Förenklad förklaring i anslutning till kompedieavsnitten 6.3 och 6.4

Tecken-beloppsrepresentation av heltal

Hur skall man kunna räkna med negativa tal i ett digitalt system , t ex en dator.

Det enda man kan arbeta med är ju binära siffror, dvs 0 och 1.

En enkel metod vore ju att avdela en siffra för tecknet, t ex 0 för + (plus)
och 1 för - (minus) och använda resten av siffrorna för talets storlek (absolutbelopp).

Med 8 bitars ordlängd kan man då skriva t ex talet $+14 = +1110_2$ som

Teckenbit ⟶ 0 0001110 ⟵ Belopp

och talet -19 som

Teckenbit ⟶ 1 0010011 ⟵ Belopp

För 8-bitars tal blir talområdet $[-(2^{8-1}-1),+(2^{8-1}-1)] = [-127,+127]$.

Teckenbyte görs genom invertering av teckenbiten.

Tecken-beloppsrepresentation blir dock mycket besvärlig att använda i praktiken när man skall addera eller subtrahera. Se följande exempel.

Addition ↓

Exempel: (+7) + (+5) = + (7+5) = +12

(+7) + (-5) = + (7-5) = +2

(-7) + (+5) = - (7-5) = -2 ⟵ Subtraktion

(-7) + (-5) = - (7+5) = -12

Det som från början såg ut som en addition kan senare visa sig bli en subtraktion.
Dessutom måste man hålla reda på och bilda rätt tecken.

2-komplementrepresentation av heltal

En enkel lösning på problemet tal med tecken får man med 2-komplementrepresentation.

Denna innebär att man behåller positiva tal som de är. Negativa tal byts ut mot 2-komplementet (egentligen 2^n-komplementet, där n är antalet bitar i talet) av motsvarande positiva tal.

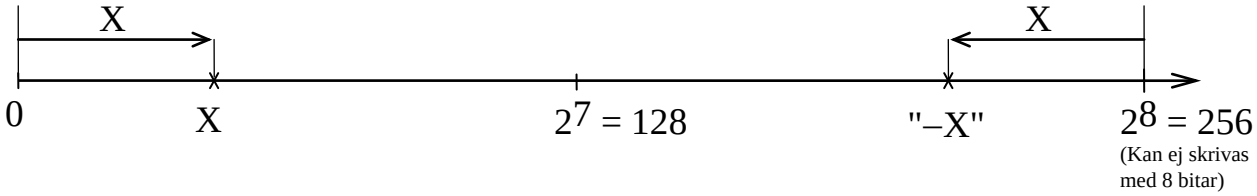
Exempel.
För 8-bitars tal gäller:

+5 används som det är dvs $(00000101)_2$

–5 byts mot $2^8 - 5 = 256 - 5 = (100000000)_2 - (00000101)_2 = (11111011)_2$

–38 byts mot $2^8 - 38 = 256 - 38 = (100000000)_2 - (00100110)_2 = (11011010)_2$

Princip: Om $X \geq 0$ används X självt för att representera talet.
Om $X < 0$ används $2^8 - |X|$ istället för X för att representera talet.



Ett åttabitars heltal kan anta ett värde i intervallet $[0,2^8-1] = [0,255]$. I figuren ovan har detta intervall visats på tallinjen.

I figuren är också talen X och motsvarande "negativa" tal "–X" utsatta.

Man ser att så länge $X < 256/2 = 128$, så kan man skilja X och "–X" åt genom att X finns i den vänstra halvan av det visade intervallet, medan "–X" finns till höger. Man begränsar därför talområdet för positiva tal till den vänstra halvan av intervallet. På detta sätt får varje positivt tal på den vänstra halvan en "negativ" motsvarighet på den högra halvan.

