GRADUAAT IN HET PROGRAMMEREN

SEMESTER 2 & 3 ACADEMIEJAAR 2019-2020

GETALLEN

Basic IT Skills

laatst bijgewerkt op 3 november 2019

howest.be

INHOUDSOPGAVE

1 GETALLENSTELSELS

1.1 INLEIDING

Wellicht best gekende getallenstelsel (en meest toegepast) is het decimaal stelsel. In computertaal zijn de waardes echter beperkt tot 0 en 1 (schakelingen) waardoor we in computertermen gaan werken in het binair stelsel. Voor grotere binaire getallen kan dan worden gewerkt met het octaal of het hexadecimaal stelsel.

We leren in dit hoofdstuk hoe we getallen (eenvoudig) kunnen omzetten van het ene naar het andere stelsel.

Om concreet te kunnen weergeven in welk stelsel gewerkt wordt, maken we gebruik van de volgende notatie :

(getal)$_{\textbf{grondtal}}$ waarbij het **grondtal** weergeeft in welk stelsel er gewerkt wordt.

Standaard wordt bij decimale weergave het grondtal niet weergegeven.

1.2 DECIMAAL STELSEL

Het decimaal stelsel is het stelsel met als grondtal 10. Dit betekent dat we over 10 cijfers beschikken om een willekeurig getal weer te geven nl. van 0 tot en met 9. De waarde van een willekeurig getal wordt bepaald door :

- De waarde van het cijfer
- De positie van het cijfer

Men spreekt van een **positionele** notatie.

Een voorbeeld : Het getal $(5213)_{10}$ =

DUIZENDTALLEN	HONDERTALLEN	TIENTALLEN	EENHEDEN
10^3	10^2	10^1	10^0
5	2	1	3

$$(5213)_{10} = 5 \cdot 10^3 + 2 \cdot 10^2 + 1 \cdot 10^1 + 3 \cdot 10^0$$

Getallen met een decimale komma volgt eveneens deze regel, men start bij de komma om de waarde van het getal te bepalen.

Een voorbeeld : Het getal $(50,31)_{10}$ =

TIENTALLEN	EENHEDEN	TIENDEN	HONDERSTEN
10^1	10^0	10^{-1}	10^{-2}
5	0	3	1

$$(50,31)_{10} = 5 \cdot 10^1 + 0 \cdot 10^0 + 3 \cdot 10^{-1} + 1 \cdot 10^{-2}$$

Algemeen

In een stelsel met grondtal b heeft een getal de volgende waarde :Plaats gerust een opsomming:

$$(a_n .. a_1 a_0 , a_{-1} a_{-2} ...) = a_n . b^n + ... + a_1.b^1 + a_0b^0 + a_{-1}b^{-1} + ...$$

Voor extreem grote of extreem kleine getallen gebruik met voor de communicatie verschillende naamgevingen, hier wordt standaard per 3 posities gewerkt.

10^n	voorvoegsel	symbool	naam	
10^{24}	yotta	Y	quadriljoen	1.000.000.000.000.000.000.000.000
10^{21}	zetta	Z	triljard	1.000.000.000.000.000.000.000
10^{18}	exa	E	triljoen	1.000.000.000.000.000.000
10^{15}	peta	P	biljard	1.000.000.000.000.000
10^{12}	tera	T	biljoen	1.000.000.000.000
10^{9}	giga	G	miljard	1.000.000.000
10^{6}	mega	M	miljoen	1.000.000
10^{3}	kilo	k	duizend	1.000
10^{-3}	milli	m	duizendste	0,001
10^{-6}	micro	µ	miljoenste	0,000 001
10^{-9}	nano	n	miljardste	0,000 000 001
10^{-12}	pico	p	biljoenste	0,000 000 000 001
10^{-15}	femto	f	biljardste	0,000 000 000 000 001
10^{-18}	atto	a	triljoenste	0,000 000 000 000 000 001
10^{-21}	zepto	z	triljardste	0,000 000 000 000 000 000 001
10^{-24}	yocto	y	quadriljoenste	0,000 000 000 000 000 000 000 001

1.3 BINAIR STELSEL

Omdat de geheugencellen van computers slechts 2 waarden kunnen aannemen worden alle gegevens in een computersysteem intern voorgesteld in het binaire stelsel.

Het binair stelsel :

- Is een stelsel met grondtal 2
- Beschikt over 2 cijfers : 0 en 1

Hier is de algemene stelling eveneens van toepassing, waarbij het grondtal 2 wordt gebruikt.

Voorbeeld : $(1011,01)_2 = (1 . 2^3 + 0 . 2^2 + 1 . 2^1 + 1 . 2^0 + 0 . 2^{-1} + 1 . 2^{-2})_{10} = (8+0+2+1+0+0,25)_{10} = (11,25)_{10}$

1.3.1 OVERGANGSTABEL DECIMAAL NAAR BINAIR (GEHELE GETALLEN)

Hieronder vind je een overgangstabel tussen decimale en binaire getallen voor gehele getallen :

DECIMAAL	BINAIR	BEREKENING
0	0	
1	1	$= 1 . 2^0$
2	10	$= 1 . 2^1 + 0 . 2^0$
3	11	$= 1 . 2^1 + 1 . 2^0$
4	100	$= 1 . 2^2 + 0 . 2^1 + 0 . 2^0$
5	101	$= 1 . 2^2 + 0 . 2^1 + 1 . 2^0$
6	110	Enz…
7	111	
8	1000	
…		

1.3.2 BEWERKINGEN BINAIRE GETALLEN

Voor de optelling van binaire getallen gelden dezelfde regels als voor het decimale stelsel :

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10 met een overdracht van 1 naar een hogere rang (carry bit)

Uitgewerkt voorbeeld :

```
      1   1
    1 1 0 1 . 1
+   1 0 0 1 . 0 1
  1 0 1 1 0 . 1 1
```

1.3.3 CONVERSIES BINAIRE GETALLEN

BINAIR NAAR DECIMAAL

- Eerste manier (zie hierboven)

$(1\,0\,1\,1\,0\,,\,1\,1)_2 = (1 . 2^4 + 0 . 2^3 + 1 . 2^2 + 1 . 2^1 + 0 . 2^0 + 1 . 2^{-1} + 1 . 2^{-2})_{10} =$
$(1 . 16 + 0 . 8 + 1 . 4 + 1 . 2 + 0 . 1 + 1 . 1/2 + 1 . ¼)_{10} \quad = (16+4+2+0{,}5+0{,}25)_{10}$
$= (22{,}75)_{10}$

- Manier door opeenvolgende vermenigvuldiging (alleen voor gehele getallen)

Binair getal	1	0	1	1	0
Berekening		**2**	**4**	**10**	**22**
Resultaat		**2**	**5**	**11**	**22**

Werking :

- Bovenaan staat het om te zetten binair getal
- Vermenigvuldig eerste significant cijfer met 2 en plaats het resultaat onder het volgend cijfer van het binair getal
- Maak de decimale som van 2^{de} getal en dit resultaat en plaats dit in de 3^{de} rij
- Vermenigvuldig deze som met 2 en plaats dit onder het volgen binair getal
- Maak terug de decimale som van dit getal met het bovenstaand getal
- Vermenigvuldig deze som met 2 en plaats dit onder het volgend binair getal
- Maak terug de decimale som van dit getal met het bovenstaand getal
- Enz.
  Het laatst verkregen getal is de decimale waarde van het binair getal.

Resultaat $(1\ 0\ 1\ 1\ 0)_2 = (22)_{10}$

DECIMAAL NAAR BINAIR

Methode 1 : Delen of vermenigvuldigen met grondtal

- Conversie van gehele decimale getallen (<u>methode der opeenvolgende deling</u>) – enkel het deel voor de komma.

