

TSEA28 Datorteknik Y (och U)

Föreläsning 4

Kent Palmkvist, ISY



Dagens föreläsning

- Olika typer av hopp
- Vanliga programkonstruktioner
- In och utmatning
- Introduktion till ARM Cortex-M (labbdatorn)

Praktiska kommentarer

- Labanmälan lab 1 öppnar Må 27/1 kl 12.45
 - Logga in på Lisam, kursens kursrum (TSEA28_2025VT_T3)
 - Välj länk Anmälan (långt ned till vänster)
 - Ange 2 olika tillfällen (A och B för lab 1)
 - Motsvarande för 4 ytterligare tillfällen (A och B för lab 2 och 3)
- Labbar genomförs i grupper om 1-2 personer/grupp
 - Max 16 grupper per labbtillfälle!
- Datoraccess på plats (MUX1)
 - Labbuppgiften görs på plats med linux-maskiner
 - Kort-access efter alla gjort lab1a
- Labbförberedelse med distansutrustning (MUX2)
 - Ingen fysisk närvaro i labb MUX2
 - Fjärrinloggning till MUX2 beskrivs på
 - www.isy.liu.se/edu/kurs/TSEA28/aboration/Work_at_home.html
 - Video (gammal) finns i lisams kursmaterial

Praktiska kommentarer, forts.

- Tips: Börja läsa igenom labbmateriel redan nu
 - Bra att vara förberedd innan 1:a lektionen
 - Vi räknar inte med att ni kan processorbeskrivningen utantill, men att ni vet var man hittar den informationen i manualen
- Manual till labbmateriel (Introduktion till Darma) uppdateras allt eftersom
 - Information för att kunna genomföra lab1 finns i materialet
 - Mer information finns i länkade dokument (ARM och Ti manualer) (se kursmaterialsiden)
 - Skicka gärna kommentarer/frågor via email

Hopp

Olika typer av hopp (DARMA)

- B (Branch)
 - Ovillkorligt (utförs alltid)
- BNE (Branch Not Equal)
 - Villkorligt, hoppa om icke lika (antag jämförelse gjorts innan)
- BEQ (Branch EQual)
 - Villkorligt, hoppa om lika (antag jämförelse gjorts innan)
- BL (Branch and Link)
 - Branch and link, subrutinanrop som sparar var nästa instruktion efter subrutinanropet är
- BX (Branch eXchange)
 - Återhopp från subrutin, återställer PC till värde efter BL-anrop

Relativa och absoluta hopp

- Skillnad mellan absoluta och relativa hopp
 - Brukar finnas både jump (absolut) och branch (relativt)
 - Även möjligt med BSR och JSR (branch respektive jump subroutine)
- Absolut hopp: Adressen i argumentet är adress
 - Exakt den adress som PC ska få när hoppet är utfört
 - Fördel: behöver inte beräknas, kan peka på fast adress även om kod flyttas
- Relativt hopp: argumentet är ett avstånd relativt aktuell PC
 - Argumentet anger hur många steg framåt eller bakåt hoppet ska göras relativt aktuellt adress i PC
 - Fördel: hopp inom rutinen behöver inte ändras även om rutinen placeras på annan plats i minnet

Villkorliga hopp

- Beroende på indata kan vissa hopp betyda olika saker
 - Tvåkomplements data ger annan betydelse hos jämförelse än positiva heltal
 - $1100 > 0100$ för positiva heltal (dvs $12 < 4$), $1100 < 0100$ (dvs $-4 < 4$) för 2-komplement (samma värden i flaggorna efter subtraktion i båda fallen)
- Villkorliga hopp ofta kombinationer av flaggor
 - BLT (branch less than) antar A-B beräknats, testar om teckenbiten (N) skiljer sig från 2-komplements spill (V) (dvs (N=0 och V=1) eller (N=1 och V = 0))
 - Om A-B beräknats så tas hoppet om $A < B$ (dvs svar korrekt negativt eller spill med positivt svar)
- Ibland kan flera namn finnas på samma operation (ger läsbar kod)
 - Datorn testar bara flaggor, den kommer inte ihåg vilken operation det var som påverkade flaggorna

