

Logiske operasjoner på binære tall. (Boolean logic).

Dette kompendiet forutsetter at du vet hvordan man regner om tall mellom det binære, det desimale og det heksadesimale systemet. Bare for å friske opp erindringen skal jeg sette opp en liten tabell:

Verdi	Desimaltall	Binærtall	Heksadesimaltall
Null	0	0	0
En	1	1	1
To	2	10	2
Tre	3	11	3
Fire	4	100	4
Fem	5	101	5
Seks	6	110	6
Syv	7	111	7
Åtte	8	1000	8
Ni	9	1001	9
Ti	10	1010	A
Elleve	11	1011	B
Tolv	12	1100	C
Tretten	13	1101	D
Fjorten	14	1110	E
Femten	15	1111	F

Vi skal nå se på hva vi kan gjøre med binære tall. Altså skal vi se på matematikken og de logiske operasjonene vi kan foreta og hvilke resultater vi kan oppnå.

Vi husker at datamaskiner bare bruker binære tall, altså bare 0 og 1. Maskinene kan heller ikke regne så veldig godt. De kan for eksempel ikke multiplisere eller dividere. De gjør dette med massevis av addisjoner eller massevis av subtraksjoner.

Vi snakket også om hvordan vi angir at tall er binære og heksadesimale. Vi setter en "b" etter binære tall og en "h" etter heksadesimale tall. Det er også vanlig å bruke "senkede tall" etter følgende system: binære tall: 1_2 desimale tall: 7_{10} og heksadesimale tall: 9_{16} . Dette bare for å ha nevnt det, vi fortsetter med b og h for binære og heksadesimale og ingenting for desimale tall. Slapp av, vi skal ikke snakke mer om heksadesimale tall i denne omgangen.

Addisjon

Det å addere binære tall er svært likt det å addere desimaltall. Begrensningen ligger i at du bare kan bruke 1 og 0. Derfor er $1b + 1b \neq 2$ (tegnet \neq betyr ulik og er motsatt av $=$), det riktige oppsettet er: $1b + 1b = 10b$. Sjekk mot tabellen over. Du bare setter tallene over hverandre, legger dem sammen og gir en i mente dersom svaret ditt er mer enn en:

Eksempel:

Forklaring: (begynn til høyre, gå til venstre)

$$\begin{array}{r} 111 \\ 0010\ 1010b \\ + \underline{0011\ 1100b} \\ 0110\ 0110b \end{array}$$

$$\begin{array}{l} 0b + 0b = 0b \\ 1b + 0b = 1b \\ 0b + 1b = 1b \\ 1b + 1b = 0b \text{ (og en i mente)} \end{array}$$

$$\begin{array}{l} 1b + 0b + 1b = 10b \text{ (0 og en i mente)} \\ 1b + 1b + 1b = 11b \text{ (en og en i mente)} \\ 1b + 0b + 0b = 1b \\ 0b + 0b + 0b = 0b \end{array}$$

One's complement

One's complement er et uttrykk som jeg har valgt å ikke oversette. Det er dette uttrykket du finner i litteraturen og det er bare unødig mas å leke Sprakrad i denne sammenhengen. Dette gjelder også andre uttrykk senere i dokumentet.

One's complement er en enkel operasjon (foregår i et trinn, bare en operasjon) og innebærer at alle bits i et binærtall skifter til motsatt verdi. Dette kalles, på engelsk, "to flip the bits". Tegnet for One's complement er tilde "~".

$$\begin{array}{r} \sim 0110\ 0100b \\ \underline{ 0110\ 0100b} \\ 1001\ 1011b \end{array}$$

Two's complement:

