1. Sistem Angka

Manusia menggunakan sistem bilangan *desimal* (basis 10) dan *duodesimal (basis 12) untuk menghitung dan mengukur (mungkin karena kita memiliki 10 jari tangan dan dua jempol kaki).* Komputer menggunakan sistem bilangan *biner* (basis 2), karena terbuat dari komponen digital biner (dikenal sebagai transistor) yang beroperasi dalam dua keadaan - hidup dan mati. Dalam komputasi, kami juga menggunakan sistem bilangan *heksadesimal* (basis 16) atau *oktal (basis 8), sebagai bentuk ringkas* untuk merepresentasikan bilangan biner.

1.1 Sistem Bilangan Desimal (Basis 10).

Sistem bilangan desimal mempunyai sepuluh lambang: 0, 1, 2, 3, 4, 5, 6, 7, 8, dan 9, disebut *angka* **d**. Ini menggunakan *notasi posisi* . Artinya, angka penting terkecil (digit paling kanan) atau disebut dengan LSB (Least Significant Bit) adalah orde 10^0(satuan atau satuan), angka paling kanan kedua adalah orde 10^1(puluhan), angka paling kanan ketiga adalah orde dari 10^2(ratusan), dan seterusnya, yang ^menunjukkan eksponen. Angka yang paling penting dalam menentukan orde adalah angka paling kanan atau disebut dengan MSB (Most Significant Bit) , misalnya,

```
735 =  700 + 30 + 5
    = 7×10^2 + 3×10^1 + 5×10^0
```

1.2 Sistem Bilangan Biner (Basis 2)

Sistem bilangan biner memiliki dua simbol: 0 dan 1, disebut *bit* . Ini juga merupakan *notasi posisi* , misalnya,

```
10110B = 10000B  + 00000B +  00100B +  00010B  + 00000B
       =  1×2^4   + 0×2^3   +  1×2^2  +  1×2^1   + 0×2^0
```

Penulisan bilangan biner dalam beberapa buku ditulis dengan akhiran **B**. Beberapa bahasa pemrograman menunjukkan bilangan biner dengan awalan **0b** atau **0B** (misalnya, **0b**1001000), atau awalan **b** dengan bit yang dikutip (misalnya, **b'10001111'**).

Digit biner disebut *bit* . Delapan bit disebut *byte* (mengapa satuan 8-bit? Mungkin karena), yaitu 8 $=2^3$

1.3 Sistem Bilangan Heksadesimal (Basis 16).

Sistem bilangan heksadesimal menggunakan 16 simbol: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, dan F, disebut *digit hex*. Ini adalah *notasi posisi*, misalnya,

```
A3EH = A00H    + 030H   +  00EH
     = 10×16^2 + 3×16^1 + 14×16^0
```

Beberapa buku teks menuliskan heksadesimal (singkatnya, hex) dengan akhiran H. Beberapa bahasa pemrograman menunjukkan bilangan hex dengan awalan **0x** atau **0X**(misalnya, **0x**1A3C5F), atau awalan **x** dengan digit hex yang dikutip (misalnya, **x'C3A4D98B'**).

Setiap digit heksadesimal disebut juga *digit heksa*. Pada umumnya, bahasa pemrograman menerima huruf kecil '**a**'dan '**f**' maupun huruf '**A**'besar '**F**'.

Komputer menggunakan sistem biner dalam operasi internalnya, karena komputer dibangun dari komponen elektronik digital biner dengan 2 status - hidup dan mati. Namun, menulis atau membaca rangkaian bit biner yang panjang merupakan hal yang rumit dan rawan kesalahan. Contoh : cobalah membaca string biner ini

1011 0011 0100 0011 0001 1101 0001 1000B,

yang nilainya sama dengan heksadesimal

B 3 4 3 1 D 1 8H

Sistem heksadesimal digunakan sebagai bentuk *ringkas* atau *singkatan* untuk bit biner. Setiap digit hex setara dengan 4 bit biner, yaitu singkatan dari 4 bit, sebagai berikut:

Heksadesimal	Biner	Desimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6

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

1.4 Konversi dari Heksadesimal ke Biner

Ganti setiap digit hex dengan 4 bit yang setara (seperti yang tercantum dalam tabel di atas), misalnya,

A3C5H = 1010 0011 1100 0101B
102AH = 0001 0000 0010 1010B

1.5 Konversi dari Biner ke Heksadesimal

Mulai dari bit paling kanan (bit paling tidak signifikan), ganti setiap kelompok 4 bit dengan digit hex yang setara (tambahkan bit paling kiri dengan nol jika perlu), misalnya,

1001001010B = 0010 0100 1010B = 24AH
100010110010111B = 0010 0010 1100 1011B = 22CBH

Penting untuk dicatat bahwa bilangan heksadesimal memberikan *bentuk ringkas* atau *singkatan* untuk mewakili bit biner.

1.6 Konversi dari Basis R ke Desimal (Basis 10)

Diberikan bilangan dasar *n* -digit *r*: (basis r), persamaan desimalnya diberikan oleh:$d_{n-1}d_{n-2}d_{n-3}...d_2d_1d_0$

$d_{n-1} \times r^{n-1} + d_{n-2} \times r^{n-2} + \ldots + d_1 \times r^1 + d_0 \times r^0$

Misalnya,

A1C2H = 10×16^3 + 1×16^2 + 12×16^1 + 2
= 4096 + 256 + 192
= 41410 (basis 10)

10110B = 1×2^4 + 1×2^2 + 1×2^1
= 16 + 4 + 2
= 22 (basis 10)

1.7 Konversi dari Desimal (Basis 10) ke Basis R

Gunakan pembagian/sisa berulang. Misalnya,

Untuk mengkonversi 261(basis 10) ke heksadesimal:
261/16 => hasil bagi=16 sisa=5
16/16 => hasil bagi=1 sisa=0
1/16 => hasil bagi=0 sisa=1 (hasil bagi=0 berhenti)
Jadi, 261D = 105H (Kumpulkan digit hex dari sisanya dalam urutan terbalik)

Prosedur di atas sebenarnya berlaku untuk konversi antara 2 sistem dasar. Misalnya,

Untuk mengkonversi 1023(basis 4) ke basis 3:
1023(basis 4)/3 => hasil bagi=25D sisa=0
25D/3 => hasil bagi=8D sisa=1
8D/3 => hasil bagi=2D sisa=2
2D/3 => hasil bagi=0 sisa=2 (hasil bagi=0 berhenti)
Jadi, 1023(basis 4) = 2210(basis 3)

1.8 Konversi Sistem Dua Bilangan dengan Bagian Pecahan

1. Pisahkan bagian bulat dan pecahannya.
2. Untuk bagian bulat, bagilah dengan R target secara berulang, dan kumpulkan sisa dalam urutan terbalik.
3. Untuk bagian pecahan, kalikan bagian pecahan dengan radix target secara berulang, dan kumpulkan bagian integralnya dalam urutan yang sama.

Contoh 1: Desimal ke Biner

Ubah 18.6875D menjadi biner

Bagian Integral = 18D

18 : 2 => hasil bagi=9, sisa=0

9 : 2 => hasil bagi=4, sisa=1

4 : 2 => hasil bagi=2, sisa=0

2 : 2 => hasil bagi=1, sisa=0

1 : 2 => hasil bagi=0, sisa=1 **(hasil bagi=0 berhenti)**

Jadi, 18D = 10010B

Bagian Pecahan = .6875D

0.6875 x 2 =1.375 => bilangan bulat adalah 1

0.375 x 2 =0.75 => bilangan bulat adalah 0

0.75 x 2 =1.5 => bilangan bulat adalah 1

0.5 x 2 =1.0 => bilangan bulat adalah 1 **(hasil kali=1 berhenti)**

Oleh karena itu .6875D = .1011B

Gabungkan, 18.6875D = 10010.1011B

Catatan penting : tidak selamanya hasil kali =1, terkadang akan terus berlanjut proses perkaliannya. Proses akan berhenti tergantung dari toleransi kesalahan yang telah ditentukan.

Contoh 2: Desimal ke Heksadesimal

Ubah 18.6875D menjadi heksadesimal

Bagian Integral = 18D

18 : 16 => hasil bagi=1, sisa=2

1 : 16 => hasil bagi=0, sisa=1 **(hasil bagi=0 berhenti)**

Jadi, 18D = 12H

Bagian Pecahan = .6875D

0.6875 x 16=11.0 => bilangan bulat adalah 11D (BH)

Oleh karena itu .6875D = .BH

Gabungkan, 18.6875D = 12.BH

1.9 Latihan (Konversi Sistem Bilangan)

1. *Ubahlah bilangan desimal* berikut menjadi bilangan *biner* dan *heksadesimal :*
    1. 108
    2. 4848
    3. 9000
2. Ubahlah bilangan biner berikut menjadi bilangan heksadesimal dan desimal:
    1. 1000011000
    2. 10000000
    3. 101010101010
3. Ubahlah bilangan heksadesimal berikut menjadi bilangan biner dan desimal:
    1. ABCDE
    2. 1234
    3. 80F
4. Ubahlah bilangan desimal berikut menjadi bilangan biner yang setara:
    1. 19.25D
    2. 123.456D

Jawaban: Anda dapat menggunakan Kalkulator Windows (calc.exe) untuk melakukan konversi sistem bilangan, dengan mengaturnya ke mode *Programmer* atau *ilmiah* . (Jalankan "calc" ⇒ Pilih menu "Settings" ⇒ Pilih mode "Programmer" atau "Scientific".)

1. 1101100B, 1001011110000B, 100011001010000B, 6CH, 12F0H, 2328H.
2. 218H, 80H, AAAH, 536D, 128D, 2730D.
3. 10101011110011011110B, 1001000110100B, 100000001111B, 703710D, 4660D, 2063 D.
4. ?? (Kamu berhasil!)

2. Memori Komputer & Representasi Data

Komputer menggunakan *sejumlah bit tetap* untuk mewakili sepotong data, yang bisa berupa angka, karakter, atau lainnya. Lokasi penyimpanan *n* -bit dapat mewakili hingga entitas yang berbeda. Misalnya, lokasi memori 3-bit dapat menampung salah satu dari delapan pola biner berikut: 000, 001, 010, 011, 100, 101, 110, atau 111.

Bilangan bulat, dapat direpresentasikan dalam 8-bit, 16-bit, 32-bit, atau 64-bit. Ketika anda sebagai seorang programmer tentunya akan memilih panjang bit yang sesuai untuk bilangan bulat akan anda proses. Pilihan Anda akan memberikan batasan pada rentang bilangan bulat yang dapat direpresentasikan. Selain panjang bit, bilangan bulat dapat direpresentasikan dalam berbagai

skema *representasi* , misalnya bilangan bulat tak bertanda (unsigned) vs bilangan bulat bertanda (unsigned). Bilangan bulat tak bertanda 8-bit memiliki rentang 0 hingga 255, sedangkan bilangan bulat bertanda 8-bit memiliki rentang -128 hingga 127 - keduanya mewakili 256 angka berbeda.

Penting untuk dicatat bahwa lokasi memori komputer hanya *menyimpan pola biner* . Terserah sepenuhnya kepada Anda, sebagai pemrogram, untuk memutuskan bagaimana pola-pola ini *diinterpretasikan* . Misalnya, pola biner 8-bit "0100 0001B" dapat diartikan sebagai bilangan bulat yang tidak bertanda 65, atau karakter ASCII 'A', atau beberapa informasi rahasia yang hanya diketahui oleh Anda. Dengan kata lain, Anda harus terlebih dahulu memutuskan bagaimana merepresentasikan sepotong data dalam pola biner sebelum pola biner tersebut dapat diproses dengan aritmetika logika. Interpretasi pola biner disebut *representasi data* atau *pengkodean*. Selain itu, skema representasi data harus disepakati oleh semua pihak, yaitu standar industri perlu dirumuskan dan diikuti dengan ketat.

