Computersystemen en embedded systemen

# Computersystemen en embedded systemen

4e druk

Leo van Moergestel

Boom


Met behulp van onderstaande unieke activeringscode kunt u toegang krijgen tot *www.Academicx.nl* voor extra materiaal. Deze code is persoonsgebonden en gekoppeld aan de 4e druk. Na activering van de code is de website 3 jaar toegankelijk. De code kan tot zes maanden na het verschijnen van een volgende druk geactiveerd worden.

Meer informatie over deze en andere uitgaven kunt u vinden
via www.boomuitgeversamsterdam.nl



1e druk 2002
5e druk 2016

Ontwerp en zetwerk binnenwerk: Studio Bassa, Culemborg
Omslagontwerp: Carlito's Design, Amsterdam

ISBN 9789058754233
NUR 123

# Voorwoord

Dit boek probeert de lezer een gedegen basiskennis op het gebied van computertechniek en embedded systemen bij te brengen. Omdat computers meestal niet op zichzelf staan maar in een netwerk opereren, is ook de nodige aandacht aan datacommunicatie en netwerken geschonken. De inhoud beperkt zich niet tot een bepaalde klasse van computers, maar probeert steeds een zo breed mogelijke blik te bieden. Veel concrete voorbeelden komen uit de wereld van de personal computer, omdat de meeste lezers daar wellicht zelf ervaring mee hebben. Voor embedded systemen hebben we herkenbare voorbeelden uit het dagelijks leven gekozen.

Dit boek is prima te gebruiken bij een cursus Inleiding computerarchitectuur daarnaast biedt het een goede basis voor lezers die zich willen verdiepen in de technologie van embedded systemen. Het kan als leidraad of naslagwerk dienen. Het boek is ook geschikt voor zelfstudie. Met vraagstukken aan het eind van ieder hoofdstuk kan de lezer zijn opgedane kennis testen.

Deze geheel herziene druk is volledig geactualiseerd, met onder andere aandacht voor security, datacompressie, virtualisatie, cloud computing en praktijkvoorbeelden van bv. smartphones (Android, iOS en Windows Phone).

Er bestaan, afhankelijk van wat de lezer wil leren, verschillende leertrajecten door dit boek, die bepaalde onderwerpen meer of minder belichten. Het hoeft dus niet per se van voren naar achteren doorgewerkt te worden en sommige onderdelen kunnen worden overgeslagen zonder dat de rest van het boek onbegrijpelijk wordt. Een drietal zwaartepunten zal ik hier schetsen:

1. *Embedded systems*
   Bedoeld voor lezers die kennis van embedded systems en computerarchitectuur willen opdoen: hoofdstuk 1, 2, 3, 4, 6, 7, 8, 9, 10, 13, 15, 17, 18 en 19.
2. *De helicopterview*
   Voor lezers die een brede blik op de computer- en netwerktechnologie willen hebben: hoofdstuk 1, 2, 3, 4, 7, 8, 9 10, 11, 12, 13, 15, 17 en 18
3. *De netwerktechnologie*
   Die gedeelten van het boek die met datacommunicatie en netwerken te maken hebben: hoofdstuk 1, 2, 3, 4, 10, 15, 17 en 18.

Natuurlijk blijft altijd de mogelijkheid bestaan het hele boek van A tot Z door te nemen of een eigen selectie van de onderwerpen te maken. Het probleem met computerboeken is dat ze meestal verouderd zijn op het moment dat de drukinkt droog is. Dit probleem heb ik op twee manieren proberen te omzeilen. Ten eerste behandel ik concepten en technieken die al geruime tijd hun waarde bewezen hebben en probeer niet te specifiek op numerieke gegevens over de laatste stand van de techniek in te gaan. Het noemen van zaken als snelheden is echter niet geheel te vermijden. Ook verdienen nieuwe ontwikkelingen hun plaats in dit boek. Voor concrete voorbeelden wijk ik vaak uit naar

kleine computers met het idee dat de lezer zelf wel in staat is een schaalvergroting toe te passen. Een 8-bits computer verschilt qua werking niet essentieel van een systeem met een 64-bits databus. De plaatjes worden wel overzichtelijker door deze schaalverkleining. De tweede manier om dit boek actueel te houden is ondersteuning via internet. Relevante artikelen, documentatie, uitwerkingen van opgaven en leertrajecten zal ik via internet beschikbaar stellen, waar nodig met verwijzingen naar hoofdstukken uit dit boek. Vooral de uitwerkingen van en een aanvulling op de opgaven uit dit boek kunnen voor zelfstudie nuttig zijn.

Voor docenten zijn o.a. de figuren beschikbaar in pdf-formaat. Het aanvullende materiaal is beschikbaar via de pagina bij dit boek op www.academicservice.nl.

Rest mij nog iedereen te bedanken die op de een of andere wijze aan de totstandkoming van dit boek heeft bijgedragen. In het bijzonder wil ik mijn collega Peter Kramer bedanken voor de vele informatieve gesprekken die we over embedded systemen gevoerd hebben.

Zaandam, januari 2016,
Leo van Moergestel

# Inhoud

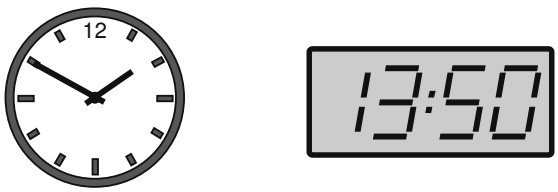
Hoofdstuk 1

# Inleiding

In deze inleiding komt een drietal onderwerpen aan bod. Eerst gaan we in op de begrippen analoog en digitaal. Vervolgens komt de twee-eenheid hard- en software aan bod. Als laatste onderwerp gaan we kijken waar en hoe computertechnologie zoal wordt ingezet.

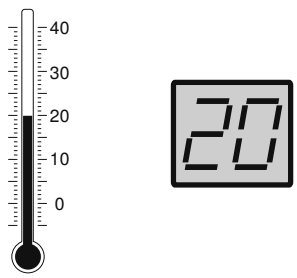
## 1.1 Analoog en digitaal

Het onderscheid tussen analoog en digitaal is het duidelijkst te zien aan de hand van een paar voorbeelden. Bekijk de analoge en digitale klok van figuur 1.1. De analoge klok heeft een wijzerplaat en de wijzers geven het tijdstip aan. De digitale klok geeft een getal dat een maat is voor de tijd.




**Figuur 1.1** *Analoge en digitale klok*

Analoge technieken worden meer en meer verdrongen door digitale technieken. Een analoog meetinstrument heeft meestal een schaal met daaraan gekoppeld een wijzer of iets dergelijks. Een tweede voorbeeld is een thermometer (zie figuur 1.2).



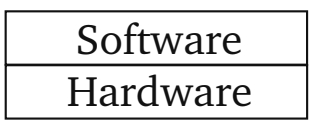

**Figuur 1.2** *Analoge en digitale thermometer*

Een analoge thermometer bevat een kolom met een vloeistof waarvan de hoogte analoog is aan de temperatuur. Een schaal geeft aan wat de temperatuur is. De digitale tegenhanger geeft als resultaat enkel een getal.

Digitale systemen werken met getallen. Bij computers blijken deze getallen uit het *tweetallig stelsel* te komen. Dit wordt vaak populair vertaald naar de uitspraak dat een computer maar tot twee kan tellen. Dat is natuurlijk onzin, mensen die het tientallig stelsel gebruiken kunnen wel degelijk verder tellen dan tien, zeker als bijvoorbeeld over salaris onderhandeld wordt.

## 1.2 Hardware en software

*Hardware* is tastbaar. De elektronica, de computerkast en allerlei mechanische onderdelen van een computer behoren tot de hardware. *Software* is de programmatuur. Een USB-stick, die zelf vanwege zijn tastbaarheid tot de hardware behoort, kan wel software bevatten. Deze twee kunnen niet zonder elkaar. Een computersysteem zonder software is zinloos. Een programma zonder computer heeft ook weinig nut. De analogie met een boek gaat hier aardig op. Papier en drukinkt waarmee letters en afbeeldingen zijn gedrukt, vormen de hardware. De software is het verhaal of de inhoud van het boek. Net zoals een schrijver auteursrecht op een verhaal heeft, heeft de softwareontwikkelaar auteursrecht op zijn product. Een probleem met software is dat het vrij simpel te kopiëren is zonder verlies van kwaliteit. Het vrij kopiëren van commerciële software is niet toegestaan.
Toch is er ook software die wel vrij te kopiëren is. Ook bij deze software horen auteursrechten. Voorbeelden van vrije software zijn het Linux-systeem en veel daarbij meegeleverde software die valt onder de GNU Public Licence (GPL).
In figuur 1.3 is het verband tussen software en hardware in een lagenmodel weergegeven. De hardware vormt het platform waarop de software draait. In de Nederlandse taal gebruiken we het werkwoord *draaien* waar in het Engels het werkwoord *to run* wordt gebruikt. Dit lagenmodel bestaat hier uit slechts twee lagen, maar verderop zullen we daar verfijningen op aanbrengen.




**Figuur 1.3** *Lagenmodel*

Soms is software al in elektronische componenten ingebakken. In dat geval spreekt men vaak van ‘firmware’. Deze ingebouwde software bestuurt bijvoorbeeld allerlei elektronische apparatuur.

## 1.3 Computers in onze samenleving

Toen de eerste computers werden gebouwd, waren ze groot en duur en werden hoofdzakelijk gebruikt voor rekenwerk. De computerrevolutie is begonnen met de komst van de microprocessors in het begin van de jaren zeventig van de twintigste eeuw. Computers werden klein en goedkoop. Allerlei toepassingen die weliswaar mogelijk

waren maar voorheen veel te duur, werden met de microprocessor realiseerbaar.
We zullen eens kijken waar we computers zoal tegenkomen en beginnen gewoon bij huis. Een modern gezin beschikt tegenwoordig vaak over één of meer tablets, smartphones en/of personal computers (pc's). Deze systemen kunnen voor spelletjes, tekstverwerking, hobby, en communicatie met andere computers gebruikt worden. Voor spelletjes zijn overigens ook speciale spelcomputers te koop, die voor beeldweergave een tv gebruiken en onder meer daardoor wat goedkoper zijn dan een pc. Deze systemen worden wel gemakkelijk als computer herkend. Anders ligt het bij elektronische apparatuur en huishoudelijke apparaten. Niet iedereen beseft dat hiervan de besturing en bediening met weliswaar kleine computersystemen, ook wel *microcontrollers* genaamd, gebeurt. Ook in de moderne auto is de microcontroller binnengedrongen en speelt een rol bij het goed laten functioneren van de motor en allerlei in de auto verwerkte beveiligings- en diagnosesystemen. In de transportwereld is de computer onder meer bekend als de 'automatische piloot' in vliegtuigen. Het spoorwegnet wordt met computers geregeld en beveiligd en natuurlijk treft men ook computersystemen aan op schepen. Het autonavigatiesysteem is een gespecialiseerde computer die met behulp van spraak en beeld de bestuurder helpt bij het vinden van de eindbestemming. De stap naar zelfrijdende auto's is inmiddels ook al gezet.

In de kantoorwereld is de pc niet meer weg te denken en elk bedrijf gebruikt computers voor administratieve en financiële toepassingen. In de industrie heeft de computer zijn weg gevonden voor de automatisering en besturing van productieprocessen. In ziekenhuizen wordt geavanceerde apparatuur bestuurd met computers of maakt men gebruik van computertechnologie om deze te kunnen laten functioneren. Het klimaat in grote gebouwen wordt geregeld met gebouwbeheersystemen die ook weer microcomputers gebruiken voor het verzamelen van meetgegevens en het bijsturen van de diverse beheersystemen.

Natuurlijk is de computer ook ingezet in militaire installaties en intelligente wapensystemen. Naast het gebruik in wapensystemen vinden we ook computers die informatie versleutelen om te voorkomen dat buitenstaanders deze informatie kunnen bereiken. Eventueel onderschepte versleutelde informatie (van de vijand) kan weer met computers onderzocht worden om een geheime boodschap te reconstrueren. Dit kraken van codes was tijdens de Tweede Wereldoorlog een belangrijke drijfveer voor het ontwikkelen van speciaal op deze taak gerichte geautomatiseerde systemen.

Wereldwijd zijn computers in netwerken gekoppeld voor uitwisseling van gegevens tussen allerlei verschillende computersystemen. Deze ontwikkeling, bekend onder de naam internet, is in de jaren zestig van de vorige eeuw begonnen als militaire toepassing, maar is al geruime tijd in veel huiskamers binnengedrongen. Naast het vinden van informatie via internet (websurfen), het uitwisselen van gegevens (e-mail, chatten), zien we steeds meer toepassingen die met internettechnologie gerealiseerd worden. Social media zoals Facebook en Twitter spelen inmiddels een belangrijke rol. De computersystemen worden juist door de verbinding met internet veelzijdiger en krachtiger. De mogelijkheden van internet blijven nog steeds toenemen. Cloudcomputing is daar een aansprekend voorbeeld van. Daarnaast zorgen smartphones ervoor dat ieder-

een overal internettoegang heeft. Steeds meer apparaten worden met een netwerk verbonden en hebben op deze wijze mogelijkheden om te communiceren. Dit zogenoemde internet of things (IoT) zal een grote invloed hebben op de toekomstige ontwikkelingen van internet.

Natuurlijk blijft de mens als gebruiker van de informatiesystemen bepalen op welke wijze ze worden ingezet. Een kwaadwillend regime dat lak heeft aan privacy zou kunnen besluiten alle burgers vanaf de geboorte van een RFID-tag te voorzien. Een dergelijk plan zou naar men mag hopen de nodige weerstand ondervinden. De techniek biedt mogelijkheden, het is aan ons haar op een goede manier in te zetten. Dit geldt natuurlijk ook voor de ontwikkelaars van software. Het toevoegen van eigenschappen aan software waar de eindgebruiker niet van gediend is, mogelijk mee misleid wordt en ook niet van op de hoogte wordt gebracht, heeft in 2015 voor de nodige opschudding gezorgd. Het betrof hier firmware (ook wel embedded software genoemd) in auto's. Een van de gevolgen is dat de Nederlandse taal in 2015 verrijkt is met het woord sjoemelsoftware, dat zelfs werd uitgeroepen tot 'woord van het jaar'.

## Hoofdstuk 2

# Computersystemen van groot naar klein

We gaan in dit hoofdstuk eerst eens kijken wat we nu precies onder een computer verstaan en in welke vormen we computers aantreffen. Ook bekijken we in vogelvlucht de werking van computers in het algemeen en zullen aan de hand van wat voorbeelden een aantal specifieke computergerelateerde begrippen leren kennen. We eindigen met beschouwingen over computernetwerken, embedded systemen en software.

