Модули за числа с плаваща запетая

Автор: гл. ас. д-р инж. Любомир Богданов

ПРОЕКТ BG051PO001--4.3.04-0042
„Организационна и технологична инфраструктура за учене през целия живот и развитие на компетенции"
Проектът се осъществява с финансовата подкрепа на
Оперативна програма „Развитие на човешките ресурси",
съфинансирана от Европейския социален фонд на Европейския съюз
Инвестира във вашето бъдеще!

Европейски съюз

Европейски социален фонд

Съдържание

Представяне на числа в двоичен вид

Цифровите схеми могат да обработват само числа, представени с битове, които може да са 0 или 1.

Проблем:

*с 1 бит може да се представят само две числа, но за да върши нещо полезно, μPU трябва да може да обработва и по-големи числа, например 0 ÷ 10, 100 ÷ 200, 1000000 ÷ 3500000, и т.н.

Решение:

*да се използва група от битове, която представя числата в двоична бройна система. Колкото повече бита включва една група, толкова по-голямо десетично число може да представи тя.

Представяне на числа в двоичен вид

Прав код без знак (straight binary) – групата от битове се съпоставя директно на положителни числа от десетичната бройна система [1].

Decimal number	Binary number
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

Представяне на числа в двоичен вид

Проблем:

*Как може да се представят целочислени стойности със знак?

Решение:

*прав код (signed magnitude)

*обратен код (one’s complement)

*допълнителен код (two’s complement)

Проблем:

*Как може да се представят дробни числа (числа със запетая)?

Решение:

*Числа с фиксирана запетая

*Числа с плаваща запетая (IEEE 754-1985)

Представяне на числа в двоичен вид

Прав код (signed magnitude) – най-старшият бит (т.е. битът най-вляво) от двоичното число се заделя, за да се указва знак. Когато MSb = 1, значи че знакът е минус, а оставащите младши битове указват отрицателно число. Когато MSb = 0, знакът е плюс и числото е положително.

Проблем:
при такова представяне, в обхвата от всички числа има **две нули.**

1000.0000 (0-)
0000.0000 (0+)

Представяне на числа в двоичен вид

Работата с такива числа ще доведе до усложняване на хардуера, за да се преодолее този проблем.

Decimal number	8-bit binary signed magnitude number
-127	1111 1111
...	...
-3	1000 0011
-2	1000 0010
-1	1000 0001
-0	1000 0000
+0	0000 0000
1	0000 0001
2	0000 0010
3	0000 0011
...	...
127	0111 1111

Представяне на числа в двоичен вид

Обратен код (one’s complement) – отрицателните числа се представят с инвертирани битове на техния положителен еквивалент.

При събиране на две такива числа трябва да се следи за пренос (Carry). Ако има, този бит трябва да бъде добавен обратно към резултата, иначе числото ще е грешно.

Проблем:

при такова представяне, също има **две нули.**

1111.1111 (0-)

0000.0000 (0+)

Представяне на числа в двоичен вид

Decimal number	8-bit binary one's complement number
-127	1000 0000
…	…
-3	1111 1100
-2	1111 1101
-1	1111 1110
-0	1111 1111
+0	0000 0000
1	0000 0001
2	0000 0010
3	0000 0011
…	…
127	0111 1111

