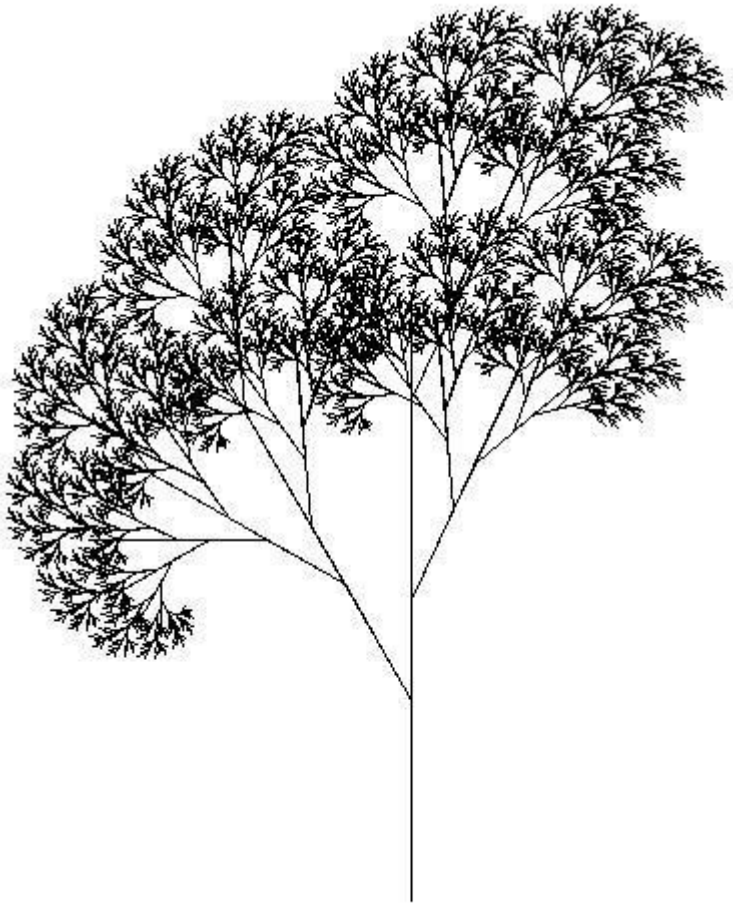


# Informatica 2

Met uitwerkingen n.a.v. document van Elvire Theelen in Luc
bijgewerkt door Peter van Diepen



remie.woudt@gmail.com

Voorblad:
Boom getekend met de programmeertaal LOGO, gebruik makend van recursie.

# Inhoud

# 2 Digitaal

# 2.1 Talstelsels

Een computer is eigenlijk niets anders is dan heel veel schakelaars. En schakelaars kennen (meestal) maar twee standen, aan of uit.

## 2.1.1 Binair

Om te kunnen berekenen wat een computer precies zou moeten doen en hoe we ons dat dan voor moeten stellen, is een systeem nodig dat overeenkomst vertoont met die schakelaars. Aangezien we normaal gesproken rekenen met getallen is het logisch om bij het rekenen in een computer gebruik te maken van een **binair** of tweetallig stelsel. Oftewel een stelsel met maar twee getallen, 0 en 1.

Wanneer we in ons gewone 10-tallig stelsel vanaf 0 gaan tellen, tellen we eerst tot en met 9 en daarna wordt het meest rechtse getal opnieuw 0. Maar er komt dan een 1 voor. Met het tweetallig stelsel werkt het net zo.

Je begint met een 0, dan een 1 maar daarna zijn we al door onze getallen heen. Dus dan zetten we er een 1 voor en beginnen we rechts weer met 0. En zo gaan we door.
Kijk maar eens naar de tabel op de volgende pagina. In de linker kolom staat het decimale getal, in de rechter kolom het tweetallig of binaire getal met dezelfde waarde.

Zoals je in de tabel ziet wordt het getal 4 dus binair voorgesteld als 100. Oei, dit zou verwarrend kunnen zijn. We kunnen zo niet zien of hier nou het getal honderd bedoeld wordt of het binaire getal dat 4 voorstelt. Om dat onderscheid duidelijk te maken geven we binaire getallen vaak met een kleine 2 tussen haakjes erachter die iets lager staat dan het getal zelf.

Dus zo: $100_{(2)}$.

Omdat het bij binaire getallen handig is ze in groepjes van 4 te plaatsen en omdat ieder getal eigenlijk de stand van een schakelaar voorstelt, worden ook nullen vooraan het getal geschreven als we daarmee een groepje van 4 krijgen. Je zou dus kunnen schrijven: $4_{(10)} = 0100_{(2)}$.
Hier staat dus eigenlijk: 4 in het tientallig stelsel wordt in het tweetallig stelsel geschreven als 0100.

| decimaal | binair |
|---|---|

| | |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 10 |
| 3 | 11 |
| 4 | 100 |
| 5 | 101 |
| 6 | 110 |
| 7 | 111 |
| 8 | 1000 |
| 9 | 1001 |
| 10 | 1010 |
| 11 | 1011 |
| 12 | 1100 |
| 13 | 1101 |
| 14 | 1110 |
| 15 | 1111 |
| 16 | 1 0000 |

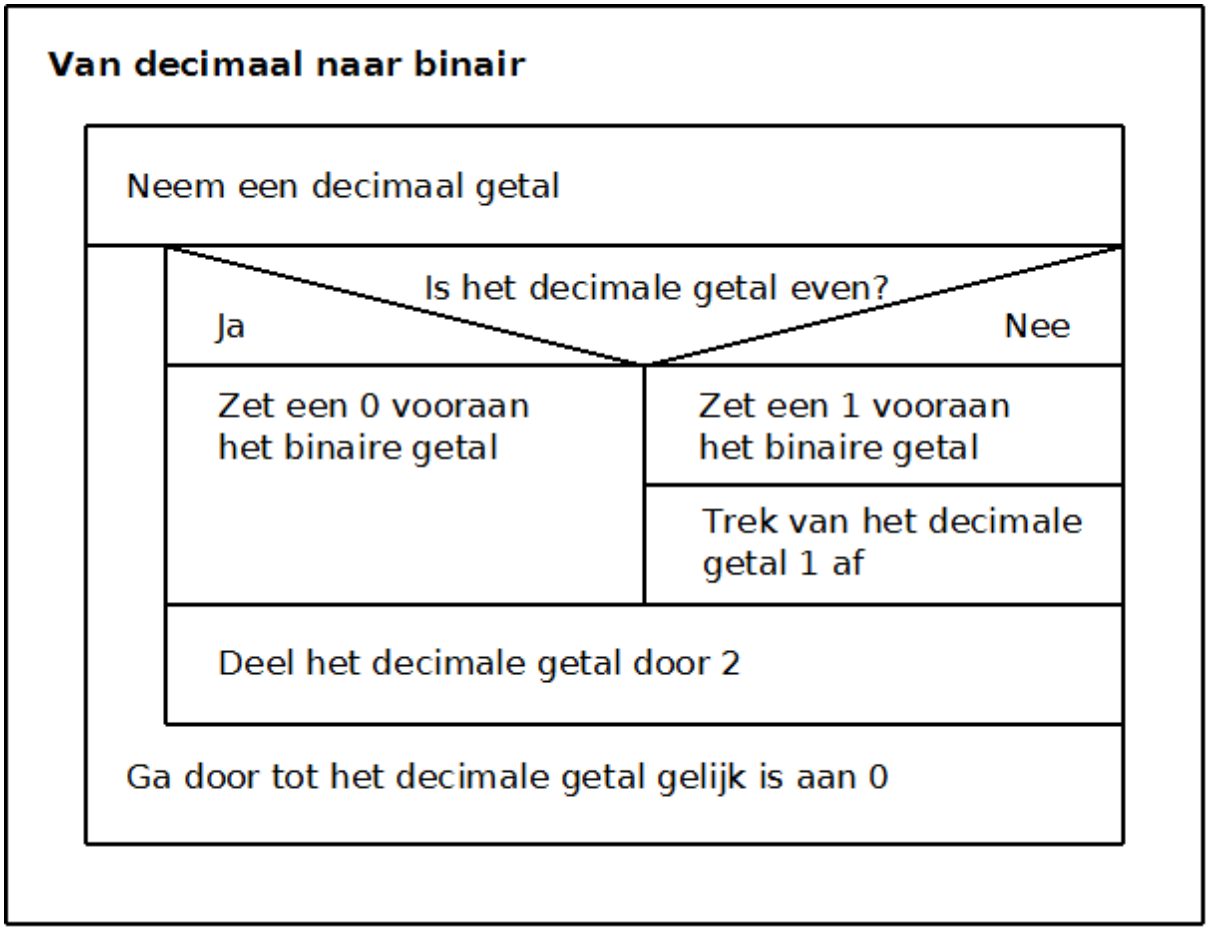
***Opdracht 1***

- Kun jij, als je kijkt naar deze tabel, nu bepalen hoe je het getal $20_{(10)}$ binair schrijft?

