HOOFDSTUK 1

COMPUTERS EN GEGEVENS

1.1 Mogelijkheden van een computer

Bij een informatiesysteem speelt de computer een centrale rol. Wat kan zo'n computer nu eigenlijk?

Indien men aan verschillende mensen deze vraag stelt, krijgt men verschillende antwoorden. Deze antwoorden zijn tot verschillende groepen terug te brengen al naar gelang de opleiding van degene aan wie men de vraag stelt.

- Een eindgebruiker zonder programmeer-opleiding zal antwoorden : "de computer doet de boekhouding van mijn bedrijf"; "de computer bestuurt de lopende band van mijn bedrijf"; "de computer maakt de rapporten van de studenten"; "de computer voegt in een tekst een andere tekst tussen en herschikt de ganse tekst"; ...
- Iemand die alleen maar heeft leren programmeren, weet dat de computer deze taken aankan, pas nadat hij beschikt over :

  1. gegevens;
  2. een programma met de uit te voeren bevelen.

  Bvb. : om aan het einde van het academiejaar de rapporten van de studenten te kunnen maken :

  1. moeten de uitslagen van de studenten van alle examens zich in de computer bevinden;
  2. moet er ook een programma zijn dat door de computer moet uitgevoerd worden.

  In een dergelijk programma zou kunnen staan :

```
punten = in.readLine();
rapport = punten;
double x = som/aantal;
c = a+b;
```

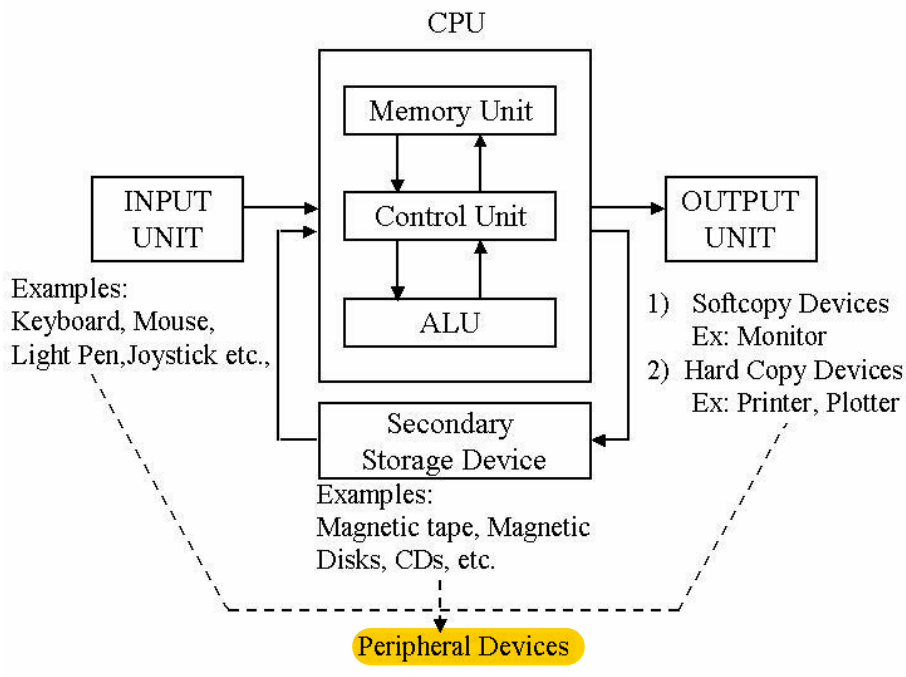
. . .

Deze bevelen worden door de computer uitgevoerd. Het zijn bevelen van de programmeertaal Java.

- Beroeps-informatici weten dat dergelijke bevelen niet rechtstreeks door elektronische apparatuur uitgevoerd worden. Alhoewel ze reeds vrij elementair zijn (vanuit het standpunt van een gewone gebruiker) zijn ze toch nog te complex voor de elektronische apparatuur. De elektronische apparatuur voert zgn. machinebevelen uit. Elk Java-bevel (idem voor bevelen van andere talen als Python, C++, ...) moet vervangen worden door meerdere machinebevelen. In de hiernavolgende tekst vindt men tal van voorbeelden van machinebevelen.

1.2 Basisstructuur van een computer

Indien men de literatuur (in hard copy of online) zou raadplegen over de basisstructuur van een computersysteem, zou heel verschillende uitgebreide schema's krijgen. Echter als men in essentie gaat kijken naar de basis bouwstenen van een computersysteem zou je volgend schema kunnen gebruiken (zie figuur 1.1):



Figuur 1.1: Basisschema Computersysteem

Input/output unit Elke computersysteem heeft wel ooit eens input nodig om verder te kunnen werken of om mee te werken in het algemeen. Denk maar aan het toetsenbord en de muis als voor je computer zit te werken, gamen, ontspannen, ... Of je smartphone/smartwatch waarbij je de sensoren boven op het scherm je vingertoppen en bewegingen registreert om hem aan te sturen. Je kan dit zelfs in de veel bredere zin van het woord zien. Een simpele drukknop of sensor

van een IOT device is ook een input device dewelke bepaalde functionaliteiten van het IOT systeem in gang zetten.
Naast de input is er ook vaak een medium nodig om output te laten generen door het systeem. Zoals onder andere scherm, printer, headset, ...

