Elementy arytmetyki komputerowej

cz. I Elementy systemów liczbowych

/materiał pomocniczy do wykładu Informatyka sem II/

Spis treści

1. Wprowadzenie

Arytmetyka komputerowa (ang. computer arithmetic) zajmuje się problemami realizacji obliczeń w urządzeniach cyfrowych. Obejmuje ona zagadnienia teoretyczne takie jak systemy liczbowe przydatne w projektowaniu urządzeń liczących lub w programowej realizacji obliczeń. Zagadnienia związane z systemami liczbowymi obejmują konstrukcję nowych systemów liczbowych i badanie własności systemów. Przedmiotem arytmetyki komputerowej jest ponadto realizacja podstawowych operacji arytmetycznych takich jak dodawanie i odejmowanie liczb bez znaku i ze znakiem, mnożenie i dzielenie jak i obliczanie standardowych funkcji arytmetycznych (sinus, cosinus, logarytm naturalny, arcus sinus, arcus tangens, pierwiastek kwadratowy i funkcja e^x). Arytmetyka komputerowa zajmuje się także syntezą szybkich i efektywnych układów do realizacji wyżej wymienionych operacji. Ten dział arytmetyki komputerowej wiąże się ścisle z technologią wielkiej skali integracji VLSI (Very Large Scale of Integration). Aktualne zagadnienia arytmetyki cyfrowej to m.in. bardzo szybka realizacja obliczeń z wykorzystaniem potokowania (ang. pipelining) i wykonywanie operacji arytmetycznych z małym poborem mocy (ang. low-power arithmetic). Zagadnienie to ma szczególne znaczenie w związku z rozwojem urządzeń przenośnych, jak również ze wzrostem stosowanych częstotliwości zegarowych, zważywszy na fakt iż pobór mocy w urządzeniach VLSI jest liniowo proporcjonalny do częstotliwości zegara. Do aktualnych zagadnień należy również arytmetyka odporna na błędy (ang. fault-tolerant arithmetic).

W poniższym opracowaniu zestawiono nieco rozszerzoną treść czterech wykładów w ramach części I wykładu Informatyka obejmującego podstawy informatyki.

2. Uwagi wstępne o systemach liczbowych

Systemy reprezentacji liczb rozwijały się wraz z rozwojem języka. Najstarsze metody reprezentacji liczb to reprezentowanie przy użyciu różnych przedmiotów np. kamieni lub patyczków (łac. calculus oznacza kamyczek). Jednak gdy powstała konieczność stosowania większych liczb, ich reprezentowanie z użyciem pojedynczych przedmiotów jak też porównywanie takich liczb stawało się coraz trudniejsze. Pierwszym udoskonaleniem było grupowanie przedmiotów reprezentujących jednostki (np. liczbę 23 reprezentowano jako 4 grupy po 5 kamyczków oraz 3 osobne). Główny przełom stanowiło jednak wprowadzenie większych przedmiotów reprezentujących grupę 5 lub 10 jednostek.

Metoda ta rozwinęła się do formy symbolicznej, gdy zaczęto stosować specjalne symbole do oznaczenia większych jednostek. Znanym przykładem jest zapis rzymski. Jednostkami w

tym systemie są 1, 5, 10, 50, 100, 500, 1000, 10000 i 100000, które są oznaczane odpowiednio symbolami I, V, X, L, C, D, M, ((I)) i (((I))). Liczbę w tym systemie reprezentuje się w postaci łańcucha symboli o wartości malejącej od lewej strony. Aby skrócić zapis przyjęto, że symbol znajdujący się po lewej stronie większego symbolu reprezentuje wartość ujemną. Np. liczbę 9 bezpośrednio można by zapisać jako VIIII, jednak wygodniejszy jest zapis IX. Podobnie liczbę 450 można by zapisać jako CCCCL, jednak zapis LD jest bardziej dogodny.

System rzymski nie jest odpowiedni do reprezentowania dużych liczb jak również trudno jest wykonywać operacje arytmetyczne z jego użyciem. Istotną innowacją było wprowadzenie przez Chińczyków systemu pozycyjnego(wagowego), w którym wartość reprezentowana przez symbol nie zależy tylko od niego samego, ale także od jego położenia względem innych symboli. Przykładowo, w liczbie 444 każda z cyfr "4", reprezentuje inną wartość. Skrajna od lewej reprezentuje 400, środkowa 40, a skrajna z prawej 4.

System liczbowy, *S* można określić jako zbiór

$$S=\{\mathrm{X}, \mathcal{X}, \mathrm{F}\}, \tag{1}$$

gdzie X- zbiór wartości abstrakcyjnych liczb,

$\mathcal{X}$- zbiór reprezentacji liczb,

F: $\mathcal{X} \rightarrow$ X.

Ogólnie reprezentację liczby można zapisać następująco

$$\left(x_{n-1}, x_{n-2}, ..., x_0\right), \tag{2}$$

gdzie x_i , i=0,1,2,...,n-1 są cyframi reprezentacji. Dla popularnych systemów pozycyjnych nie stosuje się zwykle przy zapisie nawiasów ani przecinków.

3. Przegląd wybranych systemów liczbowych

W systemie wagowym liczba X jest przedstawiana w sposób następujący:

$$X = \sum_{i=-m}^{n-1} d_i w_i \ , \tag{3}$$

gdzie $w_i \in \{W\}$, a $d_i \in \{D\}$, zbiór W zwany jest zbiorem wag, a zbiór D zbiorem cyfr.

Jeżeli wszystkie wagi są potęgami pewnej stałej r, $w_i = r^i$, to taki system wagowy nazywamy systemem z stałą podstawą (ang. fixed-radix number system). Liczbę X w takim systemie można zapisać jako

$$X = \sum_{i=-m}^{n-1} d_i r^i . \tag{4}$$

Zbiór dozwolonych cyfr jest taki sam na każdej pozycji.

Istnieją też , będące w powszechnym użyciu, systemy wagowe nie będące systemami o stałej podstawie, np. system zliczania czasu. Niech T oznacza czas w ciągu doby wyrażony w sekundach dla pewnej liczby godzin, minut i sekund. T możemy zapisać jako

$$T = E_0 + E_1 \cdot 60 + E_2 \cdot 60 \cdot 60 , \tag{5}$$

gdzie poszczególne cyfry mogą przyjmować wartości z następujących zakresów

$$0 \le E_0 < 60 ,$$

$$0 \le E_1 < 60 ,$$

$$0 \le E_2 < 24 .$$

Bezwzględną dokładność reprezentacji liczby w systemie wagowym wyznacza waga najmniej znaczącej pozycji czyli r^{-m}. W systemie ze stałą podstawą reprezentacja zera unikalna, gdyż nie istnieje liczba różna od $\{0,0,...,0\}$, której wartością byłoby zero. W systemie takim liczby najmniejsza, N i największa, P reprezentowalne na n pozycjach, mają postać

$$N = (0,0,0,...,0) , \tag{6}$$

oraz

$$P = (r-1, r-1, r-1,..., r-1) . \tag{7}$$

Sposób reprezentacji liczb wpływa nie tylko na łatwość odczytywania liczb, a także na złożoność algorytmów arytmetycznych stosowanych przy realizacji operacji na liczbach. Systemy pozycyjne zawdzięczają swą popularność, przynajmniej częściowo, dostępności prostych i wygodnych algorytmów do realizacji działań na liczbach reprezentowanych w tych systemach. Istnieją jednak systemy niepozycyjne, takie jak resztowy system liczbowy, które mają przewagę nad systemami pozycyjnymi przy realizacji niektórych działań arytmetycznych w pewnych dziedzinach zastosowań.