Olika villkorliga hopp i ARM

Condition Code	Meaning (for cmp or subs)	
EQ	Equal	Z==1
NE	Not Equal	Z==0
GT	Signed Greater than	(Z==0) && (N==V)
LT	Signed Less Than	N!=V
GE	Signed Greater Than or Equal	N==V
LE	Signed Less Than or Equal	(Z==1) (N!=V)
CS or HS	Unsigned Highter or Same (or Carry Set)	C==1
CC or LO	Unsigned Lower (or Carry Clear)	C==0
MI	Negative (or Minus)	N==1
PL	Positive (or Plus)	N==0
VS	Signed Overflow	V==1
VC	No Signed Overflow	V==0
HI	Unsigned Higher	(C==1) && (Z==0)
LS	Unsigned Lower or Same	(C==0) (Z==0)

Vanliga programkonstruktioner

Villkorliga hopp, exempel

- Implementera motsvarande pseudokod

- Antag variabel A i register r0
- Antag 2-komplementsform

start:	start:	; cmp beräknar r0-42, dvs A-42
	cmp r0,#42	; kan även göra subs r0,#42
		; men då förstörs r0 eftersom
		; resultatet från subtraktionen
		; sparas i r0
	ble notlarger	; ble är motsats till bgt, alltså
		; hoppa om B-A gav ett positivt svar
	:	
	sekvens1	; instruktioner i sekvens1
	:	
	b done	; hoppa till efter if-satsen
if (A > 42) then	notlarger:	; Början på sekvens2
sekvens1	:	
else	:	
sekvens2	sekvens2	; Instruktioner som körs of A <= 42
	:	
done:	done:	; plats för 1:a instruktion efter if-sats

Loop, exempel

- Implementera motsvarande pseudokod

- Antag variabel i R0
- Antag 2-komplement

start:	start:	; cmp beräknar r0-42, dvs A-42
	cmp r0,#42	; kan även göra subs r0,#42 men
		; då förstörs r0 eftersom resultatet
		; sparas i r0 (Beräknar r0-42)
	bge done	; bge är motsats till bgt, alltså hoppa
		; om B-A gav ett positivt svar
	:	
	sekvens1	; instruktioner i sekvens
	:	; (ingen riktig assemblerinstruktion)
	:	
	b start	; hoppa till start av loopsekvensen
while (A < 42) :	done:	; plats för kod efter loop
sekvens1		
done:		

In och utmatning

Användning av bl i exempeldator

- bl är subrutinanrop som gömmer koden för hur sensorer och display hantera
 - Kallas ibland för drivrutiner (byte hårdvara kräver bara byte av drivrutin)
 - Någon måste fortfarande skriva rutinerna och bygga in gränssnittet i datorn
- Dessa subrutiner är exempel på olika I/O-funktioner
 - Skicka ut data till display
 - Läsa av sensor och låssensor
- Hur?
 - Måste finnas en del i modelldatorn där display och sensorer kan anslutas
 - Värde i minne eller register måste kunna skickas ut, värde från sensor kunna läsas av

In och utdata, speciella instruktioner

- Vissa datorer använder speciella instruktioner för att skicka data in och ut från I/O
 - Kräver extra instruktioner
 - in R,iadress ; läs av inport nr iadress och sparar i R
 - out R,iadress ; skicka värde i R till utport nr iadress
 - Kräver extra signaler ut från processorn (styrsignaler)
 - Anger om data ska till/från I/O eller minne
 - In och utdata påverkar inte tillgänglig mängd minne
 - Eget adressutrymme för I/O separat från vanligt minne
 - out R,100 ; R:s värde ut på I/O enhet 100, INTE i minne adress 100
 - str R,100 ; skriver till minne adress 100, påverkar inte I/O-enheter