Memory unit en RAM Het geheugen (Random Access Memory) in een computersysteem wordt voornamelijk gebruik voor het tijdelijk opslaan van gegevens. Dit geheugen wordt rechtstreeks aangestuurd door de CPU (Central Processing Unit) via de Memory controler unit dewelke geintegreerd is in de CPU.

ALU De arithmetical and logical unit is een onderdeel van de CPU dat instaat voor het uitvoeren van standaard wiskundige bewerkingen (+, -, *, /) en ook logische bewerkingen.

Control Unit De control unit in de CPU zorgt ervoor dat de bevelen één voor één van het werkgeheugen naar de CPU gekopieerd worden (waar deze gegevens exact terecht komen, behandelen we verder in deze cursus) en vervolgens uitgevoerd worden.

Storage Naast het snelle werkgeheugen, hetgeen vluchtig van aard is, heeft elk computersysteem ook nood aan een vorm van geheugen dat toelaat om gegevens (van eender welk type) op een "permanente"manier op te slaan.

1.3 Bit and bits

Op elk computersysteem staan er gegevens (onder welke vorm dan ook). Wij als mens zijn gewoon om een bepaald soort gegevens op een bepaalde manier voor te stellen en te begrijpen zoals onder andere geschreven tekst lezen, luisteren naar een audio-fragment, kijken naar een video, ... Echter een computersysteem kan niet rechtstreeks om met deze verscheidenheid aan gegevens. De enigste taal die een computersysteem begrijpt is de binaire taal. Deze bestaat uit twee tekens: 0 en 1 ook wel bit (afkorting van binary digit) genoemd. De manier waarop er gebruik wordt gemaakt van een bit en op welke wijze deze gegroepeerd worden, laat toe om eender welke vorm van gegevens op een computersysteem op slaan of te verwerken.

Met enkel een 0 of 1 is men reeds in staat om een twee zaken voor te stellen. Zo zou mijn met enkel het gebruik van een 0 of 1 de aanwezigheden van elke student tijdens de les kunnen voorstellen: 0 = 'afwezig', 1 = 'aanwezig'

Het gebruik van 1 bit volstaat niet om meer complexere gegevens te bewaren. Echter wanneer we meerder van deze bits zouden combineren, zijn we op een slimmere en krachtigere manier gebruik aan het maken van deze eenvoudige eenheid. Zo zou je perfect gebruik kunnen maken van 3 bits om alle bloedgroepen voor te stellen:

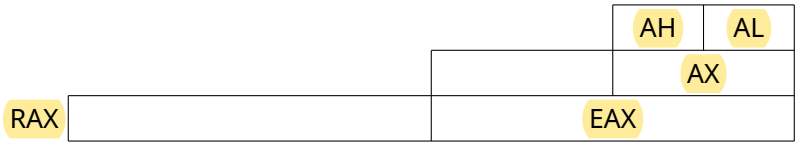
O+	000
O-	001
A+	010
A-	011
B+	100
B-	101
AB+	110
AB-	111

Met 3 bits hebben ook niet genoeg mogelijkheden om complexere gegevens voor te stellen. Echter nu zou het wel duidelijk moeten zijn dat kracht van de bit niet ligt in de bit zelf, maar in het combineren en groeperen van meerdere bits.

Registers De CPU is het kloppende hart van elk computersysteem. Deze moet dus op een vlotte manier om kunnen gaan met al de bits. Hiervoor maakt de cpu gebruik van registers. Dit is een klein stukje geheugen dat de CPU ten allen tijden kan gebruiken om een bepaald aantal bits tijdelijk in op te slaan. De CPU kan deze registers gebruiken voor input of output van bepaalde instructie die hij uitvoert. Elk register heeft een bepaalde lengte: 8, 16, 32, 64, ... Deze lengte geeft aan uit hoeveel bits een register bestaat. In een moderne computer zal dit een 64-bits register zijn. Echter doorheen deze cursus gaan wij ons voornamelijk focussen op een 32-bits register. Hoewel deze lengte niet veel meer gebruikt wordt, laat het ons wel toe om het visueel begrijpbaar te houden. Algemeen heeft een register een aantal eigenschappen:

- De inhoud van een register noemen een we een bitrij
- De lengte van een register ligt vast. Dit is steeds de keuze van de CPU fabrikant.
- De inhoud van een register kan gekopiëerd worden naar een andere plaats
- Wanneer een register een nieuwe inhoud krijgt, is de vorige inhoud verdwenen.

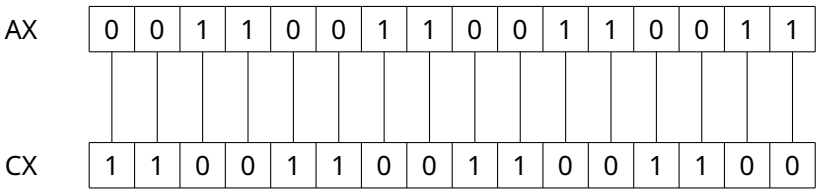
Het aantal registers die een CPU kan gebruiken zijn eigenlijk redelijk beperkt. In totaal zijn er maar 16 registers in een Intel processor. Van deze 16 registers worden er in een Intel processor maar 4 gebruikt voor het uitvoeren van berekeningen. Voor een Intel 32bit processor zijn dat: EAX, EBX, ECX, EDX. Voor een Intel 64bit processor zijn dit RAX, RBX, RCX, RDX. Voor compatibiliteit en oudere programma's te kunnen uitvoeren, is het ook mogelijk om maar een klein deel van het register aan te spreken. Later in de cursus gaan we nog wat dieper op in, echter kunnen we dit nu al even verduidelijken aan de hand van het volgende schema:



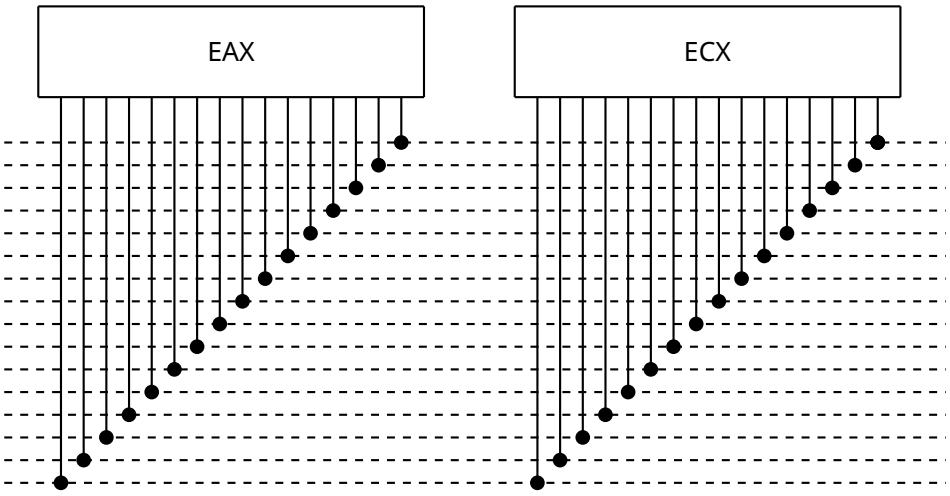
Gegevensoverdacht Zoals in vorige paragraaf reeds is aangehaald, is het mogelijk om de inhoud van een register te kopiëren naar een andere plaats. Dit kan naar het geheugen zijn, SSD, ...

Maar ook naar een andere register(s). Om de bits van het ene register naar het andere register te verplaatsen is er één of andere vorm van verbinding nodig tussen deze registers. Deze verbinding bestaat uit één of meerdere geleiders.

Directe verbinding en busverbinding Uit vorige paragraaf is duidelijk dat er tussen registers een vorm van verbinding is, die het toelaat om bits over te dragen van de ene naar de andere register. Deze verbinding kan bestaan uit een directe verbinding tussen twee registers.



Echter het is niet mogelijk om van elk register een directe verbinding te voorzien naar elk ander register. Dit zou gaan over $12 \times 32 = 384$ verbindingen om enkel alleen directe verbindingen te voorzien tussen alle bits van EAX, EBX, ECX, EDX. Dit is technisch gezien niet gemakkelijk om te verwezenlijken en zou veel ruimte in beslag nemen. Om dit probleem aan te pakken, heeft men de busverbinding ontwikkeld. Bij de busverbinding zijn alle bits van een een register verbonden met een centrale bus. Hierdoor volstaat voor de CPU om het bevel te geven aan register EAX om al (of een deel) van zijn bits op de bus te plaatsen. Vervolgens geeft de CPU het bevel aan register ECX om de bits die op de bus zitten op te nemen. Volgend schema verduidelijkt deze overdracht.



Parallel en seriële bus In de vorige paragraaf hebben we beschreven welke soort verbindingen er bestaan in een computersysteem. Hierbij hebben we het enkel gehad over de fysieke verbindingen die er aanwezig. Van nature uit zijn de verbindingen die we in de vorige paragraaf hebben besproken, parallelle verbinding. Dit wil zeggen dat er voor elke bit een aparte geleider is voorzien en het overzetten van al deze bits gebeurd parallel. Alle bits worden dus tegelijk gekopieerd. Dit is echter redelijk duur omdat er voor elke bit een aparte geleider nodig is. Daarom zijn we tot een seriële bus verbinding gekomen. Bij een seriële bus, gaan we nog steeds gebruik maken van een bus verbinding. Echter in plaats van voor elk bit een aparte bus te voorzien, gaan we alle bits

aansluiten op één centrale bus. Om er nu voor te zorgen dat alle bits kunnen gekopieerd worden, gaan we elke bit één na één op de bus plaatsen.

1.3.1 Getallen in bits

Een computersysteem kan alleen maar werken met 0 en 1. Door meerdere van deze bits met elkaar te combineren, zijn we in staat om gegevens voor te stellen waarmee de computer aan de slag kan gaan. Daar een computer initieel is ontworpen om te helpen bij het uitvoeren van berekeningen, is het ook logisch om hiermee te beginnen. Maar hoe kunnen we nu getallen voorstellen door enkel gebruik te maken van meerdere bits? Laat ons hiervoor beginnen bij meest eenvoudige getallen die we kennen: de gehele getallen. Deze kunnen zowel positief als negatief zijn.