W systemach cyfrowych liczby są kodowane przy użyciu cyfr binarnych zwanych bitami. Stosowanie systemu binarnego wynika z powodów technologicznych, gdyż w fizycznych urządzeniach łatwiej jest reprezentować dwa stany, którym przypisuje się cyfry binarne. Operandami algorytmów arytmetyki komputerowej są więc kody reprezentujące liczby. Zbiór

kodów takich jest jednak zbiorem dyskretnym i skończonym, stąd nie wszystkie reguły arytmetyki konwencjonalnej, stosują się do algorytmów arytmetyki komputerowej. Stosuje się zasadniczo dwa rodzaje reprezentacji liczb w komputerach :

-reprezentację stałoprzecinkową (ang. fixed-point representation), gdzie liczba pozycji przeznaczonych do zakodowania części całkowitej liczby i części ułamkowej liczby jest stała,

-reprezentację zmiennoprzecinkową (ang. floating-point representation), w której osobno jest kodowana jest mantysa wraz ze znakiem oraz osobno cecha określająca relację między liczbą a mantysą.

Przykład 3.1. Reprezentacja stałoprzecinkowa o 8 pozycjach może być zapisana następująco

$$\text{xxxx.xxxx}, \tag{8}$$

kropka (przecinek) oddziela cześć całkowitą od części ułamkowej. W systemie dziesiętnym można przy użyciu takiej reprezentacji przedstawiać liczby z zakresu [0000.000, 9999.9999], a w systemie binarnym [0000.0000,1111.1111].

Reprezentacja zmiennoprzecinkowa w systemie dziesiętnym ma postać

$$m \cdot 10^c, \tag{9}$$

gdzie m jest mantysą a c cechą. Mantysa i cecha są liczbami stałoprzecinkowymi. Reprezentacja taka nie jest jednoznaczna, np. liczba 12.35 może być reprezentowana jako $1.235 \cdot 10^1$, czy też $0.1235 \cdot 10^2$, zwykle przyjmuje się, że $0 \le m < 1$.

Reprezentacja zmiennoprzecinkowa w systemie binarnym w najprostszym przypadku może mieć następującą postać

$$m \cdot 2^c, \tag{10}$$

w której m i c są liczbami binarnymi stałoprzecinkowymi oraz $0.5 \le m < 1$. Zakres ten wynika stąd, że jeżeli chcemy aby ułamek binarny miał cyfrę 1 po kropce oddzielającej część ułamkową od części całkowitej, to jego minimalną wartością będzie $2^{-1} = 0.5$. W praktyce stosuje się jednak z różnych względów reprezentacje o bardziej złożonej formie niż (10). Zwykle stosuje się standardowe reprezentacje zmiennoprzecinkowe okreslone standardami IEEE 754 i IEEE 854.

Jak zaznaczono powyżej, liczby w systemach cyfrowych są kodowane przy użyciu bitów. Kodowanie oznacza, że każdej liczbie, która ma być reprezentowana w danym systemie cyfrowym, przypisuje się pewien kod-ciąg bitów reprezentujący tę liczbę. Przypisanie to powinno być logiczne i systematyczne, tak aby umożliwiało prostą realizację operacji arytmetycznych, jak również proste sprawdzanie osobliwości lub specjalnych przypadków.

Może to być np. sprawdzanie, czy kod otrzymany w wyniku jakiejś operacji reprezentuje liczbę, czy wynik jest równy zeru, czy też nastąpiło przekroczenie zakresu liczbowego(np. powstał nadmiar -ang. overflow). Stąd przykładowo, jeżeli przypiszemy pewne kody do liczb 2 i 4, a nie przypiszemy kodu dla liczby 6, to w wyniku operacji dodawania powstanie wartość niereprezentowalna w danym systemie.

Poniżej pokażemy różne sposoby przypisania 4-bitowych kodów do liczb.

Przy najprostszym podejściu 4-bitowe kody można traktować jako 4-bitowe liczby naturalne z zakresu [0,15]. Jeśli bit o najwyższej wadze potraktujemy jako bit znaku (przypisując np. 0, jeśli liczba jest dodatnia i 1 jeśli liczba jest ujemna), otrzymujemy liczby z zakresu [-7,7], liczba 0 będzie miała dwie reprezentacje (+0 i -0). Jeżeli przyjmiemy 3 bity dla części całkowitej i 1 bit dla części ułamkowej, otrzymamy liczby z zakresu [0,7.5] co 0.5. Jeśli założymy, że pierwszy bit jest bitem znaku, a pozostałe reprezentują ułamek binarny, to uzyskamy liczby z zakresu [-0.875,0.875].

Dla reprezentacji zmiennoprzecinkowej, jeżeli przeznaczymy dwa bity do kodowania cechy i dwa bity do kodowania mantysy i przyjmiemy, że 2-bitowa mantysa c należy do przedziału [0,3], a 2-bitowa cecha c do [-2,1], to przy zastosowaniu reprezentacji $m \cdot 2^c$, możemy kodować liczby z zakresu [0,6]. W tym systemie zero ma cztery reprezentacje (mantysa równa zeru dla każdej wartości cechy), 0.50, 1.00, 2.00 mają dwie reprezentacje, liczby 0.25, 0.75, 1.50, 3.00, 4.00 i 6.00 są reprezentowane jednoznacznie.

1. System zrównoważony trójkowy : r=3, $d_i \in \{-1,1\}$.

2. System z ujemną podstawą : podstawa -r , $d_i \in \{0,1,...,r-1\}$, wartość liczby w tym systemie można wyrazić następująco:

$$X = \sum_i d_i (-r)^i = \sum_{i\ parzyste} d_i r^i - \sum_{i\ nieparzyste} d_i r^i . \qquad (11)$$

Dla specjalnego przypadku r=-2 i dla zbioru cyfr $d_i \in \{0,1\}$, system ten jest nazywany negabinarnym(minus dwójkowym).