## 2.1 Standaardcomputers

De eerste associatie die de meeste mensen tegenwoordig met computers maken, is die met de personal computer, ook wel pc genoemd. In veel huishoudens is dit een gebruiksvoorwerp, vergelijkbaar met televisie of andere elektronische apparatuur. De portable versie van de pc in de vorm van de laptop heeft de desktopversie inmiddels al op veel plaatsen vervangen. De pc is veelzijdig en kan voor allerlei taken ingezet worden, maar meestal is de toepassing in de huiselijke sfeer spelletjes spelen, internetten of het doen van administratieve taken zoals tekstverwerking en aanverwante toepassingen (spreadsheet voor berekeningen en database voor gegevensopslag). De veelzijdige inzetbaarheid danken pc's en eigenlijk computers in het algemeen aan het feit dat het programmeerbare systemen zijn. Een programma bepaalt de werking. Een tekstverwerkingsprogramma geeft aan een computer een heel andere functionaliteit dan een schaakprogramma. Een programma bestaat uit een reeks eenvoudige instructies die door het systeem worden uitgevoerd. De kracht van een systeem zit hem onder meer in het feit dat per seconde enorm veel instructies kunnen worden uitgevoerd.
Bij pc's onderscheiden we de programmatuur, ook wel software genoemd, die het systeem laat functioneren en een basis voor toepassingen vormt. Deze software noemen we het *besturingssysteem* of *operating system*. Voorbeelden uit de pc-wereld hiervan zijn MS-windows, MacOS en Linux. De eindgebruiker is meestal meer geïnteresseerd in de toepassingen die op die operating systems kunnen draaien. Hier komen dus de toepassingen als tekstverwerking, database, spellen, webbrowsers enzovoort om de hoek kijken.

Systemen die veel informatie aan een netwerk kunnen geven, de zogenoemde servers, stijgen meestal qua prestatie boven een pc uit. Hele grote systemen zoals mainframes en supercomputers vormen de top qua systeemprestatie (en prijs). Al deze systemen hebben met de pc gemeen dat ze een operating system kennen waarop de voor het gebruik noodzakelijke toepassing in de vorm van software draait. Overigens worden deze zware systemen vaak samengesteld uit een combinatie van computers die samen als een geheel functioneren. Kijken we naar de systemen die wat prestatie betreft in de buurt van een pc zitten, dan komen we bij apparaten die meestal ook wel als computer

herkend worden. Denk bijvoorbeeld aan spelcomputers. De naam geeft al aan dat het hier om computers gaat. Tegenwoordig is de term gameconsole wat gebruikelijker. Tablets en smartphones zijn computersystemen die min of meer vergelijkbaar zijn met pc's. Deze categorie is duidelijk voor het gebruikersgemak ontworpen en kent inmiddels al een enorme variëteit aan software, de zogenoemde apps. In veel technische toepassingen treffen we computers aan die een onderdeel van een groter geheel zijn. In deze embedded systemen is de computer een component van een groter geheel.

## 2.2 Embedded systemen

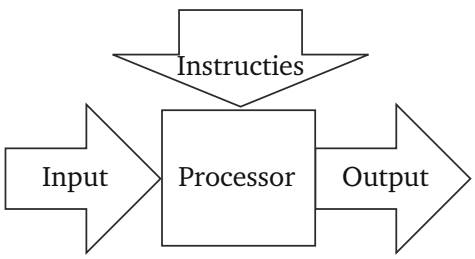
Een plaats waar computers min of meer sluipend hun intrede hebben gedaan is in de embedded systemen. Hier zitten computersystemen verstopt in apparatuur, gebruiksvoorwerpen of technische systemen (zoals auto's) zonder dat de gebruiker zich daar direct bewust van is. Vooropgesteld moet worden dat dit geen definitie is van embedded systemen. Bij sommige embedded systemen kan de eindgebruiker zelf nog 'programmeren'. Bij de zogenoemde deeply embedded systemen is dat niet meer mogelijk. Deze computers mogen eigenlijk niet falen, omdat daarmee de werking van het systeem waarin ze 'ge-embed' zijn, onderuitgehaald wordt. In praktijk kan dit natuurlijk nooit voor honderd procent zo zijn, daarom past men technieken toe die bekend staan onder de naam fault tolerance. Een fault tolerant systeem is in staat eventueel optredende fouten op te vangen of op eigen kracht te herstellen.
Twee eigenschappen zijn kenmerkend voor embedded systemen:

1 User interface of gebruikersinterface: dit is de wijze waarop de gebruiker met het systeem omgaat. De algemene term om de interactie tussen gebruikers en computers aan te duiden is mens-computer-interactie ofwel human-computer interaction (HCI).
2 Doel van het systeem: welke taken moet het systeem uitvoeren.

In deze twee eigenschappen onderscheiden embedded systemen zich van algemeen inzetbare computers. We zullen dit nader bekijken. Standaardcomputers hebben een gebruikersinterface die bij laptops gebaseerd is op een toetsenbord, (aanraak)beeldscherm en eventueel een muis. Een embedded systeem heeft vaak een interface die hiervan sterk afwijkt of soms helemaal geen interface. Een of enkele drukknoppen, een eenvoudig aanraakscherm (touchscreen), één of meer lampjes (ledjes) vormen de user interface. Sommige embedded systemen hebben helemaal geen directe interactie met een gebruiker. Dit lijkt wat moeilijk voor te stellen, maar het gaat hier om computers die informatie van elektronische opnemers binnenkrijgen en daarop op een voorgeschreven manier reageren. Ook computers die netwerkverkeer in een datacommunicatienetwerk regelen, hebben vaak alleen netwerkaansluitingen en geen directe user interface.
De tweede eigenschap van embedded systemen waarin ze zich onderscheiden van standaardcomputers, is het doel van een embedded computersysteem. Een standaardcomputer is veelzijdig inzetbaar en kan een grote diversiteit van functies uitvoeren. Denk maar aan een laptop. Die is op het ene moment een tekstverwerker, het volgende ogenblik een spelmachine en daarna of wellicht tegelijk een afspeler voor muziek. Een embedded systeem is gebouwd en geprogrammeerd voor één taak of hoogstens voor een klein aantal specifieke taken.

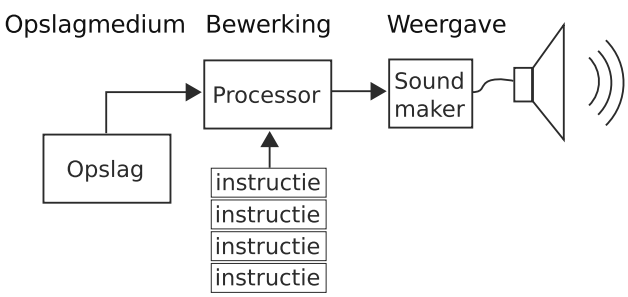
## 2.3 Computereigenschappen

Laten we eens kijken wat al deze computers gemeenschappelijk hebben. We hebben al gezien dat ze programmeerbaar zijn, maar wat betekent dat? Alle computers, van klein tot groot, voeren instructies uit: ze voeren een programma uit. Een programma is een instructiestroom die door het hart van het systeem wordt uitgevoerd. Dit hart van de computer is de processor. Vanwege de centrale rol die de processor heeft, wordt deze ook wel CPU ofwel central processing unit genoemd. De Nederlandse term hiervoor is centrale verwerkingseenheid, afgekort tot CVE. Dit onderdeel voert de instructies waaruit een programma bestaat uit. Daarnaast hebben we nog te maken met gegevens (data) waarop het programma zijn bewerkingen uitvoert, de input, en gegevens die als gevolg van de uitvoering van het programma worden geproduceerd, de output. In figuur 2.1 hebben we dit schematisch weergegeven.




**Figuur 2.1** *Gegevensstroom door een computer*

Om wat meer concreet te zijn kiezen we een voorbeeld uit de audio-elektronicawereld, een mp3-speler. In flashgeheugen staat een bestand, ofwel een hoeveelheid samenhangende gegevens, in mp3-formaat. Dit is een muziekbestand waarbij men de hoeveelheid informatie voor de opslag sterk gereduceerd heeft zonder daarbij veel aan kwaliteit te hoeven inleveren. De processor kan de gegevens van dit bestand bewerken aan de hand van een programma om er gegevens van te maken die geschikt zijn om naar speciale hardware voor geluidsweergave te sturen. Via een luidspreker is de muziek te horen. In figuur 2.2 is dit systeem schematisch weergegeven. Het flashgeheugen en enkele andere bouwstenen zijn als blokjes aangegeven. Met 'soundmaker' bedoelen we de hardware die van de digitale geluidssignalen analoge signalen maakt die door een luidspreker kunnen worden weergegeven.



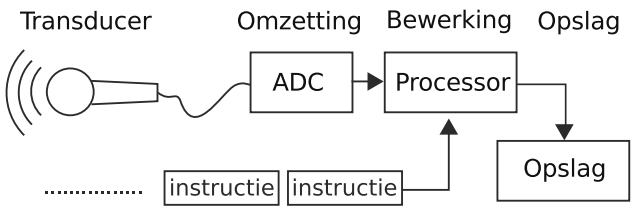

**Figuur 2.2** *Schematische opbouw van een mp3-speler*

## 2.4 Digitale gegevens

De gegevens waarover we hier spreken zijn digitaal. Het zijn dus feitelijk getallen en we zullen gaan zien dat deze getallen in het tweetallig stelsel worden weergegeven.
In de technologie van vandaag worden gegevens bij voorkeur in digitale vorm bewerkt. Dit heeft een aantal voordelen:

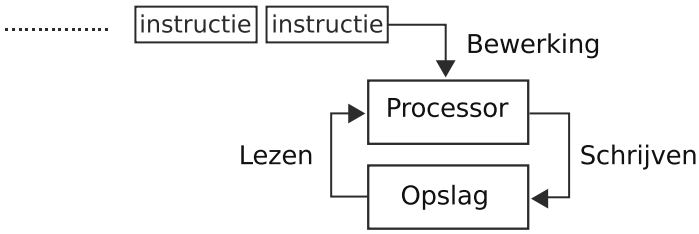
- Eenvoudig op te slaan. Digitale gegevens kunnen op allerlei manieren zonder verlies opgeslagen worden voor archivering of latere bewerking.
- Complexe bewerkingen zijn mogelijk. Met een combinatie van veel eenvoudige instructies kunnen ingewikkelde bewerkingen op de gegevens worden uitgevoerd.
- Geen effecten van temperatuurdrift of componentveroudering, zoals die in analoge technologie een rol speelt. De bewerkingen op digitale gegevens zijn op voorhand te analyseren en geven altijd dezelfde uitkomst.
- Verliesvrij transport over grote afstanden.

Veel gegevens zijn van nature al getallen, zoals financiële transacties, bepaalde meetgegevens enzovoort. Andere gegevens, zoals lettertekens en afbeeldingen, worden omgezet naar digitale vormen. We spreken dan van coderen. Analoge gegevens kunnen met een speciale component gedigitaliseerd worden. De component die dit doet, noemen we een analoog-digitaalconverter, ook wel ADC genoemd. Hierbij gaan we ervan uit dat de gegevens al als een elektrisch signaal beschikbaar zijn. Mocht dat niet het geval zijn, dan hebben we eerst een transducer nodig om de informatie om te zetten. Een eenvoudig voorbeeld hiervan is de microfoon, die geluid (drukgolven in de lucht) omzet in een overeenkomstig analoog elektrisch signaal. Op basis van een transducent @transducer@ in de vorm van een microfoon en een ADC kunnen we een geluidsregistratiesysteem maken. Figuur 2.3 laat zien hoe zo'n systeem is opgebouwd.

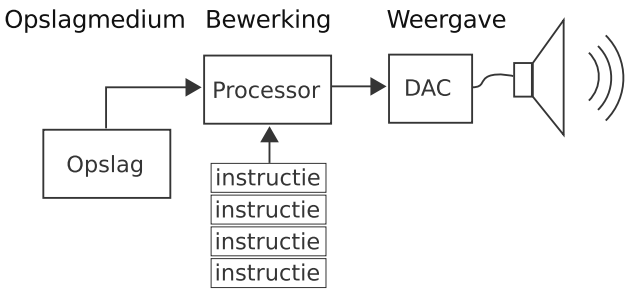



**Figuur 2.3** *Dataopslag van geluidsgegevens*

Hebben we de gegevens eenmaal opgeslagen, dan kunnen we bewerkingen op die gegevens gaan uitvoeren (zie figuur 2.4). Bij audiodata kan dat een filterbewerking zijn om ruis te onderdrukken. Ook is het mogelijk om de hoeveelheid data in te perken als we bijvoorbeeld een audiobestand willen omzetten naar een mp3-bestand. De component die van digitale data een daarmee evenredig analoog signaal maakt, noemen we digitaal-analoogconverter ofwel DAC. Deze component zijn we al in figuur 2.2 tegengekomen, maar toen werd hij 'soundmaker' genoemd. Een technisch beter plaatje van de mp3-speler is dus figuur 2.5.



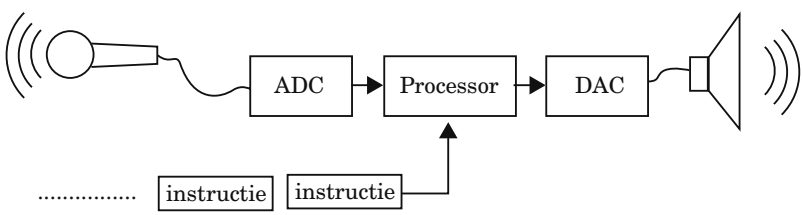

**Figuur 2.4** *Bewerking van data*



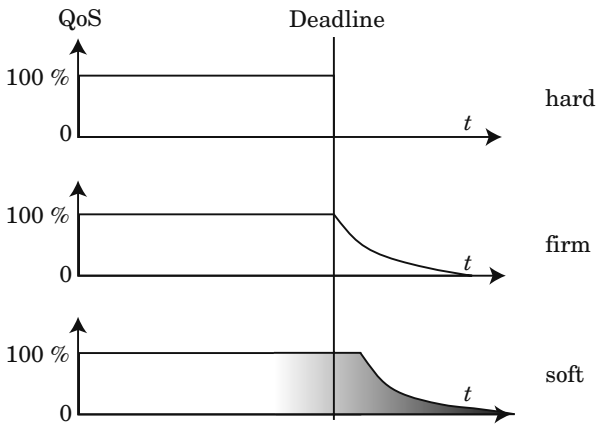

**Figuur 2.5** *Weergave van digitaal opgeslagen geluid*

## 2.5 Realtime

De drie stappen opname, bewerking en weergave kunnen we ook direct achter elkaar uitvoeren (zie figuur 2.6). Dit brengt ons bij een nieuw begrip, namelijk tijdsgarantie, of in computerjargon: realtime. De processor moet in deze situatie in staat zijn de binnenkomende stroom van geluidsgegevens bij te houden. Als dat niet lukt, vallen er gegevens weg en gaat de weergave haperen. Hoewel dit niet meteen rampzalige gevolgen heeft, zal de luisteraar niet tevreden zijn met het resultaat.




