DICTAAT EMBEDDED SYSTEMS

Figuur 1: voorbeelden van ingebedde systemen

Versie: 1.1
Bewerkt door: R.A. van Neijhof
Oktober 2018

INHOUDSOPGAVE

H1. Inleiding

Er wordt wel gezegd dat we in het digitale tijdperk leven. Dit tijdperk is begonnen ergens tussen 1950 en 1970 met de opkomst van digitale computers en digitale systemen in het algemeen. Een belangrijke bijdrage aan het digitale tijdperk is de massaproductie van digitale systemen, zoals computers, huishoudelijke apparaten en smartphones. De productie van een chip (een plakje silicium met daarop miljoenen transistoren) wordt steeds goedkoper en het aantal transistoren op een chip neemt nog steeds toe. Een microprocessor of een microcontroller is voor een paar euro te koop.

Digitale systemen zijn "intelligenter" dan analoge systemen. Neem als voorbeeld de thermostaat van de centrale verwarming. Een jaar of 30 geleden was dit een simpele aan/uit-regeling. Door toevoeging van drukknoppen, een display en een microprocessor met software kan de gebruiker het complete dagritme van de verwarming instellen. Dit is een voorbeeld van wat we een '*embedded system'* noemen. Het is zelfs mogelijk om de thermostaat draadloos te bedienen met een smartphone of tablet. Daarnaast kunnen er allerlei gegevens worden bijgehouden als temperatuur - verloop over de dag, luchtvochtigheid en luchtdruk. Voor de laatste twee is de thermostaat eigenlijk niet bedoeld, maar het is mooi meegenomen.

Het internet heeft ontegenzeggelijk bijgedragen aan de digitalisering van systemen. Denk hierbij alleen maar aan websites met een webshop. Bedrijven hebben dan geen winkel meer nodig en kunnen voorraden tot een minimum beperken. De laatste trend is het Internet of Things (IoT), een netwerk van kleine en grote apparaten die informatie met elkaar uitwisselen. [1]

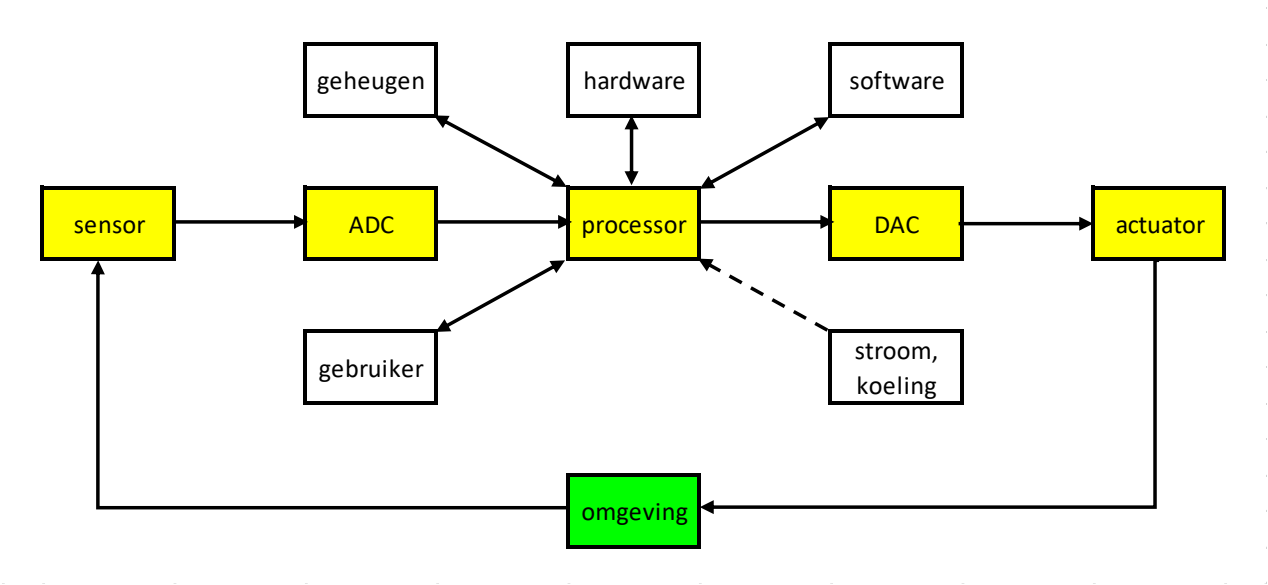
In dit hoofdstuk bespreken we enkele eigenschappen of kenmerken van een embedded systeem, ook wel ingebed systeem of geïntegreerd systeem genoemd. In de loop der jaren zijn er nogal wat definities geschreven over wat een embedded system nu precies is. Enkele voorbeelden zijn:

1. Een embedded system is een elektronisch systeem (hardware én software) dat is geïntegreerd in gebruiksartikelen of apparaten, met de bedoeling deze een vorm van intelligent gedrag te bezorgen. De essentie van een embedded system is dat er software zit ingebed in een hardware-apparaat. Voorheen had je elektronische meet- en regelsystemen die geheel uit hardware bestonden. Bij een embedded system nemen processoren (microprocessoren en/of controllers) met software een gedeelte van deze elektronische meet- en regeltaken over. Doordat de software eenvoudig vervangen kan worden, is het apparaat flexibeler aan te passen aan toekomstige eisen. [2]
2. An embedded system is a dedicated computer system designed for one or two specific functions. This system is embedded as a part of a complete device system that includes hardware, such as electrical and mechanical components. The embedded system is unlike the general-purpose computer, which is engineered to manage a wide range of processing tasks. [3]
3. Een Embedded System is een elektronische besturing welke in een product is ingebed, en daarmee een onveranderlijk deel van dat product en zijn functionaliteit is geworden. Onder Embedded Software verstaan we software in het ingebedde systeem, welke alleen bedoeld is om de hardware zijn voor het product vastgelegde taak te doen uitvoeren en welke door de gebruiker van het product niet is te veranderen. [4]

1.1 Onderdelen

Een algemeen ingebed systeem bestaat uit een sensorgedeelte dat de omgeving waarneemt, een communicatiegedeelte dat de waargenomen informatie converteert (bijvoorbeeld digitaliseert) en doorstuurt naar een informatie verwerkend gedeelte (CPU en software), en een actuatorgedeelte dat

het gedrag van de omgeving (waarin het systeem is ingebouwd) aanstuurt op basis van de beslissingen genomen door het informatie verwerkend gedeelte. [2]



Figuur 2: Onderdelen van een ingebed systeem [5]

1.2 Toepassingen

Omdat chips steeds goedkoper en flexibeler worden (voor de processor, zowel als de sensoren en de actuatoren), komen er steeds meer ingebedde systemen op de markt: pinautomaten, kopieer-machines, industriële meetapparatuur, robots, ziekenhuisapparatuur, mobiele telefoons, magnetrons, wasmachines, foto- en filmapparaten, consumentenelektronica, moderne auto's en andere mechatronische systemen. Het toenemende gebruik van embedded systems wordt gezien als de derde innovatiegolf in de ICT, na de explosieve opkomst van het internet en de mobiele telefonie. Dit is het gevolg van de voortdurend dalende productiekosten en omvang van transistoren, zodat ingebedde systemen in principe overal kunnen worden ingebouwd en energiezuiniger worden. Kritieke toepassingen van ingebedde systemen op het vlak van betrouwbaarheid, veiligheid en levensduur vinden we in de medische wereld (pacemakers, multi-DSP-hoorapparaten en -implantaten zoals het cochleair implantaat), de militaire wereld (besturingssystemen voor raketten) en de meeste toestellen in de vlieg-, ruimte- en onderwaterindustrie. Het informatie verwerkend gedeelte is reactief: het moet op ieder ogenblik (en soms binnen strikte reële tijdbeperkingen) op een betrouwbare manier kunnen reageren op veranderingen in de omgeving. [2]

1.3 Algemene kenmerken van een ingebed systeem:

1. Heterogeen: Dankzij de samenwerking tussen verschillende vakgebieden is het voor embedded systemen mogelijk om te communiceren met verschillende soorten netwerken en andere embedded systemen.
2. Onopvallend: Een ingebed systeem zit ingebouwd in machines of apparaten, meestal zonder dat de gebruiker zich bewust is van zijn aanwezigheid. Dit is mogelijk door een andere manier van omgaan met het ingebedde systeem dan met een "normale" computer: in plaats van opdrachten te krijgen van de mens via een toetsenbord of schakelaar, reageert een ingebed systeem vaak automatisch op de omgeving, zonder tussenkomst van de mens. Dit wordt bereikt door het gebruik van sensoren, die bijvoorbeeld stemgeluid, temperatuur of luchtkwaliteit kunnen analyseren en er vervolgens op reageren.
3. Zuinig: Om zo zuinig mogelijk te kunnen opereren kunnen ingebedde systemen bijvoorbeeld energie winnen uit hun omgeving; energie zoals zonlicht of lichaamswarmte kunnen gebruikt

worden om een ingebed systeem te voeden. Zuinigheid is van groot belang, omdat de meeste ingebedde systemen niet aan het elektriciteitsnet gekoppeld zijn.

4. Flexibel: Flexibiliteit is belangrijk om ingebedde systemen aan te passen aan individuele wensen van de gebruiker of naburige ingebedde systemen. Hier komt ook heterogeen om de hoek kijken: een ingebed systeem moet immers wel kunnen communiceren met een naburig systeem. Een systeem kan zo zelfs zelf-configurerend zijn.
5. Betrouwbaar: Ze moeten gedurende heel lange tijd zelfstandig en betrouwbaar functioneren. Denk bijvoorbeeld aan de ABS-functie in auto's, die het zelfs in de meest kritieke situaties niet mag laten afweten.

Ingebedde systemen zullen we in de toekomst dus nog veel meer zien dan nu het geval is. De auto-industrie maakt veelvuldig gebruik van ingebedde systemen. In moderne auto's zitten 20 tot 80 microprocessors die verschillende functies automatisch kunnen activeren aan de hand van de externe omstandigheden: koplampen die aanspringen bij invallende duisternis, automatische ruitenwissers, antiblokkeersysteem (ABS) of airbags enzovoort. Ook zullen er ingebedde systemen komen die dienen als een soort boordcomputer voor een woning, zoals de Home Control Box. [2][6]

Als NSE – student leer je o.a. hardware aan te sturen. Embedded software ligt heel dicht tegen de machine aan. Vaak ben je op bit niveau bezig software te ontwikkelen. Daarom is het bijvoorbeeld van belang dat je soepel met 'enen en nullen' kunt rekenen of dat je een ander talstelsel, dat veel wordt toegepast in dit domein, goed begrijpt en kunt gebruiken bij specifieke toepassingen. Het volgende hoofdstuk bespreekt derhalve vier talstelsels.

H2. Talstelsels

2.1 Introductie

Het begrip getal stelt ons in staat om allerlei zaken te kunnen tellen (0, 1, 2, 3, enzovoort; in de wiskunde spreekt men over de verzameling $\mathbb{N}$, de verzameling van de natuurlijke getallen) en zo een hoeveelheid van iets voor te kunnen stellen. Het voorstellen gebeurt niet altijd op dezelfde wijze. We zouden kralen of kruisjes kunnen gebruiken om een bepaalde hoeveelheid weer te geven of symbolen (cijfers) gebruiken die een bepaalde hoeveelheid voorstellen. [7]

Welke van de volgende rekensommen zijn juist? [8]

- `1 + 1 = 10`
- `1 + 7 = 10`
- `1 + 9 = 10`
- `1 + F = 10`
- `10 + 3 = 1`
- `I + IV = V`
- `10 + 350 = 0`

Mocht je dit voorleggen aan een willekeurig persoon, dan bestaat de kans dat hij of zij zal aangeven dat enkel de derde juist is. Nochtans kunnen ze allemaal juist zijn, mits voldoende uitleg gegeven wordt over het soort talstelsel dat gebruikt wordt. In onze (westerse) wereld werken wij echter graag met machten van tien, in ons decimaal talstelsel. Het is ook logisch, want we hebben tien vingers.

- `1 + 1 = 10, is juist in het binair talstelsel. Decimaal gezien staat er nl. 1 + 1 = 2.`
- `1 + 7 = 10, is juist in het octaal talstelsel. Decimaal gezien staat er nl. 1 + 7 = 8.`
- `1 + 9 = 10, is juist in het decimaal talstelsel.`
- `1 + F = 10, is juist in het hexadecimaal talstelsel. Decimaal gezien staat er nl. 1 + 15 = 16.`
- `10 + 3 = 1, is juist bij uren. Als het nu 10 uur is en je telt er 3 uur bij, is het inderdaad 1 uur.`
- `I + IV = V, is juist bij Romeinse cijfers. Decimaal gezien staat er nl. 1 + 4 = 5.`
- `10 + 350 = 0, is juist bij hoeken in graden: 10 + 350 = 360°. Dan ben je terug bij het begin, dus 0°.`

Zonder het te weten, gebruik je standaard al een talstelsel bij het rekenen: het decimale talstelsel. Maar hoe zit het dan met de andere talstelsels? Dit hoofdstuk gaat over het gebruik van verschillende positiestelsels en hoe je kunt omrekenen van het ene naar het andere. Het gaat dus niet over Romeinse cijfers, uren of hoeken. Uren en hoeken gaan zelfs niet zozeer over talstelsels, maar over modulair rekenen (noodzakelijk bij bv. RSA en cryptografie).

2.2 Leesbaarheidsregels

Om in dit hoofdstuk geen verwarring te krijgen tussen de verschillende talstelsels, worden alle getallen op de plek waar het niet uit de context opgemaakt kan worden, gevolgd door het grondtal van hun talstelsel als subscript. Een ander schrijfwijze is het getal vooraf te laten gaan door een prefix. Geen subscript of prefix houdt automatisch in dat het getal in het decimale stelsel staat geschreven!

naam talstelsel	grondtal	subscript	prefix	voorbeeld
decimaal	10	geen	geen	---
binair	2	$\ldots_2$	%... of 0b...	%10 = 2; 0b11 = 3; $111_2 = 7$
octaal	8	$\ldots_8$	0... (nul...)	010 = 8; $11_8 = 9$
hexadecimaal	16	$\ldots_{16}$	0x...	0x10 = 16; $A_{16} = 10$

Tabel 1: Leesbaarheidsregels

Verder is het gebruikelijk om, wanneer je een getal binair opschrijft, de cijfers te groeperen per 4 of per 8, om de leesbaarheid te bevorderen. In dit hoofdstuk worden ze gegroepeerd per 4.

Werking
Hoe "werkt" zo'n positiestelsel precies? Hoeveel kruisjes staan hier: XXXX? Dat zijn er vier. Normaal zal men dit aantal noteren als 4. Ook een IT'er die het hexadecimale stelsel gebruikt, schrijft voor dit aantal 4. Een computer daarentegen noteert in het binaire stelsel het aantal vier als 100?
Het volgende aantal kruisjes, XXXXXXXXXXXXXXXX, is zestien. Dit aantal schrijft men gewoonlijk als 16, een IT'er in het hexadecimale stelsel echter als 10 en een computer schrijft 10000 in het binaire stelsel. Laten we dit eens bekijken aan de hand van enkele voorbeelden, om dan de algemene regels te presenteren.

2.3 Het decimale talstelsel

Het decimale talstelsel is het meest gebruikte talstelsel. Dit talstelsel gebruikt tien (deca = tien) cijfers. Deze zijn: `0 1 2 3 4 5 6 7 8 9` .
Iedereen zal hierna drieëntwintig kruisjes zien staan XXXXXXXXXXXXXXXXXXXXXXX, genoteerd als 23. Een computer schrijft in het binaire stelsel hiervoor 10111.
Dat er 23 kruisjes staan, is gewoon een kwestie van tellen. Maar waarom schrijft men hiervoor in het decimale stelsel 23? Tijd dus voor een uitgebreide uitleg.

Het tellen gebeurt zoals in een "ouderwetse" kilometerteller in de wagen of met een ouderwets kliksysteem zoals voor het tellen van bijvoorbeeld het aantal personen die ergens passeren:

Figuur 3: Een decimale teller, waar je de volgende en de vorige cijfers ziet.

Figuur 4: Een decimale handteller.

Figuur 5: Een telraam met het getal 1981.

Het telraam telt maar 100 kralen, maar toch kan men veel grotere getallen dan 100 voorstellen. Dan wordt de eerste rij gebruikt voor het tellen van eenheden, de tweede rij voor tientallen, de derde rij voor honderdtallen,... Het getal op het telraam is dan niet 19, maar 1981.

Laten we de eerste negen kruisjes tellen.

```
dec kruisjes
  0
  1 X
  2 XX
  3 XXX
  4 XXXX
  5 XXXXX
  6 XXXXXX
  7 XXXXXXX
  8 XXXXXXXX
  9 XXXXXXXXX
```

Tot nu toe kunnen we de aantallen kruisjes met één symbool (cijfer) aangeven. Maar bij de volgende zitten we met een probleem: in het decimale talstelsel hebben we namelijk niet één symbool om het volgende kruisje voor te stellen. We weten echter dat we verder kunnen tellen met:

```
dec kruisjes
 10 XXXXXXXXXX
 11 XXXXXXXXXX X
 12 XXXXXXXXXX XX
 ...
 19 XXXXXXXXXX XXXXXXXXX
```

De reden dat dit "werkt" is omdat het decimaal talstelsel een positiestelsel is. Zo zijn 12 en 21 opgebouwd uit dezelfde cijfers - nl. '1' en '2' - maar zijn 12 en 21 toch niet dezelfde getallen. De '1' bij 12 is een tiental, wat betekent dat we al één groep van 10 kruisjes hebben. Bij '21' is het tiental een '2', wat betekent dat we al twee groepen van 10 kruisjes hebben.

Als we bij ons voorbeeld 19 één kruisje erbij plaatsen, hebben we een extra groep van tien kruisjes, bij de groep die we al hadden. We hebben dus twee groepen van tien kruisjes, wat we aangeven met 20. Zo kunnen we weer verder tellen:

```
dec kruisjes
 20 XXXXXXXXXX XXXXXXXXXX
 21 XXXXXXXXXX XXXXXXXXXX X
 22 XXXXXXXXXX XXXXXXXXXX XX
 23 XXXXXXXXXX XXXXXXXXXX XXX
```

Dit tellen kunnen we verder doen tot bv. 99, wat betekent dat we 9 groepen van 10 kruisjes en een restgroepje van 9 kruisjes hebben. Als we er nu één kruisje bijplaatsen, hebben we 10 groepen van elk 10 kruisjes. Dit stellen we dan voor door het getal 100.

Het getal 162 betekent dan 1 groep van “10 keer 10 kruisjes" (dus 10^2) , 6 groepen van “10 kruisjes" en 1 restgroep van 2 kruisjes. Of in formulevorm:

$$\mathbf{1 \times 10^2 + 6 \times 10^1 + 2 \times 10^0 = 1 \times 100 + 6 \times 10 + 2 \times 1 = 100 + 60 + 2 = 162}$$

2.4 Het binaire talstelsel

Het binaire talstelsel wordt bijna niet gebruikt door de mens. De hedendaagse computers werken bijna allemaal met binaire getallen: elektriciteit of geen elektriciteit, licht of geen licht, aan of uit, waar of niet waar etc.. Dit talstelsel gebruikt slechts twee (bi = twee) cijfers: `0 1 .`

Figuur 6: Dit is een binair telwerk. Elk wieltje - voor zover men nog van wieltjes kan spreken - heeft slechts twee posities. Er zijn dan ook geen cijfers boven en onder het afleesvenster te zien.

We kunnen dezelfde methode gebruiken zoals bij het decimale talstelsel, namelijk door het tellen van kruisjes:

```
dec  bin  kruisjes
  0    0
  1    1  X
```

Je zou geneigd zijn om '2' te noteren bij XX, maar het cijfer '2' is geen geldig cijfer in het binaire talstelsel. We merken echter op dat we één groep van twee X-en hebben. Bij het tellen van XX schrijven we binair dan ook 10. Vandaar ook de uitdrukking “Er zijn 10 soorten mensen: diegene die binair begrijpen en diegene die dat niet kunnen”. Zo kunnen we verder tellen:

```
dec  bin  kruisjes
  2   10  XX
  3   11  XX X
```

Als we nu een kruisje toevoegen t.e.m. XXXX, dan merken we dat we één groep van twee keer twee kruisjes hebben. Dit klinkt in woorden nogal ingewikkeld, maar als getal is dit het binaire getal 100:

```
dec  bin  kruisjes
  4  100  XX XX
  5  101  XX XX X
  6  110  XX XX XX
  7  111  XX XX XX X
  8 1000  XX XX XX XX
  9 1001  XX XX XX XX X
 10 1010  XX XX XX XX XX
 11 1011  XX XX XX XX XX X
 ...
```

Het binaire getal 101 betekent dan 1 groep van “2 keer 2 kruisjes" (dus 2^2), 0 groepen van “2 kruisjes" en 1 restgroep van 1 kruisje. Of in formulevorm:

$$\mathbf{1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 1 \times 4 + 0 \times 2 + 1 \times 1 = 4 + 0 + 1 = 5.}$$

Dus $101_2 = 5$.

2.5 Het octale talstelsel

Het octale talstelsel wordt bijna niet meer gebruikt. Dit talstelsel gebruikt acht (octo = acht) cijfers:
`0 1 2 3 4 5 6 7 .`

Telvoorbeeld:

```
dec   bin    oct kruisjes
  0 000 000   0
  1 000 001   1  X
  2 000 010   2  XX
  3 000 011   3  XXX
  4 000 100   4  XXXX
  5 000 101   5  XXXXX
  6 000 110   6  XXXXXX
  7 000 111   7  XXXXXXX
  8 001 000  10  XXXXXXXX
  9 001 001  11  XXXXXXXX X
 10 001 010  12  XXXXXXXX XX
 11 001 011  13  XXXXXXXX XXX
 12 001 100  14  XXXXXXXX XXXX
 ...
 15 001 111  17  XXXXXXXX XXXXXXX
 16 010 000  20  XXXXXXXX XXXXXXXX
 17 010 001  21  XXXXXXXX XXXXXXXX X
 18 010 010  22  XXXXXXXX XXXXXXXX XX
```

Het octale getal 164 betekent dan 1 groep van “8 keer 8 kruisjes" (dus 8^2), 6 groepen van “8 kruisjes" en een restgroep van 4 kruisje. Of in formulevorm:

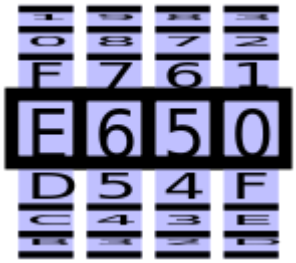
$$\mathbf{1 \times 8^2 + 6 \times 8^1 + 4 \times 8^0 = 1 \times 64 + 6 \times 8 + 4 \times 1 = 64 + 48 + 4 = 116.}$$

Dus $164_8 = 116$.