Bv. Getal $(123)_{10}$ = ( )$_2$

123	:	2	=	61	Rest :	1
61	:	2	=	30	Rest :	1
30	:	2	=	15	Rest :	0
15	:	2	=	7	Rest :	1
7	:	2	=	3	Rest :	1
3	:	2	=	1	Rest :	1
1	:	2	=	**0**	Rest :	1

Resultaat : $(123)_{10} = (1111011)_2$

- Conversie van gebroken decimale getallen (<u>methode der opeenvolgende vermenigvuldiging</u>) tot de gewenste nauwkeurigheid – enkel het deel na de komma

Bv. $(0,82)_{10}$ = ( )$_2$

0,82	.	2	=	**1**	,64	=>	0,**1**
0,64	.	2	=	**1**	,28	=>	0,1**1**
0,28	.	2	=	**0**	,56	=>	0,11**0**
0,56	.	2	=	**1**	,12	=>	0,110**1**
0,12	.	2	=	**0**	,24	=>	0,1101**0**
0,24	.	2	=	**0**	,48	=>	0,11010**0**
0,48	.	2	=	**0**	,89	=>	0.1101000
0,96	.	2	=	**1**	,92	=>	0,11010001

Resultaat : $(0,82)_{10} = (0,11010001\ldots)_2$

Getallen met een deel vòòr en na de komma worden omgezet door beide voorgaande methodes te combineren.

Methode 2 : De waardenrij

- Uitwerking
  - Zet de decimale waarde van de cijferposities op een rij
    - Voor de komma ga je tot het grootst mogelijke waarde die in het getal kan
    - Na de komma ga je tot de gewenste nauwkeurigheid
  - Start met het decimale getal
  - Zet onder de grootste waarde kleiner dan het decimaal getal een 1.
  - Trek deze waarde af van het decimale getal
  - Controleer waar het volgend net kleiner getal zich bevindt. Wanneer de waarde niet in het getal kan, dan wordt een 0 onder de waardenrij geplaatst
  - Voer verder uit tot de rest van het getal gelijk is aan 0 (of tot de gewenste nauwkeurigheid)

Bv. $(91,72)_{10}$ = ($)_2$

POSITIE	26	25	24	23	22	21	20	,	2-1	2-2	2-3
Decimaal	**64**	**32**	**16**	**8**	**4**	**2**	**1**		**0,5**	**0,25**	**0,125**
Binair getal	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**,**	**1**	**0**	**1**

Voor de komma : $91 - \mathbf{1} \cdot 2^6 = 27 - \mathbf{0} \cdot 2^5 = 27 - \mathbf{1} \cdot 2^4 = 11 - \mathbf{1} \cdot 2^3 = 3 - 0 \cdot 2^2 = 3 - 1 \cdot 2^1 = 1 - 1 \cdot 2^0 = \mathbf{0}$

Na de komma : $0,72 - \mathbf{1} \cdot 2^{-1} = 0,22 - \mathbf{0} \cdot 2^{-2} = 0,22 - \mathbf{1} \cdot 2^{-3} = 0.095$ …

Resultaat = $(91,72)_{10} = (1011011,101)_2$

1.4 HET OCTAAL STELSEL

Het octale stelsel is vooral in de beginjaren van de computer gebruik om de binaire getallen overzichtelijk weer te geven. Omdat de huidige computers vrijwel rekenen met even aantallen bits en worden voor de representatie van binaire gegevens meestal het hexadecimale stelsel toegepast. (Zie verder)

Het octaal stelsel :

- Is een stelsel met grondtal 8
- Beschikt over 8 cijfers: van 0 tot en met 7

Een voorbeeld :

$(1454)_8 = (1 \cdot 8^3 + 4 \cdot 8^2 + 5 \cdot 8^1 + 4 \cdot 8^0)_{10} = (512 + 256 + 40 + 4)_{10} = (812)_{10}$

1.4.1 OVERGANG DECIMAAL => OCTAAL => BINAIR

DECIMAAL	OCTAAL	BEREKENING	BINAIR
0	0		000
1	1	$= 1 . 8^0$	001
2	2	$= 2 . 8^0$	010
3	3		011
4	4		100
5	5		101
6	6		110
7	7	$= 7 . 8^0$	111
8	10	$= 1 . 8^1 + 0 . 8^0$	1 000
9	11	$= 1 . 8^1 + 1 . 8^0$	1 001
10	12		1 010
11	13		1 011
12	14		1 100
…			

1.4.2 CONVERSIES OCTALE GETALLEN

OCTAAL NAAR DECIMAAL

- Zet ieder getal om naar zijn decimale waarde afhankelijk van de plaats in de waardenrij en hun positie ten opzichten van de komma.

Voorbeeld : $(3705,2)_8$ = ( )$_{10}$

= $(3 . 8^3 + 7 . 8^2 + 0 . 8^1 + 5 . 8^0 + 2 . 8^{-1})_{10}$
= $(1536 + 448 + 0 + 5 + 0,25)_{10}$
= $(1989,25)_{10}$

Resultaat : $(3705,2)_8 = (1989,25)_{10}$

DECIMAAL NAAR OCTAAL

Methode 1 : Delen of vermenigvuldigen met grondtal

- Voor gehele getallen :
  - opeenvolgende delingen door 8 tot 0, restwaarden in omgekeerde volgorde levert het getal op in octale weergave.
- Voor gebroken getallen (na de komma) :
  - opeenvolgende vermenigvuldiging met 8. Eerste cijfer levert de octale weergave na de komma op.
- Voor een gebroken getal met waardes voor en na de komma gebruik je beide methoden.

Voorbeeld :

$(1273)_{10}$ = ( )$_8$

1273	:	8	=	159	Rest	:	1
159	:	8	=	19	Rest	:	7
19	:	8	=	2	Rest	:	3
2	:	8	=	0	Rest	:	2

Resultaat : $(1273)_{10} = (4771)_8$

Voorbeeld :

$(0,18)_{10}$ = ( )$_8$

0,18	.	8	=	**1**	,44	=> 0,**1**
0,44	.	8	=	**3**	,52	=> 0,1**3**
0,52	.	8	=	**4**	,16	=> 0,13**4**
0,16	.	8	=	**1**	,28	=> 0,134**1**
0,28	.	8	=	**2**	,24	=> 0,1341**2**

Resultaat : $(0,18)_{10} = (0,13412..)_8$

Combineren (zie voorgaande voorbeelden) : $(1273,18)_{10} = (2371,13412..)_8$

Methode 2 : De waardenrij

Positie	8^4	8^3	8^2	8^1	8^0	,	8^{-1}	8^{-2}
Decimaal	4096	512	64	8	1	,	0,125	0,015625
Binair getal	**0**	**3**	**7**	**0**	**7**	**,**	**2**	**0**

Voorbeeld :

$(1991,25)_{10} = (\quad)_8$

Voor de komma : $1991 - \mathbf{3} \cdot 8^3 = 455 - \mathbf{7} \cdot 8^2 = 7 - \mathbf{0} \cdot 8^1 = 7 - \mathbf{7} \cdot 8^0 = 0$

Na de komma : $0,25 - \mathbf{2} \cdot 8^{-1} = 0$

Resultaat :

$(1991,25)_{10} = (3707,2)_8$

OCTAAL NAAR BINAIR

Er bestaat een logisch verband tussen octale en binaire getallen.

Een octaal cijfer komt overeen met een binair cijfer bestaande uit 3 bits. Met 3 binaire cijfers kun je 8 combinaties maken gaande van cijfer 0 tot cijfer 7.

Een voorbeeld :

$(3\ 7\ 1\ 5\ ,\ 6\ 4)_8 = (011\ 111\ 001\ 101\ ,\ 110\ 100)_2 = (11111001101,1101)_2$

BINAIR NAAR OCTAAL

Dit logisch verband geldt uiteraard ook in de andere richting.

Om een binair getal eenvoudig om te zetten naar zijn octaal equivalent onderneem je volgende stappen :

- Vertrek vanaf de komma naar links voor gehele getallen en groepeer de cijfers per 3 (eventueel vooraan aanvullen)
- Vanaf de komma naar links groepeer je eveneens de binaire getallen in groepjes van 3 (eventueel achteraan aanvullen)
- Zet ieder groepje van 3 bits om naar zijn octale waarde.

