**ייצוג מספרים ממשיים**

ייצוג מספרים במחשב הוא עניין מורכב יותר ממה שאולי היינו מצפים. כדאי לחלק את הדיון בו למספר היבטים.

**1. סוגי מספרים המיושמים במחשב**

נקודת השקפה אחת הוא **סוגי** המספרים הנתמכים בטכנולוגיות החישוב הממוכן הקיימים היום, ממחשבי כיס עד למחשבים אישיים, שרתים ומחשבי על.

אם נסקור את כל סוגי הטכנולוגיה אנחנו נגלה באופן עקרוני שלושה סוגי מספרים:

* מספרים שלמים (integers)
* מספרים ממשיים הממומשים בשיטת הנקודה הקבועה (fixed point)
* מספרים ממשיים הממומשים בשיטת הנקודה הצפה (floating point)

מספרים שלמים (0, 1, 1-,2, 2- ...) הם סוג של מספרים המיושם בשפות התכנות העוברים קומפילציה (C, פסקל, ADA, Fortran, ...) וגם בפלטפורמות תכנות אחרות.

בשפת C מספרים כאלו מיושמים במשתנים מסוג int, unsigned int, long, , unsigned long ,int וכו'.

הייצוג של המספרים הללו הינו **מדויק**, והם מספרים במרווחים שווים זה מזה. במידה והם יכולים להיות שליליים המספרים הללו הם בין מספר שלם מירבי למספר שלילי ביותר שערכו המוחלט גדול בדרך כלל באחד מהמספר המירבי. למשל עבור מספרים שלמים 16 ביט הטווח הוא בין 32767 ל-32768-. בגרסתם חסרת הסימן הטווח הוא בין אפס למספר מירבי למשל ב-16 ביט זה יהיה בין 0 ל-65535. בהקשר של אנליזה נומרית המספרים הללו בעלי חשיבות מינימלית.

מספרים ממשיים בעלי נקודה קבועה (fixed point) אינם נתמכים בכל פלטפורמה. אולי אפילו לא ברוב הפלטפורמות. מספרים הללו מיושמים בדרך כלל בפלטפורמות שנועדו בעיקר או גם לעיבוד נתונים כמו שפות COBOL ו-PLI. הם מיושמים גם במכונות חישוב ולפחות בחלק ממחשבי הכיס. אלה מספרים בעלי מספר קבוע של ספרות (בדרך כלל עשרוניות) בתוספת ציון היכן מסתיים החלק השלם. בדרך כלל מספר הספרות לפני ואחרי הנקודה קבועים. למשל אם מדובר במספרים בדיוק של חמש ספרות עשרוניות לפני הנקודה ושניים אחריה, זה יכול להיות מספרים כמו

,5643.21

45673.21,

432.1,

39876.54

וכו'. לשיטה הזו יש יתרון בכך שאם מספר מתקבל בצורה של **שבר** במסגרת הדיוק, הייצוג שלו יהיה **מדויק** וזה נכון גם עבור חישובים מסוימים כמו חיבור וחיסור. התכונה הזו של דיוק איננה נכונה עבור ייצוג מספרים בשיטת הנקודה הצפה (שתתואר בהמשך). שיטת הנקודה הקבועה זו היא אולי השיטה הטובה ביותר עבור חישובים פיננסיים אבל בדרך כלל לא בשימוש באנליזה נומרית, למרות שבתאים מסוימים אפשר, אבל המגבלות שהשיטה הזו מטילה מחמירים מדי, וחוץ מזה החישובים בשיטה הזו עשויים להיות די איטיים, בגלל שהשיטה הזו בדרך לא נתמכת בחומרה (נושא שנדון בו בהמשך).

השיטה האחרונה והעיקרית בכל הקשור לאנליזה נומרית היא שיטת הייצוג של מספרים ממשיים היא שיטת הנקודה הצפה floating point. בתיכון קראנו לייצוג הזה "ייצוג מדעי" של מספרים ממשיים. הוא ממומש בצורה של מספר **קבוע** של **ספרות משמעותיות**  המייצגות את **הדיוק** של המספר והוא מוכפל בגורם בדוגל **קבוע** המייצג את **סדר הגודל** שלו, בנוסף לביט נוסף הממש מציין **סימן** (חיובי או שלילי). בגרסתה העשרונית (שהיא לאו דווקא הצורה שבה היא מיושמת במחשב) דוגמאות למספרים בייצוג כזה הינם

4.5542 x 1026

2.32527 x 10-19

-7.8912 x 102

-9.8 x 10-76

מספר המיוצג בשיטה הזו יש שלושה מרכיבים:

- מציין סימן (חיובי או שלילי)

- מספר **ממשי** **מנורמל** (בעשרוני בין 1.0 ל-....9.9999) בדיוק של עד מספר **קבוע** של **ספרות משמעותיות** הנקרא significant או mantissa המייצג את הדיוק של הנתון.

