TSEA28 Datorteknik Y (och U)

Föreläsning 3

Kent Palmkvist, ISY

LINKÖPINGS UNIVERSITET

Dagens föreläsning

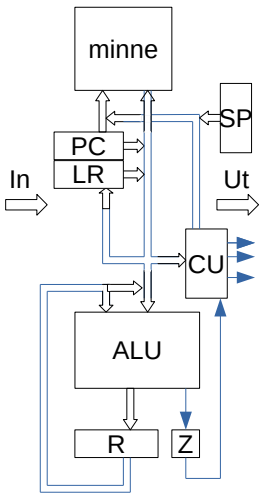
- Kort introduktion till laborationsdatorn
- Teckenrepresentation
- Talrepresentation
  - Binär aritmetik
  - 2-komplement

Praktiska kommentarer

- Kursanmälan
  - Behövs för att kunna komma åt funktioner i kursrummet på Lisam
  - Använder Lisams anmälningsfunktion för labanmälan
- Labanmälan kommer finnas tillgänglig med start måndag lunch (23/1 kl 12.30)
- Labbprogram tillgängligt i labben mux1 och mux2 (båda låsta, öppnas efter 1:a labhalvan slutförts)
  - Hårdvara ansluten bara i MUX2 (kan köra distansläget på dom)
  - MUX1-maskinerna enbart för inmatning och assemblering, ej köra/testa kod än (görs på plats)
  - Lite mer information om hur man ansluter till mux1/mux2 finns på
    www.isy.liu.se/edu/kurs/TSEA28/laboration/Work_at_home.html

Enkel datormodell

- Ett minne lagrar program, data och stack
- I processorn sitter ett antal register
  - PC, SP, R, Z (flagga), ev. LR
- Till processorn finns ett antal instruktioner implementerade
  - mov, ldr, str, add, cmp, bne, beq, b, bl, bx
    - Består av instruktionstyp + argument
  - Vissa kombinationer av register och argument tillåtna
    - T ex kan inte flytta värde direkt från PC till R
    - Begränsas av styrenheten CU och hur registren kopplats ihop
- Add, cmp och mov använder sig i detta exempel av 28-bitars data (men 32 bitar i varje minnescell????)
  - Potentiellt problem, löses senare



Instruktioner så här långt

- Flytta data
  - mov, ldr, str, push, pop
- Beräkna nya värden (addera, jämför), påverka Z-flagga
  - add, cmp
- Ändra instruktionsföljd (hopp)
  - bne, beq, b, bl, bx lr
- Eventuella speciella funktioner
  - In, out

Fler register behövs (ett R-register är för lite)

- Svårt t ex skriva värden i en tabell
  - Adressen till plats i tabellen (index) behöver beräknas
    - Lagras i R
  - Värdet som ska placeras på denna plats i tabellen behöver också lagras i R
    - Behöver fler register
- Fler register brukar finnas
  - Labbdatorn ARM har 16 (bl a ingår PC, SP och LR i dessa 16)
  - Gamla labdatorn (68000) har 16 (8 vanliga plus 8 speciella för adresser inkl SP)
  - 6502 (Apple II) har ett register A samt två indexregister för adresser.
    - Kombinerat med speciell adresseringsmode med minnesadress 0-255
  - 80x86 (laptop/stationär dator): 15 register (32 bitars register för 80386 och uppåt)

ARM-baserad labbutrustning

- En Cortex M processor från ARM (designad av ARM, implementerad av Texas Instruments)
  - 32-bitars processor (interna 32-bitar register)
    - Kan även flytta byte (8-bitar) och half-word (16-bitar) mellan register och minne
  - Little-endian (lagrar minst signifikant byte på lägst adress i minnet)
    - Kan i vissa implementationer ställas om i programmet (men inte i aktuell implementation)
- Subrutinanrop använder ett LR-register.
  - Manuell hantering av stack i samband med subrutinanrop
- Tillverkas av många chiptillverkare (TI, Broadcom, Samsung, NXP, etc.)
  - Vi kommer använda Ti Tiva TM4C123G
  - Andra exempel på ARM processorhårdvara: Raspberry Pi, Arduino Duo, Apple A3 (mfl telefonchip), Qualcomm Snapdragon, etc.

Labbutrustning, forts.

