

# **Представяне на числа в двоичните универсални компютри**

Кирил Иванов

Април 2022 година

## **Предговор**

Този текст е **съсредоточен върху фундаменталните идеи** за представяне на числа в масово разпространените универсални компютри.

Тези идеи, точно защото са базови за съвременните универсални компютърни архитектури, **предопределят простите типове и начина за работа с тях** в масово използваните днес езици за програмиране от високо ниво и **влият на новосъздаваните езици**.

Всъщност всичко описано тук се отчита и понякога е отправна, начална аргументация при избора на базовите идеи и за много аспекти на специализираните архитектури. Обаче там вече се използват по-ефективни решения и се прилагат и качествено нови идеи, които са уместни именно заради конкретните особености на съответното целево предназначение. Такива особености остават извън обхвата на този текст.

**Основната цел** на текста е да покаже как именно **стремежът към изграждане на програмируеми устройства с максимална гъвкавост и пределна простота** поражда конкретните практически решения. Затова са проследени и различни причинноследствени зависимости.

Обхванатият материал е **предназначен** главно за читатели, изучаващи компютърни програмиране и архитектури на машинно и на асемблерно ниво и може да бъде съществено полезен при всяко съприкосновение с тези нива (например при трасиране на програми). Няколко от засегнатите въпроси представляват интерес и при изучаването на системата от примитивни типове и възможностите за изчисления в който и да било език за програмиране от високо ниво.

Този текст е **достъпен** за ученици, познаващи основните свойства на бройни системи, подобни на Арабската. Обаче в по-голямата си част обхванатият материал излиза извън рамките на актуалните в момента учебни програми. Само отделни програми за профилирано обучение включват съществена част от засегнатите тук въпроси.

## Съдържание

1. Означения и съкращения, използвани в този текст .....	3
2. Пояснение за начина на описание .....	4
3. Няколко особено често използвани свойства на записите в ПБС .....	4
4. Предимства на числовите компютри пред аналоговите .....	7
5. Необходимост от кодиране за представянето на числа в паметта .....	9
6. За масовото разпространение на двоичните компютри .....	9
7. За значението на сумирането на цели числа .....	10
8. Препълване и флагове .....	11
9. Код без знак .....	12
10. Прав код .....	14
11. Обратен код .....	17
12. Допълнителен код .....	22
13. Изместен код .....	27
14. Кодове с двоично кодиране на десетичните цифри .....	29
15. Умножение на цели числа и точност на представянето .....	30
16. Фундаментални принципи за кодиране на цели числа .....	30
17. Примери за получаване на кодове на цели числа .....	36
18. Примери за намиране на кодирано цяло число по известен код .....	45
19. Въведение в представянето на дробни числа .....	47
20. Формат с фиксирана запетая .....	48
21. Формат с естествена запетая .....	48
22. Формат на типа <i>Decimal</i> в езика <i>C#</i> и в платформата <i>.NET</i> .....	49
23. Видове кодове с плаваща запетая .....	51
24. Точност на представянето на числа при работа с КПЗ .....	54
25. Някои следствия от нормализацията на КПЗ .....	56
26. Примери за получаване на КПЗ .....	59
27. Примери за получаване на кодирано число от известен КПЗ .....	68
28. Разполагане на кодове в паметта .....	70
29. Следствия от машинните числови формати за езиците от високо ниво .....	71

## Препратки към задачи с номера:

1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12; 13; 14; 15; 16; 17.

## 1. Означения и съкращения, използвани в този текст

По-нататък ще използваме следните означения (те не са общоприети, но са изключително удобни):

ДК – допълнителен код

$ДК_n(V)$  – допълнителен код на числото  $V$  в  $n$  разряда

ИК – изместен код

$ИК_n(V)$  – изместен код на числото  $V$  в  $n$  разряда

КБЗ – код без знак

$КБЗ_n(V)$  – код без знак на числото  $V$  в  $n$  разряда

КДКДЦ – код с двоично кодиране на десетичните цифри

$КДКДЦ_n(V)$  – код с двоично кодиране на десетичните цифри на числото  $V$  в  $n$  разряда

КЕЗ – код с естествена запетая;

КПЗ – код с плаваща запетая;

$КПЗ_{k,n}(V)$  – код с плаваща запетая с  $k$  разряда за представяне на порядъка и  $n$  разряда за представяне на мантисата на числото  $V$ ;

КФЗ – код с фиксирана запетая;

$МДК_n(V)$  – модифициран допълнителен код на числото  $V$  в  $n$  разряда

$МОК_n(V)$  – модифициран обратен код на числото  $V$  в  $n$  разряда

НКДКДЦ – непакетиран код с двоично кодиране на десетичните цифри

НКПЗ – нормализиран код с плаваща запетая;

$НКПЗ_{k,n}(V)$  – нормализиран код с плаваща запетая с  $k$  разряда за представяне на порядъка и  $n$  разряда за представяне на мантисата на числото  $V$ ;

ОК – обратен код

$ОК_n(V)$  – обратен код на числото  $V$  в  $n$  разряда

ПБС – позиционна бройна система (подразбира се система от типа на Арабската)

ПК – прав код

$ПК_n(V)$  – прав код на числото  $V$  в  $n$  разряда

ПКДКДЦ – пакетиран код с двоично кодиране на десетичните цифри

## 2. Пояснение за начина на описание

Тъй като с кодовете *от един и същи вид*, каквато и да е тяхната разрядност, се работи по еднотипен, аналогичен начин, в повечето от приведените в този текст примери се използват разрядности без практическо приложение, например 5-разрядни, 9-разрядни или 11-разрядни кодове. Обаче в практиката се следват много строги правила за дължините на хардуерно поддържаните кодове. Например цели числа се кодират в брой разряди, който е степен на двойката (в точка 16 е пояснено защо), а по-абстрактно – в брой разряди, равен на степен на основата на ПБС, върху която се базира паметта.

Освен това, тук се разглеждат само кодове в двоична памет, защото тя е масово разпространената днес в универсалните компютри (в точка 6 са очертани основните причини за това). С памети и кодове, базирани върху ПБС с друга основа, се работи аналогично.

В теорията на машинните изчисления кодовете се разглеждат просто като число с определени характеристики и обикновено стойностите им се записват десетично или чрез формули. Точно така строго математически се доказват различните твърдения за изчислителните алгоритми и свойствата на представянията. Обаче в този текст кодовете ще записваме двоично и с точно толкова цифри, колкото е разрядността на кода. Това позволява максимално осезаемо описание на начините на кодиране и свойствата на кодовете.

Текстът е ориентиран към повишена нагледност. Затова аргументацията се опира повече на интуитивни разсъждения, отколкото на строги доказателства. По същата причина е използвано необичайно оцветяване и форматиране.

## 3. Няколко особено често използвани свойства на записите в ПБС

За разбирането на представянето на числа в цифровите компютри (наричани също и дискретни) са особено важни следните няколко елементарни свойства на записите в ПБС от типа на Арабската (пропускаме доказателствата, защото са очевидни):

### Свойство 1

За всяка основа  $s$  на ПБС е вярно  $1 \underbrace{0 \dots 0}_{n(s)} = s^n$ .

Например:

$$1000_{(10)} = 10^3;$$

$$1 \underbrace{0 \dots 0}_{n(10)} = 10_{(10)}^n;$$

$$10000_{(2)} = 2^4 = 16_{(10)};$$

$$1 \underbrace{0 \dots 0}_{n(2)} = 2^n;$$

$$100_{(16)} = 16_{(10)}^2 = 256_{(10)};$$

$$1 \underbrace{0 \dots 0}_{n(16)} = 16_{(10)}^n.$$

### **Свойство 2**

За всяка основа  $s$  на ПБС е вярно  $0, \underbrace{0 \dots 0}_{n-1} 1_{(s)} = s^{-n}$ .

Например:

$$0,001_{(10)} = 10^{-3};$$

$$0, \underbrace{0 \dots 0}_{n-1} 1_{(10)} = 10^{-n};$$

$$0,0001_{(2)} = 2^{-4};$$

$$0, \underbrace{0 \dots 0}_{n-1} 1_{(2)} = 2^{-n};$$

$$0,01_{(16)} = 16^{-2} = \frac{1}{256};$$

$$0, \underbrace{0 \dots 0}_{n-1} 1_{(16)} = 16^{-n} = \frac{1}{16^n}.$$

**Свойство 3**

Нека  $\xi$  е цифрата с най-голяма стойност в ПБС с основа  $S$ , т.е.  $\xi_{(S)} = S - 1$ .

Тогава  $\underbrace{\xi \dots \xi}_{n}_{(S)} = 1 \underbrace{0 \dots 0}_{n}_{(S)} - 1 = S^n - 1$  и това е най-голямото  $n$ -цифрено

$S$ -ично число.

Например:

$$999_{(10)} = 1000_{(10)} - 1 = 10^3 - 1;$$

$$\underbrace{9 \dots 9}_{n}_{(10)} = 1 \underbrace{0 \dots 0}_{n}_{(10)} - 1 = 10^n - 1;$$

$$1111_{(2)} = 10000_{(2)} - 1 = 2^4 - 1 = 16 - 1 = 15;$$

$$\underbrace{1 \dots 1}_{n}_{(2)} = 1 \underbrace{0 \dots 0}_{n}_{(2)} - 1 = 2^n - 1;$$

$$2222_{(3)} = 10000_{(3)} - 1 = 3^4 - 1 = 81 - 1 = 80.$$

**Свойство 4**

Във всяка  $S$ -ична ПБС и за всяко естествено число  $n$  е вярно:

$$0 \leq \overline{a_{n-1} \dots a_0}_{(S)} < S^n \text{ (каквито и да са цифрите } a_{n-1}, \dots, a_0 \text{)}.$$

**Свойство 5**

С  $n$  цифри в ПБС с основа  $S$  може да бъдат записани точно  $S^n$  различни цели неотрицателни цели числа.

Следователно различните възможни кодове с разрядност  $n$  от един и същ вид са точно  $2^n$  (видът определя начина на интерпретация на цифрите).

Обаче броят на кодираните числа може да бъде различен от  $2^n$ . Така, когато в представянето на цели числа някое от тях се кодира по повече от един начин (с повече от един код), тогава броят на кодовете е по-голям от броя на кодираните числа. Съответно с код за числа с дробни части може да се представят безброй числа.

### Свойство 6

Сравняването на числа, записани в една и съща ПБС с *равен брой цифри в цялата част*, каквито и да са дробните части, може да се извършва лексикографски (както се сравняват думите в обичайните речници).

Така:

$$b_k < c_k \Rightarrow a_{n-1} \dots a_{k+1} b_k a_{k-1} \dots a_0 < a_{n-1} \dots a_{k+1} c_k d_{k-1} \dots d_0$$

и съответно, когато  $b_k \neq c_k$ , тогава е вярно твърдението

$$a_{n-1} \dots a_{k+1} b_k a_{k-1} \dots a_0 < a_{n-1} \dots a_{k+1} c_k d_{k-1} \dots d_0 \Leftrightarrow b_k < c_k.$$

Аналогично:

$$b_k < c_k \Rightarrow 0, a_1 \dots a_{k-1} b_k a_{k+1} \dots < 0, a_1 \dots a_{k-1} c_k d_{k+1} \dots$$

и съответно, когато  $b_k \neq c_k$ , тогава е вярно твърдението

$$0, a_1 \dots a_{k-1} b_k a_{k+1} \dots < 0, a_1 \dots a_{k-1} c_k d_{k+1} \dots \Leftrightarrow b_k < c_k.$$

Например:

$$0,00\mathbf{1}9_{(10)} < 0,00\mathbf{2}89999_{(10)}, \text{ защото } \mathbf{1} < \mathbf{2};$$

$$164,000\mathbf{1}99999_{(10)} < 164,000\mathbf{2}12_{(10)}, \text{ защото } \mathbf{1} < \mathbf{2};$$

$$1011\mathbf{1}0_{(2)} > 1011\mathbf{0}1,11111_{(2)}, \text{ защото } \mathbf{1} > \mathbf{0};$$

$$110,011\mathbf{1}_{(2)} > 110,011\mathbf{0}11_{(2)}, \text{ защото } \mathbf{1} > \mathbf{0}.$$

Всъщност точно до такова лексикографско сравняване свеждаме всяко сравняване на числа, когато ги подравняваме по дробния разделител. Това важи за всяка ПБС.

## 4. Предимства на числовите компютри пред аналоговите

Характерната особеност на аналоговите компютри е, че числата се представят чрез някакви величини, които могат да имат безброй различни стойности. За тази цел може да послужи всяка величина, която е в състояние да се изменя в непрекъснат интервал – електростатично напрежение, сила на електрически ток, интензитет на магнитно поле, сила на светлината, концентрация на разтвор, потенциална или кинетична енергия, скорост, температура... – буквално всяка.

Съответно, за да се получи някаква търсена стойност от някакви дадени стойности, се използва материална конструкция, подходяща материална система, където началните стойности са представени с подходящи налични величини и след функционирането на конструкцията (системата) в продължение на краен период от време някъде в нея се получава величина с търсената стойност. При това, трябва да бъде възможно измерването на получената стойност. Именно такава конструкция се нарича **аналогов компютър**.

Съществено е също, че макар получаваната стойност да е с безкрайна точност, при нейното измерване неизбежно възниква закръгляне, което може да бъде и с много голяма грешка.

Обаче при такъв замисъл конструкцията (системата) трябва да има структура, която точно съответствува на зависимостта между дадените и търсената стойности. Следователно всеки един аналогов компютър е подходящ само за конкретен вид алгоритми, с точно определени зависимости между участващите величини. За да се изпълняват качествено различни алгоритми са необходими качествено различни конструкции. Това превръща програмирането чрез аналогови компютри в инженерна дейност, която освен всичко останало изисква и съответен строителен материал и други ресурси.

Също така, възникват проблеми с измерването на величините. Най-напред, всяка величина малко или повече е изменчива във времето и понякога е трудно да се уцели моментът, подходящ за измерване. Обаче по-важно се оказва, че самото измерване представлява самостоятелна и често трудна задача, особено когато искаме повишена точност на резултата. С цифров компютър несъизмеримо по-лесно може да се правят изчисления с поддръжка на хиляди точни цифри.

В общия случай затруднения възникват дори и с подготвянето на началните състояния на величините към момента на стартиране на изчислителния процес.

Още един проблем е работата с много големи или с много малки стойности. Съответните величини се поддържат трудно, а понякога въобще са недостижими на практика в условията на обкръжаващия ни свят.

Описаните трудности правят неприемливо пълноценното използване на аналогови компютри за *универсални* цели. Такъв подход повече подхожда за практически експерименти, отколкото за що-годе полезно програмиране.

При цифровите компютри, наричани още дискретни, числата се описват чрез краен брой различни помежду си величини, интерпретирани като стойности на цифри. Затова отсъствуват описаните по-горе неудобства, съпътстващи използване на аналогови схеми. По тази причина именно цифровите компютри са получили разпространение като универсални програмируеми конструкции, удобни за най-разнообразни алгоритми. Всъщност ограниченията за приложимост на цифровите устройства произхождат от самите възможности за алгоритмизиране по известните ни до момента начини. Така за някои задачи вече е доказано, че не съществува алгоритъм от познатите ни видове за намиране на решени (използва се терминът алгоритмично неразрешими проблеми), а за множество задачи са ни известни *само* силно неефективни алгоритми (по-точно – неефективни при голям брой входни данни).



Аналогови устройства сега се използват обикновено за строго ограничени цели, с тясна специализация и много често тези устройства са интегрирани с цифрови системи като техни части.

### **5. Необходимост от кодиране за представянето на числа в паметта**

Очевидно трябва да можем да работим с отрицателни и дробни числа. Обаче ако допуснем някои разряди в паметта да имат стойности минус или дробна запетая, различни от цифра, тогава възникват редица неудобства.

Например при кодирането на едно число може да потрѣбват голям брой цифри, но само един минус и една запетая. Тогава поддръжката за всеки разряд на стойности и цифри, и други е неоправдана. А когато само някои разряди може да имат стойност „-“ или „,“, тогава къде ще се разполагат те? А за какво ще служат, когато в същото място в паметта поискаме да запишем цяло неотрицателно число в код без знак?

Подход, при който някои разряди могат да имат стойност „-“ или „,“, би усложнил избора на машинни формати за данни и инструкции, структурата на паметта и въобще всички съставляващи на хардуера и машинния език. Би възникнала и неефективност в различни аспекти на работата на хардуера и софтуера.

Естественото решение за универсалните компютри е паметта да се състои от равноправни, уеднаквени разряди, а стойностите им да могат да бъдат само цифри.

Следователно за физическото представяне на отрицателно или дробно число е необходимо на това число да бъде съпоставян код, съставен само от цифри (и съответно имащ цяла неотрицателна стойност). Такъв вид код специално е предназначен да бъде записван в паметта.

### **6. За масовото разпространение на двоичните компютри**

Троичните компютри (базирани на троична ПБС) имат памет с много по-голям капацитет от този на двоичните. Обаче те изискват и съществено по-сложни хардуерни схеми, което поражда технологични и конструктивни трудности, а ги прави и съществено по-скъпи. В същото време двоичните компютри предоставят капацитет на паметта и функционалност, относително приемливи за болшинството потребители при по-ниски производствени разходи и продажни цени.

По този начин двоично базираната техника се оказва достатъчно примамлива за купувачите и с достатъчно евтино производство за да носи по-голяма печалба на производителите. Това се оказва решаващо за масовото разпространение на двоичните компютри.

Така основната движеща сила за избора между възможните алтернативни конструктивни решения и в това число между различните БС, на които може да се основава паметта, за универсални програмируеми устройства се оказва точно икономическата целесъобразност, произтичаща от комплексната оценка на баланса между извличаната полза за потребителите на техниката и необходимите разходи за производителите на тази техника.

Този текст е съсредоточен върху масово използваните универсални компютри и затова тук се разглежда само представянето на числа в двоична памет. Всъщност това се оказва достатъчно за да бъдат разбрани основните фундаментални идеи за представяне на числа в цифрови компютри и свързаната с тях логика на изграждането на електронни програмируеми устройства.

## 7. За значението на сумирането на цели числа

За да бъде алгоритъмът толкова гъвкав, колкото бихме искали да можем да го направим, много често се налага той да може да работи с разнообразни набори от входни и междинни данни. Затова пък трябва да има възможност за съответно удобно адресиране (указване на местоположението) на данни в паметта.

Нека припомним, че **адресът** е просто едно цяло неотрицателно число, един номер на минимална адресируема клетка от паметта, за каквато вече десетки години се избира осемразрядният байт. Но с адреси се работи само на машинно ниво, а в езиците от по-високо ниво, включително асемблерно, се използват указатели или техните аналози референции и псевдоними. **Указателят** е единно цяло, съставено от адрес и тип на стойност, разположена от този адрес. **Референциите** и **псевдонимите** се различават от указателите главно по правилата за създаването и употребата им.

За да може един и същ алгоритъм или отделна машинна инструкция в различни моменти да работи с различни данни е необходимо адресите на тези данни *да може да се изчисляват по време на изпълнение*. Така често се налага адресът на операнда да се получава чрез сумиране на две или три цели числа именно в процеса на изпълнение на машинната инструкцията, дори когато тя е измежду най-бързо изпълняваните. Такива начини на адресация типично се поддържат от универсалните процесори.

Следователно самото сумиране на цели числа трябва да става само за малка част от минималното време, предвидено за изпълнението на една инструкция, а значи – то трябва да бъде реализирано с пределната възможна ефективност по скорост и по време.

