ir. F.J. Dijkstra

Computers

organisatie / architectuur / communicatie

Wolters Noordhoff

Vierde druk

Computers

Computers

organisatie / architectuur / communicatie

Ir. Frank Dijkstra

Wolters-Noordhoff Groningen | Houten

Ontwerp omslag: G2K Designers Groningen/Amsterdam
Omslagillustratie: G2K Designers Groningen/Amsterdam

Wolters-Noordhoff bv voert voor het hoger onderwijs de imprints Wolters-Noordhoff, Stenfert Kroese, Martinus Nijhoff en Vespucci.

Eventuele op- en aanmerkingen over deze of andere uitgaven kunt u richten aan: Wolters-Noordhoff bv, Afdeling Hoger Onderwijs, Antwoordnummer 13, 9700 VB Groningen, e-mail: info@wolters.nl

1 2 3 4 5 / 10 09 08

ISBN (ebook) 978-90-01-84930-6
ISBN 978-90-01-26707-0
NUR 123

Woord vooraf

Dit boek dankt zijn oorsprong aan de cursussen 'Computerorganisatie' die ik een aantal jaren verzorgde voor de opleiding Bedrijfskundige Informatica van de Hogeschool Utrecht en voor de opleiding Hogere Informatica in 's-Hertogenbosch. De bestaande literatuur bleek niet te voldoen omdat die te gespecialiseerd was, slechts een beperkt aantal onderwerpen behandelde of een groot aantal zaken in het vage liet. Het boek is bedoeld voor het eerste of tweede leerjaar van het Hoger Beroepsonderwijs. Hierbij denk ik in eerste instantie aan BI- en HI-opleidingen, maar ook bij de opleidingen elektrotechniek, de AOT, het Nautisch onderwijs, Hogere Labaratoriumscholen en dergelijke is het bruikbaar. Daarnaast is het geschikt voor zelfstudie omdat geen speciale voorkennis wordt verondersteld.
Bij veel opleidingen wordt voortdurend de vraag gesteld in hoeverre het noodzakelijk is nog iets van computerarchitectuur af te weten. Naarmate de computer volwassener wordt, is de noodzaak daartoe afgenomen, op dezelfde manier als het rijden in een auto niet meer vereist dat we op welke manier dan ook op de hoogte zijn van wat zich in het binnenwerk afspeelt. Voor wie een computer gebruikt om zich informatie te verschaffen, een tekst te schrijven of een spelletje te spelen, is de computer zelf van weinig belang. Dit boek is dan ook bedoeld voor de nieuwsgierigen die zich niet tevreden stellen met het gebruiken van een computer, maar ook willen weten hoe het werkt, hoe de onderliggende principes zijn, en hoe het mogelijk is om uit een handjevol transistoren een instrument te maken dat zoveel intelligentie blijkt te hebben. Daarnaast is het bedoeld voor degenen die door hun technische opleiding iets over de mogelijkheden van de computer moeten weten: er wordt geen broodrooster meer gemaakt zonder ingebouwde intelligentie en geen bedrijf kan nog overleven als niet verregaand geautomatiseerd wordt.
Eén van de belangrijke oogmerken van dit boek is ook het introduceren en verklaren van een groot aantal termen, begrippen en acroniemen die gebruikt worden in de computertechniek en de informatica. Veelal zijn het Engelse woorden, waarvoor soms ook wel Nederlandse vertalingen bestaan. Ik heb bewust geen poging gedaan de Engelse termen te vertalen: vrijwel alle literatuur over dit onderwerp is in het Engels geschreven en vertaling van specifieke begrippen werpt alleen maar een barrière op bij verdere studie.
Een computer is een ingewikkeld instrument dat we op verschillende manieren kunnen beschouwen: als een elektronische schakeling, als een rekenmachine, als een verzameling programma's die uitgevoerd kunnen worden. In dit boek wordt daarom een indeling gemaakt van de computer in verschillende conceptuele niveaus, die hiërarchisch boven elkaar gelegen zijn. Deze niveaus worden 'bottom up' behandeld: van elk niveau worden alleen die onderdelen behandeld die nodig zijn om het volgende goed te kunnen begrijpen. Het doel hiervan is de omvang van het boek beperkt te houden en de minder technisch geschoolde lezer niet lastig te vallen met begrippen en problemen die voor het onderwerp van minder belang zijn. De behandeling is echter nauwkeurig genoeg om ook de technisch ingestelde lezer voldoende aanknopingspunten te bieden voor verdere studie. Om die reden is ook gekozen voor de IEC-notatie van poorten. Dit maakt het boek bruikbaar bij opleidingen als Elektrotechniek, waar deze notatie ook bij andere vakken gebruikt wordt.

Voor een diepgaander behandeling van de verschillende onderwerpen zijn vele boeken voorhanden. Hierbij denk ik vooral aan de drie boeken van Andrew S. Tanenbaum: '*Structured Computer Architecture*', '*Operating Systems*' en '*Computer Networks*', waarvan ook Nederlandse vertalingen bestaan. De terminologie en de indeling van de stof in dit boek zijn zo gekozen dat ze hiermee in overeenstemming zijn.

Bij de derde druk

In de derde druk van dit boek is een aantal zaken veranderd en aangepast. Een extra hoofdstuk is opgenomen met uitwerkingen en antwoorden van alle opgaven, zodat de student meer mogelijkheden heeft zichzelf te testen. Aan het einde van elk hoofdstuk is bovendien een lijst met kernbegrippen toegevoegd die op zichzelf een klein repertitorium van het hoofdstuk vormt. Daarnaast wordt een programma meegeleverd dat de in het boek beschreven elementaire processor simuleert. De student kan dit programmeren vanaf één enkele instructie tot aan een compleet programma en alle uitvoeringsstappen en interne bewerkingen volgen. Voor de docent is de simulator bruikbaar bij presentaties en het geven van opdrachten.

Bij de vierde druk

De vierde druk is in grote lijnen gelijk aan de voorgaande. Aanvullingen en veranderingen zijn vooral te vinden in de hoofdstukken 4, 7 en 8. Enkele opgaven zijn toegevoegd, waaronder ook een aantal voor de ZEP2-simulator.
Verder is een website beschikbaar: www.computers.wolters.nl. Hier kunnen PowerPointpresentaties van alle tekeningen uit het boek worden gedownload, evenals de simulatiesoftware en een docentenhandleiding.

januari 2006

Ir F.J. Dijkstra, 's-Hertogenbosch, januari 2006

Inhoud

Coderingen

1

Het eerste hoofdstuk van dit boek houdt zich bezig met de manier waarop getallen, teksten en andere gegevens in een computer worden opgeslagen. Alle informatie moet eerst vertaald worden naar een digitale vorm. In paragraaf 1.1 wordt uitgelegd wat de begrippen *analoog* en *digitaal* precies betekenen.
De kleinst mogelijke digitale eenheid wordt een *bit* genoemd. Door combinaties van bits ontstaat een computertaal waarin alle mogelijke gegevens kunnen worden gecodeerd.
Om te begrijpen hoe getallen in bits worden omgezet, maken we in paragraaf 1.2 een uitstapje naar andere talstelsels dan het ons vertrouwde tientallige stelsel. In paragraaf 1.3 onderzoeken we hoe een computer aldus gecodeerde getallen kan optellen en vermenigvuldigen. Wanneer we niet alleen willen optellen maar ook aftrekken, hebben we negatieve getallen nodig; hierover gaat paragraaf 1.4. Een aantal andere mogelijkheden voor codering van getallen wordt in paragraaf 1.5 behandeld, waarbij ook de in computers veelgebruikte wetenschappelijke notatie ter sprake komt.
Behalve getallen moet de computer ook andersoortige gegevens kunnen verwerken. Letters en andere tekens hebben daarom eigen codes; dit wordt in paragraaf 1.6 verder uitgewerkt.
Paragraaf 1.7 gaat over methodes om bij het coderen zo min mogelijk bits te gebruiken en over manieren om te zorgen dat de informatie beveiligd kan worden tegen ongewenste inkijk (compressie en encryptie). Ten slotte worden in paragraaf 1.8 codes behandeld die het ontstaan van fouten kunnen tegengaan.

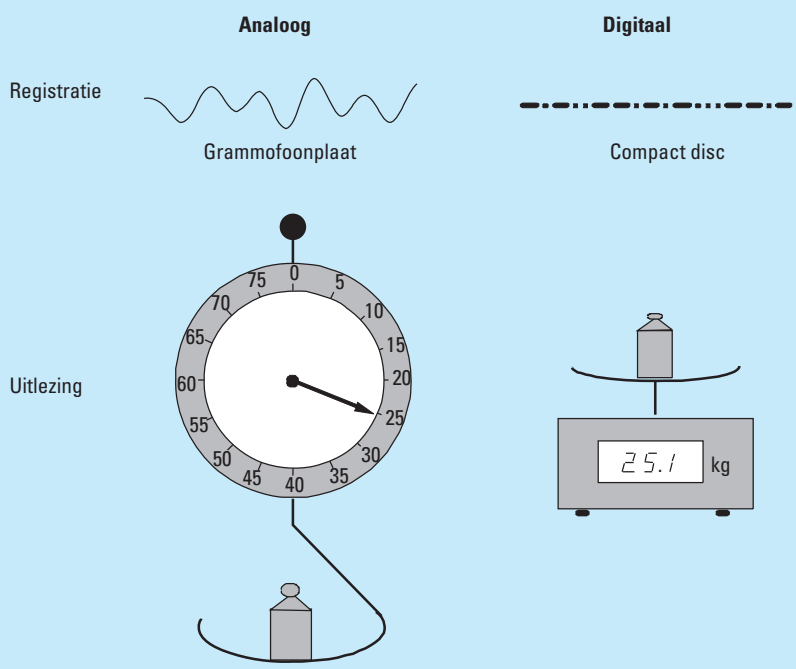
1.1 De voorstelling van gegevens in de computer

Tegenwoordig worden gegevens opgeslagen en verwerkt met behulp van computers. De te verwerken gegevens kunnen betrekking hebben op de letters van een tekst, op getallen, op meetwaarden, maar het kunnen evengoed foto's zijn of geluidsfragmenten. Al die gegevens nemen binnen de computer een of andere vorm aan: de stand van een schakelaar, de hoogte van een spanning of de sterkte van een stroom. Essentieel voor de computers waarmee we tegenwoordig werken, is dat elke van deze gebruikte vormen slechts in twee verschillende toestanden kan voorkomen: een schakelaar staat aan óf uit, een hoge spanning óf een lage, wel stroom óf geen stroom. Ook buiten de computer heeft informatie vaak deze vorm: een vakje op een enquêteformulier is wel aangestreept óf niet, een lamp is aan óf uit, enzovoort. Deze vorm van informatie noemen we *digitaal*.

