Тема:

Представлення чисел у комп'ютері

Лекція 2

Викладач: Порєв Віктор Миколайович

Позиційні системи числення

У типовій позиційній системі числення деяке число *A* представляється множиною цифр: $a_{n-1}\ a_{n-2}\ \dots\ a_1\ a_0$,
й обраховується по формулі:

$$A = a_{n-1} r^{n-1} + a_{n-2} r^{n-2} + \dots + a_2 r^2 + a_1 r + a_0, \qquad (1)$$

де: r – основа системи числення,
n – розрядність,
a_i – цифри числа, зазвичай дорівнюють 0, 1, … , r-1.

Чому така система зветься позиційною? **Позиція цифри** визначає **показник степені *r***. Наприклад, у числі 1035 позиції цифр 1,0,3 та 5 при ***r*** = 10 означають степені 10:

$$1035_{10} = 1\cdot10^3 + 0\cdot10^2 + 3\cdot10^1 + 5\cdot10^0$$

У залежності від ***r*** систему числення називають **десятковою** (***r*** = 10), **двійковою** (***r*** = 2), **вісімковою** (***r*** = 8), **шістнадцятковою** (***r*** = 16). Приклади запису числа 2016 у різних системах

$$2016_{10} = 11111100000_2 = 3740_8 = 7E0_{16}$$

У повсякденному житті люди найчастіше використовують десяткову (***r*** = 10) позиційну систему числення.

Двійкова система числення

Двійкова система посідає особливе місце в комп'ютерах. Вона використовується для внутрішнього уявлення інформації в комп'ютері, оскільки електронні вузли більшості цифрових схем комп'ютера реалізують дворівневу логіку (високий потенціал відповідає 1, низький – 0). Наприклад, для запису у пам'ять процесор видає на адресну шину двійковий код адреси, на шину даних – двійковий код числа.

Цифри (розряди) двійкових чисел називаються **бітами**. Кожний біт може мати значення 0 або 1. Вісім двійкових розрядів утворять **байт**. Наприклад:

$$\mathbf{10110011 = 2\cdot 2^7 + 0\cdot 2^6 + 1\cdot 2^5 + 1\cdot 2^4 + 0\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 = 179_{10}}$$

Звичайний двійковий код, який описується формулою (1), використовується для представлення у комп'ютері *n*-бітових **чисел без знаку**. Діапазон представлення чисел: **від 0 до 2^n-1**.

Вісімкова система числення

Вісімкові числа ($r = 8$) мають цифри від 0 до 7.

Ця система числення використовується не дуже часто.

Шістнадцяткова система числення

Шістнадцяткові числа (r = 16) мають цифри від 0 до 15. Цифри від 0 до 9 записуються звичайно, а для запису цифр від 10 до 15 використовуються букви A – F:

цифри	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
запис	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Шістнадцяткову систему програмісти використовують часто. Можна відмітити традицію записувати адреси пам'яті саме у такій системі. Інший приклад – бітові маски, багато програмістів полюбляють записувати не у двійковому, а саме у шістнадцятовому коді.

Зручність використання шістнадцяткової системи обумовлюється:

- меншою розрядністю чисел, порівняно з системами r=2, 8, 10 (оскільки, чим більше ***r***, тим менше потрібно розрядів)
- простотою переводу у двійкову систему і навпаки

Програміст на Асемблері може використовувати як двійкову, так і більш звичну для людини десяткову, а також вісімкову та шістнадцяткову системи.

Далі розглянемо приклади запису чисел у програмах на Асемблері

Двійкові числа. У програмах на Асемблері запис двійкового числа завершується символом **b** (від англ. *binary*). Наприклад:

```
A db 11101010b
B dw 110011110101b
D dd 10000001001000110100b

mov al,11101010b
mov ax,110011110101b
mov eax,10000001001000110100b
```

Вісімкові числа. Їхній запис повинен завершуватися символом **o** (від англ. *octal*). Наприклад:

```
mov al,352o
mov ax,6365o
mov eax,2011064o
```

Десяткові числа. Наприклад:

```
mov al,234
mov ax,3317
mov eax, 528948
```

Шістнадцяткові числа. Їхній запис завершується символом **h** (від англ. *hex*). Також є ще одна особливість запису: якщо старша (ліва) цифра шістнадцяткового числа є буквою, то **ліворуч дописується нуль**. Наприклад:

```
mov al,0EAh
mov ax,0CF5h
mov eax,81234h
```

