Упражнение №1
Представяне на числата и бройни системи

Във всекидневния ни живот по навик използваме десетична бройна система за представяне на числата. Без усилие записваме произволни стойности - на положителни и отрицателни, на цели и дробни числа. При работа с по-големи стойности съществуват методи за компактен запис на числата, напр. като се използва мантиса, даваща една базова стойност и експонента, представляваща коефициент, степен на 10, по който трябва да бъде умножена мантисата, за да се получи действителната стойност на изразяваното число.

В основата на десетичната бройна система, както и на всяка друга, както ще стане видно по-нататък, стои използването на известно количество различни символи, в случая 10 на брой:

0 1 2 3 4 5 6 7 8 9

При изчерпване на всички възможни символи (преброили сме повече от 10 обекта), се преминава към представяне на количеството с повече от един разред. Така говорим за разред на единиците (10^0), разред на десетиците (10^1), на стотиците (10^2) и т.н. Цифрата във всеки разред е коефициентът, по който трябва да се умножи количеството, представляващо съответната степен на бройната система:

0 1 2 .... 9 **10** 11 12 ... 99 **100** 101 102 ... **1000** ...

За пълнота, тъй като от математическа гледна точка е възможно използването на произволна бройна система, би трябвало при записа числата да се представят като

3427_{10} 349517_{10} $64{,}28_{10}$ 95810063_{10} ....

За наше улеснение не добавяме индекса 10, който посочва коя е основата на бройната система, а го подразбираме.

Освен това, при представяне на стойностите, по подразбиране приемаме че това е един съкратен запис на сумата от единиците, десетиците, стотиците и т.н.

За едно число пълният запис би представлявал

$349517_{10} = 3.10^5 + 4.10^4 + 9.10^3 + 5.10^2 + 1.10^1 + 7.10^0 =$

$= 300\,000 + 40\,000 + 9\,000 + 500 + 10 + 7$

В цифровата и изчислителната техника този подход е неудобен по практически съображения. Макар и възможно, създаването на електронен елемент с 10 различни, стабилни и повторяеми състояния е много трудно, скъпо и ненадеждно. По тази причина се използват елементи само с две състояния, които за удобство е прието да се означават с

0 - нула и 1 - единица.

Важно е да се подчертае, че тук не става дума за количества, а за ***състояния***.

Електронните елементи с две състояния се наричат *двоични* (*бинарни*), а аритметиката, изградена на базата на такива елементи се нарича двоична аритметика. При електронната им реализация се използват прости, евтини средства и технически решения, което позволява постигане именно на характеристиките, необходими за построяване на цифрови устройства и цифрови електронни изчислителни машини (ЦЕИМ) - компютрите. Чрез добавянето на еднакви, прости конструктивни блокове могат както в десетичната бройна система де се представят произволни стойности. Един блок, който може да се намира в състояние "0" или "1", представлява един двоичен разред, наречен **бит** (**bit**). Наименованието му идва от **bi**nary uni**t** - двоична единица. Понякога се използва и терминът *digit*. Това дава и наименованието на тази област от електрониката - дигитална или цифрова електроника. В нея елементите могат да се намират само или в едното състояние - "**0** - нула", което се отъждествява с ниско ниво на напрежението (*low*), слаб ток, отворен контакт и др. под. или в противоположното състояние "**1** - единица" - високо ниво (*high*), силен ток, затворен контакт и т.н. В цифровата техника не съществуват междинни състояния на елементите, което изключва всякакво двусмислие. Или е топло (температурата е над някакъв праг), или студено (под този праг). Преходът между двете състояния е рязък и в идеалния случай - мигновен. От своя страна, за един елемент всяко състояние може да се запазва неограничено дълго време, ако той не бъде изведен от него посредством подходящо въздействие. За съхранение на двоичната информация в цифровите машини се използват специални елементи с памет - тригери (*flip-flop*). Всеки тригер може да съхранява, да помни, 1 бит информация. Колкото по-голямо е двоичното число, толкова повече тригери са необходими - по един за всеки негов бит. За съхранение на числена информация в машините тригерите са обединени в неделими групи - регистри и операциите на запис и на прочитане на състоянието на всеки тригер са едновременни за всички тригери от даден регистър. Ако при запис на ново число състоянието на някой от тригерите в регистъра трябва да остане същото, записът въпреки това се изпълнява, но не се наблюдава изменение на съответния бит. При прочитане на съдържанието на регистъра то не се променя - не се загубва. Всеки тригер запазва състоянието, в което е бил установен при записа.