Probeer hiervoor de regelmaat te ontdekken in de binaire kolom en dan als het ware door te tellen.

*Binair doortellen (kijk naar de regelmaat in de kolommen) levert voor $20_{(10)}$ het binaire getal $10100_{(2)}$ op.*

Het getal $20_{(10)}$ omrekenen naar binair is nog wel te doen. Even doortellen in de binaire kolom. Maar hoe reken je $169_{(10)}$ om? Met de juiste hulpmiddelen is dat redelijk eenvoudig.




*Afbeelding 1: Van decimaal naar binair omrekenen*

In afbeelding 1 zie je een structuurdiagram dat je daarbij kunt gebruiken. Laten we eens proberen het decimale getal 169 met dit diagram om te zetten naar een binair getal. Denk erom dat je de getallen die je moet opschrijven steeds vooraan plaatst. Je schrijft dus van rechts naar links! In de rechterkolom van de tabel zie je de binaire waarden ontstaan.

| actie | resultaat |
|---|---|
| Is 169 even?<br>Nee. Schrijf een 1 op, trek 1 af van 169 (wordt nu 168) en deel door 2 (wordt nu 84).<br>Het getal is niet 0 dus ga je door. In het diagram betekent dat dat je weer naar boven gaat en opnieuw bij de vraag “is het decimale getal even?” terecht komt. | **1** |
| Is 84 even?<br>Ja, schrijf een 0 vooraan en deel 84 door 2 (wordt nu 42). Het getal is niet 0 dus ga je door. | **01** |
| Is 42 even?<br>Ja, schrijf een 0 vooraan en deel 42 door 2 (wordt nu 21). Het getal is niet 0 dus ga je door. | **001** |
| Is 21 even?<br>Nee. Schrijf een 1 vooraan, trek 1 af van 21 (wordt nu 20) en deel door 2 (wordt nu 10). Het getal is niet 0 dus ga je door. | **1001** |
| Is 10 even?<br>Ja, schrijf een 0 vooraan en deel 10 door 2 (wordt nu 5). Het getal is niet 0 dus ga je door. | **0 1001** |
| Is 5 even?<br>Nee. Schrijf een 1 vooraan, trek 1 af van 5 (wordt nu 4) en deel door 2 (wordt nu 2). Het getal is niet 0 dus ga je door. | **10 1001** |
| Is 2 even?<br>Ja, schrijf een 0 op en deel 2 door 2 (wordt nu 1). Het getal is niet 0 dus ga je door. | **010 1001** |
| Is 1 even?<br>Nee. Schrijf een 1 op, trek van 1 af (wordt nu 0) en deel door 2 (blijft 0). Het getal is 0 dus ben je klaar. | **1010 1001** |

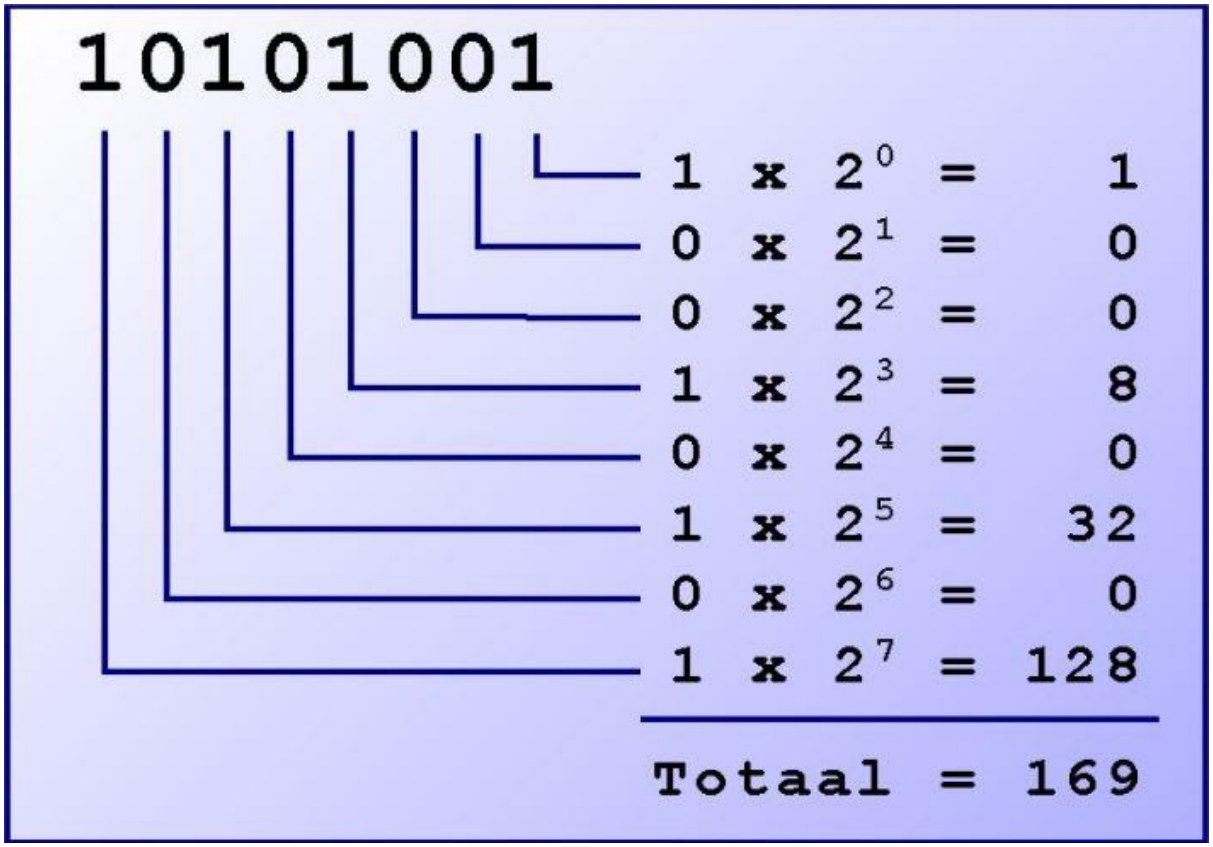
En de uitkomst is, zo heb je in de rechterkolom kunnen volgen: **10101001**$_{(2)}$.

***Opdracht 2***

- Bereken de binaire waarde van 124$_{(10)}$.

*Dat is 111 1100(2)*

Het omrekenen van binair naar decimaal is gelukkig iets minder omslachtig. Laten we het binaire getal **10101001**$_{(2)}$ van net eens bekijken. We weten dat er 169$_{(10)}$ uit moet komen maar hoe kun je dat terugrekenen?




*Afbeelding 2: Van binair naar decimaal omrekenen*

In afbeelding 2 zie je hoe dat in zijn werk gaat. Even herhalen wat je bij wiskunde al gehad hebt. Een getal tot de macht 0 is altijd 1.

We gaan nu aan de slag van rechts naar links. Het meest rechtse getal hoort bij $2^0$. Aangezien dat meest rechtse getal een 1 is betekent dat, dat we 1 maal $2^0$ hebben. Uitkomst 1.

Het volgende getal van rechts af gerekend is een 0. En dat getal hoort bij $2^1$ en dus hebben we 0 maal $2^1$. Uitkomst 0.

En zo kunnen we doorgaan. Als we dan alle uitkomsten bij elkaar tellen weten we het decimale getal, hier dus 169 (maar dat wisten we al).

***Opdracht 3***

- Bereken de decimale waarde van $10011_{(2)}$.

