Задача 1.

1	Преобразовать числа 31, 141, -246, -92, 3715 в десятичном представлении в прямой, обратный и дополнительный код двоичной системы. Для последнего числа использовать переход с использованием основания 8 или 16
	Совершить переход от двоичной системы чисел представленных в однобайтном обратном дополнительном коде в десятичную систему 1001 1101 0001 0011 1110 0100

Теория

В вычислительных устройствах численная информация может быть представлена в трех вариантах: прямом, обратном и дополнительных кодах. Прямой код обычно используется для хранения и передачи между устройствами. Обратный и дополнительных коды чаще можно встретить непосредственно в оперативной памяти непосредственно в процессе обработки.

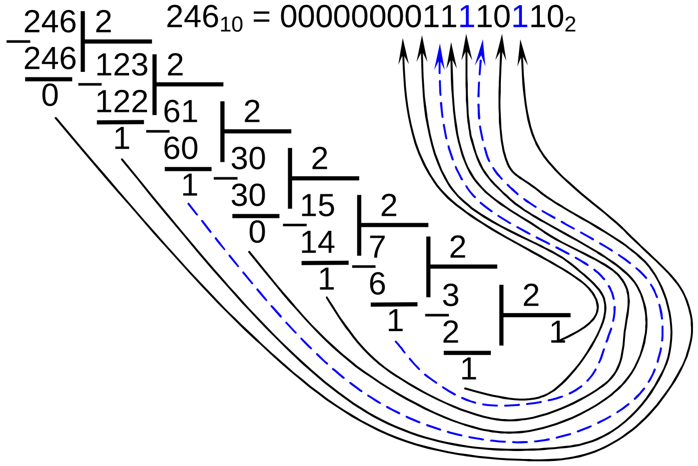
При выполнении задания должен быть представлен непосредственно процесс получения двоичного кода. Двоичный код можно получить разными способами, например делением и вычитанием.

Информация в вычислительных устройствах хранится в регистрах, которые могут быть 8-, 16-, 32- и т. д. бит. Поэтому при формировании ответа надо выбирать один из этих вариантов. Т.е. старшие биты определять нулями. В конкретных вычислительных устройствах разрядность регистров имеет фиксированный размер.

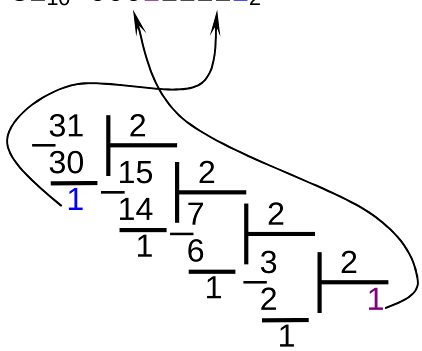
Деление.

Суть метода в делении числа одного кода на новое основание пока остаток не окажется меньше нового основания.

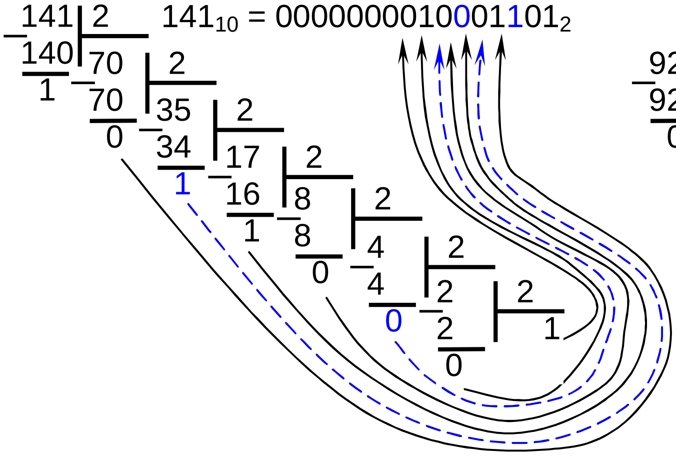
$246_{10} = 0000000011110110_2$



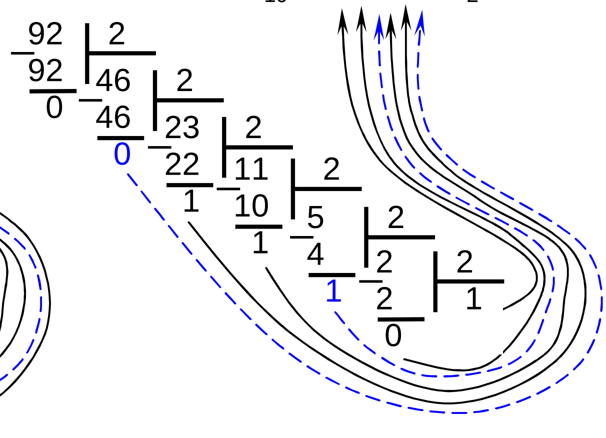
$31_{10}=00011111_2$



$141_{10} = 0000000010001101_2$



$92_{10} = 01011100_2$



В примерах для чисел 141 и 246 использовалось два байта так как с помощью одного их записать нельзя. А 31 и 92 с помощью одного. Но записав их с помощью двух байт (16 бит) тоже будет правильно.

Вычитание.

При вычитании надо помнить ряд числа 2 в степенях от нуля и до 10 хотя бы с учетом диапазона чисел задания. После это занимаемся разложение чисел задания на члены этого ряда

2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1024	512	256	128	64	32	16	8	4	2	1
						−31	−15	−7	−3	−1
						16	8	4	2	1
						15	7	3	1	0
						1	1	1	1	1

Тогда для одного байта $31_{10} = 000111111_2$

2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1024	512	256	128	64	32	16	8	4	2	1
			−141				−13	−5		−1
			128				8	4		1
			13				5	1		0
			1	0	0	0	1	1	0	1

Тогда для двух байт 141_{10} = 0000000010001101_2

2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1024	512	256	128	64	32	16	8	4	2	1
				−92		−28	−12	−4		
				64		16	8	4		
				28		12	4	2		
		92_{10} =	0	1	0	1	1	1	0	0

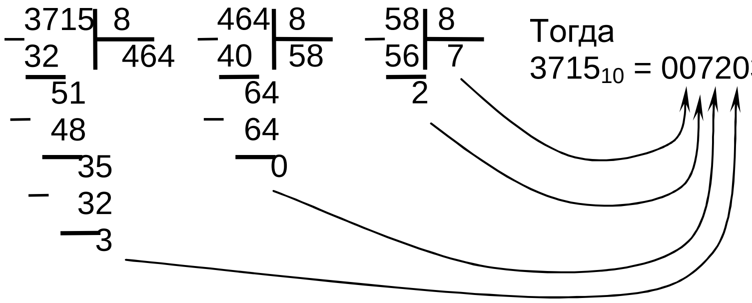
Тогда для одного байта $94_{10} = 01011100_2$

2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1024	512	256	128	64	32	16	8	4	2	1
			−246	−118	−54	−22		−6	−2	
			128	64	32	16		4	2	
			118	54	22	6		2	0	
			1	1	1	1	0	1	1	0

Тогда для двух байт 246_{10} = 0000000011110110_2

Переход через другое основание.

Для разложения последнего числа надо использовать деление на 8 или 16. Запись восьмеричного числа аналогична двоичному: последний остаток записывается первым, первый – последний. Для записи двухбайтного числа в восьмеричном коде используют шесть знаков. Старший знак используются под знак. Остальные триады это веса числа. Далее двоичный код получаем непосредственно представляя отдельный числа восьмеричного кода с помощью трех бит двоичного кода.