Voorbeeld :

$(10110110,1011)_2 = (\quad)_8$

$= (010\ 110\ 110\ ,\ 101\ 100)_2 = (2\ 6\ 6\ ,\ 5\ 4)_8 = (266,54)_8$

1.5 HEXADECIMAAL STELSEL

Wellicht iets meer gebruikt in informatica om de binaire getallen om te zetten in zijn hexadecimale waarde :

- MAC – adressen van PC's
- IPv6 Adressen
- Kleurencodes (Web Frontend Basics)
- …

Het hexadecimaal stelsel :

- Is een stelsel met grondtal 16
- Beschikt over 19 cijfers(tekens) : van 0 tot en met 9 en van A tot en met F

Een voorbeeld :

$(3AC7)_{16} = (3 \cdot 16^3 + A \cdot 16^2 + C \cdot 16^1 + 7 \cdot 16^0)_{10} = (3 \cdot 16^3 + 10 \cdot 16^2 + 12 \cdot 16^1 + 7 \cdot 16^0)_{10} =$

$(12288 + 2560 + 192 + 7)_{10} = (15047)_{10}$

1.5.1 OVERGANG DECIMAAL => HEXADECIMAAL => BINAIR

DECIMAAL	HEXEDECIMAAL	BEREKENING	BINAIR
0	0		0000
1	1	$= 1 . 16^0$	0001
2	2	$= 2 . 16^0$	0010
3	3		0011
4	4		0100
5	5		0101
6	6		0110
7	7		0111
8	8		1000
9	9		1001
10	A	$= 10 . 16^0$	1010
11	B	$= 11 . 16^0$	1011
12	C		1100
13	D		1101
14	E		1110
15	F		1111
16	10	$= 1 . 16^1 + 0 . 16^0$	1 0000
17	11		1 0001
18	12		1 0010
19	13		1 0011
20	14		1 0100

1.5.2 CONVERSIES HEXADECIMALE GETALLEN

HEXADECIMAAL NAAR DECIMAAL

Voorbeeld :

$$(20C,B)_{16} = (2 \cdot 16^2 + 0 \cdot 16^1 + C \cdot 16^0 + B \cdot 16^{-1})_{10} = (512 + 0 + 12 + 0{,}6875)_{10} = (524{,}6875)_{10}$$

Gezien je voor het hexadecimaal stelsel snel grote getallen bereikt zijn er tabellen beschikbaar die het rekenen iets eenvoudiger maken :

- Gehele getallen

HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	268 435 456	1	16 777 216	1	1 048 576	1	65 536	1	4 096	1	256	1	16	1	1
2	536 870 912	2	33 554 432	2	2 097 152	2	131 072	2	8 192	2	512	2	32	2	2
3	805 306 368	3	50 331 648	3	3 145 728	3	196 608	3	12 288	3	768	3	48	3	3
4	1 073 741 824	4	67 108 864	4	4 194 304	4	262 144	4	16 384	4	1 024	4	64	4	4
5	1 342 177 280	5	83 886 080	5	5 242 880	5	327 680	5	20 480	5	1 280	5	80	5	5
6	1 610 612 736	6	100 663 296	6	6 291 456	6	393 216	6	24 576	6	1 536	6	96	6	6
7	1 879 048 192	7	117 440 512	7	7 340 032	7	458 752	7	28 672	7	1 792	7	112	7	7
8	2 147 483 648	8	134 217 728	8	8 388 608	8	524 288	8	32 768	8	2 048	8	128	8	8
9	2 415 919 104	9	150 994 944	9	9 437 184	9	589 824	9	36 864	9	2 304	9	144	9	9
A	2 684 354 560	A	167 772 160	A	10 485 760	A	655 360	A	40 960	A	2 560	A	160	A	10
B	2 952 790 016	B	184 549 376	B	11 534 336	B	720 896	B	45 056	B	2 816	B	176	B	11
C	3 221 225 472	C	201 326 592	C	12 582 912	C	786 432	C	49 152	C	3 072	C	192	C	12
D	3 489 660 928	D	218 103 808	D	13 631 488	D	851 968	D	53 248	D	3 328	D	208	D	13
E	3 758 096 384	E	234 881 024	E	14 680 064	E	917 504	E	57 344	E	3 584	E	224	E	14
F	4 026 531 840	F	251 658 240	F	15 728 640	F	983 040	F	61 440	F	3 840	F	240	F	15
8		**7**		**6**		**5**		**4**		**3**		**2**		**1**	

- Gebroken getallen (na de komma)

HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0,0	0,0000	0,00	0,0000 0000	0,000	0,0000 0000 0000	0,0000	0,0000 0000 0000 0000
0,1	0,0625	0,01	0,0039 0625	0,001	0,0002 4414 0625	0,0001	0,0000 1525 8789 0625
0,2	0,1250	0,02	0,0078 1250	0,002	0,0004 8828 1250	0,0002	0,0000 3051 7578 1250
0,3	0,1875	0,03	0,0117 1875	0,003	0,0007 3242 1875	0,0003	0,0000 4577 6367 1875
0,4	0,2500	0,04	0,0156 2500	0,004	0,0009 7656 2500	0,0004	0,0000 6103 5156 2500
0,5	0,3125	0,05	0,0195 3125	0,005	0,0012 2070 3125	0,0005	0,0000 7629 3945 3125
0,6	0,3750	0,06	0,0234 3750	0,006	0,0014 6484 3750	0,0006	0,0000 9155 2734 3750
0,7	0,4375	0,07	0,0273 4375	0,007	0,0017 0898 4375	0,0007	0,0001 0681 1523 4375
0,8	0,5000	0,08	0,0312 5000	0,008	0,0019 5312 5000	0,0008	0,0001 2207 0312 5000
0,9	0,5625	0,09	0,0351 5625	0,009	0,0021 9726 5625	0,0009	0,0001 3732 9101 5625
0,A	0,6250	0,0A	0,0390 6250	0,00A	0,0024 4140 6250	0,000A	0,0001 5258 7890 6250
0,B	0,6875	0,0B	0,0429 6875	0,00B	0,0026 8554 6875	0,000B	0,0001 6784 6679 6875
0,C	0,7500	0,0C	0,0468 7500	0,00C	0,0029 2968 7500	0,000C	0,0001 8310 5468 7500
0,D	0,8125	0,0D	0,0507 8125	0,00D	0,0031 7382 8125	0,000D	0,0001 9836 4257 8125
0,E	0,8750	0,0E	0,0546 8750	0,00E	0,0034 1796 8750	0,000E	0,0002 1362 3046 8750
0,F	0,9375	0,0F	0,0585 9375	0,00F	0,0036 6210 9375	0,000F	0,0002 2888 1835 9375
1		**2**		**3**		**4**	

De werking met de tabellen is als volgt :

Voorbeeld :

$(2CB3,5A)_{16}$ = ( )$_{10}$

a. **Geheel deel :**

Rang	4	3	2	1
Waarde	2	C	B	3

4ste rangwaarde	=	$2 . 16^3$	=	8192
3de rangwaarde=		$C . 16^2$	=	3072
2de rangwaarde	=	$B . 16^1$	=	176
1de rangwaarde	=	$3 . 16^0$	=	3

Geheel deel = $(8192 + 3072 + 176 + 3)_{10} = (11433)_{10}$

b. **Deel na komma**

Rang	1	2
Waarde	1	A

1ste rangwaarde	=	$1 . 16^{-1}$	=	0,0625
2de rangwaarde	=	10.16^{-2}	=	0,03906250

Deel na komma = $(0,0625+0,0390625)_{10} = (0,1015625)_{10}$

Resultaat :

$(2CB3,1A)_{16}$ = $(11433,1015625)_{10}$

DECIMAAL NAAR HEXADECIMAAL

Methode 1 : Delen of vermenigvuldigen met grondtal

- Voor gehele getallen :
  - opeenvolgende delingen door 16 tot 0, restwaarden in omgekeerde volgorde levert het getal op in hexadecimale weergave.
- Voor gebroken getallen (na de komma) :
  - opeenvolgende vermenigvuldiging met 16. Eerste cijfer levert de hexadecimale weergave na de komma op
- Voor een gebroken getal met waardes voor en na de komma gebruik je beide methoden.