1.1.1 Digitaal en analoog

Het Latijnse *digitus* betekent *wijsvinger*, zodat *digitaal* vertaald zou kunnen worden met *aanwijsbaar* of *telbaar*. Het daarvan afgeleide Engelse woord *digit* betekent dan ook *cijfer*. Als er maar twee verschillende toestanden mogelijk zijn, moeten we eigenlijk spreken van *binair digitaal* ofwel 'met twee telbare toestanden', maar bij een computer wordt het woord binair weggelaten en spreken we alleen van een digitale computer.
Het woord digitaal wordt vaak gebruikt als tegenstelling van *analoog*. Analoge (letterlijk *soortgelijke*) informatie gebruikt geen cijfers of codes, maar stelt een fysische grootheid (zoals druk of temperatuur) voor met behulp van een

Figuur 1.1 **Analoog en digitaal**



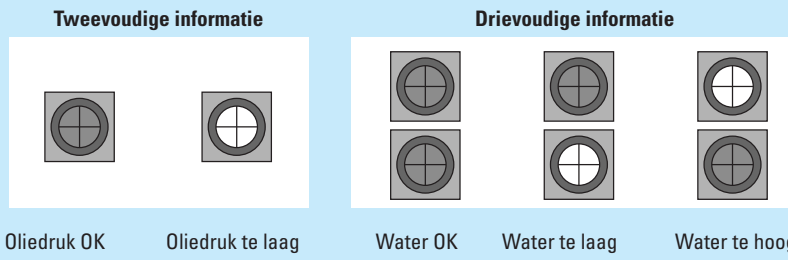
stroom of een spanning. Deze stromen of spanningen variëren op dezelfde manier als de grootheden die ze representeren. Een hogere temperatuur betekent dus een hogere spanning. Analoge instrumenten worden met wijzers uitgelezen, digitale instrumenten met cijfers. Een voorbeeld van analoge informatieopslag is de grammofoonplaat, de digitale tegenhanger is de compact disc (figuur 1.1).

Vroeger was er ook sprake van *analoge computers*. Hiermee kon met behulp van elektronische componenten een simulatie van een fysisch systeem worden gemaakt, zodat je kon voorspellen hoe zo'n systeem zich in werkelijkheid zou gedragen. Dit soort simulaties wordt tegenwoordig ook met digitale computers uitgevoerd.

1.1.2 Coderen

Een aantal gegevens laat zich gemakkelijk op een digitale manier vertalen. Als we bijvoorbeeld van een automotor willen weten of de oliedruk voldoende hoog is, hoeven we maar twee toestanden te onderscheiden: voldoende óf onvoldoende. Dit gegeven kan eenvoudig met een enkel lampje worden aangegeven. Maar soms willen we meer mogelijkheden kunnen onderscheiden. Bij een wasmachine zouden we bijvoorbeeld over de waterstand in de machine de informatie *te laag*, *goed* of *te hoog* willen hebben, een gegeven dat uit drie toestanden bestaat. Het ligt voor de hand een lampje voor *te laag* en een lampje voor *te hoog* te nemen. Beide lampjes uit betekent dan *goed*. We hebben daarmee de drievoudige informatie vertaald of *gecodeerd* in het aan of uit zijn van twee lampjes (figuur 1.2). Het interpreteren van de lampjes, het terugvertalen dus, noemen we *decoderen*.

Figuur 1.2 **Tweevoudige en drievoudige informatie**



Symbolen
Binnen de computer bestaan alleen signalen met twee verschillende toestanden: binaire signalen. Deze twee toestanden kunnen we met twee symbolen aangeven, zonder dat het nodig is om precies te weten hoe ze in werkelijkheid gerepresenteerd worden. Als symbolen worden vaak gebruikt:

H en *L* (*high* en *low*),
of *T* en *F* (*true* en *false*),
of *1* en *0* (de meest gebruikte vorm).

Bij de laatste voorstelling moet je je realiseren dat 1 en 0 geen getallen voorstellen, maar eenvoudig namen zijn voor twee verschillende toestanden.

Waarheidstabellen

Ons voorbeeld van de waterstand in de wasmachine zou er met de notatie *T* en *F* in tabelvorm zo uitzien:

Waterstand	Te laag	Te hoog
te laag	T	F
goed	F	F
te hoog	F	T

Een dergelijke tabel heet een *waarheidstabel*. Zo'n tabel geeft van alle mogelijke toestanden de binaire codering aan. In het voorbeeld komt één code (uiteraard) niet voor, namelijk 'te laag = True' en 'te hoog = True'. Deze code zou eventueel kunnen dienen om een vierde toestand aan te geven:

Waterstand	Te laag	Te hoog
apparaat stuk	T	T

In dit boek geven we de toestanden True en False in het algemeen aan met 1 en 0, zodat de complete waarheidstabel wordt:

Waterstand	Te laag	Te hoog
te laag	1	0
goed	0	0
te hoog	0	1
apparaat stuk	1	1

Bits en bytes

Een schakelaar is een voorbeeld van een binair systeem. Hij kan namelijk in twee toestanden verkeren: open of gesloten. Binnen een computer vinden we vele duizenden elektronische schakelingen die in twee posities kunnen verkeren, we kunnen er dus een 0 of een 1 in opslaan. Zo'n bewaarplaats wordt een *bit* genoemd. In de literatuur vinden we twee verschillende verklaringen voor deze naam: een afkorting van *binary digit* (2-tallig cijfer) of ook wel van *binary information ticket* (binair informatiekaartje).

We hebben al gezien dat we met twee bits vier verschillende combinaties kunnen maken. In het algemeen geldt dat we in n bits 2^n verschillende toestanden kunnen coderen:

4 bits geeft	2^4	= 16	combinaties (ga ze na!)
8 bits geeft	2^8	= 256	combinaties
10 bits geeft	2^{10}	= 1 024	combinaties
16 bits geeft	2^{16}	= 65 536	combinaties

enzovoort.

Vier bits noemen we een *nibble* en acht bits een *byte*.

Als we getallen willen coderen met behulp van een gegeven aantal bits, moet elk getal met een aparte combinatie worden aangegeven. In een nibble kunnen we 16 verschillende getallen opslaan, in een byte passen 256 getallen en in 16 binaire posities kan elk getal van 1 tot 65 536 (of van 0 tot 65 535) ondergebracht worden.

De opslagplaats van een zeker aantal bits noemen we een *register*. Een computer werkt in het algemeen met registers, dus met een aantal bits tegelijkertijd. Dit aantal wordt wel een *woord* genoemd. Met een *32-bits computer* bedoelen we dat deze met registers van 32 bits werkt. In een dergelijk register passen woorden van 32 bits, waarin getallen van 1 tot en met 2^{32} (ruim 4 miljard) gecodeerd kunnen worden.
Ga zelf na dat we, om willekeurige getallen tussen 0 en 1000 te kunnen opbergen, gebruik moeten maken van registers van ten minste 10 bits.
Tegenwoordig worden veel computers uitgerust met woorden van 64 bits. Reken ook eens uit hoe groot de getallen zijn die daarin kunnen worden opgeslagen.

1.2 Talstelsels

We zullen ons nu bezighouden met de codering van getallen in een computer. Om deze codering te kunnen begrijpen, zullen we eerst de gewone schrijfwijze voor getallen bestuderen. Die schrijfwijze blijkt samen te hangen met het getal 10, dat we het *grondtal* noemen. Daarom praten we over het *10-tallig talstelsel* of afgekort het *10-tallig stelsel*.

1.2.1 Het tientallig stelsel

Kenmerken van het 10-tallig stelsel zijn:
- er is een grondtal: 10;
- er zijn 10 symbolen: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9;
- als een getal groter is dan 9 wordt een nieuwe positie geopend waarvan de betekenis is: maal 10 (weegfactor);
- als een getal groter is dan $9 \times 10 + 9$, wordt weer een nieuwe positie geopend met weegfactor 10×10, enzovoort.

Zo is het getal 15 671 in het 10-tallig stelsel als volgt opgebouwd:

10^4	10^3	10^2	10	1			
1	5	6	7	1	… waarde	→	$1 \cdot 1$
							$7 \cdot 10$
							$6 \cdot 10^2$
							$5 \cdot 10^3$
							$1 \cdot 10^4$
							——— +
							15 671

De waarde van het cijfer hangt dus af van de plaats waar het staat. We noemen een dergelijk stelsel een *plaatsafhankelijk* stelsel.

1.2.2 Talstelsels met een ander grondtal

We kunnen de eigenschappen van ons decimale talstelsel overdragen naar soortgelijke stelsels met een ander grondtal. Hiervoor zullen we eerst de regels die we gevonden hebben voor het 10-tallig stelsel uitbreiden naar een stelsel met een willekeurig grondtal.

Regels voor een plaatsafhankelijk stelsel met grondtal n:
1 Er zijn n symbolen: 0, 1, 2, … , n –1
2 De posities geven de waarde aan, met van achter naar voren:
n^0, n^1, n^2, n^3, n^4, …

De wiskunde leert dat we n ook mogen schrijven als n^1 en dat n^0 het getal 1 voorstelt. Een ander woord voor grondtal is *radix*.