Използвайки двоичните елементи, работим в двоична бройна система – **BIN***ary number system*. В нея числата се представят с помощта само на два символа - 0 и 1. Един-единствен двоичен разред позволява записа само на две числа - нула и едно. Когато трябва да се представят и записват по-големи стойности, се използват съответното количество двоични разреди. Принципът на формиране на двоичните числа е същият, както при десетичните:

двоичното число се представя чрез коефициентите пред степените на основата на бройната система, която в случая е две.

Пълният запис на едно произволно двоично число 10011011101_2 би бил

$$10011011101_2 = 1.2^{10} + 0.2^9 + 0.2^8 + 1.2^7 + 1.2^6 + 0.2^5 + 1.2^4 + 1.2^3 + 1.2^2 + 0.2^1 + 1.2^0$$

По определение, във всяка бройна система, число повдигнато на степен 0 е равно на единица!

$$\mathbf{2^0 \equiv 1}$$

Следователно, след заместване на степените на 2 с техните еквиваленти

10011011101_2 =

= 1.1024 + 0.512 + 0.256 + 1.128 + 1.64 + 0.32 + 1.16 + 1.8 + 1.4 + 0.2 + 1.1 =

= 1024 + 128 +64 + 16 + 8 + 4 +1 = **1245_{10}**

Всяка двоична единица (даден разред) се нарича бит. В цифровата техника съществуват стандартни начини за представяне на величините чрез групи от битове. Най-малката група е един бит. От математическа гледна точка с един бит могат да се представят две състояния. Използването на по-големи групи позволява представяне на по-големи величини. Група от 4 бита се нарича тетрада (*nibble*), от 8
- байт (*Byte*), от 16 - дума (*Word*).

1	бит (*bit*)

1	1	0	1	тетрада (*nibble*)

1	0	1	1	0	0	1	1	байт (*Byte*)

2 тетради

1	0	1	1	0	0	1	1

=

1	0	1	1	0	0	1	1	1 байт

1	1	0	1	0	0	1	1	1	1	0	0	0	1	0	1	0	дума = 2 байта = 4 тетради

Както в десетичната аритметика, най-левият разред (бит15 - b15) е старшият бит на думата (MSB - *Most Significant Bit*). Той съответства на най-високата положителна степен на основата на използваната бройна система. Най-десният (бит0 - b0) е младшият бит на думата (LSB - *Least Significant Bit*) - пред нулевата или най-високата отрицателна степен на основата. По същия начин характеризираме байтовете в думата - най-левият е старшият байт, а най-десният - младшият.

старши бит *младши бит*

MSB															**LSB**
b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	**b0**
1	1	1	0	0	0	0	1	1	1	0	1	1	1	1	**1**
	старши байт							*младши байт*							

Както се вижда от приведените примери, колкото по-малка е основата на бройната система, толкова повече разреди са наобходими за представяне на една и съща величина: **$10011011101_2 = 1245_{10}$**

10011011101_2 -> единадесет двоични разреда

1245_{10} -> четири десетични разреда

На базата на степените на 2 съществуват и други бройни системи. Числото 8 –> 2^3 = 8, е основа на осмичната бройна система **OCT***al number system*, в която се използват само осем символа:

0 1 2 3 4 5 6 7.

Аналогично, 2^4 = 16, което число е основата на шестнадесетичната бройна система **HEX***adecimal number system.* В нея, понеже броят на различните символи, които могат да се изобразят с десетичните цифри е по-малък от 16, за недостигащите символи се използват (като стандарт!) първите 6 букви от латинската азбука. Така се получава наборът

0 1 2 3 4 5 6 7 8 9 A B C D E F .

В цифровата и компютърната техника се използват основно двоичната **BIN** и шестнадесетичната **HEX** бройни системи. Първата, понеже това е естественият начин за директно въвеждане на информацията в цифровите машини, а втората – защото позволява съкратен запис на числата. Това се вижда от следната таблица за преминаване от една бройна система в друга. След запълване на съответния (младши) разред - когато целият набор от различни символи е използван, се добавя нов разред отляво - старши.