3. Systemy nieredundacyjne ze cyframi ze znakiem: podstawa r i zbiór cyfr $d_i \in \{-\alpha, r-1-\alpha\}$. Przykładowo niech r=10, $\alpha = 5$ i $d_i \in \{-5,-4,-3,-2,-1,0,1,2,3,4\}$. Cyfry ujemne oznaczamy stosując znak minus, wtedy

(4 -3 1) reprezentuje liczbę $371 = 400 - 3 \cdot 10 + 1$,

(-4 3 1) reprezentuje liczbę $-369 = 400 + 3 \cdot 10 + 1$,

4. Systemy redundancyjne nadmiarowe ze znakiem. W systemach tych $d_i \in \{\alpha,...,\beta\}$ oraz $\alpha + \beta \geq r$. W systemach takich pewne wartości mają więcej niż jedną reprezentację, np.dla $d_i \in \{-7,..,0,..,7\}$ i *r*=10 liczba dziesiętna 295 ma następujące reprezentacje:

$$(3\ \ -1\ \ 5) = (3\ \ 0\ \ -5) = (1\ \ -7\ \ 0\ \ -5).$$

5. Systemy z ułamkową podstawą: *r*=0.1 i $d_i \in \{0,...,9\}$. Wartość liczby w takim systemie można wyrazić jako

$$X = \sum_i d_i 10^{-i} \tag{12}$$

6. Systemy liczbowe z podstawą niewymierną, w których np. $r = \sqrt{2}$ i $d_i \in \{0,1\}$, wartość liczby przedstawia zależność

$$X = \sum_i d_i (\sqrt{2})^i \ . \tag{13}$$

7. Systemy z podstawą zespoloną:, przykładowo *r*=2*j*, gdzie $j = \sqrt{-1}$, i $d_i \in \{0,1,2,3\}$. Wartość liczby wyraża się tutaj zależnością

$$X = \sum_i d_i (2j)^i \tag{14}$$

Zanim przejdziemy do systemów liczbowych mających specjalne znaczenie w informatyce, rozważymy jeszcze kwestię długości reprezentacji dla systemów o podstawie *r* i standardowym zbiorze cyfr [0,*r*-1]. Liczbę cyfr konieczną do reprezentacji liczb naturalnych z przedziału [0,max] można wyrazić jako

$$k = \lfloor \log_r \max \rfloor + 1 = \lceil \log_r (\max + 1) \rceil . \tag{15}$$

Należy zauważyć, że liczba różnych wartości w przedziale [0,max] jest równa max+1.

Stosując reprezentację stałoprzecinkową z k cyframi dla części całkowitej i *l* cyframi dla części ułamkowej, wartość max można wyrazić w sposób następujący

$$\max = r^k - r^{-l} = r^k - ulp\ , \tag{16}$$

gdzie *ulp*(ang. unit in least(significant) position), oznacza cyfrę na najmniej znaczącej pozycji.

4. Systemy binarny naturalny, oktalny i heksadecymalny

Rozważymy teraz bliżej systemy binarny, oktalny i heksadecymalny mające zasadnicze znaczenie w informatyce.

a) system binarny(dwójkowy) naturalny

W systemie binarnym r=2 oraz $d_i \in \{0,1\}$

$$X = \sum_{i=0}^{n-1} d_i 2^i . \tag{17}$$

Przykład 4.1. Znaleźć wartości dziesiętne liczb binarnych.

a) $X_1=(11)_2$, liczba ta ma wartość 3 w systemie dziesiętnym,

b) $X_2=(11000)_2$, jest równa 24 w systemie dziesiętnym,

c) $X_3=(1111111)_2$, odpowiada 127 w systemie dziesiętnym.

Maksymalna wartość liczby binarnej reprezentowanej na n pozycjach o wagach od 1 do 2^{n-1} jest równa

$$X = 2^n - 1 . \tag{18}$$

Warto zauważyć, że liczba 2^n wymaga reprezentacji na n+1 pozycjach. Stosując system binarny często używa się określeń bajt, liczba dwu-, czterobajtowa, a także n-bajtowa. Bajt oznacza liczbę złożoną z 8 cyfr binarnych o wagach od 1 do 2^7 i maksymalnej wartości równej 255. Podobnie liczba dwubajtowa oznacza liczbę złożoną z 16 cyfr binarnych o wagach od 1 do 2^{15} i maksymalnej wartości równej 65535. Dla liczby czterobajtowej składającej się z 32 bitów maksymalna wartość wynosi $2^{32}-1$ czyli 4294967295.

b) system oktalny(ósemkowy)

W systemie oktalnym r=8 oraz $d_i \in \{0,1,2,3,4,5,6,7\}$

$$X = \sum_{i=0}^{n-1} d_i 8^i . \tag{19}$$

Przykład 4.2. Znaleźć wartości dziesiętne liczb oktalnych

a) liczba oktalna $X_1=(31)_8$ liczba ta ma wartość 25 w systemie dziesiętnym,

b) liczba oktalna $X_2=(77)_8$, ma wartość 63 w systemie dziesiętnym,

c) liczba oktalna $X_3=(775)_8$, ma wartość 509 w systemie dziesiętnym,

c) system heksadecymalny

W systemie heksadecymalnym r=16 oraz $d_i \in \{0,1,2,3,4,5,6,7,A,B,C,D,E,F\}$

$$X = \sum_{i=0}^{n-1} d_i 16^i . \tag{20}$$

W systemie tym cyfry powyżej 10 oznaczono kolejnymi literami alfabetu, i tak
A=10, B=11, C=12, D=13, E=14 i F=15.

Przykład 4.3. Określić wartość dziesiętną danych liczb heksadecymalnych

a) $X_1=(2A)_{16}$, liczba ta ma wartość 42 w systemie dziesiętnym,

b) $X_2=(FF)_{16}$, ma wartość 255 w systemie dziesiętnym,

c) $X_3=(ABC)_{16}$, ma wartość 2748 w systemie dziesiętnym.

5. Konwersje międzysystemowe

Konwersja międzysystemowa to operacja znajdowania reprezentacji liczby X w pewnym systemie liczbowym, gdy dana jest reprezentacja tej liczby w innym systemie liczbowym.

Rozpatrzymy kolejno następujące konwersje:

- konwersję z systemu binarnego naturalnego do systemu dziesiętnego i z dziesiętnego do binarnego,

-konwersję ułamków z systemu binarnego do systemu dziesiętnego i z systemu dziesiętnego do systemu binarnego,

-konwersję z systemu oktalnego do systemu dziesiętnego i z systemu dziesiętnego do systemu oktalnego,

-konwersję z systemu heksadecymalnego do systemu dziesiętnego i z systemu dziesiętnego do systemu heksadecymalnego,

-konwersję z systemu binarnego do systemu oktalnego i z systemu oktalnego do systemu binarnego,

-konwersję z systemu binarnego do systemu heksadecymalnego i z systemu heksadecymalnego do systemu binarnego.

