2.4 Лабораторна робота 4. Виконання операцій додавання та віднімання чисел з плаваючою комою

Мета роботи:

- вивчення форматів чисел з плаваючою комою стандарту *IEEE 754-2008*;
- вивчення алгоритмів виконання математичних операцій додавання та віднімання чисел з плаваючою комою;
- вивчення стандартних прийомів програмної реалізації математичних операції над числами з плаваючою комою;
- отримання навичок розробки програм, що виконують операції над числами з плаваючою комою.

2.4.1 Завдання

Порядок виконання лабораторної роботи:

1. Визначити варіант завдання по табл. 2.10, 2.11 відповідно до молодших розрядів $h_4 \dots h_1$ номеру залікової книжки, записаного у двійковій системі числення.

2. Перевести числа з десяткової системи числення в формати *IEEE 754-2008* (табл. 2.10). Проміжні обчислення оформити як у прикладі табл. 2.15.

3. Розробити блок-схему алгоритму виконання заданої математичної операції над числами з плаваючою комою.

4. Реалізувати математичну операцію програмно. Програма має підтримувати нормалізовані числа та число нуль. Підтримку денормалізованих чисел, нескінченностей та невизначеностей не реалізовувати. Виконувати ведення і виведення операндів за допомогою апаратних засобів стенду: клавіатури та індикаторів або РК дисплея.

Табл. 2.10. Числа для ручного переведення

h_2h_1	**Перевести в *IEEE 754***	**Перевести в десяткову систему**	**Формат числа**
00	63,8125	`c3 d4 00 00`	*binary32*
	−259,34375	`3f c2 80 00 00 00 00 00`	*binary64*
01	−47,6875	`45 18 00 00`	*binary32*
	304,59375	`40 67 38 00 00 00 00 00`	*binary64*
10	92,5625	`c4 7a 08 00`	*binary32*
	−203,78125	`40 3b b0 00 00 00 00 00`	*binary64*
11	−38,4375	`3c 80 00 00`	*binary32*
	367,84375	`c0 45 20 00 00 00 00 00`	*binary64*

Табл. 2.11. Варіанти математичних операцій

h_4h_3	**Операція**	**Формат операндів та результату**
00	Додавання	*binary32*
01	Віднімання	*binary32*
10	Додавання	*binary64*
11	Віднімання	*binary64*

2.4.2 Формати чисел з плаваючою комою по стандарту *IEEE 754-2008*

Стандарт *IEEE 754-2008* визначає формати чисел з плаваючою комою, правила виконання математичних операцій над цими числами, правила округлення, виникнення та обробку виключних ситуацій та інше. Для виконання даної роботи необхідно в першу чергу вивчити формати представлення чисел з плаваючою комою. Стандарт визначає формати для двійкової та для десяткової систем числення різної точності. Структура числа з плаваючою комою (рис. 2.4) однакова для всіх форматів:

- Знак мантиси s: «0» — додатна, «1» — від'ємна.

- Зміщений порядок (покажчик степеня, експонента):

$$e_{\text{зм}} = e + 2^{N_e - 1} - 1,$$

  де e — істинний порядок, N_e — кількість біт, відведена для запису порядку. Завдяки зміщенню, значення зміщених порядків $e_{\text{см}}$ завжди є додатними.

- Мантиса m довжиною N_m біт. Нормалізована завжди, коли це можливо (тобто, коли можна зберігати відповідне значення порядку).

Значення числа з плаваючою комою визначається за формулою:

$$X = m \cdot 2^e.$$

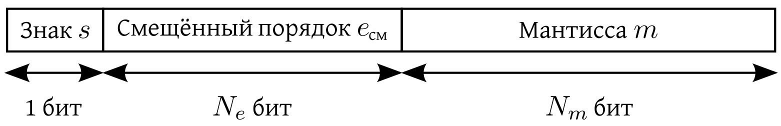

Рис. 2.4. Загальна структура форматів чисел з плаваючою комою стандарту *IEEE 754-2008*

Стандарт визначає п'ять форматів, призначених для виконання обчислень. В даній роботі ми обмежмось двома найбільш широко вживаними форматами. Назва кожного формату складається з назви системи числення та кількості біт, відведених для запису числа. Найбільш розповсюдженими форматами є *binary32* (відповідає вбудованому типу `float` мови Сі) та *binary64* (відповідає типу *double*). Детальна інформація про ці формати представлена в табл. 2.12, а структура чисел представлена на рис. 2.5, 2.6.

