TSEA28 Datorteknik Y (och U)

Föreläsning 4

Kent Palmkvist, ISY

LINKÖPINGS UNIVERSITET

Dagens föreläsning

- Resultat från beräkningar
  - Flaggor
- Andra operationer
  - Villkorliga hopp
  - Rotation, skift
- Vanliga programkonstruktioner
- Introduktion till ARM Cortex-M (labbdatorn)

Praktiska kommentarer

- Labanmälan lab 1 öppnar Må 23/1 kl 12.30
  - Logga in på Lisam, kursens kursrum (TSEA28_2023VT_7H)
  - 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/laboration/Work_at_home.html
    - Video finns i lisams kursmaterial

Praktiska kommentarer, forts.

- Tips: Börja läsa igenom labbmaterial 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 labbmaterial (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 kursmaterialsidan)
  - Skicka gärna kommentarer/frågor via email

Aritmetiska operationer och flaggor

Instruktioner för aritmetiska beräkningar

- Addition och subtraktion
  - Ingen skillnad hur det går till (additionsalgoritmen)
  - Använder samma instruktion för både positiva heltal och 2-komplement
  - Resultat tolkas olika beroende på talrepresentation
  - Samma bitmönster ut oberoende av indatats talrepresentation
  - Implementeras i ALU
  - Samma ordlängd på indata och utdata
- Multiplikation
  - Två olika instruktioner för positiva heltal respektive 2-komplement
  - Dubbel längd på resultatet
  - Generellt: m-bitars tal gånger n-bitars tal ger produkt med m+n bitar

Intressanta beräkningsresultat

- I tidigare exempel jämfördes likhet mellan tal
  - Beräknas genom subtraktion (A-B) och kontrollera om resultat = 0
  - Indikera resultat i Z-registret (1-bit) där Z=1 om result = 0, Z=0 annars
- Jämför med hur två tal A och B kan jämföras i vanliga programmeringsspråk
  - A = B
  - A > B
- Dessutom kan A och B vara tvåkomplement eller positiva heltal
  - Resultat utanför talområdet för 2-komplement?
  - Minnessiffra (carry/borrow) från beräkning?

Mer information om resultatet (flaggor)

- Efter aritmetisk operation (Addition, subtraktion) indikerar flaggor egenskaper hos resultat/beräkning
- Resultat 0? Lagras i Z-flaggan (Z=1 om resultat = 0)
  - Samma som i modelldatorn om comp ersätts med subtraktion
- Resultat negativt? Lagras i N-flaggan (=MSB i resultatet) för 2-komplement.
- Resultat gav minnessiffra ut från teckenbit, carry? Lagras i C-flaggan
  - Kan även ses som overflow om positiva heltal in (inte för 2-komplement)
  - Visar på lån (engelska: borrow) för subtraktion av positiva heltal
- Resultat gav spill (engelska: overflow) för 2-komplement? Lagras i V-flaggan
  - Antar indata och utdata i 2-komplements form.
  - Detekteras mha minnessiffra in till och ut från teckenbitens position är olika

Varning ang. flaggor

- Hur flaggor sätts och hur de används kan variera mellan processorfamiljer
  - Subtraktionens interaktion med C-flaggan skiljer sig mellan ARM och t ex 68000
  - Läs alltid manual för aktuell processor innan programmering
- ARM: Flaggor ändras bara om aritmetiska/logiska instruktionen anger det
  - Indikerat med extra S i slutet på instruktionen

    Add R0,#1 ; utför addition, men sätter inte flaggor

    Adds R0,#1 ; utför addition, och ändrar flaggor (Z, C, N, V).
  - Undantag: cmp ändrar alltid flaggor

Exempel på genererade flaggor

- 8-bitars addition (antag R-register i dator är 8 bitar)

```
mov R,#127
adds R,#3
```

```
                  Z = 0 (resultat inte noll)
  01111111        N = 1 (MSB i resultat är ett)
+ 00000011        C = 0 (fick ingen minnessiffra ut från MSB)
----------        V = 1 (130 får inte plats i
  10000010              8 bitar tvåkomplement)
```

- Vissa instruktioner kan även addera C, dvs om C=1 läggs även 1 till, adc -10+3

```
Instruktion som sätter C flaggan
mov R,#-10
adcs R,#3
```

```
          1       Z = 0 (resultat inte noll)
  11110110        N = 1 (MSB i resultat)
+ 00000011        C = 0 (ingen minnessiffra)
----------        V = 0 (-6 får plats inom 8 bitar)
  11111010
```