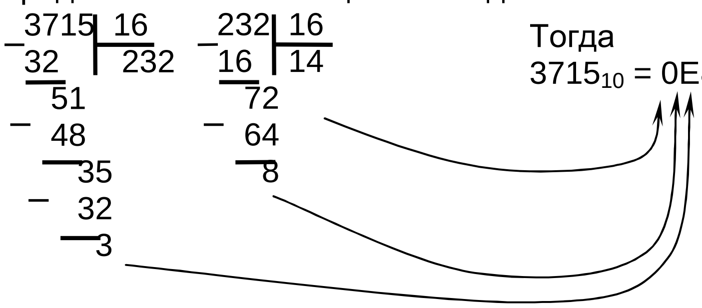


Тогда

$3715_{10} = 007203_{8} = 0\ 000\ 111\ 010\ 000\ 011_{2}$

0 0 7 2 0 3

Использование шестнадцатеричного кода аналогично восьмеричного. Отличие только в необходимости использовать числа больше 9 с помощью букв. 10 –A, 11 – B, 12 – C, 13 – D, 14 – E, 15 – F. Для записи двух байт шестнадцатеричного кода должно быть 4 символа. Дальше каждый символ шестнадцатеричного кода представляем с помощью 4 бит двоичного.



Тогда

$3715_{10} = 0E83_{16} = 0000\ 1110\ 1000\ 0011_{2}$

0 E 8 3

Запись чисел в разных кодах

Дальше остается только записать числа в прямом, обратном и дополнительном коде. Для положительных чисел код совпадает с темы значениями, что уже получены.

$31_{10} = 000111111_{2} = 000111111_{ПК} = 000111111_{ОК} = 000111111_{ДК}$

$141_{10} = 0000000010001101_{2} = 0000000010001101_{ПК} = 0000000010001101_{ОК} =$
$=0000000010001101_{ДК}$

$3715_{10} = 0000111010000011_{2} = 0000111010000011_{ПК} = 0000111010000011_{ОК} =$
$= 0000111010000011_{ДК}$

Для отрицательно числа в прямом кода на 8 или 16 месте должна появится 1. Зависит от выбранной вами разрядности. Для обратного кода необходимо поменять все биты кроме бита знака на противоположные. Для дополнительного кода необходимо увеличить обратный на 1. т.е фактически математически прибавить 1.

$–246_{10} =–(0000000011110110_{2}) = 1000000011110110_{ПК} = 1111111100001001_{ОК} =$
$= 1111111100001010_{ДК}$

$–92_{10} =–(01011100_{2}) = 11011100_{ПК} = 10100011_{ОК} = 10100100_{ДК}$

Вторая часть задания.

Обратное преобразование двоичного кода в десятичный. В задание все числа даны в дополнительном коде при однобайтном представлении. В первую очередь следует обратить на восьмой бит. Это бит знака. Он определяет преобразование. С числами с нулем впереди можно сразу осуществлять преобразование помня ряд степеней с основанием два

$0001\ 0011_{2} = +(0 \cdot 2^{6} + 0 \cdot 2^{5} + 1 \cdot 2^{4} + 0 \cdot 2^{3} + 0 \cdot 2^{2} + 1 \cdot 2^{1} + 1 \cdot 2^{0}) = 19_{10}$

Для других чисел необходимо сделать преобразование. Есть два варианта:

1. От исходного числа отнять единицу. Таким образом получить обратный код. Затем все биты кроме бита знака поменять на противоположный

$1001\ 1101_{\text{ДК}} = 1001\ 1101_{\text{ДК}} - 0000\ 0001 = 1001\ 1100_{\text{ОК}} = 1110\ 0011_{\text{ПК}} =$

$1110\ 0011_{\text{ПК}} = -(1 \times 64 + 1 \times 32 + 0 \times 16 + 0 \times 8 + 0 \times 4 + 1 \times 2^1 + 1 \times 1) = -99_{10}$

$1110\ 0100_{\text{ДК}} = 1110\ 0100_{\text{ДК}} - 0000\ 0001 = 1110\ 0011_{\text{ОК}} = 1001\ 1100_{\text{ПК}} =$

$1001\ 1100_{\text{ПК}} = -(0 \times 64 + 0 \times 32 + 1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2^1 + 0 \times 2^0) = -28_{10}$

2. Сначала поменять все биты на противоположные кроме бита знака. А потом прибавить единицу

$1001\ 1101_{\text{ДК}} = 1110\ 0010 = 1110\ 0010 + 0000\ 0001 = 1110\ 0011_{\text{ПК}}$.

$1110\ 0100_{\text{ДК}} = 1001\ 1011 = 1001\ 1011 + 0000\ 0001 = 1001\ 1100_{\text{ПК}}$

При использовании второго варианта студенты делают меньше ошибок

Задача 2

1.	A	B	D	$C_1 = A + B$, $C_2 = A - B$, $C_3 = –A + B$, C_4 = D + B.
	111	35	47,48	Вычислить C_1, C_3, C_4 в двоичном дополнительном коде, а C_2 в обратном (размер байт). Представить: C_1 в прямом коде; C_4 – в восьмеричном; C_3 – в 16-ричном; C_2 – в двоично-десятичном. Указать случаи переполнения

Преобразование в двоичный код

Первый этап – получение двоичного кода. Путь получения – любой метод. В данном примере двоичный код рассчитан через восьмеричный код.

$$\begin{array}{r|l} 111 & 8 \\ \hline -8 & 13 \\ 31 & \\ -24 & \\ 7 & \end{array} \qquad \begin{array}{r|l} 13 & 8 \\ \hline -8 & 1 \\ 5 & \end{array} \qquad 111_{10} = 157_8 = 01\ 101\ 111_2$$

$$\begin{array}{r|l} 35 & 8 \\ \hline -32 & 4 \\ 3 & \end{array} \qquad 35_{10} = 43_8 = 00\ 100\ 011_2$$

Третье число преобразуем по частям. Отдельно часть что до запятой и что после

$$\begin{array}{r|l} 47 & 8 \\ \hline -40 & 5 \\ 7 & \end{array} \qquad 47_{10} = 57_8 = 00\ 101\ 111_2$$

Для десятичной достаточно четырех бит. И в целом число *D* будет занимать полтора байта.

Числа после запетой можно получать путем умножения на два или разложение с использованием ряда 2 с отрицательными степенями. Важным аспектом является округление, поскольку процесс преобразования часто не имеет окончания. Для получения полбайта надо будет получить 5 бит. Если последний пятый бит будет 1, то его добавляю к предыдущий, если ноль то предыдущие биты оставляют без изменения.

$$\times\begin{array}{r} 0,48 \\ 2 \\ \hline 0,96 \end{array} \quad \times\begin{array}{r} 0,96 \\ 2 \\ \hline 1,92 \end{array} \quad \times\begin{array}{r} 0,92 \\ 2 \\ \hline 1,84 \end{array} \quad \times\begin{array}{r} 0,84 \\ 2 \\ \hline 1,68 \end{array} \quad \times\begin{array}{r} 0,68 \\ 2 \\ \hline 1,36 \end{array}$$

$$\times\begin{array}{r} 0,19 \\ 2 \\ \hline 0,38 \end{array} \quad \times\begin{array}{r} 0,38 \\ 2 \\ \hline 0,76 \end{array} \quad \times\begin{array}{r} 0,76 \\ 2 \\ \hline 1,52 \end{array} \quad \times\begin{array}{r} 0,52 \\ 2 \\ \hline 1,04 \end{array} \quad \times\begin{array}{r} 0,04 \\ 2 \\ \hline 0,08 \end{array}$$

$$\times\begin{array}{r} 0,72 \\ 2 \\ \hline 1,44 \end{array} \quad \times\begin{array}{r} 0,44 \\ 2 \\ \hline 0,88 \end{array} \quad \times\begin{array}{r} 0,88 \\ 2 \\ \hline 1,76 \end{array} \quad \times\begin{array}{r} 0,76 \\ 2 \\ \hline 1,52 \end{array} \quad \times\begin{array}{r} 0,52 \\ 2 \\ \hline 1,04 \end{array}$$

$0,47_{10}$ » 01111_2 » 1000_2 После округления

$0,19_{10}$ » 00110_2 » 0011_2 После округления

$0,72_{10}$ » 10111_2 » 1100_2 После округления

2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
0,5	0,25	0,125	0,0625	0,03125
	$\begin{array}{r} 0{,}48 \\ -0{,}25 \\ \hline 0{,}23 \end{array}$	$\begin{array}{r} 0{,}23 \\ -0{,}125 \\ \hline 0{,}105 \end{array}$	$\begin{array}{r} 0{,}105 \\ -0{,}0625 \\ \hline 0{,}0425 \end{array}$	$\begin{array}{r} 0{,}0425 \\ -0{,}03125 \\ \hline 0{,}01125 \end{array}$
0	1	1	1	1