Voorbeeld in het octale stelsel

Het talstelsel met grondtal of radix = 8 noemen we het *octale stelsel*. Volgens de regels die we net hebben opgeschreven geldt hiervoor:

- De symbolen zijn: 0, 1, 2, 3, 4, 5, 6, 7.
- De posities betekenen: ... 8^4 8^3 8^2 8 1.

Dus:

1 5 6 7 1 (octaal):

$1 \cdot 1 = 1$
$7 \cdot 8 = 56$
$6 \cdot 8^2 = 384$
$5 \cdot 8^3 = 2\,560$
$1 \cdot 8^4 = 4\,096$
——— +
7 097 (decimaal!)

We noteren dit als: 8R 15 671 = 10R 7 097 (de R van radix).

Om de waarde van een getal te bepalen moeten we dus weten wat de radix is van het gebruikte talstelsel. Bij een radix van 8 wordt met 15 671 kennelijk iets anders bedoeld dan met dezelfde cijfers bij een radix van 10. Wanneer de mogelijkheid bestaat dat we een talstelsel met een ander grondtal dan 10 tegenkomen, moeten we dus altijd het grondtal bij het getal vermelden.

1.2.3 Omrekenen

We moeten nu nog een manier hebben om stelsels met verschillende grondtallen in elkaar om te rekenen. Om het ons gemakkelijk te maken rekenen we altijd via het decimale stelsel. We hebben dan twee rekenmethoden nodig:

1 Van een getal abcd met grondtal R naar het decimale stelsel. Hiervoor gebruiken we (zie het voorbeeld in de vorige paragraaf)
$a \times R^3 + b \times R^2 + c \times R + d$.
2 Van een decimaal getal naar een getal met grondtal n.

Voor de tweede rekenmethode bestaan verschillende manieren. We geven een voorbeeld: het omrekenen van 10 000 (decimaal) naar het 4-tallig stelsel.

Eerste manier: *op de manier van een staartdeling*
De waarden van de verschillende posities in het 4-tallige stelsel zijn:

...	...	4^7	4^6	4^5	4^4	4^3	4^2	4^1	4^0
...	...	16 384	4 096	1 012	256	64	16	4	1

16 384 is groter dan ons getal 10 000 en komt dus niet voor (heeft de waarde 0). De waarde 4 096 zit er twee keer in:

	10 000	
4 096	8 192	2×
	1 808	
1 024	1 024	1×
	784	
256	768	3×
	16	
64	0	0×
	16	
16	16	1×
	0	
4	0	0×
	0	
1	0	0×
	0	

Op deze manier vinden we: 10R 10 000 = 4R 2 130 100. Deze manier is gemakkelijk te begrijpen, maar wel omslachtig. De tweede manier gaat sneller.

Tweede manier: *herhaald delen en de resten noteren*

10 000	: 4	=	2 500	rest 0
	: 4	=	625	rest 0
	: 4	=	156	rest 1
	: 4	=	39	rest 0
	: 4	=	9	rest 3
	: 4	=	2	rest 1
	: 4	=	0	rest 2

De resten noteren we nu van beneden naar boven en we vinden:
4R 2 130 100.
Ga zelf na waarom deze methode werkt.

1.2.4 **Het tweetallig stelsel**

Voor een computer is vooral het 2-tallig stelsel van belang. Een bit van een computer kan namelijk de waarden 0 en 1 voorstellen en dit zijn juist voldoende symbolen voor het 2-tallig stelsel. Als we onze regels voor talstelsels op het 2-tallig stelsel loslaten, vinden we:

1 Er zijn twee symbolen: 0, 1
2 De waarden van de posities zijn: … 2^6 2^5 2^4 2^3 2^2 2^1 2^0
… 64 32 16 8 4 2 1

Voorbeeld met het tweetallig stelsel
Het getal 2R 11 1011 0101 betekent in het 10-tallig stelsel:

1	×	1	=	1
0	×	2	=	0
1	×	4	=	4
0	×	8	=	0
1	×	16	=	16
1	×	32	=	32
0	×	64	=	0
1	×	128	=	128
1	×	256	=	256
1	×	512	=	512
				+
				949

Omgekeerd kunnen we 750 decimaal (of 10R 750) als volgt naar het 2-tallig stelsel omrekenen:

750	: 2 =	375	rest 0
	: 2 =	187	rest 1
	: 2 =	93	rest 1
	: 2 =	46	rest 1
	: 2 =	23	rest 0
	: 2 =	11	rest 1
	: 2 =	5	rest 1
	: 2 =	2	rest 1
	: 2 =	1	rest 0
	: 2 =	0	rest 1

Als we de resten van onder naar boven noteren wordt het getal dus 2R 10 1110 1110.

Eigenschappen van het tweetallig stelsel
We zien aan de voorbeelden dat voor het 2-tallig stelsel geldt:
- Er zijn weinig symbolen, dus eenvoudige rekenregels.
- De getallen worden behoorlijk lang, dus voor een mens onoverzichtelijk.

Voor een computer bestaat het laatste probleem niet. Alle computers werken daarom op een of andere manier met het binaire talstelsel. Er worden wel verschillende manieren van coderen gebruikt, die we later nog zullen tegenkomen.

1.2.5 Het octale stelsel

Omdat in een binair stelsel de getallen onoverzichtelijk worden voor mensen, is het gemakkelijker in een afgeleid stelsel te schrijven. Hiervoor worden het 8-tallige of het *octale* stelsel en het 16-tallige of *hexadecimale* stelsel gebruikt. Van het octale stelsel hebben we in paragraaf 1.2.2 al een voorbeeld

gezien. Als we 10R 750 omrekenen naar octale en binaire vorm, vinden we (ga dit zelf na):

octaal	1	3	5	6
binair	1	011	101	110

Als we deze getallen vergelijken, zien we dat we de octale code in de binaire terug kunnen vinden door deze van achteraf in groepen van drie te verdelen en aan elke groep een octaal cijfer toe te kennen volgens de gewone binaire telling:

000 - 0	100 - 4
001 - 1	101 - 5
010 - 2	110 - 6
011 - 3	111 - 7

Nog een voorbeeld:

binair:	(00)1	010	111	010	110	111
octaal:	1	2	7	2	6	7

We kunnen nu zeggen: de 2-tallige code 1010111010110111 is dezelfde als de octale code 127 267. Voor een mens is de octale code veel overzichtelijker en met behulp van een lijstje is het octale getal gemakkelijk weer in de juiste binaire code om te zetten.

1.2.6 Het hexadecimale of zestientallig stelsel

Ook een stelsel met een grondtal groter dan 10 kan bestaan. Het enige probleem is dat we meer symbolen nodig hebben dan de ons bekende tien. Voor het zestientallig stelsel is de lijst daarom uitgebreid met A, B, C, D, E en F. Als we nu onze regels voor talstelsels toepassen op een 16-tallig stelsel vinden we:

1 Er zijn 16 symbolen: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E en F.

2 De weegfactoren zijn: 16^0, 16^1, 16^2, 16^3, 16^4, …
of: 1, 16, 256, 4 096, 65 536, …

Meestal worden voor de extra symbolen hoofdletters gebruikt, maar kleine letters kom je ook wel tegen. Enkele voorbeelden:

10R 13 = 16R D
10R 15 = 16R F
10R 170 = 16R AA (nl. $10 \times 16 + 10$)

Om het hexadecimale stelsel om te rekenen naar het binaire gebruiken we dezelfde methode als bij het octale stelsel, alleen zijn de bijbehorende groepen nu 4 bits groot.

0000	- 0	1000	- 8
0001	- 1	1001	- 9
0010	- 2	1010	- A
0011	- 3	1011	- B
0100	- 4	1100	- C
0101	- 5	1101	- D
0110	- 6	1110	- E
0111	- 7	1111	- F

Als we het getal uit ons vorige voorbeeld: 1010111010110111 om willen rekenen naar het zestientallig stelsel, verdelen we het in groepen van vier en vertalen we elke groep volgens voorgaand lijstje:

```
1010.1110.1011.0111   binair
   A    E    B    7    hexadecimaal
```

Omdat het hexadecimale stelsel meer symbolen kent, is het korter dan het octale:

16R AEB7 = 8R 127 267

Voor de beginner is de verschijning van letters in getallen wat raar. Laat het een troost zijn dat we er vrijwel nooit mee rekenen. We gebruiken het stelsel alleen om lange binaire getallen op een voor ons overzichtelijke wijze te beschrijven. Het getal AEB7 is voor ons immers gemakkelijker te onthouden dan 1010111010110111.
Met het gegeven lijstje is de omzetting van binair naar hexadecimaal eenvoudig. Aangezien de meeste computers werken met binaire getallen in veelvouden van vier bits is het hexadecimale stelsel handiger om de voorkomende binaire getallen weer te geven dan het octale stelsel. Tegenwoordig wordt dan ook meestal het hexadecimale stelsel gebruikt.

1.3 Rekenen met binaire getallen

Wanneer we decimale getallen vertalen naar een binaire voorstelling, kunnen we ze in een computer opslaan, die er dan mee kan gaan rekenen. In deze paragraaf bekijken we hoe het optellen en vermenigvuldigen met positieve getallen verloopt.

1.3.1 Getallen in de computer

In principe zijn er twee manieren om decimale getallen in een digitaal werkende computer op te slaan.

1 BCD-code
We kunnen de decimale tekens stuk voor stuk vertalen in een binaire code volgens het lijstje van paragraaf 1.2.6. In elke nibble komt dan een enkel decimaal cijfer te staan. Deze manier noemen we *BCD-code* (Binary Coded Decimals). De computer werkt dan in feite decimaal en kent ook decimale (dus *niet*-binaire) rekenregels.

