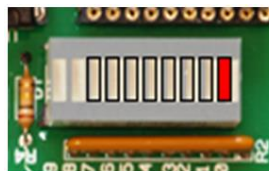


Registers			
R0	00000001	R8	
R1	40020C14	R9	
R2	40021010	R10	

Fortsätt nu och utför nästa instruktion:

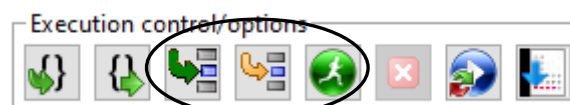
```
STR R0, [R1, #0]
```

Värdet i register R0 skrivs till ljusdiodrampen.



Röda indikatorer tolkar du som '1', övriga tolkas som '0'. Endast 8 av de 10 indikatorerna används.

Då du testat ett programs funktion genom att utföra det instruktionsvis kan du också utföra det med något Run-kommandot.



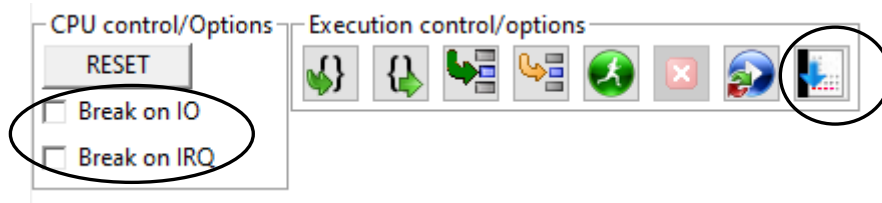
Här finns det tre olika varianter, det första alternativet tillåter dig följa programexekveringen i detalj eftersom debuggern uppdaterar vyn mellan varje instruktion. Det båda andra alternativen används för att utföra programmet så snabbt som möjligt, endast programräknaren i registerfönstret uppdateras, det första alternativet, med en gul böjd pil, kontrollerar om det finns *brytpunkter* i programmet, det andra alternativet ignorerar brytpunkter.

Brytpunkter

En *brytpunkt* är ett ställe där programexekveringen avbryts. Brytpunkter är bara meningsfulla då du använder Run, stop at breakpoint. Brytpunkter kan hanteras direkt i programfönstret eller i *brytpunktstabellen* via debuggerns verktygslist. Brytpunktstabellen har plats för upp till 20 samtidiga brytpunkter i programmet.

Brytpunkten kan vara *permanent* och läggs då in i brytpunktstabellen. Varje gång programmet ska utföra instruktionen på denna adress kommer simulatoren att avbryta exekveringen. Brytpunkten kan också vara *tillfällig*, programmet exekveras tills denna adress uppträder *nästa gång*. Brytpunkten tas därefter automatiskt bort av simulatoren.

Slutligen finns två speciella brytpunktsfunktioner: Break on IO, då denna aktiverats stoppas programexekveringen då programmet gör någon in- eller utmatning, respektive Break on IRQ, vilken gör att programexekveringen stoppas vid upptakten av avbrottshantering.



Brytpunkter kan sättas ut på olika sätt;

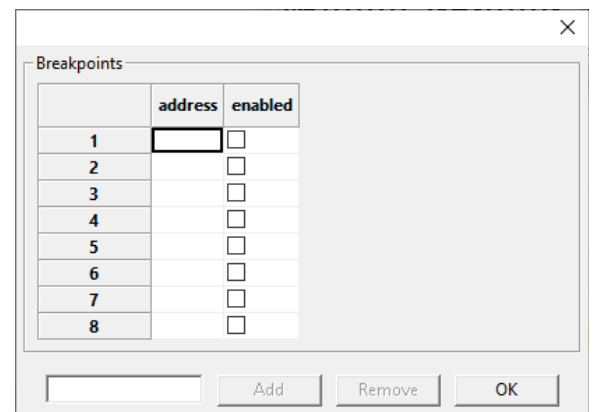
- Om programfönstret visar den adress där du vill sätta brytpunkten: identifiera adressen i programfönstret och klicka på dess rad. En *popup* meny visas då.
 - Go to.. sätter en tillfällig brytpunkt och startar programexekveringen
 - SetBreakpoint sätter en permanent brytpunkt här.

	contents		decode
20000000	4803	LDR	R0, [PC, #12]
20000002	4904	LDR	R1, [PC, #16]
20000004	6008	STR	R0, [R1, #0]
20000006	4904	LDR	R1, [PC, #16]
20000008	4A04	LDR	R2, [PC, #16]
2000000A	6810	LDR	R0, [R2, #0]
2000000C	6008	STR	R0, [R1, #0]
2000000E	E7FC	B	0x2000000A
20000010	FFFF	STPB	R0, [R1, #0]

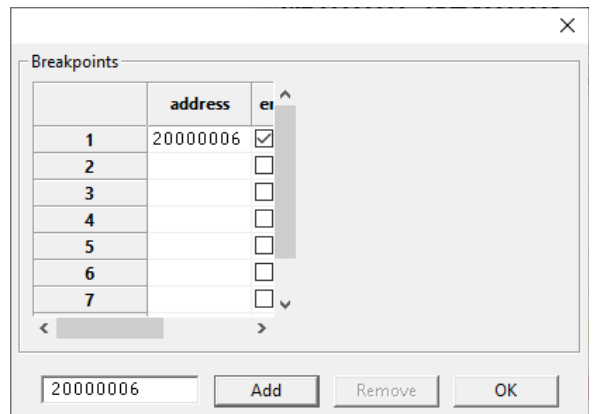
Go to...
SetBreakpoint

För att stoppa programmet på adresser som inte syns i programfönstret kan du identifiera dessa exempelvis genom att granska listfilen från ditt program.