- מספר **שלם**, חיובי או שלילי, בגודל של עד מספר **קבוע** של ספרות, המייצג את **החזקה** של מקדם סדר הגודל קבוע מראש (10 בייצוג עשרוני) הנקרא exponent.

הדוגמאות שלעיל כולי להיות, למשל, ייצוג מספרים של עד חמש ספרות עשרונית ב-significant, ועד שני ספרות ב-exponent.

על הייצוג של המספרים הללו בחומרה במחשבים כלליים (כמו מחשבים אישיים, תחנות עבודה, שרתים) נרחיב יותר מאוחר אבל בשלב זה נציין שבדרך כלל הייצוג שם הוא לא עשרוני, כלומר לא מבוסס על חזקות חיוביות או שליליות של 10 אלא **בינארי** כלומר מבוסס על חזקות חיוביות או שליליות של 2.

יחד עם זאת אפשר כבר עכשיו לציין את התכונות העיקריות של מספרים המיוצגים בשיטה הזו:

המספרים המיוצגים בשיטה הזו הם מספר **סופי וידוע מראש** של מספרים מיוצגים באופן **סימטרי** משני צדי האפס. המספר החיובי ביותר והמספר השלילי ביותר **שווים** זה לזה **בערכם המוחלט**. המרחקים (או ההפרשים) בין המספרים המיוצגים **אינם שווים**.הצפיפות של המספרים המיוצגים היא מירבית סמוך לאפס. בסביבות מחצית המספרים המיוצגים הם בין 1.0 ל-1.0-. אך ככל שמתרחקים מהראשית, ביחוד ברגע שעוברים ימינה מ-1.0 או שמאלה מ-1.0- הפער או המרחק בין המספרים המיוצגים הולך וגדל. כאשר אנחנו צריכים לייצג מספר שאיננו מיוצג במפורש עלינו לבחור בין שני המספרים הקרובים אליו ביותר, הגדול הראשון או הקטן הראשון ממנו.

היחס בין המספרים המיוצגים והלא מיוצגים על ציר המספרים נראה ציורית כך:

\_x\_\_\_\_\_x\_\_\_x\_\_\_x\_\_x\_x\_xxxxx\_x\_x \_\_x\_\_\_\_x\_\_\_\_x\_\_\_\_\_x\_\_\_\_\_\_

0

המספרים הללו לא תמיד מקימים אפילו את אקסיומות המספרים המובנים מאליו ביותר. למשל סכום וחיסור של מספרים כאלו הם לא תמיד אסוציאטיביים או קומוטטיביים כאשר מדובר במספרים שונים מאד בסדרי גודל. לדוגמא התסריטים הבאים יכולים בהחלט להתרחש:

חוסר קומוטטיביות:

2.13 x1023+9.6x10-11-2.13 x1023 = 0

2.13 x1023-2.13 x1023+9.6x10-11 = 9.6x10-11

חוסר אסוציטיביות:

(2.13 x1023-2.13 x1023)+9.6x10-11 = 9.6x10-11

2.13 x1023(-2.13 x1023+9.6x10-11 ) = 0

כמובן, כאשר אנחנו עם מספרים באותו סדר גודל, אין את הבעיות הללו, וזה מה שקורה רוב הזמן. אבל ישנם מצבים שהדבר הזה קורה מסיבות ענייניות, למשל בזמן חישוב e-x על ידי טור טיילור. בחישוב כזה, למשל עבור x=10 התוצאה היא מספר קטן הנובע מחישוב של מספרים יחסית גדולים המבטלים זה את זה.

תכונה חשובה שלא מוערכת מספיק היא העובדה שמספרים שלמים מיוצגים גם בשיטה הזו בצורה **מדויקת**. כלומר שמספר כמו 23.0 הייצוג הוא מדויק ולא 23.00001 או 22.99999. גם סכום וכפל של מספרים שאין להם חלק שבור יהיה מדויק, בתנאי שהתוצאה לא תהיה גדולה מדי. כלומר 23.0\*117.0 = 2691.0 בדיוק ולא קירוב שלו.