Zasadniczo konwersję międzysystemową dla systemów ze stałą podstawą można wykonać stosując procedurę konwersji przedstawioną poniżej, jednakże dla niektórych konwersji międzysystemowych istnieją prostsze metody. Rozpatrzymy najpierw ogólną procedurę konwersji.

Oznaczmy przez r podstawę systemu ze stałą postawą, z którego dokonujemy konwersji, a przez R podstawę systemu, do którego ma następować konwersja. Procedura konwersji polega na dzieleniu liczby przedstawionej przy podstawie r przez liczbę R,

przedstawioną przy podstawie *r*. Kolejne reszty z dzielenia stanowią kolejne cyfry reprezentacji przy podstawie *R*. Cyfra najmłodsza (cyfra o najmniejszej wadze) jest otrzymywana jako pierwsza.

5.1 System binarny ⇔ system dziesiętny

a) *konwersja z systemu binarnego naturalnego do systemu dziesiętnego*

Stosuje się tutaj zależność (17)

$$X = \sum_{i=0}^{n-1} d_i 2^i .$$

Przykład 5.1 Znaleźć reprezentacje dziesiętne liczb binarnych 11,110 i 11000.

Liczbie binarnej $X_1 = 11$ w systemie dziesiętnym odpowiada liczba $1 \cdot 2^1 + 1 \cdot 2^0 = 3$, $X_2 = 110$ odpowiada liczba 6, a $X_3 = 11000$ liczba 24.

b) *konwersja z systemu dziesiętnego do systemu binarnego naturalnego*

Stosujemy tutaj ogólny algorytm konwersji, o którym wspomniano wyżej, dla *r*=10, i *R*=2.

Przykład 5.2. Znaleźć reprezentacje binarne liczb 30 i 23.

X	*R*	reszta	X	*R*	reszta
30	:2	0	23	:2	1
15	:2	1	11	:2	1
7	:2	1	5	:2	1
3	:2	1	2	:2	0
1	:2	1	1	:2	1
0			0		

Liczba dziesiętna 30 ma postać binarną 11110, a liczbie 23 odpowiada liczba binarna 10111.

c) *konwersja ułamków z systemu binarnego do systemu dziesiętnego*

$$X = \sum_{i=-m}^{0} d_i 2^i . \tag{21}$$

Należy zauważyć, że powyższym wzorze wagi są ułamkowe i wynoszą kolejno $2^{-1}, 2^{-2}, 2^{-3}, \ldots$. Zależność (21) można zapisać również w nieco wygodniejszej postaci

$$X = \sum_{i=0}^{m} d_i 2^{-i} . \tag{22}$$

Tak więc np. ułamkowi binarnemu 0.11 odpowiada ułamek dziesiętny 0.75, a ułamkowi binarnemu 0.011 wartość 0.375.

d) *konwersja ułamków z systemu dziesiętnego do systemu binarnego*

Algorytm konwersji tej przedstawimy na przykładzie.

Przykład 5.3.

Dany jest ułamek dziesiętny należy znaleźć jego reprezentację binarną. Należy tutaj zauważyć, że niektóre ułamki dziesiętne mają nieskończone rozwinięcia binarne, stąd przy konwersji takich ułamków musimy ograniczyć się do pewnej skończonej liczby cyfr binarnych. Zamienimy tutaj ułamki 0.375, 0.625 i 0.446 na postać binarną.

0. | 375 | *x* 2	0. | 625 | *x* 2	0. | 446 | *x* 2
0 | 750 | *x* 2	1 | 250 | *x* 2	0 | 892 | *x* 2
1..| 500 | *x* 2	0 | 500 | *x* 2	1 | 784 | *x* 2
1 | 000 | *x* 2	1 | 000	1 | 568 | *x* 2
		1 | 136 | *x* 2
		0 | 272 | *x* 2
		0 | 544 | *x* 2
		1 | 088 | *x* 2
		0 | 176 |

Otrzymujemy więc następujące reprezentacje binarne :

dla 0.375 0.011,

dla 0.625 0.101

dla 0.446 0.01000010.

Należy zauważyć, iż dla 0.446 wyznaczono tylko pierwszych 8 cyfr binarnych rozwinięcia.

5.2 System oktalny $\Leftrightarrow$ system dziesiętny

a) *konwersja z systemu oktalnego do systemu dziesiętnego*

Konwersję tę można zrealizować stosując (19)

$$X = \sum_{i=0}^{n-1} d_i 8^i .$$

Przykładowo dla liczby oktalnej $X = (1275)_8$ otrzymujemy następującą wartość dziesiętną

$$X = 1 \cdot 8^3 + 2 \cdot 8^2 + 7 \cdot 8^1 + 5 \cdot 8^0 = 701 .$$

b) *konwersja systemu dziesiętnego do systemu oktalnego*

Konwersję tę można wykonać stosując ogólną procedurę konwersji przez dzielenie przez R=8.

Przykład 5.4. Zrealizować konwersję liczby dziesiętnej X=701 na postać oktalną.

X	R	reszta
701	:8	5
87	:8	7
10	:8	2
1	:8	1
0		

Otrzymujemy więc liczbę oktalną $(1275)_8$.

5.3 System heksadecymalny ⇔ system dziesiętny

a) konwersja z systemu heksadecymalnego do systemu dziesiętnego

Konwersję ta może być zrealizowana przy zastosowaniu (20), czyli

$$X = \sum_{i=0}^{n-1} d_i 16^i .$$

Przykład 5.5.. Zrealizować konwersję liczby $X = (2BD)_{16}$ na postać dziesiętną.

Stosując powyższą zależność otrzymujemy

$$X = 2 \cdot 16^2 + B \cdot 16^1 + D \cdot 16^0 = 2 \cdot 16^2 + 11 \cdot 16^1 + 13 \cdot 16^0 = 701$$

b) konwersja z systemu dziesiętnego do systemu heksadecymalnego

Postać heksadecymalną liczby otrzymujemy realizując dzielenie przez R=16.

X	R	reszta
701	:16	13=D
43	:16	11=B
2	:16	2
0		

5.4 System binarny ⇔ system oktalny

Konwersje te można wykonać bez konieczności wykonywania dzielenia, co jest ważnym czynnikiem stanowiącym o przydatności systemu oktalnego w praktyce informatycznej. System umożliwia 3-krotne skrócenie zapisu liczby binarnej, co ma istotne znaczenie dla zewnętrznego reprezentowania liczb binarnych. Możliwość takiej konwersji wynika ze związku między podstawami systemu oktalnego i systemu binarnego, gdyż $8 = 2^3$. Umożliwia to podział liczby binarnej na segmenty 3-bitowe i traktowanie każdego segmentu jako cyfry w systemie oktalnym.

a) konwersja z systemu binarnego do systemu oktalnego

Aby zrealizować tę konwersję zastosujemy metodę opisaną powyżej.