Переведення чисел в іншу систему

Обчислення за формулою

Нехай надане двійкове число, наприклад, 1110101_2. Як перевести його в десяткову систему? Для цього можна скористатися безпосередньо формулою обчислення чисел у позиційних системах:

$$1\cdot2^6 + 1\cdot2^5 + 1\cdot2^4 + 0\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 =$$

$$1\cdot64 + 1\cdot32 + 1\cdot16 + 0\cdot8 + 1\cdot4 + 0\cdot2 + 1\cdot1 = 117_{10}$$

Фактично, ми кожну *i*-ту двійкову цифру перетворили у десяткове значення, потім помножили на десяткове 2^i та підсумували всі добутки. **Такий спосіб зручно використовувати для перетворення в десяткову систему**, тому що ми звикли виконувати множення та додавання саме в цій системі. А як перетворити те ж саме число, наприклад, у вісімкову систему? Можна поступити так само, але тут вже потрібно виконувати операції множення та додавання у вісімковій системі:

$$1\cdot[2^6]_8 + 1\cdot[2^5]_8 + 1\cdot[2^4]_8 + 0\cdot[2^3]_8 + 1\cdot[2^2]_8 + 0\cdot[2^1]_8 + 1\cdot[2^0]_8 =$$

$$1\cdot100_8 + 1\cdot40_8 + 1\cdot20_8 + 0\cdot1_8 + 1\cdot4_8 + 0\cdot2_8 + 1\cdot1_8 = ?$$

Групування бітів

Для перетворення **двійкового числа у вісімкове** існує дуже простий алгоритм. Розряди двійкового числа записують групами по три біти кожна, починаючи з молодших розрядів, і кожна **трійка бітів утворює одну** вісімкову цифру. Наприклад:

```
1011110101 = 1 011 110 101
             1   3   6   5  = 1365₈
```

Так само легко перетворювати **з двійкової в шістнадцяткову** систему. Тут одна шістнадцяткова цифра складається з **чотирьох бітів**. Групувати четвірки бітів треба починати з молодших бітів:

```
1011110101 = 10 1111 0101
              2    F    5   = 2F5₁₆
```

Алгоритм ділення на *r*

Може використовуватися для будь-яких цілих *r*. Виконуються такі дії:

1. Ділимо число на основу нової системи числення. Залишок від цілочисельного ділення є молодшою цифрою результату
2. Часткове знову ділимо на основу системи. Залишок від ділення є наступною цифрою результату
3. І так далі, таке ділення продовжуємо доки часткове не стане дорівнювати нулю.

Наприклад, перетворимо число 91_{10} у двійкову систему послідовним діленням на 2:

91 : 2 = 45, залишок **1** (це молодший біт результату)
45 : 2 = 22, залишок **1**
22 : 2 = 11, залишок **0**
11 : 2 = 5, залишок **1**
5 : 2 = 2, залишок **1**
2 : 2 = 1, залишок **0**
1 : 2 = 0, залишок **1** (це старший біт)

Результат = $\mathbf{1011011}_2$.

Представлення цілих чисел зі знаком

В позиційній системі числення число $A \geq 0$ представляється у вигляді набору цифр

$A = a_{n-1}\ a_{n-2} \ldots\ a_1\ a_0$

Як представити від'ємне число *A*, якщо всі цифри a_i невід'ємні?

Для цього використовуються спеціальні коди.

Розглянемо деякі різновиди таких кодів, які мають відношення до архітектури x86.

Прямий код

Для представлення позитивних та від'ємних чисел у прямому коді додається окремий розряд для знаку числа. Наприклад: +26, -11. Люди зазвичай позитивним числам знак "+" не дописують.

В електронних цифрових комп'ютерах використовується двійкова система. Кожний розряд – це один біт. Для знаку виділили один біт – він зветься знаковим, а інші розряди – значущими. **В прямому двійковому коді знак "+" кодується нулем, а знак "–" одиницею**. Знаковий біт записується ліворуч. Наприклад:

0 011010_2 $(+26_{10})$

1 001011_2 (-11_{10})

Якщо загалом, повинно бути n розрядів (включно зі знаком), то прямий двійковий код можна описати так:

$$A = s\ a_{n-2} \ldots a_1\ a_0,$$

де s – знаковий біт.