Същата потребност възниква и защото има много видове инструкции от машинния език, които трябва да стават бързо и просто и в заедно с това по някакъв начин използват сумиране. Например такива са някои начини за предаване на управлението (за преход), варианти на pop и push и междинни действия в различни изчисления с кодове на дробни числа. За такива обработки също са много съществени простотата и скоростта на сумиращите схеми.

Описаните и други подобни съображения превръщат събирането на цели числа в много съществен фактор за простотата и бързодействието на процесора. Затова на

много места в този текст ще коментираме особености на сумирането при различни представяния на числа.

За сумирането (точно както и за изваждането) е типично разрядностите на операндите и резултата да бъдат еднакви. Въпреки това, заради удобството на работа с разнотипни данни, много машинни езици включват и малък брой инструкции, получаващи сума с по-голяма разрядност от тази на събираемите. В същото време, за разбирането на фундаменталните идеи за машинна аритметика и за изграждане на универсални компютри е достатъчно да разглеждаме само сумиране с равни разрядности на операнди и резултат. Затова тук ще се ограничим само с този вариант.

## 8. Препълване и флагове

Всяка машинна инструкция е предназначена за обработка на данни с точно фиксирана разрядност. С това се опростява хардуерната реализация на машинния език и се ускорява изпълнението на програмния код. Обаче при изчисленията е възможно за кодирането на математически точния резултат да са необходими повече разряди, отколкото са предвидени от съответната машинна инструкция. Тогава казваме, че възниква **препълване** (това интуитивно съответствува на препълването на разрядната решетка, осигурена за резултата). В такива случаи пак има резултат (щом има разряди, предназначени за него – в тях има и стойност), но той просто няма да бъде верен. Затова препълването означава грешка.

Когато изчисляваме, ние непременно трябва да знаем, верен ли е резултатът. Това може лесно да се установява на машинно ниво. Но какъвто и да е, той трябва да бъде отделен от признака за препълване, за да се съхрани простотата на кода, представящ полученото число. Тук много удачна се оказва идеята за разряд, наричан **флаг за препълване**, който да се модифицира в процеса на изчислението, така че стойността му да показва, дали е възникнало препълване. Този разряд се намира извън кода резултат. След изчислението флагът остава достъпен за следващите инструкции и чрез него те могат да разклонят алгоритъма, според наличието на грешка.

Всъщност тази идея е съществено доразвита в процесорите.

**Флаг** наричаме обикновено един разряд, а само в много редки изключения флагът може да бъде в два или повече разряда, чиято стойност може да се използва за управление на изчислителния процес.

Има два вида флагове:

**Флаг на състоянието** наричаме този, който отразява какво вече се е случило (в последната от изпълнените машинни инструкции, влияещи на флага).

Такива флагове типично се използват за разклоняване на алгоритъма. Например в тях се отразяват препълванията при работа с различни кодове, полученият нулев резултат, знакът (знаковият разряд) на резултата, четността на броя на разрядите в младшия байт на резултата (това е актуално само при някои изчисления) и други.

**Управляващ флаг** наричаме такъв, чиято стойност или определя вариант, по който ще се изпълняват определени машинни инструкции, или забранява и разрешава изпълнението на някакво действие.

Например такъв е флагът за прекъсване, който забранява или разрешава на процесора да реагира на сигнали за така наречените маскируеми прекъсвания, т. е. такива, които *може* да бъдат игнорирани и въпреки това да продължи изпълнението на смислен изчислителен процес. В частност маскируеми са прекъсванията по сигнали, изпращани от периферни устройства.

Пример за сигнал за немаскируемо прекъсване, т. е. такъв, на който процесорът задължително *трябва* да реагира на границата между две поредни изпълнявани инструкции, е сигналът за грешка в работата на шината за данни (да напомним, че шината за данни е „сноп“ от линии, по които физически едновременно се предават всички цифри на една данна).

Обикновено в архитектурите се поддържат и инструкции, които могат да предават управлението (да извършват разклоняване на алгоритъма) според стойностите на някои управляващи флагове.

Флаговете са вградени в процесора и ние си ги представяме групирани в една клетка памет, наричана **флагов регистър**. (Нека припомним, че **регистърът** е клетка памет, вградена в процесорна схема.)

Често в различните блокове на процесора има съответни на тях отделни флагови регистри. Например в блока за работа с плаваща запетая има флагов регистър, обвързан с изпълняваните там машинните инструкции, а в управляващото и аритметико-логическо устройство има свой флагов регистър, задействуван главно при работа с цели числа и при различни управляващи инструкции.

В някои случаи бихме искали да можем едновременно да обработваме като една данна цял флагов регистър или негова част. Например да ги запишем в стека или да ги прочетем наведнъж от стека. Затова машинният език осигурява и малък брой инструкции с такива цели.

## 9. Код без знак

В компютърното програмиране термините „число без знак“ и „код без знак“ се отнасят само за цели числа (неотрицателни). Причината е, че за цели числа е удобно и

дори необходимо да се поддържат различни представяния, и съответни изчислителни възможности, и за неотрицателни, и за произволни цели числа, обаче за числа с дробна част няма да има осезаема полза, но би имало силно усложнение на хардуера и на машинния език, ако се поддържат варианти на кодове и за числа без знак, и за знакови числа.

При аритметика с реални числа въобще се следва принципът, всички възможни представяния първо да се преобразуват в един и същи вид разширен формат с плаваща запетая и чак после с този формат да се правят изчисленията. Съответно получените резултати отново може да се преобразуват в някакъв друг желан вид код (евентуално с допълнително закръгляне). Така тежките хардуерни реализации за изчисления с код с плаваща запетая, например за тригонометрични или показателни функции, се реализират само един път за само един вид код.

Кодът без знак се нарича така, защото знакът на кодираното число не се представя по никакъв начин в паметта, съдържаща кода, а се подразбира в съответните машинни инструкции.

В практиката обикновено КБЗ се използва за представяне на цели неотрицателни числа. Теоретически чрез КБЗ може да представят и само цели неположителни числа, но това се избягва, защото при умножение и делене няма да може с един и същ вид код да се представят и операндите, и получаваният резултат.

КБЗ, когато съществува, съвпада с кодираното число. Т. е.  $\text{КБЗ}_n(A) = A$ .

Следователно всяко кодирано число има единствен КБЗ и всеки КБЗ представя точно едно число.

Примери за КБЗ:

$$\text{КБЗ}_5(0) = 00000_{(2)};$$

$$\text{КБЗ}_3(7) = 111_{(2)};$$

$$\text{КБЗ}_5(7) = 00111_{(2)};$$

$$\text{КБЗ}_4(3) = 0011_{(2)};$$

$$\text{КБЗ}_6(3) = 000011_{(2)}.$$

Разрядността на кода определя броя на цифрите, с които кодираното число се записва. Съответно в  $n$ -разряден код може да бъдат кодирани  $2^n$  числа. Те съставляват точно интервала  $[0; 2^n - 1]$ .

Очевидно аритметичните операции с КБЗ се изпълняват, точно както се работи със самите кодирани числа.

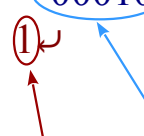
При сумиране и изваждане с КБЗ признак за препълване (т. е. за получаване на неверен резултат) е преносът или заемът наляво от старшия от разрядите, предвидени за резултата.

Примери за сумиране с КБЗ без препълване:

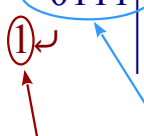
кодове без знак	кодирани числа	;	кодове без знак	кодирани числа
+ 00101	5		+ 101101	45
01010	10		000101	5
01111	15		110010	50

Примери за сумиране с КБЗ с препълване:

кодове без знак	кодирани числа	;	кодове без знак	кодирани числа
+ 11101	29		+ 1101	13
00101	5		1010	10
00010	2		0111	7



резултат (неверен)



резултат (неверен)

преносът единица наляво от старшия разряд е признак за препълване

Когато възниква препълване при сумиране на  $n$ -разрядни КБЗ, тогава резултатът е точно остатък от делението на действителната сума с делител  $2^n$ .

## 10. Прав код

В масово разпространените днес универсални компютри ПК не се използва, главно заради усложнените изчисления с него. Кодирането на нулата по два различни начина също е някакъв, макар и много малък недостатък.

В ПК старшият разряд е знаков, 0 за плюс и 1 за минус, а останалите разряди съдържат точно двоичния запис на абсолютната стойност на числото.

Изборът на нула за представяне на знак плюс има цел да може неголемите цели положителни числа да се представят чрез един и същ код и с КБЗ, и с ПК.

Примери:

$$\text{ПК}_6(+3) = 000011_{(2)};$$

$$\text{ПК}_4(+3) = 0011_{(2)};$$

$$\text{ПК}_6(-3) = 100011_{(2)};$$

$$\text{ПК}_4(-3) = 1011_{(2)}.$$

В  $n$ -разряден ПК абсолютната стойност на кодираното число трябва да се побира в  $n-1$  разряда. Обаче най-голямото  $n$ -цифрено двоично число е точно  $2^{n-1}-1$ . Следователно интервалът на кодираните числа за  $n$ -разряден ПК е  $[-(2^{n-1}-1); 2^{n-1}-1]$ .

Математическата формула за ПК, когато е възможен в искания брой разряди, е:

$$\text{ПК}_n(A) = \begin{cases} A, & A \geq 0 \\ 2^{n-1} - A, & A \leq 0 \end{cases}$$

Така за нулата се получават два различни кода, според знака, който се асоциира с нея. Например  $\text{ПК}_4(+0) = 0000_{(2)}$  и  $\text{ПК}_4(-0) = 1000_{(2)}$ . Това все пак е някакъв малък недостатък на ПК.

При сумирането с ПК признакът за препълване (получаване на неверен резултат) е преноса към знаковия разряд.

Примери за сумиране с ПК без препълване (т. е. с получаване на верен резултат):

$\begin{array}{r l} \text{ПК}_5(A) & A \\ \hline + 00101 & +5 \\ 01010 & +10 \\ \hline 01111 & +15 \end{array}$	$\begin{array}{r l} \text{ПК}_5(A) & A \\ \hline + 10101 & -5 \\ 11010 & -10 \\ \hline 11111 & -15 \end{array}$	$\begin{array}{r l} \text{ПК}_6(A) & A \\ \hline + 101101 & -13 \\ 000101 & +5 \\ \hline 101000 & -8 \end{array}$	$\begin{array}{r l} \text{ПК}_6(A) & A \\ \hline + 000101 & +5 \\ 101101 & -13 \\ \hline 101000 & -8 \end{array}$
---	---	---	---

фактически се извършва **сумиране**

фактически се извършва **изваждане**

Тук се виждат най-съществените недостатъци на сумирането с ПК:

- знаковият разряд се обработва отделно и по различен начин от всички останали;
- може да се наложи сравняване на абсолютните стойности на операндите;

- може да се изважда 1-та абсолютна стойност от 2-та, а може и 2-та от 1-та;
- сумирането (както и изваждането) изисква привличане в една и съща изчислителна схема на два качествено различни алгоритъма – за действително сумиране и за действително изваждане.

Сумата на числа с противоположни знакове винаги принадлежи на интервала с краища тези числа. Следователно сумата на два ПК с еднаква разрядност може да се кодира в ПК в същия брой разряди, в който са събираемите. Затова препълване може да възникне само при сумиране на числа с еднакъв знак (знаков разряд).

Когато възниква препълване при сумиране на  $n$ -разрядни ПК, тогава резултатът е точно знаковият остатък (получен чрез отсичане със запазване на знака) от делението на действителната сума на операндите с делител  $2^{n-1}$ .

Примери за сумиране с ПК с препълване (с получаване на неверен резултат):

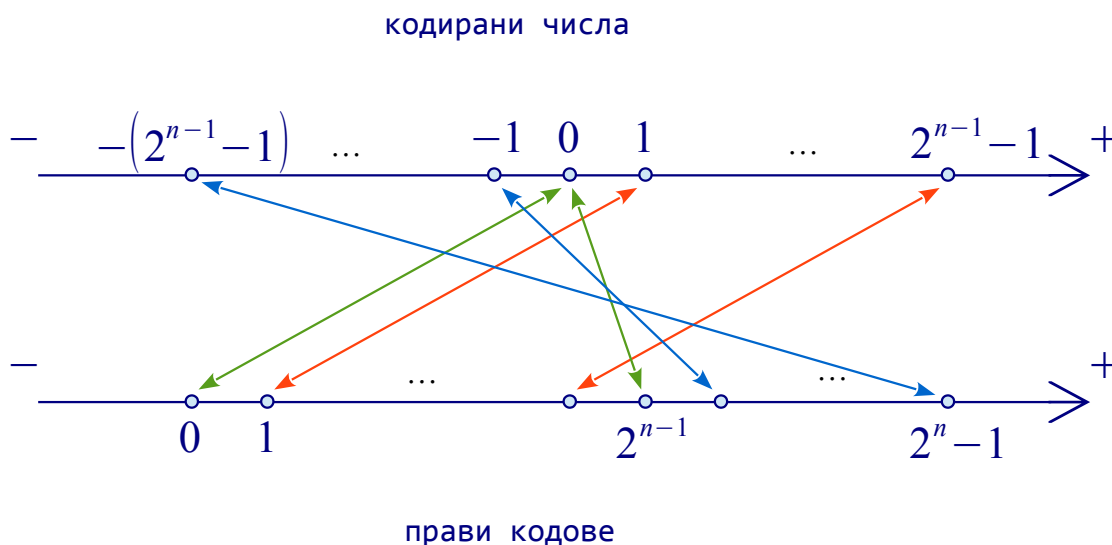
$\text{ПК}_5(A)$ $\hline + 01000$ $01001$ $\hline 00001$	$A$ $\hline +8$ $+9$ $\hline +1$	;	$\text{ПК}_5(A)$ $\hline + 01111$ $00001$ $\hline 00000$	$A$ $\hline +15$ $+1$ $\hline +0$	;	$\text{ПК}_5(A)$ $\hline + 11000$ $11001$ $\hline 10001$	$A$ $\hline -8$ $-9$ $\hline -1$	;	$\text{ПК}_5(A)$ $\hline + 11000$ $11000$ $\hline 10000$	$A$ $\hline -8$ $-8$ $\hline -0$
<div style="border: 1px solid red; border-radius: 50%; width: 20px; height: 20px; display: flex; align-items: center; justify-content: center; margin: 0 auto;">1</div>			<div style="border: 1px solid red; border-radius: 50%; width: 20px; height: 20px; display: flex; align-items: center; justify-content: center; margin: 0 auto;">1</div>			<div style="border: 1px solid red; border-radius: 50%; width: 20px; height: 20px; display: flex; align-items: center; justify-content: center; margin: 0 auto;">1</div>			<div style="border: 1px solid red; border-radius: 50%; width: 20px; height: 20px; display: flex; align-items: center; justify-content: center; margin: 0 auto;">1</div>	

пренос единица към знаковия разряд;  
 това е признак за препълване

ПК на отрицателно число започва с едно, а на положително – с нула. Следователно, при равна разрядност, всеки ПК на отрицателно число е по-голям от всеки ПК на положително число. Т. е. наредбата на кодираните числа е **различна** от наредбата на техните ПК. Отделно наредбата на отрицателните числа е точно противоположна на наредбата на техните кодове.



Съответствието между кодирани числа и техните ПК е следното:



## 11. Обратен код

В масово разпространените днес универсални компютри ОК не се използва, главно защото сумирането с ОК е различно от това с КБЗ, а също и поради двата различни кода на нулата.

В ОК старшият разряд също е знаков (0 за плюс и 1 за минус), но в аритметичните действия всички разряди се обработват по един и същ начин, при това значително по-прост, отколкото при КБЗ.

Изборът на нула за представяне на знак плюс в ОК, съвсем аналогично на ПК и ДК, има цел да може неголемите цели положителни числа да се кодират чрез един и същ код и с КБЗ, и с ОК.

Получаването на двоичен ОК за отрицателно число става чрез поразрядно инвертиране на абсолютната стойност на числото, записана с толкова цифри, колкото е разрядността на кода. Това преобразование се нарича допълнение до едно.

Тук се има предвид, че едно е с единица по-малко от основата  $2$  на бройната система. Съответното преобразование при ПБС с основа  $S$  се нарича допълнение до  $S-1$ . За получаването на ОК в памет, базирана на ПБС с основа  $S$ , в  $S$ -ичния запис на абсолютната стойност на кодираното число всяка цифра със стойност  $\xi$  се заменя с цифрата със стойност  $S-1-\xi$ . Така сумата на двете цифри (началната и заменящата) винаги е  $S-1$ , откъдето идва и названието на преобразованието.

Пример за получаване на ОК в 6 разряда:

кодирано число:	$-11_{(10)}$
абсолютна стойност с 6 цифри:	$001011_{(2)}$
обратен код в 6 разряда:	$110100_{(2)}$

Примери за ОК:

$$\text{ОК}_6(+3) = 000011_{(2)};$$

$$\text{ОК}_6(-3) = 111100_{(2)};$$

$$\text{ОК}_4(-3) = 1100_{(2)}.$$

При получаването на  $n$ -разряден ОК на отрицателно число, след поразрядно инвертиране трябва да бъде правилен знаковият разряд, т. е. да бъде единица за минус. Следователно в запис на абсолютната стойност на кодираното число с  $n$  цифри старшият разряд трябва да бъде нула. Оттук интервалът на кодираните числа отново е  $[-(2^{n-1}-1); 2^{n-1}-1]$ , точно както при ПК.

За всяка разрядност нулата има два различни ОК. Например  $\text{ОК}_4(+0) = 0000_{(2)}$  и  $\text{ОК}_4(-0) = 1111_{(2)}$ . Това е недостатък на ОК, макар и относително малък, защото сравняването на число с нула се прави изключително често на ниско ниво на програмиране заради поддръжката на флаг за нулев резултат, а при ОК проверката за нула трябва да отчита два качествено различни възможни кода на нулата.

Сумата на  $n$ -цифрено цяло неотрицателно число и числото, получено от него чрез поразрядно инвертиране, се записва с  $n$  единици, т. е. тя е равна на  $2^n - 1$ .

Например:

$$\begin{array}{r} +0010 \\ 1101 \\ \hline 1111 \end{array} \text{ и } 1111_{(2)} = 2^4 - 1 = 15.$$

Следователно математическата формула за ОК, когато той е възможен в искания брой разряди, е:

$$\text{ОК}_n(A) = \begin{cases} A, & A \geq 0 \\ 2^n - 1 + A, & A < 0 \end{cases}$$

Тук се отчитат и двата различни кода за нулата при различни знакове, асоциирани с нея.

От формулата се вижда, в какъв смисъл ОК на отрицателно число може да се разглежда като допълнение на абсолютната му стойност до  $2^n - 1$ .

Сумирането с ОК става на два етапа – първо се намира сумата на кодовете, а после към нея се прибавя преносът, който е възникнал наляво от старшия разряд. Това се доказва лесно чрез използването на горната формула.

При сумиране на ОК признакът за препълване (получаване на неверен резултат) е неправилният знаков разряд.