Тогда $0{,}48_{10}$ » 01111_2 » 1000_2

2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
0,5	0,25	0,125	0,0625	0,03125
		$\begin{array}{r} 0{,}19 \\ -0{,}125 \\ \hline 0{,}065 \end{array}$	$\begin{array}{r} 0{,}065 \\ -0{,}0625 \\ \hline 0{,}0025 \end{array}$	
0	0	1	1	0

Тогда $0{,}19_{10}$ » 01100_2 » 0110_2

2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
0,5	0,25	0,125	0,0625	0,03125
$\begin{array}{r} 0{,}72 \\ -0{,}5 \\ \hline 0{,}22 \end{array}$		$\begin{array}{r} 0{,}22 \\ -0{,}125 \\ \hline 0{,}095 \end{array}$	$\begin{array}{r} 0{,}095 \\ -0{,}0625 \\ \hline 0{,}0325 \end{array}$	$\begin{array}{r} 0{,}0325 \\ -0{,}03125 \\ \hline 0{,}00125 \end{array}$
1	0	1	1	1

Тогда $0{,}72_{10}$ » 10111_2 » 1100_2

С учетом преобразований число D в размере полтора байта будет иметь вид:
$D = 47{,}48 = 00101111\ 1000_2$.
В записи присутствую пробелы, но это для удобства. В реальный вычислительных системах с фиксированной точной никаких отделений нет. И правила суммирования применяются в общем виде.

Теория.

Размер сумматора в микропроцессоре всегда имеет ограничение на длину бит числа которые он может сложить. Исходя из этого, может наблюдаться, что получаемое значение будет выходить за отведенные рамки. Для выявления таких проблем существует механизм переполнения, который заключается в том, что перед непосредственно проведением операции суммирования бит знака удваивается. Если в результате суммирования эти два бита получаются одинаковые, значит суммирование прошло корректно, если нет, то образовалось переполнение.
Современные процессора редко содержат одновременно возможность суммирования и вычитания. Присутствует только одна команда. А противоположную операцию заменяю путем изменения знака числа.

$C_1 = A + B$

Значения чисел равны A = 01101111; B = 00100011_2. В модифицированном коде это будет A = 00.1101111; B = 00.0100 011. Точка несет вспомогательную роль для понимания. Поскольку числа положительные, то процесс сложения не отличается прямой, обратный или дополнительный это код.
В каждом разряде может быть 4 ситуации. Если в разряде один ноль, а другая единица, то результатом будет 1. Если оба нуля, то результат 0. Если две единицы, то результат в этом разряде 0, но одна единица переходит в следующий

разряд. Если две единицы и еще третья с предыдущего разряда, то результат в этом разряде 1, одна единица переходит в следующий разряд. Переход единицы в следующий разряд обычно обозначается точкой.

$$\begin{array}{rl}
 & \cdot\quad\cdot\quad\cdot\ \cdot\ \cdot\ \cdot \\
 & 00.1101111_{\text{дк}} \quad (A) \\
+ & 00.0100011_{\text{дк}} \quad (B) \\
\hline
 & 01.0010010_{\text{дк}}
\end{array}$$

Данный размер демонстрирует переполнение

C*₂ = *A* – *B

C_2 должно быть вычислено в обратном коде. Поэтому выражение преобразуем для исключения вычитания $C_2 = A + (-B)$ и преобразуем *B* в обратный код. Поскольку *A* положительное, преобразования не требуется. Если при выполнении суммирования в обратном коде получается 1 за рамками разрядной сетки, то ее следует прибавить к младшему биту.

$-B = -(000100011) = 11.1011100_{\text{ОК}}$.

$$\begin{array}{rl}
 & \cdot\ \cdot\ \cdot\quad\cdot\ \cdot\ \cdot\ \cdot \\
 & 00.1101111_{\text{ОК}} \quad (A) \\
+ & 11.1011100_{\text{ОК}} \quad (-B) \\
\hline
 & 100.1001011_{\text{ОК}} \\
+ & 1_{\text{ОК}} \\
\hline
 & 00.1001100_{\text{ОК}}
\end{array}$$

Как видим, получилось положительное число. Переполнение отсутствует.
Для примера рассмотрим вариант сложения в обратном коде для *B – A.*

$-A = -(00.1101111) = 11.0010000_{\text{ОК}}$.

$$\begin{array}{rl}
 & 00.0100011_{\text{ОК}} \quad (B) \\
+ & 11.0010000_{\text{ОК}} \quad (-A) \\
\hline
 & 11.0110011_{\text{ОК}}
\end{array}$$

Получилось отрицательное число. Переполнение тоже отсутствует.

C*₃ = –*A* + *B

C_3 должно быть вычислено в дополнительном коде. Поэтому выражение преобразуем для исключения вычитания $C_2 = (-A) + B$ и преобразуем *A* в дополнительный код. Поскольку число *B* положительное, преобразования не требуется. Если при выполнении суммирования в дополнительном коде получается 1 за рамками разрядной сетки, то ее следует игнорировать.

$-A = -(00.1101111) = 11.0010001_{\text{ДК}}$.

$$\begin{array}{rl}
 & \cdot\ \cdot \\
 & 11.0010001_{\text{ДК}} \quad (-A) \\
+ & 00.0100011_{\text{ДК}} \quad (B) \\
\hline
 & 11.0110100_{\text{ДК}}
\end{array}$$

Как видим, получилось отрицательное число. Переполнение отсутствует.
Для примера рассмотрим вариант сложения в обратном коде для *– B + A.*

$-B = -(000100011) = 11.1011101_{\text{ДК}}$.

$$\begin{array}{rl}
 & \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot \\
 & 11.1011101_{\text{ДК}} \quad (-B) \\
+ & 00.1101111_{\text{ДК}} \quad (A) \\
\hline
 & \cancel{1}00.1001100_{\text{ДК}}
\end{array}$$

Получилось положительное число. Переполнение тоже отсутствует.

$C_4 = D + B$

Отличительной особенностью данного суммирования, что надо правильно расположить целую часть над целой.

$$
\begin{array}{rl}
 & \cdot\ \ \cdot\cdot\cdot\ \cdot \\
+ & 00.0100011000\,0_{ДК}\quad (B) \\
 & 00.0101111100\,0_{ДК}\quad (D) \\
\hline
 & 00.1010010100\,0_{ДК}
\end{array}
$$

Формирование окончательного результата

Представить: C_1 в прямом коде; C_4 – в восьмеричном; C_3 – в 16-ричном; C_2 – в двоично-десятичном.

C_1 у нас получился с переполнение, поэтому результат на выходе может быть разный. Одни процессора «выбросят» флаг и потребую увеличения разрядной сетки. Тогда единицу в поле знака можно интерпретировать как весовой коэффициент 128. Старые процессора рассмотрят единицу как признак отрицательного числа.

$C_1 = 01.0010010_{ДК}$ => 00000000 10010010 = (128 + 16 + 2) = 146.

$C_1 = 01.0010010_{ДК} = 10010010_{ДК} = 10010001_{ОК} = 11101110_{ПК} = –(64 + 32 + 8 + 4 + 2) =$
$= –110.$

Получать десятичные числа не обязательно. Вычисления проведены для понимания проблемы переполнения.

$C_4 = 00.10100101000 = 001\ 010\ 010\ 100\ 000 = 12240_8$

Для записи восьмеричного кода отдельно для целой части добавили количество нулей для кратности 3 битам. Последние три бита помечены синим цветом. А дальше три бита представлены с помощью десятичных чисел.

$C_3 = 11.0110100_{ДК} = 1011\ 0100_2 = B4_{16}$

Любой редактор шестнадцатеричного кода осуществляет представление не осуществляя преобразование в прямой код или еще куда.