- För att hantera brytpunkterna kan du öppna dialogen för brytpunktstabellen:



- För att lägga till en ny brytpunkt skriver du in adressen i fönstret längst ned till vänster, exempelvis adress 20000006, och klickar på Add



- Brytpunkter kan vara *aktiva* (enabled) eller *inaktiva*. Programmet stoppas inte vid en inaktiv brytpunkt men det kan vara en praktisk funktion då man tillfälligt vill undvika brytpunkten men samtidigt behålla den i brytpunktstabellen.
- Du kan tillfälligt inaktivera en brytpunkt genom att märka av **enabled**-boxen.

	address	enabled
1	20000006	☒
2		☐
3		☐
4		☐

	address	enabled
1	20000006	☐
2		☐
3		☐
4		☐

I programfönstret märks raderna enligt:

- Om den instruktion som står i tur att utföras har en brytpunkt visas denna med gul text mot svart bakgrund.
- En aktiv brytpunkt illustreras av röd bakgrund.
- En inaktiv brytpunkt illustreras av blå bakgrund.
- För att ta bort en brytpunkt klickar du först på dess nummer i brytpunktstabellen och därefter Remove.

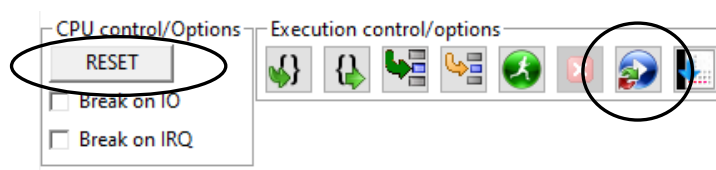
20000000	4803	LDR	R0, [PC, #12]
20000002	4904	LDR	R1, [PC, #16]
20000004	6008	STR	R0, [R1, #0]
20000006	4904	LDR	R1, [PC, #16]
20000008	4A04	LDR	R2, [PC, #16]
2000000A	6810	LDR	R0, [R2, #0]
2000000C	6008	STR	R0, [R1, #0]
2000000E	E7FC	B	0x2000000A
20000010	FFFF	STOP	R0, R0, R0

	address	enabled
1	20000006	☒
2		☐
3		☐
4		☐
5		☐
6		☐
7		☐
8		☐

Återstart av program

Programmet kan förberedas för start på två olika sätt

- Restart, här förbereds programmet på samma sätt som om det hade laddats till MD407, dvs. hårdvaran (portar och klockor) har initierats och är klar att användas.
- RESET, här genomför simulatoren samma procedur som vid reset av MD407, dvs. portar och klockor försätts i initialtillstånd, programräknare och stackpekare initieras från minnet, därefter kan programmet startas.



Simulatorns hantering av minnet

Användningen av adressrummet i en MD407 framgår av figuren till höger.

Adressrymden hos MD407-simulatorn används på följande sätt.

0x0800 0000 - 0x080F FFFF	1 MB Flash
0x2000 0000 - 0x2001 BFFF	112 kB SRAM
0x4000 0000 - 0x5FFF FFFF	512 MB Periferikretsar
0xE000 0000 - 0xFFFF FFFF	512 MB ARM systemregister

Adresser utanför dessa block används ej.

Tillverkar-specifikt	FFFF FFFF	
	E010 0000	
	E00F FFFF	
Periferibuss		
	E000 0000	
	DFFF FFFF	
Externa enheter		
	A000 0000	
	9FFF FFFF	
Extern RAM-minne		
	6000 0000	
	5FFF FFFF	
Periferikretsar		
	4000 0000	
	3FFF FFFF	
Block 1 SRAM		
	2000 0000	
	1FFF FFFF	
Block 0 kod		
	0000 0000	

Under fliken Memory kan minnesinnehållet studeras och i vissa fall ändras. Minnesinnehåll visas i block om 128 bytes.

- Du kan ändra visningsintervallet med hjälp av rullningslisten till vänster. Som alternativ kan du ange en startadress för blocket som ska visas (Display base). Efter att ha skrivit in en ny adress måste du trycka ned Enter-tangenten för att ändringarna ska få effekt.
- Allow changes in RO-memory beskrivs längre ned.
- Du kan välja (Display size) om du vill att minnet ska visas som word (32 bitar), halfword (16 bitar) eller byte (8 bitar).

