

יצוג מספרים ממשיים

יצוג מספרים ממשיים - תאוריה

היצוג של מספרים ממשיים במחשב מזכיר את מה שקרוי כתיבה "מדעית" של מספרים:

כשימושים פיזיקליים ומדעיים אחרים אנחנו נוטים לכתוב מספר בצורה כמו

$$-5.71325 * 10^{13}$$

שמשמעותו למעשה

$$(-1) * (5 * 10^0 + 7 * 10^{-1} + 1 * 10^{-2} + 3 * 10^{-3} + 2 * 10^{-4}) * 10^{13}$$

כלומר הוא מורכב למעשה משלושה חלקים: סימן, ערך המיוצג בצורה מנורמלת (בין 1.0 ל- 9.9999...) וסדר גודל (חזקה של 10).

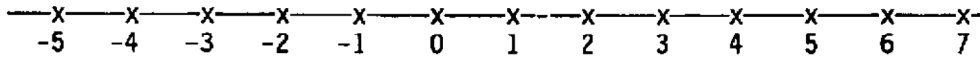
בחומרת מחשב המספרים הממשיים מיוצגים באופן עקרוני באותה צורה, אבל בכינארי במקום כדצימל. היצוגים של מספרים בחומרת המחשב (שלמים וממשיים) הם כגדלים קבועים מראש. במידה ורוצים ליצג מספרים בגודל רצוי או שרירותי יש לעשות זאת בתוכנה.

לפיכך "מספר ממשי" במחשב הוא למעשה שבר עם מספר סופי של ספרות משמעותיות + הכפלה בגורם הקובע סדר גודל + סימן.

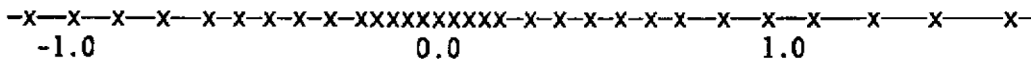
כאן אנחנו כבר יכולים לעמוד על הבדל בין היצוג של מספרים שלמים לבין היצוג של מספרים ממשיים:

בעוד מספרים שלמים המיוצגים במחשב נמצאים במרווחים שווים על הציר הממשי בין מקסימום למינימום (+32767 -- -32768 במקרה של מספרים 16 ביט עם סימן), זה אינו נכון עבור מספרים ממשיים. קבוצת המספרים הממשיים המיוצגים בחומרת המחשב נעשים צפופים יותר ככל שמתקרבים ל-0.0. בערך מחצית מכל המספרים נמצאים בין +1.0 ל- -1.0. ככל שמרחקים מ-0.0 המרווחים ביניהם הולכים וגדלים.

שלמים



ממשיים



יצוג מספרים בפועל

מספר ממשי מיוצג בחומרת המחשב בצורה:

א. יצוג מיוחד לאפס: כל הביטים אפס (0000000...0008).

ב. כל מספר שונה מאפס בצורה:

SIGN	EXPONENT	SIGNIFICAND
------	----------	-------------

כאשר

SIGN - ביט סימן (0 חיובי, 1 שלילי)

EXPONENT - מספר שלם שהוא חזקה של 2

SIGNIFICAND - סידרה של ספרות בינאריות המיצג מספר ממשי מנורמל (בין 1.0 ל- 1.9999...99). לפעמים עם ה-1.0 המוביל ולפעמים לא.

הערך של המספר הוא

$$(-1)^{\text{SIGN}} * (1.0 + \text{SIGNIFICAND}[k] * 2^{-k}) * 2^{\text{EXPONENT}}$$

דוגמא:

אח המספר 1.625 ניתן ליצג:

$$1.625 = 1.0 + 1 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3} = 1.0 + 0.5 + 0 + 0.125$$

לפיכך

SIGN = 0

EXPONENT = 0

SIGNIFICAND (1.0 ללא יצוג ל-) = 1010000...0

היצוג הזה הוא היצוג התאורטי.

כפועל ערייך חסרים מספר פרטים: כמה ביטים מוקצה ל-EXONENT ול-
SIGNIFICAND, האם ה-1.0 מיוצג ואיך מיוצג ה-EXONENT.

2. יצוג מספרים ממשיים ב-PC.