**Figuur 2.6** *Opname, bewerking en weergave van geluid*




**Figuur 2.7** *Drie vormen van realtime*

Wanneer een systeem tijdsgaranties kan geven, noemen we het systeem realtime. Dit houdt in dat een gegeven deadline in de tijd gehaald moet worden. In praktijk maakt men nog onderscheid tussen drie vormen van realtime:

1 *Hard realtime*. Bij hard realtime moet de tijdsgarantie (deadline) absoluut gehaald worden. Als dat niet lukt, kan dit fatale gevolgen hebben. Denk hierbij aan een systeem dat een kerncentrale moet besturen en binnen zekere tijd kleppen open of dicht moet zetten, of aan een pacemaker die op het juiste moment een puls moet afgeven.
2 *Firm realtime*. Bij firm realtime moet de tijdsgarantie ook gehaald worden. Als dat niet lukt, functioneert het systeem onder de maat. Men zegt dat de *Quality of Service* (QoS) afneemt. De eerder besproken mp3-speler is hier een goed voorbeeld van.
3 *Soft realtime*. Bij soft realtime mag de tijdsgarantie nu en dan overschreden worden zonder dat dit als fataal of slecht functioneren wordt aangemerkt. Gemiddeld moet de tijdsgarantie wel gehaald worden. Als dat niet lukt, neemt ook hier de QoS af. Meestal houdt dit in dat er toch een maximum aan de tijd zit waarbinnen het systeem zijn werk klaar moet hebben.

In figuur 2.7 zijn de drie vormen van realtime grafisch weergegeven. De tijdsgarantie is een punt op de *t*-as. Langs de *y*-as staat QoS. Merk op dat in de figuur bij soft realtime de QoS weliswaar niet aangetast wordt bij overschrijding van de deadline, maar dit moet wel gecompenseerd worden door bijvoorbeeld ook een aantal keren vóór de deadline iets af te handelen, waardoor gemiddeld de deadline gehaald wordt. In de figuur is dit aangegeven met een grijs gebied.

## 2.6 Processors

Uit de beschouwing die we tot nu toe gevolgd hebben, volgt dat de processor een belangrijke rol speelt in het functioneren van een computersysteem. In pc's treffen we

meestal een processor van Intel of AMD aan. Dit is een veelzijdig inzetbare processor. Kijken we in het algemeen naar processoreigenschappen, dan vinden we de volgende aandachtspunten:

- *Generieke en specifieke processors.* Processorfabrikanten hebben de keuze tussen het maken van een universeel inzetbare processor of een processor die is toegesneden op het oplossen van een specifiek probleem of een klasse van problemen. In het eerste geval spreken we van een generieke processor (zoals de processor in een pc). Voorbeelden van specifieke processors zijn processors voor driedimensionale beeldweergave en processors voor bewerkingen op audiosignalen. Generieke processors zijn flexibel inzetbaar, specifieke processors zijn optimaal voor specifieke toepassingen.
- *Snelheid.* De snelheid van een processor hangt af van allerlei factoren. Een aanduiding die nogal eens gebruikt wordt, is de *MIPS* (millions of instructions per second). Het is nogal riskant om processors van verschillend pluimage op basis van MIPS-cijfers met elkaar te vergelijken. Ook de klokfrequentie, zeg maar de snelheid waarmee het processorhartje klopt, geeft niet direct een maat voor de prestatie. Wel is het zo dat bij identieke processors, degene met de hoogste klokfrequentie ook het snelst is.
- *Energieverbruik en daarmee samenhangend de warmteontwikkeling.* Batterijgevoede apparatuur moet zuinig met het energieverbruik omspringen. We zeggen dat de vermogensdissipatie laag moet zijn. Ook apparatuur die uit het lichtnet gevoed wordt, kan baat hebben bij een zuinige processor. De warmteontwikkeling is dan namelijk laag, wat weer leidt tot geringe problemen met betrekking tot de koeling. Een pc met loeiende ventilatoren is geen pretje voor de gebruiker. Hier speelt wel een specifiek probleem. Een processor werkt op basis van een signaal dat snel in de tijd wisselt. Dit signaal wordt de klok genoemd. Het aantal keren per seconde waarmee de klok wisselt wordt de klokfrequentie genoemd. Om een processor en daarmee samenhangend een computersysteem beter te laten presteren wordt de klokfrequentie van de processor hoog gemaakt. Een hogere klokfrequentie geeft binnen de processor een grotere warmteontwikkeling. Dus hoe sneller, hoe meer energieverbruik er is door dezelfde processor.
- *Prijs.* Natuurlijk speelt de prijs een rol bij de keuze van een processor, vooral als we een processor voor een relatief goedkoop embedded systeem zoals een stukje speelgoed moeten selecteren. In zo'n geval legt de prijs van de processor relatief meer gewicht in de schaal dan in geval van een laptop.

In grote computersystemen zoals supercomputers is het niet ongebruikelijk om een groot aantal processors samen te laten werken. Dergelijke multiprocessorsystemen danken hun prestatie aan het evenwichtig verdelen van de uit te voeren taken over de beschikbare processors. Dit evenwichtig verdelen wordt 'load balancing' genoemd. Een systeem met meer dan één processor heet een *multiprocessor.* De ontwikkelingen op het gebied van multiprocessorsystemen zijn terechtgekomen bij de pc, tablet en smartphone. Deze systemen hebben meestal ook een multiprocessor aan boord, waarbij de processoren (cores) op een chip zijn samengebracht. Dit wordt aangeduid met de term multicore processor.

## 2.7 Netwerken

Bij netwerken denkt vrijwel iedereen tegenwoordig aan internet. Internet is een wereldwijd vertakt netwerk, waarop computersystemen aangesloten kunnen worden. Voordelen van een internetaansluiting zijn dat een gebruiker voor het uitvoeren van bepaalde werkzaamheden niet het huis uit hoeft, als hij tenminste van thuis uit toegang tot internet heeft. Ook zijn op eenvoudige wijze softwareverbeteringen (updates) en nieuwe programmatuur in het systeem aan te brengen. Het nadeel is dat op dezelfde eenvoudige wijze ook destructieve programma's zoals virussen het systeem binnen kunnen komen. Een computersysteem op een netwerk heeft als het ware een achterdeur gekregen die niet altijd even goed op slot zit.
Het aantal toepassingen in de communicatieve sfeer neemt enorm toe. Hierbij valt te denken aan e-mail, chatten, beeldtelefonie, social media en uitwisselen van bestanden van allerlei aard. Ook komen diensten die door andere systemen worden aangeboden binnen handbereik. Een ander voordeel van internet is dat informatie eenvoudig vanaf elke plek toegankelijk is.
De werking van netwerken is gebaseerd op technologie die gegevens via een transportmedium van het ene systeem naar het andere kan sturen. Transportmedia zijn kabels, radiogolven of licht. Om het transport in goede banen te leiden, worden aan beide kanten (de zender en de ontvanger) bepaalde regels en voorschriften gevolgd. Deze regels en voorschriften noemen we een *communicatieprotocol*.
Netwerken kunnen ook draadloos zijn. Het gsm-netwerk is daar een voorbeeld van. Dergelijke netwerken kunnen ook computersystemen en embedded systemen laten communiceren. In principe bieden de opvolgers van gsm grotere datasnelheden. Dat hier veel in is geïnvesteerd, geeft aan dat een veelheid van toepassingen ligt te wachten. Als we ons tot spraak zouden blijven beperken, zou gsm in de standaardvorm goed genoeg zijn.

## 2.8 Alles komt bij elkaar

Kijken we nog eens terug naar de stille revolutie van de embedded systemen, dan zien we dat dit mogelijk is geworden door de voortschrijdende integratie van halfgeleidercomponenten op een siliciumchip. Eenmaal ontworpen is de productie van krachtige processors met allerlei toegevoegde mogelijkheden niet duur meer en kunnen complexe computersystemen op een chip ingezet worden als componenten in een groter technisch systeem. De draadloze telefonie is hier een goed voorbeeld van. In de telefoon zitten chips die redelijk ingewikkeld zijn; door de geringe omvang en het lage energiegebruik zijn de uiteindelijke apparaten licht, betrouwbaar en gemakkelijk te hanteren.
Kijken we verder terug in het verleden, dan zien we dat de elektromotor een vergelijkbare ontwikkeling heeft doorgemaakt. In het begin waren motoren duur en werden als een op zichzelf staand systeem gezien. De diversiteit van motoren is daarna enorm toegenomen en kleine motortjes zijn een onderdeel van een groter technisch geheel geworden. Kijken we bijvoorbeeld naar een klassieke desktop-pc – toch een tamelijk statisch ding –, dan blijkt dat daar toch al gauw een stuk of acht elektromotoren in zitten.

Voor diegenen die zich afvragen welke motoren dat zijn:
- ventilator in de voeding;
- ventilator op het koelblok van de CPU;
- spindlemotor (aandrijfmotor) van de floppydrive (mocht die er nog in zitten);
- spindlemotor van de harde schijf;
- besturingsmotor van de lees-/schrijfkoppen;
- spindlemotor in dvd/cd-romspeler;
- besturing van de laserpositie;
- besturingsmotor van de dvd/cd-lade.

Op dezelfde wijze zijn computers componenten geworden in allerlei technische systemen. Een moderne tv bevat computers, een afwasmachine bevat computers, een kamerthermostaat bevat een computer en zo kunnen we een hele waslijst aan producten opvoeren die allemaal één of meer computersystemen ingesloten hebben.
De hiervoor genoemde voorbeelden bestonden al voor de opkomst van de computer als component. In de wereld van embedded systemen kunnen drie typen van realisaties worden aangemerkt:
1 Bestaande technologie krijgt één of meer embedded computers. Voorbeelden hiervan zijn huishoudelijke apparatuur zoals tv's en witgoed.
2 Bestaande toepassingen uit de computerwereld worden als embedded systeem uitgevoerd. Hier vinden we gameconsoles en van computers afgeleide producten, zoals smartphones.
3 Op basis van embedded technologie worden geheel nieuwe toepassingen ontwikkeld die gebaseerd zijn op de nieuwe mogelijkheden. Een meetinstrument dat in een vertrek de afstand tussen muren bepaalt op basis van een reflectiemeting en een daaraan gekoppelde berekening is hier een voorbeeld van.

Embedded systemen waren in het verleden veelal op zichzelf staande systemen. Daar komen nu toepassingen bij die via een netwerk (al dan niet draadloos) kunnen communiceren. Deze ontwikkeling waarbij allerlei apparaten met het internet verbonden zijn, wordt het Internet of Things (IoT) genoemd. De drie technologieën die de basis vormen voor de huidige en toekomstige ontwikkelingen op het gebied van informatieverwerking en -voorziening zijn de volgende:
- *Internet*: het beschikbaar komen van informatie op een wereldwijd netwerk.
- *Systeemintegratie*: het beschikbaar komen van krachtige systemen op basis van embedded computersystemen.
- *Draadloze digitale communicatie*: het beschikbaar zijn van digitale informatie op een willekeurige plek.

Het samenkomen van deze drie technologieën vormt de basis voor de toekomstige informatiemaatschappij. Dit samenkomen vindt plaats in moderne embedded systemen. Kenmerken van deze moderne embedded systemen zijn:
- autonoom opererend zonder menselijke tussenkomst;
- zelfconfiguratie; adaptief (past zich aan de omgeving en toepassing aan);
- kan de omgeving verkennen en ontdekken (discovery);
- kan communiceren met andere devices;

- kan samenwerken met andere devices;
- vertoont intelligent gedrag;
- heeft een eenvoudige user interface (past zich aan de user aan);
- is energiezuinig.

Deze kenmerken hoeven natuurlijk niet allemaal aanwezig te zijn. Het hangt van de toepassing af of een kenmerk nodig of wenselijk is.

## 2.9 Software

We besteden aandacht aan programmeertalen en software in embedded systemen.

### 2.9.1 Programmeertalen

Computersystemen van welke soort dan ook moeten geprogrammeerd worden. Zoals al is opgemerkt voert een processor instructies uit. Deze verzameling van instructies moet ergens toe leiden en de soorten instructies en de volgorde waarin ze worden uitgevoerd bepalen hoe dat gaat. Een programmeur heeft dit ooit bedacht en ingevoerd. Het resultaat is dus de software (de verzameling instructies) waarmee een systeem werkt.
Nu blijkt in de praktijk dat de meeste processors een verzameling van instructies kennen die heel elementaire stappen uitvoeren. Om op basis van deze elementaire stappen een complex programma samen te stellen, is een lastige klus. Daarom worden computerprogramma's vrijwel altijd geschreven in een zogenoemde 'hoge programmeertaal' (high level language, HLL). Zo'n hoge programmeertaal is een eenduidige beschrijving van de acties die door het systeem moeten worden uitgevoerd, maar nu niet op basis van elementaire instructies voor de processor, maar in een voor mensen wat meer hanteerbare schrijfwijze. In veel hoge programmeertalen wordt de term statement gebruikt voor een actie die een computer moet uitvoeren. Een voorbeeld van een statement in C, C++ of Java is:

```
bedrag = prijs + btw
```

Ook zonder kennis van programmeren is de bedoeling hier duidelijk. Probleem is dat de processor dit weer niet snapt. We moeten dus een mechanisme hebben om onze opdrachten, geschreven in een hoge programmeertaal, om te zetten naar instructies voor de processor. Gelukkig kunnen we dit weer met behulp van software doen, zodat we deze klus niet handmatig hoeven te klaren. Er zijn twee manieren om deze omzetting, ook wel vertaling genoemd, uit te voeren:

1. Het programma in de hoge programmeertaal kan eerst in zijn geheel naar processorinstructies worden omgezet. We noemen dit proces compileren en de software die dat doet heet een *compiler*. De resulterende instructies kunnen worden bewaard en vervolgens zo vaak als we maar willen, worden uitgevoerd. Een voorbeeld van een taal die gecompileerd wordt is C++.
2. We kunnen de opdrachten, uitgedrukt in de hoge programmeertaal, regel voor regel overzetten naar machine-instructies. Telkens als een regel is omgezet, zal deze ook

meteen worden uitgevoerd. We werken dus telkens met kleine stukjes vertaling die ook meteen worden uitgevoerd. Deze aanpak heet interpreteren en het programma dat dit uitvoert heet een *interpreter*. De taal Python is een voorbeeld van een geïnterpreteerde taal.