*Dat is 19(10)*

Overigens is het omrekenen van talstelsels een typisch voorbeeld van iets dat je heel goed door een computer kunt laten doen.

Eén 0 of 1 van een binair getal noemen we een **bit** (binary digit). Een blokje van 4 bits (zoals we hieronder tegen zullen komen) noemen we een **nibble**. 8 bits (dus 2 nibbles) vormen een **byte**.

### 2.1.2 Hexadecimaal

Naast het binaire stelsel is er nog een talstelsel dat in de computerwereld veel gebruikt wordt, namelijk het **hexadecimale stelsel** oftewel het **16-tallig stelsel**.

Wel lastig, 16-tallig dus dat telt van 0 t/m 15. Maar welke symbolen gebruiken we dan boven de 9? Want dan zijn onze cijfers op. Kijk maar in de tabel op de volgende bladzijde, dan is het wel duidelijk. Voor het hexadecimale stelsel wordt dus voor de hoogste getallen de letters A t/m F gebruikt.

Waarom is nu het hexadecimaal stelsel ook van belang voor de werking van computers? Zoals eerder gezegd werken computers binair en dat kunnen we ons voorstellen met nullen en enen. Maar bij grote getallen wordt dat al gauw erg onoverzichtelijk. Kijk maar eens naar dit getal:

**100001111100111011101011**

De kans op een foutje is groot hè? Het wordt al beter als we het in blokjes van 4 indelen:

**1000 0111 1100 1110 1110 1011**

Zet nu eens, met behulp van de tabel hieronder, onder die blokjes van 4 de hexadecimale code:

**1000 0111 1100 1110 1110 1011**
**8 7 C E E B**

En die hexadecimale getallen die er nu onder staan, zijn achter elkaar geschreven precies de hexadecimale waarde van dat binaire getal. Dus eigenlijk is hexadecimaal een soort van verkorte schrijfwijze voor een binair getal.
En dat is toch wel lekker handig want het getal dat hier staat is de kleur skyblue die je misschien wel als achtergrond zou willen gebruiken in jouw html-pagina. Dan is #87CEEB toch wel even handiger invoeren dan 100001111100111011101011.

| decimaal | binair | hexadecimaal |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 10 | 2 |
| 3 | 11 | 3 |
| 4 | 100 | 4 |
| 5 | 101 | 5 |
| 6 | 110 | 6 |
| 7 | 111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | A |
| 11 | 1011 | B |
| 12 | 1100 | C |
| 13 | 1101 | D |
| 14 | 1110 | E |
| 15 | 1111 | F |
| 16 | 1 0000 | 10 |

Meestal heeft een hexadecimaal getal een hekje (#) voor het getal staan, zodat je weet dat het om een hexadecimaal getal gaat. Maar ook deze notatie $87CEEB_{(16)}$ wordt wel gebruikt.

Zoals net al genoemd, wordt de hexadecimale waarde onder andere gebruikt om kleurwaarden aan te geven in een html-pagina. Wanneer je zo'n hexadecimaal getal splitst in 3 blokjes, dus zo: 87 CE EB en dan voor ieder van die waarden afzonderlijk

de decimale waarde berekent, dan heb je de **RGB-waarden** van jouw kleur. RGB staat voor rood-groen-blauw. De waarden van die drie kleuren kunnen variëren van 0 tot 255 en op die manier heb je de mogelijkheid 256 x 256 x 256 = 16.777.216 kleuren vast te leggen.

Zo geef je wit (#FFFFFF) aan als rgb(255,255,255) en zwart (#000000) als rgb(0,0,0).

Het omrekenen tussen hexadecimaal en binair is dus niet zo moeilijk. Lastiger is het om hexadecimaal om te rekenen naar decimaal en omgekeerd. Maar zolang we het in de blokjes van 2 kunnen doen, om bijvoorbeeld de RGB-waarde uit te rekenen, valt het wel mee.

Zo kunnen we het getal #87CEEB als volgt naar RGB-waarden omrekenen :

$87_{(16)} = 7 \times 16^0 + 8 \times 16^1 = 7 \times 1 + 8 \times 16 = 135_{(10)}$

$CE_{(16)} = 14 \times 16^0 + 12 \times 16^1 = 14 \times 1 + 12 \times 16 = 206_{(10)}$

$EB_{(16)} = 11 \times 16^0 + 14 \times 16^1 = 11 \times 1 + 14 \times 16 = 235_{(10)}$

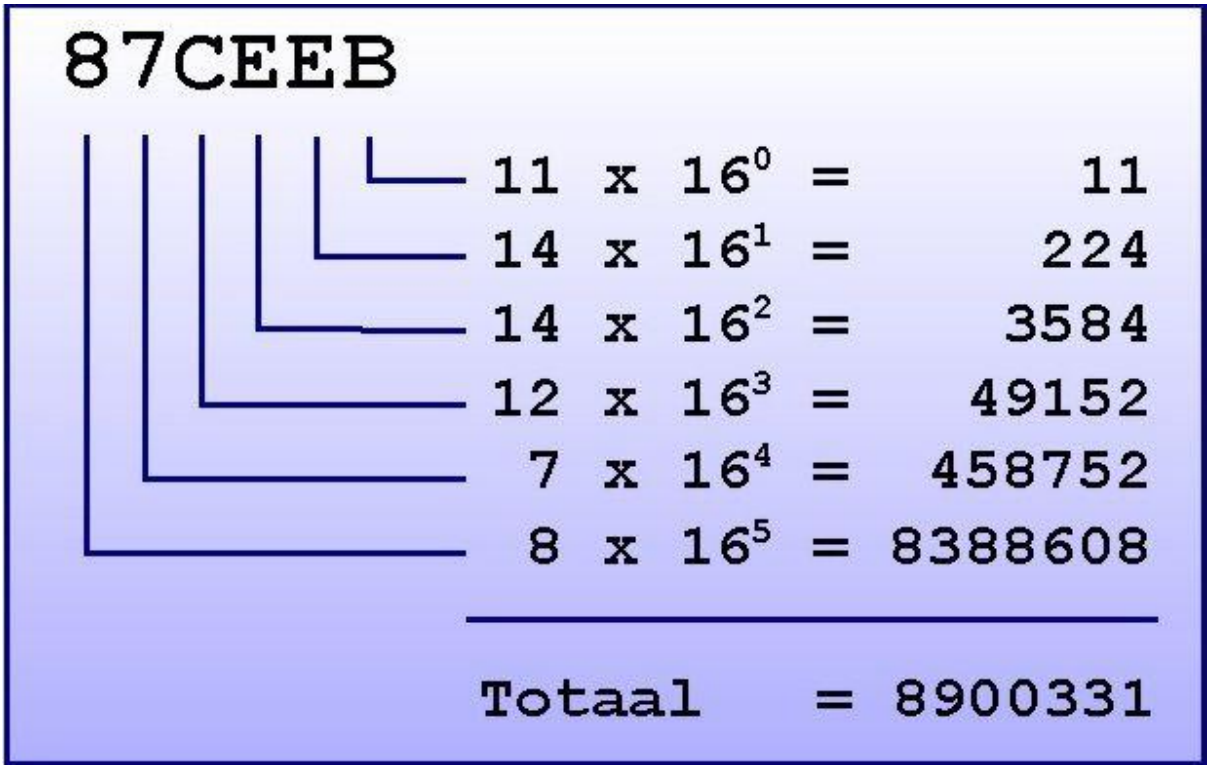
Dus #87CEEB is gelijk aan rgb(135,206,235)

***Opdracht 4***

- Bereken de RGB-waarde van #438F29

*Dat is RGB(67,143,41).*

Als je een hexadecimaal getal helemaal wilt omrekenen naar een decimaal getal dan kun je dat doen zoals in afbeelding staat. Dit gaat dus net zo als het omrekenen van binair naar decimaal. Alleen neem je hier uiteraard machten van 16.




*Afbeelding 3: Van hexadecimaal naar decimaal omrekenen*

Voor het omrekenen van decimaal naar hexadecimaal kun je het binaire stelsel als tussenstap gebruiken. Laten we weer het decimale getal 169 gebruiken. De binaire waarde daarvan is **1010 1001**.

Deel dit nu in in groepjes van 4 en zoek bij ieder groepje de hexadecimale waarde op:

```
1010 1001
  A    9
```

Het decimale getal 169 is dus gelijk aan #A9.