Samtliga talvärden X på den vänstra halvan (dvs "positiva tal") har den mest signifikanta biten $x_7 = 0$. På motsvarande sätt har samtliga talvärden X på den högra halvan (dvs "negativa tal") den mest signifikanta biten $x_7 = 1$. Biten längst till vänster kallas därför **teckenbit**.

För 8-bitars tal blir talområdet $[-2^{8-1},+(2^{8-1}-1)] = [-128,+127]$.
Talet 0 betraktas som positivt. Talet –128 har ingen positiv motsvarighet.

Teckenbyte görs genom 2-komplementering, dvs $X_{2k} = 2^8 - X$
(Fungerar ej för X = –128!)

$(X_{2k})_{2k} = 2^8 - (2^8 - X) = X$ (Teckenbyte två gånger ger det ursprungliga talet.)

Den stora fördelen med 2-komplementrepresentation är att man kan använda vanlig binär addition och subtraktion.

Exempel på addition med 8-bitars ordlängd:

Istället för –5 används 256 – 5 = 251 = 11111011_2

Istället för –7 används 256 – 7 = 249 = 11111001_2

(+7) + (+5) Utförs som: 7 + 5 = 12

```
           00001110 carry
Binärt:    00000111
         + 00000101
           00001100 = 12
```

(+7) + (–5) Utförs som: 7 + (256 – 5) = 256 + 2

```
           11111110 carry
Binärt:    00000111
         + 11111011
          100000010 = 256+2 (motsvarar 2 om minnes-
                             siffran ut stryks)
```

(–7) + (+5) Utförs som: (256 – 7) + 5 = 256 – 2

```
           000000010 carry
Binärt:    11111001
         + 00000101
           11111110 = 256-2 (motsvarar -2)
```

(–7) + (–5) Utförs som: (256 – 7) + (256 – 5) = 256 + 256 – 12

```
           11111110 carry
Binärt:    11111001
         + 11111011
          111110100 = 256 + 256-12 (motsvarar -12 om minnes-
                                    siffran ut stryks)
```

Subtraktion behandlas på motsvarande sätt. Operationen X – Y ersätts standardmässigt med operationen X + 256 – Y = X + (255 – Y) + 1 = X + Y_{1k} + 1.

Exempel på subtraktion med 8-bitars ordlängd:

Istället för –5 används 256 – 5 = 251 = 11111011_2

Istället för –7 används 256 – 7 = 249 = 11111001_2

5_{1k} = 255 – 5 = 250 = 11111010_2

251_{1k} = 255 – 251 = 4 = 00000100_2

(+7) – (+5) Utförs som: 7 + 5_{1k} +1 = 7 + 250 +1 = 256 + 2

```
          11111111 carry
Binärt:    00000111
         + 11111010
          100000010 = 256 + 2 (motsvarar 2 om minnes-
                               siffran ut stryks)
```

(+7) – (–5) Utförs som: 7 + $(256 – 5)_{1k}$ + 1 = 7 + 251_{1k} + 1 = 7 + 4 + 1 = 12

```
          000001111 carry
Binärt:    00000111
         + 00000100
           00001100 = 12
```

(–7) – (+5) Utförs som: (256 – 7) + 5_{1k} + 1 = 256 – 7 + 250 + 1 = 256 + 256 – 12

```
          11111110 carry
Binärt:    11111001
         + 11111010
          111110100 = 256 + 256-12 (motsvarar -12 om minnes-
                                    siffran ut stryks)
```

(–7) – (–5) Utförs som: (256 – 7) + $(256 – 5)_{1k}$ + 1 = 256 – 7 + 251_{1k} + 1 =
= 256 – 7 + 4 +1 = 256 – 2

```
          000000011 carry
Binärt:    11111001
         + 00000100
           11111110 = 256 - 2 (motsvarar -2)
```

Sammanfattning av addition och subtraktion med 2-komplementaritmetik

Vi konstaterar att vanlig binär addition och subtraktion kan användas även för tal med 2-komplementrepresentation. Det problem som kan inträffa kallas **overflow** och innebär att gränsen för det tillgängliga talområdet har överskridits.