2.6 Het hexadecimale talstelsel

Men zocht naar een manier om met één symbool verder te kunnen tellen dan 9. Zo besloot men voor het hexadecimale talstelsel om verder te tellen met de letters A, B, C, D, E en F. Binnen het hexadecimale talstelsel zijn de letters A t/m F dus cijfers. De volledige lijst van de 16 (hexadeca = 16) cijfers is dus (van klein naar groot): `0 1 2 3 4 5 6 7 8 9 A B C D E F.`

Opnieuw kunnen we het tellen als basis gebruiken:



Figuur 7: Een hexadecimale teller, waar je de volgende en de vorige cijfers ziet.

```
dec hex kruisjes
  0   0
  1   1 X
  2   2 XX
  ...
  9   9 XXXXXXXXX
```

Nu zijn we geneigd om 10 te plaatsen bij XXXXXXXXX, maar als we hetzelfde principe gebruiken zoals bij de andere talstelsels, dan betekent de '1' van '10' dat we al één groep van 16 kruisjes hebben. De

decimale 10 kunnen we hier dus niet gebruiken. Als we kijken bij de gebruikte "cijfers" van het hexadecimale talstelsel, dan merken we dat na de '9' een 'A' komt. We kunnen dus verder tellen met:

```
dec hex kruisjes
 10   A XXXXXXXXXX
 11   B XXXXXXXXXXX
 12   C XXXXXXXXXXXX
 13   D XXXXXXXXXXXXX
 14   E XXXXXXXXXXXXXX
 15   F XXXXXXXXXXXXXXX
```

Pas nu zijn we op het moment gekomen dat we een groep van 16 kruisjes zullen kunnen maken:

```
dec hex kruisjes
 16  10 XXXXXXXXXXXXXXXX
 17  11 XXXXXXXXXXXXXXXX X
 18  12 XXXXXXXXXXXXXXXX XX
 ...
 26  1A XXXXXXXXXXXXXXXX XXXXXXXXXX
 27  1B XXXXXXXXXXXXXXXX XXXXXXXXXXX
 ...
 32  20 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
 33  21 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX X
```

Het hexadecimale getal 1A2 betekent dan 1 groep van “16 keer 16 kruisjes" (dus 16^2) , 10 (=A) groepen van “16 kruisjes" en 1 restgroep van 2 kruisjes. Of in formulevorm:

$$\mathbf{1 \times 16^2 + 10 \times 16^1 + 2 \times 16^0 = 1 \times 256 + 10 \times 16 + 2 \times 1 = 256 + 160 + 2 = 418.}$$

Dus $1A2_{16} = 418$.

En zo is 0xFF = 255, want

$$\mathbf{15 \times 16^1 + 15 \times 16^0 = 15 \times 16 + 15 \times 1 = 240 + 15 = 255.}$$

Een overzicht van de bekendste talstelsels:

```
dec    bin oct hex kruisjes
  0      0   0   0
  1      1   1   1 X
  2     10   2   2 XX
  3     11   3   3 XXX
  4    100   4   4 XXXX
  5    101   5   5 XXXXX
  6    110   6   6 XXXXXX
  7    111   7   7 XXXXXXX
  8   1000  10   8 XXXXXXXX
  9   1001  11   9 XXXXXXXXX
 10   1010  12   A XXXXXXXXXX
 11   1011  13   B XXXXXXXXXXX
 12   1100  14   C XXXXXXXXXXXX
 13   1101  15   D XXXXXXXXXXXXX
 14   1110  16   E XXXXXXXXXXXXXX
 15   1111  17   F XXXXXXXXXXXXXXX
 16  10000  20  10 XXXXXXXXXXXXXXXX
 17  10001  21  11 XXXXXXXXXXXXXXXXX
 18  10010  22  12 XXXXXXXXXXXXXXXXXX
```

```
...
30  11110   36  1E XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
31  11111   37  1F XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
32 100000   40  20 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
33 100001   41  21 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Wat zijn talstelsels?
Een talstelsel is een systeem om getallen weer te geven in de vorm van een rij symbolen (cijfers). Bij de eerder besproken talstelsels bepaalt de plaats die een cijfer in de rij inneemt hoe we de bijdrage van dat cijfer aan het getal moeten interpreteren. We spreken dan ook over positiestelsels. Alhoewel de benaming 'decimaal talstelsel' gebruikelijk is, zou 'decimaal positiestelsel' duidelijker uitleggen dat we specifiek met een positiestelsel te maken hebben.

Merk op dat alle positiestelsels talstelsels zijn, maar niet alle talstelsels zijn positiestelsels. Zo is de waarde van I in het Romeinse getal IV '1', maar zo ook in het Romeinse getal VI. Dit is niet zo voor het decimale getal 21 versus 12: bij het tweede getal heeft '1' namelijk de waarde van '10'.

Merk op:
Bij het decimale talstelsel is het heel ongebruikelijk om voorloopnullen te laten staan, in de andere talstelsels wordt dit soms wel gedaan (bv. bij ASCII of Wireshark). Alle talstelsels hebben een nul (0). Het grootste cijfer in een talstelsel is (grondtal - 1), omdat de nul ook meetelt als cijfer.
Voor het talstelsel met grondtal n geldt dus dat de mogelijke cijfers komen uit de verzameling $\{\mathbf{0}, \mathbf{1}, \mathbf{2}, \dots, (\boldsymbol{n}-\mathbf{1})\}$.

2.7 Omrekenen

Om van het ene talstelsel om te rekenen naar het andere, is er een aantal methoden. Zo een omrekening wordt ook wel een conversie tussen twee systemen genoemd.

<u>Van een ander stelsel naar het decimale stelsel</u>

Algemeen
Een natuurlijk getal x laat zich in het decimale positiestelsel uitdrukken als een reeks van termen van machten van een ander natuurlijk getal, het grondtal, n:
$x = \sum_{i=0}^{k} x_i n^i$
of
$\boldsymbol{x = x_k n^k + \cdots + x_2 n^2 + x_1 n^1 + x_0 n^0}$
waarbij de coëfficiënten x_i elementen zijn uit de verzameling $\{0,1,2, \dots, (n-1)\}$.

In het n-tallige stelsel wordt x nu voorgesteld door de rij cijfers:
$\boldsymbol{x_k \dots x_2 x_1 x_0}$
De coëfficiënten x_i vormen in volgorde de cijfers van het getal. Het meest linkse cijfer x_k is de coëfficiënt van de hoogste macht van het grondtal, het meest rechtse x_0 de coëfficiënt van de eenheden (de 0-de macht van het grondtal).

Bovenstaande formules zien er ingewikkeld uit, maar met een voorbeeld wordt alles duidelijker.
Het decimale getal 1432 wordt met bovenstaande formule:
$1 \times 10^3 + 4 \times 10^2 + 3 \times 10^1 + 2 \times 10^0 = 1 \times 1000 + 4 \times 100 + 3 \times 10 + 2 \times 1 = 1000 + 400 + 30 + 2 = 1432.$
In het octale stelsel, dus met grondtal 8, gebruikt men de cijfers 0, 1, ..., 7.
Het getal 1432 in het octale stelsel betekent in het decimale stelsel:
$1 \times 8^3 + 4 \times 8^2 + 3 \times 8^1 + 2 \times 8^0 = 1 \times 512 + 4 \times 64 + 3 \times 8 + 2 \times 1 = 512 + 256 + 24 + 2 = 794.$

Dus 1432_8 = 794. En ook 01432 = 794.
Dit staat bekend als de Somregel. Het uitschrijven van alle gewichten en daarna optellen.

472_8 = …
Als voorbeeld hier het omrekenen van het octale getal 472_8naar een decimaal getal. We nemen de volgende stappen:

1. Voor het bepalen van de positie tellen we vanaf 0 en niet vanaf 1.
2. Bepaal voor elke positie het gewicht als het grondtal tot de macht de positie. Bij positie 2 is het gewicht dus 8^2=64.
3. Reken voor elk cijfer de bijdrage uit door het cijfer te vermenigvuldigen met het gewicht. Bij cijfer 4 is de bijdrage dus 4 * 64 = 256.
4. Tel de bijdragen van de cijfers bij elkaar op.

De uitwerking is dan als volgt: het getal is 472_8

positie	gewicht	cijfer	bijdrage
0	1	2	2
1	8	7	56
2	64	4	256
		totaal	314

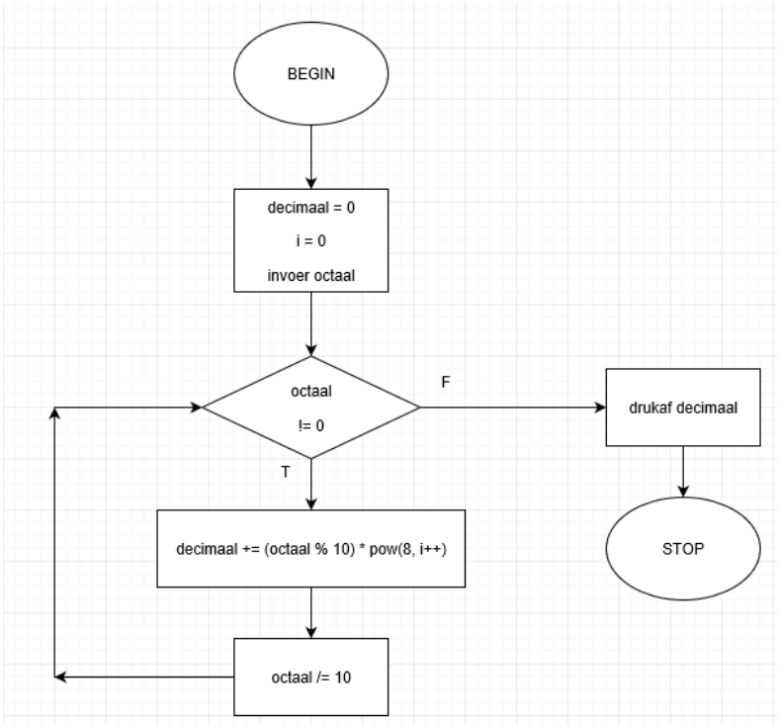
Het resultaat is :

$$\mathbf{472_8 = 4 \times 8^2 + 7 \times 8^1 + 2 \times 8^0 = 4 \times 64 + 7 \times 8 + 2 \times 1 = 256 + 56 + 2 = 314}$$

Dus 472_8 = 314.
Merk op dat in bovenstaande uitwerking alles aan de rechterkant in het decimale talstelsel wordt uitgerekend.

472_8 = …, met computeralgoritme

Bovenstaande methode wordt als mens vaak gebruikt, maar mocht je dit in een programma wensen te gieten, kan onderstaand algoritme van pas komen:



Figuur 8: algoritme voor de conversie van een octaal getal naar een decimaal getal.

Omgezet naar bijvoorbeeld Java – code:

NaarDecimaal.java

```
1  package naardecimaal;
2  import java.util.Scanner;
3  /**
4   * conversie van een octaal getal naar een decimaal getal
5   * @author ravneijhof
6   */
7  public class NaarDecimaal {
8
9      public static void main(String[] args) {
10         Scanner invoer = new Scanner(System.in);
11         int decimaal = 0, i = 0, octaal;
12         System.out.print("Voer een octaal getal in: ");
13         octaal = invoer.nextInt();
14         while(octaal != 0){
15             decimaal += (octaal % 10) * (Math.pow(8, i++));
16             octaal /= 10;
17         }
18         System.out.printf("Omgerekend naar decimaal is dit: %d\n", decimaal);
19     }
20
21 }
```

Figuur 9: het algoritme van figuur 8 omgezet naar Java – code.

Van het decimale stelsel naar een ander stelsel

Het omrekenen van het decimale stelsel naar een ander stelsel is wellicht een stukje ingewikkelder. Er is hiervoor een aantal methoden. Eén daarvan is het 'herhaalt delen door het grondtal en de rest noteren'. Ook wel de Euclidische deling genoemd. Herhaal het delen net zo vaak totdat het quotiënt gelijk is aan nul. Noteer vervolgens alle resten van beneden naar boven. Het is het makkelijkst uit te leggen aan de hand van een voorbeeld.

23 = …$_2$

```
 D / G =  Q R      Uitleg
------------------------
23 / 2 = 11 1      23 = 11 * 2 + 1
11 / 2 =  5 1      11 =  5 * 2 + 1
 5 / 2 =  2 1       5 =  2 * 2 + 1
 2 / 2 =  1 0       2 =  1 * 2 + 0
 1 / 2 =  0 1       1 =  0 * 2 + 1
```

Antwoord: 23 = 10111_2 = 0b10111 = %10111

Stel, we willen 1203 omzetten naar hexadecimaal. Hiervoor kunnen we dezelfde techniek gebruiken als het vorige voorbeeld. Het enige verschil is dat het grondtal (deler) ditmaal 16 is, maar ook hier blijven we delen tot we als quotiënt op 0 uitkomen.

```
   D /  G =  Q  R      Uitleg
-----------------------------------------
1203 / 16 = 75  3     1203 = 75 * 16 + 3
  75 / 16 =  4 11       75 =  4 * 16 + 11
   4 / 16 =  0  4        4 =  0 * 16 + 4
```

Het antwoord wordt dan: 1203 = $4B3_{16}$ = 0x4B3

Shortcuts

Om van het ene stelsel naar het andere om te rekenen, wordt vaak als tussenkomst het decimale stelsel gebruikt. Wil men van octaal naar hexadecimaal, dan is het vaak octaal -> decimaal -> hexadecimaal. Dit vergroot de kans op rekenfouten aanzienlijk. Er zijn echter handigheidjes om van binair naar hexadecimaal om te rekenen (en vice versa) of van binair naar octaal om te rekenen (en vice versa).

De truc zit er hem in, dat een groep van 4 cijfers in het binaire stelsel, precies één cijfer in het hexadecimale stelsel heeft. Of dat een groep van 3 cijfers in het binaire stelsel, precies één cijfer in het octale stelsel heeft Zolang je onderstaande tabel in gedachten houdt, kun je elke kant op omrekenen.

bin	hex	bin	oct
0000	0	000	0
0001	1	001	1
0010	2	010	2
0011	3	011	3
0100	4	100	4
0101	5	101	5
0110	6	110	6
0111	7	111	7
1000	8		
1001	9		
1010	A		
1011	B		
1100	C		
1101	D		
1110	E		
1111	F		

Hexadecimaal <-> binair

Wil je weten wat $30F2_{16}$ is in binair? Neem elk getal apart en schrijf deze op:

```
hex     3     0    F    2
bin  0011  0000 1111 0010
```

Conclusie: $30F2_{16}$ = 00110000 11110010_2 .

De andere kant op werkt net zo gemakkelijk. Wil je weten wat 11000011110010_2 is in hexadecimaal?

Groepeer eerst het binaire getal per 4 cijfers, beginnend aan de rechterkant.

```
bin 11 0000 1111 0010
```

Is er op het eind geen groep van 4, dan kun je hier nullen aan toevoegen.

```
bin 0011 0000 1111 0010
```

Nu kun je per groep kijken welke cijfer er bij hoort

```
bin 0011 0000 1111 0010
hex    3    0    F    2
```

Conclusie: $11000011110010_2 = 30F2_{16}$.

Octaal <-> binair
De omzetting octaal naar binair (en omgekeerd) is net zo gemakkelijk. Werk alleen met groepjes van drie bitjes in plaats van vier!

Decimaal <-> binair
De shortcut-omzetting van decimaal naar binair lukt niet! Dit omdat het grondtal 10 geen macht is van twee. Om één decimaal cijfer voor te stellen zijn 3 bits namelijk te weinig (daar kan je maximaal het getal 7 en niet 9 mee voorstellen) en zijn 4 bits te veel (daar kan je maximaal niet 9 mee voorstellen, maar 15).

Als je het toch probeert, krijg je fouten.
Zo zou 74 met een shortcut-omzetting waar vier bits worden gebruikt

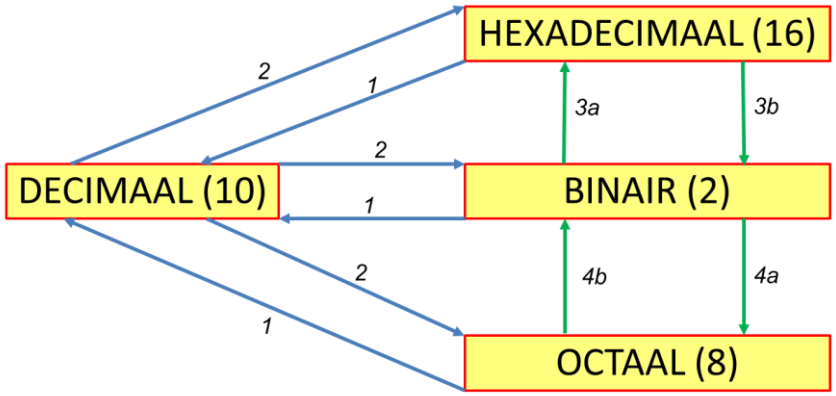
```
dec   7    4
bin 0111 0100
```

dus 74 = 01110100_2 worden, maar dat binaire getal is 116, te bepalen via de somregel.
Het juiste antwoord is 74 = 1001010_2 . Te bepalen via de Euclidische deling.

Conclusie: de omzetting decimaal <-> binair mag NIET via een shortcut!

2.8 Samenvatting

Voor het omrekenen tussen talstelsels zijn de volgende methoden gezien:



Figuur 10: conversie mogelijkheden.

1. ...? -> ... de somregel
2. ... -> ...? de Euclidische deling
3. $...{}_{16}$ <-> $...{}_2$ shortcut
4. $...{}_8$ <-> $...{}_2$ shortcut

Rekenmachines
Bovenstaande manieren zijn handig om het principe van talstelsels te begrijpen, maar als je in de praktijk een omrekening moet doen, is het handiger om gebruik te maken van de mogelijkheden van je besturingssysteem. Zo bieden de rekenmachines van Windows of Gnome standaard de

mogelijkheid om tussen talstelsels om te rekenen. Vaak moet je wel de modus van het rekentoestel veranderen.

NB. Op het tentamen is het gebruik van een rekenmachine <u>NIET</u> toegestaan.

2.9 Opdrachten

Als je het omrekenen van het ene naar het andere talstelsels wilt oefenen, dan kan je de volgende opdrachten uitproberen. Als controle kan je de rekenmachine gebruiken.

a. 101011_2 = …
b. $78A_{16}$ = …
c. 11000011110010_2 = …$_{16}$
d. $DEAD_{16}$ = …$_2$
e. 123_{16} = …
f. 123 = …$_2$
g. 49233 = …$_{16}$
h. 10101 = …$_{16}$
i. 1110101_2 = …

H3. Toepassingen van talstelsels

Afhankelijk van de situatie leent het ene talstelsel zich beter dan een ander. In hoofdstuk 2 is uitleg gegeven over de algemene werking van een talstelsel. Binnen dit hoofdstuk 'toepassingen' is het de bedoeling om te kijken naar waar deze talstelsels voorkomen in de computer of het computer-gebruik. Het is niet de bedoeling om een volledige lijst op te stellen, maar om toch enkele belangrijke voorbeelden te behandelen. [9]

3.1 Binair

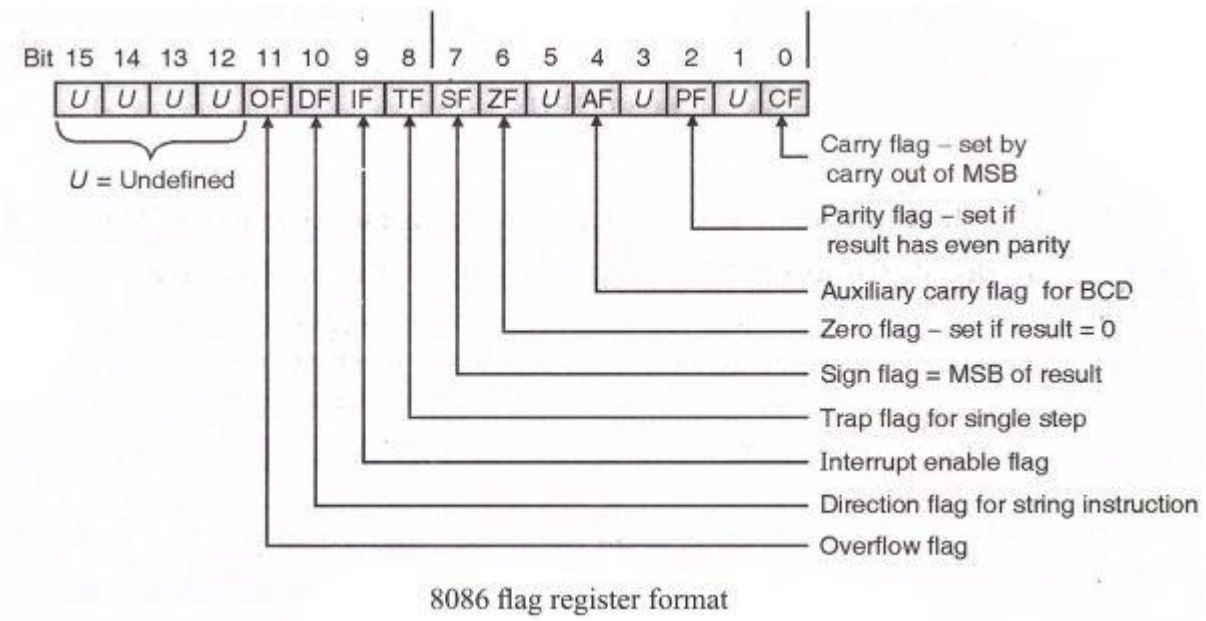
We weten al dat een reeks van bits snel behoorlijk lang kan worden. Vandaar zal je eerder het decimale of hexadecimale talstelsel in gebruik zien, gezien hun compactere vorm. Toch zijn er ook situaties waarbij de binaire notatie nodig is. Dit komt o.a. voor bij subnetting bij netwerken of het gebruik van een *vlag*. Deze *vlag* refereert naar een of meer bits, waaraan een betekenis gekoppeld wordt. In het geval van zaken in verband met netwerken (bijvoorbeeld TCP, IPv4 en IPv6) kan je een programma zoals “Wireshark” gebruiken om dit tot op binair niveau te bekijken.

FLAGS-register

Het FLAGS-register is een register in de Intel x86 microprocessor dat de staat bevat waar de processor zich in bevindt. Dit register is 16 bits breed. Een mogelijk voorbeeld:

```
01011010 11101011
```

In dat register wordt bijvoorbeeld de zevende bit gebruikt voor de zero-flag: is het resultaat van een bewerking al of niet nul. In het voorbeeld staat deze vlag op 1. Merk op dat we uit de hexadecimale voorstelling, nl. 0x5AEB, niet onmiddellijk kunnen afleiden wat de zevende bit als waarde heeft. De binaire notatie is hier dus aangewezen.



Figuur 11: flags – register van een 8086 Intel processor [10]

IPv4

In een IPv4 pakket zitten drie bits die als vlag worden gebruikt. Als je dit bekijkt via een packet sniffer merk je bijvoorbeeld hexadecimaal 40. Je weet dan echter nog niet hoe de vlag van drie bits eruit ziet. Binair wordt $40_{16} = 01000000_2$. De vlag is dus 010. Daaruit kan besloten worden dat dit IP pakket niet mag gefragmenteerd worden en dat er geen verdere fragmenten komen:

```
0... .... = Reserved bit: Not set
.1.. .... = Don't fragment: Set
..0. .... = More fragments: Not set
```

3.2 Octaal

De octale voorstelling wordt niet zoveel meer gebruikt. De octale voorstelling wordt wel nog gebruikt bij het `chmod`-commando in Unix/Linux en het vertelt het systeem hoeveel toegang het moet geven t.o.v. een bestand of map.
Hiervoor bestaan leesrechten (r, van read), schrijfrechten (w, van write) en uitvoerrechten (x, van execute). Dat zijn 3 soorten rechten, met 8 (2^3) combinaties, die dus als een octaal getal kunnen worden voorgesteld:

```
rwx  bin  oct permissie
---------------------------------
---  000    0 niets
--x  001    1 enkel uitvoeren
-w-  010    2 enkel schrijven
-wx  011    3 schrijven en uitvoeren
r--  100    4 enkel lezen
r-x  101    5 lezen en uitvoeren
rw-  110    6 lezen en schrijven
rwx  111    7 volledig
```

Deze rechten worden toegekend voor de eigenaar, de groep en de andere gebruikers.