Представяне на числа в двоичен вид

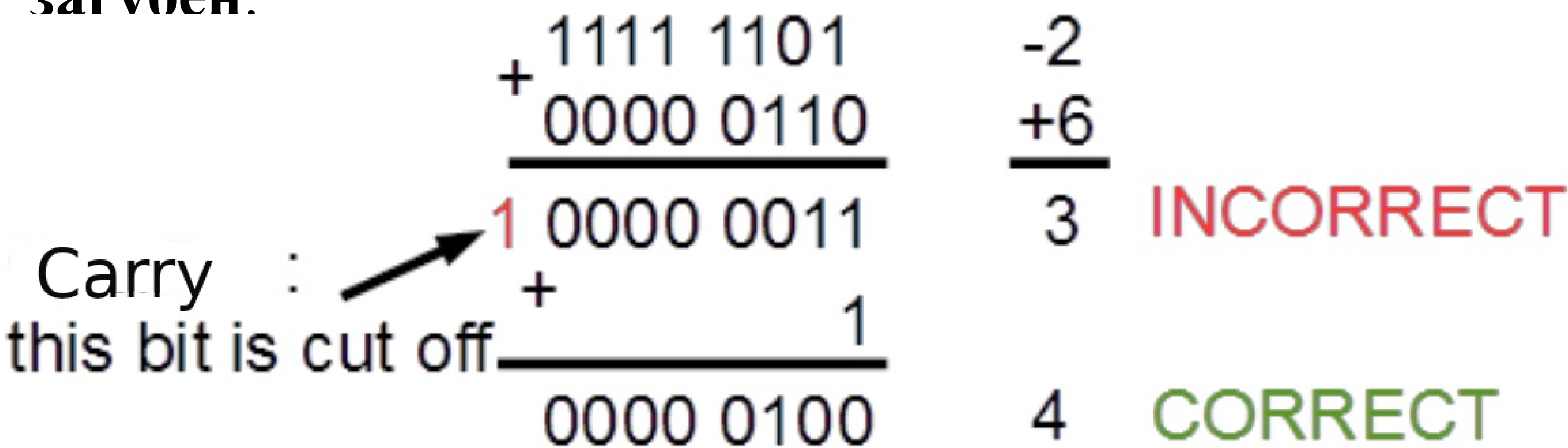
Пример – да се съберат числата -5 и +3. Нека те се представят с обратен код.

1111 1010	-5	
+ 0000 0011	+3	
1111 1101	-2	CORRECT

Представяне на числа в двоичен вид

Пример – да се съберат числата -2 и +6. Нека те се представят с обратен код.

ВНИМАНИЕ! Дължината на целочислените стойности е фиксирана (n = 4, 8, 16, 24, 32 бита). Всеки бит, който се пренесе на позиция n + 1 **бива загубен**.



Представяне на числа в двоичен вид

Допълнителен код (two's complement) – отрицателните числа се представят с инвертирани битове на техния положителен еквивалент и след това добавяне на числото 1.

При такова представяне **има само една нула** (0000.0000).

Аритметиката се извършва **със същия хардуер**, с който се извършва аритметиката на числа, представени с **прав код без знак**.

Компилаторът е отговорен за преобразуването на числата в допълнителен код, преди да се заредят в паметта.

Представяне на числа в двоичен вид

С 8-битов допълнителен код могат да се представят числата -128 ÷ +127 (докато с обратния код имаме неефективно ползване на битовете: -127 ÷ +127).

Допълнителният код е най-често използвания в цифровите системи.

Представяне на числа в двоичен вид

Decimal number	8-bit binary one’s complement number
-128	1000 0000
…	…
-3	1111 1101
-2	1111 1110
-1	1111 1111
0	0000 0000
1	0000 0001
2	0000 0010
3	0000 0011
…	…
127	0111 1111