Användning av flaggor

- Två huvudsakliga användningsområden
- Skicka bitvärde mellan beräkningar
  - Exempel: addera två 64-bitars tal i en dator som bara kan addera 32-bitars tal
  - Addera först minst signifikant 32-bitars del (påverkar även C-flagga)
  - Addera därefter mest signifikant 32-bitars del inkl. C-flaggan
- Styr exekveringsflödet mha resultat
  - T ex välj en annan programdel om två värden olika (jämför studenträknarexemplet)
  - Villkorliga hopp (conditional branching)

Exempel på flaggor för långa additioner

- Antag två 16-bitars tal ska adderas i en 8-bitars dator
  - 0xABCD+0x7654 (två hexadecimala tal)
  - Kan bara addera 8 bitar per instruktion
- Addera först minst signifikant byte
  - 0xCD+0x54 = 0x21 samt C=1
- Addera sedan mest signifikant byte plus C flaggan
  - 0xAB + 0x76 + 1 = 0x22 samt C=1
    - + 1 kommer från C-flaggans värde i föregående steg
- Svar: 0xABCD+0x7654 = 0x2221 samt C=1
  - Eventuell overflow kontrolleras bara efter andra additionen

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)
  - ARM har 14 möjliga villkorliga hopp (se manual)
- 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:
  if (A > 42) then
    sekvens1
  else
    sekvens2
done:
```

```
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
notlarger:        ; Början på sekvens2
     :
    sekvens2      ;  Instruktioner som körs of A <= 42
     :
done:             ; plats för 1:a instruktion efter if-sats
```

Loop, exempel

- Implementera motsvarande pseudokod
  - Antag variabel i R0
  - Antag 2-komplement

```
start:
   while (A < 42) :
     sekvens1
done:
```

```
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
done:       ; plats för kod efter loop
```


Logiska operationer och deras användning

Logiska operationer

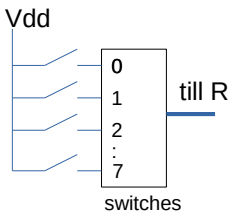
- AND R,#value (bitvis AND funktion)
  - Bit för bit and (1 om båda är 1)
  - mov R,#0x23
  - and R,#0x42 ; 00100011 & 01000010 = 00000010
- ORR R,#value (bitvis OR funktion)
  - Bit för bit or (1 om någon eller båda är 1)
  - mov R,#0x23
  - orr R,#0x42 ; 00100011 | 01000010 = 01100011
- EOR R,#value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

0x42 = hexadecimalt (4*16+2)

Testa om bit är 0 eller 1

- Kontrollera om specifik bit i indata är =1

```
ldr R,switches   ; läs av många switchar där
                 ; varje switch går till en bit
ands R,#0x04     ; 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
```



Minnesinnehåll switches

7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0

= 0x04

- Värden på andra bitar i inläst värde har ingen påverkan!

Sätt en bit till 0 eller 1 eller invertera

- Sätt specifik bit

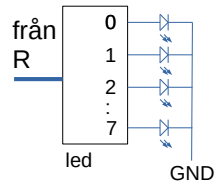
```
ldr R,led      ; läs nuvarande tända lysdioder
orr R,#0x01    ; tänd lysdiod 0 utan att
str R,led      ; påverka övriga lysdioder
```

- Nollställ specifik bit

```
ldr R,led      ; läs nuvarande tända lysdioder
and R,#0xFD    ; släck lysdiod 1 utan att
str R,led      ; påverka övriga lysdioder
```

- Toggla (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
```

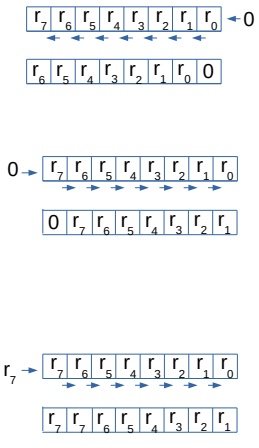


Minnesinnehåll led

7	6	5	4	3	2	1	0

Skiftoperationer

- 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

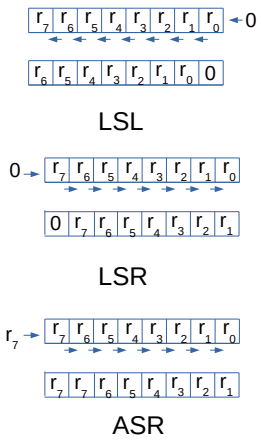
- Argument till shiftinstruktioner anger antal steg
- Antag R = 11001011