**2. תמיכה באריתמטיקה ממשית בחומרה**

היבטים נוספים שעלינו להביא בחשבון הינם האם ייצוגי המספרים נתמכים ב**רמת החומרה** או **ברמת התוכנה**, ואיך הם נתמכים שם.

בכל המחשבים הכלליים היום ישנה תמיכה ברמת החומרה לביצוע חישובים במספרים ממשיים. בדרך כלל מדובר בתמיכה במספרים ממשיים מסוג floating point בבסיס בינארי. זאת בנוסף לתמיכה באריתמטיקה של מספרים שלמים. תמיכה ברמת החומרה היא שיש **פקודות מכונה הממומשות במפורש ב-CPU** המקבלות נתונים בפורמט ממשי ועושה עליהם פעולה אריתמטית נוסך חיבור, חיסור, כפל, חילוק, חישוב לוגריתם, העלאה בחזקה ... ועוד. החישובים הללו יהיו תמיד בדיוק קבוע או במספר דיוקים קבועים. למשל במחשב האישי המספרים הממשיים הם floating point על בסיס בינארי בדיוקים הם ב-32, 64 ו-80 ביט. שימוש בתמיכה הזו נותנת את זמני החישוב המהירים ביותר אבל מחיבות השלמה עם הדיוקים הנתמכים.

ברוב פלטפורמות התכנות התמיכה במספרים ממשיים מתבססת על התמיכה בחומרה. בשפות תכנות לקומפילציה כמו ++ C,C, פסקל וכו' ניתן לבחור עבור משתנה באיזה מהדיוקים הנתמכים לממש אותו. בשפות תכנות מבוססות על תוכנת פירוש כמו python, javascript שבו לא מכריזים על משתנים בדרך כלל נבחר הדיוק המירבי, לפעמים תוכנת הפירוש תבחר את הדיוק המירבי רק כאשר היא חשה ש"צריך אותו".

משתנים ממשיים נוסך fixed point ממומשים בדרך כלל **ברמת התוכנה** ולא בחומרה. גם אריתמטיקה עשרונית מיושמת בדרך כלל ברמת התוכנה.

אם ניקח את המחשב האישי בתור דוגמא, כאמור המספרים הממשיים הנתמכים בחומרה הם floating point על בסיס **בינארי** בדיוקים הם ב-32, 64 ו-80 ביט.

מספר ממשי floating point על בסיס **בינארי** מורכב מביט סימן, מ-significant שהוא מספר ממשי מנורמל בין 1.0 ל-...1.999 בייצוג של סכום חזקות שליליות של 2 ומ-exponent שלם המייצג את החזקה של 2 שנותן את סדר הגודל שאותו יש להכפיל את ה-significant. אין צורך אפילו לייצג את ה-.1 של ה-significant.

לדוגמא, מספר ממשי עם 1101001 = significant ואם exponent בעשרוני הוא 5 הערך שהוא מייצג הינו:

המספר המנורמל יהיה

1.0 + 1\*2-1 + 1\*2-2 + 0\*2-3 + 1\*2-4 + 0\*2-5 + 0\*2-6 + 1\*2-7

1.0+ 0.5 + 0.25 + 0 + 0.0625+ 0 + 0 + 0.0078125

= 1.8203125

וערך המספר יהיה

1.8203125 \* 25 = 1.8203125 \* 32 = 58.25

בשיטה זו ניתן לייצג בצורה מדויקת רק מספרים שהחלק השבור שלהם הוא סכום חזקות שליליות של 2. כלומר אפשר לייצג במדויק שברים כמו 0.625 (0.5+0.125) או 0.1875 (0.125+00625) . אפילו מספר צנוע לחלוטין כמו ל-1.4 אין ייצוג בינארי סופי! אפשר להתקרב עד כדי הפרש קטן כרצונינו למשל

1.0 + 2-2 +2-3 + 2-6 =

1.0 + 0.25 + 0.125 +0.015625 = 1.390625

אבל גם אם נמשיך להוסיף חזקות שליליות אחרות של 2 אנחנו או נעבור את 1.4 או נשאר מתחת לו. לרוב השברים העשרוניים הסופיים **אין** ייצוג בינארי סופי. זה מסביר מדוע כאשר אנחנו מספקים לתוכניות C או שפות תכנות אחרות שברים סופיים בעשרוני אנחנו מקבלים תוצאות מקורבת גם כאשר כל שאנחנו עושים הם הדפסה או חיבור וחיסור.