### 2.1.3 Rekenen met binaire getallen

Eigenlijk gaat het rekenen met binaire getallen net zo als met decimale getallen. Bedenk wel dat je nooit hoger dan een 1 kunt krijgen. Zo geldt dus de volgende optelling:

```
  1
  1
—— +
 10
```

Nog een voorbeeld:

```
  10101001
  10001111
———————— +
 100111000
```

Controleer deze berekening.

***Opdracht 5***

Bereken de volgende opgaven:

```
      1001
  10001001
———————— +

  11111111
  10111101
———————— +
```

```
     1001
  10001001
———————— +
  10010010

  11111111
  10111101
———————— +
 110111100
```

# 2.2 Poorten en waarheidstabellen

Een computer bestaat dus uit een groot aantal schakelaars. Geen schakelaars zoals we die in huis hebben, waarmee we het licht aan doen bijvoorbeeld. Maar toch schakelaars die wel of niet een stroompje doorlaten. De schakelaars in een computer zijn gemaakt van halfgeleider-materiaal en zitten met meerdere in een **chip**.

Je kunt schakelaars combineren. Zo ken je van thuis ongetwijfeld de lamp die je met twee schakelaars kunt bedienen. Vaak op de overloop. Je kunt zo'n lamp dan beneden aanzetten en boven weer uitschakelen (of omgekeerd). We noemen dat een wisselschakeling of hotel-schakeling.

In computers worden schakelaars ook gecombineerd zodat ze verschillende functies kunnen krijgen. Een combinatie van schakelaars noemen we een poort. We onderscheiden daarbij 4 poorten:

- De EN-poort of AND-poort
- De OF-poort of OR-poort
- De NIET-poort of NOT-poort
- De exclusiefOF-poort of XOR-poort

Daarnaast zijn er nog afgeleide poorten, de nietEN of NAND poort en de nietOF of NOR poort. Deze laatste twee zijn eigenlijk een combinatie van een AND en een NOT poort en een OR en een NOT poort. In het vervolg zullen we hier de Engelse termen gebruiken.

Een poort heeft altijd één of twee ingangen en één uitgang. Het is verder niet van belang te weten wat er precies in zo'n poort zit. Het is de werking die telt. Wanneer we een spanning zetten op één of beide ingangen gebeurt er, afhankelijk van het type poort, iets aan de uitgang.

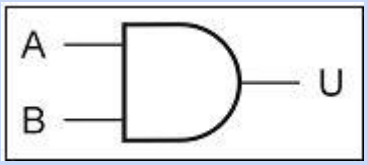
In onze voorbeelden zullen we de ingangen steeds aangeven met A en B en de uitgang met U. Verder zullen we niet meer over spanningen spreken maar over een 1 of een 0.

### 2.2.1 De AND-poort (EN-poort)

In afbeelding 4 zie je het symbool van de AND-poort. Een AND-poort heeft de volgende eigenschappen:

**Uitgang U heeft de waarde 1 wanneer zowel ingang A als ingang B beide 1 zijn. In alle andere gevallen heeft uitgang U de waarde 0.**

Om het allemaal nog wat duidelijker te maken, stellen we ons alle mogelijke combinaties van de waarden die A en B kunnen hebben en welke waarde U dan heeft voor in de vorm van een tabel. Zo'n tabel noemen we een waarheidstabel.




| A | B | U |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

*Afbeelding 4: De AND-poort en de waarheidstabel ervan*

Zo'n waarheidstabel moet je per regel bekijken.

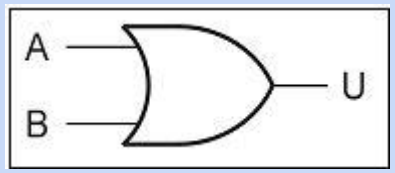
Op de eerste regel zien we een 0 bij A en ook een 0 bij B. En als A en B beide 0 zijn is U ook 0.
Maar ook als één van beiden 1 is heeft U nog steeds de waarde 0.
Alleen wanneer zowel A als B beide 1 zijn krijgt U ook de waarde 1.

### 2.2.2 De OR-poort (OF-poort)

In afbeelding 5 zie je het symbool van de OR-poort. Een OR-poort heeft de volgende eigenschappen:

**Uitgang U heeft de waarde 1 wanneer ingang A, ingang B of beide 1 zijn.**
**Uitgang U heeft alleen dan de waarde 0 als beide ingangen 0 zijn.**

Ook van de OR-poort kunnen we de waarheidstabel opstellen. Zie afbeelding 5.




| A | B | U |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

*Afbeelding 5: De OR-poort en de waarheidstabel ervan*

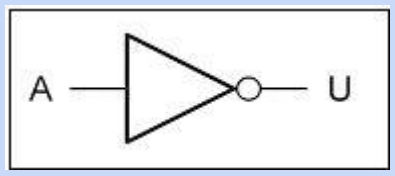
De betekenis zal inmiddels duidelijk zijn. De meest rechtse kolom geeft de waarden van U. Deze is alleen 0 wanneer A en B beide 0 zijn.

### 2.2.3 De NOT-poort (NIET-poort)

De NOT-poort is de eenvoudigste omdat hij maar één ingang heeft. Hij heeft de volgende eigenschappen:

**De NOT-poort keert het signaal om. Dus een 1 aan de ingang wordt een 0 aan de uitgang en omgekeerd.**

De NOT-poort wordt ook wel inverter genoemd.




| A | U |
|---|---|
| 0 | 1 |
| 1 | 0 |

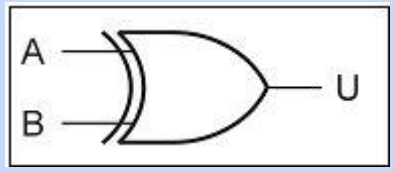
*Afbeelding 6: De NOT-poort en de waarheidstabel ervan*

### 2.2.4 De XOR-poort (exclusiefOF-poort)

De XOR-poort lijkt in zijn gedrag op de OR-poort met één verschil:

**Uitgang U heeft de waarde 1 wanneer ingang A of ingang B de waarde 1 heeft. Maar wanneer ze beiden 0 of beiden 1 zijn heeft uitgang U de waarde 0.**

Kun je de waarheidstabel al raden? Je ziet hem in afbeelding 7. Deze is vrijwel gelijk aan die van de OR-poort met alleen de laatste waarde in de U kolom anders.



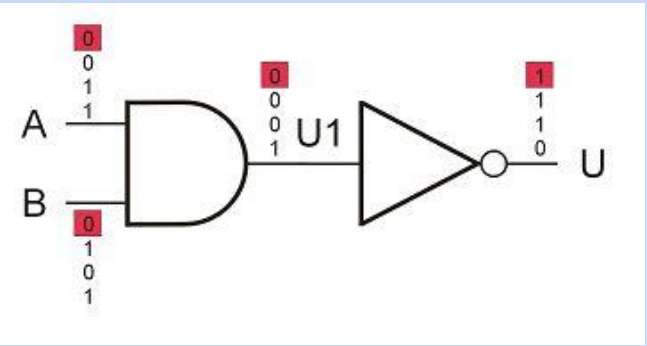

| A | B | U |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

*Afbeelding 7: De XOR-poort en de waarheidstabel ervan*

## 2.2.5 Poorten koppelen

De poorten zijn de bouwstenen van een computer. Het wordt dus tijd om er iets mee te gaan doen. Eerst maar eens kijken wat er gebeurt als we poorten met elkaar verbinden. Kijk maar eens naar afbeelding 8.

Hier is een AND-poort met aan de uitgang een NOT-poort getekend. Bij de aansluitingen van de poorten staat aangegeven welke waarden die aansluitingen kunnen aannemen. Dit zie je ook terug in de waarheidstabel.




| A | B | U1 | U |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

*Afbeelding 8: Een AND-poort met een NOT-poort erachter*

Wanneer, zoals in rood aangegeven, op ingang A een 0 staat en op ingang B ook, komt er, volgens de waarheidstabel op de uitgang U1 ook een 0. Maar dat is ook de ingang van de NOT-poort. Een 0 op de ingang van een NOT-poort geeft een 1 op de uitgang U. In de tabel zie je wat uiteindelijk de waarde van U zal zijn bij de verschillende waarden van A en B. Controleer de andere waarden!