Voorbeeld :

$(16241,712)_{10}$ = ( )$_{16}$

- Geheel getal door opeenvolgende deling door 16
  -
  - 16241 : 16 = 1015 Rest : 1
  - 1015 : 16 = 63 Rest : 7
  - 63 : 16 = 3 Rest : 15 = $(F)_{16}$
  - 3 : 16 = 0 Rest : 3

Dus : $(16241)_{10}$ = $(3F71)_{16}$

- Gebroken getallen door opeenvolgend vermenigvuldiging met 16
-
    - 0,712 . 16 = 11,395 => 0,B
    - 0,395 . 16 = 6,272 => 0,B6
    - 0,272 . 16 = 4,352 => 0,B64
    - 0,352 . 16 = 5,632 => 0,B645

  Dus : $(0,712)_{10}$ = $(0,B645)_{16}$

Resultaat = $(16241,712)_{10}$ = $(3F71,B645)_{16}$

Methode 2 : Met de tabellen

Werkwijze :

Gehele getallen

1. Zoek de grootste waarde uit de tabellen net kleiner dan de decimale waarde (of gelijk aan)
2. Je kent nu de positie van het grootste getal => Maak een rooster met evenveel vakken als er posities zijn en vul deze waarde in dit rooster.
3. Verminder het getal met de gevonden waarde uit de tabel en kijk of er terug een waarde kan gevonden worden in de volgende kolom van de tabel, opnieuw net kleiner of gelijk aan de waarde
4. Doe dit voor alle posities, en je vindt de hexadecimale waarde van het geheel getal

Gebroken getallen (getallen na de komma)

1. Zoek de grootste waarde uit de tabel net kleiner of gelijk dan het gebroken getal (deze waarde komt op positie 1)
2. Verminder het getal met de gevonden waarde, en zoek nu in de volgende kolom opnieuw de grootste waarde kleiner dan dit getal. Vul dit in voor positie 2.
3. … Dit kun je dan verder doen tot je een nulwaarde bekomt, in ons geval kunnen we dit doen tot positie 4.

Voorbeeld : $(9251,42)_{10}$ = ($)_{16}$

Gehele getal :

$(9251)_{10}$

1. Waarde dichtst in tabel is op positie 4 HEX **2** = $(8192)_{10}$
2. Er zijn dus **4 posities** nodig om het getal te maken :

Positie	4	3	2	1
Waarde	2			

3. Restbepaling : $9251 - \mathbf{2} \cdot 16^3 = 9251 - 8192 = 1059$
4. Waarde dichtst in tabel op positie 3 zoeken : HEX **4** = $(1024)_{10}$

Positie	4	3	2	1
Waarde	2	4		

5. Restbepaling : $1059 - 4 \cdot 16^2 = 1059 - 1024 = 35$
6. Waarde dichtst in tabel op positie 2 zoeken : HEX **2** = $(32)_{10}$

Positie	4	3	2	1
Waarde	2	4	2	

7. Restbepaling : $35 - 2 \cdot 16^1 = 35 - 32 = 3$
8. Waarde dichtst in de tabel op positie 1 zoeken : HEX **3** = $(3)_{10}$

Positie	4	3	2	1
Waarde	2	4	2	3

9. Restbepaling : $3 - \mathbf{3} \cdot 16^0 = 3 - 3 = 0$

Geheel getal : $(9251)_{10} = (2423)_{16}$

Getal na komma :

$(0,42)_{10}$

1. Waarde dichtst in tabel is op positie 1 HEX 6 = $(0,3750)_{10}$

Positie	1	2	3
Waarde	6		

2. Restbepaling : $0,42 - \mathbf{6} \cdot 16^{-1} = 0,42 - 0,375 = 0,045$
3. Waarde dichtst in tabel op positie 2 zoeken : HEX **B** = $(0,0429\ldots)_{10}$

Positie	1	2	3
Waarde	6	B	

4. Restbepaling : $0,045 - 4 \cdot 16^{-2} = 0,00203125$
5. Waarde dichtst in tabel op positie 2 zoeken : HEX **8** = $(0,0019\ldots)_{10}$

Positie	1	2	3
Waarde	6	B	8

6. Restbepaling : $0,00203125 - 8 \cdot 16^{-3} = 0,00007813\ldots$

Getal na komma : $(0,42)_{10} =$ $(0,6B8)_{16}$

Resultaat : $(9251,42)_{10} =$ $(2423,6B8)_{16}$

2 NEGATIEVE GETALLEN

2.1 INLEIDING

Negatieve getallen worden in het decimaal stelsel voorgesteld als zijnde hun waarde voorafgegaan door een min-teken. (-)

Bij computers echter beschikken we niet over het min-teken, maar enkel 2 tekens (0 en 1).
Hieruitvolgend zal het min- teken dat we kennen uit het decimaal stelsel ook moeten voorgesteld door een 0 of een 1.

De afspraak is dat de meest linkse bit een voorstelling geeft van het + of – teken :

0 : Positief getal
1 : Negatief getal

Bij voorstelling in een register :

SIGN-BIT	MAGNITUDE
(0 of 1)	010111010..

Er zijn een aantal mogelijkheden voor de voorstelling van netgatieve getallen.

2.2 MOGELIJKE VOORSTELLINGEN VAN NEGATIEVE GETALLEN

2.2.1 TEKEN – MAGNITUDE SYSTEEM

In dit systeem bestaat een getal uit een tekenbit en de absulute waarde van het getal.

Een voorbeeld in een 8-bits register :

0	0	0	0	1	1	0	0	+ 12
1	0	0	0	1	1	0	0	- 12

Het bereik van dit 8-bits register :

Maximum 0 1111111 => + 127 = $(2^7 – 1)$
Minimum 1 1111111 => - 127 = - $(2^7 – 1)$

Er zijn echter wat nadelen aan dit syteem :

- 2 representaties van 0 (+0 en -0)
- Bij berekeningen levert het bitgewijze optellen geen correct resultaat

Wegens deze redenen wordt dit stelsel in de praktijk nooit gebruikt.

2.2.2 COMPLEMENT – SYSTEEM

Voorstelling positief getal : 0 in tekenbit + magnitude van de andere bits

Voorstelling negatief getal :1 in tekenbit + grondtal-complement van de magnitude van de ander bits

Complement van een stelsel
Wanneer N het grondtal is van een stelsel dan is :

- Het N-complement = elk cijfer aftrekken van (N-1) + 1 (LSB +1)
- Het (N-1)-complement = elk cijfer aftrekken van (N-1)

Enkele voorbeelden :

1. Stel N = 10 (decimaal stelsel)
   - Het 10 complement van 624 is 376
   - Het 9 complement van 624 is 375
2. Stel N=16 (hexadecimaal stelsel)
   - Het 16 complement van 5A9 is A57
   - Het 15 complement van 5A9 is A56
3. Stel N=2 (Binair stelsel)
   - Het 2 complement van 1101 is 0011
   - Het 1 complement van 1101 is 0010

Voor het berekenen van een binair complement kan een eenvoudige vuistregel gebruikt worden :

- Bij 1 complement veranderen alle bits van waarde
- Bij 2 complement neemt men eerst het 1 complement en telt bij dit resultaat 1 op

Voor de weergave :

- Positieve getallen : Sign-bit 0 en Magnitude
- Negatieve getallen : Sign-bit 1 en Complement van magnitude

Een overzichtstabel (8-bits)

Bereik	+127 tot en met +0 - 0 tot en met -127	+127 tot en met +0 - 0 tot en met -127	+127 tot en met -128
Decimaal	Teken + Magnitude (zie2.1)	1 – complement	2 – complement
+ 127	0111 1111	0111 1111	0111 1111
+ 126	0111 1101	0111 1101	0111 1101
+ …			
+ 1	0000 0001	0000 0001	0000 0001
+ 0	**0000 0000**	**0000 0000**	**0000 0000**
- 0	**1000 0000**	**1111 1111**	**(1) 0000 0000**
- 1	1000 0001	1111 1110	1111 1111
- 2	1000 0010	1111 1101	1111 1110
- …			
- 126	1111 1110	1000 0001	1000 0010
- 127	1111 1111	1000 0000	1000 0001
- 128	-	-	**1000 0000**

Het 1 -complement levert net zoals het teken en magnitude systeem een nadeel op : er zijn nog altijd 2 representaties voor 0. Het rekenen kan nu wel bitsgewijs.

Bij het 2 complement zijn voorgenoemde problemen wel weggewerkt. In de praktijk wordt dan ook bijna uitsluitend gebruik gemaakt van het 2-complement systeem. Voor deze cursus wordt vanaf nu dan ook nog alleen met dit systeem verder gewerkt.