2.4.3 Нормалізація

Число з плаваючою комою називається нормалізованим якщо виконується нерівність:

$$1 \leq |m| < q,$$

Табл. 2.12. Формати чисел з плаваючою комою стандарту *IEEE 754-2008*

Назва	**Основа**	**К-ть біт мантиси** N_m	**К-ть біт порядку** N_e	$\min(e)$	$\max(e)$
binary32 *Single Precision* (одинарна точність)	2	$23+1$	8	-126	$+127$
binary64 *Double Precision* (подвійна точність)	2	$52+1$	11	-1022	$+1023$

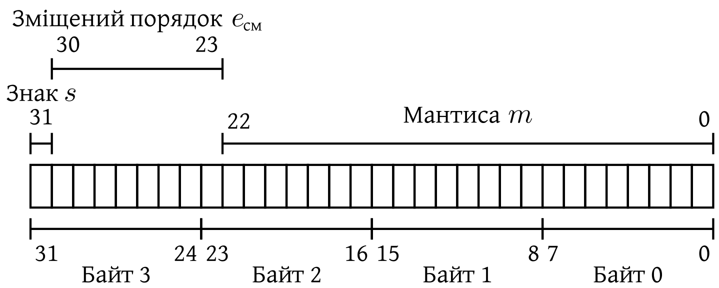

Рис. 2.5. Формат числа з плаваючою комою *binary32*

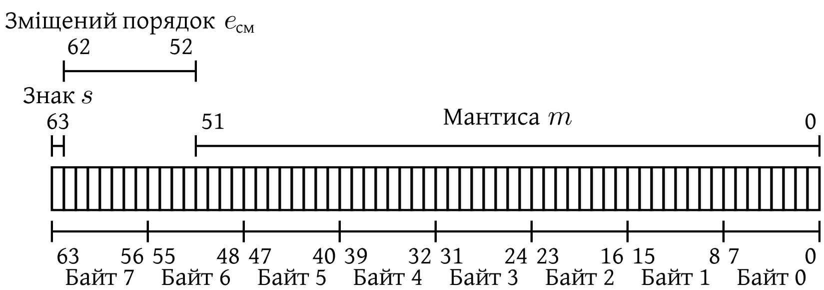

Рис. 2.6. Формат числа з плаваючою комою *binary64*

де m — мантиса, q — основа системи числення. Так як ми розглядаємо формати чисел з плаваючою комою у двійковій системі числення:

$$1 \leq |m| < 2.$$

Звідси маємо, що перша значуща цифра мантиси нормалізованого числа завжди є його цілою частиною та рівна 1. Цю цифру в форматах *IEEE 754-2008 не зберігають*. Зберігається тільки дробова частина мантиси.

Очевидно що неможливо представити число 0 в нормалізованому вигляді. Тому нуль зберігається як денормалізоване число. Стандарт передбачає два нулі: ± 0. Кодування денормалізованих чисел розглянуто далі.

Для того щоб нормалізувати число, необхідно зсувати кому мантиси до тих пір, доки в цілій частині не залишиться тільки одна значуща цифра (одиниця). Абсолютне значення порядку рівне кількості зсувів коми, а знак порядку визначається напрямом зсуву. Якщо кому зсували вліво — порядок додатній, якщо вправо — від'ємний.

2.4.4 Види чисел з плаваючою комою

Окрім звичайних нормалізованих чисел стандарт визначає деякі інші види чисел, які можуть з'являтись при обчисленнях. В цьому випадку замість дійсних порядку та мантиси в числі зберігаються спеціальні значення, які заборонені для запису звичайних чисел (табл. 2.13, 2.14).