Vid addition av två tal kan overflow inträffa om bägge talen är positiva eller om bägge talen är negativa, eftersom summans absolutbelopp då blir större än absolutbeloppet av det största av talen i additionen. Om summans absolutbelopp överskrider talområdets gräns, på den positiva eller negativa sidan, så inträffar overflow. Man upptäcker enkelt overflow genom att betrakta teckenbitarna hos de bägge talen och hos summan. Om man adderar två tal med lika tecken och summan får motsatt tecken, så har overflow inträffat. (Pos + Pos = Neg eller Neg + Neg = Pos)

Vid subtraktion av ett tal från ett annat tal kan overflow inträffa om det första talet i subtraktionen är positivt och det andra negativt eller tvärt om. Skillnadens absolutbelopp blir ju då större än det största talets absolutbelopp. Overflow har inträffat om subtraktion av ett negativt tal från ett positivt tal ger en negativ summa (Pos – Neg = Neg) eller om subtraktion av ett positivt tal från ett negativt ger en positiv summa (Neg – Pos = Pos)

Dessa iakttagelser leder oss till följande booleska uttryck för overflowfunktionen V för additionen S = X +Y eller subtraktionen S = X – Y, där X, Y och S är 8-bitars tal med 2-komplementrepresentation och med teckenbitarna x_7, y_7 och s_7:

$V(x_7,y_7,s_7) = (SUB)'(x_7'y_7's_7 + x_7y_7s_7') + SUB(x_7'y_7s_7 + x_7y_7's_7')$

Variablen SUB får värdet 1 vid subtraktion.

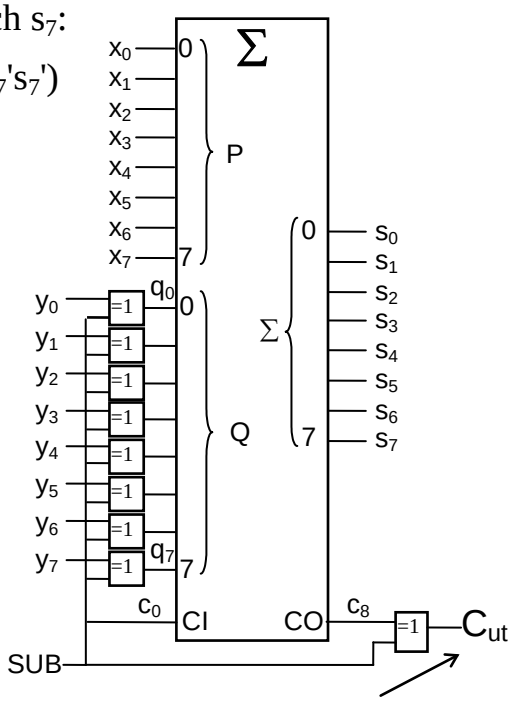
Subtraktionen X – Y utförs normalt med en adderare enligt standardmetoden $X + (2^8 – Y) = X + Y_{1k} + 1$, enligt figuren till höger. Detta innebär att talet Y finns på adderarens Q-ingång vid addition och talet Y_{1k} finns på Q-ingången vid subtraktion.

Vid addition är därför $q_7 = y_7$ och vid subtraktion är $q_7 = y_7'$.

Genom att uttrycka V som en funktion av x_7, q_7 och s_7 kan man få ett enklare uttryck.

$V = (SUB)'(x_7'q_7's_7 + x_7q_7s_7') + SUB(x_7'q_7's_7 + x_7q_7s_7') =$

$= [(SUB)' + SUB](x_7'q_7's_7 + x_7q_7s_7') = x_7'q_7's_7 + x_7q_7s_7'$



Dvs addition och subtraktion får samma overflowvillkor uttryckt i adderarens teckenbitar.