OMZETTING VAN 2-COMPLEMENT GETALLEN NAAR HUN DECIMALE VORM

Getallen die in hun complement vorm worden voorgesteld zijn uiteraard altijd negatieve getallen. Willen we weten welke getallen ze representeren dienen we ze eerst terug om te zetten naar hun positieve waarde.

De methode van omzetting is gelijkaardig aan deze hierboven beschreven : Geef elke teken een andere waarde en tel daarna bij de LSB waarde 1 op.

Voorbeeld :

$(1111\ 1011)_2 = (\ )_{10}$

=> Inverteren : 0000 0100 +1 = 0000 0101 = $(5)_{10}$

=> $(\mathbf{1}111\ 1011)_2 = (\mathbf{-}5)_{10}$

BEWERKINGEN MET NEGATIEVE GETALLEN

De computer zal een aftrekking verwerken als een optelling. De afgetrokken waarde wordt wanneer het een negatief getal is omgezet naar zijn 2 complementenwaarde.

Voorbeeld 1 :

Decimaal : $(-5)_{10} – (-10)_{10} = (1111\ 1011 – 1111\ 0110)_2 = ?$

Werkwijze : De aftrekker wordt geïnventeerd naar zijn 2 complementenwaarde zodat de aftrekking een optelling
wordt.

Binair :

	1 1111 1	Carry bit
	1111 1011	$(-5)_{10}$
+	0000 1010	$(+10)_{10}$
	1 $(0000\ 0101)_2$	$(+5)_{10}$

De carry-bit (= de bit die er als het ware 'uitvalt') kan in het 2-complementensysteem weggelaten worden. Dit in tegenstelling tot het 1-complement systeem waar gewerkt wordt let het zgn 'end around carry' : Als er zich een carry bit voordoet dan wordt hij bij het resultaat opgeteld.

Voorbeeld 2 :

Decimaal = $(+5)_{10} – (+10)_{10} = (0000\ 0101 - 0000\ 1010)_2 = ?$

Werkwijze : De aftrekker wordt geïnventeerd naar zijn 2 complementenwaarde zodat de aftrekking een optelling
wordt.

Binair :

	1	Carry bit
	0000 0101	$(+5)_{10}$
+	1111 0110	$(-10)_{10}$
	1111 1011	= geïnverteerd 0000 0101 => $(-5)_{10}$ (Sign bit = 1 => Inverteren)

Uit de voorbeelden kan ook duidelijk afgeleid worden dat her er niet toe doet in welke vorm de te berekenen waarden zich gaan aanbieden. (Positief of negatief / gecomplementeerd)

OVERFLOW

Bij het uitvoeren van de berekening kan het gebeuren dat het register dat het resultaat ontvangt te klein is.

(carry-out bit)

Bij het uitvoeren van de berekening kan het gebeuren dat het register dat het resultaat ontvangt te klein is.

(carry-out bit)

Voorbeeld (8-bits register) :

Decimaal : $(92)_{10} + (45)_{10} = (\mathbf{0}1011100)_2 + (\mathbf{0}0101101)_2 = ?$

Binaire berekening :

	1111 1	=> Carry bits
	0101 1100	$(+92)_{10}$
+	0010 1101	$(+45)_{10}$
	1000 1001	=> -(geïnverteerd 0111 0111)$_2$ = -$(119)_2$ DUS FOUT !

Wanneer we met binaire getallen werken moeten we dus rekening houden met een aantal regels rekening houdend met de grootte van het register en de sign-bit :

- Is er een carry (overdracht van waarde) naar de MSB[1] en geen carry-out (overdracht van waarde buiten het register) dan spreken we van OVERFLOW.
- Is er geen carry naar de MSB en wel een carry-out dan is er eveneens een OVERFLOW
- Is er een carry naar de MSB en een carry-out dan is er geen overflow.
- Is er geen carry naar de MSB en geen cary-out dan is er geen overflow.

Bij het voorbeeld die hierboven staat is, dan stellen we dat volgens de eerst regel hierboven vermeldt wel degelijk een overflow is.

[1] Most Significant Bit (Binaire getallen)

3 VOORSTELLING VAN GEBROKEN GETALLEN

3.1 INLEIDING

In wetenschappelijke toepassing is het soms nodig berekeningen te maken met zeer grote of kleine getallen. Omz deze getallen voor te stellen in bitnotatie gebruikt met een mantisse en exponentwaarde.

Voor decimale weergave kennen we allemaal de volgende notatie :

$587{,}1234 = 0{,}5871234 \cdot 10^{3}$

$0{,}00123 = 0{,}123 \cdot 10^{-2}$

Symbolisch voorgesteld :

$a \cdot r^{p}$ waarbij a = mantisse
r = grondtal van het gebruikte stelsel
p = exponent
r^{p} = schaalfactor

Bepaalde computersystemen werken met een mantisse die steeds een geheel getal is (integer gedeelte). De meeste computersystemen werken echter met een mantisse die een gebroken getal voorstelt kleiner dan 1 (met spreekt van een fraction). De plaats van de komma wordt dan bepaald door de waarde van het exponent (binair weergegeven).

We spreken hier algemeen van een **floating point** notatie (vlottende komma).

3.2 VOORSTELLING VAN GEBROKEN GETALLEN (INTEL PENTIUM)

Gebroken getallen worden voorgesteld in de IEEEE (Institute of Electrical & Electronic Engineers) notatie als volgt :

$\text{Getal} = S\ 1{,}M \cdot 2^{E-y}$

Waarbij : S : Sign-bit (0 : Positief getal of 1 : Negatief getal)
M : Mantisse (Let op **1,**M)
E-y : Exponent
y : Afhankelijk van formaat – zie verder

Er zijn 3 mogelijke formaten :

- Short Real (32 bits)

  $\text{Getal} = S\ 1{,}M \cdot 2^{E-127}$
  Enkel de variabele delen worden opgeslagen. => Nauwkeurigheid : 7 decimale cijfers

  Weergave :

Short Real	S	E	M
Aantal bits	1	8	23

- Long Real (64 bits)
  -
  - $\text{Getal} = S\ 1{,}M \cdot 2^{E-1027}$

- Enkel de variabele delen worden opgeslagen => Nauwkeurigheid : 15 decimale cijfers

Weergave :

Sign-bit	Exponent	Mantisse
0	10000111	00011011100000000000000

Long Real	S	E	M
Aantal bits	1	11	52

- Temporary Real (80 bits)

  - Getal = S 1,M . $2^{E-16383}$
  - Enkel de variabele delen worden opgeslagen => Nauwkeurigheid : 19 decimale cijfers

CONVERSIE VAN DECIMAAL NAAR HEXADECIMALE IEEEE SHORT REAL NOTATIE

Voorbeeld :

$(-70,75)_{10}$ = ($)_{16}$ IEEEE-Short Real

- Stap 1 : Bepaling van Sign-bit in dit geval negatief : 1
- Stap 2 : Zet het getal om naar zijn binaire notatie
  $(70,75)_{10} = (1000110,11)_2$
- Stap 3 : Bepaal mantisse en Exponent
  - $(1000110,11)_2 = (\mathbf{1},00011011)_2 . 2^{\mathbf{6}}$
  - => Mantisse : 00011011000000… (totaal 23 bits)
  - => Exponent : E-127 = **6** => E = $(133)_{10} = (10000101)_2$ (Totaal 8 bits)
- Stap 4 : Short Real Notatie (via stap 1 en 3)

- Stap 5 : Zet om naar hexadecimaal equivalent

Sign-bit	Exponent	Mantisse
1	10000101	00011011000000000000000

$(1100\ 0010\ 1000\ 1101\ 1000\ 0000\ 0000\ 0000)_2 = (C\ 2\ 8\ D\ 8\ 0\ 0\ 0\ )_{16}$

Resultaat = $(-70,75)_{10} = (C28D8000)_{16}$ IEEEE Short real

CONVERSIE VAN HEXADECIMALE SHORT REAL NOTATIE NAAR DECIMAAL

Voorbeeld

$(438DC000)_{16}$ IEEE short Real = ($)_{10}$

- Stap 1 : Zet hexadeciaal om naar binair in IEEE short Real Notatie

$(438DC000)_{16} = (0100\ 0011\ 1000\ 1101\ 1100\ 0000\ 0000\ 0000)_2$

- Bepaal Sign-bit, Mantisse en Exponent
  - Sign- Bit = 0 => Positief getal (eerste bit)
  - Exponent : E-127 = $(10000111)_2 = (135)_{10}$ => E = 135 (volgende 8 bits)
  - Mantisse = 0001101110.. (volgende 23 bits)

- Bereken het decimaal getal

  Getal = S 1,M . 2^{E-127}

  Getal = + 1,000110111… . $2^{135-127}$

  Getal = + 1,000110111 . $2^8 = (100011011,1)_2$

  Getal = $(283,5)_{10}$

4 CIJFERCODES

4.1 INLEIDING

Om getallen naar de schermweergave om te zetten, of een getalinterpretatie van een toetsenbord wordt gebruik gemaakt van cijfercodes. Hieronder vind je er enkele met hun specifieke afspraken.