Для представления числа в двоично-десятичном (BCD) необходимо получить десятичный код, а затем осуществить преобразование каждой десятичной цифры в двоичный код. двоично-десятичном код занимает обычно не менее двух байт.

$C_2 = 00.1001100_{ОК} = (64 + 8 + 4) = 76_{10} = 0000\ \ 0000\ 0111\ 0100_{2\text{-}10}$

0 0 7 6

В случае отрицательного числа под знак отводится все четыре бита

$–109_{10} = 1111\ 0001\ 0000\ 1001_{BCD}$

– 1 0 9

Задача 3

Теория

Комментарии. Положительная оценка получается, если операции сложения и вычитания выполнены в двоично-десятичном коде (BCD). Выполнение в двоичном коде не принимается.

Вторую часть со сдвигами выполнять в модифицированном коде. Даже если это на прямую не указывается.

Если двух байт не хватает, можно размер хранения увеличить до 4.

Двоично-десятичный код предусматривает запись каждого числа десятичного числа с помощью отдельных независимых тетраед. На знак используется вся старшая тетраеда, но в задаче такой результат исключен.

Сложения и вычитание выполняются по правилам двоичного кода. Используется стандартные возможности АЛУ, но из-за несовпадения кодировок требуется коррекция. Коррекция всегда производится с помощью 0110. Кодировка проводится в тех разрядах, которые в результате операций привели к появлению в тетраеде кода числа превышающего 1001. Коррекция необходима, когда происходит перенос бита между соседними тетраедами.

Операцию вычитания производим без использования обратного или обратного дополнительного кода.

При выполнении сдвига и вправо, и влево происходит перемещение позиции каждого бита на размер сдвига в нужном направлении. При сдвиге вправо крайние биты отбрасываются, однако последний отбрасываемый бит используется для процедуры округления. Для положительных чисел и прямого кода если уходит последней 1, то она добавляется

<table>
<tr><td colspan="4">2</td><td colspan="4">8</td><td colspan="4">0</td><td colspan="4">5</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
</table>

<table>
<tr><td colspan="4">–</td><td colspan="4">9</td><td colspan="4">6</td><td colspan="4">7</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr>
</table>

Сложение в формате BCD

Условие примера

<table>
<tr><td rowspan="2">2.</td><td>A</td><td>B</td><td>D</td><td>Представить числа в двоично-десятичном коде и произвести</td></tr>
<tr><td>5985</td><td>2674</td><td>1394</td><td>вычисления $C_1 = A + B$, $C_2 = A - D$. Представить результат в десятичном виде. Представить $-C_2$ в модифицированном двоичном дополнительной коде (размер два байта)и выполнить сдвиг влево 3 и право на 4. C_2 в двоичном модифицированном прямом коде и выполнить сдвиг влево на 4 и право на 3.</td></tr>
</table>

Решение

Результат преобразования в BCD

<table>
<tr><td colspan="4">5</td><td colspan="4">9</td><td colspan="4">8</td><td colspan="4">5</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
</table>

<table>
<tr><td colspan="4">2</td><td colspan="4">6</td><td colspan="4">7</td><td colspan="4">4</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr>
</table>

1				3				9				4			
0	0	0	1	0	0	1	1	1	0	0	1	0	1	0	0

Сложение

				.
+	0 1 0 1	1 0 0 1	1 0 0 0	0 1 0 1
	0 0 1 0	0 1 1 0	0 1 1 1	0 1 0 0
	0 1 1 1	1 1 1 1	1 1 1 1	1 0 0 1

Требуется коррекция в во второй и третье тетраедах

	. . .	. . . .	. .	
+	0 1 1 1	1 1 1 1	1 1 1 1	1 0 0 1
		0 1 1 0	0 1 1 0	
	1 0 0 0	0 1 1 0	0 1 0 1	1 0 0 1

Полученное число 8659 соответствует результату обыкновенных правил математики.

Вычитание

		* *	*	
–	0 1 0 1	1 0 0 1	1 0 0 0	0 1 0 1
	0 0 0 1	0 0 1 1	1 0 0 1	0 1 0 0
	0 1 0 0	0 1 0 1	1 1 1 1	0 0 0 1

Требуется коррекция в третьем разряде

–	0 1 0 0	0 1 0 1	1 1 1 1	0 0 0 1
			0 1 1 0	
	0 1 0 0	0 1 0 1	1 0 0 1	0 0 0 1

После коррекции 4591 результат соответствует правилам математики

Для выполнения операций сдвига получим двоичный код числа 4591 через восьмеричный код

– 4591	8	– 573	8	– 71	8	– 9	8
40	573	56	71	64	9	8	1
– 59		– 13		7		1	
56		8					
– 31		5					
24							
7							

Число $4591_{10} = 11757_8 = 0.001\ 001\ 111\ 101\ 111_2$

Соответственно в модифицированном коде

$4591_{10} = 11757_8 = 00.001\ 001\ 111\ 101\ 111_2$

Тогда

$–C_2 = –4591_{10} = 11.001\ 001\ 111\ 101\ 111_{ПК} = 11.110\ 110\ 000\ 010\ 000_{ОК} =$
$= 11.110\ 110\ 000\ 010\ 001_{ДК}$

Сдвинем $–C_2$

Теория сдвига и округление

Положительные числа в прямом, обратном и дополнительном коде.

При сдвиге происходит изменение числа. Сдвиг вправо приводит к уменьшению значения в 2^n, где n – где величина сдвига. Сдвиг влево приводит, соответственно, к увеличению в 2^n. Это является основной цель операции сдвига.

При выполнении сдвига существуют ограничения. Сдвиги на n позиций осуществляется не сразу. Операция проводится последовательно микропроцессором с контролем корректности выполнения операции. При сдвиге влево происходит контроль переполнения. В случае его обнаружения с помощью модифицированного кода, операция сдвига останавливается. Происходит запуск специального алгоритма отработки возникшей ошибки или просто формируется сообщение об ошибке. При сдвиге вправо происходит контроль обнуления числа. Дальше сдвиг не производится. В случае сдвига вправо, после последнего такта сдвига, производится процедура округления на основании отброшенных битов.

При сдвиге положительных чисел правила для прямого, обратного и дополнительного кодов кода одинаковы. При необходимости сдвига вправо или влево на n позиций, происходит, соответственно, последовательное выпадения n битов справа или слева. Остальные биты передвигаются с сохранением той же последовательности на освобождающие места. Крайние биты заполняются нулями. В простейшем случае процедура округления осуществляется математическим прибавлением 1, если на последнем такте была отброшена 1. Смотри две следующие таблицы

Единица, отбрасывания которой, требует добавление единицы к младшему биту, обозначена красным цветом

Исходное десятичное значение	Исходный прямой код	Сдвиг на 1 вправо (треб. по округ)	Результат с учетом округления	Десятичный эквивалент	Результат деления на 2
31	00.0011111	00.0001111(+1)	00.0010000	16	15,5
30	00.0011110	00.0001111	00.0001111	15	15
29	00.0011101	00.0001110(+1)	00.0001111	15	14,5
28	00.0011100	00.0001110	00.0001110	14	14
27	00.0011011	00.0001101(+1)	00.0001110	14	13,5
26	00.0011010	00.0001101	00.0001101	13	13
25	00.0011001	00.0001100(+1)	00.0001101	13	12,5
24	00.0011000	00.0001100	00.0001100	12	12
23	00.0010111	00.0001011(+1)	00.0001100	12	11,5

Все результаты соответствуют правилам математики

Исходное десятичное значение	Исходный прямой код	Сдвиг на 3 вправо (треб. по округ)	Результат с учетом округления	Десятичный эквивалент	Результат деления на 8
31	00.0011111	00.0000011(+1)	00.0000100	4	3,875
30	00.0011110	00.0000011(+1)	00.0000100	4	3,75
29	00.0011101	00.0000011(+1)	00.0000100	4	3,625
28	00.0011100	00.0000011(+1)	00.0000100	4	3,5
27	00.0011011	00.0000011	00.0000011	3	3,375
26	00.0011010	00.0000011	00.0000011	3	3,25
25	00.0011001	00.0000011	00.0000011	3	3,125
24	00.0011000	00.0000011	00.0000011	3	3
23	00.0010111	00.0000010(+1)	00.0000011	3	2,875

Как мы видим, соблюдение выше указанного правила округления позволяет операции сдвига, соответствовать правилам математики.