Voor de voorstelling van gehele getallen maken wij gebruik van het decimaal talstelsel. Dit laat enkel toe om gebruik te maken de volgende cijfers: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Om getallen voor te stellen die groter dan 9 zijn, worden deze cijfers naast elkaar geschreven: 4506. Wij weten nu dat dit eigenlijk vierduizend vijfhonderd en zes wil zeggen. Maar ooit hebben we ook geleerd om dit op een wiskundige manier voor te stellen:

$$\begin{aligned} 4506 &= 4000 + 500 + 0 + 6 \\ &= (4 \times 1000) + (5 \times 100) + (0 \times 10) + 6 \\ &= 4 \times 10^3 + 5 \times 10^2 + 0 \times 10^1 + 6 \times 10^0 \end{aligned} \tag{1.1}$$

In het binair talstelsel bestaan er maar 2 cijfers: 0 en 1. Echter men kan dit op net dezelfde manier benaderen zoals het decimaal talstelsel. Waarbij we bij het decimaal talstelsel een getal kunnen opdelen in machten van 10 (omdat er maar 10 cijfers bruikbaar zijn), kunnen we bij het binair talstelsel getallen opdelen in machten van 2 (omdat er maar 2 cijfers kunnen gebruikt worden).

$$\begin{aligned} 00..00011_b &= 1 \times 2^1 + 1 \times 2^0 \\ &= 2 + 1 \\ &= 3 \end{aligned} \tag{1.2}$$

1.3.1.1 Positieve gehele getallen

Voorlopig gaan we ons nog even niets aantrekken van het feit dat een geheel getal ook een teken heeft. We hebben net gezien dat gehele getallen worden voorgesteld door meerdere bits achter

elkaar. Het voorbeeld dat toen aangehaald was, was nog redelijk gemakkelijk. Echter volgend voorbeeld, maakt het al iets moeilijker:

$$\begin{aligned} 00..0101101101_b &= 1 \times 2^8 + 0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 256 + 0 + 64 + 32 + 0 + 8 + 4 + 0 + 1 \\ &= 365 \end{aligned} \tag{1.3}$$

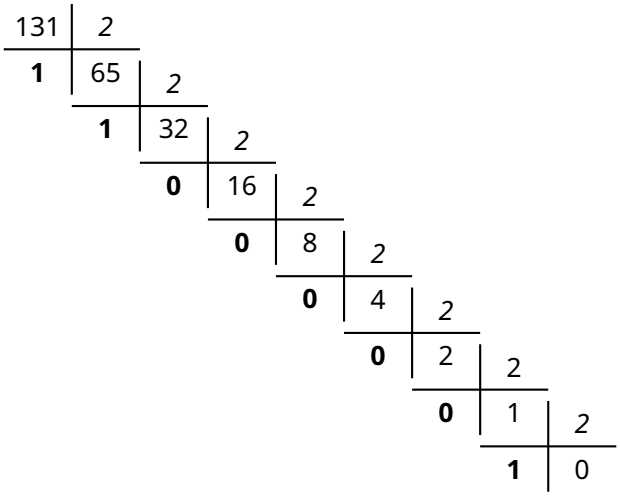
Hier kunnen we reeds een bepaald patroon herkennen. Wie immers vaker met het binaire talstelsel zal werken, zal volgende cijfers snel uit zijn hoofd kennen:

$$\begin{aligned} 2^0 &= 1 \\ 2^1 &= 2 \\ 2^2 &= 4 \\ 2^3 &= 8 \\ 2^4 &= 16 \\ 2^5 &= 32 \\ 2^6 &= 64 \\ 2^7 &= 128 \\ 2^8 &= 256 \\ 2^n &= 2^{n-1} \times 2 \end{aligned} \tag{1.4}$$

Omzetten van decimaal naar binair Tot hiertoe zijn we enkel bezig geweest met het omzetten van binair naar decimaal. Echter het is ook noodzakelijk om een getal te kunnen omzetten van decimaal naar binair. Hiervoor wordt volgend algoritme gebruikt:

- deel het getal door 2, bepaal de rest en quotiënt.
- deel het quotiënt door 2, bepaal de rest en quotiënt.
- blijf dit herhalen tot het quotiënt gelijk is aan 0
- de rij van de resten in omgekeerde volgorde is de binaire voorstelling van het getal.

Laat ons dit algoritme uitvoeren om de binaire voorstellingen van 131 te krijgen:



Indien we de resten in omgekeerde volgorde noteren krijgen we: 10000011. Om zeker te zijn dat dit correct is, gebruiken we de methode om van binair naar decimaal te gaan:

$$\begin{aligned} 10000011_b &= 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= 128 + 0 + 0 + 0 + 0 + 0 + 2 + 1 \\ &= 131 \end{aligned} \tag{1.5}$$

Leidende nullen Net zoals bij het decimaal talstelsel heeft het toevoegen van nullen vooraan het getal geen invloed op de waarde van het getal. In het binair talstelsels is dit immers net hetzelfde.

$$\begin{aligned} 10011_b &= 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= 19 \end{aligned} \tag{1.6}$$