למשל לתוכנית הבאה

int main()

{

float x, y, z;

printf("Enter two numbers:\n");

scanf("%f %f",&x, &y;(

z = x + y;

printf("%f + %f = %f\n", x, y, z);

} /\* main \*/

אנחנו עשויים לקבל פלט כמו

C:\> add.exe

Enter two numbers:

56.4 74.23

56.400002 + 74.230003 = 130.630005

C:\>

**דיוקים נתמכים בחומרה במחשב האישי**

המחשב האישי תומך במספרים ממשיים ב-3 רמות דיוק:

**דיוק רגיל Single Precision– 32 ביט, בשפת C משתנים מסוג float**: ביט אחד לסימן, 8 ביט ל-exponent ו-23 ביט ל-significant. ערך ה-exponent נע בין 126- ל-127+. דיוק המספרים הוא בערך 6-7 ספרות עשרוניות משמעותיות. תחום הערכים הוא בין

+/- 1.0 \* 2-126 ≈ 1.175 \* 10-38

לבין

+/- 1.999… \* 2127 ≈ 3.4 \* 1038

**דיוק כפול Double Precision – 64 ביט, בשפת C משתנים מסוג double**: ביט אחד לסימן, 11 ביט ל-exponent ו-52 ביט ל-significant. ערך ה-exponent נע בין 1022- ל-1023+. דיוק המספרים הוא בערך 15-16 ספרות עשרוניות משמעותיות. תחום הערכים הוא בין

+/- 1.0 \* 2-1022 ≈ 2.225 \* 10-308

לבין

+/- 1.999… \* 21023 ≈ 1.798 \* 10308

**דיוק מורחב Extended Precision – 80 ביט, בשפת C משתנים מסוג long double**: ביט אחד לסימן, 15 ביט ל-exponent ו-63 ביט ל-significant. ערך ה-exponent נע בין 16382- ל-16383+. דיוק המספרים הוא בערך 18-19 ספרות עשרוניות משמעותיות. תחום הערכים הוא בין

+/- 1.0 \* 2-16382 ≈ 3.362 \* 10-4932

לבין

+/- 1.999… \* 216383 ≈ 1.189 \* 104932

כפי שניתן להבחין, בשלושת רמות הדיוק אין ניצול מלא של הביטים המוקצים ל-exponent ובמקרה של הדיוק המורחב אין ניצול של אחד הביטים ב-significant. יש לכך מספר סיבות, בעיקר על מנת לייצג ערכים מיוחדים (אפס, אינסוף, לא מספר NaN – Not a Number, ...).

**דיוק חישובים בפלטפורמות תכנות – כללי**

ברוב הפלטפורמות של תכנות המספרים הממשיים נתמכים בדיוקים קבועים. הדיוקים הללו בדרך כלל יהיו אותם אלה שנתמכים בחומרה. בדרך כלל אפשר לבחור בין מספר קבוע של אפשריות, משום שחישובים בדיוקים גבוהים מאיטה במשהו את זמני הריצה, ולא תמיד הדבר נחוץ. במקרים אחרים הפלטפורמה מממשת את החישובים בדיוק המירבי הנתמך בחומרה.

בדרך כלל הדיוק המירבי הנתמך בחומרה מספיק לצרכים. פלטפורמות מתקדמות מסוימות מאפשרות למשתמש לבקש דיוק חישובים בגודל רצוי. במקרים מסוימים (למשל בחישובים של קבועים פיזיקליים, מזג אויר או מיקום) החישובים מחייבים דיוקים גבוהים יותר מהנתמכים בחומרה המתכנת עשוי למצוא את עצמו חייב להשתמש בתוכנות המאפשרים לבקש חישובים בדיוק מוזמן או לממש את ייצוג המספרים והחישובים המספריים בעצמו או בעזרת תוכנות ספריה מיוחדות לכך.

אם חישוב כלשהוא שמתכנת מממש נתקל בבעיות דיוק שאינן נובעות מצורך של ספרות משמעותיות רבות בתוצאה אלא בגלל הצטברות שגיאות בחישובים, הגדלת דיוק של ייצוג המספרים היא פתרון חלקי בלבד לבעיות מסוג זה. בדרך כלל הבעיה היא חוסר תחכום ברמת האלגוריתם והפתרון אמור להיות במציאת אלגוריתמים משופרים או חדשים לבעיה.