

# Computersystemen 1

---

## 1. Talstelsels

### 1.1. Talstelsels

- Getal = opeenvolging v. symbolen
- Plaats van symbool is belangrijk

#### 1.1.1. Algemene vorm

- $\sum_{i=0}^n a_i \cdot G^i \rightarrow a_n \cdot G^n + a_{n-1} \cdot G^{n-1} + \dots + a_1 \cdot G^1 + a_0 \cdot G^0$
- Grootste getal met N cijfers =  $G^n - 1$
- # getallen =  $G^n$
- G symbolen

#### 1.1.2. Decimaal

- G = 10
- 0 tot 9

#### 1.1.3. Binair

- Computers werken met elek. signalen (= spanningsniveau). Ze 0 (niets) of 1 (iets) laten herkennen is dus makkelijker dan 0 tot 9.
- G = 2
- 0 tot 1
- $[1001]_2 = [9]_{10}$
- Machten v. 2 zijn dus belangrijk:

$i$	0	1	2	3	4	5	6	7	8	9	10	11
$2^i$	1	2	4	8	16	32	64	128	256	512	1024	2048

#### 1.1.4. Hexadecimaal

- G = 16
- 0 tot 9, A tot F

#### 1.1.5. Octaal

- G = 8
- 0 tot 7

## 1.2. Veranderen van talstelsel

### 1.2.1. G-tallig => decimaal

$$\sum_{i=0}^n a_i \cdot G^i$$

$$[123]_4 = [?]_{10}$$

$$1 \cdot 4^2 + 2 \cdot 4^1 + 3 \cdot 4^0 = 27$$

### 1.2.2. Decimaal => G-talig

- deel getal door G: resultaat Q en rest R (if kommagetal dan stuk achter komma (inc. 0,) \* G = rest)
- schrijf R op
- als  $Q \geq G$  dan stap 1 opnieuw met Q
- anders: schrijf Q op
- Vb:

$$[12345]_{10} = [?]_{20}$$

12345	5
617	17 => I
30	10 => A
1	

$$\Rightarrow [1AI5]_{20}$$

### 1.2.3. Binair ☐ (Hexadecimaal || Octaal)

- Elk hexa symbool = 4 bits
- 0 = 0000, 1 = 0001, 8 = 1000, D = 1101, F = 1111
- Elk octaal symbool = 3 bits
- 0 = 000, 1 = 001, 2 = 010, 3 = 011, 4 = 100, 5 = 101, 6 = 110, 7 = 111

$$[1100\ 1010\ 1111\ 1110]_2 = [?]_{16}$$

$$1100 = C; 1010 = A; 1111 = F; 1110 = E \rightarrow [CAFE]_{16}$$

## 1.3. Negatieve getallen

- In binair kunnen we geen - voor plaatsen + er is een vast aantal bits gereserveerd
- 4 oplossingen
  - sign-bit
  - 1-complement
  - 2-complement
  - offset

### 1.3.1. Sign-bit

- MSB gebruiken om teken voor te stellen (1 is -, 0 is +)
- Nadelen
  - 2 voorstellingen voor 0 (+0 & -0)
  - 2 discontinuïteiten
    - van 011 (+3) naar 100 (-0)
    - van 111 (-3) naar 000 (+0)

### 1.3.2. 1-complement

- Sign bit + negatieve getallen inverteren (5 = 0000 0101, -5 = 1111 1010)
- Voordeel
  - Minder discontinuïteiten

### 1.3.3. 2-complement

- 1-complement + 1 optellen (positieve vorm is dan inverse + 1 aftrekken)
- Voordeel

- Rekenen zonder veel probleem met teken
- Maar één 0

### 1.3.4. Offset

- Tel bij ieder getal (ook positief) een bepaalde waarde (offset) op
- Voordeel: kan kiezen hoeveel negatieve getallen er zijn
- Nadeel: 0 wordt niet voorgesteld met 0

### 1.3.5. Algemeen voorbeeld

Decimaal	Sign-bit	1-c	2-c	Offset 127
-10	1000 1010	1111 0101	1111 0110	0111 0101
Binair	Sign-bit	1-c	2-c	Offset 127
1100 0101	-69	-58	-59	70