- Якщо $1 \leq e_{\text{см}} \leq 2^{N_e} - 2$, то число нормалізоване. В цьому випадку старший біт мантиси (який не зберігають) рівний 1.
- Якщо $e_{\text{см}} = 0$, а $m \neq 0$, то число денормалізоване. В цьому випадку старший біт мантиси (який не зберігають) рівний 0.
- Якщо $e_{\text{см}} = 0$ та $m = 0$, то це ± 0 в залежності від знаку.
- Якщо $e_{\text{см}} = 2^{N_e} - 1$ та $m = 0$, то це $\pm\infty$ в залежності від знаку.

Табл. 2.13. Види чисел з плаваючою комою в форматі *binary32*

Число	s	e	$e_{\text{зм}}$	m
$+0$	0	-127	0	0
-0	1	-127	0	0
$+\infty$	0	$+128$	255	0
$-\infty$	0	$+128$	255	0
Денормалізоване	0 або 1	-127	0	не-нуль
Нормалізоване	0 або 1	$-126 \leq e \leq +127$	$1 \leq e_{\text{зм}} \leq 254$	будь-яка
NaN	0 або 1	$+128$	255	не-нуль

Табл. 2.14. Види чисел з плаваючою комою в форматі *binary64*

Число	s	e	$e_{\text{зм}}$	m
$+0$	0	-1023	0	0
-0	1	-1023	0	0
$+\infty$	0	$+1024$	2047	0
$-\infty$	0	$+1024$	2047	0
Денормалізоване	0 або 1	-1023	0	не-нуль
Нормалізоване	0 або 1	$-1022 \leq e \leq +1023$	$1 \leq e_{\text{зм}} \leq 2046$	будь-яка
NaN	0 або 1	$+1024$	2047	не-нуль

- Якщо $e_{\text{см}} = 2^{N_e} - 1$ та $m \neq 0$, то це невизначеність *NaN* (англ. *Not a Number*, не число).

2.4.5 Переведення чисел в формати *IEEE 754-2008*

Розглянемо процес переводу в формат *binary32* на прикладі числа $-118{,}625$. Знак, порядок та мантиса визначаються наступним чином.

1. **Визначення знаку мантиси.** Так як число від'ємне, то

знаковий біт $s = 1$.

2. **Переведення модуля у двійкову систему числення:**

$$-118{,}625_{10} = 1110110{,}101_2.$$

3. **Виконання нормалазації.** Для цього будемо зсувати кому до тих пір, доки зліва від неї не залишиться одна одиниця. Абослютне значення порядку рівне кількості зсувів. Так як кому зсували вліво, то порядок додатній.

$$\begin{aligned} 1110110{,}101 &= 1110110{,}101 \cdot 2^0 = \\ &= 111011{,}0101 \cdot 2^1 = \\ &= 11101{,}10101 \cdot 2^2 = \\ &\dots \\ &= 1{,}110110101 \cdot 2^6 = m \cdot 2^e. \end{aligned}$$

4. **Визначення двійкового коду мантиси.** Ціла частина нормалізованого числа завжди рівна одиниці і тому в форматах *IEEE 754-2008* не зберігається. Дробову частину доповнюємо справа не значущими нулями до необхідної довжини та отримуємо двійковий код, що зберігається як мантиса:

$$11011010100000000000000.$$

5. **Визначення зміщеного порядку.** Істинний порядок $e = 6$. Обчислюємо зміщений порядок:

$$e_{\text{зм}} = e + 2^{N_e - 1} - 1 = 6 + 127 = 133.$$

Переводимо $e_{\text{зм}}$ у двійкову систему числення:

$$e_{\text{зм}} = 133_{10} = 10000101_2.$$

Таким чином, число $-118{,}625$ в форматі *binary32* буде мати наступний вигляд:

s	$e_{\text{зм}}$	m
1	10000101	11011010100000000000000

Або у вигляді байтів: `c2 ed 40 00`.