Примери за сумиране с ОК без препълване (**правилният знак е признак, че няма препълване**):

$OK_5(A)$	$A$	$OK_5(A)$	$A$	$OK_5(A)$	$A$	$OK_5(A)$	$A$
$+11011$	$-4$	$+10000$	$-15$	$+11011$	$-4$	$+00100$	$4$
$+01010$	$10$	$+01010$	$10$	$+10101$	$-10$	$+01010$	$10$
$00101$	$5$	$11010$	$-5$	$10000$	$-15$	$01110$	$14$
$1 \leftarrow$		$0 \leftarrow$		$1 \leftarrow$		$0 \leftarrow$	
$\hookrightarrow +1$		$\hookrightarrow +0$		$\hookrightarrow +1$		$\hookrightarrow +0$	
$00110$	$6$	$11010$	$-5$	$10001$	$-14$	$01110$	$14$

Примери за сумиране с ОК с възникване на препълване:

$OK_4(A)$	$A$	$OK_4(A)$	$A$	$OK_5(A)$	$A$	$OK_5(A)$	$A$
$+1101$	$-2$	$+0101$	$5$	$+10111$	$-8$	$+01111$	$15$
$+1000$	$-7$	$+0100$	$4$	$+10001$	$-14$	$+00001$	$1$
$0101$	$5$	$1001$	$-6$	$01000$	$8$	$10000$	$-15$
$1 \leftarrow$		$0 \leftarrow$		$1 \leftarrow$		$0 \leftarrow$	
$\hookrightarrow +1$		$\hookrightarrow +0$		$\hookrightarrow +1$		$\hookrightarrow +0$	
$0110$	$6$	$1001$	$-6$	$01001$	$9$	$10000$	$-15$

**неправилният знак е признак за препълване**

Сумата на числа с противоположни знакове винаги принадлежи на интервала с краища тези числа. Следователно сумата може да се кодира в ОК в същия брой разряди,

в който може да се кодират в ОК и двете събираеми. Затова препълване е възможно само при еднакви знакови разряди в двете събираеми.

При сумиране с ОК всички разряди се обработват по един и същ начин, без значение какви са знаковете или абсолютните стойности на събираемите. Т. е. при ОК отсъствуват изброените по-горе най-съществени недостатъци на ПК. Обаче остава недостатъкът с два кода за нулата.

Недостатък на ОК е и *разликата между алгоритмите за сумиране на числа*, представени в КБЗ и в ОК. Заради това в машинния език стават необходими две различни инструкции за сумиране с различните кодове, което малко намалява гъвкавостта на алгоритмите, защото един и същ алгоритъм вече трябва да се изпълнява по различен начин за начални данни в ОК и в КБЗ.

В машинната реализация на сумирането с ОК за разпознаване на препълването се използва **модифициран ОК**. В него се добавя  $(n+1)$ -и старши разряд, дублиращ знаковия. След това се получава резултат също с два знакови разряда и препълване има, точно когато тези два разряда са различни. По-надолу са приведени горните примери за сумиране с ОК, но вече с използване на модифициран ОК (знаковите разряди са написани удебелено).

Сумиране с *модифициран ОК* без препълване:

$\text{МОК}_6(A)$	$A$	$\text{МОК}_6(A)$	$A$	$\text{МОК}_6(A)$	$A$	$\text{МОК}_6(A)$	$A$
$\begin{array}{r} + 111011 \\ + 001010 \\ \hline 000101 \\ 1 \leftarrow \\ \hookrightarrow +1 \\ \hline 000110 \\ \text{результат} \end{array}$	$\begin{array}{r} -4 \\ 10; \\ \hline 5 \end{array}$	$\begin{array}{r} + 110000 \\ + 001010 \\ \hline 111010 \\ 0 \leftarrow \\ \hookrightarrow +0 \\ \hline 111010 \\ \text{результат} \end{array}$	$\begin{array}{r} -15 \\ 10; \\ \hline -5 \end{array}$	$\begin{array}{r} + 111011 \\ + 110101 \\ \hline 110000 \\ 1 \leftarrow \\ \hookrightarrow +1 \\ \hline 110001 \\ \text{результат} \end{array}$	$\begin{array}{r} -4 \\ -10; \\ \hline -15 \end{array}$	$\begin{array}{r} + 000100 \\ + 001010 \\ \hline 001110 \\ 0 \leftarrow \\ \hookrightarrow +0 \\ \hline 001110 \\ \text{результат} \end{array}$	$\begin{array}{r} 4 \\ 10; \\ \hline 14 \end{array}$

**еднаквите знакови разряди са признак, че няма препълване (резултатът е верен)**

Сумиране с модифициран ОК с възникване на препълване:

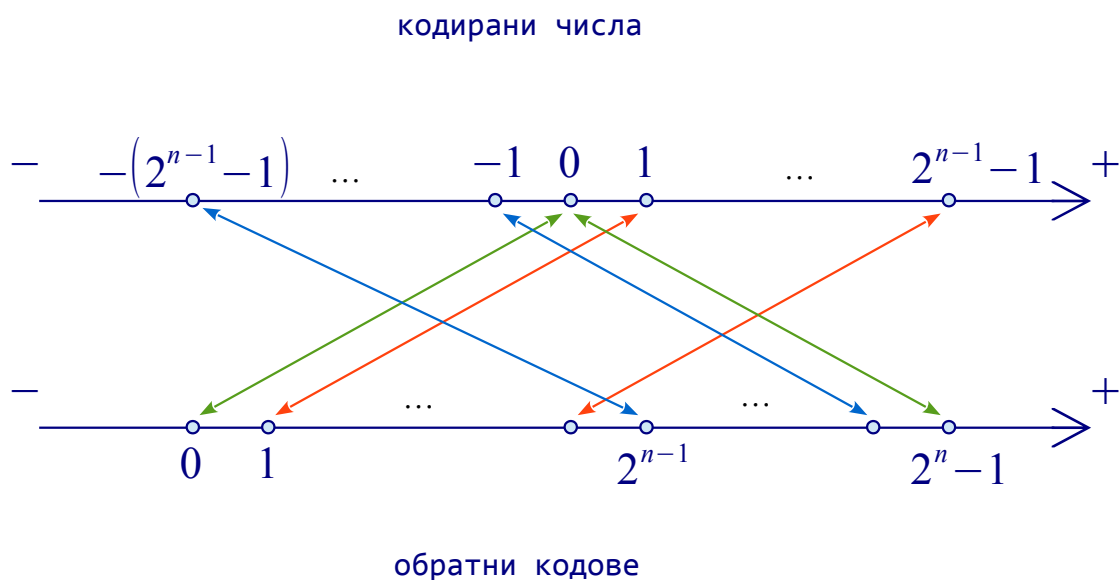
$\text{МОК}_5(A)$	$A$	$\text{МОК}_5(A)$	$A$	$\text{МОК}_6(A)$	$A$	$\text{МОК}_6(A)$	$A$
$\begin{array}{r} +11101 \\ +11000 \\ \hline 10101 \\ 1\swarrow \\ \swarrow +1 \\ \hline 10110 \\ \text{результат} \end{array}$	$\begin{array}{r} -2 \\ -7; \\ \hline \mp 5 \end{array}$	$\begin{array}{r} +00101 \\ +00100 \\ \hline 01001 \\ 0\swarrow \\ \swarrow +0 \\ \hline 01001 \\ \text{результат} \end{array}$	$\begin{array}{r} 5 \\ 4; \\ \hline \pm 6 \end{array}$	$\begin{array}{r} +110111 \\ +110001 \\ \hline 101000 \\ 1\swarrow \\ \swarrow +1 \\ \hline 101001 \\ \text{результат} \end{array}$	$\begin{array}{r} -8 \\ -14; \\ \hline \mp 8 \end{array}$	$\begin{array}{r} +001111 \\ +000001 \\ \hline 010000 \\ 0\swarrow \\ \swarrow +0 \\ \hline 010000 \\ \text{результат} \end{array}$	$\begin{array}{r} 15 \\ 1. \\ \hline \pm 15 \end{array}$
$+6$		$+0$	$-1$	$+9$		$+0$	

**различните знакови разряди са признак, че има препълване (резултатът не е верен)**

Използването на модифициран код опростява и ускорява проверката на верността на резултата, защото избягва разклонението в процеса на проверка, в зависимост от това, дали знаковете на операндите са еднакви или различни. Подобна находчивост в изграждането на хардуерните схеми, изглежда трудно достижима при разсъждаване на високо абстрактно ниво, но в действителност е закономерно следствие от внимателното проектиране на детайлите на схемата.

ОК на отрицателно число започва с единица, а на положително – с нула. Следователно при равна разрядност ОК на отрицателните числа са по-големи от ОК на положителните числа. Т. е. наредбата на кодираните числа е **различна** от наредбата на техните ОК. Обаче в рамките на един и същ знак на кодираните числа има еднаква наредба на кодирани числа и техните ОК.

Съответствието между кодирани числа и техните обратни кодове е следното:



## 12. Допълнителен код

В масово разпространените днес универсални компютри за представяне на цели числа със знак се използва ДК, а в езиците за програмиране от високо ниво – изключително само ДК. Например такива са всички знакови целочислени типове в C++, Java, C#, Free Pascal и много други езици.

В ДК старшият разряд също е знаков (отново 0 за плюс и 1 за минус), но при изчисления всички разряди се обработват по един и същи начин. Изборът на нула за представяне на знак плюс отново има за цел да може неголемите цели положителни числа да се кодират чрез един и същ код и с КБЗ, и с ДК.

Положителните числа и в ДК се представят със своята стойност. Получаването на ДК за отрицателно число (вижте примерите по-надолу) в двоична памет става, като абсолютната стойност на числото, записана с толкова цифри, колкото е разрядността на кода, се инвертира поразрядно и към получения резултат се прибавя единица.

Понякога двойката преобразования инвертиране и прибавяне на едно се нарича допълнение до две. Т. е. ДК на отрицателно число се получава чрез прилагане на допълнение до две към абсолютната стойност на числото.

Обаче този термин се отнася само за двоична ПБС. Когато паметта е базирана на ПБС с основа  $S$ , тогава не е дефинирано понятието инвертиране и за получаването на ДК първо се изпълнява допълнение до  $S-1$ , т. е. всяка цифра със стойност  $\xi$  се заменя с цифрата със стойност  $S-1-\xi$ , а после към полученото число се прибавя единица. За такова преобразование се използва терминът допълнение до  $S$ .

Само при получаването на ДК за  $-0$  е възможно (вижте примера за  $ДК_6(-0)$  по-надолу) прибавянето на единица да даде  $(n+1)$ -разряден резултат. Тогава за ДК се вземат само младшите  $n$  разряда и така се получава отново ДК нула. Т. е.  $ДК_n(+0) = ДК_n(-0) = 0$ .

Примери за получаване на ДК в 6 разряда:

кодирано число:	$-11_{(10)}$
абсолютна стойност с 6 цифри:	$001011_{(2)}$
число след поразрядно инвертиране:	$110100_{(2)}$
число след прибавяне на единица:	$110101_{(2)}$
допълнителен код:	$110101_{(2)}$

кодирано число:	$-0_{(10)}$
абсолютна стойност с 6 цифри:	$000000_{(2)}$
число след поразрядно инвертиране:	$111111_{(2)}$
число след прибавяне на единица:	$1000000_{(2)}$
допълнителен код:	$000000_{(2)}$

Щом нулата има единствен код, то броят на кодираните числа в  $n$  разряда е точно равен на броя на ДК в  $n$  разряда, т. е. – равен е на  $2^n$ . Следователно интервалът на кодираните числа в ДК в  $n$  разряда не е симетричен спрямо нулата и е различен от интервалите на кодирани числа в ПК или в ОК със същата разрядност.

Затова за всяка разрядност на код съществува единствено число, което може да се кодира в тази разрядност с ДК, обаче не може с ОК или с ПК. За разрядност  $n$  това е числото  $-2^{n-1}$ . Например:

кодирано число:	$-32_{(10)}$
абсолютна стойност с 6 цифри:	$100000_{(2)}$
число след поразрядно инвертиране:	$011111_{(2)}$
число след прибавяне на единица:	$100000_{(2)}$
допълнителен код на $-32$ в 6 разряда:	$100000_{(2)}$

$-32 = -2^5$  няма 6-разряден ОК или ПК обаче има 6-разряден ДК.

Примери за ДК:

$$\text{ДК}_6(+3) = 000011_{(2)};$$

$$\text{ДК}_4(+3) = 0011_{(2)};$$

$$\text{ДК}_6(-3) = 111101_{(2)};$$

$$\text{ДК}_4(-3) = 1101_{(2)}.$$

При получаването на  $n$ -разряден ДК на отрицателно число, след поразрядно инвертиране трябва да бъде правилният знаковият разряд (да бъде единица за минус). Следователно старшият разряд на абсолютната стойност на кодираното число трябва

да бъде нула. Оттук и от особеността на кодирането на  $-0$  следва, че интервалът на кодираните числа е  $[-2^{n-1}; 2^{n-1}-1]$ .

Като се използват разсъждения, аналогични на тези за ОК, и се отчита прибавянето на единица при получаването на ДК за отрицателните числа, се получава следната формула за ДК, когато изобщо съществува такъв:

$$\text{ДК}_n(A) = \begin{cases} A, & A \geq 0 \\ 2^n + A, & A < 0 \end{cases}$$

От формулата се вижда, в какъв смисъл ДК на отрицателно число е допълнение на абсолютната му стойност до  $2^n$ .

Лесно се вижда, че описаното преобразование, приложено към самия код на отрицателно число, дава абсолютната стойност на числото. Т. е.:

$$A < 0 \Rightarrow \text{ДК}_n(-\text{ДК}_n(A)) = 2^n - \text{ДК}_n(A) = |A|$$

При сумиране на ДК признакът за препълване (получаване на неверен резултат) е неправилния знаков разряд, както и при ОК.

Сумирането на ДК става точно както се извършва сумиране на КБЗ, но с игнориране на преноса наляво от старшия (знаковия) разряд.

От горния интервал на кодираните числа се вижда, че при сумиране на кодове на числа с противоположни знакове резултатът винаги ще бъде верен, защото сумата лежи между събираемите и следователно принадлежи на интервала на числата, кодирани със същия вид код. Следователно препълване може да се получи само при сумиране на числа с еднакъв знак.

Примери за сумиране с ДК без препълване (преносите, написани с червено, се игнорират в резултата):

$\text{ДК}_5(A)$	$A$	$\text{ДК}_5(A)$	$A$	$\text{ДК}_5(A)$	$A$	$\text{ДК}_5(A)$	$A$
$\begin{array}{r} +11100 \\ 01010 \\ \hline 00110 \end{array}$	$-4$	$\begin{array}{r} +10001 \\ 01010 \\ \hline 11011 \end{array}$	$-15$	$\begin{array}{r} +11100 \\ 10110 \\ \hline 10010 \end{array}$	$-4$	$\begin{array}{r} +00100 \\ 01010 \\ \hline 01110 \end{array}$	$4$
$\begin{array}{r} 10 \\ \hline 6 \end{array}$	$10$	$\begin{array}{r} 10 \\ \hline -5 \end{array}$	$10$	$\begin{array}{r} -10 \\ \hline -14 \end{array}$	$-10$	$\begin{array}{r} 10 \\ \hline 14 \end{array}$	$10$
$\begin{array}{r} 1 \\ \leftarrow \end{array}$		$\begin{array}{r} 0 \\ \leftarrow \end{array}$		$\begin{array}{r} 1 \\ \leftarrow \end{array}$		$\begin{array}{r} 0 \\ \leftarrow \end{array}$	

правилният знак е признак за верен резултат,  
т. е. няма препълване



Примери за сумиране с ДК с възникване на препълване:

$ДК_4(A)$	$A$	$ДК_4(A)$	$A$	$ДК_5(A)$	$A$	$ДК_5(A)$	$A$
$\begin{array}{r} +1110 \\ +1001 \\ \hline 0111 \end{array}$	$\begin{array}{r} -2; \\ -7 \\ 7 \end{array}$	$\begin{array}{r} +0101 \\ +0100 \\ \hline 1001 \end{array}$	$\begin{array}{r} 5; \\ 4 \\ -7 \end{array}$	$\begin{array}{r} +11000 \\ +10010 \\ \hline 01010 \end{array}$	$\begin{array}{r} -8; \\ -14 \\ 10 \end{array}$	$\begin{array}{r} +01111 \\ +00001 \\ \hline 10000 \end{array}$	$\begin{array}{r} 15. \\ 1 \\ -16 \end{array}$
1 ↵		0 ↵		1 ↵		0 ↵	

неправилният знак е признак за препълване,  
т. е. за неверен резултат

При сумиране в ДК всички разряди се обработват по един и същи начин, без значение какви са знаковете или абсолютните стойности на събираемите, и алгоритъмът за сумиране е същия, както при КБЗ, макар и с игнориране на преноса извън разрядната решетка. Затова при ДК отсъствуват всички изброени по-горе недостатъци на ПК и на ОК.

В машинната реализация на сумирането с ДК за разпознаване на препълването се използва модифициран ДК (аналогично на МОК). В него се добавя  $(n+1)$ -и старши разряд, дублиращ знаковия. След това се получава резултат също с два знакови разряда и препълване има, точно когато тези два разряда са различни. По-надолу са приведени горните примери за сумиране с ДК, но вече с използване на модифициран ДК (знаковите разряди са написани удебелено).

Сумиране с модифициран ДК без препълване:

$МДК_6(A)$	$A$	$МДК_6(A)$	$A$	$МДК_6(A)$	$A$	$МДК_6(A)$	$A$
$\begin{array}{r} +111100 \\ +001010 \\ \hline 000110 \end{array}$	$\begin{array}{r} -4; \\ 10 \\ 6 \end{array}$	$\begin{array}{r} +110001 \\ +001010 \\ \hline 111011 \end{array}$	$\begin{array}{r} -15; \\ 10 \\ -5 \end{array}$	$\begin{array}{r} +111100 \\ +110110 \\ \hline 110010 \end{array}$	$\begin{array}{r} -4; \\ -10 \\ -14 \end{array}$	$\begin{array}{r} +000100 \\ +001010 \\ \hline 001110 \end{array}$	$\begin{array}{r} 4. \\ 10 \\ 14 \end{array}$
резултат		резултат		резултат		резултат	
1 ↵		0 ↵		1 ↵		0 ↵	

еднаквите знакови разряди са признак, че няма препълване (резултатът е верен)

Сумиране с модифициран ДК с възникване на препълване:

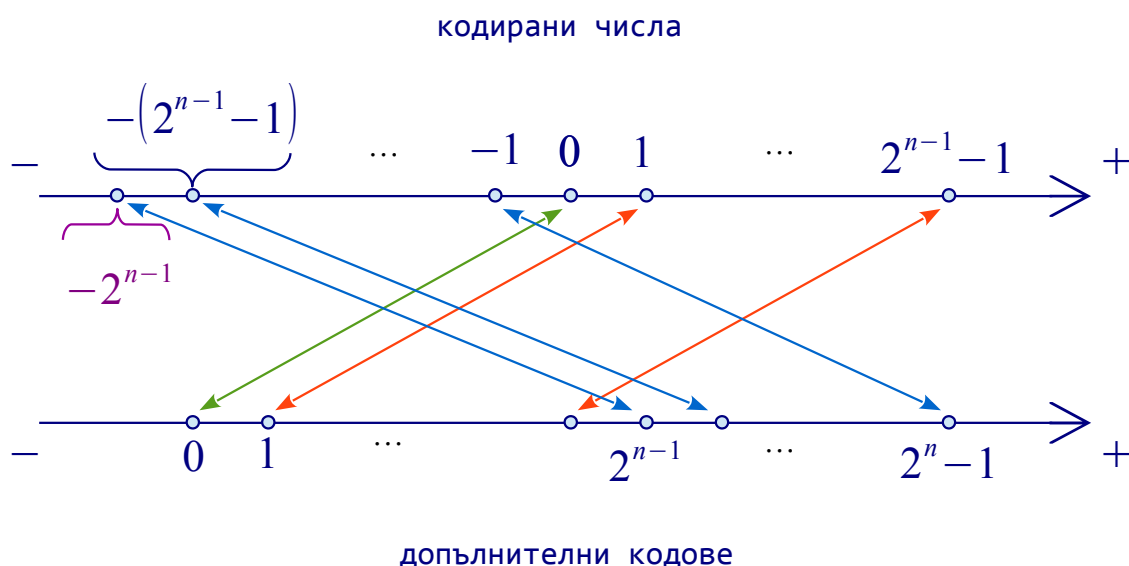
$\text{МДК}_5(A)$	$A$	$\text{МДК}_5(A)$	$A$	$\text{МДК}_6(A)$	$A$	$\text{МДК}_6(A)$	$A$
$\begin{array}{r} +11110 \\ +11001 \\ \hline 10111 \end{array}$	$\begin{array}{r} -2 \\ -7 \\ \hline +7 \end{array}$	$\begin{array}{r} +00101 \\ +00100 \\ \hline 01001 \end{array}$	$\begin{array}{r} 5 \\ 4 \\ \hline -7 \end{array}$	$\begin{array}{r} +111000 \\ +110010 \\ \hline 101010 \end{array}$	$\begin{array}{r} -8 \\ -14 \\ \hline +10 \end{array}$	$\begin{array}{r} +001111 \\ +000001 \\ \hline 010000 \end{array}$	$\begin{array}{r} 15 \\ 1 \\ \hline -16 \end{array}$
результат		результат		результат		результат	
1 ↵		0 ↵		1 ↵		0 ↵	

различните знакови разряди са признак, че има препълване (резултатът не е верен)

Използването на модифициран код опростява и ускорява проверката на верността на резултата, защото избягва разклонението в процеса на проверка, в зависимост от това, дали знаковете на операндите са еднакви или различни.