Przykład 5.6.Dana jest 12-bitowa liczba binarna 111101001101, po podzieleniu jej na 3-bitowe segmenty i przypisaniu każdemu segmentowi odpowiedniej liczby oktalnej otrzymamy oktalną reprezentację liczby

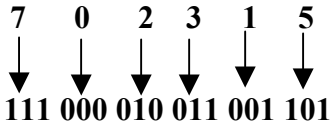
$$\begin{array}{c}(\,111\,|101|\,001\,|\,101)_2\\ (\;\;7\;\;\;5\;\;\;1\;\;\;\;\;5\;)_8\end{array}$$

czyli $(111101001101)_2=(7515)_8$.

b) konwersja z systemu oktalnego do systemu binarnego

Konwersja ta stanowi odwrócenie poprzedniej, czyli aby uzyskać reprezentację binarną, każdej cyfrze oktalnej należy przypisać jej 3-bitową reprezentację binarną,

Przykład 5.7. Dana jest 6-cyfrowa liczba oktalna $(702315)_8$, należy wyznaczyć jej reprezentację binarną



5.5 System binarny ⇔ system heksadecymalny

Konwersje te można wykonać bez konieczności wykonywania dzielenia podobnie jak w przypadku systemu oktalnego. System heksadecymalny umożliwia 4-krotne skrócenie zapisu liczby binarnej, co ma istotne znaczenie dla zewnętrznego reprezentowania liczb binarnych. Możliwość takiej konwersji wynika ze związku między podstawami systemu

heksadecymalnego i systemu binarnego, gdyż $8 = 2^4$. Umożliwia to podział liczby binarnej na segmenty 4-bitowe i traktowanie każdego segmentu jako cyfry w systemie heksadecymalnym.

a) konwersja z systemu binarnego do systemu heksadecymalnego

Przykład 5.8. Dana jest 12-bitowa liczba binarna 111101001101, po podzieleniu jej na 4-bitowe segmenty i przypisaniu każdemu segmentowi odpowiedniej liczby heksadecymalnej otrzymamy heksadecymalną reprezentację liczby

$$\begin{matrix}(1111 \mid 0100 \mid 1101)_{16} \\ F \quad 4 \quad D\end{matrix}$$

czyli $(111101001101)_2 = (F4D)_{16}$.

b) konwersja z systemu heksadecymalnego do systemu binarnego

Konwersja ta stanowi odwrócenie poprzedniej, czyli aby uzyskać reprezentację binarną, każdej cyfrze heksadecymalnej należy przypisać jej reprezentację binarną.

Przykład 5.9. Dana jest 6-cyfrowa liczba hekasdecymakna $(F0E30C)_{16}$, należy wyznaczyć jej reprezentację binarną

$$\begin{matrix}\mathbf{F} & \mathbf{0} & \mathbf{E} & \mathbf{3} & \mathbf{0} & \mathbf{C} \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \mathbf{1111} & \mathbf{0000} & \mathbf{1110} & \mathbf{0011} & \mathbf{0000} & \mathbf{1100}\end{matrix}$$

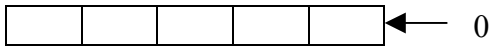
6. Przesunięcie liczby binarnej

Zanim przystąpimy do omawiania poszczególnych działań wprowadzimy pojęcie rejestru. Rejestr w technice cyfrowej to urządzenie mogące przechowywać liczbę binarną o określonej długości np. 8, 16, 32 bity. Mówimy wtedy odpowiednio o rejestrze 8-, 16-, 32-bitowym, a ogólnie *n*-bitowym.

Przesunięcie liczby binarnej w rejestrze oznacza, że poszczególne jej bity przechodzą (są przepisywane) do innych komórek rejestru lub wychodzą poza rejestr. Poniżej podamy sześć rodzajów przesunięć: przesunięcia logiczne i arytmetyczne w lewo i w prawo oraz przesunięcia cykliczne w lewo i w prawo. Poszczególne rodzaje przesunięć przedstawiono tak jak funkcjonują odpowiednie operatory w języku wyższego rzędu np. VHDL, a nie rozkazy mikroprocesora, wtedy postać tych przesunięć jest nieco inna ze względu na udział znacznika przeniesienia.

a) *przesunięcie logiczne w lewo(ang. shift-left logical-SLL)*

Przesunięcie takie polega na tym, że wszystkie bity liczby są przesuwane w lewo, przy czym skrajny bit z lewej strony (bit o najwyższej wadze) znika, a na miejsce bitu skrajnego z prawej strony (bitu o najniższej wadze) wchodzi zero.



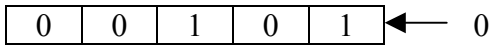
Wartość liczby *Y*, powstałej w wyniku przesunięcia liczby *X* zapisanej w rejestrze *n*- bitowym po przesunięciu *k* pozycji w lewo, można określić następująco:

$$Y = \left| 2^k \cdot X \right|_{2^n} \qquad (23)$$

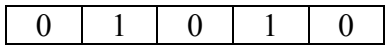
Przesunięcie o jedną pozycję odpowiada mnożeniu liczby binarnej przez 2, o ile bit o najwyższej wadze jest równy 0, jeśli zaś bit ten równy jest 1, operacja jest wykonywana modulo 2^n, co określa się czasami jako operację modulo długość rejestru.

Przykład 6.1. Przesunięcie rejestru logiczne o jedną pozycję w lewo.

Zawartość rejestru przed przesunięciem, *X*=5,

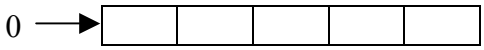


Zawartość rejestru po przesunięciu o jedną pozycję w lewo, $Y = 2 \cdot X = 10$.



b) *przesunięcie logiczne w prawo(ang. shift-right logical-SRL)*

Przesunięcie takie polega na tym, że wszystkie bity liczby są przesuwane w prawo, przy czym skrajny bit z prawej strony (bit o najniższej wadze) znika, a na miejsce bitu skrajnego z lewej strony (bitu o najwyższej wadze) wchodzi zero.



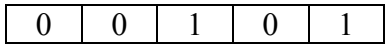
Wartość liczby *Y*, powstałej w wyniku przesunięcia liczby *X* zapisanej w rejestrze *n*-bitowym, o *k* pozycji w prawo jest równa

$$Y = \left\lfloor X \big/ 2^k \right\rfloor, \qquad (24)$$

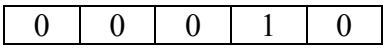
$\lfloor \cdot \rfloor$ oznacza część całkowitą argumentu. Operacja ta odpowiada więc dzieleniu całkowitemu.