Сдвиг влево. При однобайтном хранении чисел сдвиг чисел больше чем на 3 разряда приводит к переполнению для десятичных эквивалентов больше 15.

Исходное десятичное значение	Исходный прямой код	Сдвиг на 1 влево	Десятичный эквивалент	Сдвиг на влево 3	Десятичный эквивалент
31	00.0011111	00.0111110	62	01.111100	переполнение

30	00.0011110	00.0111100	60	01.111000	переполнение
29	00.0011101	00.0111010	58	01.110100	переполнение
28	00.0011100	00.0111000	56	01.110000	переполнение
27	00.0011011	00.0110110	54	01.101100	переполнение
16	00.0010000	00.0100000	32	01.000000	переполнение
15	00.0001111	00.0011110	30	00.111100	90
14	00.0001110	00.0011100	28	00.111000	86
13	00.0001101	00.0011010	26	00.110100	82

Прямой код отрицательных чисел

При сдвиге в право отрицательных чисел в прямом коде, кроме контроля за обнулением, осуществляют контроль округления. Как и для положительных чисел, при отбрасывании в прямом коде на последнем такте сдвига 1, производится прибавлении ее к младшему биту.

Исходное десятичное значение	Исходный прямой код	Сдвиг на 1 вправо (треб. по округ)	Результат с учетом округления	Десятичный эквивалент	Результат деления на 2
–31	11.0011111	11.0001111(+1)	11.0010000	–16	–15,5
–30	11.0011110	11.0001111	11.0001111	–15	–15
–29	11.0011101	11.0001110(+1)	11.0001111	–15	–14,5
–28	11.0011100	11.0001110	11.0001110	–14	–14
–27	11.0011011	11.0001101(+1)	11.0001110	–14	–13,5
–26	11.0011010	11.0001101	11.0001101	–13	–13
–25	11.0011001	11.0001100(+1)	11.0001101	–13	–12,5
–24	11.0011000	11.0001100	11.0001100	–12	–12
–23	11.0010111	11.0001011(+1)	11.0001100	–12	–11,5

Все результаты соответствуют правилам математики

Исходное десятичное значение	Исходный прямой код	Сдвиг вправо на 3 (треб. по округ)	Результат с учетом округления	Десятичный эквивалент	Результат деления на 2
–31	11.0011111	11.0000011(+1)	11.0000100	–4	–3,875
–30	11.0011110	11.0000011(+1)	11.0000100	–4	–3,75
–29	11.0011101	11.0000011(+1)	11.0000100	–4	–3,625
–28	11.0011100	11.0000011(+1)	11.0000100	–4	–3,5
–27	11.0011011	11.0000011	11.0000011	–3	–3,375
–26	11.0011010	11.0000011	11.0000011	–3	–3,25
–25	11.0011001	11.0000011	11.0000011	–3	–3,125
–24	11.0011000	11.0000011	11.0000011	–3	–3
–23	11.0010111	11.0001110(+1)	11.0000011	–3	–2,875

Все результаты соответствуют правилам математики

Сдвиг влево. При осуществлении сдвига происходит инверсия бита перед тем как перейти на позицию знака. При однобайтном размере при сдвиге с седьмой на восьмую позицию. Как и ранее, сдвиг чисел больше чем на 3 разряда приводит к переполнению для десятичных эквивалентов больше 15.

Исходное десятичное значение	Исходный прямой код	Сдвиг влево на 1	Десятичный эквивалент	Сдвиг на 3 влево	Десятичный эквивалент
–31	11.0011111	11.0111110	–62	10.111100	переполнение

–30	11.0011110	11.0111100	–60	10.111000	переполнение
–29	11.0011101	11.0111010	–58	10.110100	переполнение
–28	11.0011100	11.0111000	–56	10.110000	переполнение
–27	11.0011011	11.0110110	–54	10.101100	переполнение
–16	11.0010000	11.0100000	–32	10.000000	переполнение
–15	11.0001111	11.0011110	–30	10.111100	–120
–14	11.0001110	11.0011100	–28	10.111000	–118
–13	11.0001101	11.0011010	–26	10.110100	–104

Обратный код отрицательных чисел

При сдвиге отрицательных чисел вправо при отбрасывании 0 производится вычитание 1 из младшего бита. Ноль, отбрасывания которой, требует вычитание единицы из младшего бита, обозначена красным цветом

Исходное десятичное значение	Исходный прямой код	Исходный обратный код	Сдвиг на 1 вправо (треб. по округ)	Результат с учетом округления	Результат в прямом коде	Десятичный эквивалент	Результат деления на 2
–31	11.0011111	11.1100000	11.1110000 (–1)	11.1101111	11.0010000	–16	–15,5
–30	11.0011110	11.1100001	11.1110000	11.1110000	11.0001111	–15	–15
–29	11.0011101	11.1100010	11.1110001 (–1)	11.1110000	11.0001111	–15	–14,5
–28	11.0011100	11.1100011	11.1110001	11.1110001	11.0001110	–14	–14
–27	11.0011011	11.1100100	11.1110010 (–1)	11.1110001	11.0001110	–14	–13,5
–26	11.0011010	11.1100101	11.1110010	11.1110010	11.0001101	–13	–13
–25	11.0011001	11.1100110	11.1110011 (–1)	11.1110010	11.0001101	–13	–12,5
–24	11.0011000	11.1100111	11.1110011	11.1110011	11.0001100	–12	–12
–23	11.0010111	11.1101000	11.1110100 (–1)	11.1110011	11.0001100	–12	–11,5

Все результаты соответствуют правилам математики

При сдвиге отрицательных чисел вправо при отбрасывании 0 производится вычитание 1 из младшего бита. Ноль, отбрасывания которой, требует вычитание единицы из младшего бита, обозначена красным цветом

Исходное десятичное значение	Исходный прямой код	Исходный обратный код	Сдвиг вправо на 3 (треб. по округ)	Результат с учетом округления	Результат в прямом коде	Десятичный эквивалент	Результат деления на 8
–31	11.0011111	11.1100000	11.1111100(–1)	11.1111011	11.0000100	–4	–3,875
–30	11.0011110	11.1100001	11.1111100(–1)	11.1111011	11.0000100	–4	–3,75
–29	11.0011101	11.1100010	11.1111100(–1)	11.1111011	11.0000100	–4	–3,625
–28	11.0011100	11.1100011	11.1111100(–1)	11.1111011	11.0000100	–4	–3,5
–27	11.0011011	11.1100100	11.1111100	11.1111100	11.0000011	–3	–3,375
–26	11.0011010	11.1100101	11.1111100	11.1111100	11.0000011	–3	–3,25
–25	11.0011001	11.1100110	11.1111100	11.1111100	11.0000011	–3	–3,125
–24	11.0011000	11.1100111	11.1111100	11.1111100	11.0000011	–3	–3
–23	11.0010111	11.1101000	11.1111101(–1)	11.1111100	11.0000011	–3	–2,875

Все результаты соответствуют правилам математики

Исходное десятичное значение	Исходный прямой код	Исходный обратный код	Сдвиг влево на 1	Результат в прямом коде	Десятичный эквивалент	Сдвиг влево на 3	Результат в прямом коде	Десятичный эквивалент
–31	11.0011111	11.1100000	11.1000001	11.0111110	–62	10.0000111	10.1111000	переполнение
–30	11.0011110	11.1100001	11.1000011	11.0111100	–60	10.0001111	10.1110000	переполнение
–29	11.0011101	11.1100010	11.1000101	11.0111010	–58	10.0010111	10.1101000	переполнение
–28	11.0011100	11.1100011	11.1000111	11.0111000	–56	10.0011111	10.1100000	переполнение
–27	11.0011011	11.1100100	11.1001001	11.0110110	–54	10.0010011	10.1101100	переполнение