```
$ touch script.sh
$ ls -l script.sh
-rw-rw-r-- 1 linus leerling 0 Sep  8 22:14 script.sh
```

Hier heeft gebruiker *linus* lees- en schrijfrechten. Iedereen die behoort tot de groep *leerling* heeft ook lees- en schrijfrechten. Alle andere hebben enkel leesrechten.

Stel dat deze gebruiker zelf alle rechten wenst, waarbij gebruikers van de groep leerling enkel mogen lezen. Alle anderen mogen niets kunnen. Deze instelling kan dan worden voorgesteld door het octale getal 740. In combinatie met het commando `chmod` wordt dit dan:

```
$ chmod 740 script.sh
$ ls -l script.sh
-rwxr----- 1 linus leerling 0 Sep  8 22:14 script.sh
```

Een `chmod 800` zal dan niet lukken, daar dit geen octaal getal is:

```
$ chmod 800 script.sh
chmod: ongeldige modus: ‘800’
Typ 'chmod --help' voor meer informatie.
```

3.3 Decimaal

Bij onderstaande IPv4 adressen wordt de decimale notatie (“dotted decimal”) gebruikt:

```
127.0.0.1
192.168.0.12
209.85.147.99
91.198.174.225
91.189.89.88
```

Toch zou het eigenlijk logischer zijn om een hexadecimale notatie te gebruiken. Zo lijkt bovenstaande decimale notatie te suggereren dat een IPv4-adres bestaat uit 4 groepen van 4 getallen, waarbij elk getal bestaat uit maximum drie cijfers. Mocht dat waar zijn, zou ook 621.85.147.12 mogelijk zijn, wat

niet het geval is. Elk getal bestaat namelijk uit 8 bits, waarmee je dus enkel kan gaan tot 11111111_2 = 255 = 2^{8-1}. Dit maakt het theoretische "grootst" mogelijke IPv4-adres 255.255.255.255.
Een hexadecimale notatie zou dan meer voor de hand liggen, maar bij een IPv4 adres is dat niet gebruikelijk. Bovenstaande lijst wordt dan:

```
0x7F000001
0xC0A8000C
0xD1559363
0x5BC6AEE1
0x5BBD5958
```

Het theoretische "grootst" mogelijke IPv4-adres wordt dan hexadecimaal 0xFFFFFFFF.

3.4 Hexadecimaal

Alhoewel een computer werkt in het binaire talstelsel, is dit voor de mens niet zo handig. Zo is het decimale getal 99 binair 1100011 en telt dan 7 cijfers i.p.v. 2. Compact is anders. We merken wel dat 4 bits perfect dient om de 16 mogelijke hexadecimale cijfers voor te stellen (van $0000_2 = 0_{16}$ tot $1111_2 = F_{16}$). Elke reeks van 4 bits (een nibble) kan worden voorgesteld door één hexadecimaal symbool. Dus kan een byte altijd door twee hexadecimale cijfers voorgesteld worden. Software- en computerontwerpers werken dan ook liever met het hexadecimale talstelsel: het vergemakkelijkt de leesbaarheid en voorstelling van binaire informatie.

IPv6

Internet Protocol versie 6 (IPv6) is de opvolger van Internet Protocol versie 4 (IPv4). IPv6 is onder andere ontwikkeld om de beperkingen en tekortkomingen van IPv4 te verhelpen. Met name het tekort aan beschikbare IP-nummers. Zo bestaan IPv4 adressen uit 32 bits, terwijl IPv6-adressen uit 128 bits bestaan. Waar bij IPv4 adressen de decimale notatie het meest gebruikelijk is, is dit bij IPv6 de hexadecimale notatie:

```
3ffe:6a88:85a3:08d3:1319:8a2e:0370:7344
```

Mac-adres

Het MAC-adres is een uniek identificatienummer dat aan een apparaat (bv. een netwerkadapter) in een ethernet-netwerk is toegekend. Het bestaat uit 6 bytes, waarbij de eerste 3 bytes de fabrikant aanduiden en de volgende 3 bytes uniek moeten zijn. Het bestaat dus uit 48 bits (6*8) en wordt genoteerd in hexadecimale vorm:

```
00:24:1D:C1:71:E6
```

In dit geval duidt 00:24:1D op GIGABYTE als fabrikant.

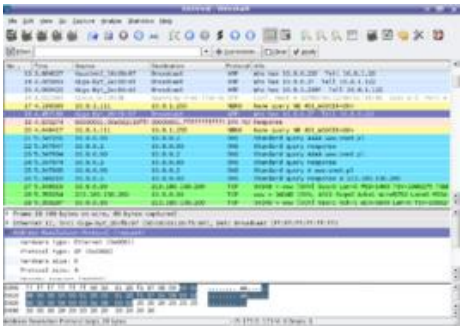
RGB

Het RGB-kleursysteem is een kleurcodering, een manier om een kleur uit te drukken met behulp van een combinatie van de drie primaire kleuren Rood-Groen-Blauw. De hoeveelheid van elke primaire kleur die benodigd is om de mengkleur te verkrijgen, wordt uitgedrukt in een getal dat meestal uit 8 bits bestaat en dus kan variëren tussen 00000000_2 en 11111111_2 of dus tussen 0 en 255.
In HTML en CSS kan de hexadecimale notatie gebruikt worden om HTML-kleuren voor te stellen. Zo betekent #FF0000 dan rood en #FFFF00 dan geel (namelijk als mengeling van rood en groen). Of #FFFFFF wit (alle basiskleuren op hun maximum) en #000000 zwart (geen enkel basiskleur). Of #A3A3A3 een lichtgrijs kleur en #616161 een donkergrijs kleur.
Als ook transparantie nodig is, aangegeven door het alpha-kanaal, worden vaak 8 extra bits gebruikt. Zo betekent 0x80FFFF00 een doorzichtig geel, met een alpha-kanaal van 50,2%. Want: 80_{16} = 128 en 128/255 = 0,50196 = 50.2%. Dit wordt dan RGBA genoemd.

Hexspeak
Doordat er letters voorkomen in het hexadecimale talstelsel heeft dat er programmeurs toe aangezet om creatief om te springen met hun "getallen". Enkele voorbeelden uit hexspeak:

- DEADBEEF ("dead beef") is vaak gebruikt om een software crash of deadlock in embedded systemen aan te geven.
- CAFEBABE ("cafe babe") is o.a. door de Java programmeertaal gebruikt om een Java bytecode class file aan te geven.
- DEADDEAD ("dead dead") is de bug check (STOP) code die getoond wordt bij het oproepen van een Blue Screen of Death. Het zijn vooral driver ontwikkelaars die dit kunnen zien, omdat dit gebruikt wordt om een memory dump te verkrijgen op een Windows NT gebaseerd systeem.
- 8BADF00D ("ate bad food") is gebruikt door Apple in iOS crash rapporten, als een applicatie te lang wacht om te starten, stoppen of te antwoorden op systeemgebeurtenissen.
- FEE1DEAD ("feel dead") is gebruikt in de Linux reboot system call.

Hex-editors en packet sniffers
Om bestanden op bit-niveau te bekijken worden hex-editors gebruikt. Alhoewel hun naam anders doet vermoeden kan je er vaak niet alleen bestanden in hexadecimale vorm bekijken, maar kan je dit ook doen in binaire, octale en decimale variant. Vaak zal men ook per byte het overeenkomstig ASCII-teken weergeven.
Bij netwerkverkeer heeft men het niet over een hex-editor, maar over een packet sniffer. Je merkt gelijkaardige mogelijkheden zoals bij de hex-editors, maar zo'n programma is slimmer, daar het de afzonderlijke protocollen kan herkennen (HTTP, TCP, IP, ARP, Ethernet,...). De bekendste packet sniffer is wellicht Wireshark.



Figuur 12: De packet sniffer "Wireshark" [11]

H4. Negatieve getallen

Tot nu toe hebben we alleen met positieve getallen gewerkt. Hoe coderen we negatieve gehele getallen in binaire systemen (lees: de computer).
In de wiskunde worden negatieve getallen gerepresenteerd door er een minteken ("-") voor te plaatsen. In de hardware worden getallen echter alleen als reeksen bits gerepresenteerd, zonder extra symbolen, dit is nu eenmaal het gevolg van het feit dat computers zijn opgebouwd uit transistoren. Er zijn verschillende binaire representaties voor integers op grond van argumenten zoals: er zijn alleen positieve getallen, of zowel positieve- als negatieve getallen, of de leesbaarheid van de code voor mensen, of de snelheid waarmee een computer met de codes kan rekenen.
Er zijn 4 bekende systemen (en 1 minder bekend systeem) voor integer representaties.
Allen zijn op verschillende momenten in de historie van de computerwetenschap gebruikt en allen om diverse redenen. De systemen zijn:

1. unsigned.
2. sign magnitude.
3. one's complement.
4. two's complement.
5. biased (de “onbekende”).

Zo ongeveer alle moderne computers werken met het 2-complement systeem. Waarom? De hardware is het snelst en de hardware is het simpelst (waardoor het snel is). In dit hoofdstuk worden de vijf systemen besproken. Hoe kan je integers in elk systeem representeren? En hoe kan je een code in een bepaald systeem converteren naar het overeenkomstige decimale equivalent? In hoofdstuk 5 wordt er detaillistisch in gegaan op de berekeningen en hardware die er voor nodig is in de diverse systemen. Dit hoofdstuk, 5, is GEEN tentamenstof maar is bedoeld voor de geïnteresseerde (NSE)lezer.

4.1 Unsigned integers

Unsigned integers, of tekenloze gehelen, of vaak ‘uints’ genoemd, zijn, net zoals gewone integers, gehele getallen. Alleen wordt er aan een uint geen ‘+’ of ‘-‘ teken gekoppeld. Dit impliceert dat een uint een niet-negatief geheel getal is. Anders gezegd, een uint is nul of positief. We kunnen uint's gebruiken als we zeker weten dat bijvoorbeeld een bepaalde variabele nooit negatief zal zijn. Zoals “tel het aantal klinkers in een woord en sla het resultaat op onder de naam *aantal*”. Dit aantal zal 0 of groter zijn.

Er bestaan programmeertalen (bijvoorbeeld *C*) die als datatype uint8_t, uint16_t, uint32_t en uint64_t kennen:

```
uint8_t aantal
```

De variabele `aantal` is van het type ‘unsigned integer’ en bestaat uit 8 bits, dus 1 byte. De binaire waarde die hierin ligt opgeslagen kan worden teruggerekend met behulp van de somregel naar het decimale stelsel zoals dat in H2 is aangeleerd. Zo vertegenwoordigt de byte 10101010 in dit systeem een decimale waarde van
$10101010_2 = 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$.
Dit komt overeen met $2^7 + 2^5 + 2^3 + 2^1 = 128 + 32 + 8 + 1 = 169$.

Het bereik van 1 byte unsigned integer is $[0, 255]$.

Oefeningen

a. De onderstaande codes zijn uint – codes. Wat is hun decimaal equivalent?
   01010101
   11111111

11000000
00001001

De antwoorden zijn respectievelijk: 85, 255, 192 en 9.

b. Converteer de getallen 70, 133, 34 en 300 naar 1 byte binaire code volgens het unsigned integer systeem.
De antwoorden zijn respectievelijk: 01000110, 10000101, 00100010 en *kn* (kan niet).

4.2 Signed magnitude

In het signed magnitude systeem werken we weer met gehele getallen. Echter nu is er wel een '+' of '-' teken aan het getal gekoppeld. Er is hierbij afgesproken dat het bit aan de linkerkant het zogenoemde 'teken-bit' is. Met een '0' als het getal positief is of positief nul en een '1' als het getal een negatief getal of negatief nul is. Het restant van de bits staat voor de grootte (magnitude) van het getal.
Nemen we als voorbeeld dezelfde bitreeks als bij de unsigned integers, dan geldt nu dat door de '1' aan de linkerkant er een negatief getal wordt gerepresenteerd.
In dit systeem staat 10101010_{sm} dus voor -…… . De grootte is gelijk aan de conversie van 0101010_2.
Met de somregel wordt dit $2^5 + 2^3 + 2^1 = 32 + 8 + 1 = 41$.
Met andere woorden: $10101010_{sm} = -41$.
Nog steeds kan er in 1 byte $2^8 = 256$ verschillende codes worden opgeslagen. De helft hiervan begint met een '0' en de andere helft met een '1'. De helft is dus positief, +0 t/m +127, en de andere helft is negatief, -0 t/m -127.

Het bereik van 1 byte signed magnitude is $[-127, 127]$.
De dubbele representatie van de nul, te weten 00000000 en 10000000, zou je als nadeel van dit systeem kunnen bestempelen. En wat bijvoorbeeld met een eenvoudige optelling als -7 + 7?
Om dit te controleren wordt eerst gekeken naar de binaire optelregels: $0 + 0 = 0, 1 + 0 = 1, 0 + 1 = 1$ en $1 + 1 = 10$. Dit laatste is een voorbeeld van wat we een 'carry' noemen. In het decimale talstelsel wordt dit uiteraard ook toegepast.

```
'1 onthouden'   1
                38
                45
                ----
                83
```

Terug naar de uitwerking van -7 + 7 in het signed magnitude systeem. We verwachten als resultaat 0, maar is dit ook zo?

```
        binair     decimaal
carry     111
      10000111      -7
  +   00000111      +7
  ------------      --
      10001110       0
```

De antwoordcode komt overeen met -14 en niet met 0. Blijkbaar heeft dit systeem dus problemen met het correct berekenen van relatief eenvoudige opdrachten.
Dan is er nog een nadeel aan dit systeem. Alle negatieve getallen beginnen met een '1' en alle positieve getallen met een '0'. Voor de hardware (dit wordt uitgevoerd door een *comparator*) is een negatief getal altijd groter dan een positief getal. Immers 1 > 0. In de wiskunde is het juist andersom. Een negatief getal is altijd kleiner dan een positief getal.

naam	bereik 1 byte	Twee codes voor 0?	(-a) + a = 0?	negatief < positief?
signed magnitude	[-127, 127]	ja	nee	nee
1s-complement				
2s-complement				
bias				

Tabel 2: samenvatting t/m signed magnitude

Oefeningen

a. De onderstaande codes zijn signed magnitude – codes. Wat is hun decimaal equivalent?
   01010101
   11111011
   11000010
   00011001
   De antwoorden zijn respectievelijk: +85, -123, -66 en +25
b. Converteer de getallen 74, -100, 100 en -52 naar 1 byte binaire code volgens het signed magnitude systeem.
   De antwoorden zijn respectievelijk: 01001010, 11100100, 01100100 en 10110100.

4.3 1s - complement

Positieve getallen worden in dit systeem voorgesteld door een bitrij beginnend met een 0 en verder door de gebruikelijke binaire voorstelling. Negatieve getallen beginnen met een 1 en het tegengestelde van een getal bestaat uit de bitrij met alle bits geïnverteerd, dus de rij met complementaire bits. Het positieve getal 79 bijvoorbeeld wordt (met 8 bits) voorgesteld door 01001111 en −79 door de rij complementaire bits 10110000. Men kan de voorstelling van −79 ook verkrijgen door de bitrij die 79 voorstelt, af te trekken van de rij met alleen enen: 11111111. Hiervan komt de naam: 1-complement. Als gevolg van deze representatie zijn er twee bitrijen die de waarde 0 voorstellen, nl. 00000000 (+0) en 11111111 (-0). [12]

Formeel beschreven betekent 1-complement, dat de bitrij $b_{n-1} \dots b_2 b_1 b_0$ een positief getal voorstelt als de bit $b_{n-1} = 0$ en een negatief getal als $b_{n-1} = 1$.
Het positieve getal $0b_{n-2} \dots b_2 b_1 b_0$ stelt het binair geschreven getal $+N$ voor, en het tegengestelde hiervan, $-N$, wordt voorgesteld door de bitrij $1b^*_{n-2} \dots b^*_2 b^*_1 b^*_0$, waarin alle bits ($b^*_i$) de geïnverteerde van b_i zijn. De som van beide ($N + (-N)$) levert de bitrij $1 \dots 111$ op, die dus ook het getal 0 voorstelt. In dit systeem geldt dus dat het optellen van twee tegengestelde getallen wel altijd als uitkomst nul oplevert!
Door de dubbele representatie van de nul, is het bereik ook hier symmetrisch verdeeld. Voor een 1 byte code volgens het 1-complement systeem is het bereik $[-127, 127]$.
Ook geldt nog steeds binnen dit systeem dat voor de hardware een negatief getal groter is dan een positief getal.

naam	bereik 1 byte	Twee codes voor 0?	(-a) + a = 0?	negatief < positief?
signed magnitude	[-127, 127]	ja	nee	nee
1s-complement	[-127, 127]	ja	ja	nee
2s-complement				
bias				

Tabel 3: samenvatting t/m 1s-complement

Oefeningen

a. De onderstaande codes zijn 1-complement – codes. Wat is hun decimaal equivalent?
   01010101
   11111011

11000010
00011001
De antwoorden zijn respectievelijk: +85, -4, -61 en +25

b. Converteer de getallen 74, -100, 100 en -52 naar 1 byte binaire code volgens het 1-complement systeem.
De antwoorden zijn respectievelijk: 01001010, 10011011, 01100100 en 11001011.

4.4 2s – complement

2s-complement of 2-complement is een getalsrepresentatie voor gehele getallen (integers) die in computers algemeen wordt gebruikt. Het systeem kent maar heeft maar één representatie voor '0' (waarover zo meteen meer). Het 2-complement is gelijk aan het 1-complement plus 1 als het tekenbit '1' is, dus bij negatieve getallen. [13]

Uit het bovenstaande kan afgeleid worden dat de representatie in 2-complement verkregen wordt door bij de representatie in 1-complement 1 op te tellen: -79 in 1-complement (8 bits) 10110000; tel er 1 bij op: 10110000 + 00000001 = 10110001, de voorstelling van -79 in 2-complement.

Net als bij 1-complement wordt bij 2-complement de meest significante bit gebruikt om aan te geven of een getal positief is of negatief. Deze meest significante bit heeft echter een andere betekenis dan bij 1-complement: in plaats van de rol van een soort minteken te vervullen, staat de meest significante bit (zeg, b_{n-1}) voor -2^{n-1}. De rest van de bits wordt "normaal" geïnterpreteerd en een negatief getal wordt dan ook gevormd door de positieve waarde van de minder significante bits op te tellen bij -2^{n-1}.
Door deze definitie van getallen is de hoogste waarde die gerepresenteerd kan worden met een bitrij waarvan de meest significante bit de waarde 1 heeft, de waarde -1. Het byte 11111111 in 2-complement betekent namelijk $-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = -128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = -128 + 127 = -1$. In tegenstelling tot 1-complement is er dus niet zoiets als '-0' in 2-complement notatie. Bijgevolg is het aantal representeerbare negatieve waarden in 2-complement ook 1 groter dan het aantal representeerbare positieve waarden. Het bereik van een 1 byte code in 2-complement is derhalve asymmetrisch: $[-128, 127]$.
De relatie tussen positieve en negatieve waarden is dan ook als volgt:
Zij $0b_{n-2} \ldots b_2 b_1 b_0$ een bitrij die de waarde $+N$ representeert. Dan is de bitrij die $-N$ representeert gelijk aan $1b^*_{n-2} \ldots b^*_2 b^*_1 b^*_0 + 1$. Met bit b^*_i als complement van bit b_i.

Laten we weer eens naar de uitwerking van -7 + 7 in het 2-complement systeem kijken. We verwachten als resultaat 0, maar is dit ook zo?