DEP	BIN	OPT	HEX
0	000**0**	0	0
1	000**1**	1	1
2	00**1**0	2	2
3	00**1**1	3	3
4	0**1**00	4	4
5	0**1**01	5	5
6	0**1**10	6	6
7	0**1**11	7	7
8	**1**000	**1**0	8
9	**1**001	**1**1	9
10	**1**010	**1**2	A
11	**1**011	**1**3	B
12	**1**100	**1**4	P
13	**1**101	**1**5	D
14	**1**110	**1**6	E
15	**1**111	**1**7	F
16	**1**0000	**2**0	**1**0

За да се избегне двусмислието, когато в един документ се използват различни бройни системи са възприети следните символни означения за бройните системи, различни от десетичната:

двоично число – % 01101101

осмично число – @ 2307

шестнадесетично число – $FP03B

В езиците за програмиране на машинно ниво съществуват и други, специфични за производителите възможности за представяне на числата. Напр. вместо $FP03B, същото число може да се напише като 0xFP03B или FP03Bhex.

В таблицата се забелязва и друга особеност – на всяка шестнадесетична цифра директно съответства група от 4 бита (тетрада), което позволява много лесния преход от двоично към шестнадесетично представяне на числата и обратно. Така

5 - > %0101 = $5 ; 10 -> %1010 = $A 11 -> %1111 = $F

При преминаването от BIN към HEX формат за по-големи числа, от двоичния запис се обособяват групи от по 4 бита (тетради), започвайки от младшия към старшия разред. Ако в последната - старшата група броят на двоичните цифри е недостатъчен за пълна тетрада тя задължително се допълва с нули в старшите разреди:

$$\mathbf{1245_{10} = 10011011101_2 = 0010\ 0110\ 1110}$$

След това получените тетради се заместват директно със съответния шестнадесетичен символ

%0010 0110 1110 = $46E

Двоичният запис се използва с предпочитане, когато трябва да се подчертае състоянието на определен бит - напр. в една управляваща програма, където всеки бит е асоцииран към определено условие или ситуация в контролираната система. Когато се представя количествена информация, числа, се предпочита шестнадесетичен (а в някои случаи - десетичен) запис, за по-голяма компактност и яснота.

Преход между бройните системи

В приведените по-горе примери беше показано елементарното преобразуване на числата, представени в бройни системи, чиято основа е цяла степен на две. В много случаи обаче данните, постъпващи от външни източници са представени с десетични числа. За обработката им в цифровите машини те предварително трябва да бъдат преобразувани в техния двоичен еквивалент.

Един алгоритъм за такова преобразуване е илюстриран със следния пример:

Число$_{10}$	Основа		Частно	Остатък		
173	: 2	=	86	1	LSB	b0
86	: 2	=	43	0		b1
43	: 2	=	21	1		b2
21	: 2	=	10	1		b3

10	: 2	=		5	0		b4
5	: 2	=		2	1		b5
2	: 2	=		1	0		b6
1	: 2	=	??	1<2	1	MSB	b7

Следователно, резултатът от преобразуването е %10101101 или $AD .

Двоичното число се получава, като след остатъкът след първото делене се запазва като младши бит на двоичното число. Деленето продължава, докато частното стане по-малко от основата на новата бройна система (делителя), в случая – 1 или 0. Остатъкът от последното делене представлява старшия бит на новото число.

Основно правило в цифровата техника е числата да се представят с една и съща разредност. Това е необходимо за правилното подреждане на битовете по старшинство. Ако възприетият формат е 8 бита, дори при деленето да се получава резултат нула, то продължава докато се запълнят (в случая - с нули) всички разреди до старшия:

Число$_{10}$	Основа			Частно	Остатък		
38	: 2	=		19	0	LSB	b0
19	: 2	=		9	1		b1
9	: 2	=		4	1		b2
4	: 2	=		2	0		b3
2	: 2	=		1	0		b4
1	: 2	=	??	1<2	1		b5
1	: 2	=	??	1<2	0		b6
1	: 2	=	??	1<2	0	MSB	b7

След преобразуването 38_{10} -> % 00100110 -> $26

Преобразуването DEP -> HEX се изпълнява по същия начин, но тъй като деленето на две е много по-лесно, обикновено се намира първо двоичната форма, която се преобразува към HEX без да се прибягва до сложни изчисления.