Setelah Anda memutuskan skema representasi data, batasan tertentu, khususnya presisi dan jangkauan akan diterapkan. Oleh karena itu, penting untuk memahami *representasi data* untuk menulis program *yang benar* dan *berkinerja tinggi* .

Batu Roset dan Penguraian Hieroglif Mesir

Hieroglif Mesir (di sebelah kiri) digunakan oleh orang Mesir kuno sejak 4000 SM. Sayangnya, sejak tahun 500 M, tidak ada lagi yang bisa membaca hieroglif Mesir kuno, hingga ditemukannya kembali Batu Rosette pada tahun 1799 oleh pasukan Napoleon (saat invasi Napoleon ke Mesir) di dekat kota Rashid (Rosetta) di Delta Nil.

Batu Rosetta (kiri) bertuliskan sebuah dekrit pada tahun 196 SM atas nama Raja Ptolemy V. Dekrit tersebut muncul dalam *tiga* aksara: teks atas adalah *hieroglif Mesir Kuno* , bagian tengah aksara Demotik, dan *teks Yunani Kuno* terendah . Karena teks tersebut pada dasarnya menyajikan teks

yang sama dalam ketiga aksara tersebut, dan bahasa Yunani Kuno masih dapat dipahami, maka teks tersebut memberikan kunci untuk menguraikan hieroglif Mesir.

Pesan moral dari cerita ini adalah : Tidak seorang pun dapat memecahkan kode data dari sebuah informasi sebelum ia mengetahui skema pengkodean yang digunakan,.

Referensi dan gambar: Wikipedia.

3. Representasi Bilangan Bulat

Bilangan bulat adalah *bilangan bulat* atau *bilangan titik tetap* dengan titik radix *tetap* setelah bit paling signifikan. Berbeda dengan *bilangan real* atau *bilangan floating point* yang posisi titik radixnya bervariasi. Penting untuk diperhatikan bahwa bilangan bulat dan bilangan floating-point diperlakukan berbeda di komputer. Mereka memiliki representasi yang berbeda dan diproses secara berbeda (misalnya, bilangan floating-point diproses dalam apa yang disebut prosesor floating-point). Angka floating-point akan dibahas nanti.

Komputer menggunakan *jumlah bit yang tetap* untuk mewakili bilangan bulat. Panjang bit yang umum digunakan untuk bilangan bulat adalah 8-bit, 16-bit, 32-bit atau 64-bit. Selain panjang bit, ada dua skema representasi untuk bilangan bulat:

1. *Unsigned Integers atau Bilangan Bulat Tak Bertanda* : dapat mewakili bilangan bulat nol dan positif.
2. *Signed Integer atau Bilangan Bulat Bertanda* : dapat mewakili bilangan bulat nol, positif dan negatif. Tiga skema representasi telah diusulkan untuk bilangan bulat bertanda:
    1. Representasi Besaran Tanda
    2. Representasi Komplemen 1 atau 1's complement (R-1)
    3. Representasi Komplemen 2 atau 2's complements (R-2)

Sebagai seorang pemrogram, anda perlu memutuskan panjang bit dan skema representasi bilangan bulat Anda, bergantung pada kebutuhan aplikasi Anda. Misalkan Anda memerlukan penghitung untuk menghitung jumlah kecil dari 0 hingga 200, Anda dapat memilih skema bilangan bulat tak bertanda 8-bit karena tidak ada bilangan negatif yang terlibat.

3.1 *n* -bit Integer Tak Bertanda Tangan

Bilangan bulat tak bertanda dapat mewakili bilangan bulat nol dan positif, tetapi bukan bilangan bulat negatif. Nilai bilangan bulat tak bertanda diartikan sebagai "*besarnya pola biner yang mendasarinya*".

Contoh 1: Misalkan *n*=8 pola binernya adalah 0100 0001B, nilai bilangan bulat tak bertanda ini adalah 1×2^0 + 1×2^6 = 65D.

Contoh 2: Misalkan *n*=16 pola binernya adalah 0001 0000 0000 1000B, nilai bilangan bulat tak bertanda ini adalah 1×2^3 + 1×2^12 = 4104D.

Contoh 3: Misalkan *n*=16 pola binernya adalah 0000 0000 0000 0000B, nilai bilangan bulat tak bertanda ini adalah 0.

Pola *n* -bit dapat mewakili 2^*n* bilangan bulat yang berbeda. Integer *n* -bit unsigned dapat mewakili bilangan bulat dari 0 hingga (2^*n*)-1 , seperti yang ditabulasikan di bawah ini:

N	Minimum	Maksimum
8	0	(2^8)-1 (=255)
16	0	(2^16)-1 (=65.535)
32	0	(2^32)-1 (=4.294.967.295) (9+ digit)
64	0	(2^64)-1 (=18.446.744.073.709.551.615) (19+ digit)

3.2 Bilangan Bulat yang Ditandai

Bilangan bulat bertanda dapat mewakili nol, bilangan bulat positif, dan bilangan bulat negatif. Tiga skema representasi tersedia untuk bilangan bulat bertanda:

1. Representasi Besaran Tanda
2. Representasi Komplemen 1
3. Representasi Komplemen 2

Dalam ketiga skema di atas, *Most Significance Bit* (MSB) disebut *bit tanda*. Bit tanda digunakan untuk mewakili *tanda* bilangan bulat - dengan 0 untuk bilangan bulat positif dan 1 untuk bilangan bulat negatif. Namun, besarnya *bilangan* bulat diinterpretasikan secara berbeda dalam skema yang berbeda.

3.3 *n* -bit Tanda Bulat dalam Representasi Besaran Bertanda

Dalam representasi besaran tanda:

- *Most Significance Bit* (MSB)adalah *bit tanda* , dengan nilai 0 mewakili bilangan bulat positif dan 1 mewakili bilangan bulat negatif.

- Sisanya *n* -1 bit mewakili besarnya (nilai absolut) dari bilangan bulat. Nilai absolut bilangan bulat diartikan sebagai "besarnya pola biner (*n* -1)-bit".

Contoh 1 : Misalkan *n*=8 dan representasi binernya adalah 0 100 0001B.

Bit tandanya adalah 0⇒ positif

Nilai absolutnya adalah 100 0001B = 65D

Oleh karena itu, bilangan bulatnya adalah +65D

Contoh 2 : Misalkan *n*=8 dan representasi binernya adalah 1 000 0001B.

Bit tandanya adalah 1⇒ negatif

Nilai absolutnya adalah 000 0001B = 1D

Oleh karena itu, bilangan bulatnya adalah -1D

Contoh 3 : Misalkan *n*=8 dan representasi binernya adalah 0 000 0000B.

Bit tandanya adalah 0⇒ positif

Nilai absolutnya adalah 000 0000B = 0D

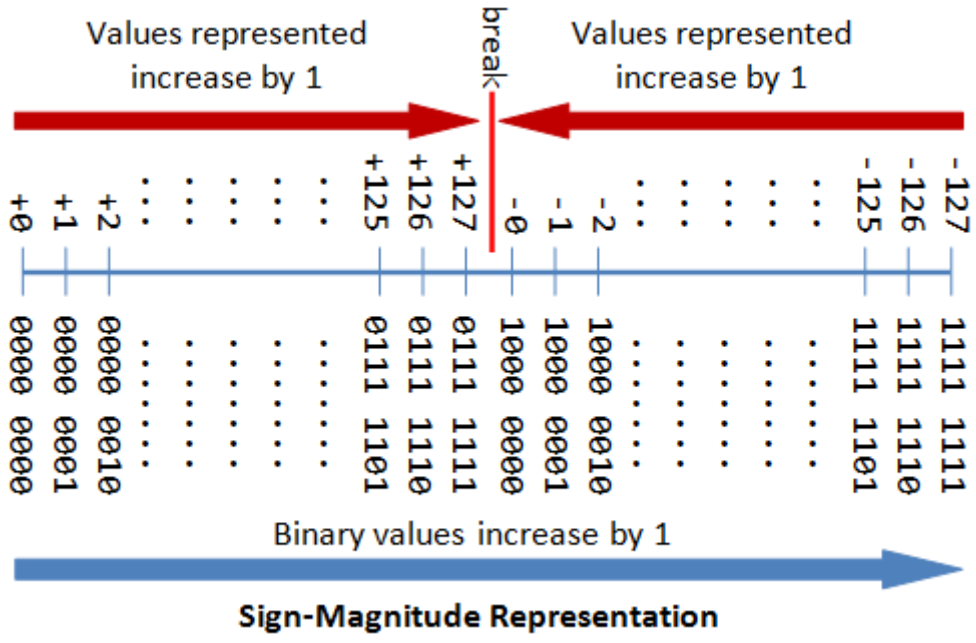
Oleh karena itu, bilangan bulatnya adalah +0D

Contoh 4 : Misalkan *n*=8 dan representasi binernya adalah 1 000 0000B.

Bit tandanya adalah 1⇒ negatif

Nilai absolutnya adalah 000 0000B = 0D

Oleh karena itu, bilangan bulatnya adalah -0D



Sign-Magnitude Representation

Kelemahan representasi besaran bertanda adalah:

1. Ada dua representasi (0000 0000B dan 1000 0000B) untuk angka nol, yang dapat menyebabkan inefisiensi dan kebingungan.
2. Bilangan bulat positif dan negatif perlu diproses secara terpisah.

3.4 *n* -bit Integer bertanda dalam Representasi Komplemen 1

Dalam representasi komplemen 1:

- Sekali lagi, *Most Significance Bit* (MSB) adalah *bit tanda* , dengan nilai 0 mewakili bilangan bulat positif dan 1 mewakili bilangan bulat negatif.
- Sisanya *n* -1 bit mewakili besarnya bilangan bulat, sebagai berikut:
    - untuk bilangan bulat positif, nilai absolut dari bilangan bulat tersebut sama dengan "besarnya pola biner (*n* -1)-bit".
    - untuk bilangan bulat negatif, nilai absolut bilangan bulat sama dengan "besarnya komplemen *(* invers *)* dari pola biner (*n* -1)-bit" (karenanya disebut komplemen 1).

Contoh 1 : Misalkan *n*=8dan representasi biner 0 100 0001B.

Bit tandanya adalah 0⇒ positif

Nilai absolutnya adalah 100 0001B = 65D

Oleh karena itu, bilangan bulatnya adalah +65D

Contoh 2 : Misalkan *n*=8 dan representasi biner 1 000 0001B.

Bit tanda adalah 1⇒ negatif

Nilai absolut adalah komplemen dari 000 0001B, yaitu, 111 1110B = 126D,

maka bilangan bulatnya adalah-126D

Contoh 3 : Misalkan *n*=8 dan representasi biner 0 000 0000B.