Zo is 10R 7914 = 0111 1001 0001 0100 in BCD-code. Het voordeel van deze manier is dat getallen gemakkelijk worden vertaald, zowel bij in- als bij uitvoer.
Er zijn echter ook nadelen aan deze voorstelling: de getallen nemen meer ruimte in dan bij een echte binaire notatie (waarom?) en een computer rekent minder gemakkelijk en minder snel dan met echte binaire getallen.

2 Binaire getallen
Het decimale getal wordt eerst vertaald naar een echt binair getal. Binnen de computer bestaan de getallen alleen in binaire vorm. Hiermee wordt ook gerekend. Alleen als uitvoer noodzakelijk is, wordt een conversie naar decimale getallen gemaakt. Ook voor echte binaire getallen bestaan nog keuzemogelijkheden. Deze zijn voornamelijk afhankelijk van de nauwkeurigheid en de representatie van negatieve getallen. In het volgende hoofdstuk gaan we deze bekijken.

Bij wetenschappelijke programma's is er doorgaans veel rekenwerk en weinig in- en uitvoer, zodat binaire getallen worden gekozen. Bij weinig rekenen en veel in- en uitvoer, zoals bij administratieve programma's, viel de keuze vroeger vaak op BCD-representatie. De huidige computers zijn echter erg snel en het omrekenen is nauwelijks tijdrovend. Men kiest daarom op het ogenblik meestal voor binaire getallen.

MSB en LSB
Een getal wordt in een computer opgeslagen in een register. Wanneer dit register op papier getekend wordt, is niet onmiddellijk duidelijk hoe het getal in het register terechtkomt: van links naar rechts of omgekeerd. Daarom wordt bij de tekening van een register met de letters MSB en LSB aangegeven waar het eerste bit (*Most Significant Bit*) en waar het laatste bit (*Least Significant Bit*) van het getal staat.

1.3.2 Optellen met binaire getallen

Optellen van binaire getallen is in principe zeer eenvoudig. De enige rekenregels zijn:

$$\begin{aligned} 0 + 0 &= 0 \\ 0 + 1 &= 1 \\ 1 + 0 &= 1 \\ 1 + 1 &= 1\,0 \end{aligned}$$

De laatste optelling kunnen we ook beschrijven met: 1 + 1 = 0 *met 1 onthouden;* de te onthouden 1 noemen we de *carry* of *overdracht*.

Laten we nu eens twee getallen binair optellen. We kiezen de decimale getallen 25 en 9 en rekenen deze eerst om naar het binaire stelsel. We schrijven dan de binaire voorstellingen onder elkaar en tellen, net als in het 10-tallig stelsel, van achter naar voren op (boven de binaire representie is de overdracht aangegeven):

```
          1  1        1          … overdracht …
25   =       1  1  0  0  1
 9   =          1  0  0  1
—+       ———————————————— +
34        1  0  0  0  1  0
```

Bij het optellen van twee binaire getallen moeten we ook eventuele overdrachten meerekenen, zodat de meest ingewikkelde rekensom wordt:

1 + 1 + 1 = 1 1

Dit heeft plaats als op de betreffende positie 1 bij 1 moet worden opgeteld en er een overdracht van de vorige positie was.

Als veel getallen tegelijk opgeteld moeten worden, dan wordt het lastig:

```
 7           0  1  1  1
 7           0  1  1  1
 6           0  1  1  0
13           1  1  0  1
——  +  ———————————————— +
33     1  0  0  0  0  1
```

De rekenregels voor zo'n optelling worden ingewikkeld doordat er overdracht is, niet alleen naar de volgende positie maar ook naar de positie die daarop volgt. Computers tellen dan ook nooit meer dan twee getallen tegelijk op. Het gegeven voorbeeld wordt binnen de computer in stappen uitgewerkt:

```
eerst   7  +   7  =  14
dan    14  +   6  =  20
dan    20  +  13  =  33
```

Reken dit zelf binair na.

1.3.3 Vermenigvuldigen met binaire getallen

Voor het vermenigvuldigen van binaire getallen zijn alle 'tafels' die de computer hoeft te kennen:

```
0 × 0  =  0
0 × 1  =  0
1 × 0  =  0
1 × 1  =  1
```

Heel wat eenvoudiger dus dan in het 10-tallig stelsel!

Laten we een wat uitgebreidere vermenigvuldiging maken. We passen dezelfde methode toe die we bij decimale getallen gebruiken. We schrijven de getallen onder elkaar, vermenigvuldigen met behulp van de tafels van achter naar voren en tellen de tussenresultaten op:

```
B    25                    1  1  0  0  1
A    13                    1  1  0  1
     ——×                   ————————    ×
     75                 1  1  0  0  1
     25 .            0  0  0  0  0  .
     ——+          1  1  0  0  1  .  .
     325       1  1  0  0  1  .  .  .
               ————————————————————    +
               1  0  1  0  0  0  1  0  1  resultaat
```

Als we het optellen nu steeds onmiddellijk doen wanneer het resultaat van een deelvermenigvuldiging ter beschikking is, luidt het recept voor het vermenigvuldigen van de getallen A en B als volgt:

a Zet het resultaat op nul.
b Herhaal:
- kijk naar de meest rechtse positie van A;
- tel B bij het resultaat als in A een 1 staat;
- schuif B een positie naar links;
- verwijder de meest rechtse positie van A, totdat A leeg is.

Vermenigvuldigen op deze manier komt dus neer op schuiven en optellen. Zoals we in hoofdstuk 2 zullen zien, zijn dit voor een computer eenvoudige bewerkingen.

De getallen die we tot dusver hebben gebruikt waren positieve gehele getallen. We noemen deze *unsigned integers*. In n bits kunnen we alle unsigned integers opnemen van 0 tot en met $2^n - 1$.

1.4 Negatieve getallen

Als we gaan aftrekken en delen is het van belang dat we ook met negatieve getallen kunnen werken. Er zijn verschillende mogelijkheden om deze te coderen. De meest gebruikte manier is *two's complement*. In deze paragraaf gaan we na hoe de computer hiermee kan optellen en aftrekken.

1.4.1 Codering van negatieve getallen

Er zijn verschillende mogelijkheden voor de codering van negatieve getallen. We bekijken drie manieren: *signed magnitude*, *one's complement* en *two's complement*.

Signed magnitude

Signed magnitude (teken + grootte) is de meest voor de hand liggende methode. In het eerste bit staat het teken en in de rest van de bits het getal. Het is gebruikelijk om een 0 te nemen als het teken positief is en een 1 als het negatief is. Als voorbeeld nemen we 8 bits (1 byte). Hierin kunnen we op deze manier getallen opslaan tussen –127 en +127:

```
–127  =  1 1 1 1 1 1 1 1
+127  =  0 1 1 1 1 1 1 1
```

op deze manier is:

```
+99 =  0 1 1 0 0 0 1 1
–99 =  1 1 1 0 0 0 1 1
```

Een nadeel van deze methode is dat er twee mogelijkheden zijn voor het getal 0, namelijk +0 = 0000 0000 en –0 = 1000 0000. Ook moet de computer eerst het teken van een getal testen voordat een berekening uitgevoerd kan worden.

One's complement

Bij *one's complement*-codering worden alle bits geïnverteerd. Een voorbeeld in 8 bits:

```
+99 =  0 1 1 0 0 0 1 1
–99 =  1 0 0 1 1 1 0 0
```

Bij deze methode wordt het eerste bit ook een tekenbit. Het blijkt dat een computer er gemakkelijker mee kan rekenen dan met signed magnitude code. Het nadeel van twee voorstellingen voor het getal nul (in dit geval 0000 0000 en 1111 1111) blijft echter bestaan.

Two's complement

De *two's complement*-methode wordt tegenwoordig het meest gebruikt. Net als bij de one's complement-codering worden alle bits geïnverteerd, maar vervolgens wordt er 1 bij geteld.

```
Het grootste getal in 8 bits is   +127 =  0 1 1 1 1 1 1 1
Het kleinste getal in 8 bits is   –128 =  1 0 0 0 0 0 0 0
                                  + 99 =  0 1 1 0 0 0 1 1
                                  – 99 =  1 0 0 1 1 1 0 1
                                  +  0 =  0 0 0 0 0 0 0 0
                                  –  0 =  0 0 0 0 0 0 0 0
```

Ook hier blijft het eerste bit een tekenbit. Het voordeel van deze notatie is dat er slechts één representatie van het getal 0 bestaat. Een klein nadeel is dat het bereik (van –128 tot +127 bij 8 bits) asymmetrisch is. Hoewel deze methode op het eerste gezicht ingewikkeld lijkt, is dit voor een computer de meest eenvoudige vorm: er hoeft bij berekeningen namelijk geen verschil gemaakt te worden tussen positieve en negatieve getallen.

1.4.2 Rekenen met getallen in de two's complement-notatie

Bij two's complement-getallen moeten we van tevoren afspreken hoe groot de registers zijn waarmee we werken. Computers werken op het ogenblik meestal met registers van 32 of 64 bits, maar om het niet al te ingewikkeld te maken nemen we voor al onze voorbeelden 8 bits. Hiervoor geldt het volgende:

```
Het grootste getal is   0  1  1  1  1  1  1  1   (127)
Het kleinste getal is   1  0  0  0  0  0  0  0   (–128)
```

Het eerste bit is tevens tekenbit. We kunnen dus onmiddellijk zien of we met een negatief getal te doen hebben. Het optellen van twee positieve getallen werkt net als bij unsigned integers, als we maar oppassen dat we de bovengrens niet passeren, dus dat de som kleiner blijft dan 127.