Optellen van binaire getallen Binair optellen heeft veel gelijkenissen met het optellen van getallen in het decimaal talstelsel. Bij het decimaal talstelsel heeft men enkel maar de mogelijkheid om de cijfers 0 tem 9 te gebruiken per plaats in een getal. Wanneer we twee getallen optellen en hierdoor hoger dan 9 komen bij een bepaalde plaats, doen we een overdacht van 1 naar de volgende plaats in het getal. Bij binair optellen, maken we ook gebruik van dit principe. Doordat we enkel maar gebruik kunnen maken van 0 of 1 per bit, zal het principe van overdacht hier veel sneller plaats vinden. Vb 3 optellen bij 47

	1	***1***	***1***	***1***		
1	0	1	1	1	1	
				1	1	+
1	1	0	0	1	0	

2 times Binair vermenigvuldigen is een complexere materie, die voorbij de inhoud van deze cursus gaat. Echter vermenigvuldigen met 2 is een bewerking die we vaker doen. Indien we dit

binair willen doen (dus niet eerst omrekenen naar decimaal, vermenigvuldigen om vervolgens terug naar binair om te rekenen) blijkt dit eigenlijk helemaal niet zo moeilijk te zijn. Stel dat we 10111 willen vermenigvuldigen met 2. Met wat we tot hier toe hebben geleerd kunnen we het volgende stellen:

$$10111_b = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \tag{1.7}$$

Wanneer we dit vermenigvuldigen krijgen we volgende oplossing:

$$\begin{aligned} x &= 2 \times (1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0) \\ &= 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 \\ &= 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\ &= 101110_b \end{aligned} \tag{1.8}$$

Conclusie: Wanneer we een binair getal willen vermenigvuldigen met 2, worden alle bits 1 plaats naar links opgeschoven en wordt er achteraan een 0 bijgevoegd.

1.3.1.2 Negatieve gehele getallen

Tot hiertoe hebben we steeds gewerkt met positieve gehele getallen. Echter elk computersysteem heeft de mogelijkheid om ook met negatieve getallen te kunnen rekenen. Zoals we weten kan een computersysteem enkel en alleen maar werken met bits. Dit wil zeggen dat het computersysteem ten allen tijden moet weten wat het teken is van het gehele getal. Om hier tussen een onderscheid te kunnen maken, zou men als volgt te werk kunnen gaan: in eender welke bitrij, reserveren we het eerste bit om aan te geven of het om een positief of negatief getal gaat. Dit doen we door respectievelijk de 0 te gebruik voor aan te geven dat het om een positief getal gaat en de 1 om aan te geven dat het om een negatief getal gaat. Dit werkt echter wel alleen als de getallen die van elkaar worden afgetrokken een zelfde lengte hebben.

Stel dat de lengte 8 bit zou zijn:

- -3 zou dan voorgesteld worden als 10000011
- 4 zou dan voorgesteld worden als 00000100

Rekening houdend met voorgaande afspraken kan men onmiddellijk zien of het om een positief/negatief getal gaat. Echter deze manier geeft niet het gewenste resultaat:

1	0	0	0	0	0	1	1	
0	0	0	0	0	1	0	0	+
1	0	0	0	0	1	1	1	

$$\begin{aligned} 10000111_b &= -(1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0) \\ &= -(4 + 2 + 1) \\ &= -7 \end{aligned} \tag{1.9}$$

Normaal zou het resultaat 1 moeten zijn, maar dit geeft als resultaat -7. Dit wil zeggen dat deze aanpak niet werkt en er dus een andere manier moet zijn om duidelijk het verschil te maken tussen

een negatief en een positief getal, op zulke manier dat er ook juist mee kan gerekend worden. Indien men als lengte 4 bit zou nemen, heeft men 16 verschillende decimale getallen die men kan voorstellen. Indien we van deze 16 getallen er 8 reserveren voor de positieve en 8 voor de negatieve, kan men volgende cijfers voorstellen:

- -1, -2, -3, -4, -5, -6, -7, -8
- 0, 1, 2, 3, 4, 5, 6, 7

Hoe we de positieve getallen voorstellen blijft nog steeds hetzelfde. Indien we de andere combinaties reserveren voor de negatieve getallen en men nog steeds de eerste bit vrijhoud om het teken te reserveren, zou men tot volgende tabel kunnen komen:

Positieve getallen	
dec.	bin.
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

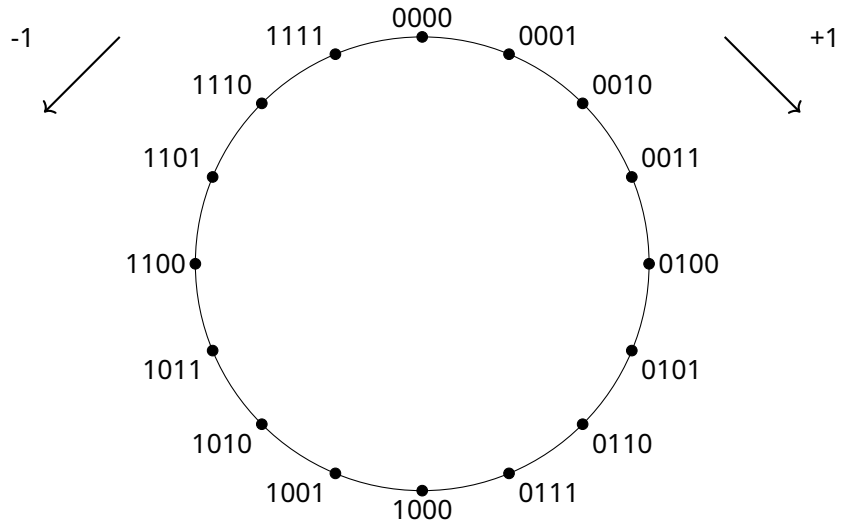
Negatieve getallen	
dec.	bin.
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

Om zeker te zijn dat dit de juiste denkmethode is, volstaat het om willekeurig twee getallen te nemen en hiervan binair de som te maken. vb -1 + 1

$$\begin{array}{cccc} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \quad + \\ \hline 0 & 0 & 0 & 0 \end{array}$$

Hieruit kunnen we dus afleiden dat deze methode wel voldoet aan alle voorwaarden die we voorop hadden gesteld.