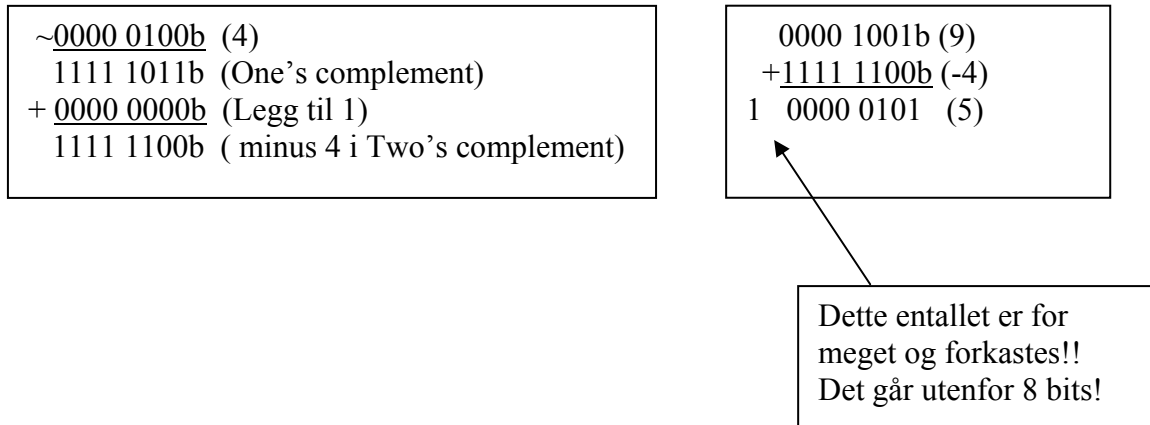
Two's complement er en totrinns operasjon, altså den har to operatorer. Formålet er å få til NEGATIVE tall. Det er noe en computer egentlig ikke forstår noe av, derfor må vi lure den. De negative tallene bruker vi for å få computeren til å trekke fra, altså subtraksjon. Det er egentlig også noe en PC ikke kan, den kan faktisk bare legge sammen. Vi vet i midlertid at ved å plusse et negativt tall og et positivt tall får vi egentlig en subtraksjon. Vi skal se på det nærmere.

Det to trinnene vi bruker er enkle, det første er å beregne One's complement til tallet, det neste trinnet er å legge til en.

$$\begin{array}{r} \sim 0110\ 0100b \text{ (opprinnelig tall)} \\ 1001\ 1011b \text{ (One's complement)} \\ + \underline{0000\ 0001b} \text{ (Legg til en)} \\ 1001\ 1100b \text{ (Negativ representasjon)} \end{array}$$

Dette var litt rart? Ikke sant?? Det er IKKE lett å se dette ”med bare et halvt øye”. Vi skal derfor se på et reelt eksempel. Husk på at i en datamaskin av den typen vi bruker (IBM kompatible maskiner bygget på von Neumans prinsipp) er det plass til åtte bits i en byte. (Det finnes unntak, men ikke i denne sammenhengen). Hva skjer da hvis det blir 9 bits? Det kan enten legges i stack og brukes i neste operasjon, eller VI KAN FORKASTE DET! Det er her det store tricket ligger. Vi forkaster alt som overstiger 8 bits, og det er den bitsen som er lengst til venstre (MSD) som blir forkastet.

Jeg skal vise hvordan en datamaskin regner ut $9 - 4$ og får 5 til svar.



På denne måten kan maskinen slippe å trekke fra (subtrahere) ved å legge til et negativt tall istedenfor å trekke fra et positivt. Det blir det samme om du legger sammen $9 + (-4)$ eller trekker fra $9 - 4$. Svaret blir allikevel 5.

Du kan tenke deg at den venstre (MSD) biten i et negativt tall (ved bruk av Two's complement) endrer verdi fra positiv til negativ. I eksempelet over blir -4 representert slik:

$\begin{array}{cccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & (-4) \\ -128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 & \end{array}$ $-128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = -128 + 124 = -4$

Vi tenker oss altså at MSD representerer -2^7 eller -128 . De andre bitsene representerer seg selv på vanlig måte som sine respektive potenser av 2.