```
       binair    decimaal
carry  1111111
       11111001     -7 (2c)
   +   00000111     +7
   ----------     ---
       00000000      0
```

Let op! Als er aan het einde van een berekening nog een '1 onthouden' overblijft, wordt deze weggegooid.

Dus in het kort is 2-complement gelijk aan 1-complement (inverteer alle bits) met daarbij 1 opgeteld. De 2-complement representatie van een integer is vooral zinvol in verband met het optellen van getallen in hardware (dit gebeurt door een component met de naam *adder* of allerlei varianten daar op). Als 2-complement gebruikt wordt, maakt het niet uit of een of beide operanden negatief zijn. Hierdoor is een optelschakeling op een computerchip eenvoudiger te implementeren dan voor andere representaties. Een aparte schakeling om een getal van een ander getal af te trekken, hoeft

niet te worden gemaakt. In dat geval wordt een van de operanden negatief gemaakt alvorens deze op te tellen.

naam	bereik 1 byte	Twee codes voor 0?	(-a) + a = 0?	negatief < positief?
signed magnitude	[-127, 127]	ja	nee	nee
1s-complement	[-127, 127]	ja	ja	nee
2s-complement	[-128, 127]	nee	ja	nee
bias				

Tabel 4: samenvatting t/m 2s-complement

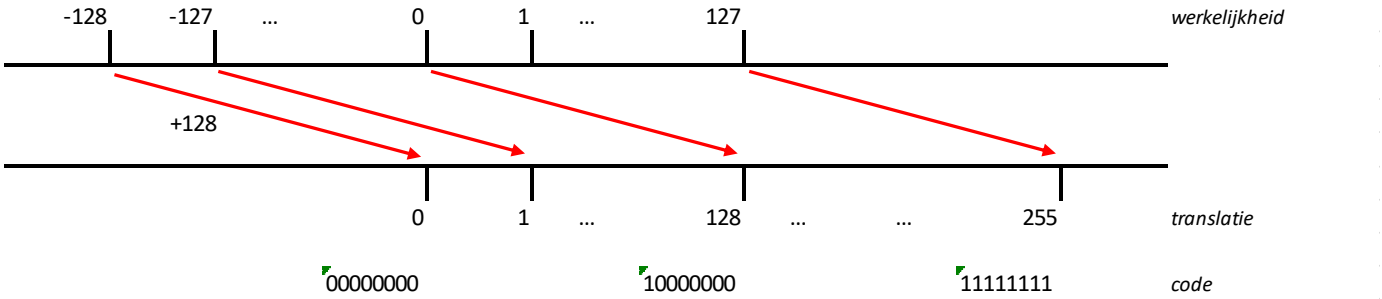
Oefeningen

a. De onderstaande codes zijn 2-complement – codes. Wat is hun decimaal equivalent?
   01010101
   11111011
   11000010
   00011001
   De antwoorden zijn respectievelijk: +85, -5, -62 en +25
b. Converteer de getallen 74, -100, 100 en -52 naar 1 byte binaire code volgens het 1-complement systeem.
   De antwoorden zijn respectievelijk: 01001010, 10011100, 01100100 en 11001100.

4.5 Bias systeem

Offset binair, ook aangeduid als bias representatie, is een binair coderingsschema waarbij “alles 0” overeenkomt met de minimale negatieve waarde en “alles 1” met de maximale positieve waarde. Er is geen standaard voor offset/bias, maar meestal is de bias K voor een n-bits binaire code $K = 2^{n-1}$. Dit heeft tot gevolg dat de "nul" waarde wordt gerepresenteerd door $10 \ldots 000$. [14]
Voor 1 byte, met $n = 8$, geldt dan als bias $K = 2^{8-1} = 2^7 = 128$.
Het minimale negatieve getal wat we kunnen representeren is -128. Immers -128 + 128 = 0 (00000000). En het grootst mogelijke positieve getal in 1 byte met dit systeem is 127, want 127 + 128 = 255 of wel de binaire code 11111111. Ook hier is dan ook sprake van een asymmetrisch bereik.

Het voordeel van dit systeem is nu dat alle negatieve getallen met een ‘0’ beginnen en alle positieve getal (en de nul) met een ‘1’. Kortom, nu geldt ook voor de hardware dat een negatief getal kleiner is dan een positief getal. In feite zorgt het bias-systeem voor een translatie van de getallenlijn waarbij de inwendige structuur blijft gehandhaafd.



Figuur 13: translateren in het Bias systeem.

Tot slot weer de uitwerking van -7 + 7 in het bias systeem bekijken. We verwachten als resultaat 0, maar is dit ook zo?

```
      binair    decimaal
carry 1111111
      01111001     −7 (bias +128 = 121)
   +  00000111     +7
   ————————————    ——
      10000000      0
```

naam	bereik 1 byte	Twee codes voor 0?	(-a) + a = 0?	negatief < positief?
signed magnitude	[-127, 127]	ja	nee	nee
1s-complement	[-127, 127]	ja	ja	nee
2s-complement	[-128, 127]	nee	ja	nee
bias	[-128, 127]	nee	ja	ja

Tabel 5: samenvatting t/m bias-systeem

Het terugrekenen van een code naar de werkelijkheid aan de hand van een voorbeeld, gaat als volgt. Gegeven de code 10101011 in het bias systeem; wat is het decimale equivalent?
Volgens de somregel is 10101011 gelijk aan $2^7 + 2^5 + 2^3 + 2^1 + 2^0 = 128 + 32 + 8 + 2 + 1 = 171$. Volgens figuur 12 is dit dus de 'translatie'. Naar boven naar de werkelijkheid geeft dit het getal $171 - 128 = 43$. Met andere woorden: $10101011_{bias} = 43$.
Oefeningen

a. De onderstaande codes zijn bias – codes. Wat is hun decimaal equivalent?
   01010101
   11111011
   11000010
   00011001
   De antwoorden zijn respectievelijk: -43, +123, +66 en -103.
b. Converteer de getallen 74, -100, 100 en -52 naar 1 byte binaire code volgens het bias systeem.
   De antwoorden zijn respectievelijk: 11001010, 00011100, 11100100 en 01001100.

H5. Integer berekeningen en hardware

Nogmaals, dit hoofdstuk is bedoeld als achtergrondinformatie. Dat wat je hier leest behoort niet tot de tentamenstof.
De verschillende systemen wat betreft het representeren van een geheel getal worden hierna weer besproken. Nu echter op een wat meer formele, soms wiskundige, manier.

5.1 Unsigned integers

De decimale waarde N van een binaire voorstelling wordt berekend aan de hand van de somregel die de waarde van een getal in een positioneel talstelsel met grondtal 2 uitdrukt. In n bits kunnen we precies 2^n verschillende waarden voorstellen. Alle bitpatronen worden m.a.w. gebruikt om een verschillende waarde voor te stellen. Voor een willekeurig aantal van n bits geldt:

- Aantal verschillende waarden: 2^n.
- Bereik: $[0, 2^n - 1]$.
- De bitrij $b_{n-1} \ldots b_2 b_1 b_0$ heeft als decimaal equivalent $N = \sum_{i=0}^{n-1} b_i \times 2^i$.

De C-norm [15] stelt:
"Een berekening waarbij unsigned integer operanden zijn betrokken, kan nooit een overflow opleveren, omdat een resultaat dat niet kan worden gerepresenteerd binnen het bereik $[0, 2^n - 1]$, berekent wordt modulo(2^n)".
Dit gedrag wordt meer informeel unsigned – integer – wrapping genoemd. De volgende operators kunnen resulteren in wrapping: optellen, aftrekken, vermenigvuldigen en left-shiften.
Voorbeeld. De berekening 100 + 200 met behulp van 1 byte unsigned integer variabelen, heeft 44 als resultaat!

```
        binair     decimaal
  carry 1
        01100100      100
    +   11001000      200
    ----------      ---
        00101100      300
```

Het is eenvoudig terug te rekenen dat de bitrij 00101100 de waarde 44 vertegenwoordigt.

[*] addition wrapping.c

```
C:\Users\InstallUser\Desktop\addition wrapping.exe
De uitkomst is 0:
Press any key to continue . . .
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

int som(unsigned short int a, unsigned short int b);

int main(int argc, char *argv[]) {
    unsigned short int x = 65000, y = 536;
    printf("De uitkomst is %u:\n", som(x,y));
    system("PAUSE");
    return 0;
}