Om bovenstaande tabel te verduidelijken, is het ook mogelijk om dit op een meer visuele manier voor te stellen, dewelke duidelijk het cyclische karakter van deze voorstellingswijze weergeeft.



Tweecomplement-voorstelling Wanneer we werken met een klein aantal bits, zou het nog lukken om de negatieve voorstelling van de getallen snel in een tabel op te schrijven. Echter wanneer het aantal bits groter wordt, is het na verloop van tijd niet meer werkbaar om dit op zulke manier op te lossen. Hiervoor moeten er dus een andere methode gevonden worden om de binaire voorstelling van een negatief getal te vinden. De volgende vaststelling zet ons alvast op de goede weg:

- Neem een willekeurige bitrij
- Schrijf het complement op van deze bitrij (vervang elke 0 door een 1 en elke 1 door een 0)
- Tel deze bitrijen met elkaar op. Dit zal altijd een bitrij vormen dat uitsluitend uit 1'en bestaat.

Hieruit kunnen volgende conclusies trekken:

- $(\text{een bitrij}) + (\text{complement van bitrij}) = -1$
- $$\begin{aligned} (\text{een bitrij}) + (\text{complement van bitrij}) + 1 &= 0 \\ (\text{complement van bitrij}) + 1 &= -(\text{een bitrij}) \\ -(\text{een bitrij}) &= (\text{complement van bitrij}) + 1 \end{aligned} \tag{1.10}$$

Conclusie: De binaire voorstelling van een negatief getal vindt men als volgt:

- De binaire voorstelling van de absolute waarde bepalen
- Complement nemen van alle bits
- 1 bijtellen

Dit is wat men noemt de tweecomplements-voorsteling.

1.3.1.3 Reals

Integers zijn niet de enige vorm van cijfers waar een computersysteem mee kan werken. Er moet ook een manier zijn waarop een computersysteem met kommma getallen om kan gaan in binaire vorm. Dit zou men bijvoorbeeld kunnen oplossen door te zeggen dat de eerste 16 bits cijfers voor komma zijn en de tweede 16 bits de cijfers na de komma zijn. Hoewel dit perfect zou werken, is dit geen slim gebruik van het aantal bits die ter onzer beschikking zijn. Om dit op te lossen, heeft men gebruik gemaakt van floating points.

Floating point Bij floating point IEEE-754 representatie laat men toe om de positie van de komma steeds op een andere plaats te zetten. Eigenlijk wordt er zelfs geen rekening gehouden met de komma in deze binaire representatie (waarom wordt later duidelijk). Dit zorgt ervoor dat men efficiënter gebruik maakt van het aantal bits die er ter beschikking zijn en dus veel grotere getallen in deze vorm kunnen noteren. Vooraleer we een komma getal kunnen omzetten naar de binaire IEEE-754 representatie moeten we even teruggrijpen naar de basis wiskunde. De IEEE-754 notatie maakt immers gebruik van de wetenschappelijke notatie van een getal.

De wetenschappelijke notatie laat toe om grote getallen met voornamelijk veel nullen, korter te noteren door deze te noteren als macht van 10 waarbij er telkens maar 1 cijfer voor de komma staat Vb:

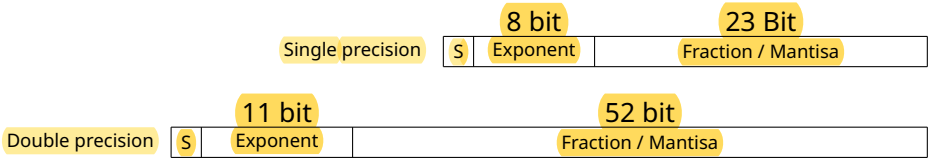
$$45000000 = 4,5 \times 10^{7}$$

De exponent bepaald dus hoeveel plaatsen de komma moet opschuiven. In het geval van een positieve exponent is dit naar rechts. Echter het is ook mogelijk om een- negatieve exponent te hebben. Dit zal er dan voor zorgen dat de komma naar links moet opschuiven. vb:

$$0,0007 = 7 \times 10^{-4}$$

Bij de IEEE-754 notitie maakt men onderscheid tussen 2 vormen:

- Single precision (32 bit)
- Double precision (64 bit)



Zoals je kan merken aan bovenstaand schema, bestaat elke notitie uit 3 vaste delen. De grote van deze delen hangt af van het totale aantal bits.