ב-PC המספרים הממשיים מיוצגים לפי הסטנדרד של ה-IEEE משנת 80:

א. גודל המספרים הממשיים הינם 32 ביט, 64 ביט, ו-80 ביט.
יצוג 32 ביט הוא ה-float של C, ובאופן כללי יותר נקרא דיוק רגיל או
דיוק יחיד Single Percision.
יצוג 64 ביט הוא ה-double של C, ובאופן כללי יותר נקרא דיוק כפול
double persion.
יצוג 80 ביט הוא ה-long double או באופן כללי יותר נקרא דיוק
מורחב Extended Percision.

ב. חלוקת הביטים לכל האורך הוא כלהלן:

32 ביט: 1 ביט ל-SIGN, 8 ביט ל-EXONENT, 23 ביט ל-SIGNIFICAND.

64 ביט: 1 ביט ל-SIGN, 11 ביט ל-EXONENT, 52 ביט ל-
SIGNIFICAND.

80 ביט: 1 ביט ל-SIGN, 15 ביט ל-EXONENT, 64 ביט ל-
SIGNIFICAND.

מנקודת ראות של דיוק בספרות עשרוניות, משתנים בדיוק רגיל (32
ביט) מישמים דיוק של 7 - 6 ספרות עשרוניות, דיוק כפול מישמים דיוק של
16 - 15 ספרות עשרוניות, דיוק מורחב (80 ביט) מישמים דיוק של 19 - 18
ספרות עשרוניות. איך מחשבים את הדיוק הזה נראה בהמשך.

ג. ה-1.0 אינו מיוצג ב-32 ביט וה-64 ביט, אבל מיוצג ב-80 ביט.

ד. היצוג של ה-EXONENT ע"י מספר שלם חיובי עם הטיה (BIAS).

ערך ה-BIAS תלוי באורך המספר:

127 עבור 32 ביט.

1023 עבור 64 ביט.

16383 עבור 80 ביט.

אגב, הערכים הקיצוניים של ה-EXONENT אינם בשימוש על מנת לאפשר

יצוג של אינסוף (infinity), לא מספר (Not a Number - NaN), ושדה ללא ערך
(Empty).

ה-EXPONENT עבור 32 ביט הוא בין 127 ל-126 וּלֹא בֵּין 128 ל-127 -
כפי שאפשר היה לעשות. כמו כן ה-EXPONENT של 64 ביט הוא בין 1023 ל-
-1022 (ולא בין 1024 ל-1023), ה-EXPONENT של 80 ביט הוא בין 16383 ל-
-16382 (ולא בין 16384 ל-16383).

לפיכך שלושת האפשרויות של יצוג מספרים ממשיים הם:

Single Percision (float)

SIGN(1b)	EXPONENT (8b)	SIGNIFICAND (23b)
31	30 23	22 0

Double Percision (double)

SIGN(1b)	EXPONENT (11b)	SIGNIFICAND (52b)
63	62 52	51 0

Extended Percision (long double)

SIGN(1b)	EXPONENT (15b)	1	SIGNIFICAND (63b)
79	78 64	63	62 0

תחומים:

טווח הערכים החיוביים (ערך מוחלט) שניתן ליצג ביצוג המספרים במחשב הינו:

32 ביט:

החזקות של 2 הם בין -126 ל- +127.
לפיכך הערכים הם בתחום:

$$1.0 * 2^{-126} = 1.175 * 10^{-38} \quad \text{בין}$$

$$1.9999... * 2^{127} = 3.4 * 10^{38} \quad \text{לכין}$$

בצורה דומה עבור 64 ביט החזקות של 2 הם בין -1022 לבין +1023 והערכים הם:

$$1.0 * 2^{-1022} = 2.225 * 10^{-308} \quad \text{בין}$$

$$1.9999... * 2^{1023} = 1.798 * 10^{308} \quad \text{לכין}$$

ובצורה דומה עבור 80 ביט החזקות של 2 הם בין -16382 לבין +16383 והערכים הם:

$$1.0 * 2^{-16382} = 3.362 * 10^{-4932} \quad \text{בין}$$

$$1.9999... * 2^{16383} = 1.189 * 10^{4932} \quad \text{לכין}$$

חישוב דיוק

כאופן כללי, אם ביצוג מספר ממשי יש כ-SIGNIFICAND M-ביטים משמעותיים, אזי הדיוק של ה-SIGNIFICAND הוא פשוט 2^{-M} .