Memory control

Display base

Display size

☒ Allow changes in RO-memory

	0x00	0x04	0x08	0x0C
0x20000000	49044803	49046008	68104A04	E7FC6008
0x20000010	55555555	40020C00	40020C14	40021010
0x20000020	00000000	00000000	00000000	00000000
0x20000030	00000000	00000000	00000000	00000000
0x20000040	00000000	00000000	00000000	00000000
0x20000050	00000000	00000000	00000000	00000000
0x20000060	00000000	00000000	00000000	00000000
0x20000070	00000000	00000000	00000000	00000000

Display size

☒ Allow changes in RO-memory

	0x0	0x2	0x4	0x6	0x8	0xA	0xC	0xE
0x20000000	4803	4904	6008	4904	4A04	6810	6008	E7FC
0x20000010	5555	5555	0C00	4002	0C14	4002	1010	4002
0x20000020	0000	0000	0000	0000	0000	0000	0000	0000
0x20000030	0000	0000	0000	0000	0000	0000	0000	0000
0x20000040	0000	0000	0000	0000	0000	0000	0000	0000
0x20000050	0000	0000	0000	0000	0000	0000	0000	0000
0x20000060	0000	0000	0000	0000	0000	0000	0000	0000
0x20000070	0000	0000	0000	0000	0000	0000	0000	0000

Display size

☒ Allow changes in RO-memory

	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
0x20000000	03	48	04	49	08	60	04	49	04	4A	10	68	08	60	FC	E7
0x20000010	55	55	55	55	00	0C	02	40	14	0C	02	40	10	10	02	40
0x20000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x20000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x20000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x20000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x20000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x20000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

READ WRITE MEMORY (RWM)

Figuren visar adressutrymmet 2000 0000₁₆-2000 007F₁₆, detta minne är av SRAM-typ, dvs. läs och skrivbart minne.

Innehållet i detta minne kan ändras genom att skriva in ett nytt (hexadecimalt) värde. Ändringsbara minnesinnehåll har en vit bakgrund.

	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
0x20000000	03	48	04	49	08	60	04	49	04	4A	10	68	08	60	FC	E7
0x20000010	55	55	55	55	00	0C	02	40	14	0C	02	40	10	10	02	40
0x20000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x20000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x20000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x20000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x20000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x20000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

READ ONLY MEMORY (FLASH)

En annan typ är de minnesareor som är reserverade som ROM-typ (*läsbart minne*). För denna kategorin minne har fönstren för minnesinnehåll *grå* bakgrund. Innehållet kan då inte ändras.

I ett verkligt system kan sådana minnen normalt sett inte skrivas, (en speciell programmeringsrutin krävs) men i simulatören vill vi i bland kunna initiera detta minne. Av denna anledning finns knappen Allow changes in RO-memory. Då den är aktiv behandlas ROM som skrivbart minne.

☐ Allow changes in RO-memory

	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
0x08000000	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x08000010	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x08000020	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x08000030	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x08000040	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x08000050	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x08000060	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x08000070	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

☒ Allow changes in RO-memory

	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
0x08000000	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x08000010	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x08000020	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x08000030	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x08000040	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x08000050	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x08000060	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0x08000070	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

PERIFERIKRETSAR OCH SYSTEMREGISTER

Adressrymder med periferikretsar och systemregister visas med röd färg mot vit bakgrund. Observera att bara en mycket liten del av dessa adressrymder används. Sådana block är inte skrivbara från *ETERM8*.

	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
0x40000000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x40000010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x40000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x40000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x40000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x40000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x40000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x40000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

ICKE ANVÄNT MINNE

Den sista kategorin i adressrymden är helt enkelt odefinierade block. I *ETERM8* illustreras dessa med frågetecken mot grå bakgrund.

	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
0x30000000	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??
0x30000010	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??
0x30000020	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??
0x30000030	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??
0x30000040	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??
0x30000050	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??
0x30000060	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??
0x30000070	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??

Studiematerial, Maskinorienterad programmering

Utöver arbetsboken *Maskinorienterad programmering med MD407* ingår även följande material utformat så att du samtidigt kan arbeta självständigt med dessa:

Tryckt häfte

- *Quick Guide* – avsedd att användas under tentamen eftersom det normal inte är tillåtet att använda utskrifter av PDF-version.

Programvaror: (Windows, Linux och MacOS)

- *ETERM8/SimServer*
- *CodeLite med GCC och GCC för ARM*, distribution som anpassats speciellt för detta material.

Dokument (PDF):

- *Quick Guide* - elektronisk form av det tryckta häftet
- *En snabb introduktion till maskinorienterad C* – kompendium som behandlar C speciellt som ett maskinnära programspråk.
- *ETERM8 och MD407, inledande övning 1* – introducerande övning inför laborationer.
- *GDB och SimServer med MD407, inledande övning 2* – introducerande övning inför laborationer.
- *GCC och SimServer med MD407, inledande övning 3* – introducerande övning inför laborationer.
- *Laborationsuppgifter med simulator, MD407* – anvisningar för att genomföra en hel laborationsserie. Samtliga uppgifter har här utformats för att kunna genomföras enbart med hjälp av de programvaror som omfattas i kursen. Det behövs alltså inte tillgång till laborationsutrustningen. För flertalet uppgifter hänvisas direkt till arbetsboken, du behöver därför även denna för att arbeta med detta häfte.