–16	11.0010000	11.1101111	11.1011111	11.0100000	–32	10.1111111	10.0000000	переполнение
–15	11.0001111	11.1110000	11.1100001	11.0011110	–30	11.0000111	11.1111000	–120
–14	11.0001110	11.1110001	11.1100011	11.0011100	–28	11.0001111	11.1110000	–118
–13	11.0001101	11.1110010	11.1100101	11.0011010	–26	11.0010111	11.1101000	–104

Все результаты соответствуют правилам математики

Исходное десятичное значение	Исходный прямой код	Исходный обратный код	Сдвиг влево на 3	Результат в прямом коде	Десятичный эквивалент
–31	11.0011111	11.1100000	10.0000001	11.0111110	–62
–30	11.0011110	11.1100001	10.1000011	11.0111100	–60
–29	11.0011101	11.1100010	10.1000101	11.0111010	–58
–28	11.0011100	11.1100011	11.1000111	11.0111000	–56
–27	11.0011011	11.1100100	11.1001001	11.0110110	–54
–26	11.0011010	11.1100101	11.1001011	11.0110100	–52
–25	11.0011001	11.1100110	11.1001101	11.0110010	–50
–24	11.0011000	11.1100111	11.1001111	11.0110000	–48
–23	11.0010111	11.1101000	11.1010001	11.0101110	–46

Дополнительный код отрицательных чисел

При сдвиге в право чисел в обратном дополнительном коде ситуация более сложная. Как и при сдвиге вправо, отрицательных чисел в обратном коде, на освобождающееся место после знака появляются единицы.

Исходное 10 значение	Исходный прямой код	Исходный обратный код	Исходный дополнительный код	Сдвиг на 1	Результат с учетом округления	Результат в обратном коде	Результат в прямом коде	Десятичный эквивалент	Результат деления на 2
–36	11.0100100	11.1011011	11.1011100	11.1101110	11.1101110	11.1101101	11.0010010	–18	18
–35	11.0100011	11.1011100	11.1011101	11.1101110	11.1101110	11.1101101	11.0010010	–18	17,5
–34	11.0100010	11.1011101	11.1011110	11.1101111	11.1101111	11.1101110	11.0010001	–17	17
–33	11.0100001	11.1011110	11.1011111	11.1101111	11.1101111	11.1101110	11.0010001	–17	16,5
–32	11.0100000	11.1011111	11.1100000	11.1110000	11.1110000	11.1101111	11.0010000	–16	16
–31	11.0011111	11.1100000	11.1100001	11.1110000	11.1110000	11.1101111	11.0010000	–16	–15,5
–30	11.0011110	11.1100001	11.1100010	11.1110001	11.1110001	11.1110000	11.0001111	–15	–15
–29	11.0011101	11.1100010	11.1100011	11.1110001	11.1110001	11.1110000	11.0001111	–15	–14,5
–28	11.0011100	11.1100011	11.1100100	11.1110010	11.1110010	11.1110001	11.0001110	–14	–14
–27	11.0011011	11.1100100	11.1100101	11.1110010	11.1110010	11.1110001	11.0001110	–14	–13,5
–26	11.0011010	11.1100101	11.1100110	11.1110011	11.1110011	11.1110010	11.0001101	–13	–13
–25	11.0011001	11.1100110	11.1100111	11.1110011	11.1110011	11.1110010	11.0001101	–13	–12,5
–24	11.0011000	11.1100111	11.1101000	11.1110100	11.1110100	11.1110011	11.0001100	–12	–12
–23	11.0010111	11.1101000	11.1101001	11.1110100	11.1110100	11.1110011	11.0001100	–12	–11,5
–22	11.0010110	11.1101001	11.1101010	11.1110101	11.1110101	11.1110100	11.0001011	–11	11

Все результаты соответствуют правилам математики

Как мы видим, при сдвиге на 1 проблемы округления нет, как при прямом или обратном коде.

Совсем другие аспекты возникаю при сдвиге на 2 и более. Теперь для обеспечения математических правил нужно следить, что отбрасываем. Правило округления требует прибавления 1, если последней отбрасывается 1 и в предыдущих битах присутствовали хотя бы одна единица.

Исходное десятичное значение	Исходный прямой код	Исходный обратный код	Исходный дополни–тельный код	Сдвиг на 3	Результат с учетом округления	Результат в обратном коде	Результат в прямом коде	Десятичный эквива-лент	Резуль-тат деления на 2
–36	11.0100100	11.1011011	11.1011100	11.1111011	11.1111011	11.1111010	11.0000101	–5	–4,5
–35	11.0100011	11.1011100	11.1011101	11.1111011 (+1)	11.1111100	11.1111011	11.0000100	–4	–4,375
–34	11.0100010	11.1011101	11.1011110	11.1111011 (+1)	11.1111100	11.1111011	11.0000100	–4	–4,25
–33	11.0100001	11.1011110	11.1011111	11.1111011 (+1)	11.1111100	11.1111011	11.0000100	–4	–4,125
–32	11.0100000	11.1011111	11.1100000	11.1111100	11.1111100	11.1111011	11.0000100	–4	4
–31	11.0011111	11.1100000	11.1100001	11.1111100	11.1111100	11.1111011	11.0000100	–4	–3,875
–30	11.0011110	11.1100001	11.1100010	11.1111100	11.1111100	11.1111011	11.0000100	–4	–3,75

–29	11.0011101	11.1100010	11.1100011	11.1111100	11.1111100	11.1111011	11.0000100	–4	–3,625
–28	11.0011100	11.1100011	11.1100100	11.1111100	11.1111100	11.1111011	11.0000100	–4	–3,5
–27	11.0011011	11.1100100	11.1100101	11.1111100 (+1)	11.1111101	11.1111100	11.0000011	–3	–3,375
–26	11.0011010	11.1100101	11.1100110	11.1111100 (+1)	11.1111101	11.1111100	11.0000011	–3	–3,25
–25	11.0011001	11.1100110	11.1100111	11.1111100 (+1)	11.1111101	11.1111100	11.0000011	–3	–3,125
–24	11.0011000	11.1100111	11.1101000	11.1111101	11.1111101	11.1111100	11.0000011	–3	–3
–23	11.0010111	11.1101000	11.1101001	11.1111101	11.1111101	11.1111100	11.0000011	–3	–2,875
–22	11.0010110	11.1101001	11.1101010	11.1111101	11.1111101	11.1111100	11.0000011	–3	–2,75

Исходное десятичное значение	Исходный прямой код	Исходный обратный код	Исходный дополнительный код	Сдвиг на 2	Результат с учетом округления	Результат в обратном коде	Результат в прямом коде	Десятичный эквивалент	Результат деления на 2
–36	11.0100100	11.1011011	11.1011100	11.1110111	11.1110111	11.1110110	11.0001001	–9	–9
–35	11.0100011	11.1011100	11.1011101	11.1110111	11.1110111	11.1110110	11.0001001	–9	–8,75
–34	11.0100010	11.1011101	11.1011110	11.1110111	11.1110111	11.1110110	11.0001001	–9	–8,5
–33	11.0100001	11.1011110	11.1011111	11.1110111(+1)	11.1111000	11.1110111	11.0001000	–8	–8,25
–32	11.0100000	11.1011111	11.1100000	11.1111000	11.1111000	11.1110111	11.0001000	–8	8
–31	11.0011111	11.1100000	11.1100001	11.1111000	11.1111000	11.1110111	11.0001000	–8	–7,75
–30	11.0011110	11.1100001	11.1100010	11.1111000	11.1111000	11.1110111	11.0001000	–8	–7,5
–29	11.0011101	11.1100010	11.1100011	11.1111000(+1)	11.1111001	11.1111000	11.0000111	–7	–7,25
–28	11.0011100	11.1100011	11.1100100	11.1111001	11.1111001	11.1111000	11.0000111	–7	–7
–27	11.0011011	11.1100100	11.1100101	11.1111001	11.1111001	11.1111000	11.0000111	–7	–6,75
–26	11.0011010	11.1100101	11.1100110	11.1111001	11.1111001	11.1111000	11.0000111	–7	–6,5
–25	11.0011001	11.1100110	11.1100111	11.1111001(+1)	11.1111011	11.1111001	11.0000110	–6	–6,25
–24	11.0011000	11.1100111	11.1101000	11.1111010	11.1111011	11.1111001	11.0000110	–6	–6
–23	11.0010111	11.1101000	11.1101001	11.1111010	11.1111011	11.1111001	11.0000110	–6	–5,75
–22	11.0010110	11.1101001	11.1101010	11.1111010	11.1111011	11.1111001	11.0000110	–6	–5,5