  LSL R,#1 => R = 10010110

  LSR R,#1 => R = 01100101

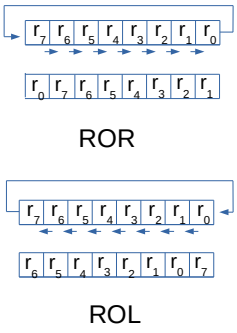
  ASR R,#1 => R = 11100101

  LSL R,#2 => R = 00101100



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 Darma 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) för kompaktare kod (mindre programminne pga 16-bitar långa instruktioner)
- Vissa modeller har även utökade instruktionsuppsättningar
  - Flyttal (decimaltal/float/double/real)
  - DSP (signalbehanding)
  - SIMD (samma operation på många data samtidigt)

THUMB instruktionsuppsättning

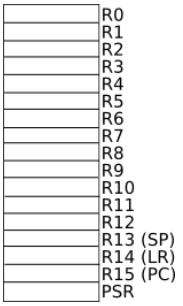
- Kompaktare kod
  - Varje instruktion är 16 eller 32 bitar lång (vanlig ARM alltid 32 bitar lång)
  - Vissa versioner av instruktioner finns inte i Thumb
- Cortex-A kan växla mellan Thumb och ARM instruktionsuppsättning i samma program
  - Välj mha LSB (minst signikant bit) i hoppadress
  - Instruktionerna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

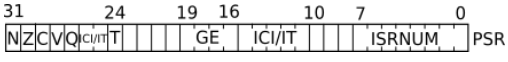
ARM Cortex M familjen

- 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

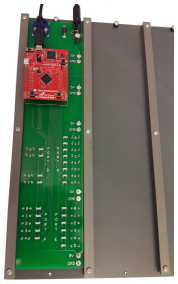
- R0 till R15
  - 32-bitars dataregister
  - R13 = SP (stackpekare)
  - R14 = LR (länkregister)
  - R15 = PC (programräknare)
- PSR
  - 32-bitars statusregister med flaggor (+ statusbitar)





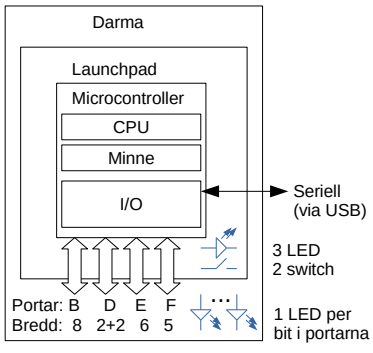
Laborationssystemet Darma

- 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

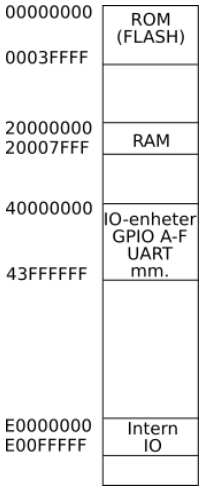
- CPU: ARM Cortex M4 processor
  - 32 bitar databuss, 32 bitars addressbuss
  - 16 MHz klocka (ca 50-150 ggr långsammare än en smartphone)
- Minne på kretsen
  - RAM (läs och skrivbart) 32 Kbyte
  - FLASH (endast läsbart) 256 KByte
- I/O-enheter
  - Parallellport, programmerbara anslutningar (val av 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



Programmeringsmiljö Code Composer Studio

- Komplett IDE (Integrated Development Environment)
  - Editering av källkod
  - Kompilering/assemblering/länkning av program (programmets bitmönster)
  - Programmering av minnet i Darma
  - 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.
  - Lämpligen 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)

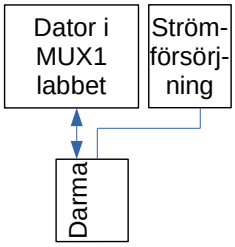
    .text
  - Starta på en jämn adress

    .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 Darma

- Programmering av Flashminnet görs varje gång övergång till debugläge görs
  - Kompilering/assemblering/länkning görs automatiskt om det behövs
  - Programmet ligger kvar i Darma 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 muxen2-003.edu.liu.se } Görs bara on ni INTE är i labbet
module load courses/TSEA28
tsea28active
ccstudio
  Skapa tomt ccsproject
  Lägg till lab1.asm från /courses/TSEA28
    (eller ladda ned frånlabbsidan)
  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

LINKÖPINGS UNIVERSITET