ДК на отрицателно число започва с единица, а на положително – с нула. Следователно при равна разрядност ДК на отрицателните числа са по-големи от ДК на положителните числа. Т. е. наредбата на кодирани числа е **различна** от наредбата на техните ДК. Обаче наредбите на кодирани числа и на техните ДК ще са еднакви само за отрицателни или само за неотрицателни кодирани числа.

Съответствието между кодирани числа и техните ДК е следното:



### 13. Изместен код

Машинните езици на масово разпространените днес универсални компютри нямат инструкции с операнди в ИК. В тях ИК се използва *само* като част от КПЗ, където допринася съществени удобства.

Отправната идея за ИК е кодовете да имат същата наредба като кодираните числа. Това е особено полезно за кодирането на дробни числа с плаваща запетая. (При ПК, ОК и ДК наредбата на кодовете е различна от наредбата на кодираните числа.)

Следващата определяща идея е – при еднаква разрядност на кодовете да може едни и същи числа да се кодират и в ИК, и в ДК. Т. е.  $n$ -разряден ИК имат точно числата от интервала  $\left[-2^{n-1}; 2^{n-1} - 1\right]$ .

Разбира се самите ИК в  $n$  разряда са числата от интервала  $\left[0; 2^n - 1\right]$ . Следователно формулата за ИК, когато е възможен, е:

$$\text{ИК}_n(A) = 2^{n-1} + A$$

Примери за ИК:

$$\text{ИК}_6(+3) = 100011_{(2)};$$

$$\text{ИК}_5(+3) = 10011_{(2)};$$

$$\text{ИК}_4(+3) = 1011_{(2)};$$

$$\text{ИК}_3(+3) = 111_{(2)};$$

$$\text{ИК}_6(-3) = 011101_{(2)};$$

$$\text{ИК}_5(-3) = 01101_{(2)};$$

$$\text{ИК}_4(-3) = 0101_{(2)};$$

$$\text{ИК}_3(-3) = 001_{(2)}.$$

Двоичният запис на ИК се получава най-лесно по формулата за ИК като се изчислява директно в двоична ПБС. Например:

$$2^6 = \underbrace{1000000}_{6 \text{ нули}}_{(2)}; \quad 19_{(10)} = 10011_{(2)}; \quad \begin{array}{r} + 1000000 \\ 10011 \\ \hline 1010011 \end{array}$$

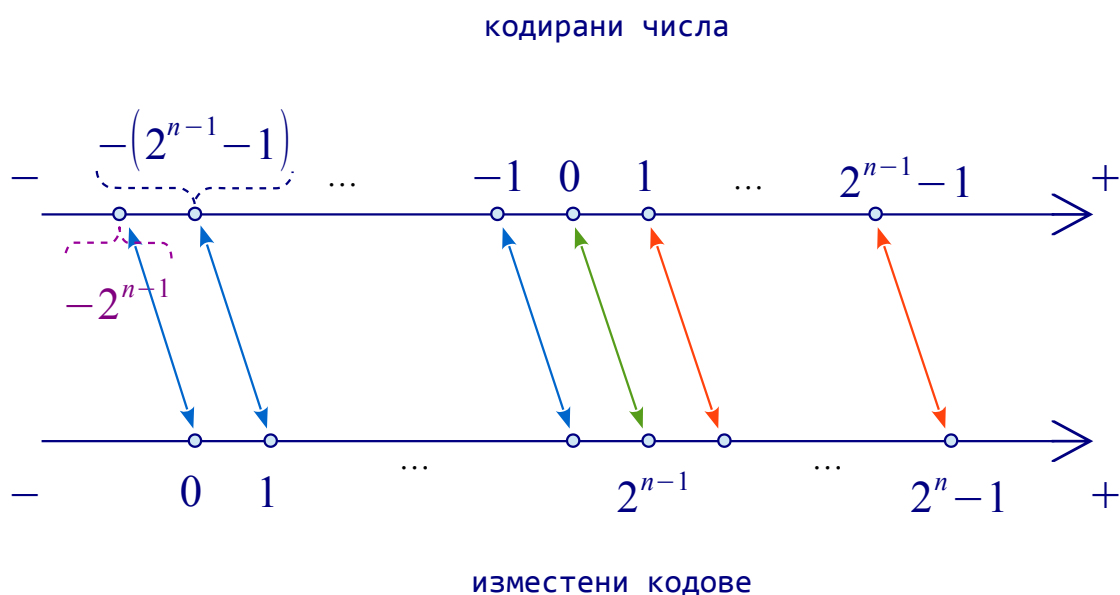
$$\Rightarrow \text{ИК}_7(19) = 2^6 + 19 = 1010011_{(2)};$$

$$2^6 = \underbrace{1000000}_{6 \text{ нули}}_{(2)}; \quad -19_{(10)} = -10011_{(2)}; \quad \begin{array}{r} - 1000000 \\ 10011 \\ \hline 0101101 \end{array},$$

като тук след всички заеми всъщност се изчислява

$$\begin{array}{r} - 0111112 \\ 10011 \\ \hline 0101101 \end{array} \text{ и } \Rightarrow \text{ИК}_7(-19) = 2^6 - 19 = 0101101_{(2)}.$$

Съответствието между кодирани числа и техните изместени кодове е следното:



Особено важно свойство на ИК е, че когато съществуват едновременно и  $\text{ИК}_n(A+k)$ , и  $\text{ИК}_n(A)$ , където  $k$  е цяло число, тогава:

$$\text{ИК}_n(A+k) = \text{ИК}_n(A) + k$$

При ИК старшият разряд не се интерпретира като знаков и не е прието да се нарича знаков, но той носи информация за знака на кодираното число, защото от формулата за ИК следва, че ИК на отрицателно число започва с нула, а ИК на неотрицателно число започва с единица.

#### **14. Кодове с двоично кодиране на десетичните цифри**

За тези кодове са много разпространени названията двоично-десетичен код и 2-10 код (което се чете „две десет код“).

При КДКДЦ числото се представя в паметта чрез редица от двоично кодирани стойности на цифрите от десетичния му запис. Такава идея потенциално е приложима и за дробни числа, но на практика не се използва.

Има два начина за кодиране на числа в КДКДЦ: **пакетиран формат**, когато две десетични цифри се записват в един байт и **непакетиран формат**, когато всяка десетична цифра е в отделен байт. За числа със знак се добавя и код на знака, като в пакетирания формат за знака на числото може да се използва само половин байт или цял байт, а в непакетирания формат знакът заема цял байт. При това, и плюсьт, и минусът може да се кодират с по няколко различни кода, в зависимост от конкретната архитектура на процесора.

На ниско ниво в масово разпространените днес универсални компютри КДКДЦ се поддържа ограничено и само за цели числа. Например няма умножение и делене с пакетирани еднобайтови кодове (терминът е обяснен малко по-надолу), а за десетбайтовите КДКДЦ (на цели числа) аритметичните действия са възможни само чрез представяне на операндите в КПЗ (първо се преобразува КДКДЦ към КПЗ, а после се изчислява с получените операнди в КПЗ). Т. е. има инструкции с операнди в КДКДЦ, но в момента на самото изчисление всички операнди вече са в КПЗ.

В практически реализираните архитектури са били използвани голям брой различни модификации на КДКДЦ, но те вече не са съществено актуални и няма да ги описваме в този текст. Ще отбележим само, че в архитектурата Intel x86 машинният език разпознава: еднобайтов НКДКДЦ без знак (т. е. код за едноцифрени десетични числа без знак) с поддръжка на четирите аритметични действия; еднобайтов ПКДКДЦ без знак (т. е. код за двуцифрени десетични числа без знак) с поддръжка само на събиране и изваждане; знаков десетбайтов ПКДКДЦ (за 18-цифрени десетични цели числа със знак) с поддръжка на аритметика чрез преобразуване на ПКДКДЦ към КПЗ, изчисляване с КПЗ и преобразуване на резултата обратно към ПКДКДЦ.

## 15. Умножение на цели числа и точност на представянето

От четирите основни аритметични операции с цели числа точно при умножението броят на цифрите, необходими за записване на резултата, може най-много да нарасне спрямо броя на цифрите в записите на операндите. Затова при умножението може най-много да се загуби точност в представянето на резултата.

По тази причина обикновено в машинните езици за умножаване на цели числа се предвиждат инструкции, които от  $n$ -разрядни множители получават  $(2 \cdot n)$ -разрядно произведение.

Обаче за удобство на често използваните изчисления с малки по абсолютна стойност операнди като изключение се добавят и инструкции, които от  $n$ -разрядни множители в ДК получават резултат също  $n$ -разрядно произведение в ДК. Това е удобно, защото произведението се побира в код със същата разрядност, с каквата са представени множителите, и алгоритъмът може да избегне преобразуването на междинни резултати от разрядност  $(2 \cdot n)$  в разрядност  $n$ .

## 16. Фундаментални принципи за кодиране на цели числа

Системата от формати за представяне на цели числа в машинните езици на масово разпространените универсални компютри следва естествени общи принципи. Част от тях са валидни въобще за всички данни и инструкции в езика. Някои такива принципи са:

**Първо**, всички величини, обработвани от централния процесор, се записват (в процесора, в оперативната памет и в портовете) *само с цифри* (точно затова компютрите се наричат цифрови).

**Второ**, използват се само кодове, които заемат *точно цяло число байтове*.

По-общо, това важи за всички данни и инструкции в машинния език и значително го опростява.

**Байтът** е най-малката адресируема клетка. Следователно достъпът до част на байт изисква в машинната инструкция, освен полетата за адрес на байт, да се добавя и допълнително поле за указване на местоположението на съответната част вътре в байта. Това усложнява машинния формат на инструкциите, а оттам и на устройството на хардуерните схеми. Освен това се увеличава и разрядността на инструкциите, а значи и на машинните програми.

Разбира се полета за назоваване на местоположение на разряд вътре в байт се използват в много инструкции, например за модифициране на отделни разряди, но вече много отдавна се избягват числови формати, които не са подравнени по краищата на байтовете.

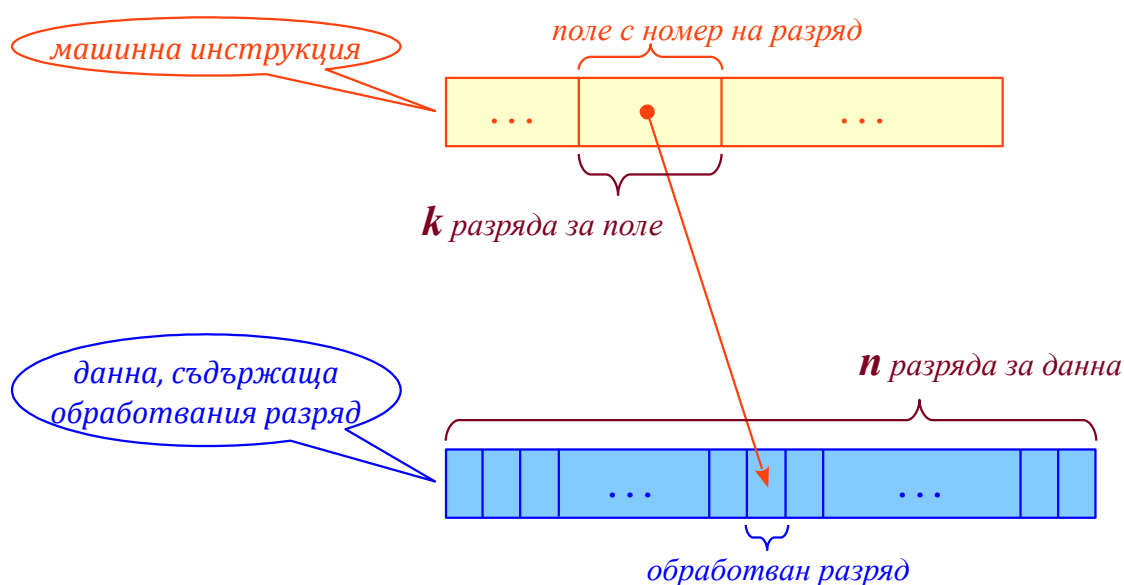
Преди десетилетия е имало експерименти с други подходи. Например процесорът Cyber 60, излязъл от употреба преди повече от 40 години, е работил с байт от 60 разряда, а целите числа са могли да заемат 16 или 24 разряда. Днес подобен подход е признат за ненужен и неудобен в масово разпространените универсални компютри.

**Трето**, кодовете на цели числа заемат *брой разряди, който е степен на двойката*. Следователно предпочитаните дължини на кодове на цели числа са 8, 16, 32, 64 и 128 разряда.

Този принцип естествено следва от стремежа към пределна простота и бързина, а едновременно с това и максимална гъвкавост, на машинните инструкции. Аргументацията на принципа е много показателна за проектирането на процесори и машинни езици.

Всеки универсален процесор трябва да поддържа инструкции за обработка на точно избрана цифра, т. е. на фиксиран разряд от кода на цяло число. Например това може да бъде необходимо в алгоритми за работа с множество, с масив от цифри, с адрес на данна или със стойността на регистъра за флагове.

Следователно в някои инструкции трябва да има *поле, назоваващо номер на разряд* в кода на цяло число.



Ако дължината на полето в инструкцията е  $k$  и дължината на данната, съдържаща обработвания разряд, е  $n$ , тогава броят на възможните стойности на полето е  $2^k$ .

За да бъде такава инструкция пределно бърза и проста, трябва всяка стойност на полето за номер на разряд да бъде валидна, за да не се налага проверка, дали съществува разряд с такъв номер, т. е. да няма разклонение в изпълнението на инструкцията, когато номерът се окаже невалиден.

Отделно искаме да не усложняваме процесора със специален режим на работа при възникване на грешка от типа „невалиден номер на разряд“. Следователно трябва броят  $2^k$  на възможните стойности на полето да бъде по-малък или равен на броя  $n$  на възможните номера на разряд.

От друга страна, за да бъде такава инструкция максимално гъвкава, тя трябва да може да назове всеки разряд в обработваната данна. Т. е. броят  $2^k$  трябва да бъде по-голям или равен на броя  $n$ .

Оттук горните две изисквания следва  $2^k = n$ .

Дължините на данна 1, 2 и 4 са неприемливи, защото са по-малки от един байт. Така се получава, че кодовете на цели числа трябва да бъдат с разрядности 8 (1 байт), 16 (2 байта), 32 (4 байта), 64 (8 байта), а при поддръжка на по-големи числа – съответно с дължини 128 (16 байта) и 256 (32 байта). Точно по тази причина всички съвременни, универсални процесори поддържат цели числа (без и със знак) в 1, 2, 4 и 8 байта. Оттам и почти всички масово използвани езици за програмиране от високо ниво също включват примитивни типове за цели числа (отново без и със знак) в 1, 2, 4 и 8 байта.

Всъщност описаната аргументация е типична за много случаи, когато в едно поле от машинна инструкция се съхранява брой на нещо – най-удобно е този брой да бъде степен на двойката. Например в архитектурата x86 обикновено са 8 или 16 регистрите, които са взаимозаменяеми в една и съща инструкция (т. е. може да бъдат назовавани в един и същ формат на инструкция).

Също от подобни разсъждения следва, че в двоичните компютри е най-удобно броят на възможните адреси в оперативната памет да бъде степен на двойката със степенен показател броя на цифрите в запис на адреса. Обаче тук имат значение и други съображения. Например процесорът Intel 8086 от фамилията x86, актуален около 1983-1990 години, е поддържал вътрешна памет от  $2^{20}$  байта, защото по онова време  $2^{16}$  е било неприемливо малко,  $2^{32}$  е било прекалено скъпо, а 20 е сума от степени



на двойката и може удобно да се изчисляват 20-разрядни адреси като сума на две 16-разрядни числа.

За 64-разрядните архитектури е типично (и обективно необходимо) физически представената памет да бъде много по-малко от възможната. Обаче и в тези машинни езици се работи с адреси с 64 разряда, а контролът, дали практически е представен назованият от инструкцията адрес, се прави извън схемата, изпълняваща и инструкцията. В този контрол типично се задейства системата за прекъсвания.

**Четвърто**, кодовете на цели числа обикновено заемат *брой байтове, който е степен на двойката*. Т. е. предпочитаните дължини на кодове на цели числа са 1, 2, 4 и 8 байта.

По същество това е следствие от предишния принцип, но може да се аргументира и отделно, защото в някои машинни инструкции искаме да има поле със стойност броя байтове, заемани от операнда, а всяка стойност на такова поле трябва да бъде валидна (аналогично на разсъжденията по-горе).

**Пето**, за всеки поддържан вид код на числа в машинния език се разпознават *варианти с различни разрядности*.

Това е необходимо за да се пишат удобно програми и за обработка на голям брой малки по абсолютна стойност числа, и за значително по-прецизни изчисления със много големи по абсолютна стойност числа

Например в машинните езици традиционно се поддържат (разпознават) варианти на ДК и на КБЗ в 1, 2, 4 и 8 байта. Съответно за КПЗ обикновено се поддържат 4-, 8- и 10-или 12-байтови кодове.

**Шесто**, всеки поддържан от машинния език код за числа позволява *представяне на числа от точно определен интервал*, зависещ от вида на кода, т. е. може да бъде кодирано всяко число от интервала и само числа от този интервал. При това, всеки такъв интервал включва нулата и за числата със знак интервалът се подбират така, че да бъде максимално симетричен спрямо нулата, което позволява най-балансирано представяне и на положителни, и на отрицателни числа.

Например 8-разрядният (еднобайтов) ДК позволява представяне точно на числата от интервала  $[-128; 127]$ , 8-разрядните ПК и ОК могат да представят числата от интервала  $[-127; 127]$ . Обаче 8-разрядният КБЗ може да представи което и да било цяло число от интервала  $[0; 255]$  и само тези числа.