Przykład 6.2. Przesunięcie logiczne rejestru o jedną pozycję w prawo.

Zawartość rejestru przed przesunięciem, *X*=5,

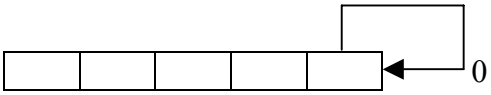


Zawartość rejestru po przesunięciu o jedną pozycję w prawo, $Y = \lfloor X / 2 \rfloor = \lfloor 5 / 2 \rfloor = 2$.



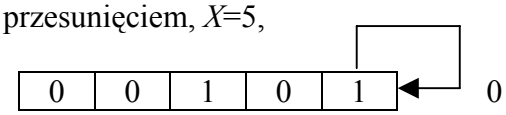
c) *przesunięcie arytmetyczne w lewo(ang. shift-left arithmetic-SLA)*

Przesunięcie takie polega na tym, że wszystkie bity liczby są przesuwane w lewo, przy czym skrajny bit z lewej strony (bit o najwyższej wadze) znika, a na miejsce bitu skrajnego z prawej strony (bitu o najniższej wadze) wchodzi ten sam bit.

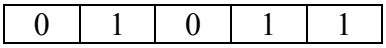


Przykład 6.3. Przesunięcie rejestru arytmetyczne o jedną pozycję w lewo.

Zawartość rejestru przed przesunięciem, X=5,

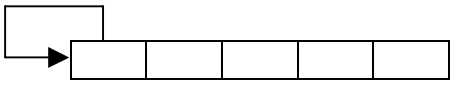


Zawartość rejestru po przesunięciu arytmetycznym o jedną pozycję w lewo, $Y = 2 \cdot X + |X|_2 = 10 + 1 = 11$.



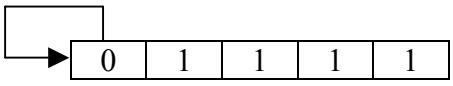
d) *przesunięcie arytmetyczne w prawo(ang. shift-right arithmetic-SRA)*

Przy realizacji takiego przesunięcia wszystkie bity liczby są przesuwane w lewo, przy czym skrajny bit z lewej strony (bit o najwyższej wadze) wchodzi na tę samą pozycję , a bitu skrajny z prawej strony (bit o najniższej wadze) znika.

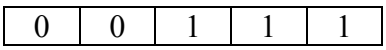


Przykład 6.4. Przesunięcie rejestru arytmetyczne o jedną pozycję w prawo.

Zawartość rejestru przed przesunięciem, X=15,

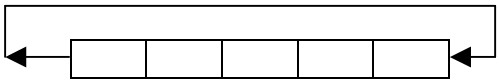


Zawartość rejestru po przesunięciu arytmetycznym o jedną pozycję w prawo, $Y = 7$.

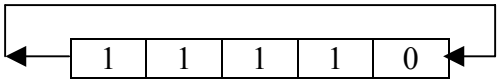


e) *przesunięcie cykliczne w lewo(ang. rotate-left logical)*

Przy realizacji takiego przesunięcia wszystkie bity liczby są przesuwane w lewo, przy czym skrajny bit z lewej strony (bit o najwyższej wadze) wchodzi na pozycję bitu skrajnego z lewej strony (bitu o najniższej wadze).

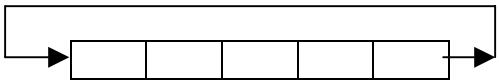


Przykład 6.5. Przesunięcie rejestru cykliczne o jedną pozycję w lewo.
Zawartość rejestru przed przesunięciem, *X*=30, po przesunięciu *Y*=29.

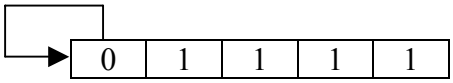


f) *przesunięcie cykliczne w prawo(ang. rotate-right logical)*

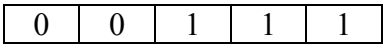
Przy realizacji takiego przesunięcia wszystkie bity liczby są przesuwane w prawo, przy czym skrajny bit z prawej strony (bit o najniższej wadze) wchodzi na pozycję bitu skrajnego z lewej strony (bitu o najwyższej wadze).



Przykład 6.6 Przesunięcie rejestru arytmetyczne o jedną pozycję w prawo.
Zawartość rejestru przed przesunięciem, *X*=15,



po przesunięciu *Y*= 7.



7. Działania arytmetyczne w systemie binarnym naturalnym

Poniżej przedstawione zostaną działania takie jak dodawanie, odejmowanie, mnożenie i dzielenie.

7.1 Dodawanie

Dodawanie dwóch liczb *n*-bitowych *A* +*B* można przedstawić następująco:

$$\begin{array}{rl} & a_{n-1}a_{n-2}a_{n-3}.....a_0 \\ + & b_{n-1}b_{n-2}b_{n-3}.....b_0 \\ \hline s_n & s_{n-1}s_{n-2}s_{n-3}.....s_0 \end{array} \longleftarrow c_{in} \qquad (25)$$

Dodawanie takie odbywa się poprzez dodawanie kolejnych cyfr obu liczb (bitów) począwszy od najmłodszej pary bitów przy uwzględnieniu przy każdym dodawaniu przeniesienia z poprzedniej pozycji. Zakłada się zwykle, że przeniesienie do najmłodszej pozycji $c_{in} = 0$. Poniżej podamy w Tabeli 1 reguły dodawania cyfr binarnych na jednej pozycji bez uwzględniania przeniesienia, a w Tabeli 2 z uwzględnieniem przeniesienia.

Tabela. 1 Reguły dodawania binarnego na jednej pozycji bez uwzględniania przeniesienia

Dodawane liczby	Suma	Przeniesienie
0+0	0	0
0+1	1	0
1+0	1	0
1+1	0	0

Tabela. 2 Reguły dodawania binarnego na jednej pozycji z uwzględnieniem przeniesienia.

a_i	b_i	$c_{in,i}$	s_i	$c_{out,i}$
0	0	0	0	0
0	0	1	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

$c_{in,i}$ - oznacza przeniesienie wchodzące do *i*-tej pozycji, s_i -sumę, a $c_{out,i}$ - oznacza przeniesienie wychodzące z *i*-tej pozycji.