Незважаючи на простоту запису, у процесорах сімейства x86 та їм подібних, **прямий код для форматів цілих чисел не використовується.**

Таке кодування прийнято для чисел **у форматах з плаваючою точкою.**

Додатковий код

Додатковий n-розрядний код числа A можна описати у такий спосіб:

$$[A]_{дк} = A \quad (якщо\ A \geq 0)$$
$$= r^n - |A| \quad (якщо\ A < 0)$$

тут вважається, що для представлення |A| достатньо (n-1) розрядів звичайного позиційного коду.

Результат віднімання $r^n - |A|$ дає число з цифрами, які позначимо d_i.

Для двійкового додаткового коду (англ. *two's complement*) від'ємних чисел цифра лівого (старшого) розряду дорівнює 1:

$$\begin{array}{rccccccccc} & 1 & 0 & 0 & 0 & . & . & . & 0 & 0 & (2^n) \\ - & & & & & & & & & & \\ & & & a_{n-2} & a_{n-3} & . & . & . & a_1 & a_0 & (|A|) \\ \hline [A<0]_{дк} = & 1 & d_{n-2} & d_{n-3} & . & . & . & d_1 & d_0 & & \end{array}$$

Додатковий код

Для невід'ємних (A≥0) чисел *n*-розрядний додатковий код утворюється так: до наявних (*n*-1) цифр звичайного коду

$$A = a_{n-2}\ a_{n-3}\ .\ .\ .\ a_1\ a_0$$

ліворуч просто дописується 0

$$[A]_{дк} = 0\ a_{n-2}\ a_{n-3}\ .\ .\ .\ a_1\ a_0$$

Старший розряд називають знаковим – його цифра прямо вказує на знак числа.

Додатковий код

Для перетворення в додатковий код від'ємного числа ($A < 0$) у двійковій системі числення замість віднімання можна скористатися таким алгоритмом:

1. Спочатку до цифр числа $|A|$ ліворуч дописується 0
2. Виконується порозрядна інверсія усіх бітів
3. Додається +1 в молодший розряд

Наприклад, якщо потрібен 8-бітовий д.к. числа (-26), то:

```
  0011010    7-бітовий код числа 26
 00011010    ліворуч дописаний 0
 11100101    порозрядна інверсія
       +1
------------
 11100110  = [-26]дк
```

Додатковий код

Подібний алгоритм можна використовувати також і для зміни знаку числа в додатковому коді. Тут потрібно виконувати тільки порозрядну інверсію та +1 в молодший розряд. Наприклад:

```
  11100110  = [-26]дк
  00011001    порозрядна інверсія
        +1
------------
  00011010  = [+26]дк
```

Корисна властивість додаткового коду: для зміни розрядності (*n*) лівий біт може розмножуватися, при цьому значення закодованого числа не змінюється.

Додатковий код

Приклади запису чисел у **8-бітовому** додатковому коді:

```
10000000  -128
10000001  -127
10000010  -126
 . . .
11111110  -2
11111111  -1
00000000   0
00000001   1
00000010   2
 . . .
01111110   126
01111111   127
```

Таким чином, у 8-бітовому додатковому коді можуть представлятися числа **від -128 до +127**.

Додатковий код

Приклади запису чисел у **16-бітовому** додатковому коді:

```
1000000000000000  -32768
1000000000000001  -32767
1000000000000010  -32766
      . . .
1111111111111110  -2
1111111111111111  -1
0000000000000000   0
0000000000000001   1
0000000000000010   2
      . . .
0111111111111110   32766
0111111111111111   32767
```

Таким чином, у 16-бітовому додатковому коді можуть представлятися числа **від -32768 до +32767**.

Загалом, для ***n*-бітового** двійкового додаткового коду діапазон чисел сягає **від -2^{n-1} до $+2^{n-1} - 1$**.

Додатковий код

Перевагою додаткового коду в порівнянні з прямим кодом є зручність виконання додавання та віднімання чисел. Знакові розряди обробляються так само, як інші розряди.