Като следствие, при сумиране на числа с противоположни знакове, резултатът *винаги* може да се кодира в толкова разряда, в колкото може да се кодират и двете събираеми.

**Седмо**, стремежът е операндите относително малки цели *положителни* числа да се кодират еднакво и с КБЗ, и с ПК, и с ОК, и с ДК.

Този подход се нарушава при ИК, защото той следва запазване на наредбата на кодираните числа и не се използва като самостоятелен за операнди на аритметични инструкции.

Този принцип дава възможност при работа с малки положителни числа един и същ алгоритъм да обработва и КБЗ, и кодове със знак. Също така преминаването от работа с КБЗ към работа със знакови операнди може да става без преобразуване (прекодиране) на междинните данни от един в друг формат.

Отделно принципът позволява сумирането и изваждането на цели числа да се изпълнява с една и съща команда и за КБЗ, и за ДК.

От еднаквото кодиране на малки положителни числа в КБЗ, ПК, ОК и ДК следва, че в тези кодове старшият разряд, отразяващ знака на кодираното число, *трябва да бъде* нула за положителни числа и единица за отрицателни.

**Осмо**, при машинни изчисления с числа, *е фиксирана разрядността на резултата*.

(Обратното се допуска само в някои рядко срещани архитектури, поддържащи КДКДЦ с променлива дължина, какъвто няма в масово разпространените универсални компютри.)

Съответно, когато възниква препълване на предвидената разрядна решетка, т. е. резултатът не може да бъде верен при избраната разрядност, тогава не се променя кодът, а само се отбелязва във флаг, че е имало грешка.

Логиката тук е много естествена:

Първо, нарушаването на принципа би довело до затруднения. Ако допуснем една машинна инструкция да може да има резултат с различна разрядност, често той би трябвало да бъде само междинна данна и по нататък да се използва пак в същата инструкция. Следователно тя трябва да може да работи с операнди с различна разрядност. Това би изисквало разклоняване в действието на инструкцията за да се отчита дължината на операнда. Така ще се усложни хардуерната схема и ще се забави изпълнението, което е значителен недостатък на такъв подход. От спазването на принципа бихме извели твърде много полза.

Второ, нарушаването на принципа няма да даде съществено преимущество. Възможността за различна разрядност на резултата би била полезна главно за избягване на препълване. Обаче дължината на всяка данна е ограничена и все в някои случаи ще възниква препълване с всички последици за отчитането му.

**Девето**, при сумиране, изваждане и логически операции обикновено *операндите и резултатът се представят в кодове от един и същ вид и с еднаква разрядност*. Обратното се допуска като изключение само в машинни инструкции, които спестяват необходимостта от привеждане на операндите към еднотипен вид преди същинското изчисление.

Като следствие сумата на числа с противоположни знакове и разликата на числа с еднакви знакове *винаги* са верни.

Обаче при *умножението* традицията е да се поддържа възможност произведението да бъде с двойно по-голяма разрядност от тази на множителите, а само в малък брой изключения инструкциите допускат равна разрядност на множителите и резултата. Последните са предназначени за улеснена работа с малки по абсолютна стойност множители.

Съответно, *деленето* на цели числа без намесване на КПЗ обикновено дава два резултата – цяла част и остатък, и по традиция разрядността на делимото е двойно по-голяма от тази на делителя, на цялата част и на остатъка. При това е интересно, че понякога при делене на цели числа е възможно да бъде генерирано уникално събитие грешка „делене с нула“, когато делителят е гарантирано ненулев, просто защото, цялата част от деленето е твърде голяма и не може да се побере в предвидената разрядност, а това се интерпретира като еквивалентно на деленето с нула.

**Десето**, стремежът е да се използват *едни и същи инструкции за сумиране и изваждане* и за КБЗ, и за ДК.

Това е и една от причините да се използва ДК и да не се поддържат ПК и ОК в масово разпространените днес универсални компютри.

Основната полза от този принцип е, че може един и същ алгоритъм да се използва и за числа без знак и за знакови операнди.

Такова решение позволява и малко да се опрости машинният език, а следователно и процесорът.

(В този текст ПК и ОК се разглеждат просто като илюстрация на идеи за представяне на цели числа в паметта на цифров компютър, които са съществували исторически, но са били съществено доразвити с въвеждането на ДК.)

**Единадесето**, често в процесорите се залага *минимална поддръжка на КДКДЦ за цели числа*, за да се подпомогне възможността за пълноценна работа с подобни кодове в езиците за програмиране от високо ниво (в частност и работата с дробни може да се реализира чрез обработки с цели числа). Обаче не се разпознават аналогични кодове на дробни числа, защото това би било съществено по-сложно без да дава осезаеми преимущества.

## 17. Примери за получаване на кодове на цели числа

### Задача 1

Да се намерят двоичните записи на ПК, ОК, ДК и ИК в 7 разряда на 26 .

#### Примерно решение

Първо получаваме двоичния запис на 26 :

$$\begin{array}{r} 26:2 \\ 13 \overline{)0} \\ 6 \overline{)1} \\ 3 \overline{)0} \\ 1 \overline{)1} \\ 0 \overline{)1} \end{array}$$

Откъдето  $26_{(10)} = 11010_{(2)}$ .

ПК, ОК и ДК на положително число, когато съществуват, са равни на самото число (то се кодира чрез себе си).

Следователно  $ПК_7(+26) = ОК_7(+26) = ДК_7(+26) = 001\ 1010_{(2)}$ .

ИК търсим по формулата  $ИК_7(+26) = 2^{7-1} + (+26)$ . Тъй като  $2^6 = 1000000_{(2)}$ , може да изчисляваме удобно в двоична ПБС:

$$\begin{array}{r} +\ 1000000 \\ \underline{111010} \\ 1111010 \end{array}$$

Следователно  $ИК_7(+26) = 1111010_{(2)}$ .

**Задача 2**

Да се намерят двоичните записи на ПК, ОК, ДК и ИК на  $-18$  в 9 и в 11 разряда.

**Примерно решение**

Първо получаваме двоичния запис на  $+18$  :

$$\begin{array}{r} 18:2 \\ 9 \overline{) 0} \\ 4 \overline{) 1} \\ 2 \overline{) 0} \\ 1 \overline{) 0} \\ 0 \overline{) 1} \end{array}$$

Откъдето  $18_{(10)} = 10010_{(2)}$ .

ПК започва с единица, заради знака минус, и продължава с абсолютната стойност в останалите 8 или 10 разряда.

Следователно

$$\text{ПК}_9(-18) = 1\,0001\,0010_{(2)} \text{ и}$$

$$\text{ПК}_{11}(-18) = 100\,0001\,0010_{(2)}.$$

(В правилния ПК знаковият, т. е. старшият, разряд е единица, както е показано с червен цвят във формулите  $1\,0001\,0010_{(2)}$  и  $100\,0001\,0010_{(2)}$ .)

За да получим ОК поразрядно инвертираме *абсолютната стойност*, записана с 9 или 11 цифри:

$$\begin{array}{ccc} 000010010 & & 00000010010 \\ \downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow & \text{и} & \downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow \\ 111101101 & & 11111101101 \end{array}$$

Следователно

$$\text{ОК}_9(-18) = 1\,1110\,1101_{(2)} \text{ и}$$

$$\text{ОК}_{11}(-18) = 111\,1110\,1101_{(2)}.$$

(В правилния ОК знаковият разряд също е единица, както е показано с червен цвят във формулите  $1\,1110\,1101_{(2)}$  и  $111\,1110\,1101_{(2)}$ .)

За да получим ДК поразрядно инвертираме *абсолютната стойност*, записана с 9 цифри:

$$\begin{array}{r} 000010010 \quad 00000010010 \\ \downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow \text{ и } \downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow \\ 111101101 \quad 11111101101 \end{array}$$

и към полученото прибавяме единица:

$$\begin{array}{r} 111101101 \quad 11111101101 \\ +1 \text{ и } +1 \\ \hline 111101110 \quad 11111101110 \end{array}$$

Следователно

$$\text{ДК}_9(+26) = 111101110_{(2)} \text{ и}$$

$$\text{ДК}_{11}(+26) = 11111101110_{(2)}.$$

(В правилния ДК знаковият разряд също е единица, както е показано с червен цвят във формулите  $\mathbf{1}11101110_{(2)}$  и  $\mathbf{1}1111101110_{(2)}$ .)

ИК търсим по формулите  $\text{ИК}_9(-18) = 2^{9-1} + (-18)$  и  $\text{ИК}_{11}(-18) = 2^{11-1} + (-18)$ .

Тъй като  $2^8 = 100000000_{(2)}$  и  $2^{10} = 10000000000_{(2)}$ , удобно е да изчисляваме в двоична ПБС:

$$\begin{array}{r} \underline{\quad 100000000} \quad \underline{\quad 10000000000} \\ \quad 10010 \text{ и } \quad 10010 \\ \hline \text{????????} \quad \text{????????????} \end{array}$$

При това възникват заеми:

$$\begin{array}{r} \underline{\quad \dot{1}\dot{0}\dot{0}\dot{0}\dot{0}\dot{0}\dot{0}\dot{0}} \quad \underline{\quad \dot{1}\dot{0}\dot{0}\dot{0}\dot{0}\dot{0}\dot{0}\dot{0}\dot{0}\dot{0}\dot{0}} \\ \quad 10010 \text{ и } \quad 10010 \\ \hline \text{????????} \quad \text{????????????} \end{array}$$

След като бъдат заети всички заеми в действителност се извършва изваждане:

$$\begin{array}{r} \underline{\quad 011111120} \quad \underline{\quad 01111111120} \\ \quad 10010 \text{ и } \quad 10010 \\ \hline 011101110 \quad 01111101110 \end{array}$$

Следователно

$$\text{ИК}_9(-18) = 011101110_{(2)} \text{ и}$$

$$\text{ИК}_9(-18) = 01111101110_{(2)}.$$

(В правилния ИК старшият разряд, за който *не* е прието да се нарича знаков, е нула за отрицателни кодирани числа, както е показано с червен цвят във формулите  $\mathbf{0}11101110_{(2)}$  и  $\mathbf{0}1111101110_{(2)}$ .)

### Задача 3

Да се намерят двоичните записи на ПК, ОК, ДК и ИК в 11 разряда на  $-374$ .

#### Примерно решение

Първо получаваме двоичния запис на  $+374$ :

$$\underline{374:2}$$

187	0
93	1
46	1
23	0
11	1
5	1
2	1
1	0
0	1

Оттук  $374_{(10)} = 101110110_{(2)}$ .

ПК започва с единица, заради знака минус, и в останалите 10 разряда продължава с абсолютната стойност на кодираното число.

Следователно  $\text{ПК}_{11}(-374) = 10101110110_{(2)}$ .

(В правилния ПК знаковият, т. е. старшият, разряд е единица, както е показано с червен цвят във формулата  $\mathbf{1}0101110110_{(2)}$ .)

За да получим ОК поразрядно инвертираме *абсолютната стойност*, записана с 11 цифри:

$$\begin{array}{r} 00101110110 \\ \downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow \\ 11010001001 \end{array}$$

$$\text{Следователно } \text{ОК}_{11}(-374) = 11010001001_{(2)}.$$

(В правилния ОК знаковият разряд също е единица, както е показано с червен цвят във формулата  $\mathbf{1}1010001001_{(2)}$ .)

За да получим ДК поразрядно инвертираме *абсолютната стойност*, записана с 11 цифри:

$$\begin{array}{r} 00101110110 \\ \downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow \\ 11010001001 \end{array}$$

и към полученото прибавяме единица:

$$\begin{array}{r} 11010001001 \\ +1 \\ \hline 11010001010 \end{array}$$

$$\text{Следователно } \text{ДК}_{11}(-374) = 11010001010_{(2)}.$$

(В правилния ДК знаковият разряд също е единица, както е показано с червен цвят във формулата  $\mathbf{1}1010001010_{(2)}$ .)

$$\text{ИК търсим по формулата } \text{ИК}_{11}(-374) = 2^{11-1} + (-374).$$

$$2^{10} = 10000000000_{(2)} \text{ и е удобно да изчисляваме в двоична ПБС:}$$

$$\begin{array}{r} -10000000000 \\ 101110110 \\ \hline \text{????????????} \end{array}$$

При това възникват заеми:

$$\begin{array}{r} -\overset{\cdot}{1}\overset{\cdot}{0}\overset{\cdot}{0}\overset{\cdot}{0}\overset{\cdot}{0}\overset{\cdot}{0}\overset{\cdot}{0}\overset{\cdot}{0}\overset{\cdot}{0}\overset{\cdot}{0}\overset{\cdot}{0} \\ 101110110 \\ \hline \text{????????????} \end{array}$$



След като бъдат заети всички заеми в действителност се извършва следното изваждане:

$$\begin{array}{r} \text{— } 0111111120 \\ \quad 101110110 \\ \hline 01010001010 \end{array}$$

$$\text{Следователно } \text{ИК}_{11}(-374) = 01010001010_{(2)}.$$

(В правилния ИК старшият разряд, за него *не* е прието да се нарича знаков, е нула за отрицателни кодирани числа, както е показано с червен цвят във формулата  $\text{0}1010001010_{(2)}$ .)

#### Задача 4

Да се намерят двоичните записи на ПК, ОК, ДК и ИК в 10 разряда на  $-96$ .

#### Примерно решение

Първо получаваме двоичния запис на  $+96$ :

$$\begin{array}{r} 96:2 \\ 48|0 \\ 24|0 \\ 12|0 \\ 6|0 \\ 3|0 \\ 1|1 \\ 0|1 \end{array}$$

$$\text{Оттук } 96_{(10)} = 110\,0000_{(2)}.$$

ПК започва с единица, заради знака минус, и продължава с абсолютната стойност в останалите 9 разряда.

$$\text{Следователно } \text{ПК}_{10}(-96) = 100110\,0000_{(2)}.$$

(В правилния ПК знаковият разряд е единица, както е показано с червен цвят във формулата  $\text{1}00110\,0000_{(2)}$ .)

За получаването на ОК поразрядно инвертираме *абсолютната стойност*, записана с 10 цифри.

$$\begin{array}{r}
 0001100000 \\
 \downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow \\
 1110011111
 \end{array}$$

Следователно  $OK_{10}(-96) = 1110011111_{(2)}$ .

(В правилния ОК знаковият разряд също е единица, както е показано с червен цвят във формулата  $\mathbf{1}110011111_{(2)}$ .)

За да получим ДК поразрядно инвертираме *абсолютната стойност*, записана с 10 цифри:

$$\begin{array}{r}
 0001100000 \\
 \downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow \\
 1110011111
 \end{array}$$

и към полученото прибавяме единица:

$$\begin{array}{r}
 1110011111 \\
 +1 \\
 \hline
 1110100000
 \end{array}$$

Следователно  $DK_{10}(-96) = 1110100000_{(2)}$ .

(В правилния ДК знаковият разряд също е единица, както е показано с червен цвят във формулата  $\mathbf{1}110100000_{(2)}$ .)

ИК търсим по формулата  $IK_{10}(-96) = 2^{10-1} + (-96)$ .

$2^9 = 1000000000_{(2)}$  и е удобно да изчисляваме в двоична ПБС:

$$\begin{array}{r}
 -1000000000 \\
 \underline{1100000} \\
 \text{??????????}
 \end{array}$$

При това възникват заеми:

$$\begin{array}{r}
 -\overset{\cdot}{1}\overset{\cdot}{0}\overset{\cdot}{0}\overset{\cdot}{0}000000 \\
 \underline{1100000} \\
 \text{??????????}
 \end{array}$$

След като бъдат заети всички заеми в действителност извършваме следното изваждане:

$$\begin{array}{r} \text{— } 0111200000 \\ 1100000 \\ \hline 0110100000 \end{array}$$

$$\text{Следователно } \text{ИК}_{10}(-96) = 0110100000_{(2)}.$$

(В правилния ИК старшият разряд е нула за отрицателни кодирани числа, както е показано с червен цвят във формулата  $0110100000_{(2)}$ .)

### Задача 5

В програма на езика C, или подобен на него, с 4-байтов тип `int` се съдържа декларацията:

```
int h = -648;
```

Какъв ще бъде 16-ичният запис на кода на променливата `h`, записан в паметта при инициализацията?

### Примерно решение

Щом променливата е от 4-байтов тип `int`, значи търсим 16-ичния запис на  $\text{ДК}_{32}(-648)$  и е най-удобно да го получим от двоичния запис на същия ДК.

Бихме могли да работим с по-малко от 32 двоични цифри, като използваме свойствата на ДК на едно и също число с различна разрядност. Обаче за по-голяма нагледност все пак ще получим 32-цифрения двоичен запис на  $\text{ДК}_{32}(-648)$ .

За тази цел първо получаваме двоичния запис на  $+648$  :

$$\underline{648:2}$$

324	0
162	0
81	0
40	1
20	0
10	0
5	0
2	1
1	0
0	1

Оттук  $648_{(10)} = 10\,1000\,1000_{(2)}$ .

За да получим ДК правим поразрядно инвертиране на *абсолютната стойност*, записана с 32 цифри:

$$\begin{array}{r}
 \overbrace{0 \dots 0}^{22 \text{ нули}} 1010001000 \\
 \downarrow \dots \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\
 \underbrace{1 \dots 1}_{22 \text{ единици}} 0101110111
 \end{array}$$

След това към полученото число прибавяме единица:

$$\begin{array}{r}
 \overbrace{1 \dots 1}^{22 \text{ единици}} 0101110111 \\
 \qquad \qquad \qquad + 1 \\
 \hline
 \underbrace{1 \dots 1}_{22 \text{ единици}} 0101111000
 \end{array}$$

Следователно  $\text{ДК}_{32}(-648) = \underbrace{1 \dots 1}_{22 \text{ единици}} 01\,0111\,1000_{(2)}$ .

За да получим 16-ичния запис (вижте следващата формула) разделяме 2-ичния запис на групи по 4 цифри и всяка такава четворка замества с една 16-ична цифра, така че стойността на 4-цифрения двоичен запис, съставляващ групата, да бъде равна на стойността на 16-ичната цифра:

1111	1111	1111	1111	1111	1101	0111	1000 <sub>(2)</sub>
F	F	F	F	F	D	7	8 <sub>(16)</sub>

Следователно  $ДК_{32}(-648) = FFFFD78_{(16)}$  и това е търсения 16-ичен запис.

## 18. Примери за намиране на кодирано цяло число по известен код

### Задача 6

Даден е 5-разрядният код  $01100_{(2)}$ .

Да се намери десетичният запис на кодираното число, когато този код се интерпретира като ПК, като ОК, като ДК и като ИК.

#### Примерно решение

Тъй като старшият разряд в кода е нула, когато това е ПК, ОК или ДК, кодираното число ще бъде положително и равно на кода си. Щом  $01100_{(2)} = 12_{(10)}$ , то кодираното число е  $12_{(10)}$ .

$$\text{Т. е. } 01100_{(2)} = ПК_5(+12_{(10)}) = ОК_5(+12_{(10)}) = ДК_5(+12_{(10)}).$$

При интерпретация на кода като изместен използваме формулата за ИК и решаваме полученото уравнение:

$$12_{(10)} = 01100_{(2)} = ИК_5(x) = 2^4 + x = 16_{(10)} + x$$

Следователно кодираното число е  $x = -4$ . Т. е.  $01100_{(2)} = ИК_5(-4)$ .

### Задача 7

Даден е 6-разрядният код  $110100_{(2)}$ .

Да се намери десетичният запис на кодираното число, когато този код се интерпретира като ПК, ОК, ДК и ИК.