Все результаты соответствуют правилам математики

Примеры выполнения сдвигов

Исходное число

11.111 110 011 100 111$_{ДК}$ = 11.111 110 011 100 110$_{ОК}$ = 11.000 001 100 011 001$_{ПК}$ = = (–1) (1+8+16+256+512) = –793$_{10}$

–793$_{10}$ = 11.111 110 011 100 111$_{ДК}$ 1® = 11.111 111 001 110 011$_{ДК}$ » (поскольку последней ушла 1 только одна 1, то необходимости прибавить 1 нет) » 11.111 111 001 110 011$_{ДК}$

–793/2 = –396,5 » –397 = 11.111 111 001 110 011$_{ДК}$

–793$_{10}$ = 11.111 110 011 100 111$_{ДК}$ 3® = 11.111 111 110 011 100$_{ДК}$ »

» 11.111 111 110 011 101$_{ДК}$ (поскольку последней ушла 1 и в предыдущих битах присутствовала 1, то необходимо прибавить 1)

–793/8 = –99,125 » –99 = 11.111 111 110 011 101$_{ДК}$

–793$_{10}$ = 11.111 110 011 100 111$_{ДК}$ 4® = 11.111 111 111 001 110$_{ДК}$ »

» 11.111 111 111 001 110$_{ДК}$ (поскольку последним ушел ноль, то не прибавляем 1)

–793/16 = –49,5625» –50 = 11.111 111 111 001 110$_{ДК}$

–793$_{10}$ = 11.111 110 011 100 111$_{ДК}$ 6® = 11.111 111 111 110 011$_{ДК}$ »

» 11.111 111 111 110 100$_{ДК}$

(поскольку последней ушла 1 и в предыдущих битах присутствовала 1, то необходимо прибавить 1)

–793/64 = –12,390625» –12 = 11.111 111 111 110 100$_{ДК}$

–800$_{10}$ = 11.111 110 011 100 000$_{ДК}$ 6® = 11.111 111 111 110 011$_{ДК}$ »

» 11.111 111 111 110 100$_{ДК}$ (поскольку последней ушла 1 и в предыдущих битах не присутствуют 1, то необходимости прибавить 1 нет)

–800/64 = –12,5» –13 = 11.111 111 111 110 011$_{ДК}$

-799_{10} = 11.111 110 011 100 $001_{ДК}$ 6® = 11.111 111 111 110 $011_{ДК}$ »
»11.111 111 111 110 $100_{ДК}$ (поскольку последней ушла 1 и в предыдущих битах присутствует 1, то необходимо прибавить 1)
–799/64 = –12,484375» –12 = 11.111 111 111 110 $100_{ДК}$

Задача 4

1	знак	порядок	знак	мантисса	Сложить числа между собой в разной последовательности. Использовать модифицированный дополнительный код. Найти эквиваленты исходных и конечных чисел в десятичном представлении. Определить погрешность вычислений. Представить конечный результат в стандарте IEEE 754.
A	1	010	1	111101	
B	0	001	0	100111	
C	1	001	1	111001	

Теория

При сложении чисел с плавающей точкой в микропроцессорах действует обычно следующий алгоритм.

1. Происходит сравнение порядка (степени).
2. Мантисса меньшего числа сдвигается на число, насколько отличается порядок чисел. Т.е. уравниваются порядки двух чисел
3. Сложение мантисс.
4. Нормализация мантиссы.

В отличии от условия задачи, числа в памяти хранятся изначально в стандарте IEEE 754.

Получение десятичных эквивалентов

Согласно условия задачи, мы имеем следующие числа

знак	порядок	знак	мантисса
1	010	1	111101

$$A = 2^{(-1) › (0 › 4 + 1 › 2 + 0 › 1)} \quad (-1) › (1 › 2^{0} + 1 › 2^{-1} + 1 › 2^{-2} + 1 › 2^{-3} + 0 › 2^{-4} + 1 › 2^{-5})$$

знак	порядок	знак	мантисса
0	001	0	100111

$$B = 2^{(+1) › (0 › 4 + 0 › 2 + 1 › 1)} \quad (+1) › (1 › 2^{0} + 0 › 2^{-1} + 0 › 2^{-2} + 1 › 2^{-3} + 1 › 2^{-4} + 1 › 2^{-5})$$

знак	порядок	знак	мантисса
1	001	1	111001

$$C = 2^{(-1) › (0 › 4 + 0 › 2 + 1 › 1)} \quad (-1) › (1 › 2^{0} + 1 › 2^{-1} + 1 › 2^{-2} + 0 › 2^{-3} + 0 › 2^{-4} + 1 › 2^{-5})$$

$A = (-1) \times (1 + 0,5 + 0,25 + 0,125 + 0,03125)\ 2^{-2} = -0,4765625.$

$B = (1 + 0,125 + 0,0625 + 0,03125)\ \ 2^{1} = 2,4375.$

$C = (-1) \times (1 + 0,5 + 0,25 + 0,03125)\ \ 0,5 = -0,890625.$

(производить вычисления десятичных эквивалентов совсем не обязательно, здесь делается для понимания процессов)

Для выполнения задания необходимо сложить три числа в разной последовательности. Особенность таких сложений в формате с плавающей точкой, что результат таких сложений обычно отличаются. В нашем случае из-за короткой длины мантиссы они скорее будут совпадать, но могут и отличаться.

Выполнение сложение

Сначала реализуем вариант (A + B) + C. Потом (A + C) + B.

Поскольку порядок числа А (–2) меньше порядка числа В (+1) на три, то для уравнивания порядков необходимо сдвинуть мантиссу числа А на эту разницу. Запишем число А модифицированном коде и сдвинем, а затем преобразуем в дополнительный код.

А = $11.111101_{\text{пк}}$ ($11.010_{\text{пк}}$) =(**à** 3)= $11.000111_{\text{пк}}$ ($00.001_{\text{пк}}$)

Так как последней при сдвиге ушла 1, то добавим к последнему биту 1. Тогда окончательно после сдвига. В скобках указан текущий порядок

А = $11.001000_{\text{пк}}$ ($00.001_{\text{пк}}$) = $11.111000_{\text{дк}}$ ($00.001_{\text{пк}}$).

Число с более высоким порядком остается. Оно у на положительное, тогда

В = 00. $100111_{\text{пк}}$ ($00.001_{\text{пк}}$).

$$\begin{array}{rll} & \text{Мантисса} & \text{Порядок} \\ & \cdot\,\cdot & \\ & 11.111000_{\text{дк}} & 00.001_{\text{пк}}\ \text{(A)} \\ + & 00.100111_{\text{дк}} & 00.001_{\text{пк}}\ \text{(B)} \\ \hline \cancel{1} & 00.011111_{\text{дк}} & 00.001_{\text{пк}}\ \text{(A+B)} \end{array}$$

В результате сложения получилось ненормализованное число. Для нормализации необходимо произвести сдвиг числа на то количество разрядов, чтобы для положительного числа после знака стояла 1, а для отрицательного числа в дополнительном коде 0.

(А + В) = $00.011111_{\text{дк}}$ ($00.001_{\text{пк}}$) = $00.111110_{\text{дк}}$ ($00.000_{\text{пк}}$).

Далее производим сравнение порядков суммы и числа С. Для выполнения процесса суммирования необходимо уравнять порядки так как у числа С порядок на 1 меньше. Кроме этого оно отрицательно и его необходимо преобразовать в дополнительный код.

С = $11.111001_{\text{пк}}$ ($11.001_{\text{пк}}$) =(**à** 1)= $11.011100_{\text{пк}}$ ($00.000_{\text{пк}}$)

И после округления и преобразования окончательно получим число пригодное для суммирования.