Елементи на двоичната аритметика

В двоичната аритметика съществуват две основни операции – събиране и изваждане. Правилата за изпълнението им са следните:

Събиране				Изваждане			
		Резултат	Пренос			Резултат	Заем
0 + 0	=	0	0	0 - 0	=	0	0
0 + 1	=	1	0	0 - 1	=	1	1
1 + 0	=	1	0	1 - 0	=	1	0
1 + 1	=	0	1	1 - 1	=	0	0

Примери:

	1111111	1	1 1
45_{10}	00101101	83_{10}	01010011
+ 83_{10}	+ 01010011	- 45_{10}	- 00101101
128_{10}	10000000	38_{10}	00100110

Преносът и заемът се отнасят до един неявно съществуващ старши разред, така че да се получи правилен резултат след операцията. За изпълнението им в системите се налага да се използват два различни изчислителни блока – *суматор* за събирането и *субтрактор* за изваждането. Това затруднява и оскъпява апаратната им реализация. По тази причина се използват други начини за постигане на същия краен резултат, като се използва различен формат за представянето на числата при изваждане, а операцията се свежда както при десетичната аритметика до събиране на положително с отрицателно число.

До тук разглеждахме работата само с цели положителни числа. За тях казваме, че са "цели числа без знак". Вижда се, че в практиката това би довело до недопустими ограничения, което налага да се използва метод за представяне и на отрицателните числа, позволяващ при това изчислителните операции с тях да се изпълняват по същия начин, както това става в обикновената аритметика.

Положителни и отрицателни числа

В "традиционната" математика отрицателните числа, независимо от начина на запис и от бройната основа имат еднакво представяне, като пред числото се поставя знак (— *минус*). В двоичната аритметика, където единствените символи, с които разполагаме са "0" и "1", такова представяне е невъзможно. Това е довело до създаването на различни методи за запис и работа с отрицателни числа.

Един от начините да се представят числата със знак е "*знак и големина*" - "*sign and magnitude*" ***S&M***, при който един от битовете в двоичния им запис се използва като бит за знак (*sign*). Това обикновено е старшият бит, а останалите служат за запис на големината (*magnitude*) на числото. Работейки с 8 бита напр., след отделяне на бита за знак, за големината остават 7 бита. Така нейният обхват е от 000000 |*0*| до 1111111 |*127*| или общо 128 различни стойности. "Долепвайки" бита за знак като старши, получаваме

+127	**0**1111111	**1**0000000	- 0
+126	**0**1111110	**1**0000001	- 001
.....			.....
+ 001	**0**0000001	**1**1111110	-126
+ 0	**0**0000000	**1**1111111	-127

+127 .... **+ 0** **- 0** .... -127

отрицателни <- | -> положителни

При това представяне положителните и отрицателните числа се различават само по своя знаков бит. **Недостатък**: Известно е, че сумата на положително и отрицателно число с една и съща абсолютна стойност е нула
(+6) + (-6) = 0.

Спазвайки изложените по-горе правила за двоичното събиране, същата операция върху числата, представени във формат ***S&M***, дава

+ 6_{10}	00000110
- 6_{10}	10000110
0	10001100 <- (-12_{10})

Видно е, че резултатът е напълно погрешен! За да се избегне това е необходимо изпълнението на сложни алгоритми за корекция. Следваща особеност е съществуването на "положителна нула - **0**0000000" и "отрицателна нула - **1**0000000".

"Допълнение до 1" и "допълнение до 2"

Друг начин за запис на отрицателните числа е т.нар. "*допълнение до 1*" (*1's complement*). Допълнението до 1 на едно двоично число се получава като състоянието на всеки от битовете му, независимо от останалите, бъде заменено с противоположното. Тази операция се нарича инверсия или допълнение. И в този случай се запазва принципът, че старшият бит (b7) е отделен за знак, а младшите битове (b6 ÷ b0) дават числената стойност. Използвайки N бита за записа, обхватът на стойностите е от $-(2^{N-1} - 1)$ до $+ (2^{N-1} - 1)$. Стандартната дължина от 8 бита позволява представянето на числата от -127_{10} до $+127_{10}$, като нулата отново има две представяния: +0 (00000000) и -0 (11111111)
Примери:

Число	+ 0	- 0	+ 39	- 39	+ 100	- 100
допълнение до 1	00000000	11111111	00100011	11011100	01100100	10011011

+127	**0**1111111	**1**0000000	- 127
+126	**0**1111110	**1**0000001	- 126
.....			.....
+ 001	**0**0000001	**1**1111110	- 1
+ 0	**0**0000000	**1**1111111	- 0

+127 .... **+ 0** **- 0** .... -127

отрицателни <- | -> положителни

Събирането на две числа във формат "допълнение до 1" става по стандартните правила за събиране на двоични числа:
- сумата на на положително и отрицателно число с една и съща абсолютна стойност е нула

+ 90_{10}	01011010
- 90_{10}	10100101
0	11111111 <- едното от възможните представяния на "0"

- при препълване към получения резултат трябва да бъде добавена стойността на преноса:

пренос

(- 53_{10}) 1 11001010

+ (+ 118_{10})	+ 01110110		
(+ 65_{10})	01000000	<- + 64_{10}	неверен резултат
	+ 1	корекция - събиране с преноса	
	01000001	<- + 65_{10}	правилен резултат

Наименованието на операцията "допълнение до 1" идва от факта, че от математическа гледна точка този формат на едно двоично число се получава като то се извади от двоично число със същата разредност, за което всички разреди са "1" (това фактически е най-голямото число, което може да се запише със същия брой битове):

$|127|_{10}$ X1111111 (2^7-1 = 127 - най-голямото число)

$|103|_{10}$ - X**0100101** (изваждане на преобразуваното число)

X1011010 (допълнение до 1 на числото; X - знаков бит).

"Механичното" преобразуване чрез инверсия на всеки бит обаче е много полесна и бърза операция, която се осъществява с една-единствена команда във всички процесори.

И двата представени формата на двоичните числа "*S&M*" и "*допълнение до 1*" имат недостатъци, които изискват допълнителни действия, за получаване на правилен резултат при изпълнение на аритметичните операции. Тези недостатъци отпадат при използване на следващия формат - "**допълнение до 2**" (*2's complement*), масово използван в съвременните компютри.

За да получим отрицателната стойност на едно число (независимо дали самото то е положително или отрицателно) във формат "**допълнение до 2**", то се изважда от първата следваща по-голяма степен на бройната основа – 2 и пред него се добавя (най-отляво - в позицията на старшия разред). Той има индикаторна функция и при аритметичните операции не се включва в количествената информация, съдържаща се в числото:

			знак	стойност							
положително число	+ 102	->	0	1	1	0	0	1	1	0	
отрицателно число	- 82	->	1	0	1	0	1	1	1	0	*2's comp*

В табличен вид числата (*2's comp*) могат да бъдат представени така:

No по ред	DEP	HEX	BIN		BIN	HEX	DEP	No по ред
128	- 128	80	**1**000 0000		**0**111 1111	7F	+ 127	128
127	- 127	81	**1**000 0001		**0**111 1110	7E	+ 126	127
126	- 126	82	**1**000 0010		**0**111 1101	7D	+ 125	126
	.....						.....	
	.....						.....	
	.....						.....	
4	- 004	FP	**1**111 1100		**0**000 0011	03	+ 003	4
3	- 003	FD	**1**111 1101		**0**000 0010	02	+ 002	3

2	- 002	FE	**1**111 1110		**0**000 0001	01	+ 001	2
1	- 001	FF	**1**111 1111		**0**000 0000	00	000	1

Както се вижда от таблицата, в този формат **всички** положителни числа се представят както обикновените двоични числа без знак, само чрез тяхната големина плюс допълнителния старши бит за знак, който винаги е **нула**. Във формат допълнение до 2 старшият бит на положително число не може да бъде единица!!

Получаването на допълнението до 2 е илюстрирано със следния пример:
Да се намери допълнението до 2 на десетичното число + 50_{10} .