int som(unsigned short int a, unsigned short int b) {
  unsigned short int som = a + b;
  return som;
}
```

Figuur 14: voorbeeld unsigned integer wrapping in C.

5.2 Signed magnitude

Tot dusver hebben we positieve gehele getallen omgezet in binaire getallen. Om alle gehele getallen te kunnen voorstellen, moeten we ook een oplossing hebben om negatieve getallen voor te stellen. Een mogelijkheid hiertoe wordt voorgesteld in de signed magnitude methode: de absolute waarde van het getal wordt binair voorgesteld en voorafgegaan door een tekenbit (1 voor negatieve getallen, 0 voor positieve getallen). Dit is gelijkaardig aan de wijze waarop we werken in het decimaal talstelsel: de absolute grootte van een getal wordt al dan niet voorafgegaan door een minteken. Het enige verschil is dat het minteken hier door een 1 wordt voorgesteld.
Bemerk dat er aldus 2 voorstellingen zijn voor het cijfer 0! Dit fenomeen doet zich ook voor in het decimaal talstelsel, maar wij hebben geleerd daar mee om te gaan (weinigen zullen '-0' schrijven als ze 0 bedoelen). Twee voorstellingen voor dezelfde waarde zijn echter niet wenselijk in een computer (het maakt het vergelijken van getallen moeilijker). Een ander probleem is dat de rekenregels voor negatieve getallen verschillend zijn van die voor positieve getallen (bijvoorbeeld indien men 1 wenst op te tellen). Ook de regel om de precisie van de getallen te veranderen (bijvoorbeeld van 4 naar 8 bit) is niet echt eenvoudig: men moet eerst de tekenbit weghalen, dan de precisie aanpassen, en naderhand de tekenbit terugplaatsen.

- Aantal verschillende waarden: $2^n - 1$, dubbele representatie voor de nul!
- Bereik: $[-(2^n - 1), 2^n - 1]$.
- De bitrij $b_{n-1} \dots b_2 b_1 b_0$ heeft als decimaal equivalent $N = (-1)^{b_{n-1}} \sum_{i=0}^{n-2} b_i \times 2^i$.

H6. Gebroken getallen

In het echte leven hebben we te maken met reële getallen, getallen met een fractioneel deel. De meeste moderne computers hebben hardware-ondersteuning voor floating point-getallen. Het gebruik van een zwevend punt is echter niet noodzakelijk de enige manier om fractionele getallen weer te geven. Dit hoofdstuk beschrijft de fixed point representatie van reële getallen. Het gebruik van fixed point datatypen wordt veel gebruikt in digitale signaalverwerking (DSP) en gametoepassingen, waarbij prestatie soms belangrijker is dan precisie. Zoals we later zullen zien, is rekenen in een fixed point formaat veel sneller dan floating point-berekeningen. [16]

Herhaal dat een binair getal zoals bijvoorbeeld 00110101_2 een decimaal equivalent heeft.
Hier geldt: $00110101_2 = 2^5 + 2^4 + 2^2 + 2^0 = 32 + 16 + 4 + 1 = 53.$
Als we het getal 53 delen door 2, weten we dat het resultaat 26,5 moet zijn. Maar hoe stellen we dit voor als we alleen integers-weergaven hebben?

6.1 Binaire punt

De sleutel om fractionele getallen weer te geven, zoals 26,5 hierboven, is het concept van de binaire punt. Een binaire punt is als een decimale komma in het decimaal stelsel. Het werkt als een scheiding tussen het gehele getal en het fractionele deel van een getal.
We weten dat het cijfer "26,5" de waarde "zesentwintig en een half" vertegenwoordigt, omdat
$2 \times 10^1 + 6 \times 10^0 + 5 \times 10^{-1} = 2 \times 10 + 6 \times 1 + 5 \times 0{,}1 = 20 + 6 + 0{,}5 = 26{,}5$, of
$2 \times 10^1 + 6 \times 10^0 + 5 \times 10^{-1} = 2 \times 10 + 6 \times 1 + 5 \times \frac{1}{10} = 20 + 6 + \frac{5}{10} = 26\frac{5}{10} = 26\frac{1}{2}.$
Hetzelfde concept van deze decimale komma kan ook worden toegepast op onze binaire weergave, waardoor een "binair punt" wordt gemaakt. Net zoals in het decimale stelsel vertegenwoordigen alle cijfers (of bits) links van het binaire punt een gewicht van $2^0, 2^1, 2^2$, enzovoort. Cijfers (of bits) aan de rechterkant van het binaire punt dragen een gewicht van $2^{-1}, 2^{-2}, 2^{-3}$, enzovoort. Anders gezegd, de gewichten rechts van de binaire punt vertegenwoordigen de waarden $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}$, enzovoort.
Bijvoorbeeld het getal 0011010.1_2 vertegenwoordigt de waarde:
$0011010.1_2 = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^1 + 1 \times 2^{-1} = 16 + 8 + 2 + 0{,}5 = 26{,}5.$

Een zorgvuldige lezer zou zich nu moeten realiseren dat het bitpatroon van 53 en 26,5 precies hetzelfde is. Het enige verschil is de positie van de binaire punt. In het geval van 53 is er een "geen" binaire punt. Als alternatief kunnen we zeggen dat de binaire punt zich helemaal rechts bevindt, op positie 0. (Denk in decimalen, 53 en 53,0 staat voor hetzelfde getal.)

6.2 Fixed point representatie

Het bovenstaande verschuivingsproces is de sleutel om fixed point representaties te begrijpen. Om een reëel getal in computers te representeren, kunnen we een type getal gebruiken met een vaste plaats voor de binaire punt binnen dat apparaat. Om conceptueel een fixed pointtype te definiëren, hebben we slechts twee parameters nodig:

- w, breedte van de getal weergave, en
- b, binaire puntpositie binnen het getal.

We zullen vanaf nu voor fixed point representatie de notatie $\langle w, b \rangle$ gebruiken. Waarbij w staat voor het aantal bits dat als geheel wordt gebruikt (de breedte van een getal), en b staat voor de positie van de binair punt geteld vanaf de minst significante bit (geteld vanaf positie 0).
Fixed point $\langle 8,3 \rangle$ zegt dan bijvoorbeeld dat de totale code uit 8 bits (1 byte) bestaat, waarvan de drie meest rechtse bits fractioneel zijn. Daarom vertegenwoordigt het bitpatroon 00010110 in dit formaat het volgende reële getal:

$00010.110_2 = 2^1 + 2^{-1} + 2^{-2} = 2 + \frac{1}{2} + \frac{1}{4} = 2\frac{3}{4} = 2{,}75.$

Merk op dat binnen een computer een bitpatroon alles kan weergeven. Daarom vertegenwoordigt hetzelfde bitpatroon, als we het naar een ander type "casten", zoals een fixed point $\langle 8{,}5\rangle$ type, het getal:

$000.10110_2 = 2^{-1} + 2^{-3} + 2^{-4} = \frac{1}{2} + \frac{1}{4} + \frac{1}{16} = \frac{13}{16} = 0{,}6875.$

Wanneer we echter dit bitpatroon als integer beschouwen, vertegenwoordigt dit het getal:

$00010110_2 = 2^4 + 2^2 + 2^1 = 16 + 4 + 2 = 22.$

Oefeningen

a. Geef de reële getallen voor de volgende fixed point representaties: 00001001 in $\langle 8{,}4\rangle$, 10000110 in $\langle 8{,}4\rangle$, 01100101 in $\langle 8{,}3\rangle$ en 01100101 in $\langle 8{,}5\rangle$.
   De antwoorden zijn respectievelijk: $\frac{9}{16} = 0{,}5625$, $8\frac{3}{8} = 8{,}375$, $12\frac{5}{8} = 12{,}625$ en $3\frac{5}{32} = 3{,}15625$.

b. Converteer de volgende reële getallen naar het gegeven fixed point formaat: $3\frac{7}{8}$ in $\langle 8{,}4\rangle$, $14\frac{11}{16}$ in $\langle 8{,}4\rangle$, $15\frac{3}{4}$ in $\langle 8{,}4\rangle$ en $15\frac{3}{4}$ in $\langle 8{,}3\rangle$.
   De antwoorden zijn respectievelijk: 00111110, 11101011, 11111100 en 01111110.

6.3 Problemen

Het fixed point systeem ziet er waterdicht uit, maar is het niet.
Er kunnen fouten optreden in de representatie van een reëel getal.

Te onderscheiden in twee typen:

1. Het gegeven getal is exact te coderen naar binair, maar past niet binnen het formaat.
2. De noemer van de breuk is geen macht van 2.

B5 =10*B4-4,5

	A	B	C	D	E
1	X=	0,5		0,1	
2					
3	1	0,50000000000000000000000000		0,10000000000000000000000000	
4	2	0,50000000000000000000000000		0,09999999999999998000000000	
5	3	0,50000000000000000000000000		0,09999999999999975000000000	
6	4	0,50000000000000000000000000		0,09999999999997753000000000	
7	5	0,50000000000000000000000000		0,09999999999975533000000000	
8	6	0,50000000000000000000000000		0,09999999999753528000000000	
9	7	0,50000000000000000000000000		0,09999999997532840000000000	
10	8	0,50000000000000000000000000		0,09999999753283800000000000	
11	9	0,50000000000000000000000000		0,09999997532837700000000000	
12	10	0,50000000000000000000000000		0,09999975328377200000000000	
13	11	0,50000000000000000000000000		0,09999753283772300000000000	
14	12	0,50000000000000000000000000		0,09997532837723100000000000	
15	13	0,50000000000000000000000000		0,09975328377230600000000000	
16	14	0,50000000000000000000000000		0,09753283772305500000000000	
17	15	0,50000000000000000000000000		0,07532837723055200000000000	
18	16	0,50000000000000000000000000		0,07532837723055210000000000	
19	17	0,50000000000000000000000000		-0,14671622769447900000000000	
20	18	0,50000000000000000000000000		-2,36716227694479000000000000	
21	19	0,50000000000000000000000000		-24,57162276944790000000000000	
22	20	0,50000000000000000000000000		-246,61622769447900000000000000	
23					

Figuur 15: Foutje in Excel?

Fouttype I:
Het gegeven getal is exact te coderen naar binair, maar past niet binnen het formaat.
De oplossing is eenvoudig: afkappen. Alles wat buiten het formaat valt, wordt weggegooid!

Het gebroken getal $6\frac{21}{32}$ is keurig te converteren naar het fixed $\langle 8,5\rangle$ type tot 11010101. Ga dit na!
Echter als we hetzelfde getal $6\frac{21}{32}$ willen converteren naar fixed $\langle 8,4\rangle$ is er wel een probleem. Door het nieuwe formaat schuift het bitpatroon 1 positie naar rechts op. Links ontstaat een extra positie voor het hele deel, dat wordt een '0' en aan de rechte kant valt het meest rechtse bitje '1' over de rand en verdwijnt. Het kan niet anders dan dat de binaire code nu 01101010 is.
Gaan we vervolgens deze code, die dus in $\langle 8,4\rangle$ formaat staat terug rekenen, dan is dit gelijk aan $6\frac{5}{8}$.
De afwijking bedraagt hier $\frac{21}{32}-\frac{5}{8}=\frac{21}{32}-\frac{20}{32}=\frac{1}{32}\approx 3\%$.

Fouttype II:
De noemer van de breuk is geen macht van 2. De oplossing hiervoor is benaderen!
Hierbij is de hoop gevestigd op een fixed – point – formaat met flink wat bits achter de binaire punt. Hoe meer bits achter de punt, hoe beter de benadering is.

Voorbeeld: Stel we willen de breuk $\frac{1}{5}$ binair representeren.
De noemer 5 is geen macht van 2. We kunnen $\frac{1}{5}$ dus alleen benaderen.
In deze uitwerking is het uitgangspunt dat er achter de binaire punt 8 bitjes ter beschikking staan. Wat best een redelijke veronderstelling is als we er bijvoorbeeld van uitgaan dat de representatie uit 2 bytes bestaat volgens het fixed point $\langle 16,8\rangle$ formaat. Wat is dan de representatie en hoe groot is de afwijking?
Voor de breuk $\frac{1}{5}$ geldt de volgende ongelijkheid: $\frac{1}{4}>\frac{1}{5}>\frac{1}{8}$. Voor de representatie zullen we daarom starten met $\frac{1}{8}$. Waarom? Wanneer we starten met $\frac{1}{4}$, wat al te groot is, en er vervolgens nog kleine breuken bij gaan tellen, zal de afwijking ten opzichte van het te representeren getal $\frac{1}{5}$ alleen maar doen toenemen.

Poging #1:

positie	.	-1	-2	-3	-4	-5	-6	-7	-8
gewicht	.	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$	$\frac{1}{256}$
code	.	0	0	1	?	?	?	?	?

We hebben nu dus te maken met de breuk $0.001_2=\frac{1}{8}$. Hoe groot is deze afwijking ten opzichte van de breuk $\frac{1}{5}$? En kunnen we de benadering beter maken?
De afwijking is $\frac{1}{5}-\frac{1}{8}=\frac{8}{40}-\frac{5}{40}=\frac{3}{40}\approx\frac{1}{13{,}3}$. Dit komt overeen met $\approx 7{,}5\%$.
Voor deze laatste breuk geldt de ongelijkheid: $\frac{1}{8}>\frac{1}{13{,}3}>\frac{1}{16}$. De benadering kan beter door ook $\frac{1}{16}$ te gebruiken.

Poging #2:

positie	.	-1	-2	-3	-4	-5	-6	-7	-8
gewicht	.	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$	$\frac{1}{256}$
code	.	0	0	1	1	?	?	?	?

We hebben nu dus te maken met de breuk $0.0011_2 = \frac{3}{16}$. Hoe groot is deze afwijking ten opzichte van de breuk $\frac{1}{5}$? En kunnen we de benadering beter maken?
De afwijking is $\frac{1}{5} - \frac{3}{16} = \frac{16}{80} - \frac{15}{80} = \frac{1}{80}$. Dit komt overeen met $= 1{,}25\%$.
Voor deze laatste breuk geldt de ongelijkheid: $\frac{1}{64} > \frac{1}{80} > \frac{1}{128}$. De benadering kan beter door ook $\frac{1}{128}$ te gebruiken.

Poging #3:

positie	.	-1	-2	-3	-4	-5	-6	-7	-8
gewicht	.	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$	$\frac{1}{256}$
code	.	0	0	1	1	0	0	1	?

We hebben nu dus te maken met de breuk $0.0011001_2 = \frac{25}{128}$. Hoe groot is deze afwijking ten opzichte van de breuk $\frac{1}{5}$? En kunnen we de benadering beter maken?
De afwijking is $\frac{1}{5} - \frac{25}{128} = \frac{128}{640} - \frac{125}{640} = \frac{3}{640} \approx \frac{1}{213{,}3}$. Dit komt overeen met $\approx 0{,}47\%$.
De benadering kan beter door ook nog $\frac{1}{256}$ te gebruiken. Dit is dan voor deze situatie de beste benadering want meer bits achter de binaire punt zijn niet beschikbaar.

Poging #4:

positie	.	-1	-2	-3	-4	-5	-6	-7	-8
gewicht	.	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$	$\frac{1}{256}$
code	.	0	0	1	1	0	0	1	1

We hebben nu dus te maken met de breuk $0.00110011_2 = \frac{51}{256}$.
De afwijking is $\frac{1}{5} - \frac{51}{256} = \frac{256}{1280} - \frac{255}{1280} = \frac{1}{1280}$. Dit komt overeen met $\approx 0{,}08\%$. Deze afwijking lijkt klein. Als we meer bits achter de binaire punt kunnen gebruiken, dan is de afwijking verwaarloosbaar? Dat hangt af van de toepassing, van de gewenste nauwkeurigheid. In het excel-voorbeeld heb je kunnen zien dat ook een heel kleine afwijking al vrij snel tot grote fouten kan gaan leiden.

6.4 Opdrachten

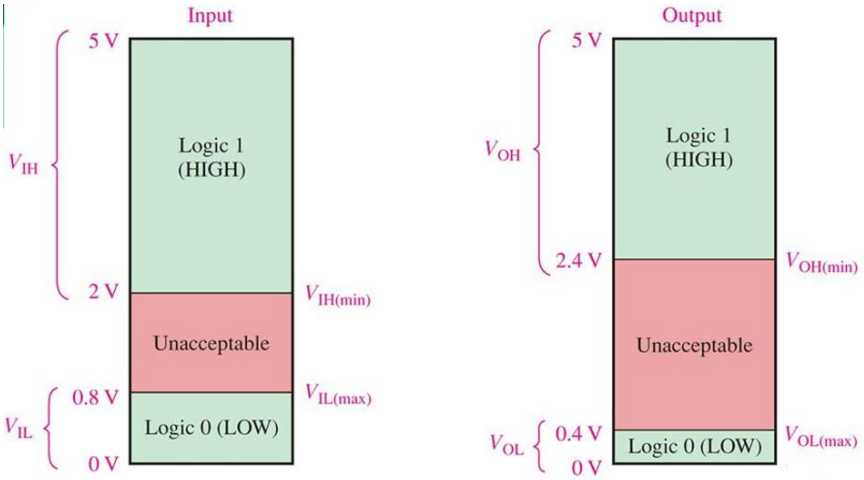
a. Codeer 7 5/16 in fixed <8,4>.
b. Codeer 7 5/16 in fixed <8,3>.
c. Codeer 560 31/1024 in fixed <32,16>.
d. Welk getal is 11001100 in fixed <8,4>?
e. Welk getal is 11001100 in fixed <8,5>?
f. Welk getal is 0000011101110001 in fixed <16,8>?

H7. Digitale poorten

Tot de jaren ’80 waren de meeste elektronische systemen analoog. Denk hierbij aan telefonie, televisie en radio. Analoog wil zeggen dat de signalen (bijvoorbeeld spanningen) in deze systemen elke mogelijke waarde tussen een bepaald minimum en maximum kunnen aannemen. Ook digitale systemen gebruiken analoge spanningen om informatie weer te geven en te bewerken. Digitale signalen zijn dus gewoon analoge spanningen, alleen doen we net alsof ze dat niet zijn.
Een digitaal signaal wordt voorgesteld als een signaal dat slechts één van twee waardes aanneemt, die we 0 en 1 noemen (of niet-waar en waar, onjuist en juist, uit en aan of . . .). Deze “digitale abstractie” stelt ons in staat om het analoge gedrag in de meeste gevallen te negeren en net te doen alsof de schakelingen echt met nullen en enen werkt. We hebben dan vooral te maken met de logische werking en het tijdgedrag.
Fabrikanten spreken trouwens liever niet over ‘0’ en ‘1’, maar over ‘L’ (laag) en ‘H’ (hoog). Het wordt dan aan de ontwerper overgelaten hoe de vertaling naar ‘0’ en ‘1’ gebeurt. In de regel vertalen we een ‘L’ naar ‘0’ en een ‘H’ naar ‘1’. Dit wordt positieve logica genoemd. Het is echter ook mogelijk om de niveaus om te draaien: een ‘L’ noemen we ‘1’ en een ‘H’ noemen we ‘0’. Dit wordt negatieve logica genoemd. In dit dictaat gebruiken we positieve logica en de notaties ‘0’ en ‘1’. We gebruiken nauwelijks de notaties ‘L’ en ‘H’. De termen ‘laag’ en ‘hoog’ komen wel voor. [17]

7.1 Spanning en logica

Nu is het heel lastig, zelfs onmogelijk, om exact één spanningswaarde te koppelen aan een logische waarde. We kunnen een logische 1 bijvoorbeeld voorstellen met 5 V, maar hoe reageert een schakeling als een spanning van 4,9 V of 5,1 V wordt aangeboden? In de praktijk worden daarom voor de spanning bereiken gebruikt. Een logische waarde wordt dan aangegeven met een minimale en maximale spanning. In figuur 16 is een schematische weergave gegeven van de interpretatie van ingangs- en uitgangsspanningen. De schakeling interpreteert een ingangsspanning als een logische 0 als de aangeboden spanning ligt tussen de referentie (ground of aarde) en een zekere maximale spanning. In de figuur is dat links te zien in gebied ’0’. Voor een logische 1 geldt dat er minimaal een bepaalde spanning moet worden aangeboden, te zien in het gebied ’1’. De maximaal aan te bieden spanning is de voedingsspanning (in de praktijk mag dat meestal iets erboven liggen).



Figuur 16: Interpretatie van logische 0 en 1. [18]

Te zien is dat de gebieden voor een logische 0 en 1 niet aaneengesloten zijn. Er is een gebiedje waarin de spanning als een ongeldig logische waarde wordt gezien. Dit zorgt ervoor dat de interpretatie van de waarden voor 0 en 1 duidelijk gerealiseerd kunnen worden (denk eraan dat elke digitale schakeling in wezen analoog is). Aan de kant van de uitgang worden 0 en 1 ook gerealiseerd

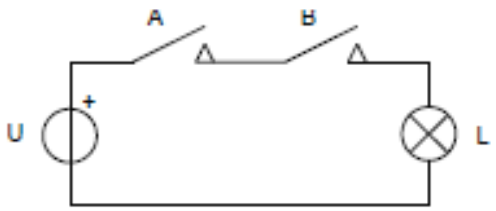
met spanningsgebieden. Een logische 0 ligt tussen de referentie en een zekere maximale spanning. Een logische 1 ligt tussen een zeker minimum en de voedingsspanning. Ook tussen deze twee gebieden is enige ruimte, de uitgangsspanning zal zich nooit in dit gebied bevinden.

In statische toestand zullen alle spanningen zich in een van de twee gedefinieerde gebieden bevinden. In de praktijk kan een spanning nooit direct van het ene gebied naar het andere gebied gaan. Dit kost enige tijd. Er worden aan de dynamische toestand eisen gesteld aan de stijg- en daaltijden van de signalen (Engels: rise time en fall time). Een ingangssignaal moet aan deze tijden voldoen anders kan de schakeling kuren vertonen.
We mogen er trouwens niet vanuit gaan dat de schakeling in dynamische toestand de bedoelde werking heeft. Die is alleen voor de statische toestand gedefinieerd. Fabrikanten geven in hun datasheets de spanningsniveaus en stromen aan.

7.2 Logische schakelingen

Het gebruik van tweewaardige, binaire of logische schakelingen in de digitale techniek is het gevolg van de eenvoud van de schakelingen. Signalen kunnen slechts twee waarden aannemen, bijvoorbeeld aan en uit, actief en niet-actief, maar voor ons doel is het handiger om 0 en 1 te gebruiken. Digitale schakelingen worden verreweg het meest ontworpen met elektronische componenten. De reden hiervoor is dat ze klein, snel en goedkoop zijn. In andere vakgebieden worden pneumatische componenten (lucht), hydraulische componenten (vloeistof) en fotonische componenten (licht) gebruikt.
In de begintijd van de digitale techniek werden de schakelingen opgebouwd uit discrete componenten zoals schakelaars, weerstanden, diodes en transistoren. De schakelingen waren over het algemeen best groot en de complexiteit was laag. De ontwerper was naast het logisch ontwerp een groot deel van zijn tijd bezig om alle elektronica goed te dimensioneren (het berekenen van de componentwaarden).

De eenvoudigste component is een schakelaar die twee standen heeft. In rust is de schakelaar open. In de actieve stand is de schakelaar gesloten. We noemen zo'n schakelaar 'normally open' (NO). We kunnen de schakelaars vertegenwoordigen door variabelen die slechts twee waarden kunnen aannemen, 0 en 1. We noemen deze variabelen binaire of logische variabelen (Java, datatype boolean). Een schakelaar die open is geven we aan met een 0, een gesloten schakelaar geven we aan met een 1. Op eenzelfde wijze kunnen we een lamp in de schakeling opnemen. Een lamp die gedoofd is geven we aan met een 0, een lamp die brandt geven we aan met een 1.
Figuur 17 beeldt een serieschakeling van twee schakelaars uit. De schakelaars zijn in rust getekend.



Figuur 17: Een logische AND-schakeling.

In deze situatie brandt de lamp niet. We geven dit aan door $A = 0, B = 0$ en $L = 0$. Nu kunnen we schakelaar B sluiten, dit geven we aan met $B = 1$. Schakelaar A is nog steeds geopend. Ook nu brandt de lamp niet. Deze situatie geven we aan met $A = 0, B = 1$ en $L = 0$. In de derde situatie is schakelaar A gesloten en B geopend. De lamp brandt nog steeds niet. We geven dit aan met $A = 1, B = 0$ en $L = 0$. Het is duidelijk dat de lamp pas brandt als beide schakelaars gesloten zijn. Dit geven we aan met $A = 1, B = 1$ en $L = 1$.

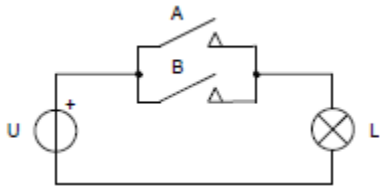
We kunnen de vier situaties in een waarheidstabel weergeven. Zie hiervoor tabel ?. In deze tabel worden de toestanden van de schakelaars en de lamp met een 0 of een 1 aangegeven. Elke rij in de tabel geeft één combinatie van toestanden van de schakelaars aan. Aan elke rij kunnen we een betekenis koppelen.
Tabel 1.4: Waarheidstabel serieschakeling (AND).
A B L betekenis

A	*B*	*L*	*betekenis*
0	0	0	schakelaars open, lamp uit
0	1	0	schakelaar A open, B gesloten, lamp uit
1	0	0	schakelaar A gesloten, B open, lamp uit
1	1	1	schakelaars gesloten, lamp aan

Tabel 6: Waarheidstabel serieschakeling (AND).

Het gedrag van de schakeling kunnen we beschrijven met een logische functie: $L = A \cdot B$ waarbij geldt dat $L = 1$ als $A = 1$ én $B = 1$, anders is $L = 0$. We noemen de "·" de AND-operator. De formule beschrijft het gedrag van de schakeling in figuur 17. De "·" moet niet verward worden met een rekenkundige vermenigvuldiging. We kunnen de punt ook weglaten en dan schrijven we: $L = AB$.

In figuur 18 is een parallelschakeling van twee schakelaars te zien. In rust (schakelaars A en B zijn beide 0) brandt de lamp niet (L = 0). Als een van de twee schakelaars gesloten is óf als beide schakelaars gesloten zijn, zal de lamp branden. Ook hiervan is een waarheidstabel te maken, tabel 7.



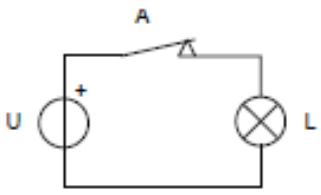
Figuur 18: De logische OR-schakeling.

A	*B*	*L*	*betekenis*
0	0	0	schakelaars open, lamp uit
0	1	1	schakelaar A open, B gesloten, lamp aan
1	0	1	schakelaar A gesloten, B open, lamp aan
1	1	1	schakelaars gesloten, lamp aan

Tabel 7: Waarheidstabel parallelschakeling (OR).

Het gedrag van de parallelschakeling is te beschrijven met de logische functie: $L = A + B$ waarbij geldt dat $L = 1$ als $A = 1$ óf $B = 1$ óf $A = B = 1$, anders is $L = 0$. De "+" wordt de OR-operator genoemd en de formule beschrijft het gedrag van de schakeling in figuur 18. De "+" moet niet verward worden met de rekenkundige optelling.

De hier gepresenteerde schakelaars zijn in ruststand geopend. Er zijn ook schakelaars die in ruststand gesloten zijn. Dit type wordt 'normally closed' (NC) genoemd. Daarom spreken we liever niet van open en gesloten maar van een ruststand en actieve stand.



In figuur 19 is een schakeling getekend met een normally closed schakelaar. In de ruststand is de schakelaar gesloten en brandt de lamp. Dit geven we aan met $A = 0$ en $L = 1$. Als de schakelaar geactiveerd wordt, zal de lamp doven. Dit geven we aan met $A = 1$ en $L = 0$. Ook van deze schakeling is een waarheidstabel op te stellen, zie tabel 8.

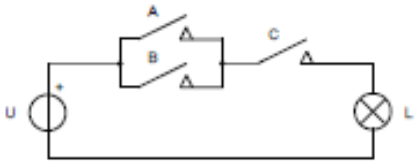
Figuur 19: Schakeling voor inverse-schakeling (NOT).

A	*L*	*betekenis*
0	1	schakelaar gesloten, lamp aan
1	0	schakelaar open, lamp uit

Tabel 8: Waarheidstabel inverse-schakeling (NOT).

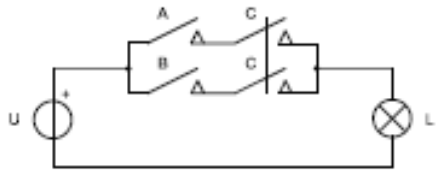
Het gedrag van deze schakeling is te beschrijven met de logische functie: $L = \bar{A}$
waarbij geldt dat $L = 1$ als $A = 0$, anders is $L = 0$. De waarde van deze functie is de inverse van de ingangswaarde. Een veelgebruikte (Engelse) term is complement. Een andere veelgebruikte term is de NOT-operatie. Er zijn diverse notatiewijzen voor de NOT-functie.
In de bovenstaande functie is een streepje boven A geplaatst (de overbar). Deze notatie is waarschijnlijk het beste vanuit het visuele oogpunt. Maar bij het gebruik van computerprogrammatuur is de invoer van het streepje onpraktisch. Andere notaties zijn onder andere: $L = \bar{A} = {\sim}A = !\,A = A' = \neg A$.

Met de functies AND, OR en NOT is het mogelijk om elke logische functie te implementeren.
Figuur 20 toont hoe drie schakelaars kunnen worden gebruikt om het licht te controleren op een meer complexe manier. Deze serie-parallelschakeling van schakelaars realiseert de logische functie (haakjes geven bewerkingsvolgorde aan): $L = (A + B) \cdot C$.
Het licht brandt als $C = 1$ en tegelijkertijd ten minste één van de schakelaars A of B gelijk is aan 1.



Figuur 20: Een gecombineerde serie- en parallelschakeling.

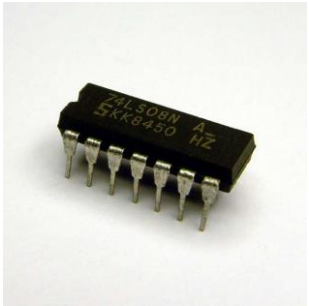
In figuur 21 is nog een schakeling gegeven. In dit schema komt de schakelaar C twee keer voor. Deze schakelaars zijn op een of andere manier mechanisch gekoppeld zodat bij bediening beide schakelaars tegelijk van stand veranderen. De logische functie die deze schakeling realiseert is:
$L = (A \cdot C) + (B \cdot C)$.
De lamp brandt als schakelaars A en C óf B en C geactiveerd zijn.



Figuur 21: Een gecombineerde serie- en parallelschakeling.

7.3 Symbolen voor logische poorten

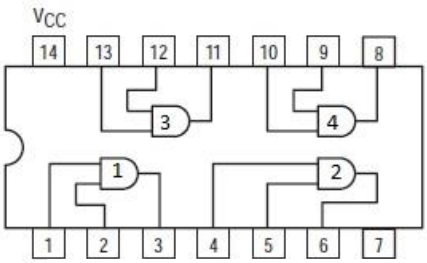
Midden jaren ’60 van de vorige eeuw kregen ontwerpers de beschikking over Integrated Circuits (ic’s of chips) met daarin enkele poorten. Een poort (Engels: gate) is een elektronische schakeling die een bepaalde logische functie vervult. Een voorbeeld daarvan is te zien in figuur 22. De foto toont een “7408 Quadruple 2-input AND gate” uit de beroemde TTL-serie [19]. Het ic bevat vier AND-poorten met drie aansluitingen elk. Daarnaast zijn nog twee voedingsspanningsaansluitingen nodig zodat het totaal op veertien aansluitingen komt. De pinaansluitingen zijn te zien in figuur 23.



Het voordeel van deze ic’s was dat ze kleiner waren dan schakelingen die met discrete componenten waren opgebouwd en de ontwerper hoefde zich minder druk te maken over de dimensionering van de schakeling. Er is een hele reeks van ic’s gemaakt, meer dan 100 stuks, waarvan de typenummers allemaal beginnen met 74. Zo zijn er ic’s met eenvoudige poorten, maar ook met complexere schakelingen.

Figuur 22: 7408 AND gate [20]

Elke digitale schakeling is te realiseren met de drie basisoperaties AND, OR en NOT. Een complexe schakeling vereist meerdere van deze operaties.

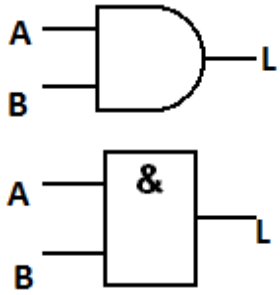


Figuur 23: Pinaansluitingen.[21]

Het is handig om een logische schakeling weer te geven met een schakelschema (of schema) met daarin grafische symbolen voor de poorten die de operaties voorstellen. In het vervolg van deze paragraaf behandelen we de logische poorten. Van elke poort wordt de werking beschreven in termen van de logische waarden 0 en 1. Daarnaast worden de grafische symbolen gegeven. Deze symbolen zijn gedefinieerd in de normbladen voor IEC/IEEE-symbolen [22, 23]. In de Engelstalige literatuur en CAD-tekenpakketten wordt echter veelvuldig gebruik gemaakt van de "distinctive shape" symbolen (vormsymbolen) die zijn afgeleid van de MIL-STD-806 [24]. In dit dictaat gebruiken we de vorm-symbolen.

7.3.1 AND-poort