4.2 BCD – CODE (BINARY CODED DECIMAL)

Bij deze werkwijze wordt een getalwaarde voorgesteld door 4 bits, zodat de getallen 0 tot en met 9 binair kunnen worden voorgesteld.

Voorbeeld :

Decimale waarde	:	164
BCD notatie	:	0001 0110 0100 (4 bits omzetting van individuele waarde)

Gebruik :

Bij een aantal I/O bewerkingen. De cijfers komen teken/teken binnen BCD-notitie waar zij worden verzameld en omgezet in kun binaire waarde. Ook bij uitvoer naar de printer kunnen binaire getallen eerst omgezet worden naar jun BCD-code en vervolgens naar de printer worden gestuurd.

Zie ook :

http://en.wikipedia.org/wiki/Binary-coded_decimal

4.3 ZONED DECIMAL

In Zoned decimal wordt elk cijfer voorgesteld als 1 Byte (= 8 bits), waarbij de rechtse 4 bits te BCD notatie is van elk cijfer en de linkse 4 bits steeds de waarde 1111 hebben, met uitzondering van de 4 linkse bits van het minst beduidende cijfer die het teken aangeeft.

1111 => ongetekend (absoluut)
1100 => Positief
1101 => Negatief

Voorbeelden :

256 in ZD-notatie :	1111 0010 1111 0101 1111 0110
+256 in ZD-notatie :	1111 0010 1111 0101 1100 0110
-256 in ZD-notatie :	1111 0010 1111 0101 1101 0110

Zie ook :

http://en.wikipedia.org/wiki/Binary-doded_decimal#Zoned_decimal

4.4 PACKED DECIMAL

In de Packed Decimal notatie wordt elk cijfer voorgesteld door zijn BCD-vorm waarna uiterst rechts 4 bits worden toegevoegd als teken.

1111	=>	Ongekend
1100	=>	Positief
1101	=>	Negatief

Enkele voorbeelden :

179 in PD-notatie	=>	0001 0111 1001 1111
+179 in PD-notatie	=>	0001 0111 1001 1100
-179 in PD-notatie	=>	0001 0111 1001 1101

Opmerking :

Indien nodig wordt de meest linkse Byte opgevuld met 4 bits met de waarde 0. (tbv representie)

+12 in PD-notatie	=>	0000 0001 0010 1100

Zie ook :

http://en.wikipedia.org/wiki/Binary-coded_decimal#Packed_BCD

5 ALFANUMERIEKE CODES

5.1 INLEIDING

Ook alfanumerieke codes hebben een specifieke codering, wellicht de bekendste is hierin de ASCII-code.

5.2 EBCDIC-CODE (EXTENDED BINARY CODE DECIMAL INTERCHANGE CODE)

De EBCDIC code is een 8-bit code (256 mogelijkheden die echter niet ten volle worden gebruikt). Deze codes zijn van toepassing bij mini-computers en mainframes.

Omzettabel :

DIGIT ↓ / ZONE →	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	NULL				SP	&	-									0
0001							/		a	j			A	J		1
0010									b	k	s		B	K	S	2
0011									c	l	t		C	L	T	3
0100	PF	RES	BYP	PN					d	m	u		D	M	U	4
0101	HT	NL	LF	RS					e	n	v		E	N	V	5
0110	LC	BS	EOB	UC					f	o	w		F	O	W	6
0111	DEL	IL	PRE	EOT					g	p	x		G	P	X	7
1000									h	q	y		H	Q	Y	8
1001									i	r	z		I	R	Z	9
1010			SM			!		:								
1011					.	$	,	#								
1100					<	*	%	@								
1101					(	)	-	'								
1110					+	;	>	=								
1111					|	~	?	"								
	Datacommunicatie				Speciale karakters				kleine letters				hoofdletters en cijfers			

De voorstelling van 1 Byte is als volgt :

Zone				Digit			
0	1	2	3	4	5	6	7

Waarbij :

0	1

2	3

Bij 11..

00 = Datacommunicatie
01 = Speciale tekens
10 = Kleine letters
11 = Hoofdletter

00 (met als digit 1-9) : A tot I
01 (met als digit 1-9) : J tot R
10 (met als digit 2-9) : S tot Z
11 (met als digit 0-9) : cijfers

Voorbeeld :

1101 0100 is het karakter m

Zie ook :

http://en.wikipedia.org/wiki/EBCDIC

5.3 ASCII-CODE (AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE)

De ASCII code is een 7-bit code waarbij een 8e bit als controlebit wordt toegevoegd (zie later), gezien de meeste systemen op 8bit gebaseerd zijn.

		Most Significant Bits							
		000	**001**	**010**	**011**	**100**	**101**	**110**	**111**
Leas Significant Bits	**0000**	NULL	DC0	blank	0	@	P	`	p
	0001	SOM	DC1	!	1	A	Q	a	q
	0010	EOA	DC2	"	2	B	R	b	r
	0011	EOM	DC3	#	3	C	S	c	s
	0100	EOT	DC4	$	4	D	T	d	t
	0101	WRU	ERR	%	5	E	U	e	u
	0110	RU	SYNC	&	6	F	V	f	v
	0111	BELL	LEM	'	7	G	W	g	w
	1000	BKSP	S0	(	8	H	X	h	x
	1001	HT	S1	)	9	I	Y	i	y
	1010	LF	S2	*	:	J	Z	j	z
	1011	VT	S3	+	;	K	[	k	{
	1100	FF	S4	,	<	L	\	l	|
	1101	CR	S5	-	=	M	]	m	}
	1110	SOM	S6	.	>	N	^	n	ESC
	1111	SI	S7	/	?	O	_	o	DEL

NULL	Null / Idle	BKSP	Backspace	DC...	Device control
SOM	Start of message	HT	Horizontal Tab	ERR	Error
EOA	End of address	LF	Line feed	SYNC	Synchronous idle
EOM	End of message	VT	Vertical Tab	LEM	Logical end of media
EOT	End of transmission	FF	Form Feed	S...	Separator information
WRU	'Who are you ?'	CR	Carriage return	ESC	Escape
RU	'Are you ... ?'	SO	Shift out	DEL	Delete / Idle
BELL	Audible signal	SI	Shift in		

Dit wordt gebruikt bij PC's en teleprocessing :

DE standaard ASCII tabel bestaat uit 94 zichtbare tekens, de spatie en 33 stuurcodes. De stuurcodes representeren geen zichtbare tekens maar kunnen opdrachten geven aan uitvoerapparaten of geven informatie over de gegevens die worden verstuurd.

Voorbeeld : ASCII-code voor A = $(100\ 0001)_2 = (65)_{10}$

Zie ook :

https://nl.wikipedia.org/wiki/ASCII_%28tekenset%29
http://www.aciitable.com

5.4 UNICODE

Unicode is een internationale standaard voor de codering van binaire codes naar grafische tekens en symbolen, vergelijkbaar met de ASCII standaard. Deze standaard voorziet alle tekens en symbolen van alle geschreven talen van een nummer.

Deze standaard wordt onderhouden door het Unicode Consortium. In tegenstallign tot ASCII (alleen Engels) of Latin-1 (West-Europa) heeft Unicode als doel alle gebruikte schriften (zoals het griekse alfabet en Chinese karakters) te ondersteunen.

