Datarepresentatie

Dit verslag werd opgesteld door:

Naam: Abdellah El Moussaoui
Studentennummer: 20246031
Email adres:
abdellah.elmoussaoui@student.uantwerpen.be

1. Zet deze positieve getallen om naar base 10

a)

10 9 8 7 6 5 4 3 2 1 0

$$(10110111100)_2 = 0 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4 + 1 \cdot 2^5 + 0 \cdot 2^6 + 1 \cdot 2^7 + 1 \cdot 2^8 + 0 \cdot 2^9 + 1 \cdot 2^{10}$$

$$= 4 + 8 + 16 + 32 + 128 + 256 + 1024$$

$$= (1468)_{10}$$

b)

$$(3A6E)_{16} = E \cdot 16^0 + 6 \cdot 16^1 + A \cdot 16^2 + 3 \cdot 16^3$$

$$= 14 + 96 + 10 \cdot 256 + 12288$$

$$= 14 + 96 + 2560 + 12288$$

$$= (14958)_{10}$$

c)

12 11 10 9 8 7 6 5 4 3 2 1 0

$$(1110000011010)_2 = 0 \cdot 2^0 + 2^1 + 0 + 2^3 + 2^4 + 2^{10} + 2^{11} + 2^{12}$$

$$= 2 + 8 + 16 + 1024 + 2048 + 4096$$

$$= (7194)_{10}$$

d)

$$(164)_8 = 4 \cdot 8^0 + 6 \cdot 8^1 + 1 \cdot 8^2$$

$$= 4 + 48 + 64$$

$$= (116)_{10}$$

e)

$$(1004)_6 = 4 \cdot 6^0 + 0 + 0 + 1 \cdot 6^3$$

$$= 4 + 216$$

$$= (220)_{10}$$

2. Zet om naar base 10

a)

$(11101011)_2$ (two's complement)

* deze getal is negatief omdat de MSB ervan 1 is.
* de positieve getal is:

$00010100 + 1 = (00010101)_2$

* $(00010101)_2 = 1 \cdot 2^0 + 1 \cdot 2^2 + 1 \cdot 2^4$

$= 1 + 4 + 16$

$= (21)_{10}$

* Dus:

$(11101011)_2 = (-21)_{10}$
(two's complement)

b)

$(1111)_2$ (two's complement)

* deze getal is negatief omdat de MSB 1 is.
* de positieve getal is:

$0000 + 1 = (0001)_2 = (1)_{10}$

" We krijgen de positieve vorm van een two's complement negatieve getal door 1 op te tellen bij de one's complement "

* Dus:

$(1111)_2$ (two's complement) $= (-1)_{10}$

c)

$(0.213)_4 = 0 \cdot 4^0 + 2 \cdot 4^{-1} + 1 \cdot 4^{-2} + 3 \cdot 4^{-3}$

$= 2 \cdot \frac{1}{4} + 1 \cdot \frac{1}{16} + 3 \cdot \frac{1}{64}$

$$= \frac{1}{2} + \frac{1}{16} + \frac{3}{64}$$

$$= \frac{32 + 4 + 3}{64}$$

$$= \frac{39}{64}$$

$(0.213)_4 = (0.609375)_{10}$

d)

$(0.987)_{15} = 0 \cdot 15^0 + 9 \cdot 15^{-1} + 8 \cdot 15^{-2} + 7 \cdot 15^{-3}$

$$= 9 \cdot \frac{1}{15} + 8 \cdot \frac{1}{15^2} + 7 \cdot \frac{1}{15^3}$$

$$= \frac{9 \cdot 15^2 + 8 \cdot 15 + 7}{15^3}$$

$$= \frac{2025 + 120 + 7}{3375}$$

$$= \frac{2152}{3375}$$

$(0.987)_{15} = (0.63762962)_{10}$

3. Zet om naar base 2.

a)

$(2021)_{10} = (11111100101)_2$

2	2021	1
2	1010	0
2	505	1
2	252	0
2	126	0
2	63	1
2	31	1

2	15	1
2	7	1
2	3	1
2	1	1
2	0	

b) We weten dat $(6)_8 = (110)_2$, dus
$(666)_8 = (110\ 110\ 110)_2$
* voor elke octal cijfer geven we 3 bits.