Przykład 7.1. Dodawanie dwóch liczb binarnych

$$\begin{array}{rr} & 101 \\ + & 111 \\ \hline & 1100 \end{array} \qquad \begin{array}{rr} & 1101 \\ + & 1110 \\ \hline & 11011 \end{array}$$

Warto zauważyć, że przy sumowaniu dwóch liczb *n*-bitowych ich suma może wymagać *n*+1 bitów, a ogólnie dla sumowania *m* składników wymaganą długość rejestru n_s dla ich sumy można określić w sposób następujący

$$n_s = \lfloor \log_2 m(2^n - 1) \rfloor + 1 . \tag{26}$$

7.2 Odejmowanie

Odejmowanie dwóch liczb *n*-bitowych *A* - *B* dla można przedstawić następująco:

$$\begin{array}{r} a_{n-1}a_{n-2}a_{n-3}.....a_0 \\ -\quad b_{n-1}b_{n-2}b_{n-3}.....b_0 \\ \hline \delta_n\delta_{n-1}\delta_{n-2}\delta_{n-3}.....\delta_0 \end{array} \qquad (27)$$

Odejmowanie takie odbywa się poprzez odejmowanie kolejnych cyfr obu liczb (bitów) począwszy od najmłodszej pary bitów przy uwzględnieniu przy każdym odejmowaniu pożyczki (przeniesienia o wartości ujemnej) z poprzedniej pozycji. Różnica może być ujemna, stąd pojawia się problem dodatkowego bitu reprezentującego znak. Zwykle przyjmuje się, że znak minus jest reprezentowany przez liczbę 1, a znak plus przez 0. Wartość rezultatu odejmowania wg (27) można więc określić następująco :

$$A - B = -\delta_n \cdot 2^n + \sum_{i=0}^{n-1} \delta_i 2^i \qquad (28)$$

Poniżej podamy w Tabeli 3 reguły odejmowania cyfr binarnych.

Tabela. 3 Reguły odejmowania binarnego na jednej pozycji z uwzględnieniem przeniesienia.

a_i	b_i	$c_{in,i}$	δ_i	$c_{out,i}$
0	0	0	0	0
0	1	0	1	-1
1	0	0	1	0
1	1	0	0	0
0	0	-1	1	-1
0	1	-1	0	-1
1	0	-1	0	0
1	1	-1	1	-1

Przykład 7.2. Odejmowanie dwóch liczb binarnych

$$\begin{array}{r} 10000 \\ -01011 \\ \hline 00101 \end{array} \qquad \begin{array}{r} 101010 \\ -010111 \\ \hline 010011 \end{array}$$

Warto zauważyć, iż przy odejmowaniu na danej pozycji, poza odejmowaniami 0-0 i 1-1 przy braku pożyczki z poprzedniej pozycji, których rezultat jest oczywisty, występuje odejmowanie, wtedy jeśli cyfra na danej pozycji jest równa 0, to odejmujemy od 1, a jeżeli cyfra jest równa 1, to od 2 (podobnie jak w systemie dziesiętnym odpowiednio od 9 i od 10).

7.3. Mnożenie

Mnożenie liczb binarnych $A \cdot B$ można zrealizować poprzez wielokrotne przesunięcie mnożnej i warunkowe sumowanie przesunietej mnoznej w zależnosci od wartosci odpowiedniego bitu mnoznika. Poniżej przedstawiono schematycznie operację mnożenia.

$$\begin{array}{r} a_3 a_2 a_1 a_0 \\ x \quad b_3 b_2 b_1 . b_0 \\ \hline p_3^{(1)} p_2^{(1)} p_1^{(1)} . p_0^{(1)} \\ p_3^{(2)} p_2^{(2)} p_1^{(2)} . p_0^{(2)} \\ p_3^{(3)} p_2^{(3)} p_1^{(3)} . p_0^{(3)} \\ p_3^{(4)} p_2^{(4)} p_1^{(4)} . p_0^{(4)} \\ \hline p_7 \quad p_6 \quad p_5 \quad . p_4 \quad p_3 \quad p_2 \quad p_1 \quad p_0 \end{array}.$$

Przykład 7.3. Obliczyć iloczyn liczb *A*=1111 i *B*=1001

$$\begin{array}{r} 1111 \\ x\ \underline{1001} \\ 1111 \\ \underline{1111\ \ \ } \\ 10000111 \end{array}$$

Mnożenie takie można opisać następującą zależnością

$$P_0 = 0,$$

$$P_{i+1} = P_i + b_i \cdot A \cdot 2^i, \quad i = 0,1,2,...,k-1, \tag{29}$$

Algorytm mnożenia w oparciu o (29) przedstawiono poniżej.

*Algorytm 1 (*mnożenie liczb *k*-bitowych metodą dodaj-przesuń, ang. shift-add)

Krok 0. Ładuj mnożną *A* do rejestru *X* i mnożnik *B* do rejestru *Y*

Krok 1. Ładuj 0 do rejestru *I* .

Krok 2. Powtarzaj kroki 2a i 2b dla *i*=0,1,2,...,*k*-1.

Krok 2a. Jeżeli $Y_i = 1$ wtedy

$$I \leftarrow I + X$$

Krok 2b. $X \leftarrow 2 \cdot X$

Krok 3. Stop.

Przykład 7.4. Przedstawimy poniżej stany rejestrów *X*, *Y* i *I* dla przykładowego mnożenia liczb 3-bitowych zrealizowanego w oparciu o *Algorytm 1*.W kroku 2a *i*-ty bit mnożnika Y_i jest

sprawdzany przy użyciu przesunięcia w prawo. Bit ten jest zapisywany do 1-bitowego-rejestru znacznika.

```
     101
   x 111
     101
    101
   101
 10000111
```

Krok 0.

0	0	0	1	1	1

$Y \leftarrow B$

0	0	0	1	0	1

$X \leftarrow A$

Krok 1.

0	0	0	0	0	0

$I \leftarrow 0$

- -

I.

Krok 2a.

0	0	0	0	1	1

Znacznik

1

0	0	0	1	0	1

$I \leftarrow I + X$

Krok 2b.

0	0	1	0	1	0

$X \leftarrow 2 \cdot X$

- -

II.

Krok 2a.

0	0	0	0	0	1

Znacznik

1

0	0	1	1	1	1

$I \leftarrow I + X$

Krok 2b.

0	1	0	1	0	0	$X \leftarrow 2 \cdot X$

III.

Krok 2a.

Znacznik

0	0	0	0	0	0		1

1	0	0	0	1	1	$I \leftarrow I + X$

Krok 2b.

1	0	1	0	0	0	$X \leftarrow 2 \cdot X$

Mnożenie typu przesuń-dodaj jest procesem czasochłonnym szczególnie dla większych liczb. Poniżej pokażemy inny przykładowy algorytm mnożenia, który może prowadzić do znacznie szybszej realizacji w układzie cyfrowym. Będzie to tzw. algorytm rosyjskich wieśniaków (ang. algorithm of Russian peasants, RP).

Przykład 7.5. Zrealizować mnożenie liczb X=24 i Y=25 stosując algorytm RP.

