1.2 - Talsystem

1.2.1 - Introduktion

- Inom digitalteknik används olika så kallade talsystem för att representera tal. Inom varje talsystem så används en viss specifik talbas, som indikerar antalet olika siffror som ingår i talsystemet.

- Som exempel, i det talsystem som vi använder till vardags, som kallas det decimala talsystemet, används siffror 0 – 9, vilket är tio olika siffror. Därmed har det decimala talsystemet talbasen tio.

- I det decimala talsystemet, så beror varje siffras värde på dess position i talet. Som exempel, se talen 25 och 250 nedan. I talet 25, så femman värd fem och tvåan värd 20, som därefter summeras sedan till 25, då

$$5 + 20 = 25$$

- I talet 250, så är femman istället värd 50 och tvåan värd 200, vilket summeras till talet 250, då

$$50 + 200 = 250$$

- Vi ser därmed genom att lägga till en nolla längst bak i talet 250, så att tvåans och femmans respektive position i talet skiftar ett steg, så ökade deras respektive värde med tio. Detta gäller för alla tal i det decimala talsystemet, då talbasen tio används.

- Därmed gäller att alla decimala tal kan tänkas utgöra en funktion av talbasen tio. Ta talet 250 som exempel, som kan summeras via följande ekvation:

$$250 = 0 * 10^0 + 5 * 10^1 + 2 * 10^2,$$

vilket är ekvivalent med

$$250 = 0 + 5 * 10 + 2 * 100 = 0 + 50 + 200$$

- Notera att tiopotensen användes för att summera siffrorna i talet ovan, talbasen multiplicerades med 10^0, 10^1 samt 10^2, beroende på siffrans position i talet, där den minsta signifikanta siffran i talet, vilket är nollan, multiplicerades med $10^0 = 1$. Detta gäller för alla tal på denna position, då detta medför ett värde mellan 0 – 9. Därmed räknas positionen på den minst signifikanta siffran i ett tal som noll, inte ett, som kan verka logiskt vid en första anblick

- Faktum är att varje siffra för ett givet tal i ett givet talsystem kan uttryckas som en funktion ur dess talbas. Värdet y_n på en given siffra x_n i ett tal kan beräknas med formeln

$$y_n = x_n * B^n,$$

där B är talbasen och n är siffrans position i talet, som beräknas från höger till vänster med startvärdet noll.

- Faktorn B^n kallas siffrans vikt, då denna avgör vilket värde en given siffra får utifrån dess talbas. Exempelvis kan siffran 8 i ett decimalt tal inneha värdet 8, 80, 800 och så vidare, beroende på dess position i talet.

- Siffran längst till höger i ett flersiffrigt tal innehar alltid minst värde i ett tal, oavsett talsystem, och kallas därför LSD *(Least Significant Digit.* Värdet på LSD betecknas y_0, där 0:an indikerar att siffran är placerad på position noll.

- Vidare gäller att siffran som är placerad till vänster om LSD räknas vara placerad på position ett. Eventuella ytterligare siffror räknas därefter vara placerade på position två, tre, fyra och så vidare upp till talet mest signifikanta siffra, som kallas MSD *(Most Significant Digit).*

- Genom att summera samtliga siffrors respektive värde y_0 – y_{max} från LSD till MSD, så kan ett givet tals värde y_{TOT} beräknas med formeln

$$y_{TOT} = y_0 + y_1 + y_2 \dots + y_{max},$$

där y_{max} utgör värdet på talets mest signifikanta siffra (MSD).

- Formeln ovan sedan kan transformeras till

$$y_{TOT} = x_0 * B^0 + x_1 * B^1 + x_2 * B^2 \dots + x_{max} * B^{mqx},$$

där x_0 – x_{max} är respektive siffra i talet och B är talbasen.

- Eftersom positionen n är noll för den minst signifikanta siffran i ett tal (LSD), så blir dess vikt B^0 alltid ett, då

$$B^0 = 1,$$

vilket gäller i alla talsystem. För LSD i ett givet tal gäller därmed att dess värde y_0 är samma som dess siffervärde x_0, eftersom

$$y_0 = x_0 * B^0 = x_0 * 1 = x_0$$

Exempel 1:

- Som exempel, lås oss analysera det decimala talet 43. Trean utgör talets minst signifikanta siffra (LSD) och är därmed placerad på position noll:

$$x_0 = 3,$$

samtidigt som fyran är placerad på position ett och utgör talets mest signifikanta siffra (MSD):

$$x_1 = 4$$

- Talet 43 är lika med summan y_{TOT} av siffrornas respektive värde y_0 samt y_1:

$$y_{TOT} = y_0 + y_1,$$

som kan transformeras till

$$y_{TOT} = x_0 * B^0 + x_1 * B^1$$

- Eftersom talet 43 utgör ett tal i det decimala talsystemet, så är talbasen B lika med tio:

$$B = 10,$$

vilket medför att

$$y_{TOT} = x_0 * 10^0 + x_1 * 10^1$$

- Genom att sätta in siffror x_0 = 3 samt x_1 = 4, så kan talets värde y_{TOT} summeras till 43, då

$$y_{TOT} = 3 * 10^0 + 4 * 10^1,$$

vilket är ekvivalent med

$$y_{TOT} = 3 * 1 + 4 * 10,$$

som kan summeras till

$$y_{TOT} = 3 + 40 = 43$$

Exempel 2:

- I detta fall analyseras ett större tal, nämligen 791. Ettan utgör talets minst signifikanta siffra (LSD) och är därmed placerad på position noll:

$$x_0 = 1,$$

nian är placerad på position ett:

$$x_1 = 9$$

och sjuan, som utgör talets mest signifikanta siffra (MSD), är placerad på position två:

$$x_2 = 7$$

- Talet 791 är lika med summan y_{TOT} av siffrornas respektive värde $y_0 – y_2$:

$$y_{TOT} = y_0 + y_1 + y_2,$$

som kan transformeras till

$$y_{TOT} = x_0 * B^0 + x_1 * B^1 + x_2 * B^2,$$

där talbasen B är tio, då 791 utgör ett tal i det decimala talsystemet:

$$B = 10,$$

vilket medför att

$$y_{TOT} = x_0 * 10^0 + x_1 * 10^1 + x_2 * 10^2$$

- Genom att sätta in siffror $x_0 = 1$, $x_1 = 9$ samt $x_2 = 7$, så kan talets värde y_{TOT} summeras till 791, då

$$y_{TOT} = 1 * 10^0 + 9 * 10^1 + 7 * 10^2,$$

vilket är ekvivalent med

$$y_{TOT} = 1 * 1 + 9 * 10 + 7 * 10,$$

som kan summeras till

$$y_{TOT} = 1 + 90 + 700 = 791$$

1.2.2 – Det binära talsystemet

- Inom digitalteknik så kan signaler endast anta två tillstånd, hög eller låg. En hög signal indikeras med en etta (1), medan en låg signal indikeras med en nolla (0).

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

- Därmed så används ett talsystem med talbasen 2:

$$B = 2,$$

vilket innebär att varje siffra endast kan anta värden 0 - 1, istället för vårt vanliga talsystem, där varje siffra kan anta alla värden mellan 0–9.

- Det talsystem som används inom digitalteknik kallas därmed det binära talsystemet, eftersom varje tal endast kan anta två olika värden, vilket är 0 eller 1.

- Värdet y_n av en binär siffra x_n, kan därmed beräknas med formeln

$$y_n = x_n * 2^n,$$

där siffran x_n utgörs av 0 – 1 och n indikerar siffrans position i talet.

- Som vi såg tidigare, så gäller att talets minst signifikanta siffra (LSD) räknas ligga på position noll. Eventuella ytterligare siffror räknas därefter vara placerade på position ett, två, tre, fyra och så vidare upp till talet mest signifikanta siffra (MSD).

- Genom att summera samtliga siffrors respektive värde y_0 – y_{max} från LSD till MSD, så kan ett givet tals värde y_{TOT} beräknas med formeln

$$y_{TOT} = y_0 + y_1 + y_2 \dots + y_{max},$$

där y_{max} utgör värdet på talets mest signifikanta siffra (MSD).

- Formeln ovan kan transformeras till

$$y_{TOT} = x_0 * 2^0 + x_1 * 2^1 + x_2 * 2^2 \ldots + x_{max} * 2^{mqx},$$

där x_0 – x_{max} är respektive siffra i talet, 2 utgör talbasen och 0 – max utgör LSD – MSD.

- Även för binära tal, så gäller att den minst signifikanta siffrans vikt 2^0 alltid blir ett, då

$$2^0 = 1$$

- För att markera att ett tal är decimalt eller binärt, så kan man lägga till talbasen nedsänkt efter talet. För det decimala talet 15 skriver man då 15_{10}. Den binära motsvarigheten till 15 är 1111, som då skrivs 1111_2. Därmed så gäller att:

$$15_{10} = 1111_2$$

- En annan notation som används i det hårdvarubeskrivande språket Verilog är att antalet bitar samt talsystem deklareras framför talet. Antalet bitar skrivs med en siffra, exempelvis 2 för ett 2-bitars binärt tal, medan talsystemet skrivs med en eller två bokstäver, exempelvis b, d och h, som står för *binary, decimal* samt *hexadecimal.*

- Mellan antalet bitar samt talsystemet så används en apostrof, som fungerar som separator. Därmed skrivs det 4-bitars binära talet 1111_2 ovan på formen 2'b1111, samtidigt som motsvarande decimala tal 15_{10} skrivs med notationen 4'd15.

Exempel 3:

- Antag att vi skall översätta det 4-bitars binära talet 1011_2 till motsvarande decimaltal. Vi genomför beräkningen från den minst signifikanta biten (LSB) längst till höger, som är placerad på position noll, upp till den mest signifikanta biten (MSB) längst till vänster, som därmed är placerad på position tre.

- Därefter noterar vi vilka av siffrorna som är nollor och vilka som är ettor. Som synes, så utgörs talet 1011_2 av tre ettor samt en nolla, där ettorna är placerade på position 0 – 1 samt 3 och nollan är placerad på position 2. Därmed gäller att

$$x_0 = 1,$$

$$x_1 = 1,$$

$$x_2 = 0$$

samt

$$x_3 = 1$$

- För att beräkna den decimala motsvarigheten till det binära talet 1011_2, så beräknar vi summan y_{TOT} av siffrornas respektive värde y_0 – y_3:

$$y_{TOT} = y_0 + y_1 + y_2 + y_3,$$

som kan transformeras till

$$y_{TOT} = x_0 * 2^0 + x_1 * 2^1 + x_2 * 2^2 + x_3 * 2^3$$

- Genom att sätta in siffror $x_0 = 1$, $x_1 = 1$, $x_2 = 0$ samt $x_3 = 1$, så kan talets värde y_{TOT} summeras till det decimala talet 11_{10}, då

$$y_{TOT} = 1 * 2^0 + 1 * 2^1 + 0 * 2^2 + 1 * 2^3,$$

vilket är ekvivalent med

$$y_{TOT} = 1 * 1 + 1 * 2 + 0 + 1 * 8,$$

som kan summeras till

$$y_{TOT} = 1 + 2 + 8 = 11_{10}$$

Exempel 4:

- Antag att vi istället skall översätta det 8-bitars binära talet $1010\ 1110_2$ till motsvarande decimaltal. Eftersom talet består av åtta bitar, så räknar deras dessa vara placerade på position 0 – 7.

- Återigen genomförs beräkningen från talets minst signifikanta siffra (LSD) längst till höger, som är placerad på position 0, upp till talets mest signifikanta siffra (MSD) längst till vänster, som i detta fall är placerad på position 7.