Bit tandanya adalah 0⇒ positif

Nilai absolutnya adalah 000 0000B = 0D

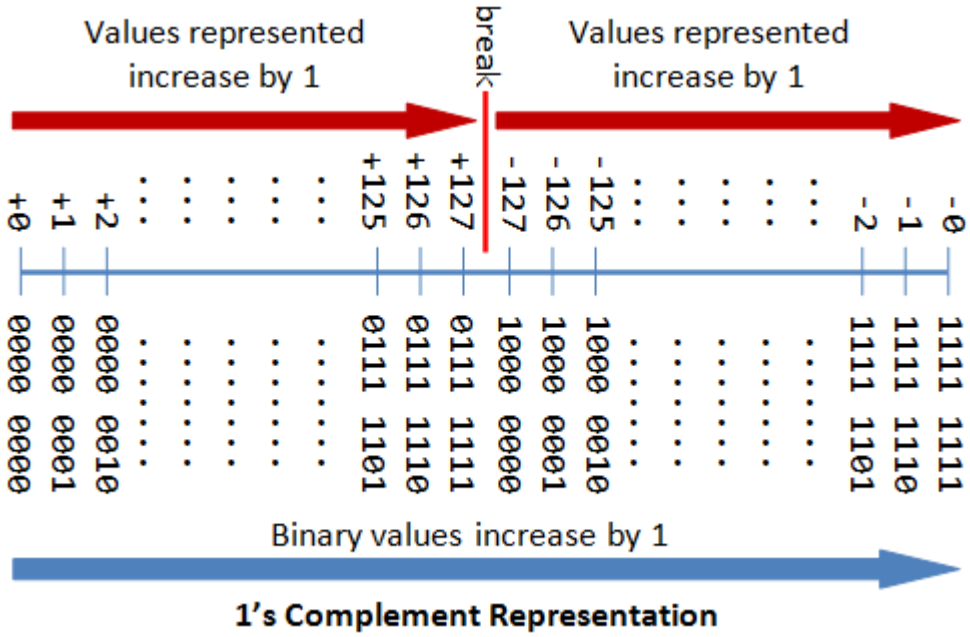
Oleh karena itu, bilangan bulatnya adalah +0D

Contoh 4 : Misalkan *n*=8dan representasi biner 1 111 1111B.

Bit tanda adalah 1⇒ negatif

Nilai absolut adalah komplemen dari 111 1111B, yaitu, 000 0000B = 0D

maka bilangan bulatnya adalah -0D



Sekali lagi, kelemahannya adalah:

1. Ada dua representasi (0000 0000Bdan 1111 1111B) untuk nol.
2. Bilangan bulat positif dan bilangan bulat negatif perlu diproses secara terpisah.

3.5 *n* -bit Integer bertanda dalam Representasi Komplemen 2

Dalam representasi komplemen 2:

- Sekali lagi, bit paling signifikan (msb) adalah *bit tanda*, dengan nilai 0 mewakili bilangan bulat positif dan 1 mewakili bilangan bulat negatif.
- Sisanya *n* -1 bit mewakili besarnya bilangan bulat, sebagai berikut:
    - untuk bilangan bulat positif, nilai absolut dari bilangan bulat tersebut sama dengan "besarnya pola biner (*n* -1)-bit".
    - untuk bilangan bulat negatif, nilai absolut dari bilangan bulat tersebut sama dengan "besarnya komplemen *dari* pola biner (*n -1)-bit ditambah satu* " (oleh karena itu disebut komplemen 2).

Contoh 1 : Misalkan *n*=8dan representasi biner 0 100 0001B.

Bit tandanya adalah 0⇒ positif

Nilai absolutnya adalah 100 0001B = 65D

Oleh karena itu, bilangan bulatnya adalah +65D

Contoh 2 : Misalkan *n*=8 dan representasi biner 1 000 0001B.

Bit tanda adalah 1⇒ negatif

Nilai absolut adalah komplemen dari 000 0001B plus 1, yaitu, 111 1110B + 1B = 127D

maka bilangan bulatnya adalah-127D

Contoh 3 : Misalkan *n*=8 dan representasi biner 0000 0000B.

Bit tandanya adalah 0⇒ positif

Nilai absolutnya adalah 000 0000B = 0D

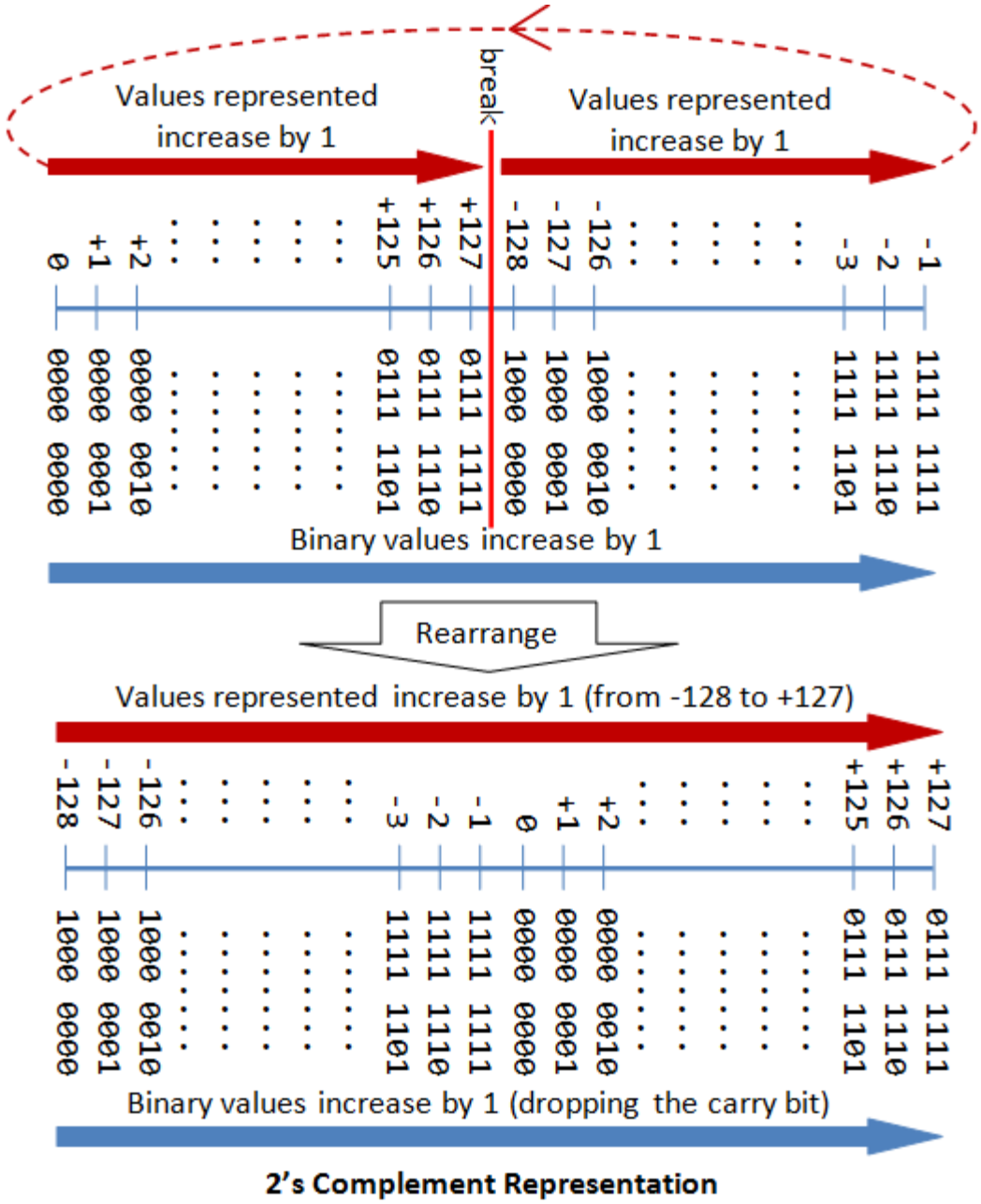
Oleh karena itu, bilangan bulatnya adalah +0D

Contoh 4 : Misalkan *n*=8 dan representasi biner 1 111 1111B.

Bit tanda adalah 1⇒ negatif

Nilai absolut adalah komplemen dari 111 1111B plus 1, yaitu, 000 0000B + 1B = 1D

maka bilangan bulatnya adalah -1D



3.6 Komputer menggunakan Representasi Komplemen 2 untuk Integer Bertanda

Kita telah membahas tiga representasi bilangan bulat bertanda: besaran bertanda, komplemen 1, dan komplemen 2. Komputer menggunakan komplemen 2 dalam merepresentasikan bilangan bulat bertanda. Hal ini dikarenakan:

1. Hanya ada satu representasi angka nol pada komplemen 2, bukan dua representasi pada besaran tanda dan komplemen 1.
2. Bilangan bulat positif dan negatif dapat dikerjakan bersama-sama dalam penjumlahan dan pengurangan. Pengurangan dapat dilakukan dengan menggunakan "logika penjumlahan".

Contoh 1: Penjumlahan Dua Bilangan Bulat Positif:

Misalkan n=8, 65D + 5D = 70D

```
65D → 0100 0001B
 5D → 0000 0101B(+
      0100 0110B → 70D (Oke)
```

Contoh 2: Pengurangan dianggap sebagai Penjumlahan Bilangan Bulat Positif dan Negatif:

Misalkan n=8, 65D - 5D = 65D + (-5D) = 60D

```
65D → 0100 0001B
-5D →  1111  1011B(+
       0011  1100B → 60D (discard carry - OK)
```

Contoh 3: Penjumlahan Dua Bilangan Bulat Negatif:

Misalkan n=8, -65D - 5D = (-65D) + (-5D) = -70D

```
-65D → 1011 1111B
 -5D → 1111 1011B(+
       1011 1010B → -70D (discard carry - OK)
```

Karena *presisi yang tetap / fixed precision* (yaitu, *jumlah bit yang tetap*), bilangan bulat bertanda komplemen *n*-bit 2 memiliki rentang tertentu. Misalnya, untuk *n*=8, rentang bilangan bulat bertanda komplemen 2 adalah -128 hingga +127. Selama penjumlahan (dan pengurangan), penting untuk memeriksa apakah hasilnya melebihi kisaran ini, dengan kata lain, apakah telah terjadi *overflow* atau *underflow*.

Contoh 4: Meluap:

Misalkan n=8, 127D + 2D = 129D (over-flow - di luar jangkauan)

```
127D → 0111 1111B
```

2D → 0000 0010B(+

1000 0001B → -127D (salah)

Contoh 5: Aliran bawah:

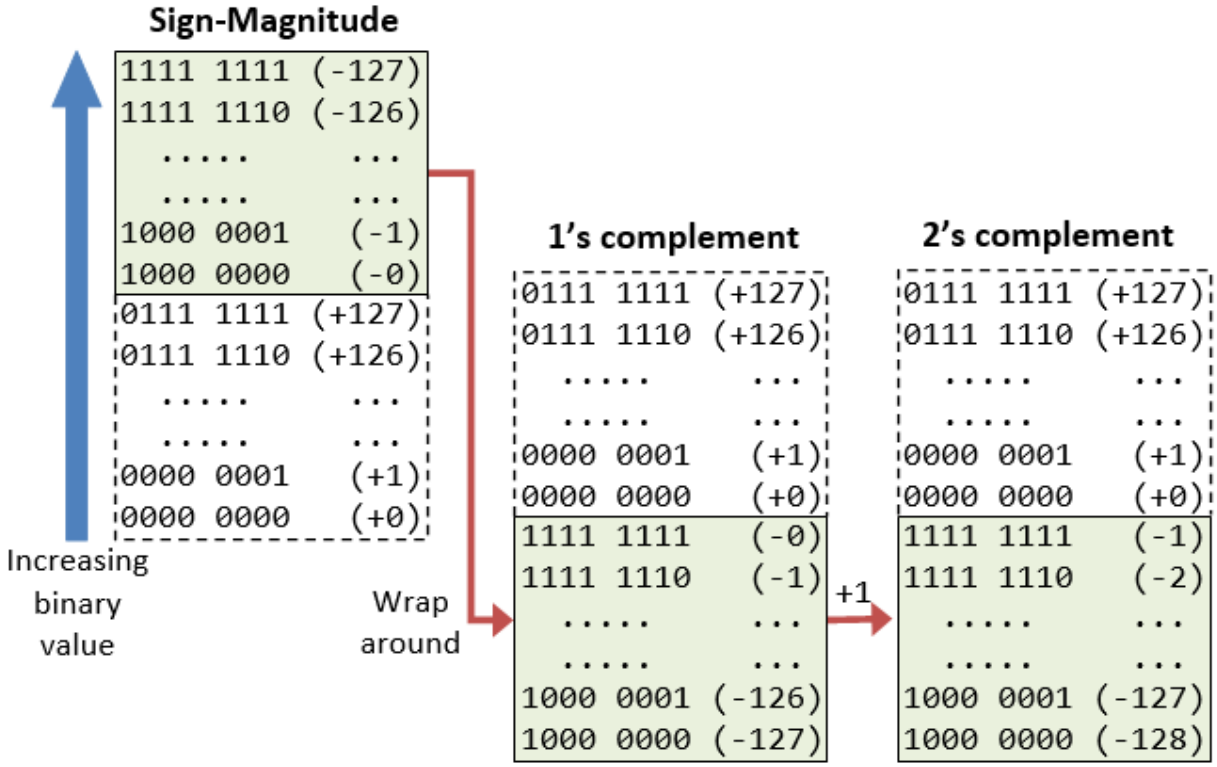
Misalkan n=8, -125D - 5D = -130D(aliran bawah - di bawah kisaran)

-125D → 1000 0011B

-5D → 1111 1011B(+

0111 1110B → +126D (salah)

Diagram berikut menjelaskan cara kerja komplemen 2. Dengan menata ulang garis bilangan, nilai dari -128hingga +127direpresentasikan secara berdekatan dengan mengabaikan bit carry.