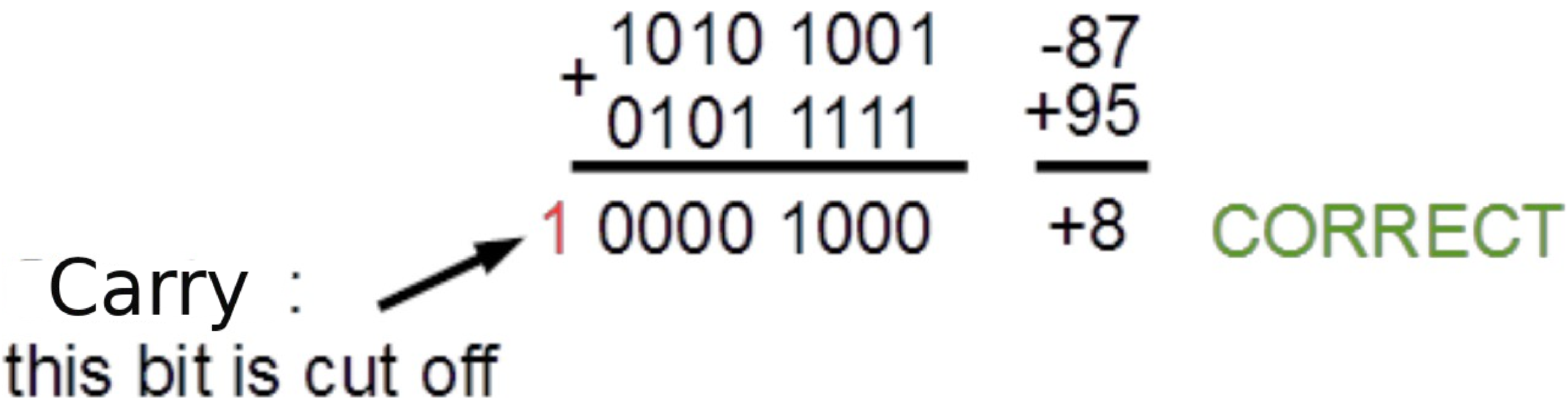
Представяне на числа в двоичен вид

Пример – да се изчисли допълнителния код на числото -87.

```
(bitwise invert)~0101 0111      +87
                 ---------
                 1010 1000
                +        1
                 ---------
                 1010 1001      -87
```

Представяне на числа в двоичен вид

Пример – да се изчисли сумата на числата в допълнителен код (-87) + (+95).



Забележка: всъщност Carry се съхранява в STATUS регистъра на μPU ядро и може да се използва за аритметика с числа с произволна разредност (софтуерно подсилване на μPU).

Представяне на числа в двоичен вид

Логическо преместване наляво/дясно (**L**ogical **S**hift **L**eft / **L**ogical **S**hift **R**ight) – добавяне на 0 към най-младшата/най-старшата част на едно число, която нула премества всички останали битове един индекс наляво/надясно.

Представяне на числа в двоичен вид

```
//0x32 = 50
uint8_t myvar = 0x32;

for(i = 0; i < 8; i++){
    myvar = myvar << 1;
}
```

LSL

(i[-1] 0050) 00110010

=================

(i[0] 0100) 0110010**0**

(i[1] 0200) 110010**00**

(i[2] 0144) 10010**000**

(i[3] 0032) 0010**0000**

(i[4] 0064) 010**00000**

(i[5] 0128) 10**000000**

(i[6] 0000) 0**0000000**

(i[7] 0000) **00000000**

Представяне на числа в двоичен вид

```c
//0x32 = 50
uint8_t myvar = 0x32;

for(i = 0; i < 8; i++){
    myvar = myvar >> 1;
}
```

LSR

(i[-1] 0050) 00110010

======================

(i[0] 0025) **0**0011001

(i[1] 0012) **00**001100

(i[2] 0006) **000**00110

(i[3] 0003) **0000**0011

(i[4] 0001) **00000**001

(i[5] 0000) **000000**00

(i[6] 0000) **0000000**0

(i[7] 0000) **00000000**

Представяне на числа в двоичен вид

Аритметично преместване наляво/дясно (**A**rithmetic **S**hift **L**eft / **A**rithmetic **S**hift **R**ight) – добавяне на 0 към най-младшата/най-старшата част на едно число, която нула премества всички останали битове един индекс наляво/надясно, **но знакът на числото се запазва при преместване надясно.** Преместването наляво не запазва знака, т.е. ASR = LSR.

Използват се числа в допълнителен код (two's complement).