- ARM designat processor, men TI lagt till I/O-enheter, minne etc.
  - Mycket kan konfigureras bland I/O-enheterna
    - > 800 olika register
    - Färdigt startprogram som sätter registren finns tillgängligt
- Dokumentation
  - ARM har dokumenterat vad en Cortex-M4 processor har för instruktionsuppsättning. > 270 sidor
  - TI har dokumenterat hur implementation på chipet med alla I/O-enheter etc. fungerar. > 1400 sidor
  - ”Kortfattad” beskrivning av ovanstående i ”Introduktion till Darma” på kursens labsida

Minnesvärdens betydelse

- ASCII, binära tal

LINKÖPINGS UNIVERSITET

Betydelsen hos bitmönster i minne

- Värde/funktion hos bitmönster beror på förväntningar
- I exempelprogrammet (studenträknaren):
  - Värde i minnesadresser som innehåller programkod tolkas som maskininstruktioner
    - Exakt mening beror på vilken processor som används
  - Värde i minnesadresser som innehåller stacken tolkas som återhoppsadresser
- Även andra saker kan representeras
  - Text, telefonnummer, bilder, pekare, ...

LINKÖPINGS UNIVERSITET

Tecken och textrepresentation

- Ett tecken (bokstav, siffror, specialtecken etc) kan lagras i minnet
- Flera standarder finns
  - ASCII: 128 olika tecken (7 bitar), senare utbyggt till 8 bitar.
    - I vissa standarder har de extra 128 kombinationerna har olika utseende på tecken beroende på språk (ISO-8859-x)
  - UTF: variabel längd för att kunna representera många fler olika tecken (även vissa engelska/amerikanska tecken som ASCII saknar)
    - UTF-8: 8 bit för de vanligaste tecknen (7-bit ascii), upp till 4 byte per tecken
    - UTF-16: 16 bitar per tecken, vissa tecken tar ännu mer utrymme

Exempel på tolkning av minnesinnehåll

- Minnet adresseras med binärt värde, svarar med binärt värde
  - Adress (binärt) Värde binärt, Decimalt, ASCII
  - 000000...0000 01001000 72 'H'
  - 000000...0001 01100101 101 'e'
  - 000000...0010 01101010 106 'j'
  - 000000...0011 00100000 32 ' '
- Se sista sidan i labbanvisningar lab1 för fullständig ASCII tabell
- Exemplet kan även ses som ett 32-bitars little-endian värde (maskininstruktion)
  - 00100000 01101010 01100101 01001000 = type 4, dvs cmp R,#00000110101001100101010001000

Beskrivning av numeriska värden

- Hittills har decimaltal använts i vissa assemblerkommandon
  - Datorn använder binära bitar, behöver översätta talet till ett binärt mönster!
  - Exemplet med 8h timer: 80000
    - Binär motsvarighet 10011100010000000
- Översättning från 0:or och 1:or till ett heltal
  - Binära tal bygger på positionssystem precis som decimaltal
  - varje siffra viktas beroende på position i talet
    - Ental, tiotal, tusental för decimaltal
    - Ental, tvåtal, 4-tal, 8-tal etc. för binära tal

Binära positiva heltal

- Bit längst till höger är minst signifikant (Least Significant Bit, LSB)
  - Vikt 1
- Bit längst till vänster är mest signifikant (Most Signficant Bit, MSB)
  - Vikt = 2^(antal bitar-1)
- Exempel: 100101_2 Indikera att detta är ett binärt tal

  MSB LSB
- Översatt till decimal form:

  $1*32+0*16+0*8+1*4+0*2+1*1 = 37_{10}$

Positiva binära heltal, generellt

- Ett n bitars binärt heltal på formen
  $a_{n-1}a_{n-2}...a_1a_0$ representerar värdet

$$värde = \sum_{i=0}^{n-1} a_i 2^i$$

- Maximalt värde är $(2^n)-1$
- 6 bitar => max är (2^6)-1 = 63 = 111111_2
- Använd defintionen vid omräkning binärt till decimalt

Översättning från decimaltal till binärtal

- Algoritm

```
Temp = tal att översätta

Loop tills Temp = 0
  Om temp udda => lägg till 1:a,till vänster
  Om temp jämn => lägg till 0:a till vänster
  Temp = Temp/2 (trunkera, dvs kasta
                 bort eventuella 0,5)
end loop
```

- Exempel: 27 (decimalt)

Temp	binärtal	
27	1	(talet är udda)
13	11	(talet är udda)
6	011	(talet är jämnt)
3	1011	(talet är udda)
1	11011	(talet är udda)
0	Klar, svar: 11011_2	

- Kontroll:

  1*16+1*8+0*4+1*2+1*1=27

Alternativ beskrivning av binära tal

- Väldigt platskrävande att skriva allt i binär form
  - Vill ha en form som är lätt att översätta (manuellt) till binär form, men fortfarande kompakt
- Klumpa ihop ett antal bitar och beskriv varje klump med ett tecken
  - 2 bitar ger 4 kombinationer, t ex 0,1,2,3
  - 3 bitar ger 8 kombinationer, t ex 0,1,2,3,4,5,6,7
    - Kallas octal representation
  - 4 bitar ger 16 kombinationer (siffrorna 0-9 räcker inte!)
    - 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
    - Kallas hexadecimal representation

Hexadecimala tal

- Bas 16 (decimala tal har bas 10)
- Ett n-siffror hexadecimalt tal

$$värde = \sum_{i=0}^{n-1} a_i (16)^i$$

där a_i är hexadecimala siffror (värde 0 - F)

Decimalt	Binärt	Hexadecimalt
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Konvertering binärt till hexadecimalt

- Dela upp ett långt binärt tal i grupper om 4 bitar (från höger)
  - 10 1101 0011 1010
- Fyll eventuellt på med 0:or till vänster så det är 4 bitar i vänstra gruppen
  - 0010 1101 0011 1010
- Översätt varje grupp för sig. Om talvärde större än 9 använd bokstäver A-F
  - 2 D 3 A
- Indikera att detta är hexdecimalt mha basindikering eller på annat sätt
  - $2D3A_{16}$, $2D3A
  - 0x2D3A <- Används i labbutrustningen (C-syntax)

Beräkningar med positiva binära tal

Att addera med binära positiva heltal

- Samma metod som för decimaltal
- Från höger lägg ihop, sätt eventuell minnessiffra i nästa sifferposition till vänster
  - $0+0=0$, $0+1=1$, $1+0=1$, $1+1=10_2$, $1+1+1=11_2$
- Exempel 7+11

```
                    1      11     111    1111
  0111  (7)      0111    0111    0111    0111
 +1011  (11)    +1011   +1011   +1011   +1011
------           ------  ------  ------  ---------
                      0      10     010   10010
```

  - Dvs $7+11=10010_2 = 16+2 = 18_{10}$ (som förväntat)
- Framför båda talen finns ett oändligt antal 0:or
- Addition kan ge en carry-bit längst ut som utökar antal siffror i svaret (som i exemplet ovan)

Att multiplicera med binära positiva tal

- Med två enbitars tal kan fyra kombinationer beräknas

  0*0 = 0

  0*1 = 0

  1*0 = 0

  1*1 = 1
- Större binärtal (många bitar) som multipliceras med en bit ger antingen 0 eller det stora talet som svar

  0*100101 = 000000

  1*100101 = 100101

Multiplikation av två tal

- Samma metod som för decimaltal
  - Beräkna partialprodukter och summera sedan dessa
- Exempel: 7*11 = 111*1011

```
    111       111       111       111
  * 1011    * 1011    * 1011    * 1011
  ------    ------    ------    ------
               111       111       111   2^0 * 7 = 7
                       111       111     2^1 * 7 = 14
                               000       2^2 * 0 = 0
                             + 111       2^3 * 7 = 56
                             --------
                              1001101   64+8+4+1=7+14+56=77
```

- Kuriosa: Går att multiplicera valfria tal (även decimalt) om man bara kan dubblera och halvera samt addera tal

Subtraktion

- Fungerar på samma sätt som för decimaltal (om svaret blir positivt)
  - 6-5

```
                    10          10          10
  1 1 0       1 ǂ 0       1 ǂ 0       1 ǂ 0
- 1 0 1     - 1 0 1     - 1 0 1     - 1 0 1
-------     -------     -------     -------
                  1           0 1       0 0 1
```

  - Låna i 1:a steget, 10 – 1 = 1 svar som förväntat
- 5-6 = ? (har inget sätt att representera negativa tal!)

```
                          10          10
  1 0 1       1 0 1       ǂ 0 1       ǂ 0 1
- 1 1 0     - 1 1 0     - 1 1 0     - 1 1 0
-------     -------     -------     -------
                  1           1 1       ? 1 1
```

- 5 < 6 alltså beräkna 6-5 och sätt – framför?

Negativa binära tal

LINKÖPINGS UNIVERSITET

Hur representera negativa tal: Tecken-belopp

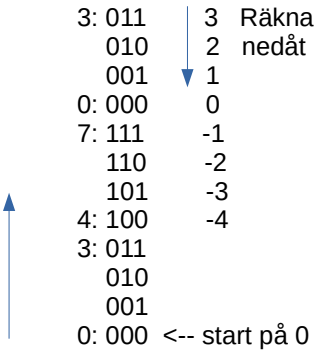
- Intuitiv lösning: Sätt en bit först som anger positivt (bit=0) eller negativt (bit=1)
  - Kallas för tecken-belopp representation (sign-magnitude)
  - Vänstraste biten 0: positivt tal, 1: negativt tal
  - Exempel (antag värden ska representeras med 4 bitar)
    - 7 = 0 111, -3 = 1 011
  - Dubbel representation av 0: +0 och -0
- Svårt att bygga addition och subtraktion
  - Måste först ta reda på om det är samma tecken på talen eller olika tecken
    - styr om subtraktion eller addition av beloppen ska göras
  - Måste även ta hänsyn till (vid subtraktion) vilket belopp som är störst

Tecken-belopp, forts.