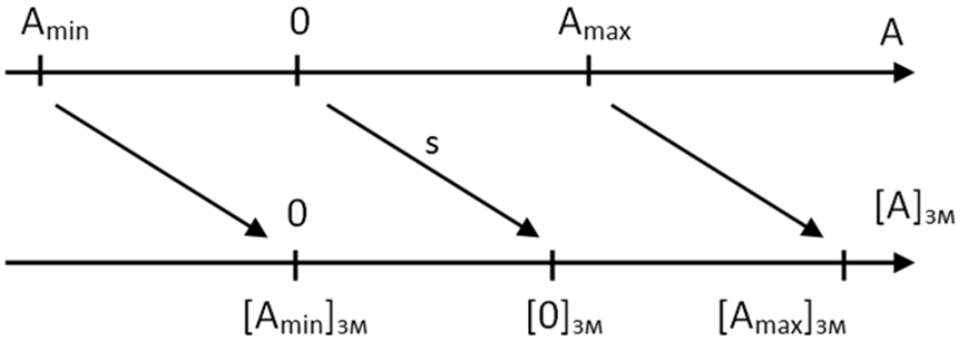
В цифрових комп'ютерах (у тому числі архітектури сімейства x86) додатковий код широко використовується для представлення цілих чисел зі знаком.

Зміщений код

До числа додається зсув – від'ємні числа перетворюються у позитивні значення, які потім записуються у звичайному двійковому коді.

$[A]_{зм} = A + s.$

Зсув визначається мінімальним від'ємним числом із діапазону чисел, які потрібно представляти. На числовій осі зсув можна відобразити так:



Зміщений код

Розглянемо приклад зміщеного 8-бітового двійкового коду.
Нехай $[A]_{зм} = A + 127$, тоді

```
00000000      -127
00000001      -126
. . . .
01111110      -1
01111111       0
. . . .
10000000      +1
11111110      +127
11111111      +128
```

Тут діапазон представлення чисел **від -127 до +128**.

Зміщене кодування використовують для представлення експонент у форматах з плаваючою точкою.

Представлення дробових чисел

Представлення дробових чисел

Дробові числа люди зазвичай записують так: ліворуч ціла частина, потім точка або кома, а праворуч дробова частина. Наприклад:

13.453125

У звичному для комп'ютера двійковому коді це число можна записати так:

1101.011101

Здавалося, розробникам комп'ютерів можна було б прийняти такий спосіб представлення дробових чисел – у розрядній сітці зарезервувати декілька розрядів для цілої частини і декілька розрядів для дробової. Такий формат зветься представленням чисел з **фіксованою точкою** і колись використовувався у деяких комп'ютерах.
Для програмістів **цей формат виявився дуже незручним**, оскільки треба писати програми так, щоб при обчисленні будь-яких арифметичних виражень масштабувати дані, щоб запобігти виходу за межі розрядної сітки або втрату точності.

Формати з плаваючою точкою

Формати з плаваючою точкою

У сучасних цифрових комп'ютерах часто використовують формати з плаваючою точкою (англомовна назва: ***floating point***). Математичний опис таких форматів:

$$V = r^E \cdot M,$$

де r – основа системи числення, E – експонента (r^E інколи називають порядком), M – мантиса.

Люди інколи також використовують таке представлення чисел – особливо при запису дуже малих або дуже великих значень. Наприклад, замість

0.00000000000000000000000000000000000000000000000000000000000921

як здається, зручніше записати $10^{-61} \cdot 9.21$ або 9.21e-61,
а замість

257000000000000000000000000000000000000

краще записати $10^{40} \cdot 2.57$ або 2.57e+40.

Формати з плаваючою точкою

Цей формат можна використовувати не тільки для представлення дробових чисел, а і для представлення цілих чисел.

Позитивною рисою формату з плаваючою точкою **є універсальність**, можливість однотипного представлення числових значень **у широкому діапазоні**. Це зручно для програмістів.

Недоліком є складніше виконання процесором операцій у форматах з плаваючою точкою порівняно із цілочисельними форматами. Для багатьох процесорів старих зразків операції з плаваючою точкою виконувалися **набагато довше** (в десятки разів) аніж цілочисельні арифметичні операції. В сучасних процесорах арифметичні операції над числами у форматах з плаваючою точкою виконуються майже так саме швидко, як і для цілочисельних форматів, проте і зараз існують значні відмінності у обробці чисел в різних форматах. Зокрема, у архітектурі x86 обробка чисел у цілочисельних форматах і форматах з плаваючою точкою розділена по окремим блокам.

Для форматів з плаваючою точкою існують офіційні стандарти.