Розглянемо переведення дробових чисел у двійкову систему числення детальніше. Для цього необхідно окремо перевести цілу і дробові частини. Ціла частина переводиться діленням на 2 та аналізом остачі. Дробова частина — множенням на 2 та аналізом цілої частини. Наприклад, для дробової частини 0,1875:

$$\begin{gathered}0{,}1875 \cdot 2 = 0{,}375\\ 0{,}375 \cdot 2 = 0{,}75\\ 0{,}75 \cdot 2 = 1{,}5\\ 0{,}5 \cdot 2 = 1{,}0\\ 0{,}1875_{10} = 0{,}0011_2\end{gathered}$$

Декілька прикладів переведення крок за кроком розібрані в табл. 2.15, 2.16, 2.17.

Для відлагодження програм в табл. 2.18 наведено ще декілька додатних чисел з плаваючою комою у вигляді байтів. Нагадаємо, що від'ємні числа можна отримати із додатній проінвертувавши знаковий біт.

2.4.6 Організація виконання операцій над числами з плаваючою комою

В програмі на Сі числа з плаваючою комою будемо представляти у вигляді структури:

```
struct ieee_binary32
{
  uint8_t bytes[4];
};

struct ieee_binary64
{
  uint8_t bytes[8];
};
```

Табл. 2.15. Приклад переведення числа в формат *binary32*

Десятковий код:	0,03125 = 1/32
Двійковий код:	0,00001
Нормалізований вид:	$1.0 \cdot 2^{-5}$
Зміщений порядок:	$-5 + 127 = 122$
Порядок у двійковому коді:	01111010
Знак:	0
Результат:	0 01111010 0000000000000...
Результат у вигляді байтів:	`3d 00 00 00`

Табл. 2.16. Приклад переведення числа в формат *binary32*

Десятковий код:	100
Двійковий код:	1100100
Нормалізований вид:	$1{,}100100 \cdot 2^{6}$
Зміщений порядок:	$6 + 127 = 133$
Порядок у двійковому коді:	10000101
Знак:	0
Результат:	0 10000101 1001000000000...
Результат у вигляді байтів:	`42 c8 00 00`

Табл. 2.17. Приклад переведення числа в формат *binary64*

Десятковий код:	$-11{,}875$
Двійковий код:	1011,111
Нормалізований вид:	$1{,}011111 \cdot 2^{3}$
Зміщений порядок:	$3 + 1023 = 1026$
Порядок у двійковому коді:	10000000010
Знак:	1
Результат:	1 10000000010 0111110000...
Результат у вигляді байтів:	`c0 27 c0 00   00 00 00 00`

Табл. 2.18. Числа з плаваючою комою в форматах *binary32* та *binary64*

Десятковий код	***binary32***	***binary64***	
+1	3f 80 00 00	3f f0 00 00	00 00 00 00
+2	40 00 00 00	40 00 00 00	00 00 00 00
+3	40 40 00 00	40 08 00 00	00 00 00 00
+4	40 80 00 00	40 10 00 00	00 00 00 00
+5	40 a0 00 00	40 14 00 00	00 00 00 00
+10	41 20 00 00	40 24 00 00	00 00 00 00
+15	41 70 00 00	40 2e 00 00	00 00 00 00
+50	42 48 00 00	40 49 00 00	00 00 00 00

```
typedef struct ieee_binary32 my_float;
typedef struct ieee_binary64 my_double;
```

Формати стандарту *IEEE 754-2008* спроектовані в першу чергу з метою створення ефективних апаратних обчислювальних пристроїв. Програмна робота з числами в форматах *binary32* та *binary64* не зручна, так як межі байт не збігаються з межами окремих компонент числа (знаку, порядку, мантиси): див. рис. 2.5, 2.6.

Числа в форматах *IEEE 754-2008* будемо називати *упакованими*. Перед виконанням будь-якої операції над ними необхідно виконати *розпаковку* — вибірку окремих компонентів числа в окремі локальні змінні. Математична операція завжди виконується над числами в розпакованому вигляді. Після виконання операції результат *упаковують* у потрібний формат *IEEE 754-2008*.