De standaard bevat momenteel meer dan 100.000 gestandardiseerde tekens en ongeveer 900.000 voor toekomstig gereserveerde codes.

De Unicode schrijft niets voor over de visuele representatie van een karakter.

Zie ook :

http://nl.wikipedia.org/wiki/Unicode

https://unicode-table.com/en/

6 FOUTAANWIJZENDE EN FOUTVERBETERENDE CODES

6.1 INLEIDING

Door de technologische vooruitgang worden computers(systemen) steeds betrouwbaarder : volledig foutloos werken ze echter niet.

Dus er is controle nodig bij gegevensoverdracht (transport) van gegevens :

- Tussen CPU en Memory, CPU en Cache
- Tussen computer en randapparatuur
- Tussen 2 deelnemers in een netwerk
- …

Controle berust op redundantie (overtolligheid), er worden meer tekens gebruikt dan eigenlijk strikt noodzakelijk is om de gegevens voor te stellen. Met andere woorden, er worden bits toegevoegd aan een bitrij. De waarde die de toegevoegde, redundante bits krijgen wordt berekend uit de waarde van de niet-redundante bits.

Bij het zenden van data wordt automatisch één of meerdere redundante bits berekend en toegevoegd. Bij ontvangst worden de redundante bits opnieuw berekend met behulp van de niet-redundante bits en worden de berekeningen vergeleken.

Zijn ze gelijk dan kan verdergegaan worden (eventueel met het verzenden van een volgende bitrij). Zijn ze ongelijk dan wordt er meestal overgegaan tot een retransmissie.
In deze cursus lichten we enkele controlemechanismen toe.

6.2 CONTROLEMECHANISMEN

6.2.1 PARITEITSCONTROLE

De meest gebruikte manier bij informatieoverdracht die teken per teken werkt is het toevoegen van één pariteitsbit per bitrij.

Men spreekt van **even pariteit** als het aantal 1 bits **even** is, wanneer het aantal bits **oneven** is spreekt met van **oneven pariteit**.

De pariteitscontrole kan zowel horizontaal als verticaal gebeuren of zelfs een combinatie van de 2 (block sum check) :

VERTICALE PARITEITSCONTROLE

Bij deze pariteitscontrole wordt 1 bit toegevoegd. In functie van de afspraken met de ontvanger spreekt men van **oneven** pariteit of **even** pariteit.

Enkele voorbeelden :

	Teken	**Toe te voegen pariteitsbit**	**Te versturen :**
Oneven pariteit	0 1 1 1 0 1 0 = '9' ASCII	**1** (er waren 4 1-bits en die worden nu op 5 gebracht = oneven)	0 1 1 1 0 1 0 **1**
	1 0 0 1 0 1 0 = 'J' ASCII	**0** (er waren 3 1-bits dus dient er geen 1-bit meer toegevoegd te worden).	1 0 0 1 0 1 0 **0**
Even pariteit	0 1 1 1 0 1 0 ='9' ASCII	**0** (er waren 4 1-bits dus dient er geen 1-bit meer toegevoegd te worden)	0 1 1 1 0 1 0 **0**
	1 0 0 1 0 1 0 ='J' ASCII	**1** (er waren 3 1-bits en die worden nu op 4 gebracht = even)	1 0 0 1 0 1 0 **1**

Dit systeem heeft wel nadelen :

Wanneer zich in de gegeven reeks een even aantal fouten voordoet, dan zal dit met verticale pariteitscontrole niet gedetecteerd worden.

HORIZONTALE PARITEITSCONTROLE

Er wordt hier gewerkt met een groep van te versturen tekens. Er wordt een reeks pariteitsbits meegestuurd met de groep aan tekens , waarbij de lengte van de pariteitsreeks even lang is als de lengte van 1 teken. De 1ste pariteitsbit is een controlebit op alle 1ste verstuurde bits, de 2de op de 2de reeks enz.

Voorbeeld : Te versturen bits : 1000011 10011111 1001111 1001100 0000011 => Controlebits :

Te versturen tekens (5)	1	0	0	0	0	1	1
	1	0	0	1	1	1	1
	1	0	0	1	1	1	1
	1	0	0	1	1	0	0
	0	0	0	0	0	1	1
Pariteitsteken bij **even** pariteit	**0**	**0**	**0**	**1**	**1**	**0**	**0**
Pariteitsteken bij **oneven** pariteit	**1**	**1**	**1**	**0**	**0**	**1**	**1**

Verstuurd bij even pariteit : 1000011 10011111 1001111 1001100 0000011 **0001100**

Verstuurd bij oneven pariteit : 1000011 10011111 1001111 1001100 0000011 **1110011**

Ook dit systeem heeft een nadeel:
Wanneer zich een even aantal bits op dezelfde rang in de verschillende tekens een verandert, dan zal de fout niet ontdekt worden door de controle.

HORIZONTALE EN VERTICALE PARITEITSCONTROLE : BLOCK SUM CHECK

Bij de Block Sum check worden de voorgaande technieken gecombineerd. Naast 1 controlebit per teken wordt een controlebit per positie in het teken meegestuurd, voor de verstuurde reeks.

We illustreren dit met een voorbeeld met even pariteitscontrole :

		Vert. par.
Groep tekens	0 1 1 1 1 1 1	**0**
	0 1 0 1 0 1 1	**0**
	0 0 1 1 0 0 0	**0**
	1 1 1 0 1 0 0	**0**
	1 1 0 0 1 0 1	**0**
	0 1 1 1 1 0 1	**1**
	1 1 0 0 0 1 0	**1**
Hor. Par.	**1 0 0 0 0 1 0**	**0**

In bovenstaand geval :

Te versturen bits : 0111111 0101011 0011000 1110100 1100101 0111101 1100010 =>

Wordt verstuurd : 0111111**0** 0101011**0** 0011000**0** 1110100**0** 1100101**0** 0111101**1** 1100010**1** **1000010****0**
(Controlebits)

Hier wordt even pariteitscontrole toegepast op de 2 pariteitsreeksen : beide reeksen dienen tot hetzelfde resultaat te leiden, zoniet zit er een fout in de pariteitsbit zelf. We kunnen via deze manier ook foutlocatie gaan toepassen.

Zoals eerder gesteld, gebeurt er bij de ontvangt van de gegevens opnieuw een berekening van de pariteitsbits. Deze herberekende reeksen worden dan vergeleken met de ontvangen pariteitsreeksen, waarna er kan vastgesteld worden of er eventuele verschillen zijn en waar die zich bevinden. => Zo kan de ontvanger de fout zelf gaan corrigeren zodat er geen retransmissie nodig is.

Voorbeeld :

								Ontvangen Vert. par.	Berekende Vert. par.
Groep tekens(7)	0	1	1	1	1	1	1	0	0
	0	1	0	1	0	1	1	0	0
	0	0	1	1	0	0	0	0	0
	1	1	1	0	1	1	0	0	1
	1	1	0	0	1	0	1	0	0
	0	1	1	1	1	0	1	1	1
	1	1	0	0	0	1	0	1	1
Ontvangen Hor. Par.	1	0	0	0	0	1	0	0	
Berekende Hor. Par.	1	0	0	0	0	0	0		1

In bovenstaand voorbeeld zien we een verschil tussen rij 4 op Verticale pariteit en kolom 6 op Horizontale pariteit. Hieruit kan geconcludeerd worden dat zich daar een fout bevindt. De meegestuurde databit 1 moet een 0 zijn.

HAMMING-CODE

De Hamming code is een foutverbeterende code (localisatie en verbetering) die tot stand komt door het gebruik van voldoende bijkomende bits en dito pariteitststesten. De toegevoegde bits worden de Hamming-bits genoemd.