Door het koppelen van poorten ontstaat er als het ware een nieuwe poort met ingangen A en B en uitgang U. Deze samengestelde poort wordt ook wel NAND-poort genoemd, een samenvoeging van NOT en AND. Zoals je aan de waarheidstabel ziet reageert hij precies tegengesteld aan de AND-poort.

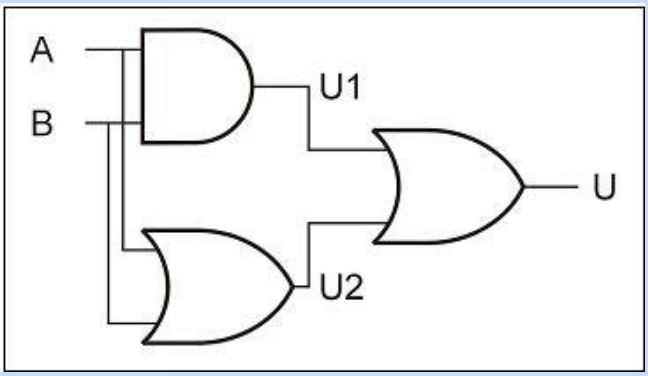
***Opdracht 6***

- Stel de waarheidstabel samen van een NOR-poort (een OR-poort met daarachter een NOT-poort).

| *A* | *B* | *U* |
|---|---|---|
| *0* | *0* | *1* |
| *0* | *1* | *0* |
| *1* | *0* | *0* |
| *1* | *1* | *0* |

Bekijk afbeelding 9:

De ingangen A en B zijn zowel aan de AND-poort als aan de OR-poort gekoppeld. De uitgangen van de AND-poort en van de OR-poort zijn de ingangen van een tweede OR-poort.
Deze schakeling lijkt een stuk ingewikkelder maar toch kunnen we deze ook met behulp van een waarheidstabel verduidelijken.




| A | B | U1 | U2 | U |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

*Afbeelding 9: Drie poorten en hun waarheidstabel*

U1 geeft de waarden aan de uitgang van de AND-poort, dus alleen een 1 wanneer zowel A als B de waarde 1 hebben. U2 geeft de waarden aan de uitgang van de onderste OR-poort dus een 0 wanneer beide 0 zijn en verder enen.

Tenslotte de U kolom. Dit is weer de waarheidstabel van een OR-poort maar hiervan zijn U1 en U2 de ingangen. Op de eerste rij zijn U1 en U2 beide 0 dus de uitgang is dan 0.

Op de tweede en derde rij heeft steeds U2 de waarde 1 en dat is voldoende om ook de uitgang 1 te laten zijn.

Op de vierde rij tenslotte hebben zowel U1 als U2 de waarde 1 dus de uitgang wordt ook een 1.

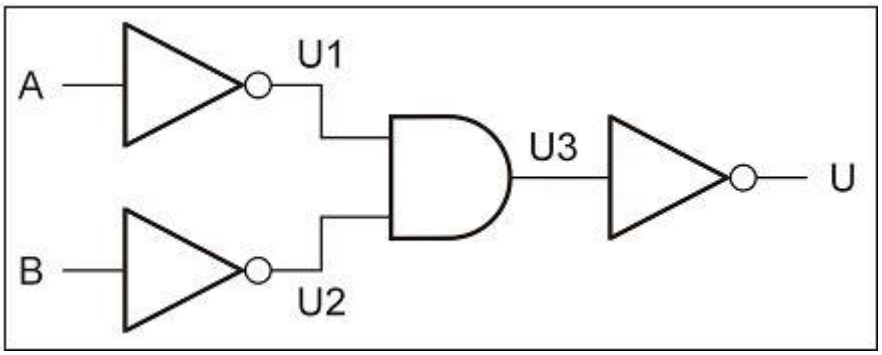
***Opdracht 7***

- Bovenstaande schakeling lijkt een beetje overdreven want hij doet precies hetzelfde als een schakeling die we al kennen. Welke?

(Tip: vergelijk de waarheidstabel)

## *De OR-poort (zie afbeelding 8)*

Het gebeurt vaker dat een bekende schakeling wordt opgebouwd uit andere poorten. Stel je wilt een OR-poort maken maar je hebt alleen maar de beschikking over AND-poorten en NOT-poorten. Probeer het eens zo:




*Afbeelding 10: Is dit een OR-poort?*

***Opdracht 8***

- Laat met een waarheidstabel zien dat de schakeling van afbeelding 10 hetzelfde doet als een OR-poort.

| *A* | *B* | *U1* | *U2* | *U3* | *U* |
|---|---|---|---|---|---|
| *0* | *0* | *1* | *1* | *1* | *0* |
| *0* | *1* | *1* | *0* | *0* | *1* |
| *1* | *0* | *0* | *1* | *0* | *1* |
| *1* | *1* | *0* | *0* | *0* | *1* |

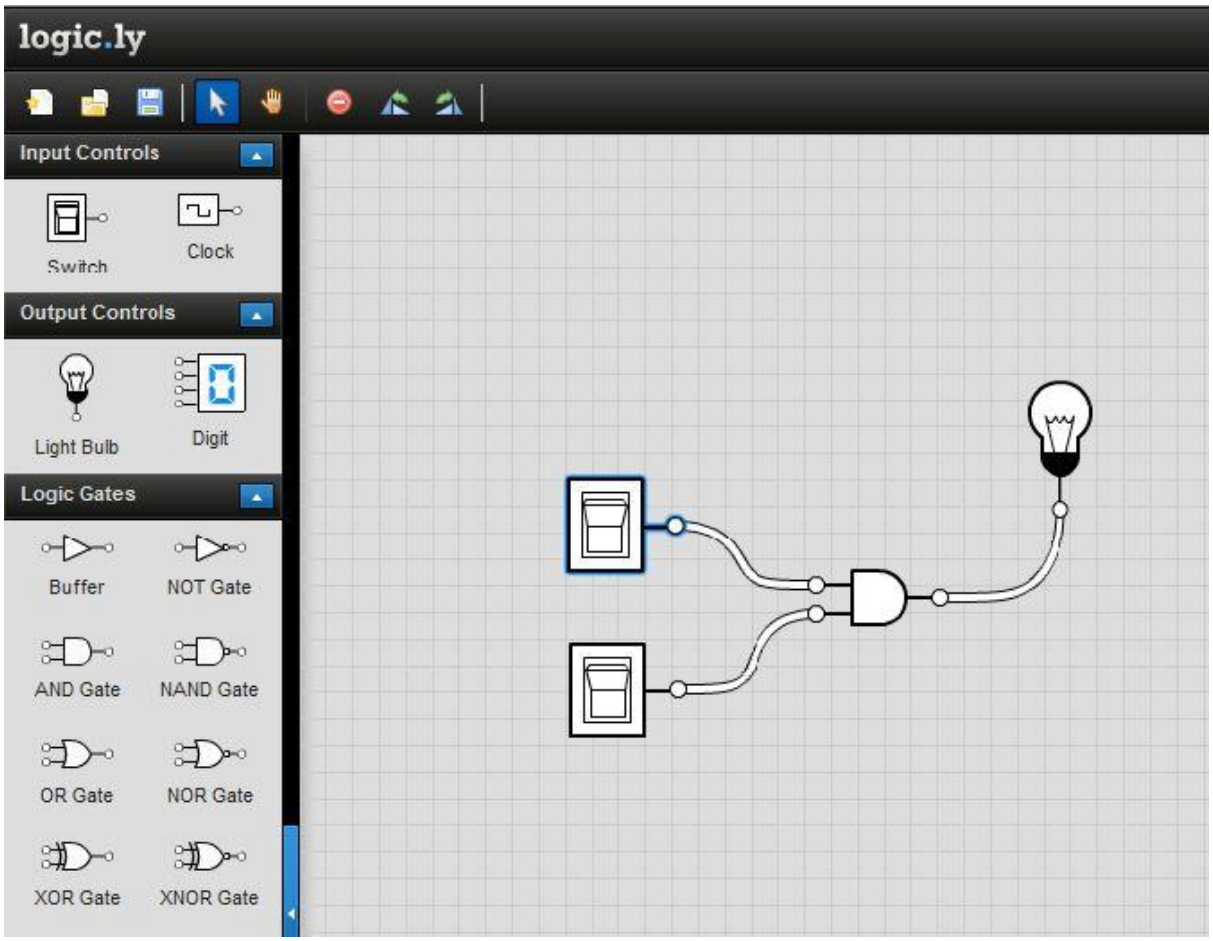
# 2.3 Poorten simuleren

Wanneer we met poorten willen werken kunnen we die natuurlijk van elektronische componenten gaan bouwen. Maar daar is nogal wat voor nodig. De elektronische componenten zijn klein, kwetsbaar en lastig aan te sluiten. Het is handiger daarvoor een simulatieprogramma te gebruiken.