In och utdata, minnesmappad I/O

- Vissa datorer (dom flesta numera) använder speciella minnesadresser för att ta emot och skicka data från/till I/O
 - Kräver inga extra instruktioner eller extra styrsignaler
 - Enkelt att hantera i högnivåspråk (t ex C/C++)
 - Kräver att vissa minnesadresser inte kan användas som vanligt minne
 - Läsning kan ge annat värde än det som skrivs till adressen
- Används av labutrustningen

; display finns på adress
; 65532, sensor på 65534

putdisplay: ; adress 2000
str R,65532
bx lr

readsensor: ; adress 2200
ldr R,65534
bx lr

Avkodning av minnesmappad I/O

- Exempel: vill ha en I/O-port på adress 0001 0010 0011 0100.
 - Alternativ 1: fullständig avkodning (DARMA)
 - Alla 16 bitarna måste undersökas
 - Kräver stor logisk nät (hårdvara) för att detektera rätt adress (långsamt, dyrt)
 - Numera ganska vanligt (hårdvara billig och snabb)
 - Alternativ 2: ofullständig avkodning
 - Avkoda bara en del av bitarna (t ex bara de första 12)
 - Går snabbare, tar mindre hårdvara
 - Porten hamnar då på alla adresser som börjar med 0001 0010 0011, dvs 16 olika
 - Skrivning till 0001 0010 0011 0100 samma som skrivning till 0001 0010 0011 1100

Kuriosa: Bitbanding (alternativ användning av adressavkodning, finns i labutrustningen...)

- Vissa I/O-register placerade på många adresser
- Del av adressen används för att nollställa vissa bitar när adressen läses
 - Exempel: I/O-port på adress 0x12300-0x123FF (alla adresser pekar på samma port).
 - Bit 9-2 i adressen anger om biten som läses från porten ska nollställas. Dvs bit 9 i adress måste vara 1 för att bit 7 i inläst värde ska vara portens värde. Om bit 9 i adress = 0 så nollställs alltid bit 7 i inläst värde från porten.
 - Exempel: Läsning på adress 0x123c6 kommer nollställa bitarna 7, 6, 5 och 0 i värdet som läses.
 - Orsak till att labbutrustningen läser adress 0x4002507c för port F eftersom bit 7-5 inte kan användas. Alla bitar läses om adress 0x4002503fc används istället.

Logiska operationer och deras användning

Logiska operationer

- AND R, #value (bitvis AND funktion) 0x42 = hexadecimalt (4*16+2)
 - Bit för bit and (1 om båda är 1)
 - mov R, #0x23
 - and R, #0x42 ; 00100011 & 01000010 = 00000010
- ORR R, #value (bitvis OR funktion)
 - Bit för bit or (1 om någon eller båda är 1)
 - mov R, #0x23
 - orr R, #0x42 ; 00100011 | 01000010 = 01100011
- EOR R, #value (bitvis Exklusiv OR)
 - Bit för bit xor (1 om endast en bit är 1)
 - mov R, #0x23
 - eor R, #0x42 ; 00100011 ^ 01000010 = 01100001

Testa om bit är 0 eller 1

- Kontrollera om specifik bit i indata är =1

```
ldr R,switches ; läs av många switchar där
```



```
ands R,#0x04 ; varje switch går till en bit
```



```
 ; Kontrollera switch kopplad
```



```
 ; till bit 2
```



```
 ; Z-flaggan visar resultat
```



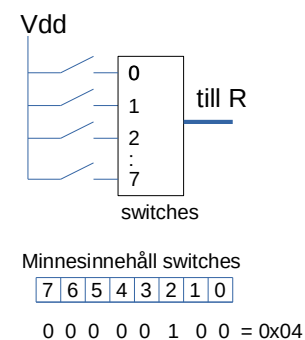
```
 ; Z=0 om bit = 1
```



```
bne pressed ; knappen tryckt (=1), gör något
```



```
 ; annars gör något annat
```
- Värden på andra bitar i inläst värde har ingen påverkan!