בכדי להבין מדוע, אולי קל יותר לחשוב במושגים עשרוניים. לכל מספר ממשי בישר הממשי R יש יצוג עשרוני מנורמל (שעשוי להיות אינסופי). אם נניח שאנחנו קוטעים את היצוג של ה-SIGNIFICAND של מספר אחרי 3 ספרות אחרי הנקודה העשרונית, אוכדן הדיוק של ה-SIGNIFICAND חסום ע"י $0.0009999...0$ או בעצם כמעט 0.001 או 10^{-3} .

המקרה הבינארי כאופן עקרוני אותו דבר. ההבדלים הם ש-2 מחליף את 10 בחזקה, וכל הביטים מיצגים דיוק שמאחורי הנקודה הכינארית (המקדם הוא תמיד 1.0 ולא מיוצג במפורש).

לפיכך הדיוק של ה-SIGNIFICAND בדיוק רגיל (23 מתוך 32) הוא $2^{-23} = 1.19 \cdot 10^{-7}$. משמעות הדבר שעם אנחנו מבצעים פעולה אחת על שני מספרים המיוצגים באופן מדויק (נניח חילוק של שני שלמים), אפשר יהיה לסמוך על 6 הספרות העשרוניות הראשונות של התוצאה, אבל לא על השביעי.

בצורה דומה הדיוק של ה-SIGNIFICAND בדיוק כפול הוא $2^{-52} = 2.22 \cdot 10^{-16}$.

הדיוק של ה-SIGNIFICAND בדיוק מורחב הוא $2^{-63} = 1.08 \cdot 10^{-19}$.

דוגמאות למספרים ממשיים ויצוגם:

הערך 1.625 מיוצג כ-

32 ביט: 3FD00000h או

SIGNIFICAND = 101b

0011 1111 1101 0000 0000 0000 0000 0000b

127 = EXPONENT

64 ביט: 3FFA000000000000h או

SIGNIFICAND = 101b

0011 1111 1111 1010 0000 0000 0000 0000 0000 0000 0000 0000 0000b

1023 = EXPONENT

80 ביט: 3FFD0000000000000000h או

SIGNIFICAND = 1101b

0011 1111 1111 1111 1101 0000 0000 0000 0000

16383 = EXPONENT

דוגמא:

הערך - $-1.25 = -(1.0 + 0*0.5 + 1*0.25)$ מיוצג כ-

SIGNIFICAND = 01b ללא יצוג ה-1.0

היצוג 32 ביט יהיה BFA00000h

SIGNIFICAND = 01b

שלי

1011 1111 1010 0000 0000 0000 0000 0000b

127 = EXPONENT

64 ביט: BFF4000000000000h או

שלילי
SIGNIFICAND = 01b

1011 1111 1111 0100 0000 0000 0000 0000 0000 0000 0000 0000 0000h

1023 = EXPONENT

80 ביט: BFFFA000000000000000h או
שלילי
SIGNIFICAND = 101b

1011 1111 1111 1111 1010 0000 0000 0000 0000

16383 = EXPONENT

רוגמא:

יצוג הערך 0.625:

$$.0.625 = 1.25 * 2^{-1} = (1.0 + 0*0.5 + 1*0.25) * 2^{-1}$$

לפיכך ה-EXPONENT (ללא ה-BIAS) הוא -1. עם ה-BIAS ה-EXPONENT יהיה:
 , עבור 32 ביט, $-1 + 127 = 126$
 , עבור 64 ביט, $-1 + 1023 = 1022$
 , עבור 80 ביט, $-1 + 16383 = 16382$

ה-SIGNIFICAND התאורטי (ללא יצוג 1.0) שווה (כמו כדוגמא הקודמת) ל-01b.

היצוג 32 ביט יהיה 3F200000h

SIGNIFICAND = 01b

0011 1111 0010 0000 0000 0000 0000 0000b

126 = EXPONENT

64 ביט: 3FE4000000000000h או

SIGNIFICAND = 01b

0011 1111 1110 0100 0000 0000 0000 0000 0000 0000 0000 0000h

1022 = EXPONENT

80 ביט: 3FFE400000000000000h או

SIGNIFICAND = 101b

0011 1111 1111 1110 1010 0000 0000 0000 0000

16382 = EXPONENT

דוגמא:

יצוג הערך 576.0:

$$576.0 = 1.125 * 512 = 1.125 * 2^9$$

$$.1.125 = 1.0 + 0*0.5 + 0*0.25 + 1*0.125$$

לפיכך ה-EXPONENT (ללא ה-BIAS) הוא 9. עם ה-BIAS ה-EXPONENT יהיה:
 $9 + 127 = 136$ עבור 32 ביט,
 $9 + 1023 = 1032$ עבור 64 ביט,
 $9 + 16383 = 16392$ עבור 80 ביט.