Представяне на числа в двоичен вид

```
//0x32 = 50
int8_t myvar = 0x32;

for(i = 0; i < 8; i++){
    myvar = myvar << 1;
}
```

ASL+

(i[-1] 0050) 00110010

===================

(i[0] 0100) 0110010**0**

(i[1] -056) 110010**00**

(i[2] -112) 10010**000**

(i[3] 0032) 0010**0000**

(i[4] 0064) 010**00000**

(i[5] -128) 10**000000**

(i[6] 0000) 0**0000000**

(i[7] 0000) **00000000**

Представяне на числа в двоичен вид

```
//0x32 = 50
int8_t myvar = 0x32;

for(i = 0; i < 8; i++){
    myvar = myvar >> 1;
}
```

ASR+

(i[-1] 0050) 00110010

==================

(i[0] 0025) **0**0011001

(i[1] 0012) **00**001100

(i[2] 0006) **000**00110

(i[3] 0003) **0000**0011

(i[4] 0001) **00000**001

(i[5] 0000) **000000**00

(i[6] 0000) **0000000**0

(i[7] 0000) **00000000**

Представяне на числа в двоичен вид

```
int8_t myvar = -50;

for(i = 0; i < 8; i++){
    myvar = myvar << 1;
}
```

ASL-

(i[-1] -050) 11001110

====================

(i[0] -100) 1001110**0**

(i[1] 0056) 001110**00**

(i[2] 0112) 01110**000**

(i[3] -032) 1110**0000**

(i[4] -064) 110**00000**

(i[5] -128) 10**000000**

(i[6] 0000) 0**0000000**

(i[7] 0000) **00000000**

Представяне на числа в двоичен вид

```
int8_t myvar = -50;

for(i = 0; i < 8; i++){
    myvar = myvar >> 1;
}
```

В С **компилаторът избира** правилните инструкции за преместващия оператор в **зависимост от типа на променливата**.

ASR-

(i[-1] -050) 11001110

==================

(i[0] -025) **1**1100111

(i[1] -013) **11**110011

(i[2] -007) **111**11001

(i[3] -004) **1111**1100

(i[4] -002) **11111**110

(i[5] -001) **111111**11

(i[6] -001) **1111111**1

(i[7] -001) **11111111**

Представяне на числа в двоичен вид

Изводи:

*операторът за преместване може да се използва за **преобразуване** на **паралелна в последователна** информация

*операторът за преместване може да се използва за **умножение и деление**, НО:

→ умножението на числа без знак е валидно, докато не настъпи пренос;

→ умножението на отрицателни числа е валидно, докато не настъпи препълване (т.е. резултат > -128 за 8-битови числа, > -32768 за 16-битови числа, и т.н.);

→ делението на отрицателни числа е валидно, докато се стигне -1.

Представяне на числа в двоичен вид

Ротиране наляво/дясно (**RO**tate **L**eft/**R**ight) – изместване на най-старшия/най-младшия бит на едно число, и добавянето му в началото/края на числото, като всички останали битове се преместват един индекс наляво/надясно.

Представяне на числа в двоичен вид

```c
uint8_t myvar = 0x53;

for(i = 0; i < 8; i++){
 myvar = __rotate_left(myvar);
}
```

В С **няма оператор за ротиране,** :-(

но има inline Асемблер :-)
и intrinsic функции, :-)

с който може да се извикат директно инструкции за ротиране, ако се поддържат от микропроцесора.