c)
* We weten dat
$(7)_{16} = (0111)_2$
$(D)_{16} = (1101)_2 = (13)_{10}$
$(B)_{16} = (11)_{10} = (1011)_2$
$(1)_{16} = (0001)_2$

* Hier geven we 4 bits voor elke hexadecimal cijfer (digit)

* Dus:
$(1BD7)_{16} = (0001101111010111)_2$

d)
$(7.75)_{10} = (111.11)_2$

2	7	1
2	3	1
2	1	1
2	0	

- $0.75 \times 2 = 1.50$
- $0.50 \times 2 = 1.00$
- $0.00 \times 2 = 0.00$

e)
* We weten dat:
$(4)_{16} = (0100)_2$
$(1)_{16} = (0001)_2$
$(D)_{16} = (13)_{10} = (1101)_2$
$(A)_{16} = (10)_{10} = (1010)_2$

Dus :

$(AD14)_{16} = (1010\ 1101\ 0001\ 0100)_2$

4. Zet om naar base 2.

a) $(-104)_{10}$

1- Signed Magnitude

* $(104)_{10} = (001101000)_2$

2	104	0
2	52	0
2	26	0
2	13	1
2	6	0
2	3	1
2	1	1
2	0	

* Om de signed magnitude van een negatieve getal voor te stellen, hoeven we alleen de MSB van de positieve getal op 1 te zetten.

* Dus :

$(-104)_{10} = (101101000)_2$ (signed magni)

2- 1's complement

* $(104)_{10} = (001101000)_2$

* Dus :

$(-104)_{10} = (110010111)_2$ (1's complement)

* Om de 1's complements te krijgen, hoeven we alleen de bits te inverteren.

3- Two's complement

* Om de 2's complement te krijgen, moeten we 1 optellen bij de 1's complement.
* Dus :

$(-104)_{10}$ = 11001011 + 1

$(-104)_{10} = (11011000)_2$ (2's complement)

4- Excess 128

* Om de XS-128 van een getal te krijgen, moeten we eerst 128 optellen bij de getal.

$(-104 + 128)_{10} = (24)_{10}$

* Dan zetten we het resutaat om naar 2 base

256	128	64	32	16	8	4	2	1
0	0	0	0	1	1	0	0	0

$(-104)_{10} = (000011000)_2$ (XS-128)

b) $(-69)_{10}$

1- SM

64	32	16	8	4	2	1
1	0	0	0	1	0	1

$(-69)_{10} = (101000101)_2$ (Signed magnitude)

" $(69)_{10} = (001000101)_2$ "

2- 1's complement

$(69)_{10} = (001000101)_2$.

$(-69)_{10} = (110111010)_2$ (1's comp)

4. Zet om naar base 2. Stel de negatieve getallen voor met 9 bits in signed magnitude (1), one's complement (2), two's complement (3) en excess 128 (4).

b) $(-69)_{10}$

1- Signed Magnitude

$(69)_{10} = (001000101)_2$

2	69	1
2	34	0
2	17	1
2	8	0
2	4	0
2	2	0
2	1	1
2	0	

Om de signed magnitude van het getal te krijgen, hoeven we alleen de MSB op 1 te zetten.

$(-69)_{10} = (101000101)_2$ (signed magnitude)

2- One's Complement

We weten dat $(69)_{10} = (001000101)_2$

Om de 1's complement te krijgen, moeten we gewoon de bits inverteren.

$(-69)_{10} = (110111010)_2$ (1's complement)

3- Two's Complement

Om de 2's complement te krijgen, moeten we gewoon 1 optellen bij de one's complement.

$(-69)_{10} = (110111010)_2$ (1's complement) + 1

$(-69)_{10} = (110111011)_2$ (2's complement)

4- Excess 128

- Om de XS 128 te krijgen, moeten we eerst 128 optellen bij het decimale getal.
  $(-69 + 128)_{10} = (59)_{10}$
- Dan zetten we het resultaat om naar 2 base.