Hamming (Richard) heeft aangetoond dat de totale woordlengte, bestaande uit het aantal Databits D en het aantal Hamming-bits H moet voldoen aan :

$D + H <= 2^H - 1$ (D+H = databits + Hamming-bits = totale verzonden bits)

Of: $2^H - H >= D + 1$

We kunnen met deze formule gaan berekenen hoeveel Hamming-bits er aan Databits moeten worden toegevoegd :

D = 1 Dus $2^H - H >= 1 + 1$ Dus $2^H - H >= 2$ Dus H=2 (2^2-2 =) 2>=2 Dus 2 Hamming bits toevoegen
D = 2 Dus $2^H - H >= 2 + 1$ Dus $2^H - H >= 3$ Dus H=3 (2^3-3 =) 5>=3 Dus 3 Hamming bits toevoegen
D = 3 Dus $2^H - H >= 3 + 1$ Dus $2^H - H >= 4$ Dus H=3 (2^3-3 =) 5>=4 Dus 3 Hamming bits toevoegen
D = 4 Dus $2^H - H >= 4 + 1$ Dus $2^H - H >= 5$ Dus H=3 (2^3-3 =) 5>=5 Dus 3 Hamming bits toevoegen
D = 5 Dus $2^H - H >= 5 + 1$ Dus $2^H - H >= 6$ Dus H=4 (2^4-4 =) 12>=6 Dus 4 Hamming bits toevoegen

Versturen van Hamming-gecodeerde databits

Algoritme : Het algoritme van de Hamming-code stelt :

1. Alle bitposities die tweemacht zijn worden gebruikt aan pariteitsbits (Posities 1,2,4,8,…).
2. Alle overige bitsposities worden gebruikt voor de databits. (Posities 3,5,6,7,9,…)
3. Iedere pariteitbit berekent de pariteit voor een aantal bits uit de Hammingbitnotatie.
    a. Positie 1 wordt bepaald door posities 3,5,7,9,… (Posities waarbij $2^0 = 1$)
    b. Positie 2 wordt bepaald door posities 3,6,7,10,11,… (Posities waarbij $2^1 = 1$)
    c. Positie 4 wordt bepaald door posities 5,6,7,12,12,… (Posities waarbij $2^2 = 1$)
    d. Enz…

We verduidelijken met een uitgewerkt voorbeeld:

We wensen een gegevenreeks van 4 Databits te versturen (D_0,D_1,D_2 en D_3). Zoals hierboven aangetoond zullen we bijgevolg 3 Hammingbits moeten toegevoegd worden. (H_0,H_1 en H_2) waardoor er in totaal 7 bits zullen worden verstuurd.

Te versturen reeks : 0001

D_0	D_1	D_2	D_3
0	0	0	1

Posities van de Hamming bits (allen en 2macht, dus 2^0,2^1,2^3,2^4,…). In de overige locaties worden dan de databits geplaatst.

Positie	1	2	3	4	5	6	7
2macht ?	2^0	2^1		2^2			
(Binair)	001	010	011	100	101	110	111
Data-Hamming	H_0	H_1	D_0	H_2	D_1	D_2	D_3
Teken			**0**		**0**	**0**	**1**

Om de waardes van de Hamming bit te bepalen, kijken we welke bits er gecontroleerd worden. Hiervoor kunnen we de binaire notatie gebruiken van de posities.

De Hamming bit is de Positie waar in binaire notaties slechts één 1 voorkomt, en deze zal de posities gaan controleren waar op dezelfde plaats ook een 1 voorkomt in binaire notatie.

De waarde van de Hamming bit wordt gevonden door even pariteitstoepassing.

Positie	1	2	3	4	5	6	7
(Binair)	001	010	011	100	101	110	111
Data-Hamming	H_0	H_1	D_0	H_2	D_1	D_2	D_3
Teken			**0**		**0**	**0**	**1**

Hamming bit op positie 1 H_0 wordt bepaald uit posities 3,5 en 7. Om even pariteit te bekomen wordt deze dus : 1.
Hamming bit op positie 2 H_1 wordt bepaald uit posities 3,6 en 7. Om even pariteit te bekomen wordt deze dus : 1.
Hamming bit op positie 4 H_2 wordt bepaald uit posities 5,6 en 7. Om even pariteit te bekomen wordt deze dus : 1.

Het resultaat :

Positie	1	2	3	4	5	6	7
(Binair)	001	010	011	100	101	110	111
Data-Hamming	H_0	H_1	D_0	H_2	D_1	D_2	D_3
Teken	**1**	**1**	**0**	**1**	**0**	**0**	**1**

Gevolg :

Data 0001 wordt verstuurd als **1101**001 met 3 **Hammingbits**.

Ontvangen en controleren van Hamming-gecodeerde databits

Bij ontvangst van een reeks worden een aantal EXOR-bewerkingen uitgevoerd ter verificatie van de gestuurde codering evenveel als er Hamming-bits aanwezig zijn.

EXOR staat voor Exclusive OR en wordt voorgesteld als ⊕

x	y	A⊕B
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

Het resultaat van deze controles worden samengebracht en vormen de **Pointer**.

We gebruiken het voorgaand voorbeeld :

Positie	1	2	3	4	5	6	7
(Binair)	001	010	011	100	101	110	111
Data-Hamming	H_0	H_1	D_0	H_2	D_1	D_2	D_3
Teken	**1**	**1**	**0**	**1**	**0**	**0**	**1**

Pointer	P_0	P_1	P_2
Waarde	?	?	?

P_0 is een pariteitscontrole van de bits die H_0 bewaakt. => $P_0 = H_0 \oplus D_0 \oplus D_1 \oplus D_3 = 1 \oplus 0 \oplus 0 \oplus 1 = 0$
P_1 is een pariteitscontrole van de bits die H_1 bewaakt. => $P_1 = H_1 \oplus D_0 \oplus D_2 \oplus D_3 = 1 \oplus 0 \oplus 0 \oplus 1 = 0$
P_2 is een pariteitscontrole van de bits die H_1 bewaakt. => $P_2 = H_2 \oplus D_1 \oplus D_2 \oplus D_3 = 1 \oplus 0 \oplus 0 \oplus 1 = 0$

Waardoor :

Pointer	P_0	P_1	P_2
Waarde	**0**	**0**	**0**

Bij een foutloze datatransmissie is $P_0 = P_1 = P_2$ ($=P_4 = P_5$ …) = 0

Voorbeelden

Posities :	1	2	3	4	5	6	7	
	H_0	H_1	D_0	H_2	D_1	D_2	D_3	
Te versturen data								
0101	0	1	0	0	1	0	1	Ontvangen
EXOR controle								**Resultaat**
P_0	0		0		1		1	**0**
P_1		1	0			0	1	**0**
P_2				0	1	0	1	**0**

Data = 0101 => Verstuurd = 010010 => Na controle : Data = 0101

Posities :	1	2	3	4	5	6	7	
	H_0	H_1	D_0	H_2	D_1	D_2	D_3	
Te versturen data								
0100	1	0	0	1	1	0	0	Ontvangen
EXOR controle								**Resultaat**
P_0	1		0		1		0	**0**
P_1		0	0			0	0	**0**
P_2				1	1	0	0	**0**

Data : 0100 => Verstuurd : 1001100 => Na controle Data : 0100

Wanneer een fout optreedt, dan duidt de pointer de foutieve bit in het geteste woord zodat de fout verbeterd kan worden zonder dat hierbij hertransmissie nodig is.

We illustreren dit met onderstaand voorbeeld :

Data : 0100 => Data verstuurd : 1001100 => Data ontvangen : 1001110
Door middel van de pointer kan worden aangetoond dat D_2 in bovenstaand geval verkeerd is.

Posities :	1	2	3	4	5	6	7	
	H_0	H_1	D_0	H_2	D_1	D_2	D_3	
Te versturen data								
0100	1	0	0	1	1	1	0	Ontvangen
EXOR controle								**Resultaat**
P_0	1		0		1		0	**0**
P_1		0	0			1	0	**1**
P_2				1	1	1	0	**1**

De pointer geeft ook onmiddellijk de locatie van de fout weer : $(P_2P_1P_0)_2 = (110)_2 = (6)_{10}$ => Fout in positie 6

Data : 0100 => Data verstuurd : 1001100 => Data ontvangen : 1001110
=> Na controle : Datacorrectie op positie 6 : 1001100 => Data : 0100 !

Facultatief

De Hammingcode ligt aan de basis van vele ECC-chips (Error Check and Correct) die niet alleen bij gegevenstransmissie maar ook in hooggeïntegreerde geheugens veel gebruikt wordt.