Bereik (1.6.2.4): ?? //TODO

## 1.4. Binair talstelsel in informatica

- 8 bits = byte
- 16 bits = word
- 32 bits = double word
- 64 bits = qword

### 1.4.1. Java

#### 1.4.1.1. Java datatypes

- 2-complement
- byte: 1 byte, -128 tot 127
- short: 2 bytes, -32768 tot 32767
- int: 4 bytes, ong. -2 milj tot ong 2 milj
- long: 8 bytes
- bereik is dus beperkt. Als je te grote waardes wilt gebruiken: overflow

#### 1.4.1.2. Getallen omzetten in Java

- `Integer.toString(i, r)` -- `i` = om te zetten getal, `r` = nieuw grondtal
  - `Integer.toBinaryString(i)`
  - `Integer.toHexString(i)`
  - `Integer.toOctalString(i)`
- Ook in classes `Byte`, `Short` en `Long`

#### 1.4.1.3. BigInteger

- Zeer grote getallen zonder kans op overflow
- Zo groot als nodig (vraagt extra memory)
- Niet op gebruikelijke manier te gebruiken

```
BigInteger a = BigInteger.valueOf(6);
BigInteger b = BigInteger.valueOf("7");
BigInteger product = a.multiply(b);
System.out.println(product);
```

### 1.4.2. Grootheden

- 1024 ( $2^{10}$ ) bytes = kilobyte
- Om commerciële redenen wordt soms factor 1000 ipv 1024 gebruikt. (bv. harddisks). Bij 1 TB is dit verschil bijna 100 GB.

### 1.4.3. MSB / LSB

- MSB: Most Significant Bits/Bytes; LSB: Least Significant Bits/Bytes; Bit/Byte hangt af v context
- Most: meest linkse bit/byte; Least: meest rechtse bit/byte

### 1.4.4. Little endian / big endian

- Big endian: eerst MSB opslaan, als laatste LSB
- Little endian: eerst LSB opslaan, als laatste MSB

## 1.5. Binair rekenen

### 1.5.1. Optellen / aftrekken

cijferen:

carry		0	0	1	1	0	0	0	0
48	=	0	0	1	1	0	0	0	0
+ 28	=	0	0	0	1	1	1	0	0
76	=	0	1	0	0	1	1	0	0

- Let op: Wanneer 2 MSB van carry verschillend zijn is er overflow

Vb: 50 + 50

carry		<u>0</u>	<u>0</u>	1	1	0	0	1	0
50	=	0	0	1	1	0	0	1	0
+ 50	=	0	0	1	1	0	0	1	0
100	=	0	1	1	0	0	1	0	0

### 1.5.2. Vermenigvuldiging

#### 1.5.2.1. Cijferen

- Nadelen:
  - Veel optellingen
  - Optellingen vragen veel tijd

#### 1.5.2.2. Algoritme van Booth

- Sommige decimale vermenigvuldigingen zijn makkelijk, bv:  $12345 * 999 = (12345 * 1000) - 12345$
- Algoritme:

- 2 getallen: a en b, vermenigvuldigen
- a en b hebben elk n bits
- maak 3 getallen met  $2n+1$  bits
  - $A = \text{aaaa aaaa 0000 0000 0}$
  - $A' = \text{AAAA AAAA 0000 0000 0}$  (waar AAAA AAAA = 2-complement van a)
  - $P = \text{0000 0000 bbbb bbbb 0}$
- Herhaal n keer:
  - Kijk naar 2 LSB van P
    - = 01 =>  $P = P + A$
    - = 10 =>  $P = P + A'$
    - Anders: doe niets
  - Doe een arithmetic shift right op P (alles 1 naar rechts schuiven, link 0 bijvoegen)
- Schap de LSB van P
- Het resultaat staat in de 16 (of  $2n$  ???) LSB van P

## 1.6. Oefeningsvoorbeelden

### 1) Verander het talstelsel

----