$(59)_{10} = (000111011)_2$

2	59	1
2	29	1
2	14	0
2	7	1
2	3	1
2	1	1
2	0	

- Dan krijgen we de excess 128 vorm:
  $(-69)_{10} = (000111011)_2$ (excess 128)

c) $(-128)_{10}$

1- Signed Magnitude

$(128)_{10} = (010000000)_2$

2	128	0
2	64	0
2	32	0
2	16	0
2	8	0
2	4	0
2	2	0
2	1	1
2	0	

Om de signed magnitude van het getal te krijgen, hoeven we alleen de MSB op 1 te zetten.

$(-128)_{10} = (110000000)_2$ (signed magnitude)

2- One's Complement

We weten dat $(128)_{10} = (010000000)_2$

Om de 1’s complement te krijgen, moeten we gewoon de bits inverteren.

$(-128)_{10} = (101111111)_2$ (1’s complement)

3- Two’s Complement

Om de 2’s complement te krijgen, moeten we gewoon 1 optellen bij de one’s complement.

$(-128)_{10} = (101111111)_2$ (1’s complement) + 1

$(-128)_{10} = (110000000)_2$ (2’s complement)

4- Excess 128

- Om de XS 128 te krijgen, moeten we eerst 128 optellen bij het decimale getal.
  $(-128 + 128)_{10} = (0)_{10}$
- Dan zetten we het resultaat om naar 2 base.

$(0)_{10} = (000000000)_2$

2	0	0
2	0	

- Dan krijgen we de excess 128 vorm:
  $(-128)_{10} = (000000000)_2$ (excess 128)

d) $(-3D)_{16}$

1- Signed Magnitude

$(3D)_{16} = (000111101)_2$

Hex	Decimal	Binaire
3	3	0011
D	13	1101

Om de signed magnitude van het getal te krijgen, hoeven we alleen de MSB op 1 te zetten.

$(-3D)_{16} = (100111101)_2$ (signed magnitude)

2- One's Complement

We weten dat $(3D)_{10} = (000111101)_2$

Om de 1's complement te krijgen, moeten we gewoon de bits inverteren.

$(-3D)_{16} = (111000010)_2$ (1's complement)

3- Two's Complement

Om de 2's complement te krijgen, moeten we gewoon 1 optellen bij de one's complement.

$(-3D)_{16} = (111000010)_2$ (1's complement) + 1

$(-3D)_{16} = (111000011)_2$ (2's complement)

4- Excess 128

- Om de **XS 128** te krijgen, moeten we eerst 128 optellen bij het decimale getal.
  $(-3D)_{16} + (128)_{10} = (-(3*16^1 + 13*16^0) + 128)_{10} = (-61 + 128)_{10} = (67)_{10}$

- Dan zetten we het resultaat om naar 2 base.

$(67)_{10} = (001000011)_2$

2	67	1
2	33	1
2	16	0
2	8	0
2	4	0
2	2	0
2	1	1
2	0	

- Dan krijgen we de excess 128 vorm:
  $(-3D)_{16} = (001000011)_2$ (excess 128)

5. Voor de onderstaande enkele precisie IEEE-754 bitpatronen, geef de numerieke waarde als een significant in base 2 met een exponent

a) 0 10001110 00111010000000000000000

- **Sign bit**: 0 (positief)
- **Mantisse**: 00111010000000000000000
- **Exponent**: $(10001110)_2$
  $(10001110)_2 = 128 + 8 + 4 + 2 = (142)_{10}$

128	64	32	16	8	4	2	1
1	0	0	0	1	1	1	0

We hebben **254 ≥ exponent ≥ 1,** dan:

0 10001110 00111010000000000000000 =

$(-1)^0 + 1.00111010000000000000000 * 2^{142-127}$

Dus

0 10001110 00111010000000000000000 = **+1.0011101 · 2^{15}**

b) 1 00111100 10110000000000000000000

- **Sign bit**: 1 (negatief)
- **Mantisse**: 10110000000000000000000
- **Exponent**: $(00111100)_2$

$00111100)_2$ = 32 + 16 + 8 + 4 = $(60)_{10}$

128	64	32	16	8	4	2	1
0	0	1	1	1	1	0	0

We hebben **254 ≥ exponent ≥ 1,** dan:

1 00111100 10110000000000000000000 =

$(-1)^1$ + 1.10110000000000000000000 * 2^{60-127}

Dus

1 00111100 10110000000000000000000 = **-1.1011 · 2^{-67}**

c) 0 11111111 00000000000000000000000

- **Sign bit**: 0 (positief)
- **Mantisse**: 00000000000000000000000
- **Exponent**: $(11111111)_2$

$11111111)_2$ = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = $(255)_{10}$

128	64	32	16	8	4	2	1
1	1	1	1	1	1	1	1

De exponent hier is gelijk aan 255, de mantisse is 0 en het sign bit is 0. Dus de waarde van dit IEEE-754 getal is **+∞**

d) 0 00000000 00101011100000000000000

- **Sign bit**: 0 (positief)
- **Mantisse**: 00101011100000000000000
- **Exponent**: $(00000000)_2$

De exponent hier is gelijk aan 0, maar de mantisse is verschillend van 0. Dit is een representatie in fractionele vorm.

Dus
0 00000000 00101011100000000000000 = $(-1)^0 + 0.001010111 * 2^{-126}$

= **$+1.010111 \cdot 2^{-129}$**

e) 1 00010100 11100110000000000000000

- **Sign bit**: 1 (negatief)
- **Mantisse**: 11100110000000000000000
- **Exponent**: $(00010100)_2$

$00010100)_2 = 16 + 4 = (20)_{10}$

128	64	32	16	8	4	2	1
0	0	0	1	0	1	0	0

We hebben **254 ≥ (E) ≥ 1,** dan:
1 00010100 11100110000000000000000 =
$(-1)^1 + 1.11100110000000000000000 * 2^{20-127}$

Dus
1 00010100 11100110000000000000000 = **$-1.1110011 \cdot 2^{-107}$**

f) 0 11111111 11010100010001010100010

Hier hebben we de exponent gelijk aan 255 en de mantisse is niet gelijk aan 0.

Dus dit is geen getal **(NaN)**.

g) 0 00001011 01101000000000000000000

- **Sign bit**: 0 (positief)
- **Mantisse**: 01101000000000000000000
- **Exponent**: $(00001011)_2$

$00001011)_2 = 8 + 2 + 1 = (11)_{10}$

128	64	32	16	8	4	2	1
0	0	0	0	1	0	1	1

We hebben **254 ≥ (E) ≥ 1,** dan:
0 00001011 01101000000000000000000 =
$(-1)^0 + 1.01101000000000000000000 * 2^{11-127}$

Dus
0 00001011 01101000000000000000000 = **$+1.01101 * 2^{-116}$**

6. Stel deze getallen voor in het IEEE-754 formaat met enkele precisie.

a) $(2078.25)_{10}$

- We moeten dit getal eerst naar binair omzetten.

2048	1024	512	256	128	64	32	16	8	4	2	1
1	0	0	0	0	0	0	1	1	1	1	0

0.25 * 2 = 0.5

$0.50 * 2 = 1.0$

$0.00 * 2 = 0.0$

Dus

$(2078.25)_{10} = (100000011110.01)_2 = (\mathbf{1.0000001111001 * 2^{11}})_2$

- Het IEEE-754 formaat met enkele precisie wordt als volgt geschreven: S EEEEEEEE MMMMMMMMMMMMMMMMMMMMMMM
    - *S*: het sign bit (1 bit)
    - *E*: de exponent in excess-127 (8 bits)
    - *M*: de mantisse (23 bits)
- Nu gaan we dit getal voorstellen in het IEEE-754 formaat met enkele precisie $\mathbf{1.0000001111001 * 2^{11}}$
    - S = 0 (omdat het getal positief is)
    - $E = 11 + 127 = (138)_{10} = (10001010)_2$

128	64	32	16	8	4	2	1
1	0	0	0	1	0	1	0

    - M = 00000011110010000000000

- Dus het resultaat is:
  0 10001010 00000011110010000000000

b) $(2010)_{10}$

- We moeten dit getal eerst naar binair omzetten.