Стандарт IEEE 754 та його розвиток

У 1985 році був прийнятий стандарт IEEE 754. У цьому стандарті, зокрема, міститься опис двійкових форматів чисел з плаваючою точкою. Ці формати зараз підтримуються у багатьох комп'ютерних архітектурах, у тому числі процесорах сімейства x86.

Кроки стандартизації форматів та арифметики з плаваючою точкою:

1. Перша версія стандарту: IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Std 754-1985

2. Пізніше був прийнятий стандарт міжнародної комісії IEC: IEC 60559:1989, Binary floating-point arithmetic for microprocessor systems

3. У 2008 році була оприлюднена нова версія стандарту IEEE 754: IEEE Standard for Floating-Point Arithmetic. IEEE Std 754-2008 (Revision of IEEE Std 754-1985). У цьому варіанті стандарту запропоновані деякі нові різновиди форматів

Стандарт IEEE 754 та його розвиток

Нижче у таблиці наданий перелік стандартних двійкових форматів, які апаратно підтримуються у процесорах сімейства x86

Оригінальна назва формату	Назва згідно стандарту IEEE 754 версії 2008	Розрядність (біт)	Точність (біт)	Діапазон представлення чисел	Примітка
Half precision floating-point	Binary 16	16	11	±(від 2^{-14} до 65504)	Апаратно реалізовані тільки операції перетворення даних: команди VCVTPH2PS та VCVTPS2PH
Single precision floating-point	Binary 32	32	24	±(від 2^{-126} до 2^{+127})	Один з двох основних форматів з плаваючою точкою
Double precision floating-point	Binary 64	64	53	±(від 2^{-1022} до 2^{+1023})	Один з двох основних форматів з плаваючою точкою
Double extended precision floating-point		80	64	±(від 2^{-16382} до 2^{+16383})	Тільки для команд x87 FPU

Стандарт IEEE 754 та його розвиток

У стандарті IEEE 754-2008 передбачені й інші формати, зокрема двійковий 128-бітовий формат, який зветься Binary128. Цей формат процесорами сімейства x86 апаратно не підтримується (поки що взагалі невідомо, які процесори інших типів підтримують цей формат). Відомо про підтримку цього формату на програмному рівні компіляторами деяких мов програмування.

Потрібно розрізнювати **апаратну** і **програмну** реалізації підтримки певних форматів із плаваючою точкою. Програмно можуть бути реалізовані операції над даними взагалі у будь-якому форматі, проте **швидкість роботи при апаратній підтримці набагато вища**.

Стандартні формати з плаваючою точкою

Розглянемо представлення чисел згідно стандарту IEEE 754. Узагальнений опис двійкового формату можна надати у такому вигляді:

$$V = (-1)^S \, 2^E \cdot M,$$

де S – знак, E – експонента, M – мантиса.

Мантиса (M) складається з цілої та дробової частини. Рекомендовано використовувати так звані **нормалізовані** мантиси. Нормалізовані двійкові мантиси знаходяться у діапазоні від 1 до 2 і відповідно мають такі двійкові коди

1.000...0 (дорівнює 1)

. . . . .

1.111...1 (дорівнює $2-2^{-n}$, де n-розрядність дробової частини)

Нормалізована мантиса позначається як $1.F$, де F – дробова частина. У чому сенс нормалізованих мантис? Вони дозволяють заощадити пам'ять. Тут можна вказати такі фактори

Стандартні формати з плаваючою точкою

1. Використання ненормалізованих мантис могло б призвести до того, що одне й те саме число у форматі з плаваючою точкою можна було б представляти декількома різними кодами. Наприклад, десяткове число 1.5 у двійковому форматі з плаваючою точкою можливо було б представляти як

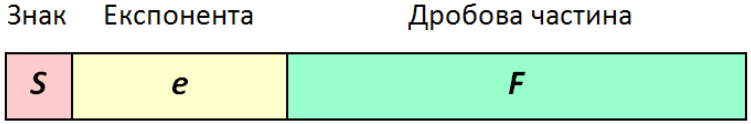
$2^0 \cdot$ **1.1000..0** (мантиса нормалізована)
або
$2^1 \cdot$ **0.1100..0** (тут мантиса є ненормалізованою)
або
$2^2 \cdot$ **0.0110..0** (тут також ненормалізована мантиса)
і так далі.