Одні математичні операції зручно виконувати в ДК (додавання, віднімання), а інші – в ПК (множення, ділення). Тому в залежності від операції виконують розпаковку числа в три змінні (знак, істинний порядок в ДК, мантиса в ПК) або в дві (істинний порядок в ДК, мантиса в ДК).

Розпаковка в три змінні (знак, зміщений порядок, мантиса в ПК).

```
void my_float_unpack_to_ems(
  const my_float *f, /* покажчик на число, що розпаковується */
  int8_t  *e,        /* покажчик на істинний порядок в ДК */
  uint32_t *m,       /* покажчик на мантису в ПК */
  uint8_t  *s)       /* покажчик на знак */
{
  uint8_t b0 = f->bytes[0];
  uint8_t b1 = f->bytes[1];
  uint8_t b2 = f->bytes[2];
  uint8_t b3 = f->bytes[3];
  uint8_t tmp_e = ((b0 << 1) | (b1 >> 7));
  *s = b0 & 0x80;
  if(tmp_e != 0)
  {
    /* Нормалізоване число */
    *m = ((0x80 | b1) << 16) | (b2 << 8) | b3;
  }
  else
  {
    /* Денормалізоване число або нуль */
    *m = ((b1 & 0x7f) << 16) | (b2 << 8) | b3;
  }
  *e = tmp_e - 127; /* Отримання істинного порядку */
}
```

Розпаковка в дві змінні (істинний порядок в ДК, мантиса в ДК).

```
void my_float_unpack_to_em(
  const my_float *f, /* покажчик на число, що розпаковується */
  int8_t  *e,        /* покажчик на істинний порядок в ДК */
  int32_t *m)        /* покажчик на мантису в ДК */
{
  uint32_t tmp_m;
  uint8_t tmp_s;
  my_float_unpack_to_ems(f, e, &tmp_m, &tmp_s);
  if(!tmp_s)
  {
```

```
    *m = tmp_m;
  }
  else
  {
    *m = -tmp_m;
  }
}
```

Упаковка з трьох змінних (знак, істинний порядок в ДК, мантиса в ПК).

```
void my_float_pack_from_ems(
    my_float *f, /* покажчик на число-результат */
    int8_t    e, /* істинний порядок в ДК */
    uint32_t  m, /* мантиса в ПК */
    uint8_t   s) /* знак */
{
  uint8_t tmp_e = e + 127; /* Отримання зміщеного порядку */

  f->bytes[3] = m & 0xff;
  m >>= 8;
  f->bytes[2] = m & 0xff;
  m >>= 8;
  f->bytes[1] = (tmp_e << 7) | (m & 0x7f);
  if(s)
  {
    s = 0x80;
  }
  f->bytes[0] = s | (tmp_e >> 1);
}
```

Упаковка з двох змінних (істинний порядок в ДК, мантиса в ДК).

```
void my_float_pack_from_em(
    my_float *f, /* покажчик на число-результат */
    int8_t    e, /* істинний порядок в ДК */
    int32_t   m) /* мантиса в ДК */
{
  if(m >= 0)
  {
```

```
    my_float_pack_from_ems(f, e, m, 0);
  }
  else
  {
    my_float_pack_from_ems(f, e, -m, 0x80);
  }
}
```

2.4.7 Рекомендації для написання програми на мові Сі

При виконанні завдання на мові Сі рекомендовано дотримуватись наступного шаблону (функції розпаковки та упаковки дозволяється скопіювати з тексту вище):

```
/* Обрати необхідний тип в залежності від варіанту. */
struct ieee_binary32
{
  uint8_t bytes[4];
};

struct ieee_binary64
{
  uint8_t bytes[8];
};

typedef struct ieee_binary32 my_float;
typedef struct ieee_binary64 my_double;

/*
 * Допоміжні функції.  Наприклад, розпаковка та упаковка.
 */
...

/*
 * Реалізація математичної операції, заданої по варіанту.
 * Обрати тип аргументів в залежності від варіанту.
 */
void lab_operation(my_float *x, my_float *y, my_float *z)
{
  ...
```

```
}

void main(void)
{
  my_float x, y, z;

  /* завантаження операнду x */
  x.bytes[0] = 0x11;
  ...

  /* завантаження операнду y */
  y.bytes[0] = 0x11;
  ...

  /* виконання математичної операції */
  lab_operation(&x, &y, &z);

  /* виведення результату z на індикатор */
  ...
}
```