Sätt en bit till 0 eller 1 eller invertera

- Sätt specifik bit

```
ldr R,led ; läs nuvarande tända lysdioder
```



```
orr R,#0x01 ; tänd lysdiod 0 utan att
```



```
str R,led ; påverka övriga lysdioder
```
- Nollställ specifik bit

```
ldr R,led ; läs nuvarande tända lysdioder
```



```
and R,#0xFD ; släck lysdiod 0 utan att
```



```
str R,led ; påverka övriga lysdioder
```
- Togglar (0->1 eller 1->0) specifik bit

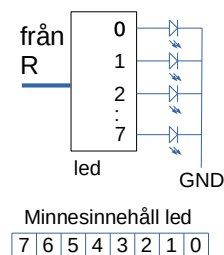
```
ldr R,led ; läs nuvarande tända lysdioder
```



```
eor R,#0x06 ; ändra lysdiod 1 och 2
```

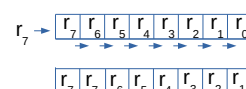
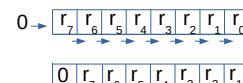
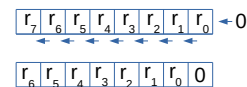


```
str R,led ; uppdatera utsignalen
```



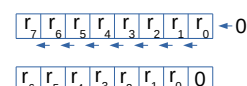
Skiftooperationer

- LSL, Logical shift left
 - Skifta R åt vänster (fyll på med 0 till höger)
 - Motsvarar multiplikation med 2
- LSR, Logical Shift Right
 - Skifta R åt höger (fyll på med 0 till vänster)
 - Motsvarar division med 2 (för positiva heltal)
 - Blir inte rätt för tvåkomplement!
- ASR, Arithmetic Shift Right
 - Skifta R åt höger, kopiera MSB (teckenbit!)
 - Motsvarar division med 2 (för 2-komplement)

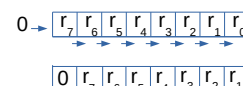


Exempel på skiftinstruktioner

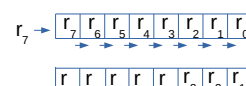
- Argument till skiftinstruktioner anger antal steg
- Antag R = 11001011
 - LSL R,#1 => R = 10010110
 - LSR R,#1 => R = 01100101
 - ASR R,#1 => R = 11100101
 - LSL R,#2 => R = 00101100



LSL



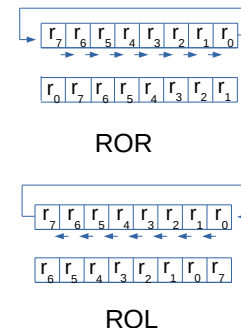
LSR



ASR

Rotation

- Rotation höger (ROR)
 - Kopierar ibland in data även i C-flaggan
- Rotation vänster (ROL)
 - ARM saknar separat ROL instruktion
 - Rotation vänster kan fås genom högerrotation 8-n steg för ett 8-bitars register
 - Ex: ROR R, #7 => samma som ROL R, #1 om registret är 8 bitar långt
 - ARM har 32 bitars register => ROL R0, #n = ROR R0, #(32-n)
 - Bytet till ROR sköts av assemblern



Andra möjliga instruktioner

- Multiplikation, division
 - I många enklare processorer saknas division (ev. även multiplikation)
- Bitmanipulering
 - Testa och sätt/nollställ enskilda bitar
 - Kan implementeras med vanlig and/or istället
- Se kapitel i Introduktion till Darna eller ARM:s manual
 - Många fler som jag inte kommer presentera nu
 - Vissa kommer diskuteras i samband med metoder för snabbare exekvering av program