- **S (sign)** Dit bestaat steeds uit 1 bit. 0 = positief; 1 = negatief
- **Exponent** De exponent wordt steeds tekenloos genoteerd. Om dit te bekomen, wordt er bij de exponent een bias bijgeteld (exponent $=$ real exponent $+$ bias). Bij single precision is dit 127, bij double precision is dat 1023. 00 en FF zijn gereserveerd en kunnen dus niet gebruikt worden.

- **Fraction/Mantisa** Dit bevat alle bits na de komma van de wettenschappelijke notitie (in binaire vorm).

De omzetting van een decimale komma getal naar een IEEE-754 notatie gebeurt in verschillende stappen.

1. Bepaal de sign bit.
2. Converteer zowel het getal voor de komma als het getal na de komma naar zijn binaire vorm.
3. Schrijf het bekomen binaire getal in zijn wetenschappelijke vorm. Er moet steeds 1 voor de komma staan. Alles na de komma vormt de fraction/mantisa. De 1 voor de komma zetten we niet bij in de notatie omdat dit niet nodig is. De RFC zegt immers dat de wetenschappelijke notitie steeds een 1 voor de komma moet bevatten. Dus men kan er altijd vanuit gaan dat er een 1 voor de fraction/mantisa moet staan.
4. Bereken de correct exponent.
5. Zet alle bovenstaande resultaten in het IEEE-754 formaat.

Vb. -263,3 omzetten naar IEEE-754 Single precision:

1. 1 (het getal is negatief)
2. $263,3 = 100000111,01001100110011$
   - $263 = 100000111$
   - De omzetting van een getal na de komma in binaire vorm hebben we nog niet gedaan. Echter dit is niet zo moeilijk. Daar waar we voor de komma steeds moeten delen door 2, gaan we na de komma steeds vermenigvuldigen met 2. Dit zal steeds een 0 of 1 voor de komma als resultaat geven en houden we bij. Daarna trekken we dit van het bekomen resultaat af en vermenigvuldigen terug met 2. Het geheel blijven we herhalen tot het cijfer na de komma 0 is of we een herhaald patroon kunnen herkennen.

$0,3 \times 2$	0,6	0
$0,6 \times 2$	1,2	1
$0,2 \times 2$	0,4	0
$0,4 \times 2$	0,8	0
$0,8 \times 2$	1,6	1
$0,6 \times 2$	1,2	1
		0
		0
		1
		1
		0
		...

3. $100000111,01001100110011 = 1,00000111010011001100110 \times 2^8$
4. $8 + 127 = 135 = 10000111$
5. 1.10000111.00000111010011001100110

Voor de omzetting van IEEE-754 notatie naar decimale notitie kan men gebruik maken van volgende formule:

$$x = (-1)^S \times (1 + \text{Fraction/Mantisa}) \times 2^{\text{Exponent}-\text{Bias}}$$

1.3.2 Hexadecimale getallen

Een nadeel van de binaire voorstelling is dat ze snel lang kan worden. Dit voornamelijk omdat we maar met 2 cijfers mogen werken, waardoor we veel bits moeten combineren om een groot getal voor te stellen. Het decimaal talstelsel maakt het al iets gemakkelijker door het feit dat er hier meer cijfers zijn waarmee we kunnen werken. Ook hier is het nadeel dat de notitie van een getal snel veel cijfers kan innemen om een groot getal voor te stellen. Om hiervoor een oplossing te vinden, bestaat er binnen de informatica ook nog een ander talstelsel: het Hexadecimaal talstelsel. Binnen dit stelsel, mag men gebruik maken van 16 symbolen: Plots kunnen we met 1 symbool de

dec.	hex	bin.
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111

dec.	hex	bin.
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

cijfers 0 tot en met 15 voorstellen.

Hexadecimaal naar decimaal Ook hier kunnen we met dezelfde methode als bij alle geziene talstelsels aan de slag voor het omzetten naar decimaal: opdelen in machten.

$$\begin{aligned} A5B &= A \times 16^2 + 5 \times 16^1 + B \times 16^0 \\ &= 10 \times 16^2 + 5 \times 16^1 + 11 \times 16^0 \\ &= 10 \times 256 + 5 \times 16 + 11 \times 1 \\ &= 2651 \end{aligned} \tag{1.11}$$

Hexadecimaal naar binair We weten dat $A5D = A \times 16^2 + 5 \times 16^1 + D \times 16^0$. Hieruit kunnen we afleiden dat de binaire voorstelling van A5D de som is van:

- de binaire voorstelling van D

$$\begin{aligned} D &= 13 \times 16^0 \\ &= 13 \times 1 \\ &= 1101_b \end{aligned} \tag{1.12}$$

- de binaire voorstelling van 5

$$\begin{aligned} 5 &= 5 \times 16^1 \\ &= 5 \times 16 \\ &= 1010000_b \end{aligned} \tag{1.13}$$

- de binaire voorstelling van A

$$\begin{aligned} A &= 10 \times 16^2 \\ &= 10 \times 256 \\ &= 2560 \\ &= 101000000000_b \end{aligned} \tag{1.14}$$

Bijgevolg is de binaire voorstelling van A5D:

$$\begin{array}{cccccccccccc} & & & & & & & & 1 & 1 & 0 & 1 & \\ & & & & & 1 & 0 & 1 & 0 & 0 & 0 & 0 & \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & + \\ \hline 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & \end{array}$$

Conclusie: vervang ieder hexadecimaal cijfer door zijn binaire voorstelling.