Shift left:

Shift left er en enkel operasjon (bare en operator) som flytter alle bits en plass til venstre og legger til en null på slutten av tallrekken. Verdien blir doblet fordi den multipliseres (ganges) med to. Tegnet for "shift left" er "<<".

Eksempel:

$$\begin{array}{r} \ll 0000\ 1111\text{b (15)} \\ \hline 0001\ 1110\text{b (30)} \end{array}$$

$$\begin{array}{r} \ll 0010\ 1001\text{b (41)} \\ \hline 0101\ 0010\text{b (82)} \end{array}$$

Shift right:

Shift right er en enkel operasjon (bare en operator) som flytter alle bits en plass til høyre og legger til en null til venstre i tallrekken. Resultatet er at tallet halveres (deles på 2), men hvis det er en rest så forkastes den. Det blir ikke "rundet av", den bare blir borte...

Tegnet for "Shift right" er ">>".

$$\begin{array}{r} \gg 0101\ 0010\text{b (82)} \\ \hline 0010\ 1001\text{b (41)} \end{array}$$

$$\begin{array}{r} \gg 0010\ 1001\text{b (41)} \\ \hline 0001\ 0100\text{b (20)} \end{array}$$

Så langt matematiske operasjoner, vi skal se på de logiske operasjonene.

I en logisk operasjon blir to binære tall sammenlignet bit for bit. For eksempel blir bit nr. 1 i tall A sammenlignet med bit nr. 1 i tall B, bit nr. 2 i tall A blir sammenlignet med bit nr. 2 i tall B, bit nr. 3 i tall A blir sammenlignet med bit nr. 3 i tall B osv. Resultatet av sammenligningen avhenger av hvilken logisk operasjon vi benytter. De er: **AND**, **OR** og **XOR**. (dertil kommer NOT og et par til, men de kan vente litt)

Logisk AND:

Denne operasjonen sammenligner to binære tall bit for bit. Hvis **begge** bitsene i en gitt posisjon er 1 blir svaret 1, ellers blir det 0. Tegnet for AND er "&"

$$\begin{array}{r} 0000\ 1100b \\ \& \underline{0000\ 1010b} \\ 0000\ 1000b \end{array}$$

Logisk OR:

Denne operasjonen sammenligner to binære tall bit for bit. Hvis **den ene, eller den andre, (eller begge bitsene)** er 1 blir svaret 1, ellers blir det 0. Tegnet for OR er "V"

$$\begin{array}{r} 0000\ 1100b \\ \vee \underline{0000\ 1010b} \\ 0000\ 1110b \end{array}$$

Logisk XOR:

Denne operasjonen sammenligner to binære tall bit for bit. Hvis **nøyaktig en** av de to bitsene er 1, blir svaret 1, ellers blir det 0. Tegnet for XOR er "^"

$$\begin{array}{r} 0000\ 1100b \\ \wedge \underline{0000\ 1010b} \\ 0000\ 0110b \end{array}$$

Det finnes også en operator som heter NOT, med tegnet "~", men vi lar den være til en annen gang. Du kan jo se litt på hva de andre gjør og så se om du finner ut hvordan NOT må operere. Lykke til.

Oppgaver:

1) Legg sammen (adder) disse binære tallene:

```
0001 0101
+ 0001 0010
-----
=====
```

2) Finn One's complement av dette binære tallet

```
~0010 1010
-----
```

3) Finn two's complement av dette binære tallet

```
0010 1010
```

4) Foreta en shift left på dette binære tallet:

```
<< 0010 1010
```

5) Foreta en logisk AND på disse to binære tallene:

```
0011 0101
& 0001 1110
-----
```

6) Foreta en logisk OR på disse binære tallene:

```
0011 0101
| 0001 1110
-----
```

7) Foreta en XOR på disse to binære tallene:

```
0011 0101
^ 0001 1110
-----
```

9) konverter dette binære tallet til desimaltall: (Forsøk først i hodet, så ser du hva du kan!):

0011 0101b = _____