We tellen nu een positief getal (42) bij een negatief getal (–13) op.

```
                          + 42 =    0 0 1 0 1 0 1 0
+ 13 = 0000 1101 ⇒        – 13 =    1 1 1 1 0 0 1 1   (bits inverteren
                                                       en 1 bijtellen)
                                  ———————————————— +
                            29 =  1 0 0 0 1 1 1 0 1
```

De eerste 1 past echter niet meer in het register. Dit blijkt een voordeel: als we die gewoon weglaten, ontstaat 0001 1101 ofwel +29, het juiste antwoord. Door deze eigenschap gaat het optellen van een negatief en een positief getal precies zo als het optellen van twee positieve getallen. De computer hoeft niet van tevoren te weten of één van de twee negatief is. Hiermee is meteen een methode gevonden om twee getallen van elkaar af te trekken: neem van het tweede getal het two's complement en tel dan op.

We tellen nu twee negatieve getallen op.

```
–13    1 1 1 1 0 0 1 1
–42    1 1 0 1 0 1 1 0
     ———————————————— +
     1 1 1 0 0 1 0 0 1  =  –55    (reken na!)
```

Ook hier geldt dat de optelling precies zo verloopt als bij twee positieve getallen, als we tenminste de 1 die niet meer in het register past gewoon weglaten.

1.4.3 **Overflow**

Het is mogelijk dat de som van twee getallen te groot of te klein wordt om in het register te passen:

```
+99 = 0 1 1 0 0 0 1 1
+99 = 0 1 1 0 0 0 1 1
     ———————————————— +
     1 1 0 0 0 1 1 0  =  –58!
```

```
–99 = 1 0 0 1 1 1 0 1
–99 = 1 0 0 1 1 1 0 1
     ———————————————— +
     0 0 1 1 1 0 1 0  =  +58!
```

Dit heet *overflow*. Hoewel het resultaat niet eens uit het register hoeft te lopen, is toch een fout ontstaan. Overflow treedt op als de som van twee negatieve getallen positief wordt of de som van twee positieve getallen negatief. Deze situatie moet de computer dus kunnen signaleren.

1.4.4 Schuiven

In het algemeen geldt bij een plaatsafhankelijk stelsel dat schuiven het gewicht van de cijfers verandert. Als we in het 10-tallig stelsel alle cijfers één positie naar links schuiven, komt dit neer op vermenigvuldigen met 10, aangenomen dat we in de vrijgekomen rechterpositie een 0 schrijven. Op dezelfde manier kunnen we een binair getal met 2, 4, 8 … vermenigvuldigen door het 1, 2, 3 … plaatsen naar links te schuiven en de vrijkomende plaatsen rechts met nullen aan te vullen.
Naar rechts schuiven betekent in dit geval delen door 2 voor elke plaats dat het getal opschuift. Links moeten dan nullen binnengeschoven worden voor een positief getal en énen voor een negatief getal. We noemen dit *arithmetisch schuiven*. De meeste computers kennen hier speciale opdrachten voor. In hoofdstuk 3 gaan we daar verder op in.

1.5 Andere getallencodes

We beschrijven in het kort nog enkele andere manieren om getallen te coderen.

1.5.1 Packed decimals

Packed decimals is een BCD-code met één decimaal getal per nibble (4 bits). De laatste nibble bevat dan een code voor het teken, bijvoorbeeld 1100 voor een + en 1101 voor een –. De voor- en nadelen van een dergelijk systeem zijn al genoemd in paragraaf 1.3. Sommige computers kunnen rechtstreeks met deze getallen rekenen, andere vertalen deze notatie eerst naar een binaire voorstelling.

Een voorbeeld:

```
+265 =  0010.0110.0101.1100
           2    6    5    +

–265 =  0010.0110.0101.1101
           2    6    5    –
```

1.5.2 Unpacked decimals

Unpacked decimals is een BCD-code waarin elk byte één decimaal getal voorstelt. Elk cijfer wordt voorafgegaan door vier *zonebits*. In feite ontstaat nu een *tekencode* (zie paragraaf 1.6). Ook de + en de – worden gecodeerd zoals in de bijpassende karaktercode staat. Voorbeeld:

```
+265 =  0011 0010   0011 0110   0011 0101   0010 1011
           2             6             5             +
```

Vergelijk dit met de ASCII-tabel, zie tabel 1.1.

Dit formaat wordt alleen bij in- en uitvoer gebruikt. Om te rekenen worden eerst de zone-bits verwijderd en het getal daarna eventueel nog naar binaire code omgerekend. Bij uitvoer worden deze bewerkingen in omgekeerde volgorde uitgevoerd.

1.5.3 Floating point formaat

Vooral voor wetenschappelijk rekenwerk is de floating point representatie belangrijk. Bij een dergelijke voorstelling wordt een *mantisse* en een *exponent* opgegeven. Om te zien hoe het werkt, geven we eerst een voorbeeld van de floating point-voorstelling in het 10-tallig stelsel, zoals die ook op een wetenschappelijke calculator te vinden is:

teken	mantisse	exponent	
+	0,123	– 3	$= +0{,}123 \times 10^{-3} = +0{,}000123$
–	0,345	+ 7	$= -0{,}345 \times 10^{7} = -3\,450\,000$

Het voordeel van een dergelijke notatie is de constante nauwkeurigheid en de mogelijkheid zeer grote en zeer kleine getallen te coderen. De constante nauwkeurigheid wordt verkregen door de eis dat het eerste cijfer na de komma ongelijk aan 0 is. Wanneer het resultaat van een berekening anders uitvalt, wordt het resultaat opgeschoven tot de mantisse aan deze voorwaarde voldoet. Hierbij wordt de exponent met 1 vermeerderd of verminderd voor elke positie die naar rechts of naar links wordt geschoven.

Een binair floating point-formaat bestaat uit drie of vier velden voor teken(s), mantisse en exponent. Vroeger hadden de meeste computers een eigen floating point-formaat. Sinds enkele jaren is een standaardisatie vastgesteld door het IEEE (*Institute for Electrical and Electronics Engineers*) waar de meeste fabrikanten zich aan conformeren. Er zijn standaardisaties voor een voorstelling in 32 bits (*single precision*) en in 64 bits (*double precision*). De laatste heeft een grotere nauwkeurigheid en een groter bereik. In figuur 1.3 zien we het 32-bits formaat.

Figuur 1.3 **IEEE 32-bit floating-point formaat**

Teken : 0 is positief, 1 is negatief.

Exponent : loopt van –126 tot +127 (excess code).

Mantisse : is genormaliseerd tot het eerste bit een 1 is. Dit wordt niet genoteerd (dit heet een *hidden bit*). De complete mantisse is dus 24 bits groot.

Het grootste getal in dit formaat is $1{,}1111\ 1111\ 1111\ 1111\ 1111\ 111 \times 2^{127}$, dit is ongeveer $3{,}4 \times 10^{38}$.
De genoemde *excess notatie* voor de exponent houdt in dat bij de waarde van de exponent 127 wordt opgeteld. In dit formaat neemt de kleinst mogelijke exponent van –126 de waarde 0000 0001 aan en de grootst mogelijke van 127 de waarde 1111 1110.
De waarde 0 (precies) zou in dit formaat niet voor kunnen komen. Om dit toch mogelijk te maken wordt bij zeer kleine getallen (exponent 0000 0000) weer een ander formaat gehanteerd, waarbij afgestapt wordt van de verplichte 1 die bij de mantisse wordt gevoegd.

Het rekenen met floating point-getallen is lastiger en duurt langer dan met integers. Om twee van deze getallen op te tellen moeten de volgende stappen worden genomen:

a De exponenten van de twee getallen moeten worden gelijkgemaakt.
b De mantissen moeten worden opgeteld.
c Het resultaat moet zo nodig weer genormaliseerd worden, dat wil zeggen aangeschoven tot er een 1 voor de komma staat.

Sommige programmeertalen nemen als default aan dat de voorkomende getallen floating point-waarden zijn. In dat geval kan het de moeite waard zijn in het programma expliciet aan te geven wanneer met integers gerekend mag worden.

1.6 Tekencodes

Tekencodes worden gebruikt om letters en leestekens te coderen. In het Engels spreken we van *characters*. Om alle 26 letters van het alfabet te coderen zijn minimaal 5 bits nodig. Als we ook nog alle leestekens en cijfers willen coderen en bovendien onderscheid willen maken tussen hoofdletters en kleine letters, zijn ruim 90 verschillende codes nodig van minstens 7 bits. In paragraaf 1.6.1 wordt een dergelijke code behandeld.
Een computer werkt echter graag met veelvouden van 2. Vandaar dat voor een teken meestal 8 bits gebruikt worden. Dit is één byte per character en er zijn dan 256 codes mogelijk.
Een aantal codes wordt in dat geval waarschijnlijk niet gebruikt. Dit noemen we *redundant* (overtollig). Redundante codes kunnen, net zoals in het voorbeeld van de wasmachine in paragraaf 1.1.2, soms gebruikt worden om fouten te detecteren. In paragraaf 1.8 gaan we hier nader op in.

1.6.1 De ASCII- of USASCII-code

Er zijn verschillende tekencodes (charactercodes) in gebruik. Verreweg de meest gebruikte code is ASCII of USASCII. ASCII staat voor *American Standard Code for Information Interchange*. Het is een zevenbits code, maar meestal wordt een byte per teken gebruikt. Het achtste bit kan gebruikt worden voor foutendetectie (zie paragraaf 1.8), maar het wordt ook wel gebruikt om diverse 'vreemde' of grafische tekens te coderen. De code wordt zowel binnen de computer gebruikt als bij het transport van tekens naar randapparaten, zoals printers.

In tabel 1.1 zien we de ASCII-code. Hierbij staan horizontaal de eerste 3 bits uit en verticaal de laatste 4 bits. De hoofdletter J heeft dus volgens de tabel de hexadecimale code 4A, dit is binair 100 1010. De drie eerste bits (*de most significant bits*) worden wel *zonebits* genoemd.

De codes met zonebits 000 en 001 bevatten geen letters maar hebben een speciale betekenis. We noemen deze codes besturingstekens (*control characters*). Met het toetsenbord van de pc kunnen we ze oproepen door de CTRL-toets tegelijk met een letter in te toetsen. <CTRL>-A geeft een 'soh'-teken, <CTRL>-B een stx-teken enzovoort.