2.4.8 Виконання операцій додавання та віднімання

Нехай необхідно додати два числа з плаваючою комою: $x = m_x 2^{e_x}$ та $y = m_y 2^{e_y}$. Так як їх суму z також необхідно подати у вигляді числа з плаваючою комою, то:

$$z = m_z 2^{e_z} = m_x 2^{e_x} + m_y 2^{e_y}.$$

Таким чином, необхідно знайти m_z та e_z. Це можна зробити тривіально у випадку коли порядки операндів рівні: $e_x = e_y$. Але у загальному випадку це не так. Тому виконують так зване *вирівнювання порядків*. Ця процедура полягає в тому, що обирають деякий спільний порядок для двох аргументів та, змінюючи мантису, приводять обидва аргументи до цього порядку. Звичайно спільним обирають більший порядок. Нехай $e_x > e_y$ (звичайно, в програмній реалізації необхідно по аналогії розглянути і випадок $e_x < e_y$). Необхідно збільшити порядок числа y на $e_\Delta = e_x - e_y$;

це можна зробити, зменшивши його мантису в 2^{e_Δ} разів. Маємо:

$$\begin{aligned} z &= m_x 2^{e_x} + m_y 2^{e_y} = \\ &= m_x 2^{e_x} + m_y 2^{e_\Delta} 2^{e_x} = \\ &= (m_x + m_y 2^{e_\Delta})\, 2^{e_x} \\ &= m_z 2^{e_z}, \end{aligned}$$

звідки $m_z = m_x + m_y 2^{e_\Delta}$, $e_z = e_x$.

Або, іншими словами, для вирівнювання порядків мантису числа з меншим порядком зсувають вправо на кількість розрядів, рівну різниці порядків e_Δ. Під час виконання правого зсуву e_Δ молодших двійкових цифр мантиси вийдуть за розрядну сітку. Це може призвести до втрати точності.

При додаванні мантис може виникнути переповнення на один розряд. В цьому випадку переповнення усувають виконанням правого зсуву та збільшенням порядку на одиницю. Але може трапитись так, що порядок неможливо збільшити, так як він вже був рівний максимальному. В такому випадку результат вважають рівним $+\infty$ або $-\infty$ в залежності від знаку операнду з більшим за абсолютною величиною порядком.

Якщо після додавання мантис не виникло переповнення, результат z може виявитись денормалізованим. Необхідно виконати його нормалізацію за допомогою лівого зсуву та зменшення порядку на кількість виконаних зсувів.

Можливо, що мантиса результату рівна нулю. В цьому випадку результат звичайно теж рівний нулю. Для того, щоб сформувати коректне число з плаваючою комою, що рівне нулю, необхідно окрім нульової мантиси записати ще і нульовий порядок.

Отже, для додавання чисел з плаваючою комою необхідно виконати наступні кроки:

1. розпаковка операндів (отримати істинний порядок та мантису в ДК);
2. вирівнювання порядків;
3. додавання мантис;
4. перевірка мантиси результату на 0 та формування коректного нуля;
5. нормалізація або усунення переповнення;

6. упаковка результату.

Зауважимо, що перевірка на 0 має обов'язково виконуватись до нормалізації, так як число 0 неможливо нормалізувати.

Операція віднімання чисел з плаваючою комою виконується аналогічно.

Блок-схема алгоритму додавання представлена на рис. 2.7. Умова 2 визначає, який порядок більший, а значить буде обрано за спільний. На кроках 3, 4 та 5, 6 виконується вирівнювання порядків. На кроці 7 виконується додавання мантис за запис порядку результату. Умова 8 за допомогою двох знакових розрядів перевіряє, чи наявне переповнення. Умова 9 перевіряє, чи можна позбутись переповнення правим зсувом. Якщо переповнення усунути не можна, виконується крок 10 (або в даній лабораторній роботі дозволяється завершити роботу програми з помилкою). Кроки 12, 13 відповідають за формування коректного нуля. На кроках 14, 15 виконується усунення денормалізації вправо.