Beide methoden hebben hun voor- en nadelen. Zo gaat een programma dat geïnterpreteerd wordt, onmiddellijk werken en is het vaak wat eenvoudiger te volgen bij eventuele problemen. Een interpreter levert meestal een trager resultaat omdat telkens de vertaalslagen erbij komen. Dit zegt dus meteen wat het voordeel van compileren is. Hierbij hoeft er maar één keer vertaald te worden en vervolgens kunnen we het vertaalde programma, dat nu geheel uit processorinstructies bestaat, uitvoeren.
Er bestaan heel veel hoge programmeertalen. De twee belangrijkste stromingen zijn proceduregerichte talen en objectgeoriënteerde talen (OO). Deze twee stromingen verschillen qua benadering van de probleemoplossing. Bij proceduregebaseerde talen worden taken die moeten worden uitgevoerd geformuleerd in procedures. De talen werken op structuren van gegevens, de zogenoemde datastructuren. Voorbeelden van deze talen zijn Pascal en C.
Bij OO-talen wordt het probleem uitgevoerd door een samenspel van objecten. In een object vinden we gegevens en de daarop betrekking hebbende handelingen gecombineerd terug. Voorbeelden van OO-talen zijn Smalltalk, Java en C#.
Er bestaan ook talen die beide vormen bieden, de hybride talen. Het meest bekende voorbeeld hiervan is C++.
Aan het schrijven van een programma in een hogere taal gaat voor de softwareontwerper nog een traject vooraf. De aanpak en structuur zullen eerst duidelijk moeten zijn. Een populaire taal voor het ontwerpen van OO-oplossingen is UML (unified modeling language). UML bevat weer meer onderdelen met elk een specifiek doel, zoals een hulpmiddel om te beschrijven hoe de software gebruikt gaat worden en een visuele weergave om aan te geven welke objecten er gebruikt gaan worden en wat hun interactie is.

### 2.9.2 Software in embedded systemen

Voor embedded systemen wordt in principe dezelfde wijze van programmeren gevolgd als voor standaardcomputersystemen. We komen hier dus zowel proceduregebaseerde talen als OO-talen tegen. Wel moeten we hier wat kanttekeningen maken.

- Een embedded systeem is meestal niet het systeem waar de software op ontwikkeld wordt. Hiermee bedoelen we het volgende. Als we de software op een ander systeem ontwikkelen, kan het zijn dat daar een processor gebruikt wordt die een heel andere verzameling elementaire instructies kent dan de processor in het embedded systeem dat uiteindelijk het programma moet gaan uitvoeren. Wat we dan voor het compileren nodig hebben is een *crosscompiler*. Dit is een compiler die werkt op een systeem met processor A, maar instructies genereert voor processor B. Het vertaalde programma kunnen we dus niet uit laten voeren door processor A.
- Een daarmee samenhangend aspect is het volgende: hoe krijgen we ons programma in het embedded systeem en hoe gaan we een en ander testen? Ook hier gaat dat minder eenvoudig dan bij een standaardcomputersysteem. Een mogelijkheid is om het programma via een netwerkaansluiting in het embedded systeem te zetten en

vervolgens uit te voeren. Ook voor het testen zijn speciale oplossingen gevonden. In hoofdstuk 19 komen we hier op terug.
- Een belangrijk aspect vormt de grootte van de opslag die voor het programma nodig is. Ontwerpers van embedded systemen spreken van de footprint (vrij vertaald: schoenmaat) van een embedded applicatie. Embedded systemen worden of op maat ontworpen of worden gebouwd op een relatief beperkt systeem. Hoe meer er nodig is aan bijvoorbeeld opslag voor instructies, des te duurder wordt het systeem. Bij standaardsystemen is er meestal een ruime hoeveelheid opslag voorhanden, zodat de softwareontwerper hier nauwelijks rekening mee hoeft te houden.
- Wat niet onvermeld mag blijven is het feit dat embedded systemen vaker met realtime aspecten te maken hebben dan de gemiddelde applicatie op een standaardcomputer. Zeker als het een kantoorapplicatie betreft.
- Vóór het maken van de software is een keuze gemaakt tussen wat er precies in hardware en wat er in software uitgevoerd gaat worden. Deze keuze is een essentieel onderdeel van het hele ontwikkeltraject van een embedded systeem.

## 2.10 Samenvatting

Alle computers bevatten een processor die instructies uitvoert. Deze instructies werken op gegevens of data. Dit zijn digitale gegevens die eruit zien als nullen en enen. We onderscheiden standaardcomputers en embedded systemen.
Als de bewerking een tijdslimiet kent, spreken we van realtime. We onderscheiden hard-, firm- en softrealtime. Bij processors maken we onderscheid tussen specifieke en generieke processors. Netwerken verbinden computersystemen en stellen gegevens van andere systemen beschikbaar. Voor het functioneren van netwerken zijn communicatieprotocollen opgesteld. Computers hebben een programma nodig om te kunnen werken. Deze programma's worden opgesteld volgens de richtlijnen van programmeertalen. Om deze talen op een computer te kunnen uitvoeren moet de betekenis van opdrachten in de hoge programmeertaal omgezet worden naar instructies die de processor kan uitvoeren. Zo kan een taal geïnterpreteerd worden, waarbij na elke vertaalde regel deze meteen wordt uitgevoerd of gecompileerd, waarbij het hele programma eerst wordt omgezet naar instructies en het resultaat van de vertaling (de executable) kan worden uitgevoerd.

## Vragen

**2.1** Beschrijf globaal de werking van een computersysteem.
**2.2** Waarin verschilt een embedded systeem van een standaardcomputer?
**2.3** Waarin komt een embedded systeem overeen met een standaardcomputer?
**2.4** Waarom worden gegevens zo veel mogelijk gedigitaliseerd?
**2.5** Welke vormen van realtime zijn er?
**2.6** Wat is het verschil tussen interpreteren en compileren?
**2.7** Wat is een crosscompiler?

Hoofdstuk 3

# Getalrepresentaties en codesystemen

In de digitale techniek en bij digitale computers in het bijzonder maken we gebruik van het tweetallige stelsel, een getalstelsel dat alleen van nullen en enen gebruik maakt. We zullen in dit hoofdstuk kennismaken met dit tweetallig stelsel en de wijze waarop computers met dit stelsel rekenen. Ook zullen we zien dat computers gebruikmakend van deze nullen en enen, met leestekens van teksten kunnen omgaan. Als laatste onderwerp behandelen we foutendetectie en correctietechnieken.

## 3.1 Talstelsels

Getallen kunnen worden uitgedrukt in diverse stelsels. Omdat de meerderheid van de mensen tien vingers heeft, wordt het tientallige of *decimale* stelsel in het dagelijks leven het meest gebruikt. In dit stelsel worden getallen geschreven als veelvouden van de machten van het grondtal 10. Een getal als 805 betekent dan het volgende:

$$8 \times 10^2 + 0 \times 10^1 + 5 \times 10^0$$

Kortom, de plaats van het cijfer bepaalt de waarde en deze waarde wordt uitgedrukt in een macht van 10. Als een cijfer $x$ op de $n$-de plaats van rechts staat, is zijn waarde $x \times 10^n$. Let wel, we tellen voor $n$ vanaf 0. De getallen die we in de decimale representatie gebruiken, maken gebruik van de cijfers 0 tot en met 9. In het algemeen worden in een stelsel met grondtal $G$ de cijfers 0 tot en met $(G - 1)$ gebruikt.
Het systeem van het tientallige stelsel is eenvoudig uit te breiden voor stelsels met andere grondtallen.

We gaan hierna in op het binaire stelsel (paragraaf 3.1.1), bits en bytes (paragraaf 3.1.2) en het octale en hexadecimale stelsel (paragraaf 3.1.3).

### 3.1.1 Binair stelsel

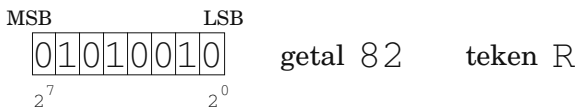
Gebruiken we grondtal 2, dan heet dit het *binaire* of *tweetallige* stelsel. In dit stelsel gebruiken we de cijfers 0 en 1, en de plaatswaarde of het gewicht wordt uitgedrukt in machten van 2. Willen we de waarde van een binair getal in het decimale stelsel weten, dan vermenigvuldigen we alle cijfers in een getal met hun plaatswaarde, en tellen alles op. Bijvoorbeeld 10 011 110 is in het decimale stelsel:

$$\begin{aligned} 10\,011\,110 &= 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\ &= 128 + 0 + 0 + 16 + 8 + 4 + 2 + 0 = 158 \end{aligned}$$

Zoals al opgemerkt gebruiken we in het binaire stelsel, met G=2, alleen de cijfers 0 en 1.

### 3.1.2 Bits en bytes

Computersystemen gebruiken voor alles wat ze doen het binaire stelsel. Dit betekent dat niet alleen getallen, maar ook andere gegevens zoals teksten, computeropdrachten (instructies) en dergelijke, allemaal met nullen en enen weergegeven worden. Een enkele 0 of 1 noemen we een *bit*. Een groep bits noemen we heel algemeen een (binaire) code. Deze code kan dus best een getal voorstellen, maar dat hoeft niet. Een groepje van acht bits heet een *byte*. Een halve byte (dus een groepje van vier bits) noemt men ook wel *nibble*. Een byte is op te vatten als acht vakjes, waarbij één vakje één bit (dus een 0 of 1) kan bevatten. In figuur 3.1 is dat weergegeven. Elk vakje heeft weer zijn eigen gewicht, uitgedrukt als een macht van 2. Het meest rechtse vakje heeft het gewicht $2^0$ en wordt de *least significant bit* of LSB genoemd. Het meest linkse vakje heeft, voor een byte, het gewicht $2^7$ en heet *most significant bit* of MSB.




**Figuur 3.1** *Een byte*

Als we hier te maken hebben met een 8-bits getal, dan blijkt deze byte gelijk te zijn aan 82. Het zou ook een letter uit de verderop te introduceren ASCII-code kunnen zijn en dan stelt deze byte de letter R voor. Net zoals we bij allerlei grootheden de voorvoegsels 'kilo-', 'mega-' en 'giga-' kennen, zijn er ook voorvoegsels om veelvouden van bits en/of bytes aan te duiden. Ze hebben weliswaar ook de naam kilo-, mega- en giga-, maar hier schuilt een addertje onder het gras. Vaak worden deze voorvoegsels gebruikt om veelvouden van machten van twee die in de buurt zitten van tienmachten aan te geven, maar het is niet precies het duizend- of miljoenvoud. Een kilobyte is $2^{10}$ bytes. Dit blijkt 1024 bytes te zijn en is dus net iets meer dan de gebruikelijke kilo. Een megabyte is $2^{20}$ bytes en een gigabyte $2^{30}$ bytes. Tabel 3.1 laat gebruikelijke en minder gebruikelijke voorvoegsels zien.

**Tabel 3.1** *Voorvoegsels*

| Voorvoegsel | Afkorting | Tweemacht | Waarde |
|---|---|---|---|
| Kilo | k of K | $2^{10}$ | 1024 |
| Mega | M | $2^{20}$ | 1048576 |
| Giga | G | $2^{30}$ | 1073741824 |
| Tera | T | $2^{40}$ | 1099511627776 |
| Peta | P | $2^{50}$ | 1125899906842624 |
| Exa | E | $2^{60}$ | 1152921504606846976 |
| Zetta | Z | $2^{70}$ | 1180591620717411303424 |
| Yotta | Y | $2^{80}$ | 1208925819614629174706176 |

De voorvoegsels uit tabel 3.1 worden weliswaar veel gebruikt door computerwetenschappers en in de informatica (en in dit boek), maar ze geven wel aanleiding tot misverstanden. Zo kan een fabrikant van harde schijven de term Gigabyte gebruiken zonder dat het duidelijk is of het om $2^{30}$ (1073741824) of $10^9$ (1000000000) byte gaat. Een verschil van ruim 7%. De standaardisatiecommissie van het IEC heeft daarom alternatieve voorvoegsels voorgesteld, die duidelijk bedoeld zijn voor machten van twee. Tabel 3.2 laat deze nieuwe voorvoegsels zien.

**Tabel 3.2** *IEC-voorvoegsels*

| Voorvoegsel | Afkorting | Tweemacht |
|---|---|---|
| Kibi | Ki | $2^{10}$ |
| Mebi | Mi | $2^{20}$ |
| Gibi | Gi | $2^{30}$ |
| Tebi | Ti | $2^{40}$ |
| Pebi | Pi | $2^{50}$ |
| Exbi | Ei | $2^{60}$ |
| Zebi | Zi | $2^{70}$ |
| Yobi | Yi | $2^{80}$ |

### 3.1.3 Octaal en hexadecimaal stelsel

Gebruiken we het stelsel met grondtal 8, dan heet dat het *octale stelsel*; kiezen we grondtal 16, dan is dat het *hexadecimale stelsel*. In het octale stelsel gebruiken we de cijfers 0 tot en met 7, in het hexadecimale stelsel de cijfers 0 tot en met 9 aangevuld met A, B, C, D, E en F om tot 16 te gebruiken cijfers te komen, dus A = 10, B = 11 enzovoort. Werken met deze stelsels gaat analoog aan het binaire en decimale stelsel, maar het octale en hexadecimale stelsel hebben een zekere verwantschap met het binaire stelsel, omdat de grondtallen van deze stelsels machten van 2 zijn. We gebruiken deze twee stelsels daarom nogal eens om binaire getallen, die de neiging hebben nogal lang en onoverzichtelijk te worden, wat compacter te schrijven, zodat ze wat overzichtelijker worden. Omzetten van getallen in het binaire stelsel naar het octale stelsel gaat door het binaire getal in stukjes van drie bits te splitsen en hiervoor de octale representatie in te vullen.

**Voorbeeld 3.1**
Het binaire getal 1100101110 splitsen, levert 1 100 101 110 en omzetten levert 1456 octaal. Waarom kan dit zo eenvoudig? In een binair getal van drie bits kunnen de getallen van 0 tot en met 7 gerepresenteerd worden. Dit zijn precies de cijfers die in het octale stelsel gebruikt mogen worden.

Voor omzetten van binair naar hexadecimaal geldt een vergelijkbaar verhaal. We splitsen nu het binaire getal in 4-bits stukjes en schrijven per brokje de hexadecimale representatie op.

**Voorbeeld 3.2**
Het binaire getal (als in voorbeeld 3.1) 1100101110 splitsen in vier bits (werkend van achteren naar voren) levert 11.0010.1110 en omzetten levert 32E hexadecimaal.

**Tabel 3.3** *Vertaaltabel met diverse codes*

| Decimaal | Binair | Octaal | Hexadecimaal |
|---|---|---|---|
| 0 | 0000 | 0 | 0 |
| 1 | 0001 | 1 | 1 |
| 2 | 0010 | 2 | 2 |
| 3 | 0011 | 3 | 3 |
| 4 | 0100 | 4 | 4 |
| 5 | 0101 | 5 | 5 |
| 6 | 0110 | 6 | 6 |
| 7 | 0111 | 7 | 7 |
| 8 | 1000 | 10 | 8 |
| 9 | 1001 | 11 | 9 |
| 10 | 1010 | 12 | A |
| 11 | 1011 | 13 | B |
| 12 | 1100 | 14 | C |
| 13 | 1101 | 15 | D |
| 14 | 1110 | 16 | E |
| 15 | 1111 | 17 | F |

Om de diverse representaties uit elkaar te houden, gebruiken we vaak tekens vóór het getal. Staat er niets vóór een getal, dan is het decimaal. % voor een getal betekent binair, bijvoorbeeld %1100101110. 0 voor een getal betekent octaal, bijvoorbeeld 01456, en 0x voor een getal betekent hexadecimaal, bijvoorbeeld 0x32E. Een alternatieve schrijfwijze is dat we eerst het grondtal noteren, gevolgd door een R (van radix) en daarachter het getal zelf. Een binair getal wordt dan geschreven als 2R10100100, een hexadecimaal getal als 16RFA2C. Nogmaals wijzen we erop dat we wel van getallen spreken, maar voor een computersysteem hoeven deze getallen niet alleen zuivere getallen voor te stellen. Bekijken we bijvoorbeeld een computer die in zijn geheugen op het oog uitsluitend binaire getallen bevat, dan kunnen we niets zeggen over de betekenis van die getallen. Het kunnen instructiecodes zijn, variabelen of delen van variabelen, onderdelen van datastructuren, tekens van teksten enzovoort.