Labbutrustningen DARMA, ARM Cortex-M

ARM processorer

- Flertal processorvarianter utvecklade under många år
 - Cortex-A för applikationer (t ex i mobiltelefoner, Rpi etc.)
 - Cortex-M för styrning och liknande (microcontroller)
 - Cortex-R för säkerhetsapplikationer (router, etc.)
- Samma grundläggande instruktionsuppsättning
 - 32-bitars instruktionsuppsättning, finns även en 64-bitars version av Cortex-A (t ex i RPi3 och Rpi4)
- Varje instruktion 32-bitar lång
 - 16 generella register R0-R15, varav flera har speciell funktion
 - R15 = programräknare, R14 = stackpekare, R13 = länkregister

ARM processorer, forts.

- Instruktionsbeskrivning
 - Operation rd, rn, operand
 - Operand kan vara konstant eller annat register
- 3 operander! Destination, källa1, källa2
 - Om 2 operander anges antas $rd = rn$
- Exempel
 - Add r3, r4, #3 ; beräkna $r4+3$, spara resultat i r3
 - Add r4, #3 ; beräkna $r4+3$, spara resultat i r4
 - Add r1, r2, r3 ; beräkna $r2+r3$, spara resultat i r1

ARM instruktionsuppsättning

- 2 typer av instruktionsuppsättningar hos ARM processorer
 - "Standard" ARM
 - Ofta exempel på nätet
 - Används t ex i Raspberry pi
 - Thumb (detta används i Cortex-M, dvs DARMA) för kompaktare kod (mindre programminne pga 16-bitar långa instruktioner)
- Vissa modeller har även utökade instruktionsuppsättningar
 - Flyttal (decimtal/float/double/real) (DARMA, men vi använder den inte)
 - DSP (signalbehandling)
 - SIMD (samma operation på många data samtidigt)

THUMB instruktionsuppsättning

- Kompaktare kod
 - Varje instruktion är 16 eller 32 bitar lång (vanlig ARM alltid 32 bitar lång)
 - Vissa versioner av instruktioner finns inte i Thumb
- Cortex-A kan växla mellan Thumb och ARM instruktionsuppsättning i samma program
 - Välj mha LSB (minst signifikant bit) i hoppadress
 - Instruktionerna startar i alla fall på jämn adress
- Cortex-M (dvs labbutrustningen) kan bara köra Thumb instruktionsuppsättning

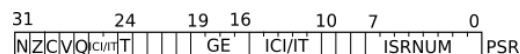
ARM Cortex M familjen

- Flera versioner finns tillgängliga (M0, M1, ... M7)
 - Samma maskinkod
 - Kod för en processor kan köras direkt i en annan processor (om specialfunktioner/specialinstruktioner undviks)
 - Olika prestanda (snabbare men större och mer effektförbrukning)
 - M0: minimal
 - M7: Snabb
 - Olika extra funktioner
 - Enhet för flyttalsberäkningar, support för signalbehandlingsinstruktioner
- Labbutrustningen har en Cortex M4F, dvs medelsnabb med stöd för flyttal

Register i Cortex M familjen

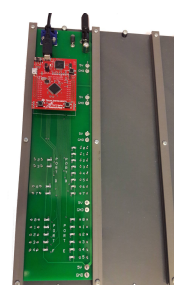
- R0 till R15
 - 32-bitars dataregister
 - R13 = SP (stackpekare)
 - R14 = LR (länkregister)
 - R15 = PC (programräknare)
- PSR
 - 32-bitars statusregister med flaggor (+ statusbiter)

	R0
	R1
	R2
	R3
	R4
	R5
	R6
	R7
	R8
	R9
	R10
	R11
	R12
	R13 (SP)
	R14 (LR)
	R15 (PC)
	PSR



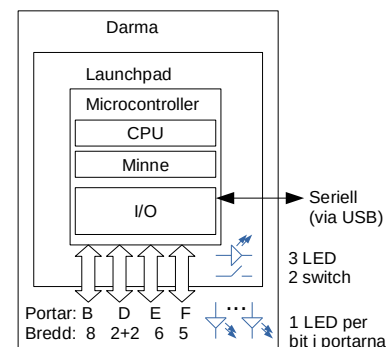
Laborationssystemet Darma

- Fysisk vy
 - Röda kretskortet längst upp innehåller processorn (TiVA C LaunchPad)
 - Resten är strömförsörjning och möjligheter att koppla in andra enheter, samt indikering av värde på anslutningarna
- Ansluts via USB-port till linuxdator
 - Både programmering och styrning
 - Även en seriekommunikationskanal