Tabel 1.1 **De ASCII-codetabel**

	0	1	2	3	4	5	6	7
0	nul	Dle	Sp	0	@	P	`	p
1	soh	Dc1	!	1	A	Q	a	q
2	stx	Dc2	"	2	B	R	b	r
3	etx	Dc3	#	3	C	S	c	s
4	eot	Dc4	$	4	D	T	d	t
5	enq	nak	%	5	E	U	e	u
6	ack	syn	&	6	F	V	f	v
7	bel	etb	'	7	G	W	g	w
8	bs	can	(	8	H	X	h	x
9	ht	em	)	9	I	Y	i	y
A	lf	sub	*	:	J	Z	j	z
B	vt	esc	+	;	K	[	k	{
C	ff	Fs	,	<	L	\	l	|
D	cr	gs	-	=	M	]	m	}
E	so	Rs	.	>	N	^	n	~
F	si	us	/	?	O	_	o	del

Het ESCAPE-teken zit onder de Esc-toets, het CR-teken (carriage return) zit onder de ENTER-toets. Verder zijn de meest gebruikte besturingstekens:

07 → bel	terminal geeft een piep
08 → bs	backspace
09 → ht	horizontale tab
0A → lf	linefeed
0C → ff	formfeed (nieuwe pagina)
0D → cr	carriage return (vooraan beginnen)

De meeste andere besturingstekens worden in de datacommunicatie gebruikt en hebben voor de pc minder betekenis.

20 → sp	spatie
7F → de	delete

Ga na dat bij de cijfers de laatste 4 bits juist de BCD-code vormen. Een binaire omzetting kunnen we dus gemakkelijk realiseren door de eerste 3 bits weg te laten.
Ga ook na dat hoofdletters en kleine letters slechts 1 bit schelen, een Caps lock-toets op een toetsenbord hoeft dus alleen dit bit te veranderen.
Als een computer tekst sorteert, doet hij dat in het algemeen volgens de inwendige character-code, waarbij hij de letters sorteert alsof het binaire getallen zijn.
Een andere tekencode die we een enkele keer nog tegenkomen, is EBDIC, *Extended BCD Interchange Code*. Dit is een 8-bits code die vroeger door IBM veel gebruikt werd. Als je de gelegenheid hebt om te programmeren, ga dan eens

na welke codering je meest geliefde computer erop na houdt door alle getallen tussen 32 en 127 als een teken af te drukken. In Pascal is hiervoor de CHR-functie en in BASIC de CHR$-functie bruikbaar. In C en Java kan een getal rechtstreeks als een teken worden afgedrukt.

1.7 Compressie en encryptie

Soms loont het de moeite om zo zuinig mogelijk te coderen. Hiermee kunnen we bij opslag aanzienlijk ruimte winnen en bij communicatie via telefoonlijnen worden de kosten minder naarmate minder bits getransporteerd hoeven te worden. Het is dan wel noodzakelijk gebruik te maken van speciale eigenschappen van het soort gegevens dat wordt overgestuurd.

1.7.1 Voorbeelden van datacompressie

We geven vier voorbeelden van datacompressie.

Voorbeeld 1
Een tekst in de Nederlandse taal zou kunnen worden overgestuurd door alle voorkomende woorden van de taal te nummeren. Voor het gemak nemen we aan dat er niet meer dan 65 536 woorden bestaan. Als dat het geval is, hoeven alleen maar de nummers van de woorden te worden overgeseind: per woord kost dit 16 bits. De ontvanger moet wel beschikken over precies hetzelfde woordenboek om de woorden weer terug te kunnen vinden! De winst ten opzichte van gewone tekst kan een factor 3 of meer bedragen.

Voorbeeld 2
Een fax verstuurt informatie over de manier waarop het papier gevuld is. Daartoe wordt een A4'tje verdeeld in meer dan een miljoen puntjes en voor elk puntje wordt doorgestuurd of het zwart (1) of wit (0) is. Bij elkaar is dit zo'n 150 kbyte informatie en om dit over een telefoonlijn te sturen kost nogal wat tijd en geld. Nu blijkt dat bij het gemiddelde faxbericht grote velden enen en nullen achter elkaar komen. Daarom wordt als volgt gecodeerd: 'eerst 126 enen, dan 259 nullen, dan 4 enen, dan 52 nullen' enzovoort.

Voorbeeld 3
Een code die vroeger in gebruik was voor het telexverkeer (de Telexcode), is een code met slechts 5 bits (tabel 1.2). Het is niet mogelijk hiermee alle letters, cijfers en leestekens te coderen. Daarom maakt deze code gebruik van *wisselkarakters*. Deze karakters kunnen wisselen tussen twee karaktersets, *letters* en *cijfers en leestekens*.

Na een wisselkarakter is er van karakterset gewisseld. Dit heeft natuurlijk alleen zin als niet te veel veranderingen van mode plaatsvinden, anders gaat alle tijd verloren met het uitzenden van wisselkarakters.

Voorbeeld 4
De Morse-code, die al meer dan een eeuw in gebruik is, vertaalt letters en cijfers in punten en strepen. Voor cijfers en letters die het meest voorkomen, zijn de kortste codes genomen. Zo is een E alleen een punt, een I twee punten, een U twee punten en een streep en een Q bestaat uit drie strepen en een punt.

Tabel 1.2 **Telexcode**

Alpha	Digits	Code	Alpha	Digits	Code
A	-	11000	N	,	00110
B	?	10011	O	9	00011
C	:	01110	P	0	01101
D	who?	10010	Q	1	11101
E	3	10000	R	4	01010
F		10110	S	'	10100
G		01011	T	5	00001
H		00101	U	7	11100
I	8	01100	V	=	01111
J	bel	11010	W	2	11001
K	(	11110	X	/	10111
L	)	01001	Y	6	10101
M	.	00111	Z	+	10001

Extra characters		
return	00010	
linefeed	01000	
alpha	11111	(wisselteken)
digits	11011	(wisselteken)
space	00100	
position	00000	

Bij gecomprimeerd opslaan van gegevens wordt tegenwoordig wel de Huffman-code gebruikt die ook iets dergelijks doet. Veel voorkomende letters worden korter gecodeerd dan de minder courante. Dit leidt inderdaad tot een compacte code, maar omdat de scheiding tussen de letters dan niet meer duidelijk is, moet de computer behoorlijk rekenen om de Huffman-code in ASCII te kunnen vertalen, en vice versa.
Soortgelijke coderingen worden gebruikt voor de ZIP- en ARJ-bestanden die in de pc voorkomen. Op deze manier kunnen bestanden zo zuinig mogelijk worden opgeslagen en zo snel en goedkoop mogelijk verstuurd. Er horen programma's bij die bestanden met deze codering weer in 'gewone' code omzetten. Ook voor de opslag op de harde schijf worden compressieprogramma's gebruikt. De huidige processoren zijn snel genoeg om compressie en decompressie 'on the fly' uit te voeren.

1.7.2 Kwaliteitsverlies

Bij de voorbeelden in paragraaf 1.7.1 kan de data na compressie weer exact gereconstrueerd worden.
Voor sommige soorten gegevens is dat niet strikt nodig. Dit geldt bijvoorbeeld voor geluid of (tv-)beelden die erg veel ruimte en dus verzendtijd in beslag nemen. Had het plaatje voor de fax in het tweede voorbeeld al 150 kbyte nodig, een scherpe kleurenfoto kan nog wel 100 maal meer ruimte in beslag nemen. Om deze bestanden zo ver mogelijk te comprimeren worden speciale methoden toegepast die de hoeveelheid data soms met een factor 100 weten te reduceren. Echter, deze grote compressie wordt alleen bereikt door een aantal gegevens die van minder belang zijn weg te laten.

Zo laat het DCC-audioregistratiesysteem zachte geluiden eenvoudig weg wanneer het totale geluidsvolume boven een bepaalde grens komt. Het argument is dat ons oor deze geluiden toch niet kan horen. Bij het comprimeren van videobeelden gaat het nog veel verder en wordt ook kwaliteitsverlies voor lief genomen die wel degelijk zichtbaar is.
Een van de coderingsstandaarden voor video is MPEG2. Deze wordt onder andere gebruikt bij de dvd (zie hoofdstuk 4). De compressie is erg hoog en het kwaliteitsverlies gering, maar het decoderen kost zo veel werk dat de processor het soms niet aankan en er speciale hardware nodig is. Binnen MPEG2 worden ook geluiden digitaal gecomprimeerd en opgeslagen volgens het bekende mp3-formaat (MPEG Audio Layer 3), dat een zelfstandig leven is gaan leiden. Met dit formaat kan bij nauwelijks waarneembare vervorming van het geluid een factor 12 winst worden geboekt.
Kan voor audio en video worden toegelaten dat de data enigszins vervormd wordt, het zal duidelijk zijn dat deze manier van comprimeren niet toepasbaar is bij de meeste andere gegevens.

1.7.3 Encryptie

Om te voorkomen dat verstuurde gegevens door derden misbruikt worden, kan het nodig zijn data te beveiligen. We doen dit met behulp van *encryptie* (versleuteling). Een primitief voorbeeld van versleuteling is het eerste voorbeeld van paragraaf 1.7.1: we moeten beschikken over het juiste woordenboek om de gecomprimeerde code te kunnen ontcijferen; het woordenboek is dus een sleutel. Inmiddels zijn vele vormen van encryptie gemeengoed geworden. Zo moet soms betaald worden om een bepaald tv-programma te zien. Als de betaling is geregeld, ontvangt de decoder in het toestel automatisch een sleutel waarmee het digitaal gecodeerde tv-signaal ontcijferd kan worden. Omdat gegevens die over internet worden verstuurd over vele min of meer publieke wegen gaan, wordt daar ook in toenemende mate versleuteling toegepast.