## 3.2 Binair rekenen

Het binaire systeem maakt rekenen heel eenvoudig. Er zijn maar een paar rekenregels. Dit maakt dit stelsel ook zeer geschikt voor automaten. Het nadeel dat getallen vaak

heel veel cijfers hebben (in het binaire systeem alleen de cijfers 0 en 1) is voor automaten juist geen probleem, omdat dat in hardwaretermen meestal 'meer van hetzelfde' betekent.

We bespreken hierna eerst conversie (paragraaf 3.2.1). Daarna gaan we in op de rekenregels voor optellen (paragraaf 3.2.2) en vermenigvuldigen en delen (paragraaf 3.2.3). In paragraaf 3.2.4 behandelen we het complement bij negatieve getallen (paragraaf 3.2.4) en in paragraaf 3.2.5 gaan we rekenen met negatieve binaire getallen. Bij al deze regels gaat het, afgezien van het onderdeel conversie, om hele getallen. We sluiten af met rekenen met niet-gehele getallen (paragraaf 3.2.6).

### 3.2.1 Conversie

Om een binair getal als een herkenbaar getal in het ons vertrouwde tientallige stelsel te schrijven, kunnen we volstaan met een optelling van de tweemachten waarmee het getal opgebouwd is. Zo is:

$$\begin{aligned} 1001101 &= 1\times 2^6 + 0\times 2^5 + 0\times 2^4 + 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0 \\ &= 2^6 + 2^3 + 2^2 + 2^0 = 64 + 8 + 4 + 1 = 77 \end{aligned}$$

Het omzetten van een decimaal getal naar een binair getal is iets lastiger, maar ook wel te doen. We nemen het getal 543. Om dit binair te schrijven, zetten we dit getal om naar machten van 2. We zoeken de hoogste macht van 2 die er nog in past. Dit blijkt $2^9$ ofwel 512 te zijn. Verminderen we ons getal met 512, dan houden we 31 over. De tweemachten $2^8$, $2^7$, $2^6$ en $2^5$ passen hier niet in. De tweemacht $2^4$ ofwel 16 past wel. Dan houden we nog $31 - 16 = 15$ over. Hier past $2^3 = 8$ in, met rest 7. Als we $2^2$ hier af halen, geeft dat als rest 3. Dit blijkt nog eens $2^1$ en $2^0$ te bevatten. Samengevat kunnen we schrijven:

$$543 = 1\times 2^9 + 0\times 2^8 + 0\times 2^7 + 0\times 2^6 + 0\times 2^5 + 1\times 2^4 + 1\times 2^3 + 1\times 2^2 + 1\times 2^1 + 1\times 2^0$$

We kunnen het getal dan binair schrijven als: 1000011111.

Deze methode om decimaal naar binair om te zetten wordt de 'som der gewichtenmethode' genoemd. Een alternatieve methode om een decimaal getal naar binair te converteren draagt de naam 'herhaald delen door 2'. Dat gaat als volgt. Gegeven een decimaal getal, dan delen we dit door 2. Dit levert een resultaat en een rest die bij deling door twee ofwel 1 ofwel 0 is. Is de rest 1 dan noteren we een 1 anders een 0. We herhalen deze aanpak voor het resultaat en gaan door met dit herhalen tot het resultaat 0 is. De bits van de gezochte binaire representatie zijn de genoteerde restgetallen, waarbij de eerste rest het minst significante bit (LSB) is en de laatste het meest significante bit (MSB). Een voorbeeld zal dit wellicht verduidelijken. We kiezen het getal 117.

117 delen door 2 levert 58 met rest 1 (LSB)
58 delen door 2 levert 29 met rest 0
29 delen door 2 levert 14 met rest 1
14 delen door 2 levert 7 met rest 0

7 delen door 2 levert 3 met rest 1
3 delen door 2 levert 1 met rest 1
1 delen door 2 levert 0 met rest 1 (MSB)

De binaire weergave (de resten van MSB naar LSB) is dus: 1110101. Dat deze aanpak werkt, is het eenvoudigst in te zien aan de hand van het binaire getal zelf. Een binair geschreven oneven getal eindigt op een 1 (het LSB), een even getal eindigt op een 0. Delen door twee is eigenlijk alle bits een positie naar rechts schuiven, de rest is het meest rechtse bit dat door het schuiven buiten de boot valt. Bij een oneven getal is dat dus 1, bij een even getal een 0. Door dit toe te passen tot het getal 'op' is, hebben we de bits van het getal gevonden. In ons voorbeeld vinden we dus na de eerste deling een rest 1 en hebben als resultaat 111010 (dit is 58). Dit is nu een even getal, dus zal de volgende deling een 0 als rest geven en als resultaat hebben we 11101 (29). Zo lopen we dus feitelijk van achter naar voren alle bits van het getal langs. Voor gebroken getallen zoals 2,75 splitsen we het getal op in het deel vóór de komma (hier 2) en het deel áchter de komma (hier 0,75). Het deel voor de komma kan op de eerder beschreven wijze omgezet worden naar een binair getal, wat voor ons voorbeeld resulteert in 10 (binair). Het deel achter de komma kunnen we gaan schrijven als som van negatieve machten van 2. Zo is 2-1=1/2=0,5 en 2-2=1/4=0.25 etc. In ons voorbeeldje zien we dat 0,75 als hoogste negatieve macht 2-1 bevat en als we het eerder getoonde recept toepassen noteren we op de eerste plek achter de komma een 1. Vervolgens werken we verder en met het resultaat van 0,75-0,5=0,25. We krijgen uiteindelijk: 2,75 = 10,11 (binair).

Ook hier bestaat een alternatieve werkwijze die werkt zoals getoond bij het voorbeeld van herhaald door 2 delen. Voor getallen kleiner dan 1 wordt het nu geen delen, maar vermenigvuldigen. Door te vermenigvuldigen met 2 zal het te vinden binaire getal bit voor bit over de komma geschoven worden. Als het resultaat groter dan 1 wordt, noteren we een 1 en trekken 1 van het resultaat af en werken verder. Is het kleiner dan 1, dan noteren we een 0. Is het getal waarmee we verder werken 0, dan zijn we klaar.

0,75 vermenigvuldigen met 2 levert 1,5 noteer 1 en werk verder met 1,5-1=0,5
0,5 vermenigvuldigen met 2 levert 1 noteer 1 en we hebben 1-1= 0 over

Omdat we 0 overhouden zijn we dus klaar, maar ook hier kunnen repeterende breuken optreden zoals ook in het tientallig stelsel 1/3 gelijk is aan 0,333333….. In de praktijk stoppen we als er voldoende nauwkeurigheid bereikt is. Als tweede voorbeeld zullen we 1/3 binair schrijven:

1/3 vermenigvuldigen met 2 levert 2/3 (kleiner dan 1) noteer 0 en ga verder met 2/3
2/3 vermenigvuldigen met 2 levert 4/3 (groter dan 1) noteer 1 en ga verder met 4/3-1=1/3
1/3 vermenigvuldigen met 2 levert 2/3 (kleiner dan 1) noteer 0 en ga verder met 2/3
Etc.

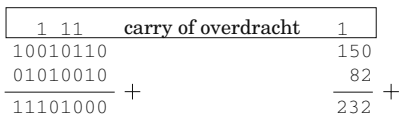
Omdat we geen 0 als eindresultaat krijgen om mee verder te gaan, zijn we nu nooit klaar. We kunnen besluiten met 16 cijfers achter de komma te volstaan. Dan krijgen we 1/3 = 0,0101010101010101.

### 3.2.2 Optellen

De rekenregels voor optellen zijn eenvoudig:

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 10$$

In het laatste geval hebben we een bit extra die overdracht- of carry-bit genoemd wordt. Hoe we dit voor een serieuze optelling van grotere getallen kunnen toepassen, is weergegeven in figuur 3.2. Rechts in de figuur is het decimale equivalent van deze optelling te zien. In hoofdstuk 4 zullen we zien dat een schakeling om deze optelling te realiseren met behulp van logische bouwstenen, redelijk eenvoudig is.



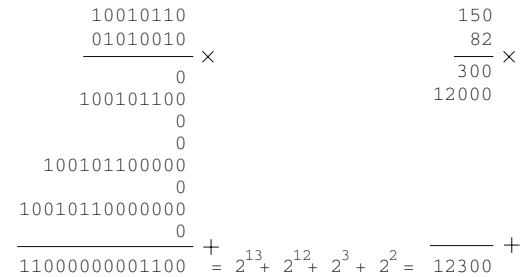

**Figuur 3.2** *Binaire optelling naast een decimale*

### 3.2.3 Vermenigvuldigen en delen

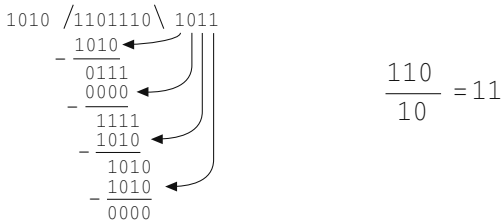
De regels voor vermenigvuldigen zijn ook eenvoudig. Een computer hoeft maar één tafel te leren:

$$0 \times 0 = 0$$
$$0 \times 1 = 0$$
$$1 \times 0 = 0$$
$$1 \times 1 = 1$$

Een vermenigvuldiging is een kwestie van schuiven en optellen. Dit is weergegeven in figuur 3.3. In deze figuur is naast de binaire vermenigvuldiging ook de decimale versie weergegeven. Een verschuiving van een bitreeks naar links met het invullen van een nul aan de rechterkant komt neer op een vermenigvuldiging met twee. Twee keer schuiven is een vermenigvuldiging met vier, drie keer schuiven is een vermenigvuldiging met acht enzovoort. Eigenlijk is de binaire vermenigvuldiging een optelling van naar links geschoven bitreeksen.
Een binaire deling is weergegeven in figuur 3.4. We hebben daar de staartdeling zoals bekend uit het tientallig stelsel toegepast. In dit voorbeeld is te zien dat we binaire getallen van elkaar kunnen aftrekken. In de volgende paragraaf zullen we zien hoe we negatieve getallen kunnen weergeven. Aftrekken van binaire getallen komt dan neer op het negatief maken van een getal en vervolgens de optelling uit te voeren. In de figuur is rechts de decimale versie gegeven.




**Figuur 3.3** *Een binaire vermenigvuldiging naast een decimale*




**Figuur 3.4** *Binaire deling naast een decimale*

Merk op dat delen door twee of machten van twee (vier, acht enzovoort) een eenvoudige operatie is. Delen door twee komt neer op het naar rechts schuiven van de bits van het te delen getal. Hierbij valt een bit weg en als dit een 1 is, zal het resultaat niet kloppen.

### 3.2.4 Negatieve getallen

We gaan in deze paragraaf in op het complement en op het complementeren in het binaire stelsel.

Complement

Bij negatieve getallen kunnen we een methode van schrijven toepassen die we het complement noemen. Een voorbeeld van complementeren is te geven aan de hand van een recorder voor bijvoorbeeld video- of cassettebanden (de inmiddels al oude technologie). Wanneer we een teller op zo'n recorder bekijken, dan kan met heen-en-weer spoelen de tellerstand veranderd worden. Stel, we zetten de teller met een resetknopje ergens halverwege de band op nul. Bij vooruitspoelen loopt de teller op en bij terugspoelen telt hij af. Komen we bij het aftellen voorbij nul, dan springt de teller in de maximale telstand en telt vervolgens weer af. Voor een teller met vier cijfertjes zeggen we nu dat 9999 overeenkomt met –1, 9998 met –2 enzovoort. Met deze truc verdelen we het totale tellerbereik in twee gebieden, een positief en een negatief gebied. We kunnen nu tellen van –5000 tot en met +4999. Hoe kunnen we nu een negatief getal vinden? Bijvoorbeeld: hoe ziet –45 eruit? We moeten dan zoeken naar het getal *N* waarvoor geldt:

$$N - 10000 = -45 \rightarrow N = 9955$$

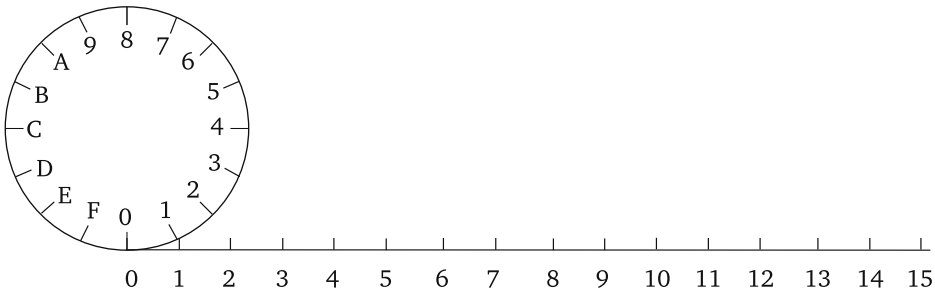
De optelling blijft 'gewoon' gelden, want 9955 + 45 = 0000. Bedenk dat we slechts vier cijfers op onze teller ter beschikking hebben. Dus 10000 kunnen we niet weergeven. We noemen 9955 het 10-complement van 45. Een alternatieve methode om het 10-complement te vinden is om eerst het 9-complement te zoeken. Dit is te vinden door het getal af te trekken van 9999 (in ons voorbeeld 9954), tellen we er 1 bij op, dan vinden we het 10-complement. Merk op dat we in het 9-complement per cijferpositie het verschil met 9 aangeven, dus 0 wordt 9, 1 wordt 8, 2 wordt 7, 5 wordt 4 enzovoort.