2048	1024	512	256	128	64	32	16	8	4	2	1
0	1	1	1	1	1	0	1	1	0	1	0

Dus

$(2010)_{10} = (11111011010)_2 = (\mathbf{1.111101101 * 2^{10}})_2$

- Nu gaan we dit getal voorstellen in het IEEE-754 formaat met enkele precisie $\mathbf{1.111101101 * 2^{10}}$

- S = 0 (omdat het getal positief is)
- E = 10 + 127 = $(137)_{10}$ = $(10001001)_2$

128	64	32	16	8	4	2	1
1	0	0	0	1	0	0	1

- M = 11110110100000000000000

- Dus het resultaat is:
  0 10001001 11110110100000000000000

c) NaN

Om NaN (not a number) in IEEE-754 voor te stellen, moet de exponent gelijk zijn aan $(255)_{10}$, en de mantisse verschillend van 0.

Dus
NaN = 0 11111111 01100110100000000000000

d) $(-42.666)_{10}$

- We moeten dit getal eerst in de positieve vorm naar binair omzetten.

32	16	8	4	2	1
1	0	1	0	1	0

0.666 * 2 = 1.332
0.332 * 2 = 0.664
0.664 * 2 = 1.328
0.328 * 2 = 0.656
0.656 * 2 = 1.312
0.312 * 2 = 0.624
0.624 * 2 = 1.248
0.248 * 2 = 0.496

0.469 * 2 = 0.992

0.992 * 2 = 1.984

0.984 * 2 = 1.968

0.968 * 2 = 1.936

…

Dus
$(42.666)_{10}$ = $(101010.10101010011111111)_2$ =
$(1.01010101010100111111111 * 2^5)_2$

- Nu gaan we dit getal voorstellen in het IEEE-754 formaat met enkele precisie.
  - S = 1 (omdat het getal negatief is)
  - E = 5 + 127 = $(132)_{10}$ = $(10000100)_2$

128	64	32	16	8	4	2	1
1	0	0	0	0	1	0	0

  - M = 01010101010100111111111

- Dus het resultaat is:
  1 10000100 01010101010100111111111

e) +∞

Om **+∞** in IEEE-754 voor te stellen, moet de exponent gelijk zijn aan $(255)_{10}$, en de mantisse en sign bit gelijk aan 0.

+∞ = 0 11111111 00000000000000000000000

f) +0

Om +0 in IEEE-754 voor te stellen, moet de exponent gelijk zijn aan $(0)_{10}$, en de mantisse en het sign bit moeten ook gelijk zijn aan 0.

Dus
+0 = 0 00000000 00000000000000000000000

g) $(1.11 * 2^{-129})_2$ (denormalized)

- We weten dat om een denormalized getal voor te stellen in IEEE-754, de exponent gelijk moet zijn aan 0. De exponent in de voorstelling van een denormalized getal in IEEE-754 is altijd gelijk aan -126.
- Dan hebben we: $\mathbf{1.11 * 2^{-129} = 0.00111 * 2^{-126}}$
- Dus het resultaat is
  0 00000000 00111000000000000000000

h) $(333.666)_{10}$

- We moeten dit getal eerst in de positieve vorm naar binair omzetten.

256	128	64	32	16	8	4	2	1
1	0	1	0	0	1	1	0	1

0.666 * 2 = 1.332
0.332 * 2 = 0.664
0.664 * 2 = 1.328
0.328 * 2 = 0.656
0.656 * 2 = 1.312
0.312 * 2 = 0.624
0.624 * 2 = 1.248
0.248 * 2 = 0.496
0.469 * 2 = 0.992

0.992 * 2 = 1.984

0.984 * 2 = 1.968

0.968 * 2 = 1.936

...

Dus
$(333.666)_{10} = (101001101.101010100111111)_2 =$
$\mathbf{(1.01001101101010100111111 * 2^8)_2}$

- Nu gaan we dit getal voorstellen in het IEEE-754 formaat met enkele precisie.
    - S = 0 (omdat het getal positief is)
    - E = 8 + 127 = $(135)_{10}$ = $(10000111)_2$

128	64	32	16	8	4	2	1
1	0	0	0	0	1	1	1

    - M = 01001101101010100111111

- Dus het resultaat is:
  0 10000111 01001101101010100111111

7. Stel dat we een 15-bit normalised floating point formaat gebruiken in base 8, met een sign bit, gevolgd door een 5-bit exponent met bias, en tenslotte drie base 8 cijfers.