Aan de voorbeelden kunnen we zien dat, afhankelijk van het doel, allerlei vormen van codering, compressie en versleuteling toegepast kunnen worden. Het vakgebied dat zich bezighoudt met de theorie en de praktijk van al deze codes heet *informatietheorie*. Met deze theorie wordt voor allerlei toepassingen naar de meest optimale manier van coderen gezocht.

1.8 Foutendetectie en correctie

Om verschillende redenen kunnen fouten optreden bij de overdracht en opslag van gegevens. Atmosferische storingen (bliksem) en elektrische apparaten kunnen stoorpulsen leveren, maar ook zijn er soms fouten in de magnetische laag van banden of disks en soms zelfs defecten in de elektrische circuits. Al deze fouten hebben tot gevolg dat een 0 verandert in een 1 of omgekeerd. Het zal duidelijk zijn dat dit desastreuze gevolgen kan hebben, speciaal voor computers die real time werken en processen besturen, zoals een chemische fabriek of een vliegtuig. Een van de manieren om fouten te bestrijden bestaat uit een speciale vorm van codering. We onderscheiden *error detecting*- en *error correcting*-code.

1.8.1 Error detecting-code

Met een error detecting-code kan de computer detecteren (ontdekken) dat er iets mis is met een teken of een groep van tekens. Een van de meest gebruikte manieren is aan een bepaalde groep bits een extra bit toe te voegen, het *pariteitsbit*. Dit bit wordt zo gekozen dat het totaal aantal enen in de code altijd even is (of naar keuze oneven).

Bijvoorbeeld: aan elke groep van 8 bits wordt een pariteitsbit toegevoegd, zodanig dat het totale aantal enen even is (*even pariteit*).

code	pariteitsbit	totaal
0110 1000	1	0 1101 0001
1001 0011	0	1 0010 0110
1000 0000	1	1 0000 0001

Als in een teken nu een fout optreedt, klopt de pariteit niet meer en kan de computer maatregelen nemen. Zijn er twee fouten in één teken, dan vindt de computer niets, dus moeten we ervan uitgaan dat een fout zeldzaam is. Als bijvoorbeeld 1 × per 100 000 tekens een fout optreedt, dan is de kans op twee fouten in één teken slechts 1 op 10 000 000 000. Als de computer erin slaagt om bij foutconstatering een fatsoenlijke recovery-procedure te vinden, dan is de betrouwbaarheid dus zeer toegenomen.
De originele IBM Personal Computer sloeg in zijn geheugen alle gegevens op in groepen van acht bits (bytes). Per byte was een negende bit toegevoegd en op elk moment dat een byte gelezen werd (zo'n 1 000 000 maal per seconde), werd gecontroleerd of de pariteit klopte. Bij een fout werd het programma onderbroken en werd een error handling-routine gestart. Dit kwam gelukkig zelden voor, zodat later in de meeste versies van de pc het negende bit is weggelaten. Bij datacommunicatie, dat wil zeggen de overdracht van gegevens tussen verschillende apparaten over langere verbindingen zoals die bij een telefoon, treden aanzienlijk meer fouten op. Dit wordt geconstateerd door een aantal tekens tot een groep te verzamelen en dan per groep een aantal controlegetallen mee te sturen, zodat de ontvangende computer kan zien of zich fouten hebben voorgedaan. Is dit het geval, dan kan de ontvanger aan de zender vragen de hele groep opnieuw over te sturen.

CRC

Het meest gebruikt in de datacommunicatie is de CRC-methode (Cyclic Redundancy Check). In dat geval wordt elk datapakket beschouwd als één aaneengesloten getal. Dit getal wordt gedeeld door een bepaalde afgesproken waarde en de rest van de berekening wordt CRC genoemd. De zender stuurt de CRC met het pakket mee zodat de ontvanger kan controleren of deling dezelfde uitkomst geeft. Wanneer dat het geval is wordt de data beschouwd als onbeschadigd. Deze methode wordt verder besproken in paragraaf 8.4.1.

1.8.2 Error correcting-codes

Error correcting-codes kunnen niet alleen fouten detecteren, maar ook corrigeren. Het aantal toegevoegde bits is aanzienlijk groter dan bij de codes uit de vorige paragraaf en bij de constatering van een fout moet de computer enig rekenwerk verrichten om de data te kunnen herstellen. Als het aantal fouten in een teken of een groep van tekens toeneemt, is de fout helemaal

niet meer te herstellen, maar vaak wel nog te constateren. Wij volstaan hier met een enkel voorbeeld; in het hoofdstuk over datacommunicatie worden nog enkele andere detectie- en correctiemethoden besproken.
Het woord 'computer' wordt overgestuurd in 7-bits ASCII-code met even pariteitsbits. Na deze groep van acht bytes wordt een controlecode overgestuurd. Deze bestaat uit een byte dat de zogenoemde *longitudinale pariteitsbits* bevat (figuur 1.4).

Figuur 1.4 **Error-correcting code**

letter	code	pariteit	
c	1 1 0 0 0 1 1	0	
o	1 1 0 1 1 1 1	0	
m	1 1 0 1 1 0 (1)	1	←
p	1 1 1 0 0 0 0	1	
u	1 1 1 0 1 0 1	1	
t	1 1 1 0 1 0 0	0	
e	1 1 0 0 1 0 1	0	
r	1 1 1 0 0 1 0	0	
	↑		
controlebyte	0 0 0 0 1 1 1	1	

Stel dat in figuur 1.4 het omcirkelde bit fout is, dan wordt dit geconstateerd op de twee aangegeven plaatsen. Het foutieve bit is hiermee gelokaliseerd en kan veranderd worden.
Wanneer meerdere fouten binnen één pakket voorkomen, is de code niet meer te corrigeren. Het is in het voorbeeld van figuur 1.4 niet moeilijk een foutcombinatie te vinden die zelfs helemaal niet gedetecteerd wordt. Error correcting-codes worden daarom alleen gebruikt als de kans op fouten gering is. In de datacommunicatie is de foutkans erg groot en komen we dit soort codes dan ook zelden tegen. Binnen de computer is de foutkans veel kleiner, zodat ze daar wel toegepast worden. Zo slaat de IBM RS6000 computer alle 32-bits gegevens (woorden) op in 40-bits brede registers. Aan elk woord worden dus 8 bits toegevoegd waarmee enkele fouten gecorrigeerd en dubbele fouten gedetecteerd kunnen worden.

Opgaven

1.1 Geef de symbolen en de positie-betekenis van het 5-tallig stelsel.

1.2 Bepaal 5R 1 234 in het 10-tallig stelsel.

1.3 10R 1 234 = 5R …

1.4 Ken je een voorbeeld van een 12-tallig stelsel? En van een 60-tallig?

1.5 Ga na in hoeverre Romeinse cijfers voldoen aan de voorwaarde voor een 'net' talstelsel.

1.6 Breng onder woorden wat de betekenis is van de komma in het 10-tallig stelsel. Breid dit vervolgens uit tot de betekenis van de komma in een willekeurig plaatsafhankelijk stelsel.

1.7 5R 12,34 = 10R …

1.8 2R 0,1111 = 10R …

1.9 10R 999 = 8R … = 2R …

1.10 16R AB3D = 10R …

1.11 10R 10 000 = 16R …

1.12 Geef alle 16 mogelijke combinaties in een register van 4 bits en vertel wat ze voorstellen:

- **a** als unsigned integer;
- **b** als signed-magnitude getal;
- **c** als one's complement-getal;
- **d** als two's complement-getal.

1.13 Voer uit in 4 bits in two's complement-code:

- 3 + 4
- 2 – 5
- 5 – 2
- 6 + 3

1.14 Voer uit in 8 bits in two's complement-code:

- 22 + 57
- 90 – 40
- 90 – 80

1.15 Bereken het grootste en kleinste getal dat een 32-bits computer kan bevatten in two's complement-code (gebruik een rekenapparaat). Doe hetzelfde voor een 64-bits computer.

1.16 Het werken met een complementsnotatie is ook voor stelsels met andere grondtallen mogelijk. In het 10-tallig stelsel kunnen we inzien hoe dit gaat met de volgende opgave.
Een automobiel heeft een doorlopende kilometerteller van 3 cijfers. De teller staat op 000. De auto kan voor- en achteruit rijden en de teller loopt dan mee (als de auto begint met een kilometer achteruit te rijden, staat de teller op 999; we vertalen dit met *–1 kilometer*.
Overflow is niet toegestaan: dan houdt de weg op!

- **a** Wat is het bereik (grootste en kleinste afstand) van de auto?
- **b** Hoe kunnen we op de teller zien dat we met een negatieve afstand te maken hebben?
- **c** Geef de representatie van het getal –45 (auto 45 km achteruit laten rijden vanaf de 0-streep).
- **d** Formuleer een regel voor het omzetten van een positief getal in een negatief getal en controleer of de regel ook opgaat bij het omzetten van een negatief getal naar een positief getal.
- **e** Formuleer regels voor optellen en aftrekken in het genoemde ten's complement-stelsel, op dezelfde manier als dat in paragraaf 1.8 voor een two's complement-getal is gebeurd.
- **f** Als de auto nu uitgerust wordt met een teller met 5 cijfers, geef dan een regel hoe we bestaande waarden van 3 cijfers vertalen naar 5 cijfers.

1.17 In paragraaf 1.3.2 is een recept (*algoritme*) voor een binaire vermenigvuldiging gegeven. Probeer zelf een dergelijk algoritme te vinden voor het uitvoeren van een binaire deling. Hint: ga eerst precies na hoe een deling in het 10-tallig stelsel wordt uitgevoerd en vertaal deze regels naar een 2-tallig stelsel.