### Complementeren in het binaire stelsel

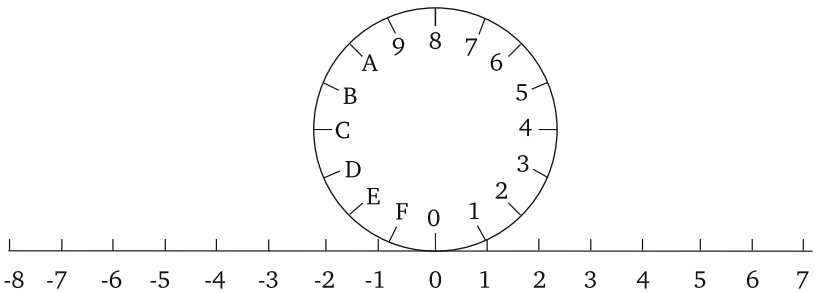
Op dezelfde wijze als met decimale getallen kunnen we ook bij binaire getallen de negatieve getallen schrijven als complementen. Voor 8-bits getallen komt dit neer op aftrekken van het getal '100 000 000' (256 decimaal). We doen dit gemakkelijker door eerst het 1-complement te bepalen (aftrekken van '11 111 111' = 255 decimaal) en er dan 1 bij op te tellen. Het blijkt nu dat het 1-complement is te krijgen door inverteren, dat wil zeggen een 1 wordt 0 en omgekeerd: een 0 wordt 1. Dit is met logische bouwstenen, die we in hoofdstuk 4 leren kennen, een eenvoudige operatie. Het 2-complement is gelijk aan het 1-complement + 1. Bij gebruik van de 2-complementnotatie kunnen we in acht bits (één byte) 127 positieve getallen, een 0 en 128 negatieve getallen kwijt. Dit zijn samen dus 256 mogelijkheden, zoals ook werd verwacht. De meeste computersystemen maken voor het weergeven van negatieve getallen gebruik van het 2-complementsysteem. Systemen die als weergave voor negatieve getallen met het 1-complement werken, zijn er niet zoveel. Een alternatieve manier om negatieve getallen weer te geven is om uitsluitend de tekenbit of sign-bit te gebruiken. Stel, er zijn $n$ bits ter beschikking om een getal weer te geven. We gebruiken dan $(n - 1)$ bits voor de absolute waarde. De resterende bit is het sign-bit en geeft aan of het getal negatief of positief is. De 1- en 2-complementnotatie hebben twee duidelijke verschillen:

1. bij de 1-complementnotatie bestaan een +0 en een −0, tegenover één 0 bij de 2-complementnotatie;
2. bij de 2-complementnotatie hebben we één negatief getal meer, zodat de verdeling asymmetrisch is.

Bij beide systemen valt op dat bij negatieve getallen de hoogste bit een 1 is, en bij positieve getallen een 0. Vandaar dat hier de hoogste bit ook vaak de *tekenbit* of *sign-bit* heet. De 2-complementmethode wordt vrijwel overal gebruikt. Figuren 3.5 en 3.6 maken de methode wat inzichtelijker. Voor vier bits hebben we op een cirkel alle mogelijke opeenvolgende waarden aangegeven. Hiervoor hebben we de hexadecimale notatie gebruikt. De opvolger van het getal F is 0, waarmee de cirkel rond is. Deze cirkel kunnen we op twee manieren over de getallenlijn laten rollen. Bij de ene manier beginnen we op 0 en rollen naar rechts. We kunnen dan de getallen 0 t/m 15 afbeelden op de binaire code. De code F (1111) wordt afgebeeld op 15.
Bij een tweede manier beginnen we ook bij 0, maar nu rollen we niet alleen naar rechts, maar ook naar links. We zien dat de F nu wordt afgebeeld op -1. Het aantal mogelijkheden blijft even groot, maar nu dekken we ook negatieve getallen af en het bereik op het positieve deel van de getallenlijn beperkt zich tot maximaal 7.




**Figuur 3.5** *Mogelijk bereik voor een 4-bits code bij alleen positieve getallen*




**Figuur 3.6** *Mogelijk bereik voor een 4-bits code bij positieve en negatieve getallen*

### 3.2.5 Rekenen met negatieve binaire getallen

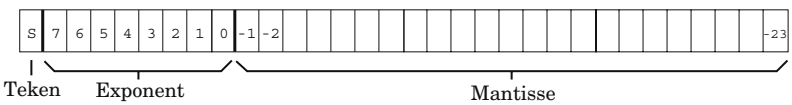
Het aftrekken van getallen kunnen we in het binaire stelsel aanpakken door op te tellen met het gecomplementeerde getal. Om de goede uitkomst te krijgen, moeten we bij het 1-complement de *carry* of overdracht bij het resultaat optellen, terwijl we die bij het 2-complement kunnen negeren. We maken altijd gebruik van het 2-complement, tenzij anders is vermeld. Ook negatieve binaire getallen kunnen op de gebruikelijke wijze in octale of hexadecimale notatie worden weergegeven. Zie tabel 3.3.

**Tabel 3.4** *1- en 2-complementsysteem voor acht bits*

| Decimaal | 1-complement | 2-complement |
|---|---|---|
| 127 | 01111111 | 01111111 |
| 1 | 00000001 | 00000001 |
| +0 | 00000000 | 00000000 |
| –0 | 11111111 | bestaat niet |
| –1 | 11111110 | 11111111 |
| –127 | 10000000 | 10000001 |
| –128 | bestaat niet | 10000000 |

### 3.2.6 Niet-gehele getallen

Tot dusver hebben we, afgezien van de conversie, alleen gehele getallen in het binaire stelsel bekeken. Voor rekentoepassingen is het ook nodig om met breuken en benaderingen van irrationale getallen (zoals $\pi$) te kunnen werken. In computerjargon worden dit floating point getallen genoemd. Meestal wordt de hoeveelheid te gebruiken bits voor de weergave van een floating point getal vastgelegd. Een bekende norm hiervoor is de IEEE 754-standaard. Zo bestaan er afspraken voor 32-bits, 64-bits en 80-bits floating point getallen. De beschikbare bits worden weer verdeeld in twee groepen die we mantisse en exponent noemen. De wijze van weergave van een getal is identiek aan de zogeheten wetenschappelijke notatie, waarbij een getal geschreven wordt als een cijferreeks maal een macht van tien. Zo wordt 123,4567 geschreven als $1{,}234567 \times 102$.
In de representatie van floating point getallen werken we zoals te verwachten niet met machten van tien, maar met machten van twee. In figuur 3.7 is een 32-bits floating point getal te zien. De exponent is acht bits groot, de mantisse 23 bits en we hebben een sign-bit S voor het teken van het getal.




**Figuur 3.7** *32-bits floating point getal*

De vraag is nu natuurlijk: waar staat de komma? De afspraak is dat het cijfer genormaliseerd is. Hiermee wordt bedoeld dat er één cijfer ongelijk aan nul voor de komma staat. Ons eerdere voorbeeld van $1{,}234567 \times 10^2$ is dus een genormaliseerd getal. In het tientallige stelsel hebben we bij genormaliseerde getallen voor het cijfer vóór de komma de keuze uit 1 tot en met 9. In het binaire systeem is dit cijfer altijd 1 (dit is het enige mogelijke cijfer ongelijk aan nul). Bij het hier gepresenteerde coderingssysteem is ervoor gekozen dit cijfer, dat toch altijd 1 is, niet weer te geven, maar het is er natuurlijk wel en, op de twee hierna te bespreken uitzonderingen na, dus altijd 1. De cijfers van de mantisse achter de komma zijn dus weergegeven in de vakjes aangeduid met –1 (voor $2^{-1}$), –2 (voor $2^{-2}$) enzovoort. De winst is dat de effectieve lengte van de mantisse toch 24 bits is, terwijl er maar 23 opgeslagen worden. De exponent geeft aan met welke macht van twee de mantisse vermenigvuldigd moet worden. Ook hier wordt meestal een kunstgreep toegepast. Bij de exponent heeft men de afspraak gemaakt dat bij de werkelijke exponent 127 wordt opgeteld. Het aldus ontstane getal wordt als exponent opgeslagen. Een exponent die kleiner is dan 127 is dus eigenlijk een negatieve exponent. Het hier beschreven systeem kent twee situaties waarbij het weggelaten bit van de mantisse niet 1 is. Bij de weergave voor het floating point getal 0 zijn zowel de exponent als de mantisse nul. Als alleen de exponent nul is, bevat de mantisse een niet-genormaliseerd getal. Op deze manier kunnen nog extra kleine getallen weergegeven worden (ook nu geldt dat de 'echte' exponent –127 is, net als bij genormaliseerde getallen).
Bij floating point getallen hebben we vaak te maken met benaderingen van getallen en kunnen afrondfouten een rol spelen. De hoeveelheid bits voor de mantisse bepaalt de nauwkeurigheid van de benadering. De grootte van de exponent bepaalt het bereik van de getallen.

Door twee vrijwel gelijke floating point getallen van elkaar af te trekken kan er een substantiële fout ontstaan. Een ander effect wat speelt is dat rationale getallen zoals 0,7 geen exacte representatie in floating point format hebben. Dit effect zien we ook bij 1/3 in het decimale systeem. Geschreven in de decimale kommanotatie levert dit een eindeloze reeks 3'tjes: hiervoor is 0,33333333 dus ook een benadering. Vertrouw dus floating point getallen nooit tot het laatste bit en test in een computerprogramma ook geen twee floating point getallen op gelijkheid.

Stellen we dat voor een 32-bits floating point getal T de waarde van het tekenbit is, E de waarde van de exponent en M de uit 23 bits bestaande bitreeks van de mantisse, dan gelden voor de waarde W van 32-bits floating point getallen de volgende afspraken.

Voor $0 < E < 255$ is:

$$W = (-1)^T \times 2^{E-127} \times 1.M$$

Voor E = 0 en M heeft ten minste ergens een 1:

$$W = (-1)^T \times 2^{E-126} \times 0.M$$

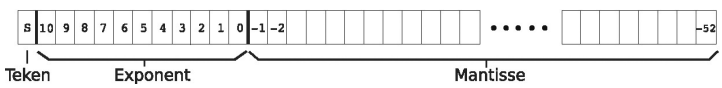
Voor E = 0 en M bevat alleen nullen: afhankelijk van het tekenbit T is

$$W = 0 \text{ of } W = -0$$

Voor E = 255 en M bevat alleen nullen hebben we nog:

$$W = -\infty \text{ als } T = 1 \text{ en } W = \infty \text{ als } T = 0$$

Bij E = 255 en M bestaat niet alleen uit nullen hebben we niet met een geldig getal te maken.




**Figuur 3.8** *64-bits floating point getal*

In figuur 3.8 is het format van een 64-bits floating point getal te zien. Bij programmeertalen wordt dit format vaak als double of double precision aangeduid. Voor de exponent zijn 11 bits beschikbaar en de mantisse is 52 bits groot.
Voor de mogelijkheden voor 64-bits floating point getallen krijgen we een met het 32-bits format vergelijkbaar overzicht, maar wel met aangepaste getallen.
Voor $0 < E < 2047$ is:

$$W = (-1)^T \times 2^{E-1023} \times 1.M$$

Voor E = 0 en M heeft ten minste ergens een 1:

$$W = (-1)^T \times 2^{E-1022} \times 0.M$$

Voor E = 0 en M bevat alleen nullen: afhankelijk van het tekenbit T is

$$W = 0 \text{ of } W = -0$$

Voor E = 2047 en M bevat alleen nullen hebben we nog:
$W = -\infty$ als T = 1 en $W = \infty$ als T = 0

Bij E = 2047 en M bestaat niet alleen uit nullen hebben we niet met een geldig getal te maken.

Passen we deze regels toe en kijken we naar het bereik van 32-bits en 64-bits floating point getallen dan vinden we:

Bereik van 32-bits floating point getallen (single precision)
Negatief van $-3{,}4028235 \times 10^{38}$ tot $-1{,}4 \times 10^{-45}$
De waarde 0
Positief van $1{,}4 \times 10^{-45}$ tot $3;4028235 \times 10^{38}$

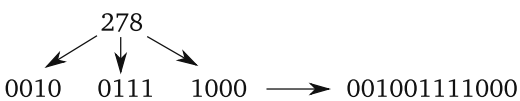
Bereik van 64-bits floating point getallen (double precision)
Negatief van $-1{,}7976931348623157 \times 10^{308}$ tot $-4{,}9 \times 10^{-324}$
De waarde 0
Positief van $4{,}9 \times 10^{-324}$ tot $1{,}7976931348623157 \times 10^{308}$

## 3.3 Codesystemen

In deze paragraaf bespreken we de BCD-code, de ASCII-code en ISO 8859, de Unicode en ISO 10464 en de Gray-code.

### 3.3.1 BCD-code

Omdat het decimale stelsel zoveel wordt gebruikt, bestaat er een binaire representatie van het decimale stelsel die BCD (binary coded decimal) heet. In deze representatie wordt een binair getal van vier bits gebruikt om er een cijfer van 0 tot en met 9 in op te slaan. In acht bits kunnen we dan getallen van 0 tot en met 99 kwijt. Bijvoorbeeld: 01111001 is 79 in BCD-code. Aangezien we normaal in vier bits een cijfer van 0 tot en met 15 kunnen opslaan, levert dit dus enig verlies op. Dit verlies wordt gecompenseerd door het feit dat veel processors rechtstreeks met getallen in deze representatie kunnen rekenen en dat ook de conversie naar normale decimale representatie zeer simpel is.




**Figuur 3.9** *Omzetting van decimaal getal naar BCD-code*

In de digitale techniek bestaan schakelingen die teller of counter heten. Binaire tellers tellen volgens het tweetallige stelsel. Daarnaast zien we ook regelmatig BCD-counters toegepast worden. Deze counters werken met groepjes van vier bits, waarbij elk groepje de cijfers 0 tot en met 9 (0000 tot en met 1001) kan weergeven. Na de 9 (1001) volgt weer

de 0 (0000). Deze tellers volgen de BCD-code en worden vaak toegepast in situaties waar een tellerstand rechtstreeks op een display wordt weergegeven. Elk groepje van vier bits stuurt dan een display aan dat de cijfers 0 tot en met 9 kan weergeven. Een dergelijk display bestaat uit zeven segmenten die in de vorm van een 'acht' staan. Het display kan een cijfer weergeven als de juiste segmenten geactiveerd worden.