De uitgang van een AND-poort wordt logisch 1 als beide ingangen logisch 1 zijn, anders wordt de uitgang logisch 0. De symbolen zijn te vinden in figuur ???, bovenin het vormsymbool, onderin het IEC/IEEE-symbool. In tabel 9 is de waarheidstabel gegeven.
De logische functie is: $L = A \cdot B = AB$.



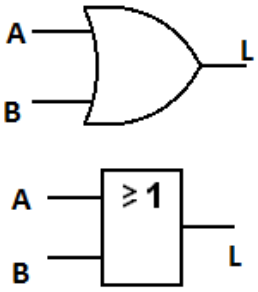
Figuur 24: Vormsymbool en IEC/IEEE symbool AND [25]

A	*B*	*L*
0	0	0
0	1	0
1	0	0
1	1	1

Tabel 9: Waarheidstabel AND-functie

7.3.2 OR-poort

De uitgang van een OR-poort wordt logisch 1 als een of meer ingangen logisch 1 is of zijn, anders wordt de uitgang logisch 0. De symbolen zijn te vinden in figuur 25, bovenin het vormsymbool, rechts het IEC/IEEE-symbool. In tabel 10 is de waarheidstabel gegeven. De logische functie is: $L = A + B$.



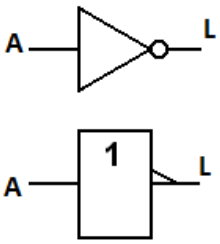
Figuur 25: Vormsymbool en IEC/IEEE symbool OR [26]

A	*B*	*L*
0	0	0
0	1	1
1	0	1
1	1	1

Tabel 10: Waarheidstabel OR-functie

7.3.3 NOT-poort

De uitgang van een NOT-poort (of '*inverter*') wordt logisch 1 als de ingang logisch 0 is, anders wordt de uitgang logisch 0. De symbolen zijn te vinden in figuur 26, bovenin het vormsymbool, onderin het IEC/IEEE-symbool. In tabel 11 is de waarheidstabel gegeven. De logische functie is: $L = \bar{A}$.



Figuur 26: Vormsymbool en IEC/IEEE symbool NOT [27]

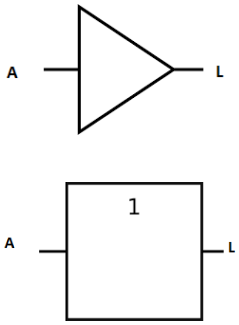
A	*L*
0	1
1	0

Tabel 11: Waarheidstabel NOT-functie

Hiervoor zijn de drie basispoorten besproken. Hiermee kan elke digitale schakeling worden ontworpen en gebouwd. In de praktijk worden nog andere poorten gebruikt. Deze poorten zijn niet strikt noodzakelijk, wel heel handig. Ze worden gebruikt bij veel voorkomende bewerkingen. Al deze poorten kunnen worden opgebouwd uit AND, OR en NOT. We behandelen nog de buffer, EXOR, NAND, NOR en EXNOR.

7.3.4 Buffer

De buffer geeft de ingangswaarde één op één door. Het heeft geen "echte" logische werking. Buffers worden gebruikt om de uitgangsstroom van een schakeling te vergroten of om een tijdvertraging te introduceren. De symbolen zijn te vinden in figuur 27, eerst weer het vormsymbool. De waarheidstabel is te vinden in tabel 12.



Figuur 27:Vormsymbool [28] en IEC/IEEE symbool [29] BUFFER.

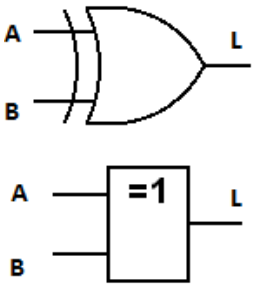
A	*L*
0	0
1	1

Tabel 12: Waarheidstabel BUFFER-functie.

De logische functie is: $L = A$.

7.3.5 EXOR-poort

De uitgang van een EXOR-poort wordt logisch 1 als exact een van de ingangen logisch 1 is, anders wordt de uitgang logisch 0. Anders gezegd: de uitgang wordt logisch 1 als de ingangen ongelijk aan elkaar zijn, anders 0. De symbolen zijn te vinden in figuur 28, bovenin het vormsymbool, onderin het IEC/IEEE-symbool. In tabel 13 is de waarheidstabel gegeven. EXOR-poorten spelen een grote rol bij rekenschakelingen.



Figuur 28: Vormsymbool en IEC/IEEE symbool EXOR [30].

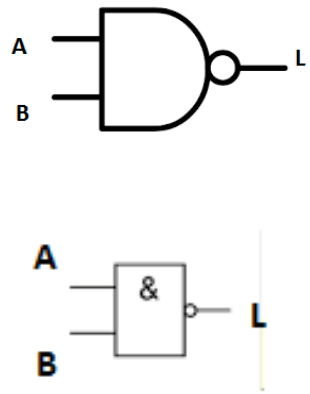
A	*B*	*L*
0	0	0
0	1	1
1	0	1
1	1	0

Tabel 13: Waarheidstabel EXOR-functie

De logische functie is: $L = A \oplus B$.

7.3.6 NAND-poort

NAND is een samentrekking van NOT en AND. De uitgang van een NAND-poort wordt logisch 0 als beide ingangen logisch 1 zijn, anders wordt de uitgang logisch 1. De symbolen zijn te vinden in figuur 29, bovenin het vormsymbool, onderin het IEC/IEEE-symbool. In tabel 14 is de waarheidstabel gegeven.
De logische functie is: $L = \overline{A \cdot B} = \overline{AB}$.



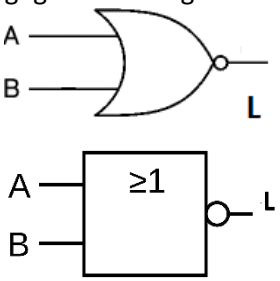
Figuur 29: Vormsymbool [31] en IEC/IEEE symbool [32] NAND.

A	*B*	*L*
0	0	1
0	1	1
1	0	1
1	1	0

Tabel 14: Waarheidstabel NAND-functie

7.3.7 NOR-poort

NOR is een samentrekking van NOT en OR. De uitgang van een NOR-poort wordt logisch 0 als een of meer ingangen logisch 1 is of zijn, anders wordt de uitgang logisch 1. De symbolen zijn te vinden in figuur 30, bovenin het vormsymbool, onderin het IEC/IEEE-symbool. In tabel 15 is de waarheidstabel gegeven. De logische functie is: $L = \overline{A + B}$.



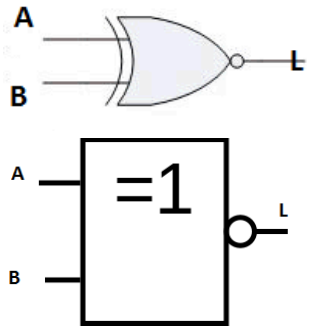
Figuur 30: Vormsymbool [33] en IEC/IEEE symbool [34] NOR.

A	*B*	*L*
0	0	1
0	1	0
1	0	0
1	1	0

Tabel 15: Waarheidstabel NOR-functie

7.3.8 EXNOR-poort

EXNOR is een samentrekking van NOT en EXOR. De uitgang van een EXNOR-poort wordt logisch 0 als exact één van de ingangen logisch 1 is, anders wordt de uitgang logisch 1. Anders gezegd: de uitgang wordt logisch 1 als de ingangen gelijk zijn aan elkaar, anders is de uitgang logisch 0. De EXNOR-poort wordt ook wel de gelijkheidspoort (Engels: equivalence gate) genoemd. De symbolen zijn te vinden in figuur 31, bovenin het vormsymbool, onderin het IEC/IEEE-symbool. In tabel 16 is de waarheidstabel gegeven. EXNOR-poorten spelen een grote rol bij rekenschakelingen.
De logische functie is: $L = \overline{A \oplus B}$.

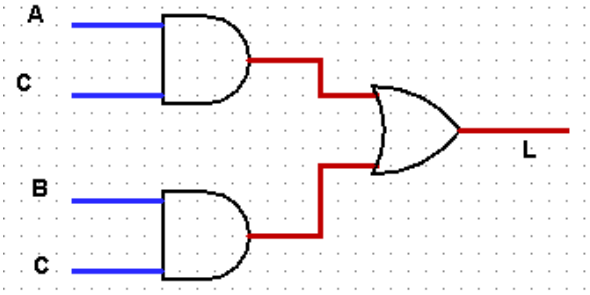


A	*B*	*L*
0	0	1
0	1	0
1	0	0
1	1	0

Tabel 16: Waarheidstabel EXNOR-functie

Figuur 31: Vormsymbool [35] en IEC/IEEE symbool [36] EXNOR.

Met behulp van de symbolen kunnen we een schakeling grafisch weergeven. In figuur 32 is het schema te zien van een schakeling die de logische functie van ??? vervult. In hoofdstuk 8 gaan we hier dieper op in.



Figuur 32: Schema AND-OR-schakeling met vormsymbolen gemaakt in Logisim.

We hebben drie manieren om een schakeling te beschrijven:

- waarheidstabel,
- logische functie, en
- schema (tekening).

Deze drie vormen geven geen informatie over de elektrische eigenschappen en het tijdgedrag. Beide laatste onderwerpen vallen buiten het bestek van deze introductiecursus.

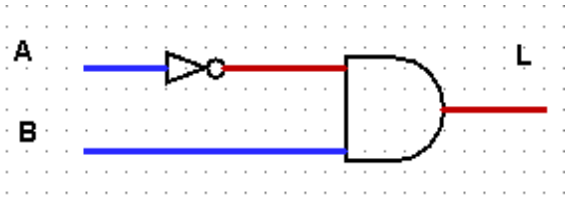
7.4 Logische schakelingen

Vanuit een waarheidstabel of logische functie kan een schakeling worden opgebouwd. In deze paragraaf behandelen hoe we een schakeling opbouwen. We gebruiken positieve logica en gaan hierbij uit van het principe van “enen sparen”. Door een logische functie te vereenvoudigen kan een schakeling met een minimum aan poorten worden gerealiseerd. We introduceren het begrip dualiteit dat te maken met het uitwisselen van AND- en OR-poorten.

In tabel 17 is de waarheidstabel van een functie te zien. De functie moet een logische 1 geven voor alle enen in de L-kolom, anders moet de functie een logische 0 geven. We gaan hierbij uit van positieve logica. De enige regel met een 1 lezen we als volgt: $L = 1$ als $A = 0$ én $B = 1$. We kunnen dit in een functie schrijven: $L = \bar{A} \cdot B = \bar{A}B$. Als we de overige drie combinaties invullen wordt L inderdaad logisch 0.

A	*B*	*L*
0	0	0
0	1	1
1	0	0
1	1	0

Tabel 17: waarheidstabel van een functie.



Figuur 33: Schema van een functie.

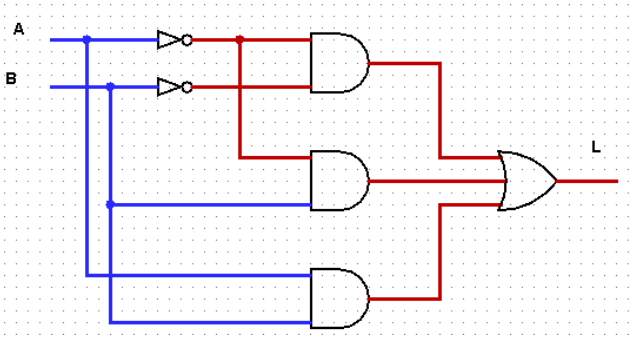
Om de functie te realiseren, hebben we een NOT-poort en een AND-poort nodig, zie figuur 33. De AND-poort geeft een logische 1 als beide ingangen logisch 1 zijn. Maar de functie moet een logische 1 geven als A logisch 0 is en B logisch 1 is. We hebben dus een NOT-poort nodig om A te inverteren. Zo zorgen we ervoor dat de AND-poort een logische 1 geeft als A logisch 0 is en B logisch 1. Voor de overige combinaties van A en B geeft de schakeling een logische 0. Controleer dit!

7.4.1 Vereenvoudigen

Stel dat we de schakeling willen bouwen zoals is aangegeven in de waarheidstabel in tabel 18. De werking van de schakeling is als volgt: uitgang $L = 1$ als $A = 0$ en $B = 0$ óf $A = 0$ en $B = 1$ óf $A = 1$ en $B = 1$, anders $L = 0$. De logische functie is: $L = \bar{A} \cdot \bar{B} + \bar{A} \cdot B + A \cdot B = \bar{A}\bar{B} + \bar{A}B + AB$.
In figuur 34 is de schakeling te zien. Dit kost nogal wat poorten: twee NOT-poorten, drie AND-poorten en een OR-poort. Het aantal ingangen is 11. Merk op dat we voor de inverse van A maar één NOT-poort gebruiken.

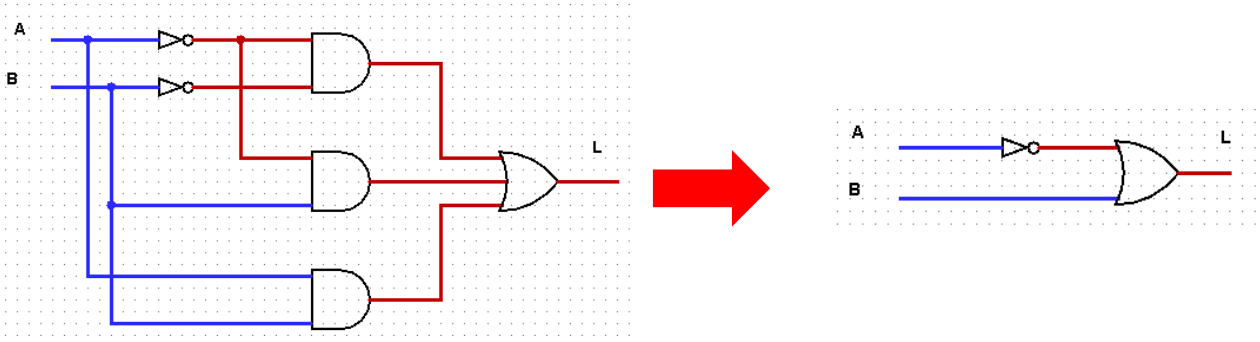
A	*B*	*L*
0	0	1
0	1	1
1	0	0
1	1	1

Tabel 18: Waarheidstabel van de te vereenvoudigen functie.



Figuur 34: Circuit van de te vereenvoudigen functie.

We zien dat de functie een logische 1 oplevert als $A = 0$. Dat is bij de eerste twee regels in tabel 18. Dus we kunnen schrijven $L = 1$ als $A = 0$. Ook is te zien dat de uitgang een logisch 1 is als $B = 1$, zie regel2 en regel4 in tabel 18. Dus: $L = 1$ als $B = 1$. Verder geldt dat $L = 0$ als $A = 1$ en $B = 0$. We kunnen de functie nu vereenvoudigen tot $L = 1$ als $A = 0$ of $B = 1$ anders geldt $L = 0$.
Dus: $L = \bar{A} \cdot \bar{B} + \bar{A} \cdot B + A \cdot B = \bar{A}\bar{B} + \bar{A}B + AB = \bar{A} + B$.
We kunnen nu de schakeling met veel minder poorten en aansluitingen realiseren, zie figuur 35.

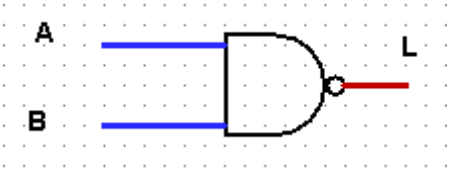


Figuur 35: Vereenvoudigd circuit.

Merk op dat beide schakelingen logisch identiek zijn. Ze vervullen dezelfde logische functie.

7.4.2 Dualiteit en De Morgan

Het begrip dualiteit kunnen we het beste uitleggen aan de hand van een voorbeeld. In figuur 36 is een schakeling rond een NAND-poort te zien.



Figuur 36: NAND-poort

De logische functie is: $L = \overline{A \cdot B} = \overline{AB}$.
Met als waarheidstabel:

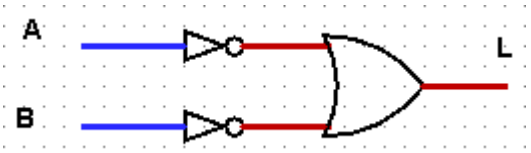
A	B	$L = \overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

Tabel 19: NAND-functie

In tabel 20 lees je een uitbreiding van tabel 19. Je kunt hierin zien dat de functie $\overline{A \cdot B}$ een logisch equivalent heeft, namelijk $\bar{A} + \bar{B}$. Deze gelijkheid staat bekend als de wet van De Morgan. Figuren 36

A	B	$L = \overline{AB}$	$\bar{A}$	$\bar{B}$	$\bar{A} + \bar{B}$
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

Tabel 20: Wet van De Morgan



Figuur 37: OR-poort met twee ontkennende ingangen.

en 37 zijn twee verschillende schema's die dus wel hetzelfde logische gedrag vertonen. Andersom geldt het ook, $\overline{A + B} = \bar{A} \cdot \bar{B}$.

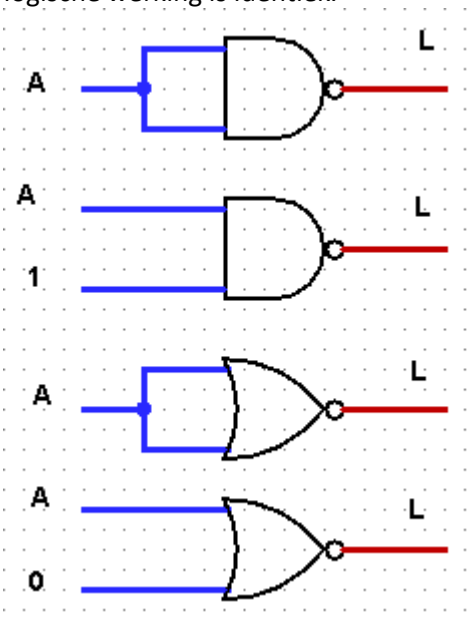
In de Boole algebra kent men het begrip 'duale vorm' van een formule. De duale vorm van een formule ontstaat uit de formule door de volgende bewerkingen:

- vervang $+$ door $\cdot$ en $\cdot$ door $+$;
- vervang 1 door 0 en 0 door 1;
- handhaaf de volgorde van de oorspronkelijke bewerkingen. [37]

7.4.3 Universele bouwstenen

De NOT-AND-poort of NAND-poort is een universele digitale bouwsteen. Elke digitale schakeling kan met alleen NAND-poorten worden gemaakt. Dat geldt ook voor de NOT-OR-poort of NOR-poort. Ook

deze poort is een universele digitale bouwsteen. Elke digitale schakeling kan met alleen NOR-poorten worden gemaakt.
Om dit aan te tonen, moeten we laten zien dat de drie basispoorten kunnen worden gerealiseerd met NAND of NOR-poorten. In figuur 38 zijn de vier configuraties te zien van NAND- en NOR-poorten geschakeld als NOT-poorten. Bij elke configuratie is de wiskundige beschrijving gegeven. Elk van de vier schakelingen realiseert een NOT-functie. Het maakt niet uit welke gebruikt wordt want de logische werking is identiek.



Figuur 38: Vier configuraties

A	$\bar{A}$	$A \cdot A$	$\overline{A \cdot A}$
0	1	0	1
1	0	1	0

$$L = \overline{A \cdot A} = \bar{A}$$

A	$\bar{A}$	$A \cdot 1$	$\overline{A \cdot 1}$
0	1	0	1
1	0	1	0

$$L = \overline{A \cdot 1} = \bar{A}$$

A	$\bar{A}$	$A + A$	$\overline{A + A}$
0	1	0	1
1	0	1	0

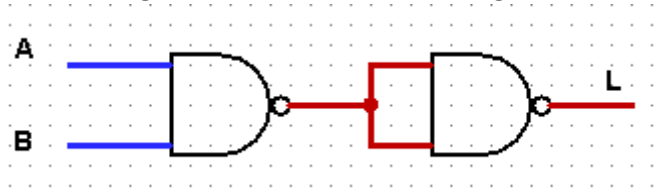
$$L = \overline{A + A} = \bar{A}$$

A	$\bar{A}$	$A + 0$	$\overline{A + 0}$
0	1	0	1
1	0	1	0

$$L = \overline{A + 0} = \bar{A}$$

Tabel 21: Vier waarheidstabellen voor de NOT-functie.

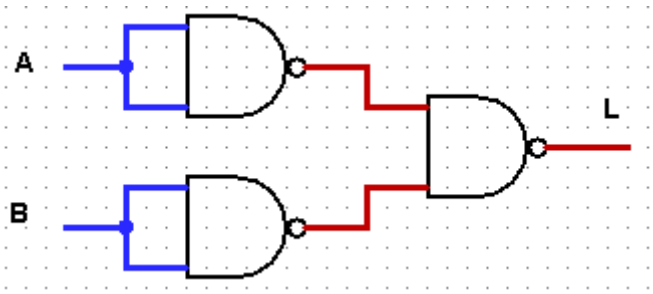
De schakeling in figuur 39 realiseert een AND-poort. De linker NAND-poort realiseert de NAND-functie van A en B. De rechter NAND-poort is geschakeld als NOT-poort. In totaal realiseert de schakeling dus een NOT-NAND-poort. In de wiskundige beschrijving is goed te zien dat de termen $\overline{A \cdot B}$ samengenomen worden. Daarna volgt de dubbele negatie.



$$L = \overline{\overline{A \cdot B} \cdot \overline{A \cdot B}} = \overline{\overline{A \cdot B}} = A \cdot B$$

Figuur 39: Realisatie van AND m.b.v. NAND.

In figuur 40 is een schakeling te zien die een OR-poort realiseert met drie NAND-poorten. De NAND-poorten links inverteren de ingangen, de NAND-poort rechts realiseert de NAND-constructie van de geïnverteerde ingangen. In de wiskundige beschrijving is te zien dat eenmaal de stelling van De Morgen is toegepast.



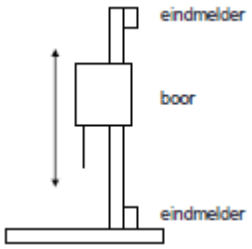
$$L = \overline{\overline{A} \cdot \overline{A} \cdot \overline{B} \cdot \overline{B}} = \overline{\overline{A} \cdot \overline{B}} = \overline{\overline{A + B}} = A + B$$

Figuur 40: Realisatie van OR m.b.v. NAND.

Uit bovenstaande tekst en figuren 38, 39 en 40, blijkt derhalve dat de drie basisfunctionaliteiten NOT, AND en OR allen kunnen worden gerealiseerd met behulp van NAND-poorten. Daarom wordt een NAND-poort volledig functioneel genoemd. Dit zelfde geldt voor de NOR-poort. Het bewijs hiervan wordt verder overgelaten aan de lezer (zie ook de opdrachten in 7.5). Waarom het nuttig is om circuits met uitsluitend NAND-poorten of NOR-poorten te maken, komt in een vervolg cursus aan bod.

7.5 Opdrachten

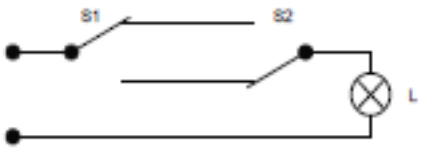
1. Ontwerp een omschakelbare buffer/NOT. Dit is een schakeling die onder besturing van een stuursignaal de ingangswaarde doorgeeft (buffer) of inverteert (NOT).
2. Ontwerp een NOT-schakeling met een NAND-poort.
3. Ontwerp een NOT-schakeling met een NOR-poort.
4. Bepaal de waarheidstabel van een EXOR waarvan één ingang geïnverteerd is.
5. De beschrijving van een AND is: de uitgang is 1 als alle ingangen 1 zijn. Hoe zou de beschrijving zijn als er van de nullen wordt uitgegaan?
6. Hoeveel verschillende functies zijn er te maken met twee variabelen?
7. Een automatische kolomboor (figuur 33) heeft twee eindmelders: een aan de bovenkant en een aan de onderkant. Een melder kan open zijn (boor is daar niet) of gesloten zijn (boor is daar wel).



Figuur 41: Kolomboor.

Hoeveel combinaties van open en gesloten zijn er mogelijk? Welke combinatie komt nooit voor? Stel een tabel op met alle mogelijkheden en geef met een paar woorden aan wat de mogelijkheden inhouden.

8. Een elektrisch schema met schakelaars kan ook als een digitaal systeem worden gezien. Een bekende schakeling is de zogeheten wisselschakeling die veel bij trappen voorkomt. Deze schakeling wordt ook wel de hotelschakeling (figuur 36) genoemd.

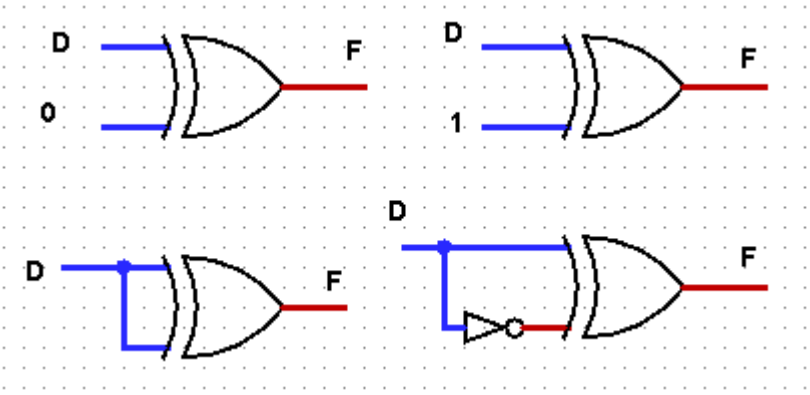


schakelaar in rust: 0
schakelaar geactiveerd: 1
lamp uit: 0
lamp brandt: 1

Figuur 42: Hotelschakeling.

Geef de waarheidstabel van de wisselschakeling.

9. Aan een EXOR-poort met twee ingangen wordt één ingang als volgt afwisselend aangesloten: 0, 1, normaal of geïnverteerd. Zie hiervoor figuur 43.



Figuur 43: Vier implementaties van een EXOR-poort.

Hieruit is op te maken dat er slechts één ingangssignaal is, namelijk D. Daardoor vereenvoudigd de werking van de poort. Bepaal de werking van deze vier mogelijkheden.

10. Als opgave 9 maar nu met een EXNOR-poort.

11. Bepaal de eenvoudigste vorm van de functies in tabel ???.

A	*B*	*L*
0	0	1
0	1	1
1	0	1
1	1	0

A	*B*	*L*
0	0	1
0	1	0
1	0	1
1	1	1

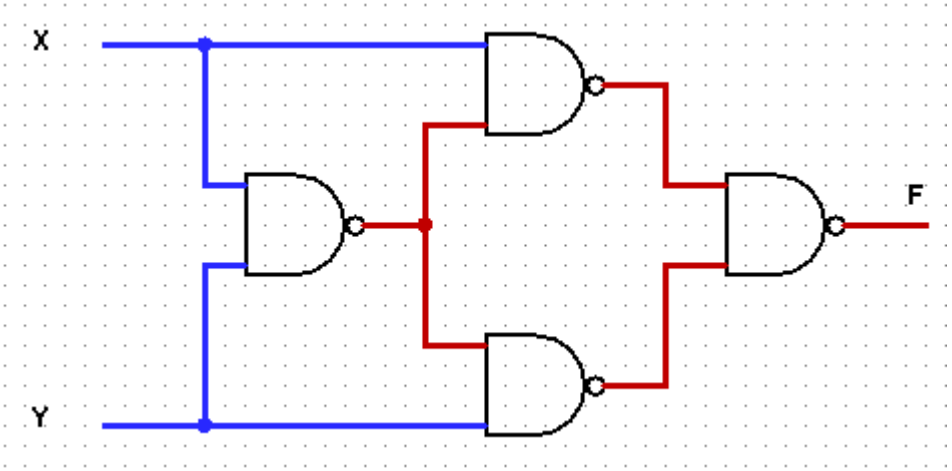
A	*B*	*L*
0	0	0
0	1	1
1	0	1
1	1	0

Tabel 22: Drie waarheidstabellen.

12. Ontwerp een buffer met behulp van NAND's.

13. Gegeven de functie: $L = \bar{A} \cdot B + A \cdot \bar{B}$
Ontwerp deze functie met alleen NAND's.

14. Gegeven de schakeling in figuur 44. Bepaal F als functie van X en Y.



Figuur 44: Circuit voor functie F.

H8. Combinatorische schakelingen

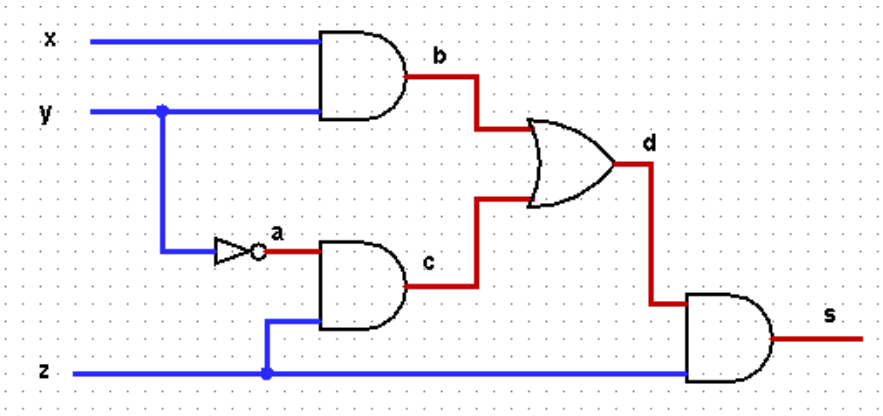
In het vorige hoofdstuk hebben we de logische poorten en een klein stukje schakelalgebra behandeld. We hebben nu genoeg gereedschap om combinatorische schakelingen te ontwikkelen. De uitgangswaarden van combinatorische schakelingen zijn alleen afhankelijk van de huidige ingangswaarden en niet van een voorafgaande reeks ingangswaarden. Combinatorische schakelingen hebben geen geheugenwerking.

In dit hoofdstuk gaan we de schakelalgebra en poorten gebruiken om schakelingen te analyseren. Realiseren van de functie kan met de bekende poorten AND, OR en NOT maar het is ook mogelijk om een schakeling met alleen NAND- of NOR-poorten te bouwen. Om de theorie te ondersteunen werken we een aantal voorbeelden uit. We beginnen met een geschreven specificatie en werken dit uit tot een schakeling met poorten. Daarbij wordt een aantal stappen doorlopen: opstellen van een waarheidstabel, minimaliseren van de logische functies, eventueel functies omwerken naar NAND- of NOR-vorm en tekenen van de schakeling met poorten. [39]

8.1 Analyse van combinatorische schakelingen

Met analyse bedoelen we het bepalen van de schakelfunctie van een schakeling met logische poorten. Het doel is te achterhalen hoe de schakeling reageert op de ingangscombinaties. Naderhand kunnen we de resultaten gebruiken om een uitspraak te doen over de uitgangswaarde bij een bepaalde ingangscombinatie of om de schakeling op een andere manier te realiseren, bijvoorbeeld met minder poorten. Het bepalen van de functie kan op twee manieren: met behulp van de schakelalgebra en door middel van waarheidstabellen.

In figuur 45 is een schakeling gegeven met de ingangen x, y en z en uitgang s. De schakeling bestaat uit vijf poorten. Verder zijn in de schakeling de interne signalen a, b, c en d opgenomen. Deze zijn niet strikt noodzakelijk maar vergemakkelijken het analyseproces.



Figuur 45: Schakeling met poorten.

De functies van de poorten zijn bekend. Van elk van de interne signalen en de uitgang kunnen we de functie bepalen uitgedrukt in de ingangssignalen en de interne signalen.
Zo is a de inverse van y en d de logische som van b en c. In onderstaande vergelijkingen zijn alle functies gegeven:

$a = \bar{y}$
$b = x \cdot y$
$c = a \cdot z$
$d = b + c$
$s = d \cdot z$

Vervolgens werken we de functie uit door de interne variabelen te vervangen door hun functie totdat de functie s volledig is beschreven door de ingangen (algebraïsche substitutie):

$$s = d \cdot z$$
$$= (b + c) \cdot z$$
$$= (x \cdot y + a \cdot z) \cdot z$$
$$= (x \cdot y + \bar{y} \cdot z) \cdot z$$

We hebben nu een, wat later zal blijken, niet-vereenvoudigde schakelfunctie verkregen. Het vinden van een uitgangswaarde voor een specifieke ingangscombinatie kost toch nog enige moeite. Voor $xyz = 101$ vinden we:

$$s = (1 \cdot 0 + \bar{0} \cdot 1) \cdot 1 = (1 \cdot 0 + 1 \cdot 1) \cdot 1 = (0 + 1) \cdot 1 = 1 \cdot 1 = 1$$

We kunnen een schakelfunctie ook bepalen door middel van het opstellen van een waarheidstabel. In de waarheidstabel nemen we alle signalen op. Links plaatsen we de ingangen x, y en z, gevolgd door de interne signalen a, b, c en d. Geheel rechts plaatsen we uitgang s. De waarden van de interne signalen zijn makkelijk te bepalen, de functies van de poorten zijn eenvoudig van aard. Uiteindelijk krijgen we de waarden zoals te zien is in tabel 23.

x	y	z	a	b	c	d	s
0	0	0	1	0	0	0	0
0	0	1	1	0	1	1	1
0	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0
1	0	0	1	0	0	0	0
1	0	1	1	0	1	1	1
1	1	0	0	1	0	1	0
1	1	1	0	1	0	1	1

Tabel 23: Waarheidtabel voor functie s.

Met behulp van de tabel is het makkelijk om een uitgangswaarde te bepalen bij een gegeven ingangscombinatie. Stel dat $xyz = 101$ is, dan vinden we in de zesde rij dat de uitgang logisch 1 is. Vanuit de waarheidstabel is direct de mintermvorm van de functie op te stellen:

$$s = \bar{x} \cdot \bar{y} \cdot z + x \cdot \bar{y} \cdot z + x \cdot y \cdot z = \sum m(1, 5, 7)$$

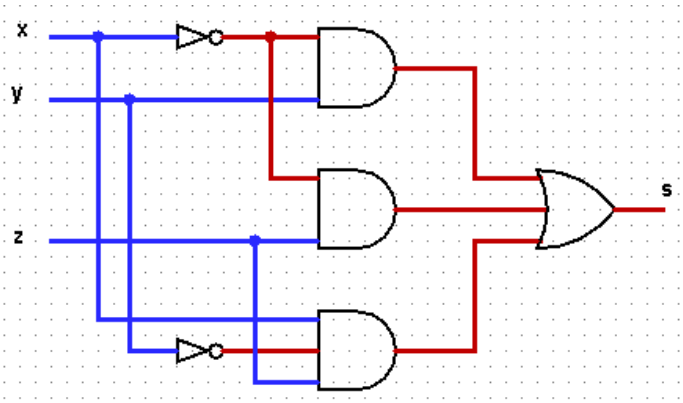
Het gebruik van waarheidstabellen bij analyse beperkt zich tot maximaal vijf variabelen. Bij meer variabelen worden de tabellen vanwege de grootte niet meer hanteerbaar en moet voor de schakelalgebra gekozen worden.

8.2 Synthese van combinatorische schakelingen

Met synthese bedoelen we het samenstellen van een schakeling die is opgebouwd uit kleinere delen. Al eerder hebben we gezien dat elke schakelfunctie te beschrijven is met de operatoren AND, OR en NOT. Bij het realiseren van een schakeling kunnen we dus gebruik maken van de drie bijbehorende poorten. Als voorbeeld nemen we de functie in de som-van-producten-vorm (SOP-vorm):

$$s = \bar{x} \cdot y + \bar{x} \cdot z + x \cdot \bar{y} \cdot z$$

Het realiseren van een schakeling gaat als volgt. Als eerste moeten de inverse van x en de inverse van y gerealiseerd worden. Hiervoor worden NOT-poorten gebruikt. Daarna worden de producttermen gerealiseerd door middel van AND-poorten. De sommatie van de producttermen wordt gerealiseerd door middel van een OR-poort. Het resultaat is te zien in figuur ?. Merk op dat voor de inverse van x slechts één NOT-poort nodig is. Deze schakeling heeft zes poorten en twaalf ingangen.

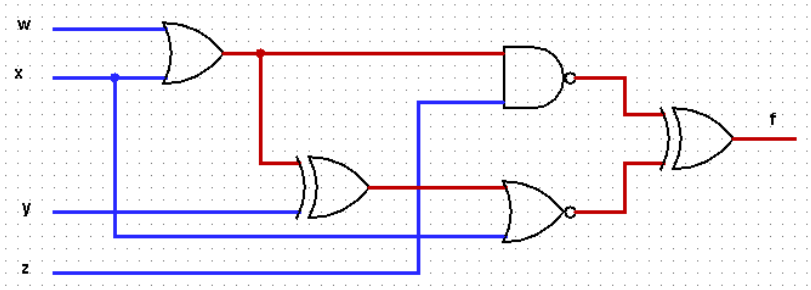


Figuur 46; Synthese van een schakeling met poorten.

In de praktijk komen nog wel andere poorten voor zoals de NAND-, NOR- en EXOR-poorten. Als voorbeeld nemen we de functie:

$$f = \overline{(w + x) \cdot z} \oplus \overline{\big((w + x) \oplus y\big) + x}$$

De schakeling is te zien in figuur 47.



Figuur 47: Synthese van een schakeling met poorten.

Deze schakeling kost vijf poorten en tien ingangen. Nu rijst de vraag: kan het niet eenvoudiger?

8.3 Minimalisatie

Uit het oogpunt van kosten, oppervlakte en energieverbruik (wat ook kosten met zich meebrengt) is het wenselijk om een schakelfunctie te minimaliseren. Er wordt gezocht naar de eenvoudigste vorm van een functie. Het is niet eenvoudig om aan te geven wat de eenvoudigste vorm is. We kunnen zoeken naar een vorm die het minste aantal operaties (poorten) en variabelen (ingangen) bevat. Hoe minder poorten, hoe minder oppervlak op een ic. Een ingang betekent bedrading (wires) op een ic en waar bedrading is, kan geen logische poort worden geplaatst. Het is dus van belang om het aantal poorten en ingangen te verkleinen.
Maar het kan ook zijn dat een vorm wordt gezocht die het eenvoudigst afleidbaar is voor poorten of transistoren op ic's. Denk hierbij aan NAND- en NOR-poorten. Deze zijn makkelijker en goedkoper (lees: minder transistoren) te realiseren dan AND- en OR-poorten. Al met al zijn er meerdere manieren om te minimaliseren. Aangezien we steeds spreken over logische poorten hanteren we de definitie dat de functie in de meest eenvoudige som-van-producten-vorm (SOP-vorm) is geschreven met:

- een zo klein mogelijk aantal producttermen;
- per productterm een zo klein mogelijk aantal variabelen.

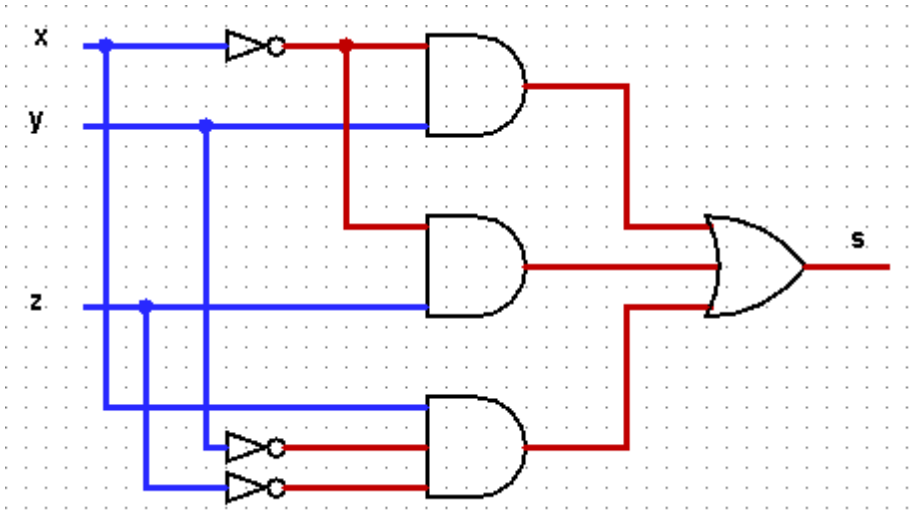
Om bestaan verschillende technieken voor de correcte minimalisatie van functies: schakelalgebra, een grafische methode genaamd Karnaugh-diagrammen en een tabellarische methode genaamd algoritme van Quine-McCluskey. De laatste twee methoden leveren direct vereenvoudigde functies in SOP-vorm. Ook nu geldt dat betreffende technieken in een vervolgcursus worden behandeld.

8.4 Realisatie met AND, OR en NOT

Realisatie vanuit de SOP-vorm is eenvoudig. De schakelfuncties zijn direct afleidbaar naar AND-, OR- en NOT-poorten. Het levert altijd een schakeling op met drie lagen (of niveaus): één laag met NOT-poorten, één laag met AND-poorten één laag met OR-poorten. Als voorbeeld nemen we de volgende functie in de SOP-vorm:

$s = \bar{x} \cdot y + \bar{x} \cdot z + x \cdot \bar{y} \cdot \bar{z}$

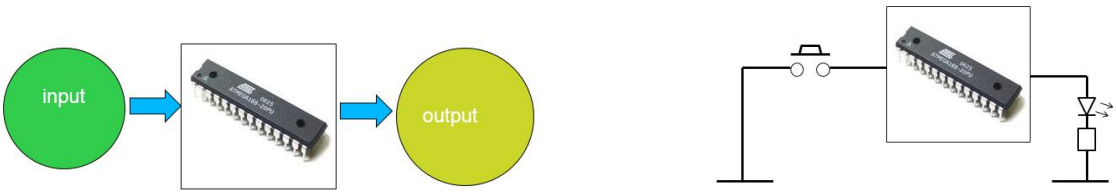
De realisatie is te zien in figuur 48.



Figuur 48: Schema voor de functie in NOT-AND-OR-vorm.

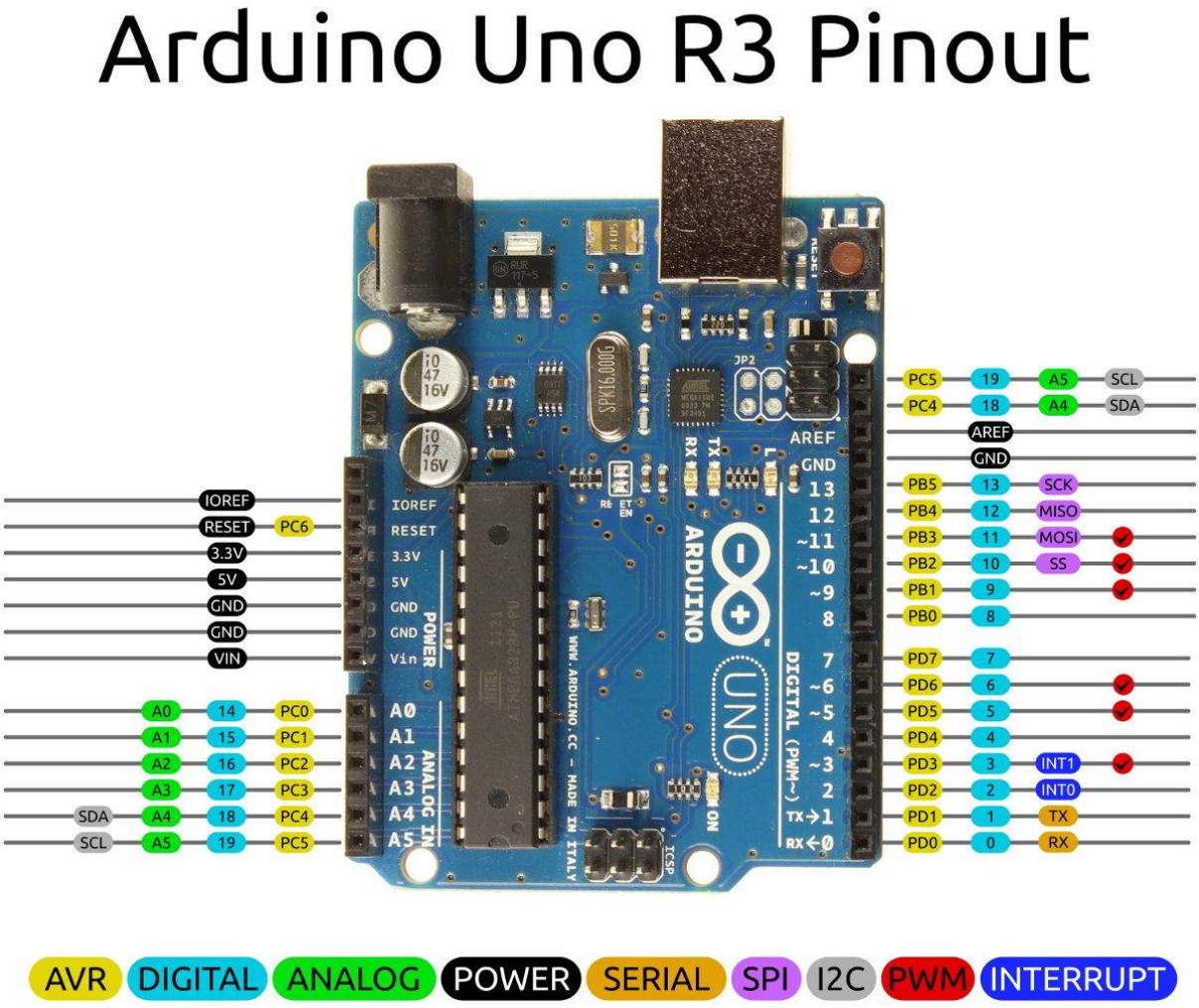
H9. Microcontrollers programmeren

Een microcontroller is een kleine computer die programma's kan uitvoeren. In het practicum heb je hiermee al ervaring opgedaan door Arduino-code te schrijven voor het aansturen van de MFS. Er bestaat nog een andere werkwijze om een microcontroller te programmeren, dat is de AVR-aanpak.



Figuur 49: Algemeen schema invoer, verwerking en uitvoer.

De AVR-programma's lezen en schrijven waarden van en naar pinnen in de vorm van digitaal logische spanningen (voltages). Het is dus van belang om te weten welke pinaansluitingen voor AVR-programmering gebruikt kunnen worden.



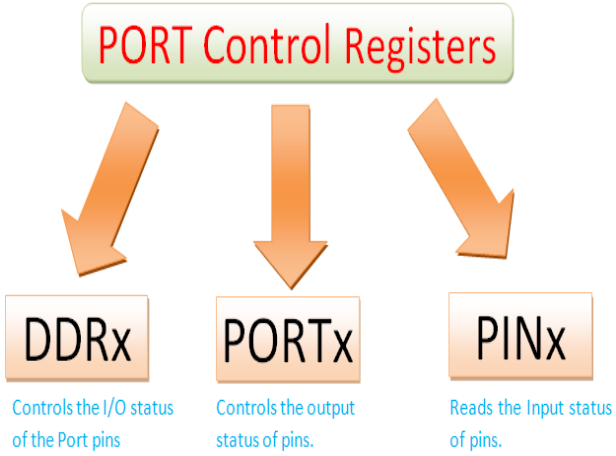
Figuur 50: Pinaansluitingen Arduino Uno. [39]

De programma's lezen en schrijven waarden van en naar pinnen of van en naar bits in registers in de vorm van digitaal logische spanningen. Hoe communiceert een Arduino met de buitenwereld? Dat kan met de seriële aansluitingen, maar de CPU kan ook gebruik maken van (via de databus) PORT B, PORT C en PORT D.
Elke register in het geheugen van de Arduino is een byte, en elke byte bestaat uit acht bits. Als je een microcontroller pin wilt configureren, ziet dit er in code nagenoeg hetzelfde uit als in de code om de waarde van een variabele op te slaan.
Bijvoorbeeld: `DDRB = 0b00000001;`
Elke groep van pinnen (hier B, C en D) heeft drie hardware-register geheugen locaties die ermee verbonden zijn.
Afspraak: laat `x` staan voor een letter van een zekere groep pinnen. Dan betekent `DDRx` dus `DDRB`, `DDRC`, of `DDRD` afhankelijk van welke groep pinnen je wilt bewerken of benaderen.

9.1 Belangrijke hardware registers

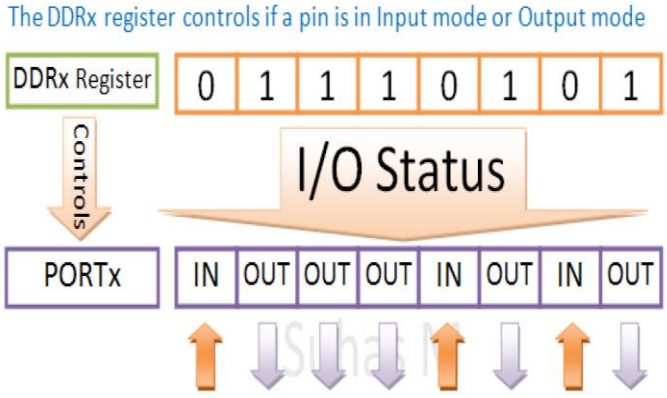
Elke groep van pinnen heeft drie hardware-registers die ermee verbonden zijn:

1. `DDRx`. Data Direction Register. Zo een register bepaalt of een pin uit deze groep is geconfigureerd voor de input- of output data richting. Een nul (0) staat voor input en een één (1) voor output. Na een reset of stroomuitval is de default dat alle pinnen op nul staan, dus op input!
2. `PORTx`. Wanneer de `DDRx` bits worden ingesteld op output voor een bepaalde pin, zal het overeenkomstige PORT register bit ingesteld kunnen worden op logisch hoog of laag (bijvoorbeeld LED aan of uit). Staan de `DDRx` bits op input, betekent het setten van het overeenkomstige PORT register bit dat de pin wordt aangesloten op een interne pull-up weerstand.
3. `PINx`. Je leest de digitale waarde af voor een pin die als input is geconfigureerd. Je kan er NIET naar schrijven.



Figuur 51: Belangrijke registers.

Elke bit in de `DDRx` bepaalt de richting van de overeenkomstige bit in `PORTx`.



Figuur 52: Input- of outputmode op bitniveau.

Voorbeeld:
Stel `DDRB = 0b00000001;`
De prefix `0b` vertelt de compiler dat de code in binaire representatie staat. Deze representatie kan natuurlijk ook in een ander formaat. Bijvoorbeeld hexadecimaal: 00000001 = 0000 0001 = 0x01
`DDRB = 0x01;` is dus dezelfde toekenning als hierboven.
Deze toekenning heeft tot gevolg dat bit nummer 0 de waarde 1 krijgt! Alle andere bits (dus 1 t/m 7) krijgen de waarde 0.

bit#	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	1

Figuur 53: Inhoud na DDRB = 0x01;

Betekenis: PB0 is geconfigureerd voor output. PB1 t/m PB7 zijn geconfigureerd voor input.
Nadeel,