Binair naar hexadecimaal Waarbij we bij het omzetten van hexadecimaal naar binair dit konden doen door gewoon elk hexadecimaal 'cijfer' te vervangen door zijn binaire voorstelling is het omgekeerde ook waar. Bij het omzetten van binair naar hexadecimaal volstaat het om de binaire voorstelling op te delen in groepen van 4 bits en hiervan het overeenkomstige hexadecimaal cijfer te noteren.

$$\begin{aligned} 1001011100111001011_b &= 100 \quad 1011 \quad 1001 \quad 1100 \quad 1011 \\ &= 4 \quad B \quad 9 \quad C \quad B \\ &= 4B9CB \end{aligned} \tag{1.15}$$

Hieruit kan men afleiden dat de hexadecimale voorstelling een afkorting is van de binaire voorstelling. Binnen de informatica wordt de hexadecimale voorstelling veel vaker gebruikt dan de binaiere voorstelling. De inhoud van register RAX is immers veel korter voor te stellen door gebruik te maken van de hexadecimale voorstelling.

Hexadecimale getallen optellen Het optellen van hexadecimale getallen gebeurt op analoge wijze als het optellen van decimale getallen. Wanneer we een som maken van twee decimale getallen, gaan we cijfers die zich op dezelfde plaats bevinden optellen. Indien men hierbij meer dan 10 uitkomt, wordt de 1 overgedragen naar het volgende cijfer. Bij hexadecimale cijfers kunnen we gaan van 0 tot en met 15. Ook hier gaan we de cijfers op dezelfde plaats individueel optellen. Indien we hier meer dan 15 uitkomen, gaan we ook een overdracht doen naar het volgende cijfers. Dit zal ook steeds een 1 zijn. Volgende voorbeelden maken dit duidelijk.

Vb 1:

$$\begin{array}{r} \text{A32} \\ \text{495} \quad + \\ \hline \text{EC7} \end{array}$$

$$\begin{aligned} 2 + 5 &= 7 \\ 3 + 9 &= 12_d = C \\ A + 4 &= 10_d + 4_d = 14_d = E \end{aligned} \tag{1.16}$$

Vb 2:

$$\begin{array}{r} \text{C3A} \\ \text{DB9} \quad + \\ \hline \text{19F3} \end{array}$$

$$\begin{aligned} A + 9 &= 10_d + 9_d = 19_d = 16_d + 3 = 3 + 1 \text{ overdragen} \\ 1 + 3 + B &= 4 + 11_d = 15_d = F \\ C + D &= 12_d + 13_d = 25 = 16_d + 9 = 19_h \end{aligned} \tag{1.17}$$

1.3.3 Tekst in bits

In een computergeheugen is er alleen plaats voor nullen en enen. Hoe kan men dan letters en andere tekens (zoals bvb. leestekens) bewaren?

1.3.3.1 ASCII

Het antwoord op bovenstaande vraag is : de symbolen A, B, C, ..., Z, a, b, ..., z, 0, 1, 2, ... worden in code-vorm bewaard. Bijna alle machines gebruiken ASCII (= American standard code for information interchange). Bij deze codering wordt iedere letter, lees- en ander teken voorgesteld door 8 bits (of 2 hexadecimale cijfers).

Enkele voorbeelden :
41, 42, 43, ... : A, B, C, ...
61, 62, 63, ... : a, b, c, ...
31, 32, 33, ... : 1, 2, 3, ...
20 : b (spatie)

1.3.3.2 Unicode en UTF-8

In ascii kan men 256 verschillende tekens voorstellen : 8 bits per teken. Oorspronkelijk was de eerste bit altijd een 0, er waren dus maar 128 tekens. Deze verzameling symbolen wordt Latin genoemd. Voor het Engels is dit voldoende. Het Engels (en in mindere mate ook het Nederlands) heeft wel de vervelende eigenschap dat men aan een woord niet kan zien hoe het moet uitgesproken worden. Andere talen zoals het Frans of het Duits hebben veel minder ambiguïteiten omdat

er zgn. diakritische tekens gebruikt worden. Een diakritisch teken geeft aan hoe een letter moet uitgesproken worden, bvb. : é, è, ü, Ä, Å, … Om een waarde te kunnen toekennen aan é, è, ê, à, â, … wordt in ascii een byte gebruikt die met een 1 begint. Dit laat meteen toe om 128 letters te voorzien van een accent of een ander diakritisch teken. De set van tekens (Latin + de 128 nieuwe tekens) wordt Latin-1 genoemd. Hiermee is het voor West-Europa in orde. Maar wereldwijd gezien, staan we hiermee nog nergens. Een nieuwe standaard, unicode, is ontwikkeld. Per teken worden 16 bits gebruikt. Goed voor 65.536 tekens. Deze worden als volgt gebruikt:

- 0000–007F : Latin
- 0080–00FF : Latin-1, bvb. : (00F5)h = õ
- …
- 0370–03FF : Grieks en Koptisch, bvb. (03F4)h =
- 0400–04FF : Cyrillisch, bvb. (03F4)h =
- enz…

Voor bvb. Devanagari, Bengali, en andere exotische tekens, zie http://unicode.org/charts/PDF/U0000.pdf
Wellicht zal de unicode met 2 bytes ook niet voldoen.