### 3.3.2 ASCII-code en ISO 8859

**Tabel 3.5** *ASCII-tabel met decimale code, gevolgd door ASCII-teken*

| **7-bits ASCII** | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | NULL | 32 | space | 64 | @ | 96 | ‘ |
| 1 | SOH | 33 | ! | 65 | A | 97 | a |
| 2 | STX | 34 | “ | 66 | B | 98 | b |
| 3 | ETX | 35 | # | 67 | C | 99 | c |
| 4 | EOT | 36 | $ | 68 | D | 100 | d |
| 5 | ENQ | 37 | % | 69 | E | 101 | e |
| 6 | ACK | 38 | & | 70 | F | 102 | f |
| 7 | BELL | 39 | ‘ | 71 | G | 103 | g |
| 8 | BS | 40 | ( | 72 | H | 104 | h |
| 9 | HT | 41 | ) | 73 | I | 105 | i |
| 10 | LF | 42 | * | 74 | J | 106 | j |
| 11 | VT | 43 | + | 75 | K | 107 | k |
| 12 | FF | 44 | , | 76 | L | 108 | l |
| 13 | CR | 45 | – | 77 | M | 109 | m |
| 14 | SO | 46 | . | 78 | N | 110 | n |
| 15 | SI | 47 | / | 79 | O | 111 | o |
| 16 | DLE | 48 | 0 | 80 | P | 112 | p |
| 17 | DC1 | 49 | 1 | 81 | Q | 113 | q |
| 18 | DC2 | 50 | 2 | 82 | R | 114 | r |
| 19 | DC3 | 51 | 3 | 83 | S | 115 | s |
| 20 | DC4 | 52 | 4 | 84 | T | 116 | t |
| 21 | NAK | 53 | 5 | 85 | U | 117 | u |
| 22 | SYN | 54 | 6 | 86 | V | 118 | v |
| 23 | ETB | 55 | 7 | 87 | W | 119 | w |
| 24 | CAN | 56 | 8 | 88 | X | 120 | x |
| 25 | EM | 57 | 9 | 89 | Y | 121 | y |
| 26 | SUB | 58 | : | 90 | Z | 122 | z |
| 27 | ESC | 59 | ; | 91 | [ | 123 | { |
| 28 | FS | 60 | < | 92 | \ | 124 | \| |
| 29 | GS | 61 | = | 93 | ] | 125 | } |
| 30 | RS | 62 | > | 94 | ^ | 126 | ~ |
| 31 | US | 63 | ? | 95 | _ | 127 | DEL |

De *ASCII-code* (American Standard Code for Information Interchange) is in gebruik om dataoverdracht in tekstvorm tussen computers te standaardiseren. Het is in feite een tabel die aan allerlei cijfers, letters en tekens een code toekent van zeven bits. De eerste 32 codes (0 tot en met 31) zijn functiecodes. Deze staan niet zozeer voor een teken op een scherm of blad papier, maar voor speciale functies als Backspace (een positie terug), Line Feed (nieuwe regel), Carriage Return (terug naar het begin van de regel) en andere codes die een rol spelen bij datacommunicatie.
De ASCII-tabel (tabel 3.5) geeft de decimale code met het bijbehorende ASCII-teken. Door de decimale code om te zetten naar binaire of hexadecimale code is ook de hexadecimale of binaire representatie van de ASCII-code te verkrijgen. Zo is de binaire ASCII-code voor de letter 'A' gelijk aan 1000001. Dit kunnen we in hexadecimale notatie schrijven als 0x41. Merk op dat de code voor de cijfers niet gelijk is aan de overeenkomstige binaire waarde. Zo is de ASCII-code voor het *cijfer* 5 gelijk aan 0110101, terwijl de binaire weergave van het *getal* vijf gelijk is aan 0000101 (als we zeven bits gebruiken). De ASCII-code speelt niet alleen bij tekstoverdracht een rol; ook bestanden met teksten erin zijn in geval het programmeercode betreft meestal gebaseerd op de ASCII-code. Veel hoge programmeertalen hebben als basisset voor de programeertaal de ASCII-code.
Doordat computers liever met bytes (acht bits) dan met codes van zeven bits werken en omdat de ASCII-code een aantal tekens mist, bestaan er ook allerlei vormen van 'Extended ASCII'. Hierbij wordt de verdubbeling van het aantal mogelijke codes door het toevoegen van een bit aan de 7-bits ASCII gebruikt om codes te definiëren voor in niet-Engelse taal gebruikelijke letters met accenten, trema's enzovoort. Een bekende standaard hiervoor is ISO 8859. In deze standaard zijn diverse coderingen voor de beschikbare achtbits code vastgelegd. Kiezen we het meest significante bit 0, dan hebben we weer te maken met de eerdergenoemde 7-bits ASCII.
Daarnaast kent de standaard een uitbreiding voor West-Europese talen. Deze standaard heet ISO 8859-1, ook wel aangeduid als Latin1. Ook talen die niet van het Romaanse schrift gebruikmaken, hebben een ISO 8859-standaard. Het Romaanse schrift is de verzameling lettertekens waar West-Europese talen gebruik van maken, zeg maar 'ons' alfabet. Talen die hun eigen combinaties van Romaanse en diakritische tekens nodig hebben, hebben ook een eigen 8859-versie. Tabel 3.6 geeft een niet-compleet overzicht. De introductie van de euro heeft ook hier zijn impact gehad. ISO 8859-15 is vrijwel gelijk aan Latin1, maar bevat op de positie van code 0xA4 (10100100) het eurosymbool.

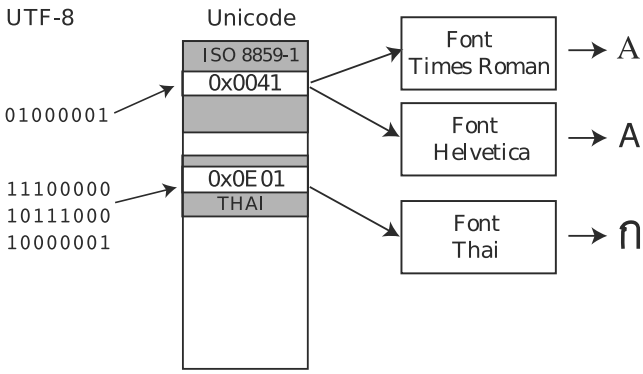
**Tabel 3.6** *ISO 8859-standaarden*

| Standaard | Naam | Toepassing |
|---|---|---|
| ISO 8859-1 | Latin1 | West-Europese talen |
| ISO 8859-2 | Latin2 | Centraal- en Oost-Europese talen |
| ISO 8859-5 | Cyrillic | Talen die het cyrillische schrift gebruiken (waaronder Russisch) |
| ISO 8859-6 | Arabic | Arabisch |
| ISO 8859-7 | Greek | Grieks |
| ISO 8859-8 | Hebrew | Hebreeuws |
| ISO 8859-9 | Latin5 | Turks |
| ISO 8859-11 | Thai | Thaise taal |
| ISO 8859-15 | Latin9 | Latin1 met eurosymboolcode (wordt ook wel Latin0 genoemd) |

### 3.3.3 Unicode en ISO 10464

ASCII is gebaseerd op de schrifttekens van ons alfabet. Daarnaast zijn er nog een heleboel andere schriften, zoals het Grieks, Hebreeuws, Arabisch, Thai, Devanagari, Han, Katakana enzovoort. Gedeeltelijk hebben de ISO 8859-coderingen dit opgelost, maar het probleem blijft dat men moeilijk combinaties van tekens uit verschillende talen kan maken. Een computersysteem uit Thailand, dat gebruik maakt van ISO 8859-11, zal bij de weergave van een in de Franse taal opgestelde tekst, alle tekens met accenten vervangen door Thaise lettertekens. De codes overlappen daar namelijk. Dus feitelijk verandert de inhoud van de tekst niet, maar de weergave klopt voor geen meter. Twee projecten zijn opgezet om een wereldwijd toepasbare codering te maken. Dit waren het ISO 10464-project en het Unicode-project. Gelukkig is men het al snel eens geworden en is ISO 10464 vrijwel gelijk aan de Unicode. De Unicode is een standaard om alle verschillende lettertekens in één codering onder te brengen. Oorspronkelijk was het voorstel om een 16-bits code toe te passen. Uiteindelijk zijn er meer standaardcoderingen uitgerold. Dit had onder meer te maken met het feit dat veel bestaande programmatuur uitgaat van ASCII-code al dan niet met extensie naar 8 bits. Naast lettertekens zijn er ook codes gereserveerd voor speciale tekens zoals cijfers, leestekens van allerlei aard, wiskundige symbolen en nog veel meer. Van de 'universal character set' (UCS) bestaan twee uitwerkingen: UCS-2 en UCS-4. UCS-2 is een 16-bits code en UCS-4 is 32 bits. Een ISO Latin1-tekstbestand kan omgezet worden naar UCS-2 door elk byte vooraf te laten gaan door 00000000 (een byte gevuld met nullen). Voor UCS-4-codering dienen we zelfs drie bytes met nullen voor de code te plaatsen. Beide coderingen laten dan nog veel ruimte vrij voor symbolen. In de meeste gevallen kan weer met een subset volstaan worden. Zo is MES-3 een subset van 2819 symbolen waar alle Europese talen mee uit de voeten kunnen.
UCS-2 kent 65534 posities voor tekens (0x0000 t/m 0xFFFD). Hierin zijn op systematische wijze alle bestaande encoding-standaarden samengevat. Deze set noemt men wel het 'basic multilingual plane' (BMP). Oorspronkelijk is uitgegaan van een 16-bits code. Hierin zijn characters in eenheden van 16 bits gecodeerd. Dit coderen wordt ook wel encoding genoemd. In UCS-2 is de encoding van tekens 16 bits, UCS-4 daarentegen is een encoding-schema van 32 bits per teken. Naast deze encoding bestaan er unicode transformation formats, afgekort met UTF. Bij deze UTF's wordt een koppeling beschreven tussen de oorspronkelijke code en een nieuwe wijze van codering. Dit lijkt wat omslachtig, maar heeft toch wel voordelen. Bij toepassing van UTF-8-codering blijkt oude software geen probleem te hebben met Unicode, zolang er van 7-bits ASCII gebruikgemaakt wordt.
UTF-8 is een codering die voor tekens uit de Unicode-set een speciale techniek van herafbeelding gebruikt. UTF-8 maakt voor elk teken uit de Unicode-set één of meer bytes.

00000000-0000007F: 0xxxxxxx
00000080-000007FF: 110xxxxx 10xxxxxx
00000800-0000FFFF: 1110xxxx 10xxxxxx 10xxxxxx
00010000-001FFFFF: 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
00200000-03FFFFFF: 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
04000000-7FFFFFFF: 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx



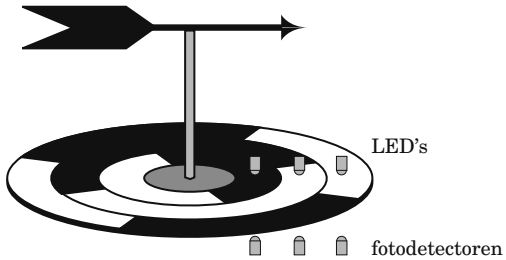

**Figuur 3.10** *UTF, UCS en fontbeschrijvingen*

UTF-16 past 16-bits codes toe. Eventueel kunnen twee 16-bits codes weer samen gebruikt worden om een groot gebied van extra tekens toegankelijk te maken. De hier genoemde coderingen beschrijven een koppeling van een binaire code naar een teken. Hoe dat teken er precies uit komt te zien, wordt niet door deze codering bepaald, maar is afhankelijk van het toegepaste lettertype of font. Het font beschrijft de uiterlijke vorm van het teken (vet, cursief, met of zonder schreven, grootte enzovoort).
Om het Unicode-verhaal te verduidelijken is in figuur 3.10 aangegeven hoe de verschillende Unicode-standaarden zich verhouden. Met UCS-2 of UCS-4 zijn characternamen en bijbehorende codes van óf 16 óf 32 bits (gehele getallen) vastgelegd. Bijvoorbeeld: U + 0041 (UCS-2-notatie) of U + 00000041 (UCS-4) stelt de letter A voor en heeft de officiële naam 'LATIN CAPITAL LETTER A'. De afbeelding op bijvoorbeeld een beeldscherm of printer wordt bepaald door een font (verzameling lettervormen, zoals Times Roman, Helvetica). Hierin kan overlap zitten aangezien er veel fonts beschikbaar zijn voor bijvoorbeeld de Latin-set. Met UTF-8 kiezen we weer met een 8-bits encoding om welk UCS-2 of UCS-4 character het gaat.

### 3.3.4 Gray-code

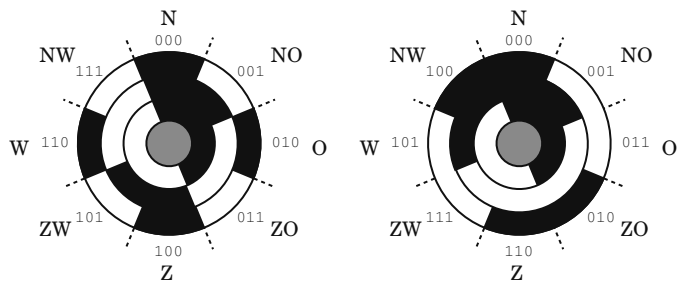
De Gray-code is afkomstig uit de meetwereld. Waarom deze code wordt gebruikt, zullen we toelichten aan de hand van een voorbeeld. We meten de stand van een as met behulp van een op de as gemonteerde schijf die op bepaalde plaatsen wél en op andere plaatsen niet lichtdoorlatend is. Met lichtgevende diodes (led's) en lichtdetectoren wordt de verdraaiing gemeten. In het voorbeeld van figuur 3.11 gaat het om een windrichtingsmeter. De fotodetectoren leveren een binair getal evenredig met bijvoorbeeld de hoekverdraaiing.

Als we uitgaan van een normale binaire volgorde, dan meten we op een bepaald moment %011 en op een volgend moment %100. Tussen deze twee metingen zijn drie bits omgeklapt. Omdat we te maken hebben met kleine onnauwkeurigheden in de schijf en spreiding in het omklappunt van de fotodetectoren, kunnen we er vrijwel zeker van zijn dat het omklappen niet tegelijk zal gebeuren. In het geval van een mechanisch systeem met schakelaartjes komt bij dit probleem ook nog het klapperen van de




**Figuur 3.11** *Windrichtingsmeter*

schakelaars (contactdender). Bovendien zou de as zich wel eens precies op de omklappositie kunnen bevinden. Gezien deze problemen en het aantal bits dat in één keer omklapt, kunnen we elk getal meten tussen %000 en %111. De afwijking kan dus zeer aanzienlijk zijn. De Gray-code is een code die veel op de binaire code lijkt, maar heeft de eigenschap dat tussen twee opeenvolgende getallen slechts één bit van waarde verandert. De afwijking kan nu dus nooit meer dan één bit zijn. In figuur 3.12 is de meetschijf voor de binaire en de Gray-code getekend.




**Figuur 3.12** *Binaire en Gray-code meetschijf*

De Gray-code heet ook wel *reflected code* of *spiegelcode*, omdat in de reeks van opeenvolgende Gray-codes per bitpositie de code gespiegeld herhaald wordt. Kijken we bijvoorbeeld naar het LSB van opeenvolgende codes dan zien we 01 gevolgd door 10. Hierna volgt een vertaaltabel voor de diverse getalrepresentaties en de Gray-code voor vier bits, zie tabel 3.7.