3.7 Rentang Integer Bertanda Komplemen *n -bit 2*

Bilangan bulat bertanda komplemen *n* -bit 2 dapat mewakili bilangan bulat **dari -2^(*n*-1) hingga +2^(*n*-1)-1**, seperti yang ditabulasikan. Perhatikan bahwa skema ini dapat mewakili semua bilangan bulat dalam rentang tersebut, tanpa celah apa pun. Dengan kata lain, tidak ada bilangan bulat yang hilang dalam rentang yang didukung.

N	minimum	maksimum

8	-(2^7) (=-128)	+(2^7)-1 (=+127)
16	-(2^15) (=-32.768)	+(2^15)-1 (=+32.767)
32	-(2^31) (=-2.147.483.648)	+(2^31)-1 (=+2.147.483.647)(9+ digit)
64	-(2^63) (=-9.223.372.036.854.775.808)	+(2^63)-1 (=+9.223.372.036.854.775.807)(18+ digit)

3.8 Menguraikan Bilangan Komplemen 2

1. Periksa *bit penanda* (dilambangkan dengan S).
2. Jika S=0, bilangan tersebut positif dan nilai absolutnya adalah nilai biner dari *n* -1 bit yang tersisa.
3. Jika S=1, bilangan tersebut negatif. Anda dapat "membalikkan *n* -1 bit dan ditambah 1" untuk mendapatkan nilai absolut bilangan negatif. Alternatifnya, Anda dapat memindai sisa *n* -1 bit dari kanan (bit paling tidak signifikan). Cari kemunculan pertama 1. Balikkan semua bit ke kiri kemunculan pertama 1. Pola yang dibalik memberikan nilai absolut. Misalnya,

n = 8, pola bit = 1 100 0100B
S = 1 → negatif
Memindai dari kanan dan membalik semua bit ke kiri dari kemunculan pertama 1 ⇒ 011 1 100B
= 60D => komplemen 2 dari 60D
Jadi, nilainya adalah -60D

Atau dengan cara :

1. Cari bit komplemennya
2. Lalu tambahkan +1

n = 8, pola bit = 1 100 0100B

0011 1011 B (komplemen dari 1 100 0100B)
1 B (+
——————————
0011 1100 B => 60D
Jadi, nilai 1 100 0100B (komplemen 2) adalah -60D. Tanda minus karena S=1

3.9 Big Endian vs Little Endian

Komputer modern menyimpan satu byte data di setiap alamat atau lokasi memori, yaitu byte memori yang dapat dialamatkan. Oleh karena itu, bilangan bulat 32-bit disimpan dalam 4 alamat memori.

Istilah "Endian" mengacu pada *urutan* penyimpanan byte dalam memori komputer. Dalam skema "Big Endian", byte paling signifikan disimpan terlebih dahulu di alamat memori terendah (atau besar terlebih dahulu), sedangkan "Little Endian" menyimpan byte paling signifikan di alamat memori terendah.

Misalnya, bilangan bulat 32-bit 12345678H (305419896_{10}) disimpan sebagai 12H 34H 56H 78H di big endian; dan 78H 56H 34H 12H di little endian. Integer 16-bit 00H 01H diinterpretasikan sebagai 0001H di big endian, dan 0100H sebagai little endian.

3.10 Latihan (Representasi Integer)

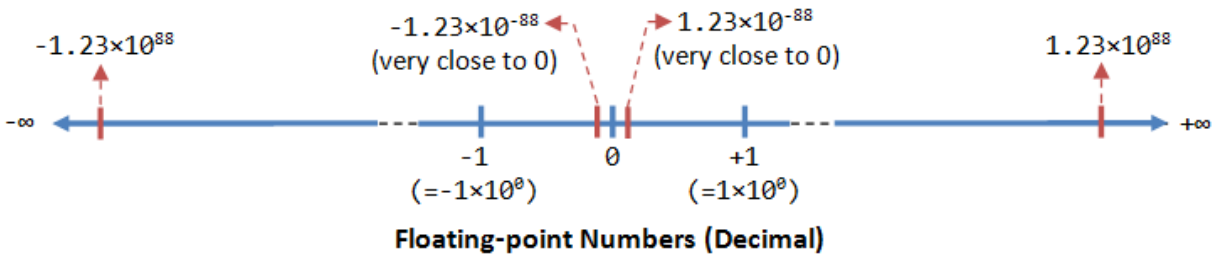
1. Berapa rentang bilangan bulat 8-bit, 16-bit, 32-bit, dan 64-bit, dalam representasi "unsigned" dan "signed"?
2. Berikan nilai 88, 0, 1, 127, dan 255dalam representasi unsigned 8-bit.
3. Berikan nilai +88, -88, -1, 0, +1, -128, dan +127dalam representasi bertanda komplemen 2 8-bit.
4. Berikan nilai +88, -88, -1, 0, +1, -127, dan +127dalam representasi besaran tanda 8-bit.
5. Berikan nilai +88, -88, -1, 0, +1, -127dan +127dalam representasi komplemen 1 8-bit.
6. [TODO] selengkapnya.

Jawaban

1. Kisaran bilangan bulat *n*-bit yang tidak bertanda adalah [0, 2^n - 1]. Kisaran komplemen *n*-bit 2 bertandabilangan bulat adalah [-2^(n-1), +2^(n-1)-1];
2. 88 (0101 1000), 0 (0000 0000), 1 (0000 0001), 127 (0111 1111), 255 (1111 1111).
3. +88 (0101 1000), -88 (1010 1000), -1 (1111 1111), 0 (0000 0000), +1 (0000 0001), -128 (1000 0000), +127 (0111 1111).
4. +88 (0101 1000), -88 (1101 1000), -1 (1000 0001), 0 (0000 0000 or 1000 0000), +1 (0000 0001), -127 (1111 1111), +127 (0111 1111).
5. +88 (0101 1000), -88 (1010 0111), -1 (1111 1110), 0 (0000 0000 or 1111 1111), +1 (0000 0001), -127 (1000 0000), +127 (0111 1111).

4. Representasi Angka Floating-Point

Bilangan floating-point (atau bilangan real) dapat mewakili nilai yang sangat besar (misalnya 1.23×10^88) atau nilai yang sangat kecil (misalnya 1.23×10^-88). Bisa juga mewakili bilangan negatif yang sangat besar (misalnya, -1.23×10^88) dan bilangan negatif yang sangat kecil (misalnya, -1.23×10^-88), serta nol, seperti yang diilustrasikan:



Floating-point Numbers (Decimal)

Bilangan floating-point biasanya dinyatakan dalam notasi ilmiah, dengan *pecahan* (F), dan *eksponen* ( ) dari *radix*E tertentu ( ), dalam bentuk . Bilangan desimal menggunakan radix 10 ( ); sedangkan bilangan biner menggunakan radix 2 ( ).rF×r^EF×10^EF×2^E

Representasi bilangan floating point tidaklah unik. Misalnya bilangan 55.66dapat direpresentasikan sebagai 5.566×10^1, 0.5566×10^2, 0.05566×10^3, dan seterusnya. Bagian pecahan dapat *dinormalisasi* . Dalam bentuk yang dinormalisasi, hanya ada satu digit bukan nol sebelum titik radix. Misalnya, bilangan desimal 123.4567dapat dinormalisasi menjadi 1.234567×10^2; bilangan biner 1010.1011Bdapat dinormalisasi sebagai 1.0101011B×2^3.

Penting untuk dicatat bahwa bilangan floating-point mengalami *kehilangan presisi* ketika direpresentasikan dengan jumlah bit yang tetap (misalnya, 32-bit atau 64-bit). Hal ini karena jumlah bilangan real *tidak terhingga* (bahkan dalam kisaran kecil, katakanlah 0,0 hingga 0,1). Di sisi lain, pola biner *n* -bit dapat mewakili bilangan berbeda *yang terbatas* . Oleh karena itu, tidak semua bilangan real dapat direpresentasikan. Perkiraan terdekat akan digunakan sebagai gantinya, sehingga mengakibatkan hilangnya akurasi.2^*n*

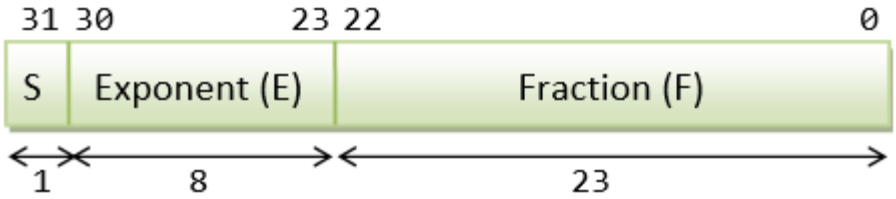
Penting juga untuk dicatat bahwa aritmatika bilangan mengambang kurang efisien dibandingkan aritmatika bilangan bulat. Ini bisa dipercepat dengan apa yang disebut *co-processor floating-point* khusus . Oleh karena itu, gunakan bilangan bulat jika aplikasi Anda tidak memerlukan angka floating-point.

Di komputer, bilangan floating-point direpresentasikan dalam notasi ilmiah *pecahan* (F) dan *eksponen* (E) dengan *radix* 2, dalam bentuk F×2^E. Keduanya Edan Fbisa positif maupun negatif. Komputer modern mengadopsi standar IEEE 754 untuk merepresentasikan bilangan floating-point. Ada dua skema representasi: presisi tunggal 32-bit dan presisi ganda 64-bit.

4.1 Nomor Titik Mengambang Presisi Tunggal IEEE-754 32-bit

Dalam representasi floating-point presisi tunggal 32-bit:

- Bit paling penting adalah *bit tanda* (S), dengan 0 untuk bilangan positif dan 1 untuk bilangan negatif.
- 8 bit berikut mewakili *eksponen* (E).
- 23 bit sisanya mewakili *pecahan* (F).