- Därefter noterar vi vilka av siffrorna som är nollor och vilka som är ettor. Vi noterar då att talet $1010\ 1110_2$ utgörs av fem ettor samt tre nollor, där ettorna är placerade på position 1 – 3, 5 och 7, samtidigt som nollorna är placerade på position 0, 4 och 6. Därmed gäller att

$$x_0 = 0,$$

$$x_1 = 1,$$

$$x_2 = 1$$

$$x_3 = 1,$$

$$x_4 = 0,$$

$$x_5 = 1,$$

$$x_6 = 0$$

samt

$$x_7 = 1$$

- Därefter beräknas summan y_{TOT} av siffrornas respektive värde $y_0 – y_7$:

$$y_{TOT} = y_0 + y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7,$$

som kan transformeras till

$$y_{TOT} = x_0 * 2^0 + x_1 * 2^1 + x_2 * 2^2 + x_3 * 2^3 + x_4 * 2^4 + x_5 * 2^5 + x_6 * 2^6 + x_6 * 2^7$$

- Genom att sätta in siffror för $x_0 – x_7$, så kan talets värde y_{TOT} summeras till det decimala talet 174_{10}, då

$$y_{TOT} = 0 * 2^0 + 1 * 2^1 + 1 * 2^2 + 1 * 2^3 + 0 * 2^4 + 1 * 2^5 + 0 * 2^6 + 1 * 2^7,$$

vilket är ekvivalent med

$$y_{TOT} = 0 + 1 * 2 + 1 * 4 + 1 * 8 + 0 + 1 * 32 + 0 + 1 * 128,$$

som kan summeras till

$$y_{TOT} = 2 + 4 + 8 + 32 + 128 = 174_{10}$$

- Beräkningarna ovan hade kunnat förenklas genom att direkt försumma samtliga nollor i beräkningen och endast beräknat de bitar som är ettor. Vi såg tidigare att ettorna i det binära talet $1010\ 1110_2$ är placerade på position 1 – 3, 5 och 7, vilket innebär att

$$y_{TOT} = 1 * 2^1 + 1 * 2^2 + 1 * 2^3 + 1 * 2^5 + 1 * 2^7$$

- Eftersom ovanstående bitar är lika med ett, så talet kunnat beräknas enbart via respektive tals vikt 2^n:

$$y_{TOT} = 2^1 + 2^2 + 2^3 + 2^5 + 2^7,$$

som kan summeras till

$$y_{TOT} = 2 + 4 + 8 + 32 + 128 = 174_{10}$$

Förenklade beräkningar av binära tal:

- Som vi har sett tidigare så medför varje tillsatt siffra i det decimala talsystemet att en given siffras värde ökar med en faktor tio. Som exempel, antag att vi har ett decimalt tal 1. Genom att placera en nolla efter talet 1, så blir denna etta värd tio.

- Binära tal blir istället två gånger större för varje tillsatt siffra; de binära talen 1, 10, 100, 1000, 1000 0, 1000 00, 1000 000 samt 1000 0000 motsvarar alltså de decimala talen 1, 2, 4, 8, 16, 32, 64 och 128 för ett 8-bitars binärt tal.

- Som vi såg tidigare, så kan ett binärt tal beräknas via vikten 2^n på de bitar som är lika med ett. Som exempel, det binära talet $0101\ 0101_2$, där ettor förekommer på position 0, 2, 4 och 6, kan beräknas med formeln

$$y_{TOT} = 1 * 2^0 + 1 * 2^2 + 1 * 2^4 + 1 * 2^6,$$

som kan transformeras till

$$y_{TOT} = 1 * (2^0 + 2^2 + 2^4 + 2^6),$$

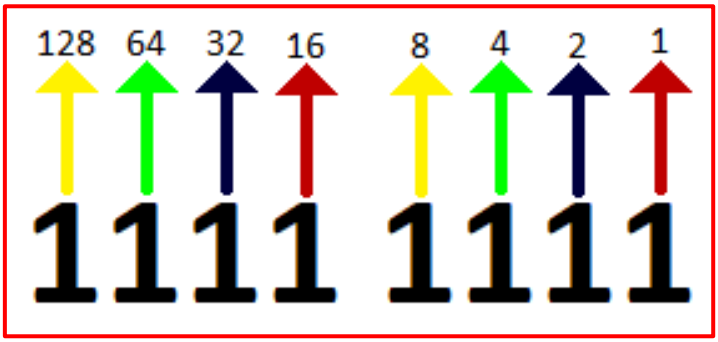
vilket är ekvivalent med

$$y_{TOT} = 2^0 + 2^2 + 2^4 + 2^6,$$

som kan summeras till

$$y_{TOT} = 1 + 4 + 16 + 64 = 85_{10}$$

- Därmed kan följande figur ritas upp, indikerar hur mycket varje bit 0 - 7 är värd, då det utgörs av en etta:



Ett 8-bitars binärt tal, som motsvarar det decimala talet 255. I figuren så indikeras vilken decimal motsvarighet varje decimalt tal har.

- Genom att använda figuren ovan, så hade det binära talet $0101\ 0101_2$ kunnat beräknas direkt genom att titta i figuren och summera de bitar som utgörs av ettor, vilket är ettorna som är värda 1, 4, 16 och 64. Därmed gäller att

$$y_{TOT} = 1 + 4 + 16 + 64 = 85_{10}$$

- Det binära talet 1111 1111 ovan är ett exempel på ett 8-bitars binärt tal som endast innehåller ettor. Den decimala motsvarigheten till detta tal är 255, vilket enkelt kan beräknas via figuren ovan. Vi beräknar från höger till vänster:

$$1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = 255$$

- För att tydliggöra vilket talsystem det handlar om så skriver vi också vilken talbas det handlar om under talen:

$$1111\ 1111_2 = 255_{10}$$

eller via notationen i Verilog:

$$8b'11111111 = 8d'255$$

- Ett 8-bitars binärt tal kan anta värden som motsvarar 0–255 i decimala tal, alltså 256 olika värden, där 0 är minvärdet och 255 är maxvärdet. Minvärdet är alltid 0.