Не тільки наведене вище число 1.5, а й багато інших чисел можна представляти різними варіантами кодів – це є ознакою збитковості, зайвого використання розрядної сітки у випадку ненормалізованих мантис.

2. Оскільки нормалізована мантиса 1.*F* має біт цілої частини, який завжди дорівнює 1, то цей біт не записується у пам'ять – він зветься "схованим бітом". Так заощаджується один біт пам'яті для запису іншої інформації – наприклад, збільшення на один біт розрядності дробової частини підвищує точність представлення чисел удвічі.

Експонента (*E*) записується у зміщеному коді *e* = *E* + *зсув*.

Таким чином, для чисел у двійковому форматі з плаваючою точкою у пам'яті зберігаються такі компоненти:



Сумарна кількість бітів ***S***, ***e*** та ***F*** означають розрядність певного двійкового формату з плаваючою точкою.

Стандартні формати з плаваючою точкою

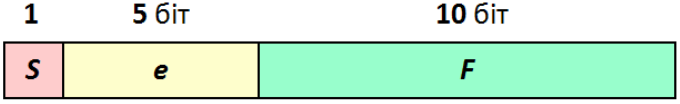
Передбачено, що деякі комбінації бітів коду двійкового формату із плаваючою точкою використовуються для позначення особливих випадків – **NaN** (не число *Not a Number*), **безкінечність** тощо. Це наведено у таблиці нижче.

Зміщена експонента $e = E + зсув$	Дроб F	Значення числа V	Назва числа
$e = e_{min} - 1$	$F = 0$	$V = (-1)^S 0$	нуль
$e = e_{min} - 1$	$F \neq 0$	$V = (-1)^S 2^{min} (0.F)$	ненормалізоване число
$e_{min} \leq e \leq e_{max}$	F	$V = (-1)^S 2^E (1.F)$	нормалізоване число
$e = e_{max} + 1$	$F = 0$	$V = (-1)^S \infty$	безкінечність зі знаком
$e = e_{max} + 1$	$F \neq 0$	$V = NaN$	не число

Робота процесора є звичайною, коли він обробляє нормалізовані числа або нуль. В інших випадках генеруються відповідні виключення

Половинний 16-бітовий формат

Цей формат для процесорів x86 є відносно новим. Використання цього формату обмежене двома командами перетворення даних. Назва формату Binary16 Half Precision. Одне число у цьому форматі потребує 16 біт пам'яті. Формат має такі складові:



Мантиса має вигляд $M = 1.F$
Експонента записується у зміщеному коді $e = E+15$
Діапазон для експоненти E: від E_{min} = -14 до E_{max}=+15

Мінімальне за абсолютною величиною ненульове число, яке можна представити у цьому форматі:
V_{min} = 0 00001 0000000000 = $+2^{-14}\times 1.0000000000 = 2^{-14} = 0.00006103515625$

Максимальне (небезкінечне) число:
V_{max} = 0 11110 1111111111 = $+2^{15}\times 1.1111111111 = 1111111111100000 = 65504$

Половинний 16-бітовий формат

e	E	Примітка
00000		0, або ненормалізоване число
00001	-14	E_{min}
. . .		
01110	-1	
01111	0	
10000	+1	
. . .		
11101	+14	
11110	+15	E_{max}
11111		$\pm\infty$, або NaN

Діапазон представлення чисел: ±(від 2^{-14} до 65504).

Цілі числа від нуля до 2048 передаватимуться як є
Цілі числа від 2049 до 4096 округлюються до найближчого парного цілого
Цілі числа від 4097 до 8192 округлюються до найближчого цілого, кратного 4
Цілі числа від 8193 до 16384 округлюються до найближчого цілого, кратного 8
Цілі числа від 16385 до 32768 округлюються до найближчого цілого, кратного 16
Цілі числа від 32769 до 65504 округлюються до найближчого цілого, кратного 32

Одинарний 32-бітовий формат

Одинарний (*single precision*) 32-бітовий формат є одним з основних форматів архітектури x86

Мантиса має вигляд $M = 1.F$

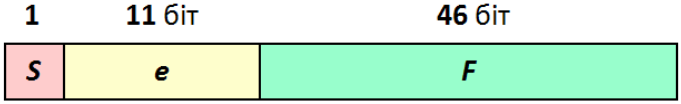
Експонента записується у зміщеному коді $e = E+127$
Діапазон для експоненти E: від E_{min} = -126 до E_{max}=+127