Op de website van **Logicly** vind je een leuk simulatieprogramma voor het werken met poorten. Je hebt er wel Flash bij nodig maar dat staat wel op de meeste computers.

Goed, aan de slag. Start de website van Logicly.

## 2.3.1 De AND-poort nagebouwd




*Afbeelding 11: De AND-poort in Logicly*

***Opdracht 9***

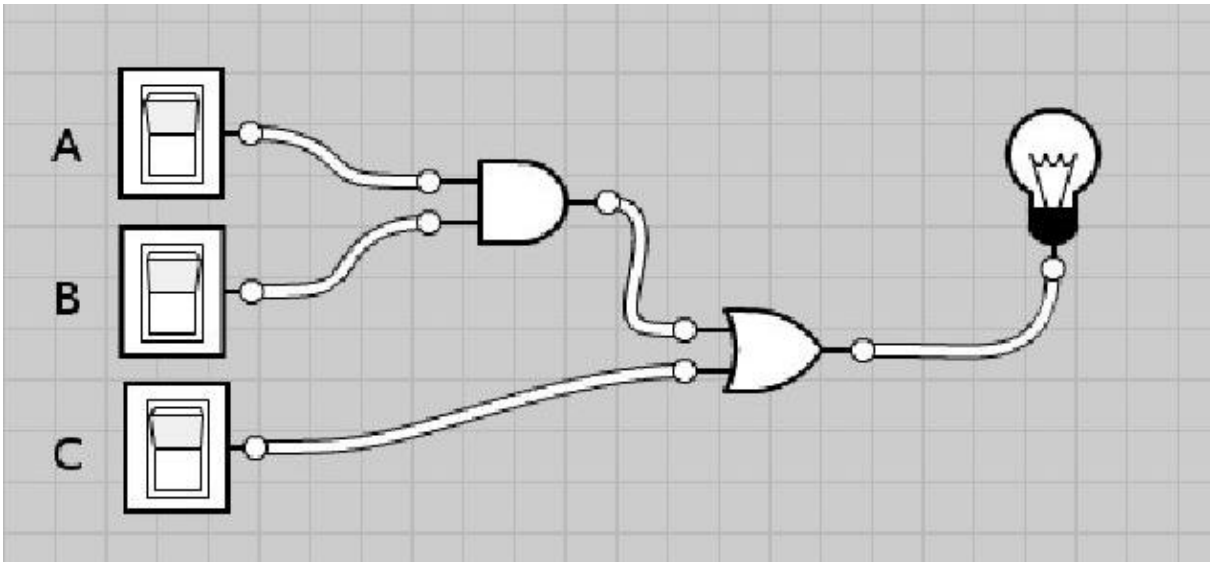
- Bouw de schakeling van de afbeelding door de onderdelen naar het middenscherm te slepen en daarna met de muis de rondjes met elkaar te verbinden.
- Klik nu met de muis op de schakelaars en zie wat er gebeurt.
- Herinner je je de waarheidstabel van de AND-poort nog? Komt die overeen met de simulatie?

| *Switch* | *Switch* | *Lamp* |
|---|---|---|
| *Uit* | *Uit* | *Uit* |
| *Uit* | *Aan* | *Uit* |
| *Aan* | *Uit* | *Uit* |
| *Aan* | *Aan* | *Aan* |

*Ja, zie afbeelding 4.*

### 2.3.2 Poorten koppelen

Zoals we al hebben gezien kunnen we **poorten koppelen**. Uiteraard kunnen we dat ook simuleren.




*Afbeelding 12: Poorten koppelen*

***Opdracht 10***

- Bouw de schakeling van afbeelding 12 na.
- In de waarheidstabel staat een kolom U1. Waar zou dat punt in de schakeling zitten?
- Teken de waarheidstabel over en maak deze af.

| A | B | U1 | C | Lamp |
|---|---|---|---|---|
| 0 | 0 | | 0 | |
| 0 | 0 | | 1 | |
| 0 | 1 | | 0 | |
| 0 | 1 | | 1 | |
| 1 | 0 | | 0 | |
| 1 | 0 | | 1 | |
| 1 | 1 | | 0 | |
| 1 | 1 | | 1 | |

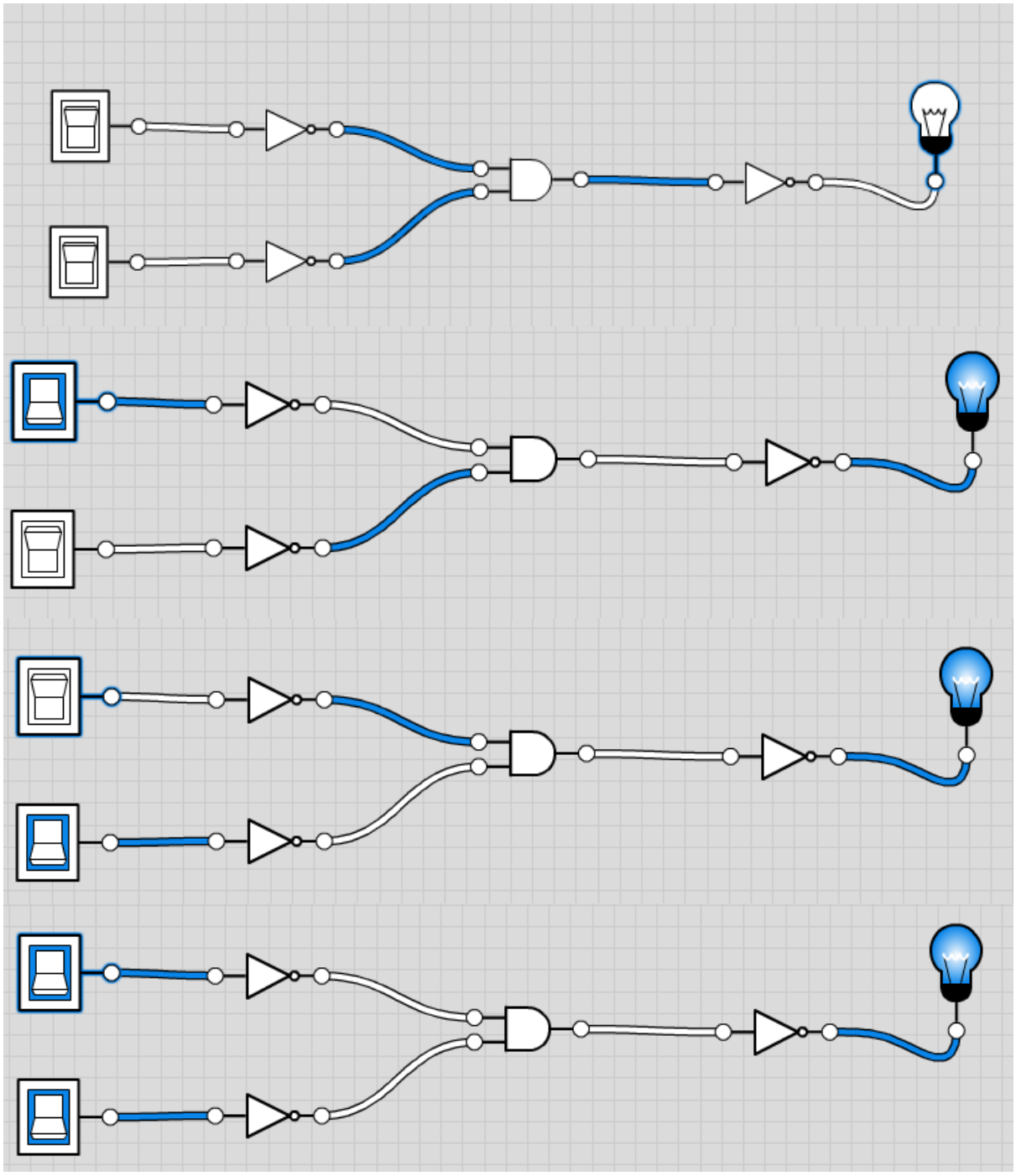
*U1 zit na de AND schakeling van A en B en voor de OR schakeling.*

| *A* | *B* | *U1* | *C* | *Lamp* |
|---|---|---|---|---|
| *0* | *0* | *0* | *0* | *0* |
| *0* | *0* | *0* | *1* | *1* |
| *0* | *1* | *0* | *0* | *0* |
| *0* | *1* | *0* | *1* | *1* |
| *1* | *0* | *0* | *0* | *0* |
| *1* | *0* | *0* | *1* | *1* |
| *1* | *1* | *1* | *0* | *1* |
| *1* | *1* | *1* | *1* | *1* |