- För att beräkna antalet kombinationer z som ett binärt tal med ett visst antal bitar kan anta, så räknas detta enkelt ut med formeln

$$z = 2^n,$$

där n är antalet bitar.

- Därmed gäller att ett 4-bitars tal kan anta 16 olika kombinationer 0000_2 – 1111_2, då

$$z = 2^4 = 16$$

- Notera att det största värdet $y_{TOT,max}$ som kan erhållas ur dessa fyra bitar är 1111_2, vilket motsvarar det decimala talet 15_{10}, då

$$y_{TOT,max} = 1 * 2^0 + 1 * 2^1 + 1 * 2^2 + 1 * 2^3,$$

vilket är ekvivalent med

$$y_{TOT,max} = 2^0 + 2^1 + 2^2 + 2^3,$$

som kan summeras till

$$y_{TOT,max} = 1 + 2 + 4 + 8 = 15_{10}$$

- Därmed ser vi att det största värdet $y_{TOT,max}$ som kan erhållas ur dessa fyra bitar är lika med antalet kombinationer z – 1, då

$$y_{TOT,max} = z - 1 = 2^4 - 1 = 15$$

- Därmed motsvarar de binära kombinationerna 0000_2 – 1111_2 de decimala talen 0_{10} -15_{10}. För enkelhets skull kan dessa kombinationer anges som de binära kombinationerna 0 – 15.

- Samma princip gäller för exempelvis ett 8-bitars tal, som kan anta 256 olika kombinationer 0000 0000_2 – 1111 1111_2, då

$$z = 2^8 = 256,$$

vilket innebär ett maxvärde $y_{TOT,max}$ på 255, då

$$y_{TOT,max} = z - 1 = 2^8 - 1 = 255$$

- Vi kan översätta det binära talet 1111 1111_2 till sin decimala motsvarighet för att verifiera detta:

$$y_{TOT,max} = 1 * 2^0 + 1 * 2^1 + 1 * 2^2 + 1 * 2^3 + 1 * 2^4 + 1 * 2^5 + 1 * 2^6 + 1 * 2^7,$$

vilket är ekvivalent med

$$y_{TOT,max} = 2^0 + 1 * 2^1 + 2^2 + 2^3 + 1 * 2^4 + 2^5 + 2^6 + 2^7$$

som kan summeras till

$$y_{TOT,max} = 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = 255_{10}$$

- Därmed motsvarar de binära kombinationerna 0000 0000_2 – 11111111_2 de decimala talen 0_{10} -255_{10}, vilket därför kan anges som de binära kombinationerna 0 – 255.

- Därmed gäller att för ett n-bitars binärt tal, så finns det 2^n olika kombinationer mellan 0 – (2^n – 1). För ett 2-bitars tal så finns det därmed $2^2 = 4$ olika kombinationer mellan 0 – (2^2 – 1) = 0 – 3.

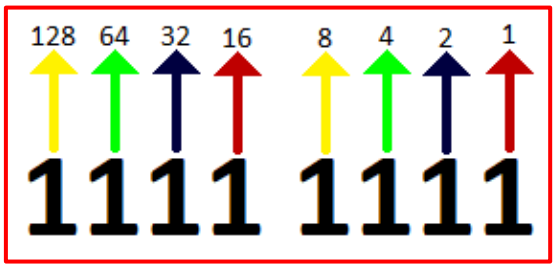
- För ett 3-bitars tal så finns det $2^3 = 8$ olika kombinationer mellan 0 – (2^3 – 1) = 0 – 7.

- För ett 16-bitars tal så finns det $2^{16} = 65\,536$ olika kombinationer mellan 0 – (2^{16} – 1) = 0 – 65 535.

- För ett 16-bitars tal så kan man få $2^{16} = 65\,536$ olika värden och maxvärdet är 2^{16} – 1 = 65 535.

Översättning från decimal till binär form

- Låt oss säga att vi istället vill översätta det decimala talet 23_{10} till ett binärt tal. För att beräkna detta tal så måste vi ta reda på vilken binär kombination vi behöver. Vi skriver därför upp en tabell med de åtta första binära bitarna, se figuren nedan.



Figur som visar hur mycket varje binär etta i ett 8-bitars binärt tal motsvarar i det decimala talsystemet.

- Via figuren ovan, så ser vi att de första binära första binära bitarna motsvarar de decimala talen 1, 2, 4, 8, 16, 32, 64 och 128, eftersom

$$1 * 2^0 = 1,$$

$$1 * 2^1 = 2,$$

$$1 * 2^2 = 4,$$

$$1 * 2^3 = 8,$$

$$1 * 2^4 = 16,$$

$$1 * 2^5 = 32,$$

$$1 * 2^6 = 64,$$

samt

$$1 * 2^7 = 128$$

Metod 1 – Omvandling via tabell:

- Via talen ovan så adderar vi de bitar som behövs för att generera talet 23, vilket är 16, 4, 2 samt 1:

$$23_{10} = 16 + 4 + 2 + 1,$$

vilket är ekvivalent med $1 * 2^4 + 1* 2^2 + 1 * 2^1 + 1 * 2^0$:

$$23_{10} = 1 * 2^0 + 1 * 2^1 + 1 * 2^2 + 1 * 2^4$$

- Därmed är bitar 5 – 7 överflödiga och kan strykas. Den binära motsvarigheten till det decimala talet 23_{10} utgörs därmed av ett 5-bitars binärt tal, där bitarna är placerade på position 0 – 4.

