TSEA28 Datorteknik Y (och U)

Föreläsning 3

Kent Palmkvist, ISY

LINKÖPINGS UNIVERSITET

Dagens föreläsning

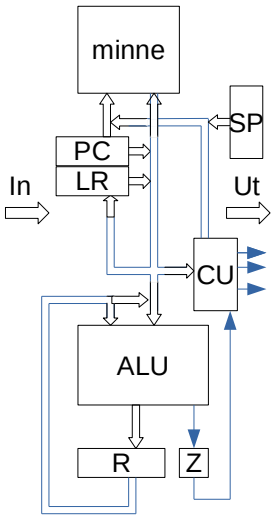
- Teckenrepresentation
- Talrepresentation
  - Binär aritmetik
  - 2-komplement
- Aritmetiska operation och flaggor

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 (22/1 kl 12.45)
- 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, push, pop
    - 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



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, ...

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 (bit position 0) 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 en symbol (skrivtecken)
  - 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: $7_{10}*11_{10} = 111_2*1011_2$

```
   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

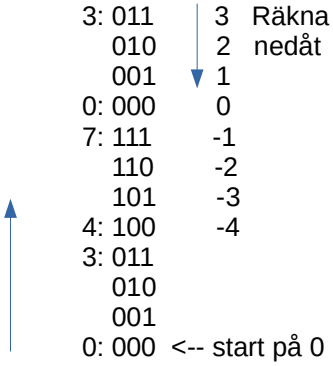
- Intuitiv lösning: Sätt en bit först som anger positivt (bit=0) eller negativt (bit=1)
  - Kallas för tecken-belopp representation (sign-magnitude)
  - Vänstraste biten 0: positivt tal, 1: negativt tal
  - Exempel (antag värden ska representeras med 4 bitar)
    - 7 = 0 111, -3 = 1 011
  - Dubbel representation 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
  - Kuriosa: IEEE 754 binary32 representation
    - Use 32 bits to represent max value of ≈ 3.402*10^38
    - Sign bit: 1 bit, exponent (E) 8 bits, 23 fractional bits
    - $(-1)^{b_{31}} \times 2\exp(E-127) \times (1+\sum_{i=1}^{23} b_{23-i} 2^{-i})$
  - Datatypen single i Matlab
  - Datatyperna float i C/C++
- Beskrivs närmare i kurser som numeriska metoder etc.

Negativa tal på rätt sätt (för hårdvara)

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

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)

Aritmetiska operationer och flaggor

Aritmetiska beräkningsinstruktioner (DARMA)

- Addition (ADD, ADDS, ADDC, ADDCS)
  - 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
  - Samma ordlängd på indata och utdata

Additions-instruktioner	C-flaggan som minnessiffra in	Utan att addera C-flaggan
Uppdaterar flaggor	ADDCS	ADDS
Uppdaterar inte flaggor	ADDC	ADD

Exempel: ADDS r0,r1 ; R0 = R0 + R1,
; uppdater flaggor

Aritmetiska beräkningsinstruktioner (DARMA)

- Subtraktion (SUB, SUBS, SBC, SBCS)
  - Ingen skillnad hur det går till (subtraktionsalgoritmen)
  - Använder samma instruktion för både positiva heltal och 2-komplement
  - Resultat tolkas olika beroende på talrepresentation
  - Samma bitmönster ut oberoende av indatats talrepresentation
  - Samma ordlängd på indata och utdata

Additions-instruktioner	C-flaggan som minnessiffra in	Utan att addera C-flaggan
Uppdaterar flaggor	ADDCS	ADDS
Uppdaterar inte flaggor	ADDC	ADD

Aritmetiska beräkningsinstruktioner (DARMA)

- Multiplikation (MUL, MULS)
  - Gör bara multiplication av positiva heltal, kastar mest signifikanta halvan av produkten
  - MUL påverkar inte flaggor, MULS påverkar flaggor
- Överkurs: SMULL, UMULL
  - 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
  - SMUL r0,r1,r2,r3 => r2*r3, placera mest signfikant halva i r1, minst signifikant halva i r0

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
  - CMP (compare) utför subtraktion och behåller bara flaggornas nya värde
- 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)

LINKÖPINGS UNIVERSITET