De getallen in 15-bit normalised floating point formaat worden als volgt geschreven:
S EEEEE MMMMMMMMM

(a) Bepaal de bias voor de exponent

We weten dat de range van getallen in 5-bit two's complement is van -16 tot 15 (-2^4 tot 2^4-1), het kleinste getal is -16.

Dus de bias hier is 16.

De exponenten worden dus als volgt gepresenteerd:

Dec	-16	-15	-14	-13	…	12	13	14	15
2’s com	10000	10001	10010	10011	…	01100	01101	01110	01111
Dec+16 excess16	0	1	2	3	…	28	29	30	31
Bin	00000	00001	00010	00011	…	11100	11101	11110	11111

(b) Stel het getal -142_{10} voor in het nieuwe formaat als een binaire string.

We moeten dit getal eerst in de positieve vorm naar base 8 omzetten.

64	8	1
2	1	6

$142_{10} = 2 * 8^2 + 1 * 8^1 + 6 * 8^0 = 216_8$

We gebruiken de normalisatie vorm waarbij de punt geplaatst wordt voor het eerste niet-nul cijfer, dus:

$\mathbf{142_{10} = 216_8 = 0.216 * 8^3}$

- S = 1 (omdat het getal negatief is)
- E = 3 + 16 = 19 = $\mathbf{10011_2}$

16	8	4	2	1
1	0	0	1	1

- M = 216_8 = 010 001 110_2

Dus het resultaat is

1 10011 010 001 110

(c) Wat is de grootste mogelijke error dat we in dit formaat kunnen uitdrukken?

We kunnen de grootste mogelijke error tellen met deze formule $\frac{1}{2^f}$ =

f: staat voor de fractional bits, hier f = 3

Dus de grootste mogelijke error is

$$\frac{\mathbf{1}}{\mathbf{8^3}} = \frac{\mathbf{1}}{\mathbf{512}}$$

(d) Wat is de kleinst mogelijke afstand tussen twee opeenvolgende getallen?

We kunnen de kleinst mogelijke afstand tellen met deze formule b^m x b^{-s}

s: staat voor het aantal cijfers in de base = 3
m: staat voor het kleinste mogelijke exponent = -16

Dus de kleinst mogelijke afstand is
8^{-16} x 8^{-3} = 8^{-19}

8. Schrijf een Python programma encodings.py dat, gebruikmakend van de files module, het volgende doet:

(a) Lees het gegeven bestand input.txt in met de correcte encoding.

Eerst moeten we **import files** om de modules van files te kunnen gebruiken.

Om het bestand te lezen, kunnen we gebruik maken van
inhoud = files.read_file('input.txt', files.UTF_8)

(b) Schrijf de inhoud van de file weg naar een bestand, en maak daarbij gebruik van de UTF-16 encoding. Noem dit bestand text_in_UTF_16.txt.

We kunnen dat doen gewoon met de volgende code:
files.write_file('text_in_UTF_16.txt', inhoud, files.UTF_16)

(c) Zet alle karakters om naar hun overeenkomstige code points. Sla deze op in een bestand genaamd code_points.txt.

We kunnen eerste de code point van alle karakters krijgen met deze code:
code_points = ' '.join([str(ord(char)) for char in inhoud])
Ik zet een spatie tussen de code points om niet te mixen.

Dan kunnen we die opslaan in een bestand zo:
files.write_file('code_points.txt', code_points, files.ASCII)
(Ik gebruik ASCII omdat die zijn gewoon normale getallen)

(d) Converteer de code points naar hun overeenkomstige HTML code. Hou rekening met line breaks! Noem dit bestand text_in_HTML.html.

We maken een for loop die door alle karakters in de inhoud loopt. Als het karakter een line break is, kunnen we dat in HTML presenteren met **
. Als het een codepunt is, kunnen we het overeenkomstige karakter van dat codepunt weergeven met **&#{CODE_POINT};.

```
html_content = ''

for char in inhoud:

    if char == '\n':
```

```
        html_content += '<br>'
    else:
        html_content += f'&#{ord(char)};'
files.write_html_file('text_in_HTML.html', html_content)
```

De hele code staat in de bestand encodings.py.