- Används i flyttalsrepresentation
  - Exempelvärde: -2.345*10^15
  - datatypen double i Matlab
  - Datatyperna float och double i C/C++
- Beskrivs närmare i kurser som numeriska metoder etc.

Negativa tal på rätt sätt

- Vill addera och subtrahera utan hänsyn till tecken
- Motivering (antag räknare med 3 bitar, börja på 0)
  - Att räkna uppåt är lätt (dvs värde+1)
  - När räknaren nått maxvärdet börjar den om på 0
    - Jfr räkna till 20 men utan att använda tiotalssiffran (0,1,2,...,8,9,0,1,2,3...8,9)
- Jämför med att räkna nedåt istället
  - Samma sekvens som tidigare, men i motsatt ordning
  - Motsvarar att i varje steg subtrahera 1
- Med denna kodning fungerar det att räkna additon och subtraktion
  - så länge förväntade svaret går att representera med antalet bitar som finns i termerna som adderas/subtraheras



Negativa tal på rätt sätt, forts.

- Här finns bara en representation av värdet 0
- Biten längst till vänster anger om talet är negativt
- Addition och subtraktion, exempel

```
-2+3    1 1 0     två trebitars tal => beräknat svar har
      + 0 1 1                          3 korrekta bitar,
      ----                       extra 4:e biten ignoreras
      1001
```

Extra bit, ignoreras

```
                                    10
                                   1       låna igen från vikt 4
                                  10       låna från vikt 8
-1-2    1 1 1         1-(-2)      0 0 1
      - 0 1 0                   - 1 1 0
      -------                   -------
        1 0 1                     0 1 1
```

011	3
010	2
001	1
000	0
111	-1
110	-2
101	-3
100	-4

2-komplementsrepresentation

- För ett n-bitar binärt tvåkomplementstal gäller
  - n-1 bitar viktade precis som för positiva binära tal
  - MSB-biten har vikt $-2^{(n-1)}$
  - I ett 3-bitars tal är MSB vikt -4 (se tabell)
- Generellt: ett n bitars binärt tvåkomplementstal på formen $a_{n-1}a_{n-2}...a_1a_0$ har representerar värdet

$$Värde = -a_{(n-1)} * 2^{(n-1)} + \sum_{i=0}^{n-2} a_i 2^i$$

Exempel: 11011_{2C} = -16*1+8*1+4*0+2*1+1*1 = -5

011	3
010	2
001	1
000	0
111	-1
110	-2
101	-3
100	-4

Teckenbyte av 2-komplementstal

- Byte av tecken Y = -X
  - Invertera alla bitar hos X
  - Addera 1 på LSB positionen => Y
  - Kasta bort eventuell minnessiffra längst till vänster (vill ha samma antal bitar i Y som i X)
- Exempel

  -(3) = -(011)

  invertera => 100

  addera till LSB + 1

  summera -3=101
- Notera inledade 0:a för att visa att 3 är ett positivt tal innan teckenbytet.

Utökning av talområde (öka ordlängd)

- Positiva binära tal (både positiva heltal och 2-komplement)
  - Lägg till nollor till vänster

Exempel: Förläng ett 5-bitars tal till 8 bitars ordlängd
01001 => 00001001

- Negativa binära tal (2-komplement)
  - Att lägga nollor fungerar inte (-3 = 101, men 0101 = 5!!)
    - 0 som mest signifikant bit skulle alltid skapa positiva tal!
- Generellt för 2-komplement: kopiera teckenbiten
  - Teckenbiten för 2-komplement är 0 för positiva tal, 1 för negativa tal
  - Kopiera teckenbiten åt vänster för att utöka talområde
  - Ett n-bitars 2-komplementtal $x_{n-1}\ x_{n-2}\ x_{n-3} \ldots x_1\ x_0$ teckenförlängs till m bitar genom att lägga till m-n kopior av x_{n-1} till vänster