Основною особливістю реалізації операцій над числами з плаваючою комою є використання змінних цілого типу для зберігання мантис (які є дробовими, але з фіксованою комою). Мантиси чисел *binary32* мають 24 розряди: 1 розряд цілої частини та 23 розряди дробової. При використанні 32-розрядного типу `int32_t` мантиса буде зберігатись у молодших розрядах. Кома зафіксована між 23-м та 22-м розрядами. Розряди 24 та 25 використовуються як знакові. Використовується два розряди для можливості визначення переповнення.

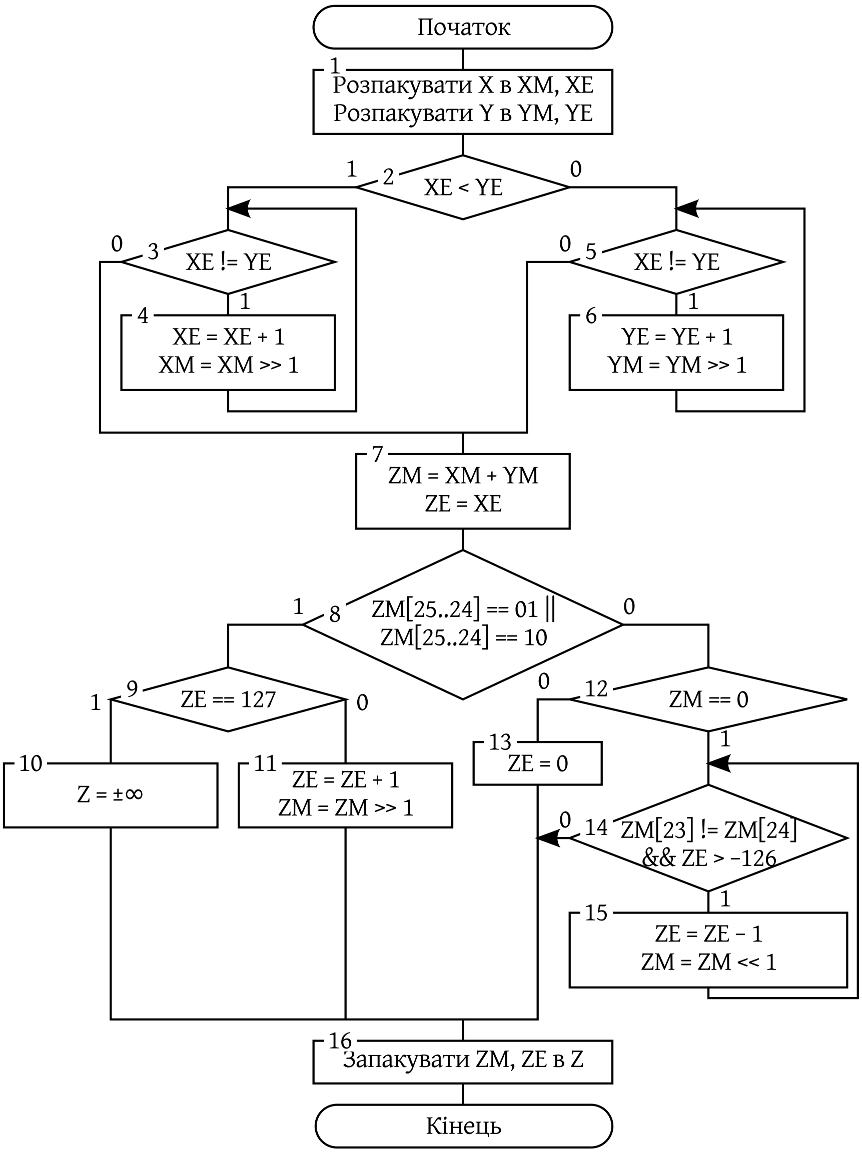

Рис. 2.7. Алгоритм додавання двох чисел з плаваючою комою в форматі *binary32*