32-bit Single-Precision Floating-point Number

Bentuk Normalisasi

Mari kita ilustrasikan dengan sebuah contoh, misalkan pola 32-bit adalah , dengan:1 1000 0001 011 0000 0000 0000 0000 0000 0000

- S = 1
- E = 1000 0001
- F = 011 0000 0000 0000 0000 0000

Dalam *bentuk yang dinormalisasi* , pecahan sebenarnya dinormalisasi dengan awalan implisit 1 dalam bentuk 1.F. Dalam contoh ini, pecahan sebenarnya adalah 1.011 0000 0000 0000 0000 0000 = 1 + 1×2^-2 + 1×2^-3 = 1.375D.

Bit tanda mewakili tanda suatu bilangan, dengan S=0bilangan positif dan S=1bilangan negatif. Dalam contoh ini S=1, ini adalah bilangan negatif, yaitu -1.375D.

Dalam bentuk yang dinormalisasi, eksponen sebenarnya adalah E-127(disebut kelebihan-127 atau bias-127). Ini karena kita perlu merepresentasikan eksponen positif dan negatif. Dengan 8-bit E, berkisar antara 0 hingga 255, skema kelebihan-127 dapat memberikan eksponen aktual -127 hingga 128. Dalam contoh ini, E-127=129-127=2D.

Jadi, bilangan yang diwakili adalah -1.375×2^2=-5.5D.

Formulir De-Normalisasi

Bentuk yang dinormalisasi memiliki masalah serius, dengan implisit di depan 1 untuk pecahan, ia tidak dapat mewakili angka nol! Yakinkan diri Anda akan hal ini!

Bentuk denormalisasi dirancang untuk mewakili angka nol dan angka lainnya.

Untuk E=0, angka-angka tersebut berada dalam bentuk yang dinormalisasi. Awalan implisit 0 (bukan 1) digunakan untuk pecahan; dan eksponen sebenarnya selalu -126. Oleh karena itu, bilangan nol dapat dilambangkan dengan E=0dan F=0(karena 0.0×2^-126=0).

Kita juga dapat merepresentasikan bilangan positif dan negatif yang sangat kecil dalam bentuk denormalisasi dengan E=0. Misalnya, jika S=1, E=0, dan F=011 0000 0000 0000 0000 0000. Pecahan sebenarnya adalah 0.011=1×2^-2+1×2^-3=0.375D. Karena S=1, itu adalah bilangan negatif. Dengan E=0, eksponen sebenarnya adalah -126. Oleh karena itu bilangan tersebut adalah -0.375×2^-126 = -4.4×10^-39, yang merupakan bilangan negatif yang sangat kecil (mendekati nol).

Ringkasan

Singkatnya, nilai (N) dihitung sebagai berikut:

- Untuk 1 ≤ E ≤ 254, N = (-1)^S × 1.F × 2^(E-127). Angka-angka ini disebut bentuk *normalisasi* . Bit tanda melambangkan tanda suatu bilangan. Bagian pecahan (1.F) dinormalisasi dengan awalan implisit 1. Eksponennya bias (atau berlebih) dari 127, sehingga mewakili eksponen positif dan negatif. Kisaran eksponennya adalah -126hingga +127.
- Untuk E = 0, N = (-1)^S × 0.F × 2^(-126). Angka-angka ini disebut bentuk *denormalisasi* . Eksponen 2^-126evaluasi ke jumlah yang sangat kecil. Bentuk yang didenormalisasi diperlukan untuk merepresentasikan nol (dengan F=0dan E=0). Ini juga bisa mewakili angka positif dan negatif yang sangat kecil mendekati nol.
- Untuk E = 255, ini mewakili nilai-nilai khusus, seperti ±INF(tak terhingga positif dan negatif) dan NaN(bukan angka). Hal ini berada di luar cakupan artikel ini.

Contoh 1: Misalkan pola representasi floating-point 32-bit IEEE-754 adalah .0 10000000 110 0000 0000 0000 0000 0000

Bit tanda S = 0 ⇒ bilangan positif
E = 1000 0000B = 128D (dalam bentuk normalisasi)
Pecahannya adalah 1,11B (dengan awalan implisit 1) = 1 + 1×2^-1 + 1×2^-2 = 1,75D
Bilangannya adalah +1,75 × 2^(128-127) = +3,5D

Contoh 2: Misalkan pola representasi floating-point 32-bit IEEE-754 adalah .1 01111110 100 0000 0000 0000 0000 0000

Bit tanda S = 1 ⇒ bilangan negatif

E = 0111 1110B = 126D (dalam bentuk normalisasi)
Pecahannya adalah 1,1B (dengan awalan implisit 1) = 1 + 2^-1 = 1,5D
Bilangannya adalah -1,5 × 2^(126-127) = -0,75D

Contoh 3: Misalkan pola representasi floating-point 32-bit IEEE-754 adalah .1 01111110 000 0000 0000 0000 0000 0001

Bit tanda S = 1 ⇒ bilangan negatif
E = 0111 1110B = 126D (dalam bentuk normalisasi)
Pecahan adalah 1,000 0000 0000 0000 0000 0001B (dengan awalan implisit 1) = 1 + 2^-23
Angkanya adalah -(1 + 2^-23) × 2^(126-127) = -0.5000000596046447753906 25 (mungkin tidak tepat dalam desimal!)

Contoh 4 (Bentuk De-Normalisasi): Misalkan pola representasi floating-point 32-bit IEEE-754 adalah .1 00000000 000 0000 0000 0000 0000 0001

Bit tanda S = 1 ⇒ bilangan negatif
E = 0 (dalam bentuk de-normalisasi)
Pecahan adalah 0,000 0000 0000 0000 0000 0001B (dengan awalan implisit 0) = 1×2^-23
Bilangannya adalah -2^-23 × 2^(-126) = -2×(-149) ≈ -1,4×10^-45

4.2 Latihan (Angka Floating-point)

1. Hitung bilangan positif terbesar dan terkecil yang dapat direpresentasikan dalam bentuk normalisasi 32-bit.
2. Hitung bilangan negatif terbesar dan terkecil dapat direpresentasikan dalam bentuk normalisasi 32-bit.
3. Ulangi (1) untuk bentuk denormalisasi 32-bit.
4. Ulangi (2) untuk bentuk denormalisasi 32-bit.

Petunjuk:

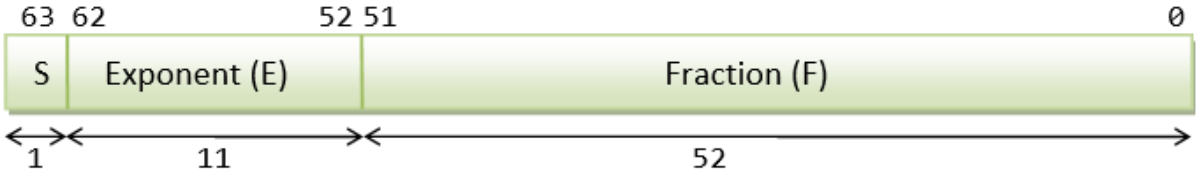
1. Bilangan positif terbesar: S=0, E=1111 1110 (254), F=111 1111 1111 1111 1111 1111. Bilangan positif terkecil: S=0, E=0000 00001 (1), F=000 0000 0000 0000 0000 0000.
2. Sama seperti di atas, tapi S=1.

3. Bilangan positif terbesar: S=0, E=0, F=111 1111 1111 1111 1111 1111. Bilangan positif terkecil: S=0, E=0, F=000 0000 0000 0000 0000 0001.
   Sama seperti di atas, tapi S=1.

4.3 Angka Titik Mengambang Presisi Ganda IEEE-754 64-bit

Skema representasi untuk presisi ganda 64-bit mirip dengan presisi tunggal 32-bit:

- Bit paling penting adalah *bit tanda* (S), dengan 0 untuk bilangan positif dan 1 untuk bilangan negatif.
- 11 bit berikut mewakili *eksponen* (E).
- Sisanya 52 bit mewakili *pecahan* (F).



64-bit Double-Precision Floating-point Number

Nilai (N) dihitung sebagai berikut:

- Bentuk yang dinormalisasi: Untuk 1 ≤ E ≤ 2046, N = (-1)^S × 1.F × 2^(E-1023).
- Bentuk yang didenormalisasi: Untuk E = 0, N = (-1)^S × 0.F × 2^(-1022). Ini berada dalam bentuk yang didenormalisasi.
- Sebab E = 2047, Nmewakili nilai khusus, seperti ±INF(tak terhingga), NaN(bukan angka).

4.4 Lebih lanjut tentang Representasi Floating-Point

Ada tiga bagian dalam representasi floating-point:

- Bit *tanda* (S) sudah cukup jelas (0 untuk bilangan positif dan 1 untuk bilangan negatif).
- Untuk *eksponen* (E), yang disebut *bias* (atau *kelebihan*) diterapkan untuk mewakili eksponen positif dan negatif. Biasnya ditetapkan pada setengah rentang. Untuk presisi tunggal dengan eksponen 8-bit, biasnya adalah 127 (atau kelebihan-127). Untuk presisi ganda dengan eksponen 11-bit, biasnya adalah 1023 (atau kelebihan-1023).
- Pecahan ( ) (juga disebut *mantissa* atau *signifikansi*) terdiri dari bit awal implisit (sebelum titik radix) dan bit pecahan (setelah titik radix) . Bit terdepan untuk bilangan yang dinormalisasi adalah 1; sedangkan bit terdepan untuk bilangan yang didenormalisasi adalah 0.F

Bilangan Floating-Point yang Dinormalisasi

Dalam bentuk yang dinormalisasi, titik radix ditempatkan setelah digit pertama bukan nol, misalnya, 9.8765D×10^-23D, 1.001011B×2^11B. Untuk bilangan biner, bit terdepan selalu 1, dan tidak perlu direpresentasikan secara eksplisit - ini menghemat 1 bit penyimpanan.

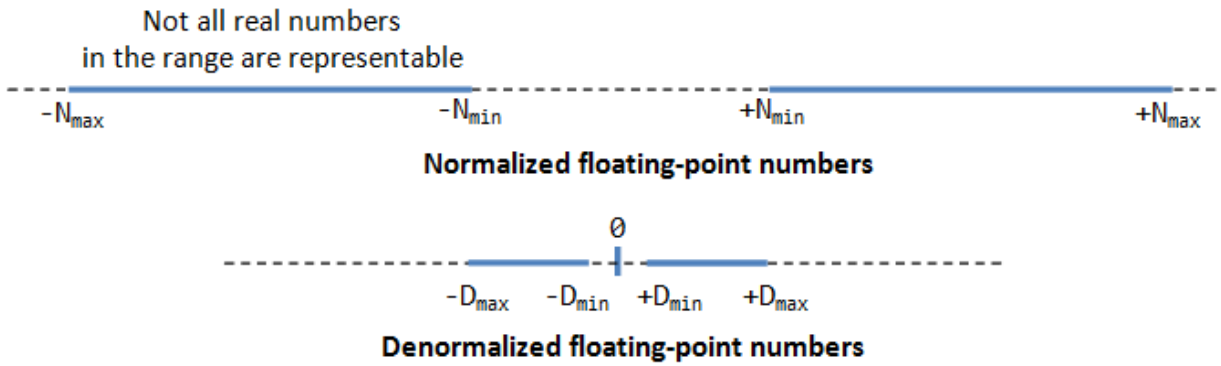
Dalam bentuk normalisasi IEEE 754:

- For single-precision, 1 ≤ E ≤ 254 with excess of 127. Hence, the actual exponent is from -126 to +127. Negative exponents are used to represent small numbers (< 1.0); while positive exponents are used to represent large numbers (> 1.0).
  N = (-1)^S × 1.F × 2^(E-127)
- For double-precision, 1 ≤ E ≤ 2046 with excess of 1023. The actual exponent is from -1022 to +1023, and
  N = (-1)^S × 1.F × 2^(E-1023)

Take note that n-bit pattern has a *finite* number of combinations (=2^n), which could represent *finite* distinct numbers. It is not possible to represent the *infinite* numbers in the real axis (even a small range says 0.0 to 1.0 has infinite numbers). That is, not all floating-point numbers can be accurately represented. Instead, the closest approximation is used, which leads to *loss of accuracy*.