Laborationssystemet Darma, mikrokontrollern TM4C123G

- CPU: ARM Cortex M4 processor
 - 32 bitar databuss, 32 bitars addressbuss
 - 16 MHz klocka (ca 50-150 ggr långsammare än en smartphone)
- Minne på kretsen
 - RAM (läs och skrivbart) 32 Kbyte
 - FLASH (endast läsbart) 256 KByte
- I/O-enheter
 - Parallellport, programmerbara anslutningar (val av in eller ut)
 - Serieport (många...)
 - Många fler (I2C, USB, timers, PWM etc.)



Microcontroller = enklare enchipsdator med processor, minne och I/O

Darma minneskarta

- 256 KB FLASH-minne (CPU kan bara läsa)
 - 0x00000000 - 0x0003FFFF programmeras från PC
 - Innehåller programkod
- 32 KB RAM (CPU kan läsa och skriva)
 - 0x20000000 - 0x200001FF stack
 - 0x20000200 - 0x20007FFF plats för variabler etc.
- I/O-kretsar, data och konfiguration
 - 0x4000C000 - 0x4000CFFF Serieport (skicka/ta emot text till/från dator)
 - 0x40004000 - 0x40005FFF GPIO (parallellport) Port B
 - 0x40025000 - 0x40025FFF GPIO (parallellport) Port F
- De flesta adresser används inte
 - Vissa adresser kan skada hårdvara?
 - Inte i denna design, men möjligt i andra

00000000	ROM (FLASH)
0003FFFF	
20000000	RAM
20007FFF	
40000000	I/O-enheter GPIO A-F UART mm.
43FFFFFF	
E0000000	Intern IO
E00FFFFF	

Programmeringsmiljö Code Composer Studio

- Komplet IDE (Integrated Development Environment)
 - Editering av källkod
 - Kompilering/assemblering/länkning av program (programmets bitmönster)
 - Programmering av minnet i Darna
 - Kommunikation med seriell anslutning över USB
 - Debugstöd, t ex köra en instruktion åt gången, undersöka registervärden etc.
- Bygger på eclipse

Code composer studio, forts.

- Vid start väljs ett workspace
 - Anger var filer ska hamna i filsystemet
 - Olika projekt placeras i samma workspace
- Varje program som ska skrivas placeras i ett eget projekt
 - Samlar ihop nödvändiga filer och definitionsfiler
 - Håller ordning på vilka filer som ändrats och vad som behöver assembleras/kompileras eller länkas.
 - Använd ett projekt för varje deluppgift, t ex lab1_grundversion och lab1_utbyggd

Assemblering, länkning, programmering

- Översättning av assemblerkod sker i flera steg innan Darma programmeras
 - Varje steg producerar meddelanden i loggfönster
 - Programkod läggs automatiskt till för att initiera darma (t ex sätter stackpekare)
 - `tm4c123gh6pm_startup_ccs.c`, `boot.asm`
- Assemblering/kompilering
 - Översätt källkodstext från `.asm` fil respektive `.c` fil till objektformat (ett mellanformat utan absoluta adresser)
 - Flera olika `.asm`-filer och `.c`-filer kan assembleras/kompileras
- Länkning
 - Kombinera ihop alla assemblerade filer, bestäm på vilka adresser allt ska hamna
 - Kan även inkludera kompilerad C-kod etc.