***Opdracht 11***

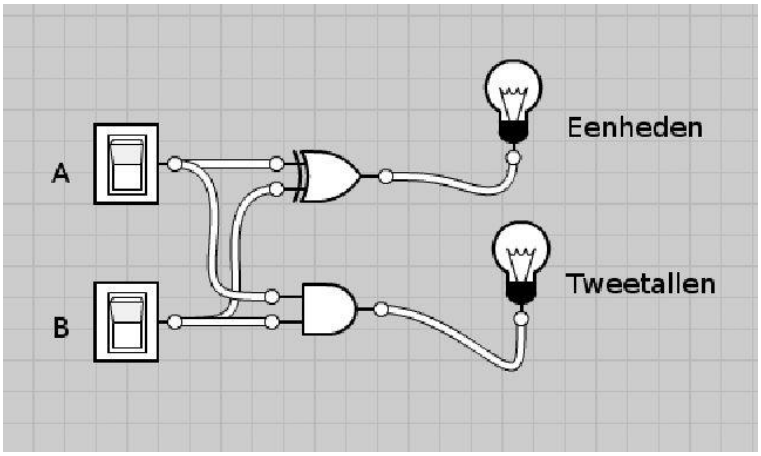
Bij opdracht 8 hebben we een schakeling gezien die opgebouwd was uit één AND poort en drie NOT poorten. Die schakeling reageerde als een OR poort.

- Probeer nu in Logicly een schakeling te bouwen bestaande uit alleen maar OR en NOT poorten maar die reageert als een AND poort.



### 2.3.3 De computer als rekenmachine

We weten dat een computer heel goed binair kan rekenen. We gaan nu proberen of we met logische schakelingen een eenvoudige binaire rekenmachine na kunnen bouwen. Laten we proberen het binaire sommetje 1 + 1 = 10 te simuleren.




A

*fbeelding 13: De half-adder*

***Opdracht 12***

- Bouw de schakeling van afbeelding 13
- Neem de waarheidstabel over en vul deze verder in.

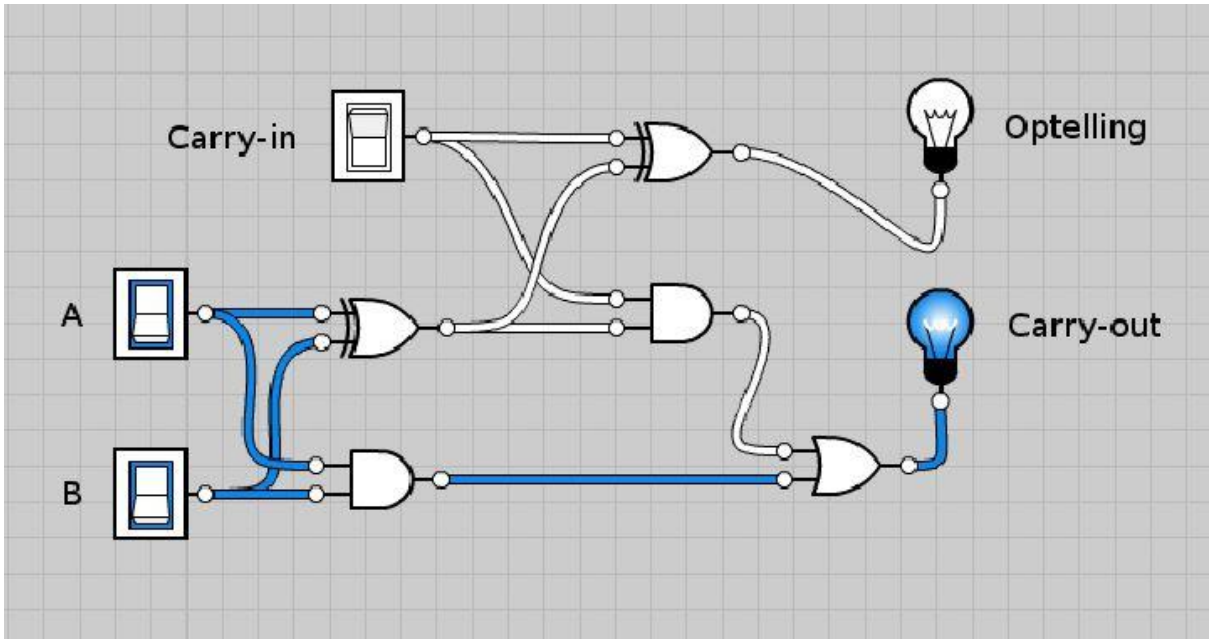
| A | B | Twee-tallen | Een-heden |
|---|---|---|---|
| 0 | 0 | | |
| 0 | 1 | | |
| 1 | 0 | | |
| 1 | 1 | | |

| *A* | *B* | *Tweetallen* | *Eenheden* |
|---|---|---|---|
| *0* | *0* | *0* | *0* |
| *0* | *1* | *0* | *1* |
| *1* | *0* | *0* | *1* |
| *1* | *1* | *1* | *0* |

Kijk nu nog eens naar de waarheidstabel. En neem de getallen bij A en B. Tel die bij elkaar op en kijk dan naar de kolommen tweetallen en eenheden. Staat er bij jou dan ook op de onderste regel: 1 + 1 = 10?

Dat was precies wat we aan wilden tonen. Deze schakeling noemen we een half-adder.

Een half-adder kan alleen maar twee getallen bij elkaar optellen en als daar dan één “onthouden” moet worden (dat noemen we een carry-out), dat doorgeven aan de volgende kolom. Maar een half-adder kan niet van een vorige kolom een “onthouden” getal (dit noemen we een carry-in) meenemen in de optelling. Daarvoor is een full-adder nodig. Bijvoorbeeld bij een binaire optelling van 11 + 11 = 110.




*Afbeelding 14: De full-adder*

***Opdracht 13***

- Bouw de schakeling van afbeelding 14
- Probeer nu uit je hoofd de waarden van de tussenpunten te bedenken en op die manier de waarheidstabel in te vullen.

| A | B | Carry-in | Carry-out | Optelling |
|---|---|---|---|---|
| 0 | 0 | 0 | | |
| 0 | 0 | 1 | | |
| 0 | 1 | 0 | | |
| 0 | 1 | 1 | | |
| 1 | 0 | 0 | | |
| 1 | 0 | 1 | | |
| 1 | 1 | 0 | | |
| 1 | 1 | 1 | | |

| *A* | *B* | *Carry-in* | *Carry-out* | *Optelling* |
|---|---|---|---|---|
| *0* | *0* | *0* | *0* | *0* |
| *0* | *0* | *1* | *0* | *1* |
| *0* | *1* | *0* | *0* | *1* |
| *0* | *1* | *1* | *1* | *0* |
| *1* | *0* | *0* | *0* | *1* |
| *1* | *0* | *1* | *1* | *0* |
| *1* | *1* | *0* | *1* | *0* |
| *1* | *1* | *1* | *1* | *1* |

In de afbeelding en in de waarheidstabel zie je in kleur de situatie waarbij zowel A als B de waarde 1 hebben. Dat is dezelfde optelling als bij opdracht 12.