1.18 Voor de liefhebber van programmeren.

- **a** Schrijf een programma waarin gevraagd wordt om een binair getal. Het programma vertaalt dat vervolgens naar een decimale waarde.
- **b** Als a, maar dan van decimaal naar binair.
- **c** Als b, maar nu worden tevens de octale en hexadecimale waarden opgegeven.
- **d** Nu worden twee binaire getallen gevraagd. Som, verschil, product en quotiënt worden binair afgedrukt.

1.19 Hoeveel verschillende combinaties zijn mogelijk met 8 bits?

1.20 Een computer werkt met registers van 12 bits. Geef aan wat het grootste en het kleinste getal is dat hierin opgeslagen kan worden bij de volgende wijzen van codering:

- **a** unsigned;
- **b** one's complement;
- **c** two's complement;
- **d** BCD unsigned;
- **e** BCD signed (packed).

1.21 16R ABCD = 10R …

1.22 10R 3 000 = 7R …

1.23 10R 3 000 = 16R … = 8R … = 2R …

1.24 Bepaal hoeveel bits minimaal nodig zijn als 10R 9 312 vertaald moet worden naar:

- **a** unsigned binaire code;
- **b** unsigned BCD-code;
- **c** two's complement-code.

1.25 Bepaal de two's complement-voorstelling van –60. Neem hierbij een registergrootte dat dit getal juist kan bevatten.

1.26 Vertaal 10R 123 en 10R –75 naar 8 bit two's complement-code en voer uit: 123 – 75. Is hier sprake van carry? En van overflow?

1.27 Vergelijk de eigenschappen (symmetrie, rekenen) van one's complement-code met die van two's complement-code.

1.28 Vertaal 10R 37 en 10R 56 naar (unsigned) binaire code en voer de vermenigvuldiging 37×56 binair uit.

1.29 Vergelijk de eigenschappen van binaire en BCD-codering wat betreft de benodigde hoeveelheid bits, de interne en externe voorstelling en de manier van rekenen. Welke voorstelling leent zich het best voor wetenschappelijk rekenwerk? Waarom?

1.30 Een code voor getallen die in de techniek wel gebruikt wordt, is de *Gray-code*. Voor 4-bits getallen geldt de volgende code:

0 - 0000	8 - 1100
1 - 0001	9 - 1101
2 - 0011	10 - 1111
3 - 0010	11 - 1110
4 - 0110	12 - 1010
5 - 0111	13 - 1011
6 - 0101	14 - 1001
7 - 0100	15 - 1000

De Gray-code wordt gebruikt bij analoog-digitaalconversie. Dit behelst het omzetten van continu verlopende (meet)waarden in digitale codes. Het speciale is dat opeenvolgende codes slechts 1 bit verschillen. Dit betekent dat bij twijfel over de waarde van een bepaald bit er geen getallen kunnen ontstaan die ver buiten de gemeten waarden liggen.

- **a** Zoek uit hoe het systeem van deze code werkt en geef een tabel voor de omzetting van de getallen 0 tot en met 31 in een 5 bits Gray-code.
- **b** Voor programmeurs. Schrijf een computerprogramma dat een 8-bits Gray-code afdrukt voor elke waarde tussen 0 en 255.

1.31 Een (unsigned) binair getal heeft 3 cijfers achter de komma. Gevraagd:

2R 101,101 = 10R …, …

1.32 a Een floating point-getal in het IEEE-formaat (paragraaf 1.5.3) heeft de waarde:

1. 10000010. 01010100000000000000000

De punten dienen om de velden aan te geven. Bereken de waarde van dit getal in het decimale stelsel.

b Geef aan hoe het decimale getal 43,75 er in dit formaat uitziet.

1.33 Gegeven de ASCII-tabel (tabel 1.1). Beantwoord de volgende vragen:

a Geef de binaire code van ; en q.

b Welk bit moet worden veranderd om van kleine letters hoofdletters te maken?

c Als er sprake is van een ASCII-code met pariteitsbit en de pariteit is even, geef dan de codering voor de letter p (sla het pariteitsbit op in het MSB).

1.34 Als je de gelegenheid hebt om te programmeren, schrijf dan een programma dat alle karakters afdrukt waarvan de omgerekende decimale waarde ligt tussen 32 en 127. Als je ondernemend bent, kun je de grenzen uitbreiden van 0 tot 255, maar het resultaat is in dat geval onzeker!

1.35 Schrijf ook een programma dat na invoer van een decimaal getal en een grondtal de waarde van dit getal met deze radix afdrukt. Voorbeeld:

invoer:	257	8
uitvoer:	401	

1.36 Wat is het verschil tussen een error correcting- en een error detecting-code? Welke van deze heeft de meeste extra bits nodig?

1.37 Bij het eerste voorbeeld van datacompressie in paragraaf 1.7 waarbij gebruikgemaakt wordt van een woordenboek, kunnen geen eigennamen, acroniemen en dergelijke worden overgestuurd (een acroniem is een naam die is samengesteld uit beginletters van andere woorden, zoals *ASCII, BCD* en *IEEE*). Bedenk een uitbreiding van het systeem waarbij dat wel mogelijk is.

1.38 Verklaar het begrip *redundantie.*

1.39 Zoek met behulp van de ASCII-tabel (tabel 1.1) de codes op voor het woord 'Opgave'. Voeg horizontale en verticale pariteitsbits toe op de manier van figuur 1.4. Bedenk een foutenpatroon dat niet gecorrigeerd kan worden met deze code. Bedenk er ook een dat zelfs niet geconstateerd wordt.

1.40 Zoek op internet de Morse-code op. Bedenk een manier om deze code op te slaan in een computer, waarbij de code van elke letter in één byte moet worden opgeslagen (denk erom dat je ook lengte-informatie moet inbouwen!).

1.41 Voor een bepaalde dataoverdracht is de kans dat een bit onjuist wordt overgedragen 1 op 10^6. We maken gebruik van de corrigerende code van figuur 1.4 waarbij aan elke groep van 64 bits 2 controlebytes worden toegevoegd (dit heeft alleen zin als er per groep niet meer dan één fout optreedt). Geef een schatting van de kans dat het misgaat.

1.42 Zoek op internet op:

- **a** de ASCII-tabel;
- **b** de EBCDIC-tabel;
- **c** een beschrijving van de CRC methode;
- **d** de Graycode;
- **e** de Telexcode (wordt ook wel Baudot code genoemd).

Lees de beschrijvingen die erbij staan.

Kernbegrippen

Algoritme Formeel recept voor de oplossing van een probleem.

Analoog Letterlijk: met dezelfde woorden, gelijksoortig. In gebruik om een manier van coderen aan te geven waarbij het object en de representatie dezelfde continuïteit vertonen.

Arithmetisch schuiven De bits in een register zodanig schuiven dat het overeenkomt met door 2 delen (1 naar rechts) of met 2 vermenigvuldigen (1 naar links). Hierbij blijft het teken van het getal bewaard.

BCD-code Binairy Coded Decimals. Een manier van coderen waarbij voor elk decimaal getal een groep van 4 bits is gereserveerd.

Binair 2-tallig, 2-voudig.

Bit *Binary Information Ticket* of *binary digit:* kleinste hoeveelheid informatie.

Byte Acht bits.

Carry Overdracht bij optellen.

Character Engels voor het Nederlandse 'teken'. In computertalen betekent het vaak 'woord van 8 bits'.

Coderen Informatie vertalen naar een voorstelling in de computer.

Compressie Informatie met minder bits herschrijven.

Data Eén of meer data-items. Het wordt meestal als enkelvoud gebruikt: 'de data wordt verstuurd'.

Default Datgene wat gebruikt wordt als niets anders wordt opgegeven.

Digitaal Manier van coderen waarbij een beperkt aantal toestanden mogelijk is. Het wordt vaak gebruikt in de betekenis van 'binair digitaal'.

Encryptie Versleuteling: de data zodanig coderen dat alleen de gebruiker het origineel kan terugvinden.

Error correcting-code Code waarbij eventueel optredende fouten kunnen worden gecorrigeerd.

Error detecting-code Code waarbij eventueel optredende fouten kunnen worden geconstateerd.

Floating point-getallen Getallen met drijvende komma; een manier om getallen op te slaan met constante nauwkeurigheid en groot bereik.

Grondtal Aantal symbolen van een plaatsafhankelijk talstelsel.

Hexadecimaal 16-tallig stelsel, waarbij naast cijfers ook de symbolen A t/m F gebruikt worden.

Huffman-code Manier van coderen waarbij veelvoorkomende codes korter zijn dan codes die minder vaak voorkomen.

Integer Geheel getal.

LSB Least significant bit, het meest rechtse bit van een getal.

MSB Most significant bit, het meest linkse bit van een getal.

Nibble 4 bits.

Octaal stelsel Talstelsel met grondtal 8.

One's complement Manier van coderen waarbij voor negatieve getallen alle bits worden omgedraaid ($0 \rightarrow 1$ en $1 \rightarrow 0$).

Overflow Ontstaat wanneer bij een two's complement-optelling van twee positieve getallen een negatief getal ontstaat en omgekeerd.

Packed decimals BCD-achtige notatie met speciale tekens voor – en +.

Pariteit Het aantal énen in een woord.

Radix Ander woord voor grondtal.

Redundant Overtollig, in de zin dat meer bits worden gebruikt dan strikt nodig is.

Register Plaats waar een aantal bits wordt opgeslagen die in één keer verwerkt worden.

Signed magnitude Codering door een bit toe te voegen voor het teken.

Talstelstel Een manier om symbolen te ordenen tot getallen.

Tekencode Code voor letters, cijfers en leestekens.

Two's complement Meest gebruikte manier van coderen voor getallen die ook negatief mogen zijn. De code ontstaat door 1 op te tellen bij het one's complement.

Unsigned integers Gehele positieve getallen.

Waarheidstabel Tabel waarin voor alle mogelijke toestanden wordt aangegeven wat de waarden van de binaire variabelen zijn.

Woord Hoeveelheid databits die een computer in één keer verwerkt.

Zonebits De eerste 3 of 4 bits van een tekencode.