```
DDRB = 0xFF;
PORTB = 0b00001000;
```

Deze toekenning heeft tot gevolg dat LED3 gaat branden. LEG UIT! Alle andere LED's branden niet.
Wat nu als je een specifiek LED aan of uit wilt zetten zonder de andere LED's te beïnvloeden?

9.2 Bit-manipulaties

Logische bewerkingen per bit stellen ons in staat elk bit te wijzigen van het register zonder expliciet na te denken over de andere bits.
Dit onderdeel (bit manipulatie) is nogal theoretisch van aard, maar onmisbaar in embedded software.
Er zijn vier typen manipulaties die we met behulp van de 'bitwise operatoren' ,zie tabel 24, kunnen uitvoeren:

1. Bit shifting.
2. Setting bits met behulp van OR.
3. Flipping bits met behulp van XOR.
4. Clearing bits met behulp van AND en NOT.

BITWISE operatoren

symbool	betekenis
<<	shift left
>>	shift right
&	bitwise AND
|	bitwise OR
~	bitwise complement
^	bitwise XOR

Tabel 24: Bitwise operatoren in de taal C.

9.2.1 Bit shifting

```
i << j;
```

Het resultaat van deze operatie is dat de bits in i over j plaatsen zijn verschoven naar links. Aan de linkerkant vallen er bits 'over het randje'. Aan de rechterkant wordt aangevuld met nullen.
`0b00001110 << 2 = 0b00111000`
Let op! 14 * 2 * 2 = 56