#### Примерно решение

Тъй като старшият разряд в кода е единица, когато това е ПК, ОК или ДК, кодираното число ще бъде отрицателно и ще се получава по различен начин при тези три интерпретации.

Когато това е ПК, абсолютната стойност на кодираното число се записва двоично с цифрите след знаковия разряд, т. е.  $10100_{(2)}$ , което е равно на  $20_{(10)}$ . Следователно търсеният 10-ичен запис на кодираното число е  $-20_{(10)}$ .

$$\text{Т. е. } 110100_{(2)} = \text{ПК}_6(-20_{(10)}).$$

Когато това е ОК, той е получен чрез поразрядно инвертиране на двоичния запис на абсолютната стойност на кодираното число. За да получим тази абсолютна стойност наново инвертираме:

$$\begin{array}{r} 110100 \\ \downarrow\downarrow\downarrow\downarrow\downarrow\downarrow \\ 001011 \end{array}$$

Тъй като  $001011_{(2)} = 11_{(10)}$ , то търсеният 10-ичен запис на кодираното число е  $-11_{(10)}$ .

$$\text{Т. е. } 110100_{(2)} = \text{ОК}_6(-11_{(10)}).$$

Когато това е ДК, той е получен чрез прибавяне на единица към резултата от поразрядното инвертиране на двоичния запис на абсолютната стойност на кодираното число. За да получим тази абсолютна стойност извършваме обратното преобразование:

Изваждаме единица:

$$\begin{array}{r} \text{— } 110\overset{\cdot}{1}\overset{\cdot}{0}0 \\ \hline 1 \\ \hline \text{?????} \end{array}$$

След заемите всъщност изпълняваме изваждането:

$$\begin{array}{r} \text{— } 110012 \\ \hline 1 \\ \hline 110011 \end{array}$$

Получената разлика инвертираме поразрядно:

$$\begin{array}{r} 110011 \\ \downarrow\downarrow\downarrow\downarrow\downarrow\downarrow \\ 001100 \end{array}$$

Тъй като  $001100_{(2)} = 12_{(10)}$ , то търсеният 10-ичен запис на кодираното число е  $-12_{(10)}$ .

$$\text{Т. е. } 110100_{(2)} = \text{ДК}_6(-12_{(10)}).$$

Когато кода  $110100_{(2)}$  интерпретираме като изместен, прилагаме формулата за ИК:

$$52_{(10)} = 110100_{(2)} = \text{ИК}_6(x) = 2^5 + x = 32_{(10)} + x$$

Следователно търсеният десетичен запис на кодираното число е  $x = 20_{(10)}$ .

$$\text{Т. е. } 110100_{(2)} = \text{ИК}_6(+20_{(10)}).$$

## 19. Въведение в представянето на дробни числа

За представяне на дробни числа на практика са използвани различни формати, а теоретично са възможни още повече видове. В момента са актуални поне четири формата: формат с фиксирана запетая, формат с естествена запетая, формат на числата от типа Decimal в езика C# и в платформата .NET (който няма пряка поддръжка чрез нито една машинна инструкция) и формат с плаваща запетая.

КПЗ се използва на машинно ниво в масово разпространените универсални компютри и традиционно се поддържа в езиците за програмиране от всички нива.

Форматът на типа Decimal навлиза все по-широко в употреба заради разпространението на .NET и възможностите, които предоставя за изчисления с повишена точност.

При форматите за дробни числа типично се използва идеята – нещо за кодираното число да не се съхранява в паметта, заделена за кода му, а да се подразбира в машинните инструкции, които работят със съответните кодове. Например, както е описано по-надолу, във формата с фиксирана запетая не се заделя място за описание на положението на дробната запетая, а във формата с плаваща запетая няма поле за съхраняване на основата на степента от експоненциалното представяне.

За практически всички представяния на дробни числа е характерно, че не се поддържат периодични записи, защото периодите без да допринесат съществена полза значително биха усложнили и алгоритмите за изчисления, и съответните хардуерни схеми. (Аритметиката с периодични записи е качествено различна от работата с неперидични записи, а закръгляния все пак ще се наложат, защото всяко смислено ограничение на броя разряди за описанието на период би попречило да се кодират някои много дълги периоди, възникващи в практическите изчисления.)

Различните представяния на дробни числа имат различни характеристики и са удобни за различни цели. Обикновено особено важна е точността на изчисленията. В зависимост от бройната система, на която се основават изчисленията, и от използвания формат за машинно представяне на числата се постига качествено различна точност. Тя допълнително може да бъде повишена чрез привличане на специални числени методи.

По-нататък по-подробно ще разгледаме КПЗ, а останалите формати само ще опишем кратко.

## 20. Формат с фиксирана запетая

В КФЗ се съхраняват знак на кодираното число и точно определен брой цифри от записа му. Мястото на дробната запетая се подразбира винаги едно и също във всички кодове от един и същ вид.

Очевидно в един и същ вид КФЗ при  $n$  цифри от цялата част и  $k$  цифри от дробната част може да се представят само числа със запис  $\pm \overline{a_{n-1} \dots a_0, c_1 \dots c_k}_{(s)}$ .  
Например при  $s=2$  може да бъдат кодирани точно (без закръгляне) само числата  $\pm \overline{0 \dots 0, 0 \dots 00}_{(2)}$ ,  $\pm \overline{0 \dots 0, 0 \dots 01}_{(2)}$ , ...,  $\pm \overline{1 \dots 1, 1 \dots 11}_{(2)}$ .

Съответно абсолютната грешка (модула на разликата между кодираното число и неговия код) в представянето на което и да било число е най-много  $\frac{1}{2} \cdot \frac{1}{s^k}$ .

Разгледаните по-горе ПК, ОК, ДК и ИК за представяне на цяло число със знак може да се интерпретират и като формати с фиксирана запетая, мястото на която се подразбира винаги точно след последната цифра. Затова разликите в начина на пресмятания с ПК, ОК, ДК и ИК показват, че аритметиката за числа във формат с фиксирана запетая съществено зависи от начина на описанието на цифрите на кодираното число.

## 21. Формат с естествена запетая

В КЕЗ се съхраняват знакът и определен брой цифри и има отделно поле за мястото на дробната запетая, за което обаче са разрешени само стойности от нула до броя на съхраняваните цифри минус единица. Т. е. задава се място на запетаята само непосредствено след някоя съхранявана цифра.

Например в КЕЗ с поле за четири двоични цифри (тогава са необходими точно два разряда за задаване на позицията на дробната запетая) може да се кодират само числата:

$$\begin{aligned} &\pm \overline{a_1, a_2 a_3 a_4}_{(2)}; \pm \overline{a_1 a_2, a_3 a_4}_{(2)}; \\ &\pm \overline{a_1 a_2 a_3, a_4}_{(2)}; \pm \overline{a_1 a_2 a_3 a_4}_{(2)}. \end{aligned}$$

При този подход някои числа имат по няколко кода.

Например

$$\begin{aligned} -0,100_{(2)} &= -00,10_{(2)} = -000,1_{(2)} \text{ или} \\ +1,100_{(2)} &= +01,10_{(2)} = +001,1_{(2)}. \end{aligned}$$



Такъв формат типично се използва в калкулаторите.

Представянето с естествена запетая може да бъде тълкувано и като особен вариант на формат с плаваща запетая при съответни ограничения за експоненциалното представяне (описано по-надолу в точка 23).

За изчисленията с КЕЗ е удобно всяка стойност на полето за позиция на запетаята да бъде действително възможна. Например при формат с  $k$  двоични разряда на *позицията на запетаята* броят на съхраняваните *цифри от кодираното число* да бъде  $2^k$ . С отчитане на допълнителен разряд за знак на кодираното число се получава, че такъв код изисква поне  $1 + k + 2^k$  разряда. Обаче това трудно се съвместява с подравняването на кода по границите на 8-разрядни байтове, особено когато броят на байтовете трябва да бъде степен на двойката.

Също се вижда, че в КЕЗ броят на разрядите на полето за позиция на дробната запетая е много малък в сравнение с броя на разрядите в полето за цифри.

## 22. Формат на типа *Decimal* в езика *C#* и в платформата *.NET*

Форматът на типа *Decimal* в езика *C#* и в платформата *.NET* може да бъде интерпретиран като по-особен представител и на формат с естествена запетая, и на формата с плаваща запетая.

Тук числото се представя във вида  $\zeta m \cdot 10^k$ , където:

$\zeta$  е знак „+“ (кодиран с 0) или „−“ (кодиран с 1), който можем да разглеждаме съответно като множител  $+1$  или  $-1$ ;

$m$  е цяло число от интервала  $0 \leq m < 2^{96}$ ;

$k$  е цяло число от интервала  $-28 \leq k \leq 0$ .

Съответно в паметта се съхраняват кодът на  $\zeta$  и двоичните записи на  $m$  и на абсолютната стойност на  $k$ .

Интерпретацията като формат с естествена запетая е възможна, тъй като десетичният запис на  $m$  е с 28 или 29 цифри и в него позицията на запетаята, зададена чрез  $k$ , е отдясно на една от тези десетични цифри.

Интерпретацията като формат с плаваща запетая се основава на представянето, което може да се разглежда като експоненциално, евентуално с уговорки. (Понятието експоненциално представяне на число понякога се поднася в широк смисъл – просто като произведение на знакова мантиса и степен, без да се отчита съответствието между запис на мантисата и нейното представяне в паметта.)

Двоично кодът от тип *Decimal* се записва със 128 цифри (съответно заема 16 байта в паметта), където отляво надясно се редуват:

- знаков разряд (старшият) за  $\zeta$ ;
- десет нули;

- пет разряда с двоичния запис на  $|k| = -k$ ;
- още 16 нули;
- 96 разряда (младшите) с двоичния запис на  $m$ .

Например числото  $-1.10^{-28}$  се записва така (цифрите са групирани по осморки и са подредени от старша към младша съответно отляво надясно във всеки ред, а цифрите от един и същ ред са старши, спрямо всички цифри в следващите редове):

```
10000000 00011100 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000001
```

Съответно максималната възможна стойност  $+(2^{96}-1).10^0$  се представя така:

```
00000000 00000000 00000000 00000000
11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111
```

Този формат, за разлика от формата с плаваща запетая, *не предвижда кодове* за минус нула, минус безкрайност, плюс безкрайност и не числа (NaN).

Абсолютната грешка, т. е. абсолютната стойност на разликата между кода и кодираното число, зависи от степенния показател  $k$  в представянето  $\zeta m.10^k$  и е по-малка от  $\frac{10^k}{2}$ . Тъй като  $-28 \leq k \leq 0$ , тази грешка остава в интервала

$$\left[ \frac{1}{2 \cdot 10^{28}}, \frac{1}{2} \right].$$

Най-важното преимущество на типа Decimal е, че поддържа поне 28 достоверни цифри от десетичния запис на кодираното число. Така се представят точно огромен брой десетични дроби, които имат периодичен двоичен запис и следователно ще се закръглят, ще се представят неточно с машинни кодове, основаващи се на двоичния запис на кодираното число, вместо на десетичния. Т. е. твърде много изчисления, които биха дали грешка при използване на КПЗ, ще дават точен резултат при работа с типа Decimal.

Съответно работата с приближени стойности ще дава по-малки грешки при използване на Decimal, вместо Double. Съвсем отделно, точността на изчисленията със закръглени числа допълнително може да се повиши чрез целенасочено прилагане на числени методи.

Затова в програмирането с езика C# и в .NET платформата за икономически и за по-прецизни научни изчисления традиционно се предпочита типът Decimal, вместо Double.

### 23. Видове кодове с плаваща запетая

Всички КПЗ се основават на представянето на кодираното число в така наречения **експоненциален вид**:

$$A = \zeta m \cdot N^k$$

Тук  $\zeta$  е знак  $+$  (кодиран с цифра 0 в КПЗ) или  $-$  (кодиран с цифра 1 в КПЗ), който заради математическата формула можем да разглеждаме съответно като множител  $+1$  или  $-1$ ,  $m$  е неотрицателно число с краен запис, наричано **мантиса**,  $N$  е цяло число, по-голямо от единица,  $k$  е цяло число със знак, наричано **порядък**.

Обикновено в експоненциалното представяне в мантисата се записват точно толкова цифри, колкото се съхраняват в паметта.

Всеки КПЗ се опира на точно съответствие между начина на записване на  $m$  в експоненциалното представяне и кодирането на цифрите на  $m$  в паметта. Например в масово разпространените процесори се използват записи от вида  $\zeta \overline{a_0, a_1 \dots a_{n(2)}} \cdot 2^k$  и се предполага двоична памет.

В паметта (във вид на редица от цифри) в три полета се съхраняват *само* кодовете за представяне на знака  $\zeta$ , на цифрите на мантисата  $m$  и на стойността на порядъка  $k$ :



Потенциално са възможни и на практика са използвани и други наредби на трите полета, но горната е най-удобна.

Основата на степента  $N$  не се съхранява в паметта, а се *подразбира* в алгоритмите за работа с КПЗ, т. е. само при точно определена основа  $N$  резултатите от изчисленията ще бъдат желаните, и съответно ще са верни, когато е възможно за предвидените разрядности.

Например в архитектурата x86 след инициализацията „float  $F=25.625$ “ в езика C++ съдържанието на четирите байта от паметта, заделени за стойността на  $F$ , ще бъде:



Този КПЗ съответства на експоненциалното представяне

$$F = +1,100110100000000000000000_{(2)} \cdot 2^4,$$

като първата цифра на мантисата не се съхранява в паметта, а се подразбира, заради което мантисата съдържа една цифра (последната нула) повече, отколкото е разрядността на полето за мантиса, а в полето за порядък се съхранява  $ИК_8(4) = 10000011_{(2)}$ .

Всъщност форматът с плаваща запетая допуска много голям брой различни варианти, образуващи обширно множество от различни КПЗ. Например възможни източници на различия са:

- подразбиращата се основа  $N$  в експоненциалното представяне може да бъде всяко цяло число, различно от нула и единица, но се използват най-често основи 2, 16 и 10;
- ПБС, в която се записва мантисата  $m$ , може да бъде различна от тази, на която се базира паметта;
- порядъкът може да се представя в различни формати в съответното поле от кода;
- първата цифра от  $m$  може да не се съхранява, когато тя винаги се подразбира единица;
- местоположението на дробната запетая в мантисата може да се подразбира във всяка позиция спрямо цифрите, но най-често е или преди, или след старшата цифра;
- дължините на полетата (броеве на разрядите в тях) за порядък и за мантиса могат да бъдат различни, но типично се поддържат КПЗ в 4, 8, 10 и 12 байта.

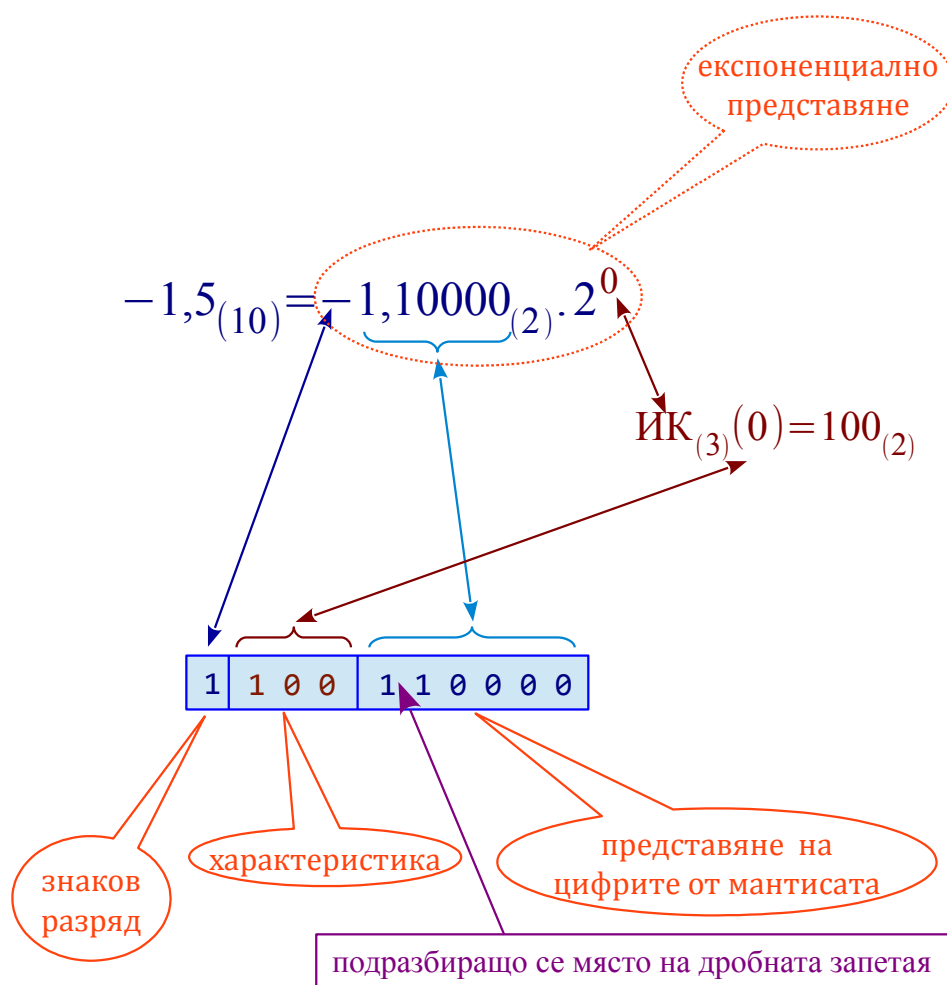
Когато порядъкът се представя чрез ИК, този ИК наричаме **характеристика**.

Когато първата цифра в мантисата е различна от нула, такъв КПЗ наричаме **нормализиран**. Съответно се приема, че нормализиран КПЗ на нулата е този на плюс нула.

Очевидно всеки КПЗ, както и всяко експоненциално представяне, се получава без закръгляне от едно единствено рационално число, а със закръгляне – от безброй реални числа.

В задачите и примерите по-нататък ще разглеждаме само КПЗ от вида  $\pm \overline{a_0, a_1 \dots a_n}_{(2)} \cdot 2^k$ , в които  $a_0$  се съхранява в паметта, а порядъкът се представя чрез изместен код (характеристика). Това достатъчно ясно показва и общите възможности на КПЗ. Обаче ще привеждаме примери с *необичайни* разрядности, каквито няма в практиката, защото свойствата на КПЗ и начинът за работа с тях са аналогични при всякакви разрядности.

Например при  $\text{КПЗ}_{3,6}(-1,5_{(10)}) = 1\ 100\ 110000_{(2)}$  с 3 разряда за



В тази схема се подразбират още основа 2 на ПБС, върху която се базира паметта, и основа 2 на степента в експоненциалното представяне на кодираното число.

Всяко число може да бъде закръглено до какъвто и да било брой цифри, подходящ за която и да било **разрядност на полето за мантиса**. Следователно това поле влияе само на броя на съхраняваните цифри, т. е. – на точността на представянето на кодираното число чрез КПЗ.

Обаче при фиксирана **разрядност на полето за порядък**, както и да се кодира той, може да се представят само ограничен брой цели числа (степенни показатели), а това ограничава обхвата на експоненциалните представяния, за които съществува подходящ КПЗ. При това, невъзможността да се кодира порядъка в предвиденото за него поле се проявява в два качествено различни варианта:

Първо, когато порядъкът е много малко отрицателно число, това означава, че кодираното число е много близко до нулата. Следователно то може приближено да се опише с нула и именно така се кодира. В този случай казваме, че възниква **антипрепълване** на КПЗ.