The *minimum* and *maximum* normalized floating-point numbers are:

Precision	Normalized N(min)	Normalized N(max)
Single	0080 0000H 0 00000001 00000000000000000000000B E = 1, F = 0 N(min) = 1.0B × 2^-126 (≈1.17549435 × 10^-38)	7F7F FFFFH 0 11111110 00000000000000000000000B E = 254, F = 0 N(maks) = 1,1...1B × 2^127 = (2 - 2^-23) × 2^127 (≈3,4028235 × 10^38)
Dobel	0010 0000 0000 0000H N(min) = 1,0B × 2^-1022 (≈2,2250738585072014 × 10^-308)	7FEF FFFF FFFF FFFFH N(maks) = 1,1...1B × 2^1023 = (2 - 2^-52) × 2^1023 (≈1.7976931348623157 × 10^308)



Bilangan Floating-Point yang Dinormalisasi

Jika E = 0, tetapi pecahannya bukan nol, maka nilainya berada dalam bentuk denormalisasi, dan diasumsikan bit terdepan 0, sebagai berikut:

- Untuk presisi tunggal, E = 0,

  N = (-1)^S × 0.F × 2^(-126)
- Untuk presisi ganda, E = 0,

  N = (-1)^S × 0.F × 2^(-1022)

Bentuk yang didenormalisasi dapat mewakili bilangan yang sangat kecil yang mendekati nol, dan nol, yang tidak dapat direpresentasikan dalam bentuk yang dinormalisasi, seperti yang ditunjukkan pada gambar di atas.

Angka floating-point minimum dan maksimum yang didenormalisasi adalah:

Presisi	D terdenormalisasi (menit)	D yang dinormalisasi (maks)
single	0000 0001H 0 00000000 00000000000000000000001B E = 0, F = 00000000000000000000001B D(min) = 0,0...1 × 2^-126 = 1 × 2^-23 × 2^-126 = 2^-1 49 (≈1,4 × 10^ -45)	007F FFFFH 0 00000000 11111111111111111111111B E = 0, F = 11111111111111111111111B D(maks) = 0,1...1 × 2^-126 = (1-2^- 23)×2^-126 (≈1,175 4942×10^-38)
Dobel	0000 0000 0000 0001H D(mnt) = 0,0...1 × 2^-1022 = 1 ×	001F FFFF FFFF FFFFH D(maks) = 0,1...1 × 2^-1022 = (1-

	2^-52 × 2^-1022 = 2^-1074 (≈4,9 × 10^-324)	2^-52)×2^-1022 (≈4.4501477170144023 × 10^-308)

Nilai Khusus

Nol : Nol tidak dapat direpresentasikan dalam bentuk ternormalisasi, dan harus direpresentasikan dalam bentuk denormalisasi dengan E=0dan F=0. Ada dua representasi untuk nol: +0dengan S=0dan -0dengan S=1.

Infinity : Nilai +infinity (misalnya, 1/0) dan -infinity (misalnya, -1/0) diwakili dengan eksponen semua angka 1 (E = 255 untuk presisi tunggal dan E = 2047 presisi ganda), F=0, dan S=0(untuk +INF) dan S=1(untuk -INF).

Bukan Angka (NaN) : NaNmenunjukkan nilai yang tidak dapat direpresentasikan sebagai bilangan real (misalnya 0/0). NaNdiwakili dengan Eksponen semua 1 (E = 255untuk presisi tunggal dan E = 2047presisi ganda) dan pecahan apa pun yang bukan nol.

5. Pengkodean Karakter

Dalam memori komputer, karakter "dikodekan" (atau "diwakili") menggunakan "skema pengkodean karakter" yang dipilih (alias "kumpulan karakter", "rangkaian karakter", "peta karakter", atau "halaman kode").

Misalnya, di ASCII (serta Latin1, Unicode, dan banyak rangkaian karakter lainnya):

- nomor kode 65D (41H) sampai 90D (5AH) mewakili 'A' sampai 'Z'
- nomor kode 97D (61H) sampai 122D (7AH) mewakili 'a' sampai 'z'.
- nomor kode 48D (30H) sampai 57D (39H) mewakili '0' sampai '9'.

Penting untuk dicatat bahwa skema representasi harus diketahui sebelum pola biner dapat diinterpretasikan. Misalnya, pola 8-bit " 0100 0010B" dapat mewakili apa pun yang hanya diketahui oleh orang yang mengkodekannya.

Skema pengkodean karakter yang paling umum digunakan adalah: 7-bit ASCII (ISO/IEC 646) dan 8-bit Latin-x (ISO/IEC 8859-x) untuk karakter Eropa Barat, dan Unicode (ISO/IEC 10646) untuk internasionalisasi (i18n).

Skema pengkodean 7-bit (seperti ASCII) dapat mewakili 128 karakter dan simbol. Skema pengkodean karakter 8-bit (seperti Latin-x) dapat mewakili 256 karakter dan simbol; sedangkan skema pengkodean 16-bit (seperti Unicode UCS-2) dapat mewakili 65.536 karakter dan simbol.

5.1 Kode ASCII 7-bit (alias US-ASCII, ISO/IEC 646, ITU-T T.50)

- ASCII (American Standard Code for Information Interchange) adalah salah satu skema pengkodean karakter sebelumnya.
- ASCII awalnya merupakan kode 7-bit. Ini telah diperluas menjadi 8-bit untuk memanfaatkan organisasi memori komputer 8-bit dengan lebih baik. (Bit ke-8 awalnya digunakan untuk *pemeriksaan paritas* di komputer awal.)

Decimal	0	1	2	3	4	5	6	7	8	9
3			SP	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	SAYA	J	K	L	M	N	HAI
8	P	Q	R	S	T	kamu	V	W	X	Y
9	Z	[	\	]	^	_	`	A	B	C
10	D	e	F	G	H	Saya	J	k	aku	M
11	N	Hai	P	Q	R	S	T	kamu	ay	w
12	X	kamu	z	{	|	}	~			

- Kode nomor 32D (20H)menjadi 126D (7EH)karakter yang dapat dicetak (displayable) yang ditabulasikan (diurutkan dalam heksadesimal dan desimal) sebagai berikut:

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	SAYA	J	K	L	M	N	HAI
5	P	Q	R	S	T	kamu	V	W	X	Y	Z	[	\	]	^	_
6	`	A	B	C	D	e	F	G	H	Saya	J	k	aku	M	N	Hai
7	P	Q	R	S	T	kamu	ay	w	X	kamu	z	{	|	}	~	

    - Kode nomor 32D (20H)adalah karakter *kosong* atau *spasi* .
    - '0'to '9': 30H-39H (0011 0001B to 0011 1001B)atau (0011 xxxxBdi mana xxxxnilai integer yang setara)
    - 'A'ke 'Z': 41H-5AH (0101 0001B to 0101 1010B)atau (010x xxxxB). 'A'menjadi 'Z'kontinu tanpa celah.
    - 'a'ke 'z': 61H-7AH (0110 0001B to 0111 1010B)atau (011x xxxxB). 'A'untuk 'Z'juga terus menerus tanpa celah. Namun terdapat kesenjangan antara huruf besar dan huruf kecil. Untuk mengkonversi antara huruf besar dan kecil, balikkan nilai bit-5.

- Kode nomor 0D (00H)ke 31D (1FH), dan 127D (7FH)merupakan karakter kontrol khusus, yang tidak dapat dicetak (non-displayable), seperti yang ditabulasikan di bawah ini. Banyak dari karakter ini digunakan pada masa-masa awal untuk kontrol transmisi (misalnya, STX, ETX) dan kontrol printer (misalnya, Form-Feed), yang kini sudah tidak digunakan lagi. Kode bermakna yang tersisa saat ini adalah:
    - 09Huntuk Tab ('\t').
    - 0AHuntuk Line-Feed atau newline (LF atau '\n') dan 0DHuntuk Carriage-Return (CR atau 'r'), yang digunakan sebagai *pembatas baris* (alias *pemisah baris* , *end-of-line*) untuk file teks. Sayangnya tidak ada standar untuk pembatas garis: penggunaan Unix dan Mac 0AH(LF atau " \n"), penggunaan Windows 0D0AH(CR+LF atau " \r\n"). Bahasa pemrograman seperti

- C/C++/Java (yang dibuat di Unix) menggunakan 0AH(LF atau " \n").
- Dalam bahasa pemrograman seperti C/C++/Java, line-feed (0AH) dinotasikan sebagai '\n', carriage-return (0DH) sebagai '\r', tab (09H) sebagai '\t'.

Decimal	HEX	Arti		Desember	HEX	Arti	
0	00	TIDAK	Batal	17	11	DC1	Kontrol Perangkat 1
1	01	SOH	Mulai dari Pos	18	12	DC2	Kontrol Perangkat 2
2	02	STX	Awal Teks	19	13	DC3	Kontrol Perangkat 3
3	03	DLL	Akhir Teks	20	14	DC4	Kontrol Perangkat 4
4	04	EOT	Akhir Transmisi	21	15	TIDAK	Ack Negatif.
5	05	ENQ	Pertanyaan	22	16	SIN	Sinkronisasi. Menganggur
6	06	ACK	Pengakuan	23	17	ETB	Akhir Transmisi
7	07	BEL	lonceng	24	18	BISA	Membatalkan
8	08	BS	Menghapus'\b'	25	19	mereka	Akhir Media
9	**09**	**HT**	**Tab Horisontal'\t'**	26	1A	SUB	Pengganti
10	**0A**	**JIKA**	**Umpan Garis'\n'**	27	1B	ESC	Melarikan diri
11	0B	VT	Umpan Vertikal	28	1C	IS4	Pemisah File
12	0C	FF	Formulir Umpan'f'	29	1D	IS3	Pemisah Grup

13	**OD**	**Kr**	**Kereta kembali'\r'**	30	1E	IS2	Pemisah Rekam
14	OE	JADI	Bergeser Keluar	31	1F	IS1	Pemisah Satuan
15	OF	SI	Pergeseran Masuk				
16	10	DLE	Pelarian Tautan Data	127	7F	DEL	Menghapus

5.2 8-bit Latin-1 (alias ISO/IEC 8859-1)

ISO/IEC-8859 adalah *kumpulan* standar pengkodean karakter 8-bit untuk bahasa barat.

ISO/IEC 8859-1, alias alfabet Latin No. 1, atau singkatnya Latin-1, adalah skema pengkodean yang paling umum digunakan untuk bahasa-bahasa Eropa Barat. Ini memiliki 191 karakter yang dapat dicetak dari aksara latin, yang mencakup bahasa seperti Inggris, Jerman, Italia, Portugis dan Spanyol. Latin-1 kompatibel dengan kode US-ASCII 7-bit. Artinya, 128 karakter pertama dalam bahasa Latin-1 (kode nomor 0 hingga 127 (7FH)), sama dengan US-ASCII. Kode nomor 128 (80H) hingga 159 (9FH) tidak ditetapkan. Kode nomor 160 (A0H) sampai 255 (FFH) diberikan sebagai berikut:

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D
A	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	MA
B	º	±	²	³	´	mikro	¶	·	¸	¹	º	»	¼	½
C	A	A	A	A	A	A	Æ	C	dan	yaitu	Ê	Ë	SAYA	SAY
D	D	N	HAI	HAI	HAI	HAI	HAI	×	HAI	Ù	Ú	Û	Ü	Ý
E	A	A	A	A	A	A	æ	C	ya	yaitu	ê	yaitu	Saya	Say
F	D	N	Hai	Hai	Hai	Hai	Hai	–	Hai	ù	kamu	û	ü	ý

ISO/IEC-8859 memiliki 16 bagian. Selain Bagian 1 yang paling umum digunakan, Bagian 2 ditujukan untuk Eropa Tengah (Polandia, Ceko, Hongaria, dll), Bagian 3 untuk Eropa Selatan (Turki, dll),

Bagian 4 untuk Eropa Utara (Estonia, Latvia, dll), Bagian 5 untuk Sirilik, Bagian 6 untuk Arab, Bagian 7 untuk Yunani, Bagian 8 untuk Ibrani, Bagian 9 untuk Turki, Bagian 10 untuk Nordik, Bagian 11 untuk Thailand, Bagian 12 ditinggalkan, Bagian 13 untuk Baltik Rim, Bagian 14 untuk Celtic , Bagian 15 untuk bahasa Prancis, Finlandia, dll. Bagian 16 untuk Eropa Tenggara.