**Tabel 3.7** *Vertaaltabel met diverse codes*

| Binair | Octaal | Decimaal | Hexadecimaal | Gray |
|---|---|---|---|---|
| 0000 | 0 | 0 | 0 | 0000 |
| 0001 | 1 | 1 | 1 | 0001 |
| 0010 | 2 | 2 | 2 | 0011 |
| 0011 | 3 | 3 | 3 | 0010 |
| 0100 | 4 | 4 | 4 | 0110 |
| 0101 | 5 | 5 | 5 | 0111 |
| 0110 | 6 | 6 | 6 | 0101 |

| | | | | |
|---|---|---|---|---|
| 0111 | 7 | 7 | 7 | 0100 |
| 1000 | 10 | 8 | 8 | 1100 |
| 1001 | 11 | 9 | 9 | 1101 |
| 1010 | 12 | 10 | A | 1111 |
| 1011 | 13 | 11 | B | 1110 |
| 1100 | 14 | 12 | C | 1010 |
| 1101 | 15 | 13 | D | 1011 |
| 1110 | 16 | 14 | E | 1001 |
| 1111 | 17 | 15 | F | 1000 |

## 3.4 Fouten in codes

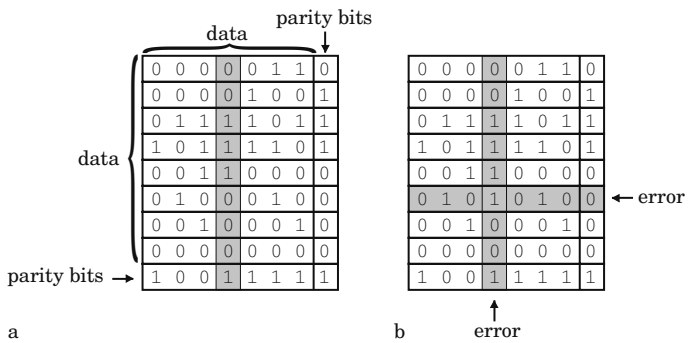
Bij het opslaan en transport van grote hoeveelheden nullen en enen kan er wel eens iets fout gaan. Een 0 wordt opgeslagen als 1 of omgekeerd wordt een 1 een 0. Voor computersystemen kan zo'n fout rampzalige gevolgen hebben. Daarom zijn er allerlei systemen bedacht om fouten op te sporen. We noemen dit *errordetectie*. Wat verder gaat is *errorcorrectie*, waarbij de fout niet alleen wordt opgespoord maar ook gecorrigeerd. De techniek van errorcorrectie wordt vaak aangeduid met de afkorting ECC. Deze afkorting staat voor 'error correcting code'.
We gaan hierna eerst in op pariteit (paragraaf 3.4.1). Daarna bespreken we de Hamming-code en de Hamming-afstand (paragraaf 3.4.2 en 3.4.3). Tot slot komt in paragraaf 3.4.4 de cyclische redundancy check (CRC) aan bod, een betrouwbaar systeem om fouten te detecteren.

### 3.4.1 Pariteit

Een eenvoudige manier om errordetectie te verwezenlijken, is om aan een code van een zeker aantal bits (bijvoorbeeld zeven of acht bits) een extra bit toe te voegen die het totale aantal 'enen' even maakt. We noemen deze extra bit een pariteitsbit en we spreken van *even pariteit*. Natuurlijk hadden we ook voor oneven pariteit kunnen kiezen.
Bij het opnieuw lezen van de code controleren we de pariteit, met andere woorden: we controleren in ons geval van even pariteit of het aantal enen ook even is. Dit systeem van errordetectie is verre van waterdicht. Als er sprake is van twee omgeklapte bits, dan glipt een foutieve code door het detectiesysteem. Toch wordt pariteitscontrole vanwege de eenvoudige praktische realisatie veel gebruikt.
Een alternatieve manier om naar pariteit te kijken, is om het puur te zien als de optelling van de afzonderlijke bits, waarbij alleen de minst significante bit (least significant bit, LSB) wordt bewaard. Voorbeelden van deze zogenoemde modulo-twee optelling zijn:

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 0$$
$$1 + 1 + 1 + 1 + 1 = 1$$

In de sommen is te zien dat de uitkomst ervoor zorgt dat het totale aantal enen, links en rechts van het isgelijkteken, in de som 'even' is; we spreken hier dus van even pariteit. Een geavanceerdere techniek is weergegeven in figuur 3.13 waarbij van een blok codes per code de pariteit wordt gemaakt. Vervolgens wordt van alle eerste bits van alle codes ook de pariteit bepaald. Zie bijvoorbeeld de verticale grijze balk in de figuur. Als bij controle blijkt dat een horizontale en een verticale pariteit niet kloppen, dan is op eenvoudige wijze de probleembit te detecteren en vervolgens te corrigeren, zie figuur 3.13b. In deze figuur is een foutieve bit geplaatst, en de pijltjes bij de daardoor verkeerde pariteitsbits komen samen op de foute bit. Ook hier is weer sprake van allerlei gaten in de detectie en correctie, maar bedenk dat het een zeldzame gebeurtenis is wanneer een bit in een computergeheugen spontaan omklapt.




**Figuur 3.13** *Horizontale en verticale pariteit*

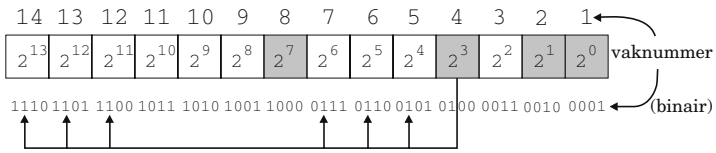
**Intermezzo**

Voor de liefhebbers volgt een korte beschouwing over de bit in de hoek van het blok helemaal rechtsonder (figuur 3.13). In het geval van even pariteit is deze bit eigenlijk het even-pariteitsbit van het hele datablok (dus zonder de andere pariteitsbits). Of we deze bit maken door de even pariteit van de pariteitsbits van boven naar beneden te nemen of van de pariteitsbits van links naar rechts maakt niet uit, omdat we een modulo-twee optelling (wat eigenlijk een pariteitsberekening is) op een willekeurige manier in deelberekeningen mogen splitsen. Hadden we gekozen voor oneven pariteit, dan was dit verhaal niet opgegaan. Oneven pariteit is een modulo-twee optelling plus 1. Als we dit in deelberekeningen uitvoeren, ontstaan er problemen als we in het ene geval een oneven aantal deelberekeningen nemen en in het andere geval een even aantal deelberekeningen. Dit probleem is in ons voorbeeld van figuur 3.13 simpel na te gaan door de horizontale en verticale pariteitsbits oneven te maken. We hoeven dan de bits alleen maar van waarde te veranderen ofwel te inverteren.
De bit in de hoek rechtsonder blijkt bij berekening niet meer eenduidig te zijn.

### 3.4.2 Hamming-code

De Hamming-code is een codering met een verzameling pariteitsbits. De code is zo bedacht dat enkele bitfouten ontdekt en gecorrigeerd kunnen worden. Stel, we hebben

een binaire code van tien bits. We voegen hier vier pariteitsbits aan toe. Hoe we die pariteit maken, zien we dadelijk. De pariteitsbits zetten we nu in een nieuwe 14-bits code op de bitposities die overeenkomen met een tweemacht. In figuur 3.14 zijn deze posities met grijs aangegeven. Het gaat dus om de bits met vaknummer 1, 2, 4 en 8.




**Figuur 3.14** *Hamming-codebits met databits*

Elke bit is de pariteit van een deelverzameling van de oorspronkelijke bits. De bitposities in de nieuwe code (van 14 bits dus met de pariteitsbits) geven we een binair nummer. Deze nummers lopen van %0001 tot en met %1110 (1 tot en met 14). De waarde van de toegevoegde bit op positie 1 is de pariteit van alle bits die zich bevinden op een positie waarvan het LSB 1 is. Dit zijn dus de bits op positie 3, 5, 7, 9, 11 en 13. Met deze bits maken we een even-pariteitsbit die ervoor zorgt dat het aantal enen als geheel even wordt. We noteren dit als:

$$P(1) = P(3, 5, 7, 9, 11, 13)$$

Voor de pariteitsbit op positie 2 (%0010) komen de bits in aanmerking waarvan de binaire waarde van de bitpositie op de tweede plek van rechts een 1 is, net zoals %0010 zelf. Dus:

$$P(2) = P(3, 6, 7, 10, 11, 14)$$

Op dezelfde wijze volgen $P(4)$ en $P(8)$:

$$P(4) = P(5, 6, 7, 12, 13, 14)$$
$$P(8) = P(9, 10, 11, 12, 13, 14)$$

Figuur 3.14 geeft dit weer voor $P(4)$. Deze aanpak geeft het grappige resultaat dat, als er iets mis is met één of meer pariteitsbits, de positie van de bit die voor deze pariteitsfouten verantwoordelijk is gemakkelijk gevonden kan worden door de vier pariteitsbits zelf als een 4-bits code te lezen met een binaire waarde. Een juiste pariteit lezen we als een 0, een niet-juiste als een 1. Met andere woorden: kloppen $P(1)$ en $P(8)$ niet, dan is de bit op positie 9 (%1001 ofwel $2^3 + 2^0$) fout.

### 3.4.3 Hamming-afstand en Hamming-gewicht

Stel we beschouwen een verzameling van codes, dan is de Hamming-afstand 1 als er in de verzameling codes een codepaar te vinden is dat maar op één bitpositie verschilt. Bij zo'n verzameling codes kan dus door het omklappen van één bit de ene code in de andere overgaan. Kiezen we voor een Hamming-afstand van 3, dan verschillen twee

willekeurige codes uit de verzameling altijd op ten minste drie plaatsen. Het omklappen van een bit levert een code op die niet in de verzameling thuishoort en die een Hamming-afstand 1 heeft met de oorspronkelijke code. De Hamming-afstand tot de andere codes is minimaal 2. Op deze wijze kan men met het toevoegen van bits kunstmatig de Hamming-afstand opvoeren, om eventuele fouten later te kunnen detecteren en corrigeren.
Van de Gray-code kunnen we zeggen dat de Hamming-afstand tussen twee opeenvolgende codes altijd 1 is. Dit was juist de reden om deze code bij metingen toe te passen.
Onder Hamming-gewicht wordt verstaan het aantal tekens in een code dat van 0 verschilt. Voor een binaire code is dat dus het aantal enen. Het Hamming-gewicht is daarmee gelijk aan de Hamming-afstand tot de code die alleen nullen bevat.

### 3.4.4 CRC

Bij grote stromen van bits, zoals die in computernetwerken en bij opslag van gegevens bestaan, is de cyclische redundancy check, afgekort tot CRC, een betrouwbaar systeem om fouten op te sporen.
Alvorens ons in deze techniek te storten, eerst even wat wiskunde. Een getal in een zeker talstelsel is te schrijven als een polynoom of veelterm. Bijvoorbeeld het getal 805 in het tientallige stelsel schrijven we als:

$$805 = 8 \times 10^2 + 0 \times 10^1 + 5 \times 10^0$$

Zo is een willekeurig (natuurlijk) getal van $N + 1$ cijfers te schrijven als:

$$G(x) = \sum_{n=0}^{N} a_n x^n = a_N x^N + a_{N-1} x^{N-1} \cdots a_2 x^2 + a_1 x + a_0$$

Hierin is $x$ het grondtal van het talstelsel en zijn $a_n$ de factoren waarmee de betreffende macht van het grondtal wordt vermenigvuldigd. In het tweetallige stelsel nemen de factoren $a_n$ alleen de waarden 0 of 1 aan; $x$ is in het binaire stelsel 2. Deze polynoomnotatie wordt hier genoemd, omdat er bij de CRC-methode sprake is van een generatorpolynoom. Deze polynoom wordt bij het opbouwen van een checksum gebruikt en meestal gegeven in de polynoomnotatie. Het is natuurlijk 'gewoon' een binair geschreven getal.
De CRC-methode voegt aan een stroom bits een tijdens het verzenden uitgerekende code toe. De ontvangkant voert dezelfde bewerking uit ter controle. De uitgerekende code is het resultaat van een deling van de datastroom, die nu als een groot binair getal wordt opgevat door een van tevoren afgesproken binair getal, de zogenoemde generatorpolynoom. Deze rest wordt aan de oorspronkelijke bits toegevoegd. Het resultaat is een iets uitgebreidere stroom bits die bij deling door de generatorpolynoom de waarde 0 oplevert. Mochten er ergens bits omklappen of verloren gaan, dan zal bij controle de deling met zeer grote waarschijnlijkheid geen 0 opleveren en wordt de bitstroom (een datablock van disk of netwerkpacket) afgekeurd: we hebben een CRC-error.
Als we nu het voorbeeld van figuur 3.15 bekijken, zien we dat de generatorpolynoom %10101 is. Schrijven we dit als som van tweemachten, dan vinden we:

$$2^4 + 2^2 + 2^0$$

Dit wordt in polynoomnotatie geschreven als:

$$G(x) = x^4 + x^2 + 1$$

Als voorbeeld van een bitstroom gebruiken we de (wel erg korte) reeks %110001110. Allereerst wordt deze reeks met vier nullen uitgebreid. Na deling ontstaat een bitstroom met CRC.

| bitstroom | bitstroom × 10000 | generator-polynoom |
|---|---|---|
| 110001110 | 1100011100000 | G(x) = 10101 |

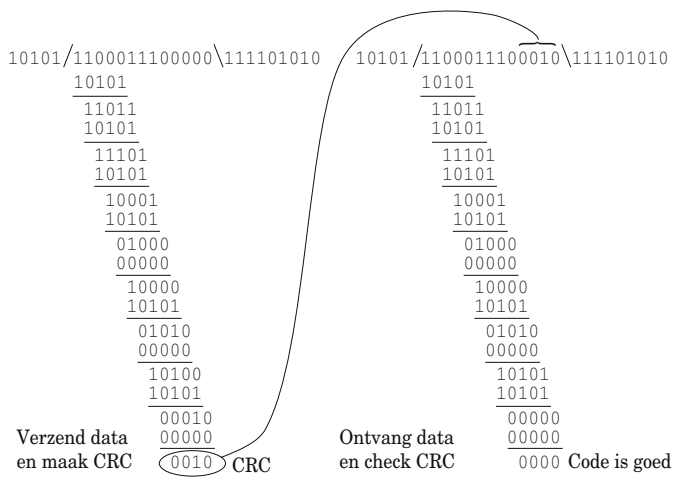
| bitstroom met CRC |
|---|
| 1100011100010 |

**Figuur 3.15** *CRC*

*G*(*x*) bestaat uit een sommatie van *x*-machten die overeenkomen met het gewicht van de bitpositie van de enen in de generatorpolynoom. De nullen leveren zoals gebruikelijk niets op. Uit figuur 3.12 blijkt dat de deling wat ongebruikelijk verloopt; er wordt namelijk bij het aftrekken met modulo-twee gewerkt, zoals ook bij de pariteitsberekening is gedaan. De rekenregels voor modulo-twee aftrekken zijn:

$$0 - 0 = 0$$
$$0 - 1 = 1$$
$$1 - 0 = 1$$
$$1 - 1 = 0$$

Deze regels blijken exact hetzelfde te zijn als modulo-twee optellen.




**Figuur 3.16** *CRC-generatie en check*

In eerste instantie is de bitstroom uitgebreid met vier nullen (een cijfer minder dan de 5-cijferige generatorpolynoom %10101). De generatorpolynoom wordt langs de bitstroom 'geschoven'. Daar waar de hoogste bit ($x^4$) samenvalt met een 1, noteren we