ROL

Първоначално: **1**010 0011

======================

Итерация (0): 0100 011**1**
Итерация (1): 100 011**10**
Итерация (2): 0001 1**101**
Итерация (3): 0011 **1010**
Итерация (4): 011**1 0100**
Итерация (5): 11**10 1000**
Итерация (6): 1**101 0001**
Итерация (7): **1010 0011**

Представяне на числа в двоичен вид

```
uint8_t myvar = 0x53;

for(i = 0; i < 8; i++){
myvar = __rotate_right(myvar);
}
```

ROR

Първоначално: 1010 001**1**

=========================

Итерация (0): **1**101 0001

Итерация (1): **11**10 1000

Итерация (2): **011**10 100

Итерация (3): **0011**10 10

Итерация (4): **00011**10 1

Итерация (5): **1000 11**10

Итерация (6): **0100 011**1

Итерация (7): **1010 0011**

Представяне на числа в двоичен вид

Числа с фиксирана запетая (fixed point number) - двоични числа, които се съпоставят на дробни числа в десетична бройна система, при които разделителната способност (резолюцията) на числото преди и числото след запетаята е фиксирана. Такива числа може да се каже, че са фиксирани целочислени стойности, кратни на някакво малко число (напр. един час се сътои от 6 x 10 минутни части).

Хардуерният модул, който ще извършва изчисленията с тези числа, трябва да знае предварително колко бита са заделени за цялото число и колко за числото след запетаята.

Представяне на числа в двоичен вид

За числа с фиксирана запетая Qn се използва формулата [3]:

$$Qn(x_q) = x_i * 2^{-n}$$

където xi е цяло число, отговарящо на дробното число xq, а n е броя на битовете, заделени за числото, представящо дробната част.

Пример – нека разредността на **цялото число** (битове за целочислена + битове за дробна част) с фиксирана запетая да е **16 бита**. Ако за **дробна част** се заделят **12 бита**, **остават 4 бита за целочислената част**. Числото се отбелязва като Q12.

Представяне на числа в двоичен вид

Пример – числото 3.625, ако се представи с фиксирана запетая, ще бъде записано в паметта на контролера като $0011.\mathbf{101000000000}_{(2)} = 14848_{(10)}$. Това число се получава по следния начин:

$$Q12(3.625) = 14848*10^{-12}$$

откъдето се вижда, че **хардуерът предварително трябва да знае** за 10^{-12}

Представяне на числа в двоичен вид

Числата с фиксирана запетая може да се използват в много приложения, където изискванията за точността на дробните числа не са големи.

Числата с фиксирана запетая са със знак.

Texas Instruments са приготвили таблица с обхватите на всички възможни 16-битови числа с фиксирана запетая и съответните им резолюции (виж следващия слайд).

Представяне на числа в двоичен вид

Type	Bits		Range		Resolution
	Integer	Fractional	Min	Max	
_q15	1	15	-1	0.999 970	0.000 030
_q14	2	14	-2	1.999 940	0.000 061
_q13	3	13	-4	3.999 830	0.000 122
_q12	4	12	-8	7.999 760	0.000 244
_q11	5	11	-16	15.999 510	0.000 488
_q10	6	10	-32	31.999 020	0.000 976
_q9	7	9	-64	63.998 050	0.001 953
_q8	8	8	-128	127.996 090	0.003 906
_q7	9	7	-256	255.992 190	0.007 812
_q6	10	6	-512	511.984 380	0.015 625
_q5	11	5	-1,024	1,023.968 750	0.031 250
_q4	12	4	-2,048	2047.937 500	0.062 500
_q3	13	3	-4,096	4,095.875 000	0.125 000
_q2	14	2	-8,192	8,191.750 000	0.250 000
_q1	15	1	-16,384	16,383.500 000	0.500 000

Представяне на числа в двоичен вид

От тази таблица се вижда основния недостатък – **максималното и минималното число**, което може да се представи **е ограничено от разредността на дробната част**.

При числата с плаваща запетая, ако дробната част е малка, то целочислената стойност може да е голяма. И обратното – малки целочислени стойности ще позволят представянето на много знаци след запетаята.