5.3 Ekstensi 8-bit US-ASCII lainnya (Ekstensi ASCII)

Selain standar ISO-8859-x, ada banyak ekstensi ASCII 8-bit yang tidak kompatibel satu sama lain.

ANSI (American National Standards Institute) (alias **Windows-1252** , atau Windows Codepage 1252): untuk huruf Latin yang digunakan dalam sistem DOS/Windows lama. Ini adalah superset dari ISO-8859-1 dengan nomor kode 128 (80H) hingga 159 (9FH) yang ditetapkan ke karakter yang dapat ditampilkan, seperti tanda kutip tunggal "pintar" dan tanda kutip ganda. Masalah umum di browser web adalah semua tanda kutip dan apostrof (dihasilkan oleh "tanda kutip cerdas" di beberapa perangkat lunak Microsoft) diganti dengan tanda tanya atau simbol aneh. Itu karena dokumen diberi label ISO-8859-1 (bukan Windows-1252), yang nomor kodenya tidak ditentukan. Sebagian besar browser dan klien email modern memperlakukan charset ISO-8859-1 sebagai Windows-1252 untuk mengakomodasi kesalahan pelabelan tersebut.

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C
8	€		‚	ƒ	„	…	†	‡	ˆ	‰	S	‹	Œ
9		'	'	“	”	•	–	—		™	S	›	œ

EBCDIC (Extracted Binary Coded Decimal Interchange Code): Digunakan pada komputer IBM awal.

5.4 Unicode (alias Kumpulan Karakter Universal ISO/IEC 10646)

Sebelum Unicode, tidak ada skema pengkodean karakter tunggal yang dapat mewakili karakter dalam semua bahasa. Misalnya, Eropa Barat menggunakan beberapa skema pengkodean (dalam keluarga ISO-8859-x). Bahkan satu bahasa seperti Cina memiliki beberapa skema pengkodean (GB2312/GBK, BIG5). Banyak skema pengkodean yang bertentangan satu sama lain, misalnya nomor kode yang sama diberikan ke karakter yang berbeda.

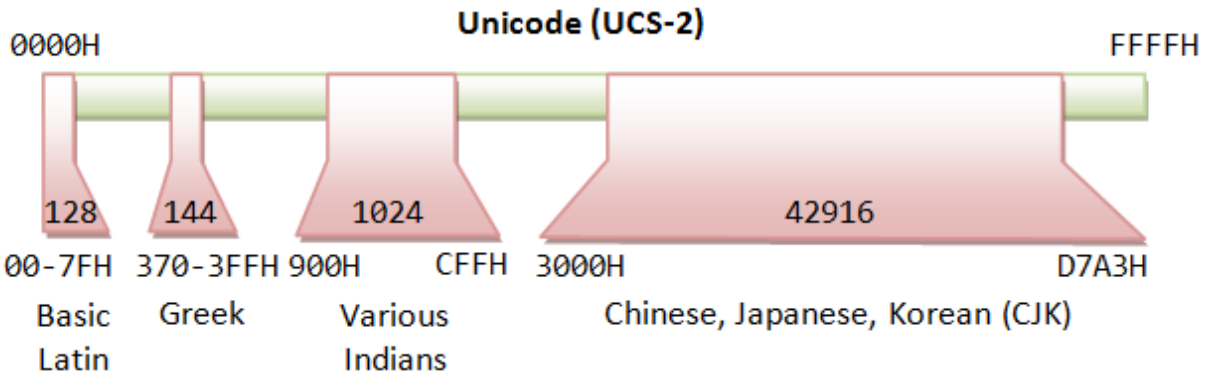
Unicode bertujuan untuk menyediakan skema pengkodean karakter standar, yang bersifat universal, efisien, seragam dan tidak ambigu. Standar Unicode dikelola oleh organisasi nirlaba yang disebut Konsorsium Unicode (@ www.unicode.org). Unicode adalah standar ISO/IEC 10646.

Unicode kompatibel dengan 7-bit US-ASCII dan 8-bit Latin-1 (ISO-8859-1). Artinya, 128 karakter pertama sama dengan US-ASCII; dan 256 karakter pertama sama dengan Latin-1.

Unicode awalnya menggunakan 16 bit (disebut UCS-2 atau Unicode Character Set - 2 byte), yang dapat mewakili hingga 65.536 karakter. Sejak itu telah diperluas menjadi lebih dari 16 bit, saat ini mencapai 21 bit. Kisaran kode hukum dalam ISO/IEC 10646 sekarang dari U+0000H hingga U+10FFFFH (21 bit atau sekitar 2 juta karakter), mencakup semua skrip sejarah terkini dan kuno. Rentang 16-bit asli dari U+0000H hingga U+FFFFH (65536 karakter) dikenal sebagai *Basic Multilingual Plane* (BMP), yang mencakup semua bahasa utama yang digunakan saat ini. Karakter di luar BMP disebut *Karakter Tambahan* , yang jarang digunakan.

Unicode memiliki dua skema pengkodean:

- **UCS-2** (Universal Character Set - 2 Byte): Menggunakan 2 byte (16 bit), mencakup 65.536 karakter dalam BMP. BMP cukup untuk sebagian besar aplikasi. UCS-2 sekarang sudah usang.
- **UCS-4** (Universal Character Set - 4 Byte): Menggunakan 4 byte (32 bit), mencakup BMP dan karakter tambahan.



5.5 UTF-8 (Format Transformasi Unicode - 8-bit)

Unicode 16/32-bit (UCS-2/4) sangat tidak efisien jika dokumen sebagian besar berisi karakter ASCII, karena setiap karakter menempati dua byte penyimpanan. Skema pengkodean dengan panjang variabel, seperti UTF-8, yang menggunakan 1-4 byte untuk mewakili karakter, dirancang untuk meningkatkan efisiensi. Di UTF-8, 128 karakter US-ASCII yang umum digunakan hanya menggunakan 1 byte, namun beberapa karakter yang kurang umum mungkin memerlukan hingga 4 byte. Secara keseluruhan, efisiensi meningkat untuk dokumen yang sebagian besar berisi teks US-ASCII.

Transformasi antara Unicode dan UTF-8 adalah sebagai berikut:

sedikit	Unikode	Kode UTF-8	Byte
7	00000000 0xxxxxxx	0xxxxxxx	1 (ASCII)

11	00000yyy yyxxxxxx	110yyyy 10xxxxxx	2
16	zzzzyyyy yyxxxxxx	1110zzzz 10yyyyy 10xxxxxx	3
21	000uuuuu zzzzyyyy yyxxxxxx	11110uuu 10uuzzzz 10yyyyyy 10xxxxxx	4

Dalam UTF-8, nomor Unicode yang sesuai dengan karakter ASCII 7-bit diisi dengan nol di depannya; sehingga memiliki nilai yang sama dengan ASCII. Oleh karena itu, UTF-8 dapat digunakan dengan semua perangkat lunak yang menggunakan ASCII. Nomor Unicode 128 ke atas, yang lebih jarang digunakan, dikodekan menggunakan lebih banyak byte (2-4 byte). UTF-8 umumnya memerlukan lebih sedikit penyimpanan dan kompatibel dengan ASCII. Kelemahan UTF-8 adalah diperlukan lebih banyak daya pemrosesan untuk membongkar kode karena panjangnya yang bervariasi. UTF-8 adalah format paling populer untuk Unicode.

Catatan:

- UTF-8 menggunakan 1-3 byte untuk karakter dalam BMP (16-bit), dan 4 byte untuk karakter tambahan di luar BMP (21-bit).
- 128 karakter ASCII (huruf Latin dasar, angka, dan tanda baca) menggunakan satu byte. Sebagian besar karakter Eropa dan Timur Tengah menggunakan urutan 2 byte, yang mencakup huruf Latin yang diperluas (dengan aksen tilde, makron, lancip, kuburan, dan lainnya), Yunani, Armenia, Ibrani, Arab, dan lain-lain. Bahasa Cina, Jepang dan Korea (CJK) menggunakan urutan tiga byte.
- Semua byte, kecuali 128 karakter ASCII, memiliki '1'bit terdepan. Dengan kata lain, byte ASCII, dengan '0'bit terdepan, dapat diidentifikasi dan didekode dengan mudah.

Contoh : 您好(Unicode: 60A8H 597DH)

Unicode (UCS-2) adalah 60A8H = 0110 0000 10 101000B
⇒ UTF-8 adalah 11100110 10000010 10101000B = E6 82 A8H
Unicode (UCS-2) adalah 597DH = 0101 1001 01 111101B
⇒ UTF-8 adalah 11100101 10100101 10111101B = E5 A5 BDH

5.6 UTF-16 (Format Transformasi Unicode - 16-bit)

UTF-16 adalah skema pengkodean karakter Unicode dengan panjang variabel, yang menggunakan 2 hingga 4 byte. UTF-16 tidak umum digunakan. Tabel transformasinya adalah sebagai berikut:

Unikode	Kode UTF-16

xxxxxxxxx xxxxxxxxx	Sama seperti UCS-2 - tanpa peng
000uuuuu zzzzyyyy yyxxxxxx (uuuuu≠0)	110110ww wwzzzzyy 110111yy y (wwww = uu

Perhatikan bahwa untuk 65536 karakter di BMP, UTF-16 sama dengan UCS-2 (2 byte). Namun, 4 byte digunakan untuk karakter tambahan di luar BMP.

Untuk karakter BMP, UTF-16 sama dengan UCS-2. Untuk karakter tambahan, setiap karakter memerlukan pasangan nilai 16-bit, yang pertama dari rentang pengganti yang tinggi, (\uD800-\uDBFF), yang kedua dari rentang pengganti yang rendah (\uDC00-\uDFFF).

5.7 UTF-32 (Format Transformasi Unicode - 32-bit)

Sama seperti UCS-4, yang menggunakan 4 byte untuk setiap karakter - tidak dikodekan.

5.8 Format File Teks Multi-Byte (misalnya Unicode).

Endianess (atau urutan byte) : Untuk karakter multi-byte, Anda perlu memperhatikan urutan byte dalam penyimpanan. Dalam *big endian* , byte paling signifikan disimpan di lokasi memori dengan alamat terendah (byte besar terlebih dahulu). Di *little endian* , byte paling signifikan disimpan di lokasi memori dengan alamat tertinggi (byte kecil pertama). Misalnya, 您 (dengan nomor Unicode 60A8H) disimpan seperti 60 A8dalam big endian; dan disimpan seperti A8 60di little endian. Big endian, yang menghasilkan hex dump yang lebih mudah dibaca, lebih umum digunakan, dan sering kali merupakan default.