ה-SIGNIFICAND התאורטי (ללא יצוג 1.0) הוא 001b.

היצוג 32 ביט יהיה 44100000h

$$\text{SIGNIFICAND} = 001b$$

0100 0100 0001 0000 0000 0000 0000 0000b

136 = EXPONENT

64 ביט: 4082000000000000h או

$$\text{SIGNIFICAND} = 001b$$

0100 0000 1000 0010 0000 0000 0000 0000 0000 0000 0000 0000b

1032 = EXPONENT

80 ביט: 40089000000000000000h או

$$\text{SIGNIFICAND} = 1001b$$

0100 0000 0000 1000 1001 0000 0000 0000 0000

16392 = EXPONENT

דוגמא:

דוגמא ליצוג בינארי לא סופי

נבדוק את היצוג של המספר התמים 1.4:

$$0.4 < 0.5 \text{ לפיכך}$$

$$.1.4 > 1.0 + 0*0.5 + 1*0.25 + 1*0.125 = 1.375$$

$$.2^{-4} = 0.0625 \text{ זה יותר מדי. } 1.4375 = 1.375 + 0.0625$$

$$.2^{-5} = 0.03125 \text{ זה עדיין יותר מדי. } 1.40625 = 1.375 + 0.03125$$

$$0.015625 = 2^{-6} = 0.015625 + 1.375 = 1.390625$$

קרוי יותר אבל
 עדיין לא מספיק. אבל התקרבו ל-1.4 עד כדי $0.009375 = 1.4 - 1.390625$.

אפשר להגיד שה-SIGNIFICAND 011001b המקביל ל-

$$1.0 + 0*0.5 + 1*0.25 + 1*0.125 + 0*0.0625 + 0*0.03125 + 1*0.015625 = 1.390625$$

מיצג את 1.4 בדיוק של 6 ספרות בינאריות.

אפשר להמשיך כך ... לעולמים. למרות של-1.4 יש יצוג עשרוני סופי, אין לו יצוג בינארי סופי! לא לכל מספר שיש לו יצוג עשרוני סופי, יש יצוג בינארי סופי. למעשה רוב המספרים שיש להם יצוג עשרוני סופי אינם כאלו!

זה למעשה אינו חדש. גם ביצוג העשרוני אין דיוק סופי למספרים כמו $2/7$ או $1/3$. שלא לדבר על מספרים כמו e או π . תמיד פעולות על מספרים ממשיים הם עד דיוק כזה או אחר.

לסיכום הנושא הזה היצוג של 1.4 הוא:

32 ביט: 3FB33333h או

SIGNIFICAND = 011001....b

 0011 1111 1011 0011 0011 0011 0011 0011b

 127 = EXPONENT

64 ביט: 3FF6666666666666h או

SIGNIFICAND = 011001....b

 0011 1111 1111 0110 0110 0110 0110 0110 0110 0110 0110 0110 0110h

 1023 = EXPONENT

80 ביט: 3FFF8333333333333333h או

SIGNIFICAND = 1011001....b

 0011 1111 1111 1111 1011 0011 0011 0011 0011

 16383 = EXPONENT

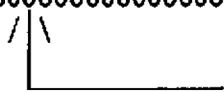
יצירת דוגמאות נוספות:

אם ברצונך לדעת איך מספר ממשי כלשהוא מיוצג (מבלי לחשב זאת בעצמך) דרך פשוטה לעשות זאת הוא להגדיר בתוכנית אסמבלר כלשהוא (נניח-myprog.asm):

```
A1 DD 1.625
AA1 DQ 1.625
AAA1 DT 1.625
```

ובצע tasm /la myprog.asm של הקובץ.
יווצר לכן קובץ myprog.lst, שבתוכו תראה (דרך תוכנית עריכה):

```
6 0000                                .DATA
7 0000 3FD00000                        A1 DD 1.625
8 0004 3FFA000000000000                AA1 DQ 1.625
9 000C 3FFF0000000000000000            AAA1 DT 1.625
```



היצוג הכינארי (בהקסה) נמצא כאן