- Genom att addera bidraget av de bitar som utgörs av nollor, vilket i detta fall endast är $0 * 2^3$ från bit 3, så erhålls ekvationen

$$23_{10} = 1 * 2^0 + 1 * 2^1 + 1 * 2^2 + 0 * 2^3 + 1 * 2^4,$$

där $1 * 2^0$ samt $1 * 2^4$ utgör minst respektive mest signifikanta bit (LSB samt MSB).

- Eftersom bitarna skall skrivas från störst till minst, så skall den mest signifikanta biten (MSB = 1 * 2^4) placeras längst till vänster och den minst signifikanta biten (LSB = 1 * 2^0) längst till höger.

- Därmed kan formeln ovan transformeras till

$$23_{10} = 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0,$$

vilket kan översättas till det binära talet 10111_2. Därmed gäller att

$$23_{10} = 10111_2$$

Metod 2 – Division med 2:

- Alternativt kan beräkningen genomföras genom att vi genomför ett flertal divisioner med 2 och antecknar heltalskvoten samt resten från varje division.

- I den första divisionen så divideras det decimala talet 23_{10} med 2. Därefter är det heltalskvoten från varje division som divideras med 2.

- När kvarvarande heltalskvot är 0, så är beräkningen färdig och resten från varje division utgör en bit i det binära talet, där den första beräknade resten utgör LSB och den sista utgör MSB.

- Därmed börjar vi med att det decimala talet 23_{10} divideras med 2, vilket medför en heltalskvot på 11 samt en rest på 1:

$$\frac{23}{2} = 11, \qquad rest\ 1$$

- Resten från denna division utgör den minst signifikanta biten LSB på den binära motsvarigheten till talet 23_{10}.
Därmed ser vi ovan att LSB / bit 0 är lika med 1.

- För att erhålla nästa bit, så divideras därefter kvoten från förra divisionen, alltså 11, med 2:

$$\frac{11}{2} = 5, \qquad rest\ 1$$

- Därmed erhålls en kvot på 5 samt en rest på 1, vilket innebär att bit 1 också är lika med 1.

- Därefter dividerar vi kvoten från ovanstående division, vilket är 5, med 2:

$$\frac{5}{2} = 2, \qquad rest\ 1$$

- Därmed erhålls en kvot på 2 samt en rest på 1, vilket innebär att även bit 2 är lika med 1.

- Därefter dividerar vi kvoten från ovanstående division, vilket är 2, med 2:

$$\frac{2}{2} = 1, \qquad rest\ 0$$

- I detta fall så erhålls en kvot på 1 samt en rest på 0, vilket innebär att bit 3 är lika med 0.

- Eftersom ovanstående kvot är lika med 1, så kommer följande division vara den sista. Avslutningsvis dividerar vi därför kvoten 1 med 2:

$$\frac{1}{2} = 0, \qquad rest\ 1$$

- I detta fall så erhålls en kvot på 0 samt en rest på 1, vilket innebär att bit 4 är lika med 1.

- Eftersom kvarvarande rest är noll, så är omvandlingen klar. Vi ser då att bit 4 utgör den mest signifikanta biten (MSB). Genom att sätta samman bitarna från MSB till LSB, så erhålls det binära talet 10111_2. Därmed gäller att

$$23_{10} = 10111_2$$

1.2.3 – Hexadecimala talsystemet

- Ett problem med det binära talsystemet är att talen lätt blir svårläsliga. Som exempel såg vi tidigare att det krävs åtta bitar för att skriva ut det decimala talet 255_{10} binärt (till $1111\ 1111_2$). För ännu högre tal så krävs ännu fler bitar. Därför används ett annat talsystem, det hexadecimala talsystemet, för att representera binära tal i moderna datorer.

- Det hexadecimala talsystemets talbas B är 16:

$$B = 16,$$

vilket innebär att varje siffra endast kan anta värden 0_{16} – F_{16}, vilket motsvarar talen 0_{10} – 15_{10} i det decimala talsystemet, där varje siffra kan anta alla värden mellan 0–9.

- Att just talbasen 16 används är för att detta medför att en hexadecimal siffra då motsvarar fyra binära siffror, se figuren nedan. Därmed minskar antalet siffror som behövs för att representera ett givet tal med en faktor fyra. Samtidigt blir omvandlingen mellan binära och hexadecimala talsystemet mycket enkelt.

Decimalt tal	Binärt tal	Hexadecimalt tal
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

- Innan vi går igenom omvandling mellan olika talsystem, så skall det hexadecimala talsystemet uppbyggnad presenteras.

- Att det hexadecimala talsystemet har talbasen 16 innebär att en hexadecimal siffra kan anta ett värde som motsvarar 0_{10} – 15_{10} i det decimala talsystemet. Det hexadecimala talsystemet är identiskt med det decimala talsystemet för siffror 0_{10} – 9_{10}. För talen som motsvarar 10_{10} – 15_{10}, så används istället bokstäverna A – F, där

$$A_{16} = 10_{10},$$

$$B_{16} = 11_{10},$$

$$C_{16} = 12_{10},$$

$$D_{16} = 13_{10},$$

$$E_{16} = 14_{10}$$

samt

$$F_{16} = 15_{10}$$

Exempel 8:

- Antag att det 8-bitars binära talet $1011\ 0110_2$ skall omvandlas till dess hexadecimala motsvarighet. För att göra detta så delar vi in de binära bitarna i grupper om fyra, som därefter omvandlas till motsvarande decimala och därigenom hexadecimala siffra.