Представяне на числа в двоичен вид

Числа с плаваща запетая (floating point numbers) – двоични числа, които се съпоставят на дробни числа в десетична бройна система. Съществуват различни стандарти, но най-често използвания е **IEEE754-1985,** който гласи:

Едно 32-битово дробно число се представя със следните битови полета:
*1 бит за **знак**
*8 бита за **експонента**
*23 бита за **мантиса** (mantissa, significand)

Представяне на числа в двоичен вид

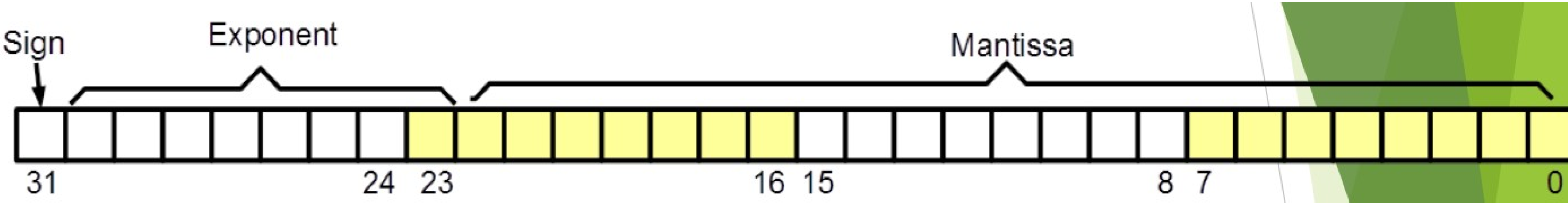
Числото +4.567 може да се представи изцяло с целочислени стойности: $+4567 \times 10^{-3}$

$+ \rightarrow$ знак

$4567 \rightarrow$ мантиса

$10^{-3} \rightarrow$ експонента -3 с основа 10

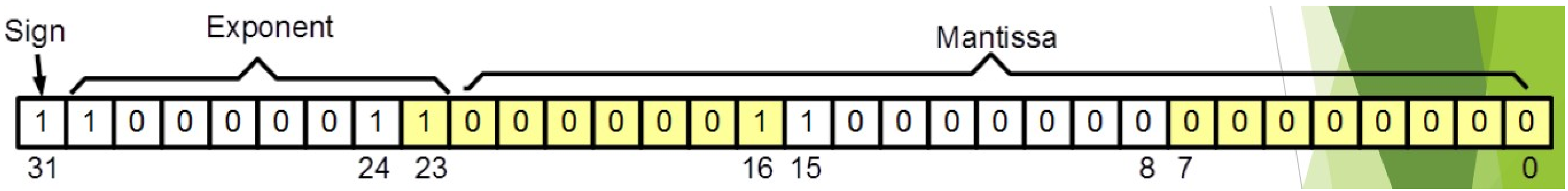
Представяне на числа в двоичен вид



$$float = (-1)^{sign} \times 2^{\mathrm{exp}onent-127} \times \left(1 + \sum_{n=1}^{23} bit_{23-n} \times 2^{-n}\right)$$

Представяне на числа в двоичен вид

Пример – намерете десетичния еквивалент на числото, представено с IEEE754-1985 дробно число:



*Знак: $(-1)^1 = -1$

*Експонента: $10000011_{(2)}=131_{(10)} \rightarrow 2^{131-127}=2^4$

*Мантиса: $1 + 0x2^{-1} + 0x2^{-2} + 0x2^{-3} + 0x2^{-4} + 0x2^{-5} + 0x2^{-6} + 1x2^{-7} + 1x2^{-8} + \ldots = 1 + 1/2^7 + 1/2^8 = 1.01171875$

*Резултат: $-1 \text{ x } 2^4 \text{ x } 1.01171875 = -16.1875$

*Или използвайте онлайн конвертора [2] :-)

Представяне на числа в двоичен вид

Barrel shifter

Литература

[1] Jason Albanus, “Coding Schemes Used with Data Converters”, SBAA042A, Texas Instruments, 2015.
[2] https://www.h-schmidt.net/FloatConverter/IEEE754.html
[3] “MSP430 IQmathLib Users Guide”, v1.10.00.05, 2015.