```
i >> j;
```

Het resultaat van deze operatie is dat de bits in i over j plaatsen zijn verschoven naar rechts. Aan de rechterkant vallen er bits over het randje. Aan de linkerkant wordt aangevuld met nullen of enen.
`0b00001110 >> 2 = 0b00000011`
Let op! 14 / 2 / 2 = 3 (hopelijk herken je dit nog uit o.a. Java Programming als 'integer division').

Bit shifters zijn binaire operatoren. De operator << schuift naar links en de operator >> naar rechts.
Aan de linkerkant van de shift-operator staat het patroon dat verschoven moet worden en aan de rechterkant staat het aantal plaatsen wat geschoven moet worden, de zogenaamde shift-count.
Bij het schuiven naar links worden er altijd nullen aan de rechterkant ingeschoven.
Bij schuiven naar rechts is het wat ingewikkelder:

- Als het patroon unsigned is worden er ook nullen ingeschoven.
- Als het patroon signed is wordt bij het inschuiven de tekenbit (bit7) gekopieerd.

Voorbeeld:

```
uint8_t i = 0;
DDRB = 0xFF;
 while (i < 8)
 {
    PORTB = (1 << i);
    _delay_ms(200);
    i++;
 }
```

Figuur 54: Wat doet deze code?

9.2.2 Setting bits met behulp van OR.

Bekijk de bitwise OR bewerking.
Welke invloed heeft een OR bewerking met een 0 bit en een ander bit? Geen invloed! Het andere bit blijft ongewijzigd.
Welke invloed heeft een OR bewerking met een 1 bit en een ander bit? Het andere bit zal altijd 1 worden!

OR

	0	0	1	1
|	0	1	0	1
=	0	1	1	1

Tabel 25: Bitwise OR-functie.

Het setten van één of meerdere bits gebeurd aan de hand van een zogenoemd bitmasker. Een bitmasker is gewoon een normale byte, samengesteld uit enen en nullen en komt dus neer op een numerieke waarde.
We gebruiken een bitmasker, samen met een bitsgewijze logische operator om een aantal bits in een doel byte te veranderen.
Voorbeeld 1:

```
DDRB = 0xFF;
PORTB = 0b11000011; //LED2 is uit
PORTB = PORTB | (1 << 2);
```

	1	1	0	0	0	0	1	1
|	0	0	0	0	0	1	0	0
=	1	1	0	0	0	1	1	1

Tabel 26: Bitwise OR voorbeeld 1.

In `(1 << 2)` is het bitmasker voor LED2 (0b00000100). Dus 0x04 is dat ook. Het resultaat is nu dat PORTB gelijk is aan 0b11000111. LED2 is nu dus aan!
Vergelijkbare code is:

- `PORTB = PORTB | 0x04;`
- `PORTB |= (1 << 2);`
- `PORTB |= 0x04;`

Kan je ook meerdere bits in een register tegelijkertijd setten? Stel LED2 en LED4 moeten beide aan. Dan kan het bitmasker `0b00010100` zijn.
Dit is te bereiken met:

```
0b00010100 = (0b00010000 | 0b00000100)
0b00010100 = ((1 << 4) | (1 << 2))
00010100 = 0x14
```

```
PORTB = 0b00010100;
PORTB = ((1<<4)|(1<<2));
PORTB = 0x14;
```

	1	1	0	0	0	0	1	1
|	0	0	0	1	0	1	0	0
=	1	1	0	1	0	1	1	1

Tabel 27: Meerdere bits tegelijkertijd setten.

9.2.3 Flipping bits met behulp van XOR

Bekijk de bitwise XOR bewerking.
Welke invloed heeft een XOR bewerking met een 0 bit en een ander bit? Geen invloed! Het andere bit blijft ongewijzigd.
Welke invloed heeft een XOR bewerking met een 1 bit en een ander bit? Het andere bit zal worden omgedraaid (= flipping = toggling). Dus 0 wordt 1 en 1 wordt 0.
Hiermee zijn we in staat om iets wat uit staat aan te zetten en andersom.

XOR

	0	0	1	1
^	0	1	0	1
=	0	1	1	0

Tabel 28: Bitwise XOR-functie.

```
PORTB = 0b11000011; //LED2 uit
PORTB ^= (1 << 2);
PORTB ^= 0x04;
```

LED2 is aan.
Uiteraard is het ook nu mogelijk meerdere bits tegelijkertijd te togglen.

	1	1	0	0	0	0	1	1
^	0	0	0	0	0	1	0	0
=	1	1	0	0	0	1	1	1

Tabel 29: Flippen van bit #2.

9.2.4 Clearing bits met behulp van AND en NOT

AND

	0	0	1	1
&	0	1	0	1
=	0	0	0	1

Tabel 30: Bitwise AND-functie

Bekijk de bitwise AND bewerking. Welke invloed heeft een AND bewerking met een 0 bit en een ander bit? Het andere bit wordt zeker 0.
Welke invloed heeft een AND bewerking met een 1 bit en een ander bit? Het andere bit blijft ongewijzigd. Hieruit valt dan af te leiden dat een bitmasker overal 1 is waar we niets willen veranderen en een 0 op de plek waar we een bit willen 'clearen'.

Schakel LED2 uit wordt dan een AND bewerking met als bitmasker `0b11111011`.
Uitgewerkt geeft dit:

```
((1 << 7) | (1 << 6) | (1 << 5) | (1 << 4) | (1 << 3) | (1 << 1) |
(1 << 0))
```

Nogal veel werk.
Echter het bitmasker `0b11111011` is exact het tegenovergestelde van allemaal nullen en alleen op plek 2 een één. Dit kunnen we dus oplossen met de bitwise NOT.

- `(1 << 2) = 0b00000100`
- `~(1 << 2) = 0b11111011`

LED2 is uit.

```
PORTB = 0b11000011;
PORTB &= ~(1 << 2);
```

LED2 is nog steeds uit!

	1	1	0	0	0	0	1	1
~	0	0	0	0	0	1	0	0
&	1	1	1	1	1	0	1	1
=	1	1	0	0	0	0	1	1

Tabel 31: Uitwerking van code-voorbeeld.

LED2 is aan.

```
PORTB = 0b11000111;
PORTB &= ~(1 << 2);
```

LED2 is nu uit!

	1	1	0	0	0	1	1	1
~	0	0	0	0	0	1	0	0
&	1	1	1	1	1	0	1	1
=	1	1	0	0	0	0	1	1

Tabel 32: Uitwerking van code-voorbeeld.

9.3 Opdrachten

Beschrijf van alle onderstaande opdrachten wat de code telkens doet.

1.
```
#include <avr/io.h>
int main(void) {
  DDRB = 0xFF;
  PORTB = 0x55;
  PORTB |= 0x08;
  while (1);
  return 0;
}
```

2.
```
#include <avr/io.h>
int main(void) {
  DDRB = 0xFF;
  PORTB = 0xAA;
  PORTB &= 0xF7;
  while (1);
  return 0;
}
```

3.
```
#include <avr/io.h>
int main(void) {
  DDRB = 0xFF;
  PORTB = 0xAA;
  PORTB ^= 0x08;
  while (1);
  return 0;
}
```

4.
```
#include <avr/io.h>
int main(void) {
  DDRB = 0xFF;
  PORTB = 0x5A;
  PORTB |= 0x14;
  PORTB &= ~0x22;
  PORTB ^= 0xc1;
  while (1);
  return 0;
}
```

Vervolg op volgende bladzijde!

5.
```
#include <avr/io.h>
int main(void) {
  DDRB = 0xFF;
  DDRA = 0x00;
  while (1) {
    if ((PINA & 0x08) == 0x08)
      PORTB = 0x7F;
    else
      PORTB = 0xFE;
  }
  return 0;
}
```

6.
```
#include <avr/io.h>
int main(void) {
  DDRB = 0xFF;
  DDRA = 0x00;
  while (1) {
    if ((PINA & 0x08) == 0x00)
      PORTB = 0x7F;
    else
      PORTB = 0xFE;
  }
  return 0;
}
```

7.
```
#include <avr/io.h>
int main(void){
  DDRB = 0xFF;
  DDRA = 0x00;
  while (1) {
    if ((PINA & 0x28) != 0x00)
      PORTB = 0x7F;
    else
      PORTB = 0xFE;
  }
  return 0;
}
```

BIBLIOGRAFIE

Veel materiaal is tegenwoordig (alleen) via Internet beschikbaar. Voorbeelden hiervan zijn de datasheets van ic's die alleen nog maar via de website van de fabrikant beschikbaar worden gesteld. Dat is veel sneller toegankelijk dan boeken en tijdschriften. De keerzijde is dat websites van tijd tot tijd veranderen of verdwijnen. De geciteerde weblinks werken dan niet meer. Helaas is daar niet veel aan te doen. Er is geen garantie te geven dat een weblink in de toekomst beschikbaar blijft. [40]

[1]: J.E.J. op den Brouw. *Digitale Techniek, Een inleiding in het ontwerpen van digitale systemen*. Deel 1, Inleiding.
[2]: Definitie embedded systemen; url: https://nl.wikipedia.org/wiki/Embedded_system (bezocht op 09-10-2018).
[3]: What is an embedded system; url: https://www.techopedia.com/definition/3636/embedded-system (bezocht op 09-10-2018).
[4]: Definitie embedded systemen; url: https://www.itpedia.nl/2011/07/15/embedded-software/ (bezocht op 09-10-2018).
[5]: Afbeelding *Onderdelen van een embedded systeem*; url: https://wikikids.nl/Embedded_system (bezocht op 09-10-2018).
[6]: Home Control Box; url https://www.fabertronics.nl/xanura-hcb-home-control-box (bezocht op 09-10-2018).
[7]: A.P. Thijssen, H.A. Vink en C.H. Eversdijk. *Digitale techniek, van probleemstelling tot realisatie, deel 2*. 3de ed. Delft, Zuid-Holland: Delftse Uitgevers Maatschappij, 1990, p. 11. ISBN: 90-6562-069-9.
[8]: Inleiding talstelsels; url: https://nl.wikibooks.org/wiki/Wiskunde/Talstelsels (bezocht 11-10-2018).
[9]: Toepassingen; url: https://nl.wikibooks.org/wiki/Basiskennis_informatica/Talstelsels (bezocht op 15-10-2018).
[10]: Flags – register; url: http://i.imgur.com/nlaSmQv.jpg (bezocht op 15-10-2018).
[11]: Whireshark; url: https://upload.wikimedia.org/wikipedia/commons/thumb/c/c4/Ethereal_Screenshot.png/266px-Ethereal_Screenshot.png (bezocht op 16-10-2018).
[12]: one – complement; url https://nl.wikipedia.org/wiki/One%27s_complement (bezocht op 23-10-2018).
[13]: two – complement; url https://nl.wikipedia.org/wiki/Two%27s_complement (bezocht op 25-10-2018).
[14]: excess representation; url https://en.wikipedia.org/wiki/Offset_binary (bezocht op 25-10-2018).
[15]: ISO/IEC 9899:2011. *Programming Languages—C*, 3rd ed (ISO/IEC 9899:2011). Geneva, Switzerland: ISO, 2011; 6.2.5, paragraaf 9.
[16]: fixed point binary; url http://www-inst.eecs.berkeley.edu/~cs61c/sp06/handout/fixedpt.html (bezocht op 29-10-2018).
[17]: J.E.J. op den Brouw. *Digitale Techniek, Een inleiding in het ontwerpen van digitale systemen*. Deel 1, §1.6 en verder.
[18]: voltage to logic; url https://d2vlcm61l7u1fs.cloudfront.net/media%2Fa41%2Fa419ce8d-a0ff-4f26-a646-bf77e5177e09%2FphpIZJbz5.png (bezocht op 30-10-2018).
[19]: Texas Instruments. *Datasheet SN7408*. Maart 1988; url http://www.ti.com/lit/ds/symlink/sn74s08.pdf (bezocht op 30-10-2018).
[20]: 7408 AND gate; url https://www.electroschematics.com/wp-content/uploads/2013/07/7408-IC-photo.jpg (bezocht op 30-10-2018).
[21]: Pinaansluitingen 7408 AND; url https://upload.wikimedia.org/wikipedia/commons/c/ca/7408.jpg (bezocht op 30-10-2018).

[22]: IEC. Normblad 617-12, *Graphical Symbols for Diagrams*, Part 12: Binary Logic Elements. Nederlands Normalisatie Instituut, 1984.
[23]: IEEE. IEEE *Graphic Symbols for Logic Functions* (Includes IEEE Std 91a-1991 Supplement, and IEEE Std 91-1984).
[24]: I. Kampel. *Practical Design of Digital Circuits: Basic Logic to Microprocessors*. Elsevier Science, 1983, p. 296. ISBN: 9781483135564.
[25]: Symbolen AND-poort; url https://upload.wikimedia.org/wikipedia/commons/4/4d/AND-poort_symbolen.png (bezocht op 01-11-2018).
[26]: Symbolen OR-poort; url https://upload.wikimedia.org/wikipedia/commons/1/19/OR-poort_symbolen.png (bezocht op 01-11-2018).
[27]: Symbolen NOT-poort; url https://upload.wikimedia.org/wikipedia/commons/2/26/NOT-poort_symbolen.png (bezocht op 01-11-2018).
[28]: Vormsymbool buffer; url https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTn-Spblbd1aGfs6qPkNCaJHhmrTWmN90fZvbD0EjxiijaOJt87 (bezocht op 01-11-2018).
[29]: IEC buffer symbool; url https://upload.wikimedia.org/wikipedia/commons/thumb/a/a7/Buffer_IEC.svg/2000px-Buffer_IEC.svg.png bezocht op 01-11-2018).
[30]: Symbolen EXOR-poort; url https://upload.wikimedia.org/wikipedia/commons/9/94/XOR-poort_symbolen.png (bezocht op 01-11-2018).
[31]: Vormsymbool NAND; url https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRrc8ib_TnV9A2PmRyPhbSS5f2-6P_JgOWp6KdW5Ezyaa1Tl-XS (bezocht op 02-11-2018).
[32]: IEC symbool NAND; url https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSqM5bPHAsg3FCNLq7G913N7FGi9BZP3nWVsjjKrgIqOTe-oBk9 (bezocht op 02-11-2018).
[33]: Vormsymbool NOR; url https://electronicsclub.info/images/gates1.gif (bezocht op 02-11-2018).
[34]: IEC symbool NOR; url https://upload.wikimedia.org/wikipedia/commons/thumb/3/37/IEC_NOR_label.svg/2000px-IEC_NOR_label.svg.png (bezocht op 02-11-2018).
[35]: Vormsymbool EXNOR; url http://www.equestionanswers.com/notes/digital-logic-gates/xnor-gate.gif (bezocht op 02-11-2018).
[36]: IEC symbool EXNOR; url https://upload.wikimedia.org/wikipedia/commons/thumb/5/59/IEC_XNOR.svg/320px-IEC_XNOR.svg.png (bezocht op 02-11-2018).
[37]: J.F.Wakerly. Digital Design, Principles and Practices. Prentice Hall Series in Computer Science, 1990, p. 156. ISBN: 0132128381.
[38]: J.E.J. op den Brouw. *Digitale Techniek, Een inleiding in het ontwerpen van digitale systemen*. Deel 1, Hoofdstuk 4.
[39]: Arduino Uno; url https://www.allaboutcircuits.com/uploads/articles/Arduino_UNO_R3_Pinout.jpg (bezocht op 06-11-2018).
[40]: J.E.J. op den Brouw. *Digitale Techniek, Een inleiding in het ontwerpen van digitale systemen*. Deel 1, Bibliografie.