С = $11.011100_{\text{пк}}$ ($00.000_{\text{пк}}$) = $11.011101_{\text{пк}}$ ($00.000_{\text{пк}}$) = $11.100011_{\text{дк}}$ ($00.000_{\text{пк}}$)

$$\begin{array}{rll} & \text{Мантисса} & \text{Порядок} \\ & \cdot\,\cdot\ \ \cdot\,\cdot\,\cdot\,\cdot & \\ & 00.111110_{\text{дк}} & 00.000_{\text{пк}}\ \text{(A+B)} \\ + & 11.100011_{\text{дк}} & 00.000_{\text{пк}}\ \text{(C)} \\ \hline \cancel{1} & 00.100001_{\text{дк}} & 00.000_{\text{пк}}\ \text{(A+B+C)} \end{array}$$

Другая последовательность сложения (А + С) + В.

Разница порядков числа А и С равна 1. Сдвигать надо число А

А = $11.111101_{\text{пк}}$ ($11.010_{\text{пк}}$) =(**à** 1)= $11.011111_{\text{пк}}$ ($11.001_{\text{пк}}$)

Далее преобразование в дополнительный код

А = $11.011111_{\text{пк}}$ ($00.001_{\text{пк}}$) = $11.100001_{\text{дк}}$ ($11.001_{\text{пк}}$)

С = $11.111001_{\text{пк}}$ ($11.001_{\text{пк}}$) = $11.000111_{\text{дк}}$ ($11.001_{\text{пк}}$)

$$\begin{array}{rll} & \text{Мантисса} & \text{Порядок} \\ & \cdot\ \ \ \ \ \ \cdot\,\cdot\,\cdot & \\ & 11.100001_{\text{дк}} & 11.001_{\text{пк}}\ \text{(A)} \\ + & 11.000111_{\text{дк}} & 11.001_{\text{пк}}\ \text{(C)} \\ \hline \cancel{1} & 10.101000_{\text{дк}} & 11.001_{\text{пк}}\ \text{(A+C)} \end{array}$$

Число ненормализованное. Нормализуем.

(А + С) = $10.101000_{\text{дк}}$ ($11.001_{\text{пк}}$) = (**à** 1) = $11.010100_{\text{дк}}$ ($00.000_{\text{пк}}$).

Сравнение порядка результата и числа В, требует сдвинуть результат еще на 1 для уравнивания порядков .

(А + С) = $11.010100_{\text{дк}}$ ($00.000_{\text{пк}}$) = (**à** 1) = $11.101010_{\text{дк}}$ ($00.001_{\text{пк}}$).

Можно перейти к сложению

Мантисса Порядок

$$
\begin{array}{rll}
 & \dot{1}\,\dot{1}\,.\,\dot{1}\,\dot{0}\,\dot{1}\,0\,1\,0_{дк} & 11.001_{пк}\ (A+C) \\
+ & 1\,1\,.\,1\,0\,1\,0\,1\,0_{дк} & 11.001_{пк}\ (A+C) \\
 & 0\,0\,.\,1\,0\,0\,1\,1\,1_{дк} & 11.001_{пк}\ (B) \\
\hline
\cancel{1} & 0\,0\,.\,0\,1\,0\,0\,0\,1_{дк} & 11.001_{пк}\ (A+C+B)
\end{array}
$$

Требуется нормализация результата

(A+C+B) = $00.010001_{дк}$ ($00.001_{пк}$) = (**à** 1) = $00.100010_{дк}$ ($00.000_{пк}$)

Как видим результаты получены разные, что является типичной ситуацией для сложения чисел с плавающей точкой. В первом случае получили 1,03125. во втором 1,0625. Вычисление исходных значений на калькуляторе дает значение 1,0703. Второй вариант ближе, что неудивительно поскольку в последнем сложении выполнялись действия с числами с близкими порядками, что требовало меньший размер сдвигов, а следовательно и потерь в точности.

Запись в стандарт IEEE754

Следует отметить, что микропроцессоры и программное обеспечение применяют иногда более сложный механизм огрубления. В последних вариантах стандарта IEEE754 прописаны несколько вариантов огрубления. Кроме этого в самом распространенном формате с плавающей точкой в мантиссе оперируют 24 битами мантиссы, а не как в нашем случае 6. Но все-таки следует отметить, что сортировка чисел при большой разнице порядков позволяет обеспечить большую точность вычислений.

Формат числа с плавающей точкой float(real), согласно IEEE754, предусматривает 32 бита. Первый бит это бит знака, далее 8 бит порядка(степени). Остальные 23 бита – это мантисса. Согласно стандарта существуют нюансы, но значения в задачах заданы таким образом, что это можно игнорировать. Порядок числа смещается на 127, чтобы исключить при работе с порядком особенности работы с отрицательными числами. В данный формат результаты необходимо записывать только нормализованный, т.е. в старшем бите мантиссы должна находится 1. Однако непосредственно в регистры хранения эта единица не записывается. Это позволяет на один бит повысить точность вычислений в данном формате.

Запишем последний результат (A+C+B) = $00.100010_{пк}$ ($00.000_{пк}$)в данном стандарте. Сначала добавим к степени 127. Это семь единиц.

$$
\begin{array}{rll}
 & 0\,0\,.\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0_{дк} & \text{Порядок} \\
+ & 0\,0\,.\,0\,1\,1\,1\,1\,1\,1\,1\,1\,1_{дк} & 127 \\
\hline
 & 0\,0\,.\,0\,1\,1\,1\,1\,1\,1\,1\,1\,1_{дк} & \text{Смещенный порядок}
\end{array}
$$

Тогда окончательно результат получим

$$(A+C+B) = \underbrace{0}_{\text{Бит знака}}.\underbrace{01111111}_{\text{8 бит порядка}}.\underbrace{0001000000\ldots..}_{\text{23 бита мантиссы без первой 1}}$$

Точки между частями поставлены для удобства. В регистрах все биты универсальны. В том же регистр из 32 бит может хранится, например, двойное целое (double integer), и все биты, кроме бита знака будут нести другой смысл. Что

Еще один пример записи результата в формат IEEE754. Пускай было получено следующее число(A+C+B) = $11.010001_{дк}$ ($11.101_{пк}$). Для записи в стандарт преобразуем в прямой код

(A+C+B) = $11.010101_{дк}$ ($11.101_{пк}$) = (**à** 1) = $11.101011_{пк}$ ($00.000_{пк}$).

Добавим к степени 127. Для выполнения сложения порядок преобразуем в дополнительный код с учетом использования для хранения порядка 8 бит.

$11.101_{ПК}$ **à** $11.00000101_{ПК} = 11.11111011_{ДК}$

$$\begin{array}{rl} & 11.1111111011_{ДК} \quad \text{Порядок} \\ + & 00.0111111111_{ДК} \quad 127 \\ \hline \cancel{1} & 00.0111111010_{ДК} \quad \text{Смещенный порядок} \end{array}$$

Тогда окончательно результат получим

$$\overset{\text{Бит знака}}{1}.\underbrace{01111010}_{\text{8 бит порядка}}.\underbrace{0101100000\ldots..}_{\text{23 бита мантиссы без первой 1}}$$

Задача 5

<table>
<tr><td>1</td><td>знак</td><td>порядок</td><td>знак</td><td>мантисса</td><td rowspan="3">Умножить (со сдвигом промеж. рез-та) два наибольших числа и поделить большее на меньшее (с восстановлением остатка) . Представить конечный результат в стандарте IEEE 754.</td></tr>
<tr><td>A</td><td>1</td><td>1001</td><td>1</td><td>10110</td></tr>
<tr><td>B</td><td>0</td><td>1011</td><td>0</td><td>10011</td></tr>
</table>

Теория

Умножение чисел в формате с плавающей точкой происходит по следующему алгоритму: сложение порядков, умножение мантисс, нормализация результата умножения мантисс и уточнение порядка.

Сам процесс умножения мантисс заменяется на ряд последовательных сложений. Есть 4 варианта таких