- Vi börjar med de fyra mest signifikanta bitarna 1011_2, som kan omvandlas på samma sätt som föregående avsnitt om binära tal. Vi ser då att det binära talet 1011_2 motsvarar det hexadecimala talet $B_{16,}$ då

$$y_{TOT} = 1 * 2^0 + 1 * 2^1 + 0 * 2^2 + 1 * 2^3,$$

vilket är ekvivalent med det decimala talet 11_{10}, då

$$y_{TOT} = 1 + 2 + 0 + 8 = 11_{10},$$

som i sin tur motsvarar det hexadecimala talet B_{16}:

$$y_{TOT} = B_{16}$$

- Därefter omvandlas de fyra minst signifikanta bitarna 0110_2 till motsvarande hexadecimala tal. Vi ser då att dessa bitar motsvarar det hexadecimala talet 6_{16}, då

$$y_{TOT} = 0 * 2^0 + 1 * 2^1 + 1 * 2^2 + 0 * 2^3,$$

vilket är ekvivalent med det decimala talet 6_{10}, då

$$y_{TOT} = 0 + 2 + 4 + 0 = 6_{10},$$

som är ekvivalent med det hexadecimala talet 6_{16}:

$$y_{TOT} = 6_{16}$$

- Därefter kan den hexadecimala motsvarigheten till det binära talet $1011\ 0110_2$ genom att sätta ihop de hexadecimala siffror som fastställdes, vilket blir $B6_{16}$. Därmed gäller att

$$10110110_2 = B6_{16}$$

Exempel 9:

- Antag istället att det hexadecimala talet $4A8C_{16}$ skall omvandlas till dess binära motsvarighet. För att göra detta så omvandlar vi varenda hexadecimal siffra till dess 4-bitars binära motsvarighet.

1. Vi börjar med den mest signifikanta siffran 4_{16}, som motsvarar det decimala talet 4_{10}. Denna siffra motsvarar det binära talet 0100_2, då

$$y_{TOT} = 0 * 2^0 + 0 * 2^1 + 1 * 2^2 + 0 * 2^3,$$

som är ekvivalent med

$$y_{TOT} = 0 + 0 + 4 + 0 = 4_{10} = 4_{16}$$

- Därmed gäller att den hexadecimala siffran 4_{16} motsvarar det binära talet 0100_2:

$$4_{16} = 0100_2,$$

vilket innebär att den första siffrans binära motsvarighet har fastställts.

2. Därefter omvandlar vi den hexadecimala siffran A_{16}, som motsvarar det decimala talet 10_{10}. Vi omvandlar därmed 10_{10} till motsvarande binära tal, vilket är 1010_2, då

$$y_{TOT} = 0 * 2^0 + 1 * 2^1 + 0 * 2^2 + 1 * 2^3,$$

vilket är ekvivalent med

$$y_{TOT} = 0 + 2 + 0 + 8 = 10_{10} = A_{16}$$

- Därmed gäller att den hexadecimala siffran A_{16} motsvarar det binära talet 1010_2:

$$A_{16} = 1010_2,$$

vilket innebär att den andra siffrans binära motsvarighet har fastställts.

3. Därefter omvandlas den hexadecimala siffran 8_{16}, som motsvarar det decimala talet 8_{10}, vars binära motsvarighet är 1000_2, då

$$y_{TOT} = 0 * 2^0 + 0 * 2^1 + 0 * 2^2 + 1 * 2^3,$$

vilket är ekvivalent med

$$y_{TOT} = 0 + 0 + 0 + 8 = 8_{10} = 8_{16}$$

- Därmed gäller att den hexadecimala siffran 8_{16} motsvarar det binära talet 1000_2:

$$8_{16} = 1000_2,$$

vilket innebär att den tredje siffrans binära motsvarighet har fastställts.

4. Slutligen omvandlas den hexadecimala siffran C_{16}, som motsvarar det decimala talet 13_{10}, vars binära motsvarighet är 1101_2, då

$$y_{TOT} = 1 * 2^0 + 0 * 2^1 + 1 * 2^2 + 1 * 2^3,$$

vilket är ekvivalent med

$$y_{TOT} = 1 + 0 + 4 + 8 = 13_{10} = C_{16}$$

- Därmed gäller att den hexadecimala siffran C_{16} motsvarar det binära talet 1101_2:

$$C_{16} = 1101_2,$$

vilket innebär att den fjärde siffrans binära motsvarighet har fastställts.

5. Genom att sätta ihop de tidigare fastställda binära motsvarigheterna, så ser vi att

$$4A8C_{16} = 0100\ 1010\ 1000\ 1101_2$$

- Det hexadecimala talsystemet medför därmed att det krävs fyra gånger färre bitar för att representera ett tal. Som ett praktiskt exempel gällande det hårdvarubeskrivande språket Verilog, så hade vi kunnat representera den binära kombinationen ovan antingen genom att skriva 16'b0100101010001101, vilket betyder att det är 16-bitars binärt tal, vars innehåll är $0100\ 1010\ 1000\ 1101_2$. Som synes blir detta lite otympligt och risken finns att användaren råkar mata in fel värde.

- Ett bättre alternativ hade varit att representera talet via dess hexadecimala motsvarighet, vilket är 16'h4A8C, som betyder att talet utgörs av ett 16-bitars hexadecimalt tal med värdet $4A8C_{16}$. Notera att denna metod är generellt sett mycket smidigare och risken för att fel värde matas in kan antas vara lägre.

Omvandling från det hexadecimala till det decimala talsystemet:

- Som vi har sett tidigare så gäller det att varje siffra för ett givet tal i ett givet talsystem kan uttryckas som en funktion ur dess talbas. Värdet y_n på en given siffra x_n i ett tal kan beräknas med formeln

$$y_n = x_n * B^n,$$

där B är talbasen och n är siffrans position i talet, som beräknas från höger till vänster med startvärdet noll.

- För hexadecimala tal, där talbasen B är 16, kan därmed värdet y_n av en hexadecimal siffra x_n beräknas med formeln

$$y_n = x_n * 16^n,$$

där siffran x_n utgörs av 0 – F, 16 utgör talbasen och n indikerar siffrans position i talet.

- Som vi såg tidigare, så gäller att talets minst signifikanta siffra (LSD) räknas ligga på position noll. Eventuella ytterligare siffror räknas därefter vara placerade på position ett, två, tre, fyra och så vidare upp till talet mest signifikanta siffra (MSD).