BOM (Byte Order Mark) : BOM adalah karakter Unicode khusus yang mempunyai nomor kode FEFFH, yang digunakan untuk membedakan big-endian dan little-endian. Untuk big-endian, BOM tampil seperti FE FFHdi penyimpanan. Untuk little-endian, BOM muncul sebagai FF FEH. Unicode mencadangkan kedua nomor kode ini untuk mencegahnya mogok dengan karakter lain.

File teks Unicode dapat menggunakan format berikut:

- Endian Besar: UCS-2BE, UTF-16BE, UTF-32BE.
- Endian Kecil: UCS-2LE, UTF-16LE, UTF-32LE.
- UTF-16 dengan BOM. Karakter pertama dari file tersebut adalah karakter BOM, yang menentukan endianess. Untuk big-endian, BOM tampil seperti FE FFHdi penyimpanan. Untuk little-endian, BOM muncul sebagai FF FEH.

File UTF-8 selalu disimpan sebagai big endian. BOM tidak berperan. Namun, di beberapa sistem (khususnya Windows), BOM ditambahkan sebagai karakter pertama dalam file UTF-8 sebagai tanda tangan untuk mengidentifikasi file sebagai kode UTF-8. Karakter BOM (FEFFH) dikodekan dalam

UTF-8 sebagai EF BB BF. Menambahkan BOM sebagai karakter pertama file tidak disarankan, karena mungkin disalahartikan di sistem lain. Anda dapat memiliki file UTF-8 tanpa BOM.

5.9 Format File Teks

Pembatas Garis atau End-Of-Line (EOL) : Terkadang, saat Anda menggunakan Windows NotePad untuk membuka file teks (dibuat di Unix atau Mac), semua baris digabungkan menjadi satu. Hal ini karena platform operasi yang berbeda menggunakan karakter yang berbeda sebagai *pembatas garis* (atau *end-of-line* atau EOL). Dua karakter kontrol yang tidak dapat dicetak terlibat: 0AH(Line-Feed atau LF) dan 0DH(Carriage-Return atau CR).

- Windows/DOS menggunakan 0D0AH(CR+LF atau " \r\n") sebagai EOL.
- Unix dan Mac hanya menggunakan 0AH(LF atau " ").\n

Akhir File (EOF) : [TODO]

5.10 Halaman Kode CMD Windows

Skema pengkodean karakter (charset) di Windows disebut *codepage* . Di shell CMD, Anda dapat mengeluarkan perintah "chcp"untuk menampilkan halaman kode saat ini, atau "chcp codepage-number"untuk mengubah halaman kode.

Perhatikan bahwa:

- Halaman kode default 437 (digunakan dalam DOS asli) adalah kumpulan karakter 8-bit yang disebut *Extended ASCII* , yang berbeda dari Latin-1 untuk nomor kode di atas 127.
- Codepage 1252 (Windows-1252), tidak persis sama dengan Latin-1. Ini memberikan kode nomor 80H hingga 9FH pada huruf dan tanda baca, seperti tanda kutip tunggal dan tanda kutip ganda yang cerdas. Masalah umum pada browser yang menampilkan tanda kutip dan apostrof dalam tanda tanya atau kotak adalah karena halaman tersebut seharusnya Windows-1252, namun salah diberi label sebagai ISO-8859-1.
- Untuk internasionalisasi dan kumpulan karakter Cina: halaman kode 65001 untuk UTF8, halaman kode 1201 untuk UCS-2BE, halaman kode 1200 untuk UCS-2LE, halaman kode 936 untuk karakter Cina di GB2312, halaman kode 950 untuk karakter Cina di Big5.

5.11 Kumpulan Karakter Cina

Unicode mendukung semua bahasa, termasuk bahasa Asia seperti Cina (karakter sederhana dan tradisional), Jepang dan Korea (secara kolektif disebut CJK). Ada lebih dari 20.000 karakter CJK di Unicode. Karakter unicode sering kali dikodekan dalam skema UTF-8, yang sayangnya memerlukan 3 byte untuk setiap karakter CJK, bukan 2 byte dalam UCS-2 (UTF-16) yang tidak dikodekan.

Lebih buruk lagi, ada juga berbagai rangkaian karakter Cina, yang tidak kompatibel dengan Unicode:

- GB2312/GBK: untuk karakter Cina *yang disederhanakan* . GB2312 menggunakan 2 byte untuk setiap karakter Cina. Bit paling signifikan (MSB) dari kedua byte diatur ke 1 untuk hidup berdampingan dengan ASCII 7-bit dengan MSB 0. Ada sekitar 6700 karakter. GBK adalah perpanjangan dari GB2312, yang mencakup lebih banyak karakter serta karakter tradisional Tiongkok.
- BIG5: untuk karakter Cina *tradisional* BIG5 juga menggunakan 2 byte untuk setiap karakter Cina. Bit paling signifikan dari kedua byte juga disetel ke 1. BIG5 tidak kompatibel dengan GBK, yaitu nomor kode yang sama diberikan ke karakter berbeda.

Misalnya, dunia menjadi lebih menarik dengan berbagai standar berikut:

	Standar	Karakter	Kode
Sederhana	GB2312	和谐	BACD D0B3
	UCS-2	和谐	548C 8C10
	UTF-8	和谐	E5928C E8B090
Tradisional	BESAR5	和諧	A94D BFD3
	UCS-2	和諧	548C 8AE7
	UTF-8	和諧	E5928C E8ABA7

Catatan untuk Pengguna CMD Windows : Untuk menampilkan karakter Cina dengan benar di shell CMD, Anda harus memilih halaman kode yang benar, misalnya 65001 untuk UTF8, 936 untuk GB2312/GBK, 950 untuk Big5, 1201 untuk UCS-2BE, 1200 untuk UCS -2LE, 437 untuk DOS asli. Anda dapat menggunakan perintah " chcp" untuk menampilkan halaman kode saat ini dan perintah " " untuk mengubah halaman kode. Anda juga harus memilih font yang dapat menampilkan karakter (misalnya Courier New, Consolas atau Lucida Console, BUKAN font Raster).chcp *codepage_number*

5.12 Urutan Penyusunan (untuk Pemeringkatan Karakter)

Sebuah string terdiri dari serangkaian karakter dalam huruf besar atau kecil, misalnya, "apple", "BOY", "Cat". Dalam mengurutkan atau membandingkan string, jika kita mengurutkan karakter berdasarkan nomor kode yang mendasarinya (misalnya, US-ASCII) karakter per karakter, urutan contohnya adalah , , karena huruf besar memiliki nomor kode yang lebih kecil "BOY"daripada "apple"huruf "Cat"kecil . Hal ini tidak sesuai dengan apa yang disebut *urutan*

kamus , dimana huruf besar dan huruf kecil yang sama mempunyai pangkat yang sama. Masalah umum lainnya dalam mengurutkan string adalah "10"(sepuluh) kadang-kadang diurutkan di depan "1"to "9".

Oleh karena itu, dalam pengurutan atau perbandingan string, urutan yang disebut *susunan* (atau *susunan*) sering didefinisikan, yang menentukan peringkat huruf (huruf besar, huruf kecil), angka, dan simbol khusus. Ada banyak urutan penyusunan yang tersedia. Anda sepenuhnya bebas memilih urutan penyusunan untuk memenuhi persyaratan spesifik aplikasi Anda. Beberapa *urutan susunan kamus yang peka huruf besar-kecil* memiliki peringkat yang sama untuk huruf besar dan kecil yang sama, yaitu, 'A', 'a'⇒ 'B', 'b'⇒ ... ⇒ 'Z', 'z'. Beberapa *urutan susunan kamus peka huruf besar-* kecil menempatkan huruf besar sebelum huruf kecilnya, yaitu, 'A'⇒ 'B'⇒'C'... ⇒ 'a'⇒ . Biasanya, spasi diberi peringkat sebelum angka hingga , diikuti dengan abjad. 'b' ⇒ 'c'...'0''9'

Urutan penyusunan seringkali bergantung pada bahasa, karena bahasa yang berbeda menggunakan kumpulan karakter yang berbeda (misalnya, á, é, a, α) dengan urutannya sendiri.

5.13 Untuk Pemrogram C

Build dan Jalankan program Char-and-Number-Representation.c. Kode sumber dapat anda dapatkan di https://github.com/ferryastika/dasar-sistem-komputasi/blob/main/Char-and-Number-Representation.c .

6. Ringkasan - Mengapa Menerapkan Representasi Data?

Bilangan bulat , simbol karakter 1bilangan floating-point , dan string sama sekali berbeda di dalam memori komputer. Anda perlu mengetahui perbedaannya untuk menulis program yang bagus dan berkinerja tinggi.1.0'1'"1"

- *Dalam bilangan bulat bertanda* 8-bit , bilangan bulat 1 direpresentasikan sebagai 00000001B.
- *Dalam bilangan bulat tak bertanda* 8-bit , bilangan bulat 1 direpresentasikan sebagai 00000001B.
- *Dalam bilangan bulat bertanda* 16-bit , bilangan bulat 1 direpresentasikan sebagai 00000000 00000001B.

- *Dalam bilangan bulat bertanda* 32-bit , bilangan bulat 1 direpresentasikan sebagai 00000000 00000000 00000000 00000001B.
- *Dalam representasi floating-point* 32-bit , angka 1.0 direpresentasikan sebagai 0 01111111 0000000 00000000 00000000B, yaitu, S=0, E=127, F=0.
- *Dalam representasi floating-point* 64-bit , angka 1.0 direpresentasikan sebagai 0 01111111111 0000 00000000 00000000 00000000 00000000 00000000 00000000B, yaitu, S=0, E=1023, F=0.
- Dalam 8-bit Latin-1, simbol karakter '1' direpresentasikan sebagai 00110001B(atau 31H).
- Dalam UCS-2 16-bit, simbol karakter '1' direpresentasikan sebagai 00000000 00110001B.
- Dalam UTF-8, simbol karakter '1'direpresentasikan sebagai 00110001B.

6.1 Latihan (Representasi Data)

Untuk kode 16-bit berikut:

0000 0000 0010 1010;
1000 0000 0010 1010;

Berikan nilainya, jika mewakili:

1. bilangan bulat 16-bit yang tidak bertanda;
2. bilangan bulat bertanda 16-bit;
3. dua bilangan bulat 8-bit yang tidak bertanda;
4. dua bilangan bulat bertanda 8-bit;
5. karakter Unicode 16-bit;
6. dua karakter ISO-8859-1 8-bit.

Jawab:

(1) 42, 32810; (2) 42, -32726; (3) 0, 42; 128, 42; (4) 0, 42; -128, 42; (5) '※ '; '耪'; (6) NUL, '※'; PAD, '※'.

REFERENSI & SUMBER DAYA

1. (Spesifikasi Nomor Titik Mengambang) IEEE 754 (1985), "Standar IEEE untuk Aritmatika Titik Mengambang Biner".
2. (Spesifikasi ASCII) ISO/IEC 646 (1991) (atau ITU-T T.50-1992), "Teknologi informasi - kumpulan karakter berkode 7-bit untuk pertukaran informasi".

3. (Spesifikasi Latin-I) ISO/IEC 8859-1, "Teknologi informasi - kumpulan karakter grafis berkode byte tunggal 8-bit - Bagian 1: Alfabet Latin No. 1".
4. (Spesifikasi Unicode) ISO/IEC 10646, "Teknologi informasi - Kumpulan Karakter Berkode Multi-Oktet Universal (UCS)".
5. Konsorsium Unicode @ http://www.unicode.org .

Mostly copied from https://www3-ntu-edu-sg.translate.goog/home/ehchua/programming/java/datarepresentation.html?_x_tr_sl=en&_x_tr_tl=id&_x_tr_hl=en