Виходячи з цього, можна оцінити діапазон представлення чисел у такому форматі приблизно як: ±(від 2^{-126} до 2^{+127})

Точність представлення чисел - приблизно 6-7 десяткових розрядів

Подвійний 64-бітовий формат

Цей формат є одним з основних форматів для процесорів x86.
Стосовно термінології. Необхідно зазначити, що, у стандарті та інших англомовних документах цей формат зветься *"double precision floating point"*, а розглянутий вище 32-бітний формат зветься як *"single precision floating point"*. Треба розуміти, що 64-бітний формат точніший не удвічі ("*double precision*"), а значно більше – його точність вища у мільйони разів, аніж у 32-бітного формату. У 64-бітного формату подвійна **довжина коду**, а не точність



Мантиса має вигляд $M = 1.F$

Експонента записується у зміщеному коді $e = E+1023$
Діапазон для експоненти E: від E_{min} = -1023 до E_{max}=+1023

Виходячи з цього, можна оцінити діапазон представлення чисел у такому форматі приблизно як: ±(від 2^{-1022} до 2^{+1023})

Точність представлення чисел - приблизно 16-17 десяткових розрядів

Розширений 80-бітовий формат

Цей формат (англ. “*double extended precision floating point*”) є нестандартним, використовується командами FPU x87. Апаратно підтримується блоком FPU.

Особливістю цього формату, окрім більшої розрядності, є можливість обробляти ненормалізовані числа. Біт цілої частини *(j)* мантиси записується у пам’ять - він може бути і нулем.

Мантиса має вигляд $M = j.F$

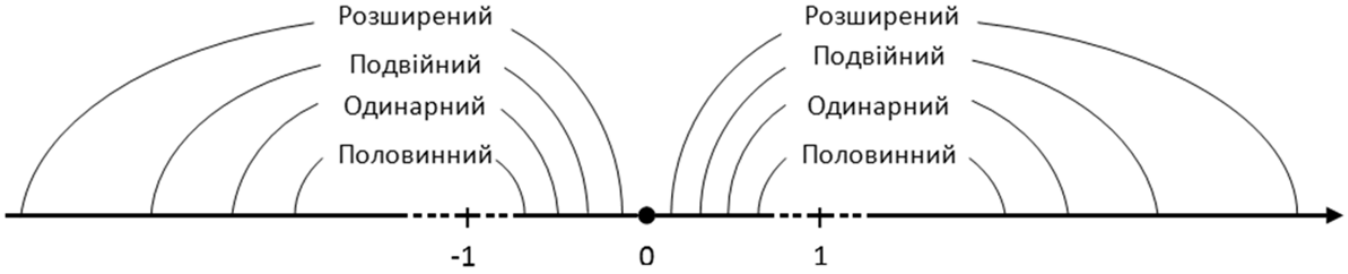
Експонента записується у зміщеному коді $e = E+16383$
Діапазон для експоненти E: від E_{min} = -16382 до E_{max}=+16383

Діапазон (для нормалізованих чисел): приблизно ±(від 2^{-16382} до 2^{+16383})

Точність представлення чисел - приблизно 20 десяткових розрядів

Порівняння двійкових форматів з плаваючою точкою

Формат	Діапазон значень
Половинний, 16 біт	± (від 2^{-14} до 65504), а також нуль
Одинарний, 32 біт	± (від $3.4 \cdot 10^{-38}$ до $3.4 \cdot 10^{+38}$), а також нуль
Подвійний, 64 біт	± (від $1.7 \cdot 10^{-308}$ до $1.7 \cdot 10^{+308}$), а також нуль
Розширений, 80 біт	± (від $3.4 \cdot 10^{-4932}$ до $1.1 \cdot 10^{+4932}$), а також нуль



Ілюстрація на числовій осі співвідношення діапазонів представлення чисел

Порівняння двійкових форматів з плаваючою точкою

При використанні форматів з плаваючою точкою необхідно враховувати те, що **діапазон** представлення чисел зростає разом із збільшенням кількості двійкових розрядів **експоненти**.

Точність представлення дробових чисел зростає із збільшенням кількості двійкових розрядів **мантиси**.

Таким чином, розширений 80-бітовий формат має найбільший діапазон та найвищу точність.

Контрольне запитання. Як у комп'ютері представити десяткове число **0.1** ?