бит 5 4 3 2 1 0

- - преобразуваме числото в двоична форма : + 50_{10} => % 1 1 0 0 1 0
- - ако числото се използва в 8-битова машина, към него добавяме бит 6 в състояние 0 и знаковия бит 7 в старшия разред. Така получаваме 8-битовото положително двоично число 0 0 1 1 0 0 1 0

  бит 7 6 5 4 3 2 1 0.
- - Изваждаме това число от първата следваща по-голяма цяла степен на 2 - това е 9-битовото число 1 0000 0000

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
	1	1	1	1	1	1	1		<- заем
1	0	0	0	0	0	0	0	0	
-	0	0	1	1	0	0	1	0	положително число
	1	**1**	**0**	**0**	**1**	**1**	**1**	**0**	*2's comp* **отрицателно число**

При изваждането старшият бит 7 автоматично се установява в "1" и е указание, че полученото число е отрицателно.
Особеност: както в традиционната математика, умножбението на отрицателно число по -1 дава положителен резултат. Допълнението до 2 на едно отрицателно число, което вече е представено в този формат го превръща отново в положително число!

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
	1	1	1	1	1	1	1		<- заем
1	0	0	0	0	0	0	0	0	
-	1	1	0	0	1	1	1	0	*2's comp* отрицателно число
	0	**0**	**1**	**1**	**0**	**0**	**1**	**0**	*2's comp* **положително число**

Освен този бавен начин за получаване на допълнението до 2, съществуват и други, по-бързи за изпълнение и подаващи се на оптимизация алгоритми .

- Допълнението до 2 може да бъде получено от допълнението до 1 чрез инвертиране на всички битове на числото, включително знаковия и събиране на единица към полученото число (в младшия му разред). При събирането се взима предвид евентуалният пренос.

Нека вземем същото число + 50_{10} => % 0 0 1 1 0 0 1 0

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
							1		<- пренос
	0	0	1	1	0	0	1	0	оригинално число
	1	1	0	0	1	1	0	1	*1's comp* след инверсията
+								1	събиране на 1 към бит 0
	1	**1**	**0**	**0**	**1**	**1**	**1**	**0**	*2's comp*

- Друг начин, при който не е необходимо да се изпълняват аритметични, а само логически действия, е следният:

  в оригиналното (положително число) се запазват отдясно наляво състоянията на всички битове до срещане на първата единица включително. След нея се инвертира състоянието на всички по-старши битове.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	0	1	1	0	0	1	0	оригинално число
1	1	0	0	1	1	1	0	инвертиране на бит 2 до бит 7
1	**1**	**0**	**0**	**1**	**1**	1	0	*2's comp* резултат - (-) число

Беше споменато, че изваждането е операция, която често се избягва поради сложното и изпълнение. Тя се замества със събиране на положително с отрицателно число. Работата с числа със знак във формат "допълнение до 2" дава веднага правилен резултат, без допълнителни корекции. Изпълнява се по същия начин, както при събиране на числа без знак, с изключение на обработката на преноса, когато има такъв.

Пример 1: събиране на положително и отрицателно число с еднакви абсолютни стойности - очакван резултат *нула*.

		2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		
		1	1	1	1	1	1			<- пренос	
+ 50_{10}		0	0	1	1	0	0	1	0	*2's comp*	положително число
- 50_{10}	+	1	1	0	0	1	1	1	0	*2's comp*	отрицателно число
		0	**0**	**0**	**0**	**0**	**0**	**0**	**0**		резултат 0 след събирането

Пример 2: събиране на по-голямо положително с по-малко отрицателно число (по абс. ст-т) - очакван резултат *положително число*.

		2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		
		1			1					<- пренос	
+ 73_{10}		0	1	0	0	1	0	0	1	*2's comp*	положително число
- 22_{10}	+	1	1	1	0	1	0	1	0	*2's comp*	отрицателно число
+ 51_{10}	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**		резултат след събирането

преносът в несъществуващия 9-и бит се игнорира

Пример 3: събиране на по-малко положително с по-голямо отрицателно число (по абс. ст-т) - очакван резултат *отрицателно число*.

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
			1	1	1				<- пренос
+ 28_{10}	0	0	0	1	1	1	0	0	*2's comp* положително число
- 50_{10} +	1	1	0	0	1	1	1	0	*2's comp* отрицателно число
- 22_{10}	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	резултат след събирането

↖ Полученият резултат е отрицателно число и то е автоматично във формат *2's comp.*