Assemblerfilens uppbyggnad

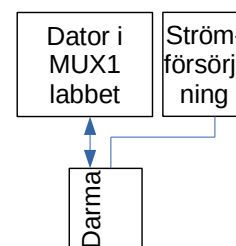
- Till alla labbar finns en mallfil som innehåller definitioner och initiering
 - Måste läggas till projektet ni ska jobba med
 - Olika mallfil beroende på vilken labb
 - Hämtas i filsystemet/laddas ned och importeras sedan in i aktuella projektet
- Assemblerprogrammet förväntas startas på platsen `main`:
 - Mallen innehåller sedan en del subrutiner som ni behöver anropa
 - `Inituart`: sätt igång seriekommunikation så utskrift kan fås på dator
 - `InitGPIOF`, `initGPIOB`: initiera portar så data kan hämtas och skickas
 - Definitioner av adresser till I/O-enheter etc finns också
 - `GPIOF_GPIODATA`: port F dataregister

Styrkommandon till assemblern

- Instruktioner i .asm-filen som inte motsvarar instruktioner till processorn
 - Beskriv förväntad form av kod (Thumb)
 - .thumb
 - Placera assemblerad kod i Flash-minnet (ROM)
 - .text
 - Starta på en jämn adress
 - .align 2
 - Ange att platsen main definieras i denna fil så man från andra filer hittar den vid länkning
 - .global main

Att komma igång

- På laborationshemsidan finns dokumentet "Introduktion till Darma"
 - Beskriver allt som behövs för att använda Darma
 - Labbarna fortsätter utvecklas, nya versioner av dokumentet kan komma att läggas ut under kursens gång.
 - Skicka gärna mail med frågor/kommentarer
- Logga in på en maskin i MUX2, öppna terminalfönster
 - Ladda modulen courses/TSEA28
 - Kör tsea28_active för att kontrollera att ingen annan redan använder maskinen
 - Starta med ccstudio
 - Video på lisam visar alla steg
- Hemma eller thinlinc (maskin utan Darmakort/TiVA C Launchpad)
 - Kan editera och kompilera, men inte simulera/köra
 - Ladda hem programvaran från www.ti.com/ccs



Programmering av Darna

- Programmering av Flashminnet görs varje gång övergång till debugläge görs
 - Kompilering/assemblering/länkning görs automatiskt om det behövs
 - Programmet ligger kvar i Darna tills nytt program laddas in
- Exekvering av program
 - Gå först till main:
 - Initieringsrutinerna från boot.asm kan inte stegas igenom
 - Kör sedan med Resume (F8, eller grön playknapp)
 - Körning kan stoppas när som helst med Suspend (Alt-F8, eller paussymbol)
 - Lämna körning med Terminate (röd stoppknapp), återgår till editering och kompilering/assemblering/länkning

Felsökning/test i ccstudio (med Darmakort)

- Kan ändra minne, register etc
 - Lägg till olika vyer mha Window->Show view
- Kan stega instruktion för instruktion (step into)
 - Kan även stega subrutin för subrutin (step over)
- Sätt brytpunkter för att automatiskt stanna när den instruktionen ska utföras
 - Dubbelklicka på radnumret i kodfönstret

Live demo av ccstudio via thinlinc + ssh

Logga in på thinlink
ssh -X muxen2-103.edu.liu.se
module load courses/TSEA28
tsea28active
ccstudio



Görs bara om ni INTE är i MUX1-labbet

Skapa tomt ccsproject
Lägg till lab1.asm från /courses/TSEA28
(eller ladda ned från labbsidan)
Editera lab1.asm
Assemblera lab1.asm

Live demo av Darma (med kort)

Logga in på maskin I MUX1/MUX2
module load courses/TSEA28
ccstudio

öppna tidigare projekt

Starta debugläge

Go main

Kör programmet

Undersök register

Undersök minne

Ändra register

Stega igenom

Sätt brytpunkt

Kommentar: Måste köra initiering
av port innan porten
kan läsas/skrivas