Второ, когато порядъкът е много голям и не се побира в предвиденото за него поле, това означава, че експоненциалното представяне съответствува на кодирано число с много голяма абсолютна стойност. За представянето на такива числа се въвеждат два специални КПЗ, интерпретирани като **минус безкрайност** и **плюс безкрайност**. Така става възможно да бъде съпоставен някакъв КПЗ на буквално всяко реално число. Затова понякога за формата с плаваща запетая се избягва терминът „препълване“.

Освен горните КПЗ се допуска и код за минус нула и отделно се предвижда интерпретация на някои КПЗ като неправилни. Те се наричат кодове за **нечисла** и традиционно се означават с **NaN**, което произхожда от „not a number“. Появата на NaN в изчисленията е сигнал за възникване на грешка в самия смисъл на математическите операции.

## 24. Точност на представянето на числа при работа с КПЗ

Точността на каквото и да било представяне на дробно число естествено се асоциира с броя на достоверно известните цифри от съответния запис на числото. За формата с плаваща запетая това е броят на известните цифри от мантисата, който пък е следствие от дължината на полето за мантиса.

За точността на КПЗ е показателна абсолютната стойност на разликата между кодираното число и неговия КПЗ, наричана **абсолютна грешка на конкретния КПЗ**. Тя зависи от порядъка. При това, увеличаването на порядъка с единица в някои случаи води до увеличаване на грешката *два пъти*.

Например при закръгляне надолу, когато

$$A = \zeta \overline{a_0, a_1 \dots a_{n-2} 0 1 a_{n+1} \dots}_{(2)} \cdot 2^k \text{ и в КПЗ на } A \text{ са представени } n-1$$

цифри, т. е. закръгля се до  $a_{n-2}$  включително, тогава абсолютната грешка на кода е

$$\overline{0, \underbrace{0 \dots 0}_{n-1} 1 a_{n+1} \dots}_{(2)} \cdot 2^k \geq \overline{0, \underbrace{0 \dots 0}_{n-1} 1}_{(2)} \cdot 2^k = \frac{1}{2^n} \cdot 2^k = 2^{k-n}.$$

Или например при закръгляне нагоре, когато  $A = \zeta \overline{a_0, a_1 \dots a_{n-2} 1 0 1 a_{n+2} \dots}_{(2)} \cdot 2^k$  и в КПЗ на  $A$  са представени  $n-1$  цифри, тогава абсолютната грешка на кода е

$$\begin{aligned} & \left( 0, \underbrace{0 \dots 0}_{n-3} 1_{(2)} - \overline{0, \underbrace{0 \dots 0}_{n-2} 1 0 1 a_{n+2} \dots}_{(2)} \right) \cdot 2^k = \\ & = 0, \underbrace{0 \dots 0}_{n-1} 1 \dots_{(2)} \cdot 2^k \geq \frac{1}{2^n} \cdot 2^k = 2^{k-n}. \end{aligned}$$

В горните два примера при порядък  $k$  абсолютна грешка на КПЗ на  $A$  е по-голяма от  $2^{k-n}$ .

$$\text{Тъй като } 1024 = 2^{10} > 10^3 = 1000, \text{ откъдето } 2^{k-n} > 10^{3 \cdot \frac{k-n}{10}}.$$

$$\text{Съответно за НКПЗ } |A| = \overline{1, a_1 \dots a_n \dots}_{(2)} \cdot 2^k \geq 2^k > 10^{3 \cdot \frac{k}{10}}.$$

Следователно при  $|A| > 10^h$  абсолютната грешка на НКПЗ, представящ  $n-1$  цифри, може да бъде по-голяма от  $10^{h - \frac{3 \cdot n}{10}}$ .

Това е огромно число.

Например при често използвания 8-байтов КПЗ от типа, наричан double в езиките C++, Java и други, в КПЗ има 52-разрядно поле за мантиса, което представя 53-цифрена мантиса (кодът се съхранява в нормализиран вид в оперативната памет и първата единица от мантисата се подразбира без да се записва в паметта). Съответно се получава, че при  $|A| > 10^h$  абсолютната грешка на КПЗ може да бъде по-голяма от

$$10^{h - \frac{3 \cdot 53}{10}} = 10^{h - \frac{159}{10}} > 10^{h-16}.$$

Оттук, тъй като максималната стойност от тип double е над  $10^{308}$ , то максималната абсолютна грешка на КПЗ от тип double може да бъде над  $10^{292}$ .

В практическите изчисления грешката допълнително намалява около  $10^4$  пъти, когато вътре в процесора, в блока за работа с КПЗ, се използват 10-байтови КПЗ, или около  $10^8$  пъти, когато в процесора се използват 12-байтови КПЗ. Но все пак абсолютната грешка на кода остава огромна.

Горните оценки показват, че КПЗ е *неприемлив за прецизни изчисления с много големи по абсолютна стойност кодирани числа*.

Съответно за работа с много по-висока точност за кодирани числа с абсолютни стойности под  $2^{96}$  може да се използва типът Decimal в C# и .NET. Обаче над границата  $2^{96}$  за точни пресмятания ще потрѣбват или други машинни представяния на числа, или специални числени методи, или, най-вероятно, и едното, и другото.

От гледна точка на прецизността на изчисленията специално внимание заслужава изваждането на много близки стойности. Това се вижда от следния пример:

$$\begin{array}{r} 1,01101011 \cdot 2^k \\ - 1,01101010 \cdot 2^k \\ \hline 0,00000001 \cdot 2^k \end{array}$$

Независимо дали по-нататък резултатът ще бъде нормализиран, в мантисата вече е останала *само една достоверна цифра*, което е значителна загуба на точност.

За минимизирането на такъв проблем при сумирането на няколко КПЗ се прилагат алгоритми с разместване на събираемите.

## 25. Някои следствия от нормализацията на КПЗ

Вече стана дума, че нормализацията на двоична мантиса позволява да се подразбира първата цифра (тя е единица) и да се запомни в паметта допълнително още една цифра, което малко подобрява точността на представяне на кодираното число.

Обаче нормализацията или денормализацията на КПЗ води до значително по-съществени следствия.

Най-напред, НКПЗ е единствен за всяко кодирано число, защото е единствена най-лявата ненулева цифра в записа на числото. Това позволява сравняването за равенство и различие на кодирани числа да става чрез лексикографско сравняване на техните НКПЗ. Обаче при ненормализирани КПЗ ще се наложи някакво изчисление.

Друга особеност е, че за някои кодирани числа съществуват по няколко различни ненормализирани КПЗ, получавани от различни експоненциални представяния. При това е възможно всички тези кодове да представят без закръгляне едно и също число.



Например:

експоненциално представянето	характеристика	съответен КПЗ
$A = -0,00011_{(2)} \cdot 2^{-5}$	$ИК_4(-5) = 0011_{(2)}$	$КПЗ_{4,6}(A) = 1\ 0011\ 000011_{(2)}$
$A = -0,00110_{(2)} \cdot 2^{-6}$	$ИК_4(-6) = 0010_{(2)}$	$КПЗ_{4,6}(A) = 1\ 0010\ 000110_{(2)}$
$A = -0,01100_{(2)} \cdot 2^{-7}$	$ИК_4(-7) = 0001_{(2)}$	$КПЗ_{4,6}(A) = 1\ 0001\ 001100_{(2)}$
$A = -0,11000_{(2)} \cdot 2^{-8}$	$ИК_4(-8) = 0000_{(2)}$	$КПЗ_{4,6}(A) = 1\ 0000\ 011000_{(2)}$

От същия пример се вижда, че за такова число  $A$  не съществува ненулев НКПЗ с 4 разряда за характеристика, защото  $-8$  е минималното число, което може да се кодира в ИК в 4 разряда.

Следователно за някои кодирани числа най-точният КПЗ е ненормализиран.

Тъкмо за да се съхрани максимална възможна точност при изчисленията в процесорния блок за работа с КПЗ междинните резултати се поддържат в ненормализиран вид. Такива междинни стойности могат впоследствие да дадат краен резултат, допускащ нормализация без загуба на точност.

Обаче, когато за някое число съществува НКПЗ, тогава именно той е най-точният, защото съдържа най-много достоверно известни цифри.

Използването на НКПЗ съвместно с характеристики за порядъците и с разполагане на полето за характеристика преди полето за мантиса позволява сравняването по абсолютна стойност на кодирани числа да се извършва чрез лексикографско сравняване на техните НКПЗ. Това е много по-бързо от изчисленията.

А именно:

Нека

$$A = \overline{1, a_1 \dots a_{n(2)}} \cdot 2^{k+h}, \text{ където } h > 0, \text{ и}$$

$$B = \overline{1, b_1 \dots b_{n(2)}} \cdot 2^k.$$

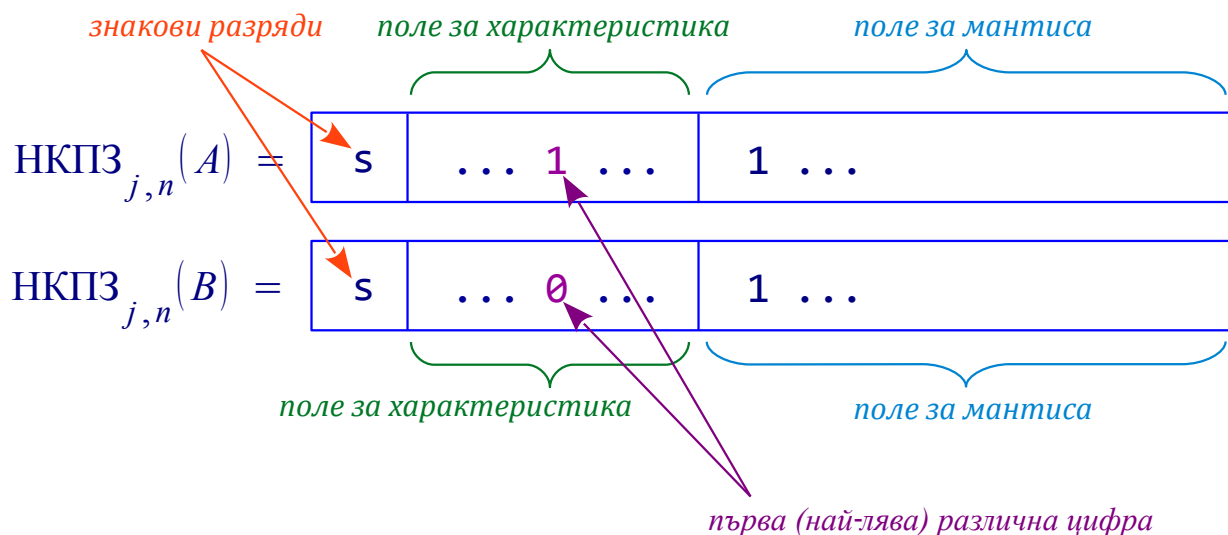
Тогава след умножаването по степента на двойката в записа на  $A$  дробната запетая ще бъде по-надясно, отколкото в записа на  $B$ . Т. е. при подравняването на запетаята ще се получи нещо от вида:

$$A = \overline{1 \quad \dots \quad , \quad \dots}_{(2)}$$

$$B = \overline{1 \quad \dots \quad , \quad \dots}_{(2)}$$

Очевидно, когато порядъкът на  $A$  е по-голям от порядъка на  $B$  и кодовете са нормализирани, тогава  $A > B$ . Обаче ИК запазва наредбата на порядъците. Затова характеристиката на  $A$  също ще бъде по-голяма от характеристиката на  $B$ .

Така се получава съответствие от вида:



Следователно НКПЗ на  $A$  ще бъде лексикографски по-голям от НКПЗ на  $B$ , когато порядъкът на  $A$  е по-голям от порядъка на  $B$ .

По-нататък нека

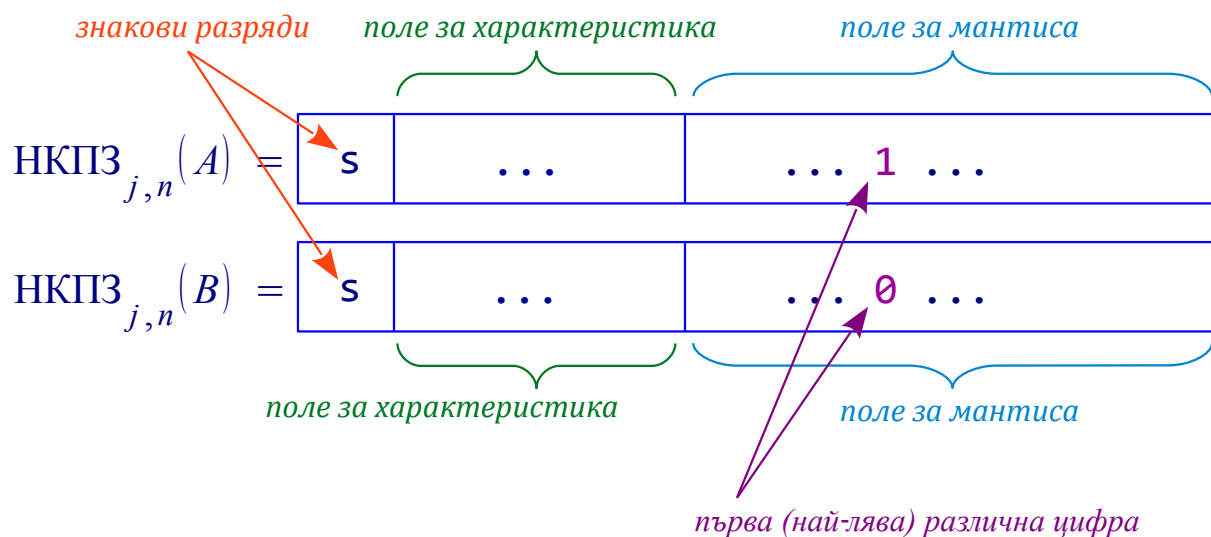
$$A = \overline{1, a_1 \dots a_j 1 a_{j+1} \dots}_{(2)} \cdot 2^k \text{ и}$$

$$B = \overline{1, a_1 \dots a_j 0 b_{j+1} \dots}_{(2)} \cdot 2^k.$$

Тогава след умножаването по степента на двойката записът на  $A$  ще бъде подравнен по дробната запетая спрямо запис на  $B$ . Т. е. за произведенията ще се получи нещо от вида:

$$\begin{aligned} A &= \overline{1 a_1 \dots a_j 1 a_{j+1} \dots, \dots}_{(2)}, \\ B &= \overline{1 a_1 \dots a_j 0 b_{j+1} \dots, \dots}_{(2)} \end{aligned}$$

За КПЗ съответно се получава нещо от вида:



Очевидно, когато са равни порядъците на  $A$  и на  $B$  и мантисата на  $A$  е по-голяма от мантисата на  $B$ , тогава  $A > B$  (независимо дали кодовете са нормализирани).

Възможността за сравняване по абсолютните стойности на кодирани числа чрез лексикографско сравняване на техните НКПЗ, което е много по-просто и по-бързо от изчисляване, е една от причините порядъкът да се представя в КПЗ с ИК.

## 26. Примери за получаване на КПЗ

Тук разглеждаме само КПЗ от вида  $\pm \overline{a_0, a_1 \dots a_n}_{(2)} \cdot 2^k$ , в които  $a_0$  се съхранява в паметта, а порядъкът се представя чрез изместен код (характеристика). С други КПЗ се работи аналогично.

### Задача 8

Да се намери двоичният запис на НКПЗ<sub>8,11</sub>(39,27).

### Примерно решение

Първо получаваме началото на двоичния запис на 39,27 с 12 цифри, започвайки да броим от най-лявата ненулева, за да можем да закръглим до 11-ата цифра.

За намирането на цялата и дробната части от търсения запис се използват различни преобразования. Затова получаваме двете части поотделно:

$$\begin{array}{r|l}
 39:2 & \\
 \hline
 19 & 1 \\
 9 & 1 \\
 4 & 1 \\
 2 & 0 \\
 1 & 0 \\
 0 & 1
 \end{array}
 \Rightarrow 39_{(10)} = 100111_{(2)}$$

Търсим още 6 цифри от дробната част (за да станат общо 12 известните):

$$\begin{array}{r|l}
 2.0,27 & \\
 \hline
 0 & 54 \\
 1 & 08 \\
 0 & 16 \\
 0 & 32 \\
 0 & 64 \\
 1 & 28 \\
 \vdots & \vdots
 \end{array}
 \Rightarrow 0,27_{(10)} = 0,010001..._{(2)}$$

$$\Rightarrow 39,27_{(10)} = 100111,010001..._{(2)}$$

По-нататък получаваме експоненциалното представяне:

$$\Rightarrow 39,27_{(10)} \approx 100111,01001_{(2)} = 1,0011101001_{(2)} \cdot 2^5$$

Изчисляваме характеристиката, т. е. ИК на 5 :

$$\begin{array}{r}
 \text{ИК}_8(+5) = 2^7 + 5 = 1000\,0000_{(2)} + 101_{(2)} \\
 + \quad 10000000 \\
 \hline
 \quad 101 \\
 10000101 \Rightarrow \text{ИК}_8(+5) = 1000\,0101_{(2)}
 \end{array}$$

$$\Rightarrow \text{НКПЗ}_{8,11}(39,27) = 0\,10000101\,10011101001_{(2)}$$

(С червен цвят е написана характеристиката, т. е. ИК на порядъка.)

**Задача 9**

Да се намери двоичният запис на  $\text{НКПЗ}_{5,15}(-81,519)$ .

**Примерно решение**

Първо получаваме началото на двоичния запис на 81,519 с 16 цифри, започвайки да броим от най-лявата единица, за да можем да закръглим до 15-ата цифра.

За намирането на цялата и дробната части от търсения запис се използват различни преобразования. Затова получаваме двете части поотделно:

$$\begin{array}{r|l}
 81 : 2 & \\
 \hline
 40 & 1 \\
 20 & 0 \\
 10 & 0 \\
 5 & 0 \\
 2 & 1 \\
 1 & 0 \\
 0 & 1
 \end{array}
 \Rightarrow 81_{(10)} = 1010001_{(2)} \text{ и }
 \begin{array}{r|l}
 2.0, 519 & \\
 \hline
 1 & 038 \\
 0 & 076 \\
 0 & 152 \\
 0 & 304 \\
 0 & 608 \\
 1 & 216 \\
 0 & 432 \\
 0 & 864 \\
 1 & 728 \\
 \vdots & \vdots
 \end{array}$$

$$\Rightarrow 0,519_{(10)} = 0,100001001\dots_{(2)}$$

$$\Rightarrow -81,519_{(10)} = -100111,010001\dots_{(2)}$$

След това получаваме експоненциалното представяне:

$$\Rightarrow -81,519_{(10)} \approx -1010001,10000101_{(2)} = -1,01000110000101_{(2)} \cdot 2^6$$

Изчисляваме характеристиката, т. е. ИК на 6:

$$\text{ИК}_5(+6) = 2^4 + 6 = 10000_{(2)} + 110_{(2)}$$

$$\begin{array}{r}
 + 10000 \\
 110 \\
 \hline
 10110
 \end{array}
 \Rightarrow \text{ИК}_5(+6) = 10110_{(2)}$$

$$\Rightarrow \text{НКПЗ}_{5,15}(-81,519) = 1\text{ }10110\text{ }101000110000101_{(2)}$$

(С червен цвят е написана характеристиката, т. е. ИК на порядъка.)

**Задача 10**

Да се намери двоичният запис на НКПЗ<sub>7,11</sub>(-16,151).