We zien dat 1 + 1 = 0 waarbij 1 wordt onthouden en naar de carry-out gaat. De carry-out is eigenlijk de verbinding met de volgende schakeling. Deze schakeling noemen we een full-adder omdat hij een getal van de vorige optelling mee kan nemen (de carry-in) en omdat hij ook een getal door kan geven naar de volgende optelling (de carry-out).

Wanneer we nu een half-adder en een full-adder gaan combineren (de tweetallen van de half-adder is de carry-in van de full-adder) kunnen we een optelling als 11 + 11=110 maken. Klinkt ingewikkeld?

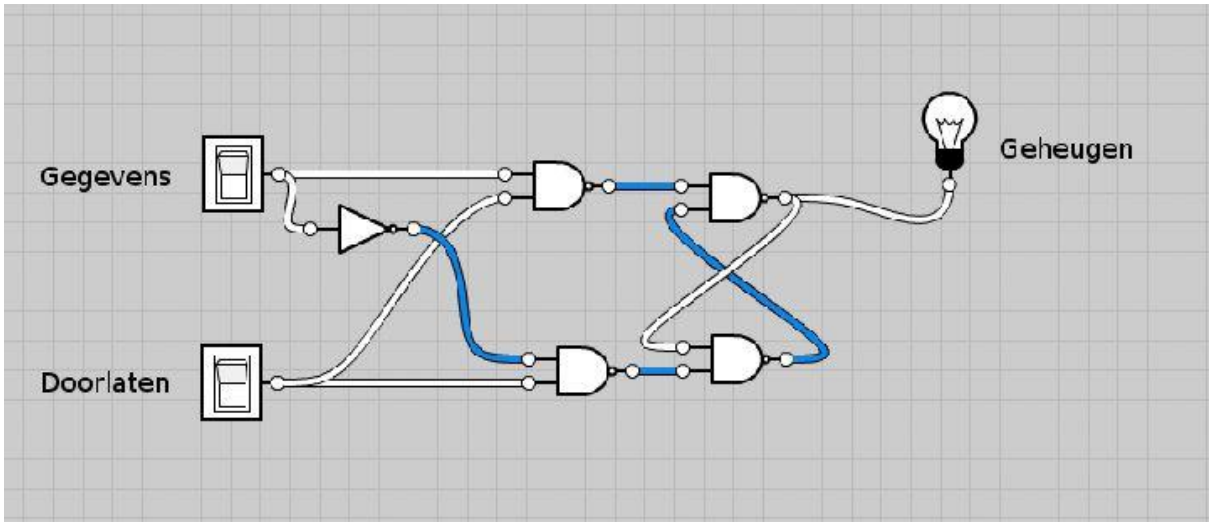
***Opdracht 14***

- Bouw een schakeling waarbij je een half-adder combineert met een full-adder zodat je in de waarheidstabel de optelling 11 + 11 kunt zien.

Probeer het eerst zelf. De oplossing staat **hier**.

### 2.3.4 De flip-flop

Tenslotte de flip-flop. Een grappige naam voor een toch wel lastig te begrijpen schakeling. Kijk maar eens naar afbeelding hieronder.




*Afbeelding 15: De flip-flop*

Om de tekening te vereenvoudigen wordt er hier gebruik gemaakt van NAND poorten. Een NAND poort is een AND poort met daarachter een NOT poort. Je herkent de NAND aan het cirkeltje bij de uitgang. De waarheidstabel van de NAND zie je hieronder.

| A | B | U |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

De flip-flop heeft twee ingangen die op een bepaalde manier samenwerken. We kunnen een flip-flop niet vangen met een waarheidstabel en er op die manier naar kijken want bij een flip-flop is de volgorde van schakelen van belang.

In het kort komt het hierop neer. De Doorlaten ingang bepaalt of de Gegevens ingang wordt doorgelaten. Bouw de schakeling en test het als volgt:

- Zet Doorlaten op 1 en schakel dan aan en uit met Gegevens. Je merkt dat Geheugen dan keurig hetzelfde doet, dus dezelfde waarde aanneemt als Gegevens
- Zorg dat Gegevens op 1 staat, zet dan Doorlaten op 0 en schakel dan met Gegevens. Je merkt dat dan Geheugen niet meer hetzelfde doet als Gegevens. De waarde van Doorlaten bepaalt dus of de waarde van Gegevens wordt doorgestuurd naar Geheugen.

- Zet Doorlaten weer op 1, zet Gegevens op 0 en zet Doorlaten daarna weer op 0. Ook nu zul je zien dat je met alleen Gegevens het Geheugen niet op 1 kunt zetten.

De flip-flop is daarmee een schakeling die vooral gebruikt wordt in geheugen-chips.

***Opdracht 15***

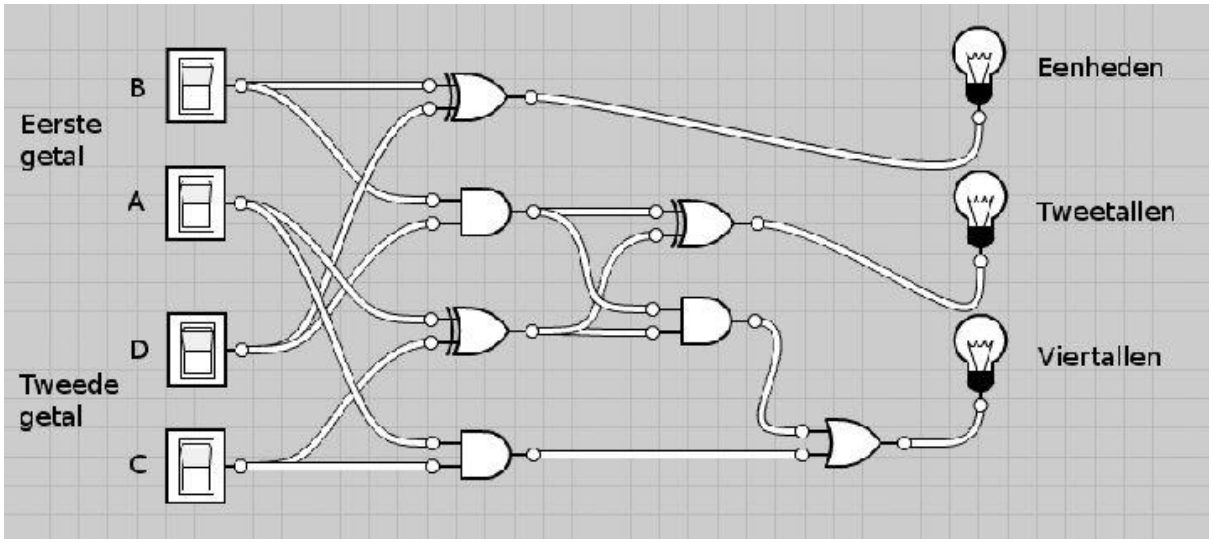
- Toon aan (geef de volgorde van schakelen weer) dat er een situatie kan zijn waarbij zowel Gegevens als Doorlaten op 0 staan en het Geheugen een 1 is maar ook dat er een situatie kan zijn waarbij zowel Gegevens als Doorlaten op 0 staan en het Geheugen een 0 is.

*Zet doorlaten op 1, daarna gegevens op 1, daarna doorlaten op 0 en dan gegevens ook weer op 0. Geheugen is nu 1.*

*Ze doorlaten op 1, daarna gegevens op 0, daarna doorlaten op 0. Geheugen is nu 0.*

### 2.3.5 Uitwerking opdracht 14

Hier eerst de schakeling:




*Afbeelding 16: De rekenmachine voor twee binaire getallen*

In de schakeling zijn de letters A, B, C en D zo gekozen dat die in de waarheidstabel naast elkaar de twee binaire getallen vormen die bij elkaar opgeteld worden. Wanneer we hier opnieuw een full-adder aan koppelen is de aansluiting waar Viertallen bij staat de carry-out voor de volgende schakeling.

Op de volgende bladzijde zie je dan de waarheidstabel.

| Eerste getal | | Tweede getal | | Vier-tallen | Twee-tallen | Een-heden |
|---|---|---|---|---|---|---|
| A | B | C | D | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 |

Voorbeeld uitwerking

$$\begin{array}{r} 10 \\ 11 \\ \hline 101 \end{array} +$$