- Genom att summera samtliga siffrors respektive värde y_0 – y_{max} från LSD till MSD, så kan ett givet tals värde y_{TOT} beräknas med formeln

$$y_{TOT} = y_0 + y_1 + y_2 \dots + y_{max},$$

där y_{max} utgör värdet på talets mest signifikanta siffra (MSD).

- Formeln ovan kan transformeras till

$$y_{TOT} = x_0 * 16^0 + x_1 * 16^1 + x_2 * 16^2 \dots + x_{max} * 16^{mqx},$$

där x_0 – x_{max} är respektive siffra i talet, 16 utgör talbasen och 0 – max utgör LSD – MSD.

- Precis som tidigare så gäller att den minst signifikanta siffrans vikt 16^0 alltid blir ett, då

$$16^0 = 1$$

- För att markera att ett tal är hexadecimalt kan man lägga till talbasen 16 nedsänkt efter talet. Som exempel, den hexadecimala motsvarigheten till talet 15_{10} skrivs därmed ut i form av A_{16}:

$$15_{10} = A_{16}$$

- Som vi såg tidigare så krävs fyra bitar för att generera talet 15_{10}, då

$$15_{10} = 1111_2$$

- I det hårdvarubeskrivande språket Verilog används därmed notationen 4’hA, vilket betyder att talet utgörs av ett 4-bitars hexadecimalt tal med värdet A_{16}, vilket motsvarar det binära talet 4’b1111 samt det decimala talet 4’d15.

Exempel 10:

- Antag att vi skall omvandla det hexadecimala talet $F6_{16}$ till dess decimala motsvarighet. Sexan utgör talets minst signifikanta siffra (LSD) och är därmed placerad på position noll:

$$x_0 = 6,$$

samtidigt som F:et är placerad på position ett och utgör talets mest signifikanta siffra (MSD):

$$x_1 = F$$

- Talet $F6_{17}$ är lika med summan y_{TOT} av siffrornas respektive värde y_0 samt y_1:

$$y_{TOT} = y_0 + y_1,$$

som kan transformeras till

$$y_{TOT} = x_0 * B^0 + x_1 * B^1,$$

där talbasen B är 16:

$$B = 16,$$

vilket innebär att

$$y_{TOT} = x_0 * 16^0 + x_1 * 16^1$$

- Genom att sätta in siffror $x_0 = 6$ samt $x_1 = F$, så kan talets värde y_{TOT} summeras till 43, då

$$y_{TOT} = 6 * 16^0 + F * 16^1,$$

vilket är ekvivalent med

$$y_{TOT} = 6 * 1 + F * 16 = 6 + 16F,$$

där F_{16} motsvarar 15_{10} i det decimala talsystemet:

$$F_{16} = 16_{10}$$

- Därmed kan det hexadecimala talet $F6_{16}$ översättas till 246_{10}, då

$$y_{TOT} = 6 + 16 * 15 = 6 + 240 = 246_{10}$$

1.2.4 - 2-komplementsrepresentation

- Som vi har sett tidigare, så kan ett 8-bitars binärt tal anta samtliga värden mellan $0000\ 0000_2$ – $1111\ 1111_2$, vilket motsvarar alla decimaltal mellan 0_{10} – 255_{10}. Totalt rör det sig alltså om 256 kombinationer, som alla består av tal överstiger eller är lika med noll.

- Vid behov kan dock hälften av dessa kombinationer allokeras till att representera negativa tal. Detta medför då 128 negativa tal samt 128 positiva tal (inklusive noll) kan representeras. Detta kan genomföras via 2-komplementsrepresentation, där samtliga binära tal vars mest signifikanta bit (MSB) utgörs av en etta, representerar ett negativt tal.

- Därmed så kommer de binära talen $1000\ 0000_2$ – $1111\ 1111_2$ representera var sitt negativt tal, vilket då motsvarar -128_{10} – (-1_{10}).

- Samtidigt kommer de binära tal vars mest signifikanta bit (MSB) utgörs av en nolla representera ett positivt tal. Därmed så kommer de binära talen $0000\ 0000_2$ – $0111\ 1111_2$, vilket motsvarar 0_{10} – 127_{10}, användas för positiva tal.

- Följande förhållande gäller mellan ett negativt decimalt tal x_{10} samt motsvarande 2-komplement y_2:

$$y_2 = x_{10} + 2^n,$$

  där n är antalet bitar som talet skall omvandlas till.

- För ett 8-bitars tal, så adderas därför $2^8 = 256$ till det negativa decimala talet, vilket innebär att

$$y_2 = x_{10} + 256,$$

  där y_2 är 2-komplementet till det negativa decimala talet x_{10}.

- 2-komplement medför därmed att ett 8-bitars tal kan anta alla värden mellan -128 upp till 127, istället för 0 upp till 255. Därmed minskar givetvis hur stort värde ett 8-bitars binärt tal kostar, till bekostnad av de negativa talen.

- Variabler som kan anta både positiva och negativa värden kallas signerade variabler, medan variabler som inte kan anta negativa värden kallas osignerade variabler. De flesta variabler vi har sett hittills har alltså utgjort av osignerade variabler.

Exempel 5:

- Som exempel, om man skulle vilja uttrycka det decimala talet -1_{10} som ett 8-bitars binärt tal, så hade vi alltså behövt addera $2^8 = 256$, vilket resulterar i det decimala talet 255_{10}, då

$$y_2 = x_{10} + 2^n = -1 + 2^8 = 255_{10}$$

- Därefter omvandlas 255_{10} till motsvarande binära tal, vilket är $1111\ 1111_2$, då

$$y_{TOT} = 1 * 2^0 + 1 * 2^1 + 1 * 2^2 + 1 * 2^3 + 1 * 2^4 + 1 * 2^5 + 1 * 2^6 + 1 * 2^7,$$

vilket är ekvivalent med

$$y_{TOT} = 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = 255_{10}$$

- Därmed gäller att det 8-bitars 2-komplementet till det decimala talet -1_{10} är $1111\ 1111_2$:

$$-1_{10} = 1111\ 1111_2$$

Exempel 6:

Anta att det decimala talet -22_{10} skall konverteras till ett 6-bitars binärt tal.