**Примерно решение**

Първо получаваме началото на двоичния запис на 16,151 с 12 цифри, започвайки да броим от най-лявата единица, за да можем да закръглим до 11-ата цифра.

За намирането на цялата и дробната части от търсения запис се използват различни преобразования. Затова получаваме двете части поотделно:

$$16_{(10)} = 10000_{(2)}$$

Търсим още 7 цифри от дробната част:

$$\begin{array}{r|l}
 2.0, 519 & \\
 \hline
 1 & 038 \\
 0 & 076 \\
 0 & 152 \\
 0 & 304 \\
 0 & 608 \\
 1 & 216 \\
 0 & 432 \\
 0 & 864 \\
 1 & 728 \\
 \vdots & \vdots
 \end{array} \Rightarrow 0,151_{(10)} = 0,0010011..._{(2)}$$

$$\Rightarrow -16,151_{(10)} = -10000,0010011..._{(2)}$$

След това получаваме експоненциалното представяне:

$$\Rightarrow -16,151_{(10)} \approx -10000,001010_{(2)} = -1,0000001010_{(2)} \cdot 2^4$$

Изчисляваме характеристиката, т. е. ИК на 4 :

$$\text{ИК}_7(+4) = 2^6 + 4 = 1000000_{(2)} + 100_{(2)}$$

$$\begin{array}{r}
 + 1000000 \\
 \underline{\quad 100 \quad} \\
 1000110
 \end{array} \Rightarrow \text{ИК}_7(+4) = 1000110_{(2)}$$

$$\Rightarrow \text{НКПЗ}_{7,11}(-16,151) = 1 \textcolor{red}{1000110} 10000001010_{(2)}$$

(С червен цвят е написана характеристиката, т. е. ИК на порядъка.)

### Задача 11

Да се намери двоичният запис на  $\text{НКПЗ}_{6,8}(-0,0098)$ .

#### Примерно решение

Първо получаваме началото на двоичния запис на 0,0098 до 9-ата цифра включително, започвайки броенето от най-лявата единица:

2.0, 0098

0	0196
0	0392
0	0784
0	1568
0	3136
0	6272
1	2544
0	5088
1	0176
0	0352
0	0704
0	1408
0	2816
0	5632
1	1264
⋮	⋮

$$\Rightarrow 0,0098_{(10)} = 0,000000101000001..._{(2)}$$

$$\Rightarrow -0,0098_{(10)} = -0,000000101000001..._{(2)}$$

Получаваме експоненциалното представяне:

$$\Rightarrow -0,0098_{(10)} \approx -0,00000010100001_{(2)} = -1,0100001_{(2)} \cdot 2^{-7}$$

Изчисляваме характеристиката, т. е. ИК на  $-7$ :

$$\text{ИК}_6(-7) = 2^5 - 7 = 100000_{(2)} - 111_{(2)}$$

Удобно е да изчисляваме в двоична система:

$$\begin{array}{r} \_ 100000 \\ \underline{\phantom{0}111} \\ \phantom{0}?????? \end{array}$$

При това възникват заеми:

$$\begin{array}{r} \_ \overset{\cdot}{1}\overset{\cdot}{0}\overset{\cdot}{0}\overset{\cdot}{0}\overset{\cdot}{0}\overset{\cdot}{0} \\ \underline{\phantom{0}111} \\ \phantom{0}?????? \end{array}$$

След като бъдат заети всички заеми в действителност извършваме изваждането:

$$\begin{array}{r} \_ 011112 \\ \underline{\phantom{0}111} \\ 011001 \end{array} \Rightarrow \text{ИК}_6(-7) = \mathbf{011001}_{(2)}$$

$$\Rightarrow \text{НКПЗ}_{6,8}(-0,0098) = \mathbf{1011001}10100001_{(2)}$$

(С червен цвят е написана характеристиката.)

### Задача 12

Да се намери двоичният запис на  $\text{НКПЗ}_{5,9}(-0,8)$ .

#### Примерно решение

Първо търсим  $-0,8_{(10)} = ?_{(2)}$ :

$$\begin{array}{r} \cancel{2.0,8} \\ \hline 1 \mid 6 \\ 1 \mid 2 \\ 0 \mid 4 \\ 0 \mid 8 \\ \vdots \mid \vdots \end{array}$$

$$\Rightarrow -0,8_{(10)} = -0, (1100)_{(2)} = -0,1100110011\dots_{(2)}$$

Получаваме експоненциалното представяне:

$$\Rightarrow -0,8_{(10)} \approx -0,110011010_{(2)} = -1,10011010_{(2)} \cdot 2^{-1}$$



Изчисляваме характеристиката, т. е. ИК на  $-1$  :

$$\text{ИК}_5(-1) = 2^4 - 1 = 10000_{(2)} - 1_{(2)} = \mathbf{01111}_{(2)}$$

$$\Rightarrow \text{НКПЗ}_{5,9}(-0,8) = \mathbf{101111}110011010_{(2)}$$

(С червен цвят е написана характеристиката.)

### Задача 13

Да се намери двоичният запис на  $\text{НКПЗ}_{3,4}(-0,121875)$ .

#### Примерно решение

Първо получаваме началото на двоичния запис на  $0,121875$  до петата цифра включително, започвайки броенето от най-лявата единица:

2 . 0 , 121875

0 | 24375

0 | 4875

0 | 975

1 | 95

1 | 9

1 | 8

1 | 6

1 | 2

⋮ | ⋮

$$\Rightarrow 0,121875_{(10)} = 0,00011111..._{(2)}$$

$$\Rightarrow -0,121875_{(10)} = -0,00011111..._{(2)}$$

Получаваме експоненциалното представяне:

$$\Rightarrow -0,121875_{(10)} \approx -0,0010000_{(2)} = -0,001000_{(2)} = -1,000_{(2)} \cdot 2^{-3}$$

Изчисляваме характеристиката, т. е. ИК на  $-3$  :

$$\text{ИК}_3(-3) = 2^2 - 3 = 1 = \mathbf{001}$$

$$\Rightarrow \text{НКПЗ}_{3,4}(-0,121875) = \mathbf{1001}1000_{(2)}$$

(С червен цвят е написана характеристиката.)

**Задача 14**

Да се намери двоичният запис на НКПЗ<sub>6,14</sub>(-0,1125).

**Примерно решение**

Първо търсим  $-0,1125_{(10)} = ?_{(2)}$ :

$$\begin{array}{r|l} 2 \cdot 0,1125 & \\ \hline 0 & 225 \\ 0 & 45 \\ 0 & 9 \\ 1 & 8 \\ 1 & 6 \\ 1 & 2 \\ 0 & 4 \\ 0 & 8 \\ \vdots & \vdots \end{array}$$

$$\Rightarrow -0,1125_{(10)} = -0,0001(1100)_{(2)} = -0,000111001100110011\dots_{(2)}$$

Получаваме експоненциалното представяне:

$$\Rightarrow -0,1125_{(10)} \approx -0,00011100110011010_{(2)}$$

$$\Rightarrow -0,1125_{(10)} \approx -1,1100110011010_{(2)} \cdot 2^{-4}$$

Изчисляваме характеристиката, т. е. ИК на  $-4$ :

$$\text{ИК}_6(-4) = 2^5 - 4 = 100000_{(2)} - 100_{(2)}$$

Удобно е да изчисляваме в двоична система:

$$\begin{array}{r} \underline{\phantom{0} 100000} \\ \phantom{0} 100 \\ \hline \phantom{0} \text{??????} \end{array}$$

При това възникват заеми:

$$\begin{array}{r} \underline{\phantom{0} \overset{\cdot}{1}\overset{\cdot}{0}\overset{\cdot}{0}000} \\ \phantom{0} 100 \\ \hline \phantom{0} \text{??????} \end{array}$$

След като всички заеми в действителност извършваме изваждането:

$$\begin{array}{r} -011200 \\ \underline{100} \\ 011100 \end{array} \Rightarrow \text{ИК}_6(-4) = 011100_{(2)}$$

$$\Rightarrow \text{НКПЗ}_{6,14}(-0,1125) = 101110011100110011010_{(2)}$$

(С червен цвят е написана характеристиката.)

### Задача 15

Да се намери двоичният запис на  $\text{НКПЗ}_{7,11}(-0,0875)$ .

#### Примерно решение

Първо търсим  $-0,0875_{(10)} = ?_{(2)}$  с поне 11 цифри след най-лявата единица:

2.0, 0875

$$\begin{array}{r|l} 0 & 175 \\ 0 & 35 \\ 0 & 7 \\ 1 & 4 \\ \hline 0 & 8 \\ 1 & 6 \\ 1 & 2 \\ 0 & 4 \\ \vdots & \vdots \end{array}$$

$$\Rightarrow -0,0875_{(10)} = -0,0001(0110)_{(2)} = -0,000101100110011..._{(2)}$$

Получаваме експоненциалното представяне:

$$\Rightarrow -0,0875_{(10)} \approx -0,00010110011010_{(2)}$$

$$\Rightarrow -0,0875_{(10)} \approx -1,0110011010_{(2)} \cdot 2^{-4}$$

Изчисляваме характеристиката, т. е. ИК на  $-4$ :

$$\text{ИК}_7(-4) = 2^6 - 4 = 1000000_{(2)} - 100_{(2)}$$

Удобно е да изчисляваме в двоична система:

$$\begin{array}{r} \_ 1000000 \\ \underline{\phantom{0}100} \\ \phantom{0}??????? \end{array}$$

При това възникват заеми:

$$\begin{array}{r} \_ \overset{\cdot}{1}\overset{\cdot}{0}\overset{\cdot}{0}\overset{\cdot}{0}000 \\ \underline{\phantom{0}100} \\ \phantom{0}??????? \end{array}$$

След като всички заеми в действителност извършваме изваждането:

$$\begin{array}{r} \_ 0111200 \\ \underline{\phantom{0}100} \\ 0111100 \end{array} \Rightarrow \text{ИК}_7(-4) = 0111100_{(2)}$$

$$\Rightarrow \text{НКПЗ}_{7,11}(-0,0875) = 1011110010110011010_{(2)}$$

(С червен цвят е написана характеристиката.)

## 27. Примери за получаване на кодирано число от известен КПЗ

Тук също разглеждаме само КПЗ от вида  $\pm \overline{a_0, a_1 \dots a_n}_{(2)} \cdot 2^h$ , в които  $a_0$  се съхранява в паметта, а порядъкът се представя чрез изместен код (характеристика).

### Задача 16

Да се намери 10-ичният запис на  $x$  при условие, че числото  $x$  се представя точно (без закръгляне) от кода  $\text{КПЗ}_{4,9}(x) = 011010110101_{(2)}$ .

### Примерно решение

Горният КПЗ на  $x$  съответства на експоненциално представяне

$$x = +0,11010101_{(2)} \cdot 2^h, \text{ където } \text{ИК}_4(h) = 1101_{(2)}.$$

Първо намираме порядъка  $h$ :

$$13 = 1101_{(2)} = \text{ИК}_4(h) = 2^3 + h = 8 + h \Rightarrow h = 5$$

След това заместваме  $h$  в експоненциалното представяне на  $x$  и преобразуваме към десетичен запис:

$$x = +0,11010101_{(2)} \cdot 2^5 = 11010,101_{(2)}$$

$$\begin{aligned} x &= 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = \\ &= 16 + 8 + 2 + \frac{1}{2} + \frac{1}{8} \end{aligned}$$

$$\text{Следователно } x = 26,625_{(10)}.$$

### Задача 17

Да се намери 10-ичният запис на  $x$  при условие, че числото  $x$  се представя точно (без закръгляне) от кода  $\text{КПЗ}_{5,9}(x) = 1\mathbf{01110}101000000_{(2)}$ .

#### Примерно решение

Горният КПЗ на  $x$  съответства на експоненциално представяне

$$x = -1,01000000_{(2)} \cdot 2^h, \text{ където } \text{ИК}_5(h) = 01110_{(2)}.$$

Първо намираме порядъка  $h$ :

$$14 = 01110_{(2)} = \text{ИК}_5(h) = 2^4 + h = 16 + h \Rightarrow h = -2$$

След това заместваме  $h$  в експоненциалното представяне на  $x$  и преобразуваме към десетичен запис:

$$x = -1,01000000_{(2)} \cdot 2^{-2} = -0,0101_{(2)}$$

$$x = -\left(\frac{1}{2^2} + \frac{1}{2^4}\right) = -\frac{5}{2^4} = -\frac{5 \cdot 5^4}{2^4 \cdot 5^4} = -\frac{5 \cdot 625}{10^4} = -\frac{3125}{10^4}$$

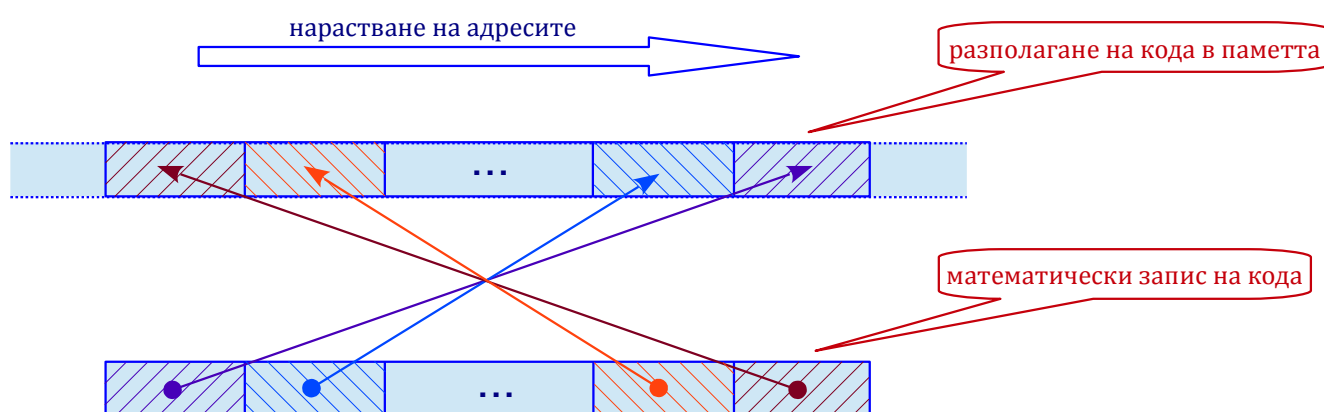
$$\text{Следователно } x = -0,3125_{(10)}.$$

## 28. Разполагане на кодове в паметта

При съхраняването в паметта кодът на число може да бъде разполаган по два начина според съгласенията на конкретната архитектура – старшите байтове от кода могат да попадат в байтовете с по-малки адреси или в байтовете с по-големи адреси.

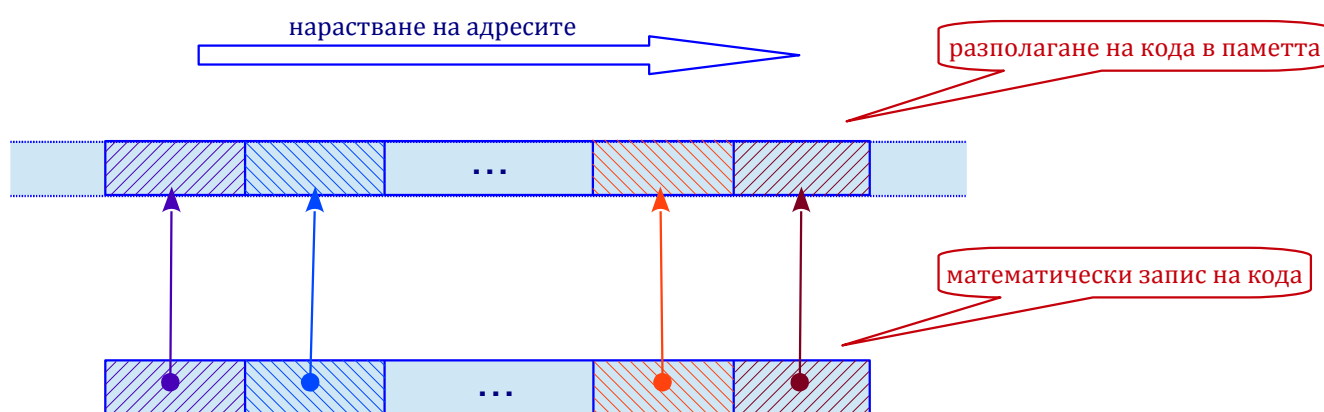
Това изобразяват следните две схеми:

### Разполагане на по-старши байт от кода на по-голям адрес в паметта



Горното разположение в паметта е типично за архитектурите, наричани x86.

### Разполагане на по-старши байт от кода на по-малък адрес в паметта



Горното разположение в паметта на байтовете от код за машинно представяне на число е типично за архитектурите на Motorola.

Наредбата на байтовете се отчита при поразрядните измествания наляво и надясно на цифрите на данните, като изместването точно съответства на действието, извършвано с математическия запис.

## **29. Следствия от машинните числови формати за езиците от високо ниво**

На ниско ниво в универсалните компютри се поддържат (разпознават) такова множество от машинни формати на данни, с което може да бъдат програмирани всякакви потенциално възможни алгоритми. При това, за обичайните потребности се осигурява приемлива ефективност на изпълнимия код. Обаче точно такива възможности искаме да предоставят и езиците от високо ниво.

Следователно същите видове данни, които се разпознават от машинни инструкции, трябва да са достъпни и в езиците за програмиране от високо ниво. Точно затова машинните формати за числа определят основните примитивни типове във всички езици за програмиране.

В същото време, хардуерната поддръжка на особено прецизни изчисления е недостатъчно изгодна икономически и не се поддържа в масово разпространените архитектури. Обаче за практиката се оказва твърде важна точността на представяне и пресмятанията с крайни десетични дробни, каквито постоянно се използват в бизнеса, науката и в областите, свързани с някакъв вид риск. За да осигурят изчисления с висока прецизност в такива случаи, много езици включват малък брой допълнителни типове.

Понякога с тази цел се въвежда специален примитивен тип, представящ десетичните числа във формат без преобразуване на дробната част в двоичен запис. Например такъв е типът `Decimal` в `C#` и `.NET`.

Като алтернативен подход или заедно с горния понякога езикът за програмиране се допълва с библиотечен тип, поддържащ изчисления с много голям брой достоверни цифри.

По такъв начин системата от примитивни типове се оказва много подобна във всички езиците за програмиране. Например практически винаги присъствуват цели числа без и със знак в 1, 2, 4 и 8 байта и КПЗ в 4, 8 и или 10, или 12 байта и тези типове имат почти едни и същи характеристики. Съответно срещащите се по-особени типове обикновено осигуряват допълнителни възможности за изчисления с повишена точност и са подчинени на сходни идеи.

Затова разбирането на фундаменталните идеи за представяне на числа в цифрови компютри вече осигурява съществени базови знания за всички езици за програмиране.

Аналогична е ситуацията и с операторите. Отново ядрото от поддържани видове оператори се заимствува от машинните езици, които пък в това отношение особено много си приличат. По-нататък се появяват значителен брой нови видове операции, характерни именно за високото ниво на програмиране. Обаче те също пряко или косвено са повлияни от принципите на хардуера, а освен това много строго се придържат към едни и същи фундаментални идеи, типични въобще за компютърния свят.

По този начин и много оператори се оказват почти едни и същи във всички езици за програмиране и заедно с това следват еднаква логика, произтичаща от машинното ниво и доразвивана по сходен начин във високото ниво на програмиране.

Горните разсъждения водят до извода, че познаването и разбирането на форматите за кодиране на числа в масово разпространените компютри е много полезна и дори необходима съставляваща на знанията за компютърното програмиране.