    $x_{n-1} \ldots x_{n-1}\ x_{n-2}\ x_{n-3} \ldots x_1\ x_0$

Exempel på översättning och ordlängdsändring

- Beskriv värdet -53 som ett 8-bitar 2-komplementtal
  - Översätt först positiva versionen, byt därefter tecken
- Börja med att konvertera 53 till binär form
  - $53 = 32+16+4+1 = 110101_2 = 0110101_{2C}$
    - Måste ha 0 först för positiva tal i 2-komplementform
- Gör om till -53 (byt tecken)
  - Invertera och lägg till ett: $1001010 + 1 = 1001011_{2C}$
  - Expandera till 8 bitar (kopiera teckenbiten)
- $-53 = 11001011_{2C}$

Att korta ordlängd

- Att minska antal bitar för att representera ett tal fungerar bara om talet får plats (dvs kan representeras) i den nya antalet bitar
- För positiva binära tal kan nollor till vänster tas bort
  - När en etta tas bort fås fel värde
- För 2-komplementsvärden kan kopior av teckenbiten tas bort
  - En teckenbit måste behållas
    - När alla teckenbitar tas bort fås fel värde
  - Exempel:
    - $111010_{2C} = 1010_{2C}$
    - 00110_{2C} inte lika med 110_{2C}

Felaktig addition av 2-komplementstal

- Fungerar inte om resultat större än talområde hos indata
  - Måste utöka talområdet innan beräkning (teckenförlänga)
  - Går inte att bara lägga till carrybiten längst till vänster, eller teckenförlänga efter beräkning
- eller olika antal bitar i indata
  - Om summan får plats i samma antal bitar som ordlängden för termerna.
  - Fungerar bara om ordlängd lika

Sparar carrybit?
```
 0111  (7)
+1101  (-3)
--------
10100  (-12) FEL!
```

Kopierar teckenbit efteråt?
```
 0111  (7)
+0101  (5)
--------
 11100  (-4) FEL!
```

Olika ordlängd
```
  0111  (7)
+  101  (-3)
--------
  1100  (-4) FEL!
```


Korrekt addering av 2-komplementtal

- Om talområdet för litet måste indata teckenförlängas först
  - Summan av två tal kräver maximalt 1 bit extra för att kunna representeras
- Olika antal bitar i indata
  - Om summan får plats i samma antal bitar som ordlängden för termerna.
  - Fungerar bara om ordlängd lika

```
  00111  (7)
+00101  (5)
----------
  01100  (12) Rätt!
```

Olika ordlängd
```
  0111  (7)
+1101  (-3)
--------
  0100  (4) Rätt
```

Summan utanför talområdet (spill)?

- För binära positiva heltal fås en minnessiffra (carry) ut om talområdet överskrids (extra bit i resultatet)
- För 2-komplement är minnessiffran inte användbar
  - Har i tidigare slide visat även om den genereras så kan svaret vara rätt (den ignoreras)
  - Summering av positiva 2-komplementstal ger ingen minnessiffra även om talområdet överstigs
- Detektera för stort resultat i 2-komplement med två minnessiffror
  - För stort tal om minnessiffra in i teckenbitsaddition inte är likadan som minnessiffra ut från teckenbitsaddition

```
1 1 1   ← minnessiffror
  111  (7)
 +101  (5)
 ----------
 1100  (12)

olika 0 1 1 1  För stort!
  0111  (7)
 +0011  (3)
 --------
  1010  (-6) Fel!

lika  1 1 1 1
  0111  (7)
 +1011  (-5)
 --------
  0010  (2) Rätt!
```


Multiplikation av 2-komplementstal

- Nästan samma metod som multiplikation av positiva binära tal
  - Teckenförläng partialprodukter innan addition (dom är också tvåkomplementstal!)
  - Subtrahera teckenbitens bidrag (negativ vikt!)
- Två olika instruktioner i en processor som stödjer 2-komplements multiplikation respektive binär multiplikation

```
    101  (-3)
  * 110  (-2)
 ------------
  000000
 +11101
  -1101
 ---------------
  000110  (6)
```

Sammanfattning: Addition/subtraktion

- Operationen ger samma resultat (bitmönster) oberoende om indata är positiva heltal eller 2-komplementsform
  - Samma algoritm! Behöver inte veta om postiva heltal/tvåkomplement när addition utförs!
  - Tolkning av beräknat bitmönster beror på hur indata tolkas (val av 2-komplement eller positiva heltal)
- Detektering av spill (overflow) beror på talrepresentation
  - Minnessiffra (carry) om positiva heltal
  - Spill (olika minnessiffra in respektive ut från mest signfikant bit) vid 2-komplement
  - Tecknet på resultatet (MSB) fel om spill uppstått (för 2-komplement)

LINKÖPINGS UNIVERSITET