- I detta fall behöver vi addera $2^6 = 64$ till det negativa talet -22_{10}, vilket resulterar i det decimala talet 42_{10}, då

$$y_2 = x_{10} + 2^n = -22 + 2^6 = 42_{10}$$

- Detta tal kan sedan omvandlas till ett 6-bitars binärt tal genom att används bitar 32, 8 och 2, då

$$y_{TOT} = 32 + 8 + 2,$$

där $32 = 2^5$, $8 = 2^3$ samt 2^1, vilket innebär att

$$y_{TOT} = 1 * 2^5 + 1 * 2^3 + 1 * 2^1$$

- Vi lägger sedan till nollor för övriga bitar, vilket medför i formeln:

$$y_{TOT} = 1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0,$$

- Genom att eliminera vikten 2^n samt plustecknen, så kan det binära talet erhållas, vilket är 101010. Därmed gäller att

$$-22_{10} = 101010_2$$

Exempel 7:

- Antag att det binära talet $1000\ 0111_2$ är signerat, alltså att detta tal kan vara både positivt och negativt. Eftersom den mest signifikanta biten (MSB) utgörs av en etta, så är talet negativt. För att ta reda på vilket negativt decimalt tal detta motsvarar, så kan vi använda formeln

$$y_2 = x_{10} + 2^n,$$

där y_2 är det signerade binära talet, x_{10} är motsvarande negativa tal och n är antalet bitar som talet skall omvandlas till.

- Formeln ovan kan omvandlas till

$$x_{10} = y_2 - 2^n$$

- Eftersom det signerade binära talet består av åtta bitar, så skall vi subtrahera med $2^8 = 256$ för att erhålla motsvarande negativa tal x_{10}. Innan vi gör detta måste vi dock omvandla det signerade talet till dess osignerade decimala motsvarighet.

- Vi ser då att det binära talet $1000\ 0111_2$ i osignerad form motsvarar det decimala talet 135_{10}, då

$$y_{TOT} = 1 * 2^0 + 1 * 2^1 + 1 * 2^2 + 0 * 2^3 + 0 * 2^4 + 0 * 2^5 + 0 * 2^6 + 1 * 2^7,$$

  vilket är ekvivalent med

$$y_{TOT} = 1 + 2 + 4 + 128 = 135_{10}$$

- Därmed ser vi att det 8-bitars signerade binära talet $1000\ 0111_2$ motsvarar talet -121_{10}, då

$$x_{10} = y_2 - 2^n = 135 - 2^8 = -121_{10}$$

2-komplementsrepresentation och minnesallokering inom programmering:

- Signering sätter en begränsning på hur stort ett visst tal kan bli med ett visst antal bitar. Som vi såg ovan så kan en 8-bitars signerad variabel anta värden upp till 127, medan en 8-bitars osignerad variabel kan anta värden upp till 255. Dock kan båda talen anta exakt $2^8 = 256$ olika värden, med skillnaden att signerade kan anta lika många negativa som positiva värden, medan signerade tal endast kan anta positiva värden, om vi antar att 0 är ett positivt tal för både signerade samt osignerade tal.

- Detta kan märkas i programmering, exempelvis programspråket C, där den huvudsakliga datatypen för heltal, int, allokerar 2 eller 4 byte datorminne, vilket är detsamma som 16 eller 32 bitar, beroende på vilken kompilator som används.

- Anta att vi använder en kompilator som allokerar fyra byte (32 bitar) för en intvariabel. Teoretiskt sett så kan då en variabel av typen int anta alla värden mellan 0 upp till $2^{32} – 1$, vilket är ca 4,3 * 10 ^ 9. Men eftersom en sådan variabel också skall kunna anta negativa tal, inte bara positiva tal och noll, så medför detta att denna variabel istället kan anta alla värden i ett intervall ungefär mellan ± 2,15 * 10 ^ 9.

- Som exempel, om vi hade velat skapa en variabel av datatypen int, som döptes till tal och som skall tilldelas värdet $5 * 10^9$, så hade det inte fungerat ifall kompilatorn allokerar fyra byte (32 bitar) för en vanlig (signerad) intvariabel. Nedan försöker vi deklarera en variabel av datatypen *int,* som döps till tal och tilldelas 5 * 10 ^ 9 i programspråket C#, vilket inte fungerar:

```
// Försöker deklarera en variabel av datatypen int, som döps till tal och tilldelas talet 5*10^9:
int tal = 5000000000;
// Talet är dock för stort för att sparas i en variabel av datatypen int,
// så vi behöver använda datatypen long.
```

- Vi hade kunnat försöka maximera hur stora positiva tal intvariabeln tal kan innehålla genom att göra den osignerad. Detta kan åstadkommas genom att sätta variabeln till *unsigned int,* vilket skrivs som *uint* i C#. Dock är talet 5 * 10 ^ 9 för stort för detta, då en variabel av datatypen *unsigned int* endast kan anta värden upp till ungefär 4,3 * 10 ^ 9.

- I detta fall behöver mer minne allokeras, så att mer än fyra byte kan lagras. Då bör istället datatypen *long int* användas, som medför att åtta byte (64 bitar) datorminne allokeras. *Long int* skrivs som *long* i C#:

```
// Deklarerar en variabel av datatypen long, som döps till tal och tilldelas talet 5*10^9:
long tal = 5000000000;
```

- Detta hade fungerat utmärkt, eftersom kompilatorn allokerar 64 bitars datorminne för variabler av datatypen *long int*. Variabeln tal hade då kunnat anta värden upp till ca 9,2 * 10^18 i osignerad form!