Y	X	
25	24	←
12	48	
6	96	
3	192	←
1	384	←
	600	

W algorytmie tym w pierwszej kolumnie wypisujemy ilorazy całkowite z kolejnych dzieleń mnożnika przez dwa, a w kolumnie drugiej kolejne iloczyny mnożnej, pomnożonej przez 2. Iloczyn uzyskujemy dodając te iloczyny X, przy których znajduje się liczba nieparzysta w pierwszej kolumnie. Jak widać, sumowane są te iloczyny, dla których w reprezentacji binarnej odpowiedni bit mnożnika jest równy 1. Algorytm ten może być zrealizowany w systemie binarnym w sposób równoległy poprzez warunkowe sumowanie słów binarnych zawierających odpowiednio przesuniętą mnożną.

7.4 Dzielenie

Dzielenie binarne w formie pisemnej może być zrealizowane przy użyciu algorytmu dzielenia stosowanego dla innych podstaw systemu liczbowego(np. r=10).

Przykład 7.6. Zrealizować dzielenie 111011:101, odpowiada ono dzieleniu 59:5 w systemie dziesiętnym.

```
  1011
111011 : 101
- 101
  1001
 -0101
  1001
 - 0101
   100
```

Uzyskaliśmy więc wynik dzielenia całkowitego równy 1011 oraz resztę 100.

Dzielenie binarne realizowane maszynowo (układowo lub programowo) jest działaniem znacznie trudniejszym niż mnożenie binarne. W przypadku mnożenia binarnego operandów *k*-bitowych, otrzymywany wynik jest w ogólnym przypadku 2*k*-bitowy. Natomiast przy dzieleniu binarnym długość ilorazu w bitach jest znacznie trudniejsza do przewidzenia. Rozpatrzymy teraz bliżej relacje między długościami binarnymi dzielnej, dzielnika, reszty i ilorazu. Przykładowo, rozważmy dzielenie całkowite liczby 2*k*-bitowej przez *k*- bitową. Aby dzielenie takie było wykonalne, reprezentacja binarna reszty otrzymywanej w trakcie procesu dzielenia nie może przekraczać 2*k* bitów, gdyż w przeciwnym przypadku nie byłoby możliwe jej odjęcie od dzielnej reprezentowanej na 2*k*-bitach. Jednak przy dzielniku *k*-bitowym iloraz może być dłuższy niż *k* bitów, a więc reszta mogłaby w ogólnym przypadku przekraczać 2*k*-bitów. Stąd dzielenie przy takich długościach rejestrów nie jest bezpośrednio wykonalne. Przyjmując więc, że reszta ma być co najwyżej 2*k*-bitowa, przy *k*-bitowym dzielniku, iloraz nie może mieć długości większej niż *k*-bitów. Jeżeli jednak okazuje się, że iloraz wymaga więcej niż *k*-bitów, sytuacja taka traktowana jest jako nadmiar.
Przyjmijmy następujące oznaczenia:

z	dzielna	$z_{2k-1}z_{2k-2}.....z_1z_0$,
d	dzielnik	$d_{k-1}d_{k-2}.....d_1d_0$,
q	iloraz	$q_{k-1}q_{k-2}.....q_1q_0$,
s	reszta	$s_{k-1}s_{k-2}.....s_1s_0$,

Podstawowe równanie dla dzielenia ma postać

$$z = d \times q + s \tag{30}$$

Pokażemy teraz warunek, który musi być spełniony, aby nie wystąpił nadmiar podczas dzielenia. Nadmiar nie wystąpi, jeżeli liczba reprezentowana przez *k* pierwszych bitów dzielnej(licząc od bitu o najwyższej wadze) jest mniejsza od dzielnika. Warunek ten można to zapisać następująco:

$$\left\lfloor \frac{z}{2^k} \right\rfloor < d \ . \tag{31}$$

Rezultatem dzielenia po prawej stronie wzoru (31) jest liczba reprezentowana przez najstarsze *k* bitów dzielnej. Jeżeli liczba ta jest mniejsza od dzielnika *d*, to jej iloraz przez *d* będzie równy zeru czyli iloraz będzie co najwyżej *k* bitowy.

Przykład 7.7. Zrealizować dzielenie 101100:111, odpowiada ono dzieleniu 44:7 w systemie dziesiętnym.

```
   110  ------>  6
 101100 : 111
-111
  1000
 -0111
    10  ------>  2
```

W tym przypadku spełniony jest warunek (31). Otrzymujemy iloraz równy 6 oraz resztę równą 3.

Dzielenie binarne liczb całkowitych, przedstawione powyżej, może być zrealizowane przy zastosowaniu następującego algorytmu

$$s^{(j)} = s^{(j-1)} - q_{k-j} \cdot d \cdot 2^{k-j} \ , \ j=1,2,..,k, \tag{32}$$

przy $s^{(0)} = z$. Cyfry dzielnika są dobierane w taki sposób, aby kolejna reszta była nieujemna.

Przykład 7.8. Stosując (32) dla *k*=3

$$s^{(1)} = s^{(0)} - q_2 \cdot d \cdot 2^2 ,$$

$$s^{(2)} = s^{(1)} - q_1 \cdot d \cdot 2^1 ,$$

$$s^{(3)} = s^{(2)} - q_0 \cdot d \cdot 2^0 .$$

Przyjmując jak w przykładzie powyżej, $s^{(0)} = z = 45$, *d*=7, otrzymujemy

$$s^{(1)} = 44 - 1 \cdot 7 \cdot 2^2 = 16 , \qquad q_2 = 1 ,$$

$$s^{(2)} = 16 - 1 \cdot 7 \cdot 2 = 2, \qquad q_1 = 1,$$

$$s^{(3)} = 2 - 0 \cdot 7 \cdot 2^0 = 2, \qquad q_0 = 0.$$

Czyli iloraz całkwity w formie binarnej ma postać $(110)_2$=6

8. Wybrane reprezentacje liczb ze znakiem

Jak pokazano powyżej w pkt.6.3 przy odejmowaniu pojawia się problem reprezentowania liczb ze znakiem. Forma reprezentacji liczb ze znakiem powinna być taka aby w możliwie dużym stopniu ułatwiała układową realizację operacji arytmetycznych na takich liczbach. Trzy najczęściej stosowane reprezentacje to:

-reprezentacja znak-moduł (kod znak-moduł,ang. sign-magnitude representation),

-reprezentacja z uzupełnieniem do 1(kod U1, reprezentacja ze zmniejszonym uzupełnieniem do podstawy systemu liczbowego, ang. one's complement representation),

-reprezentacja z uzupełnieniem do 2(kod U2, reprezentacja z uzupełnieniem do podstawy systemu liczbowego, ang. one's complement representation).

Zanim przejdziemy do omawiania poszczególnych reprezentacji, omówimy najpierw ogólną postać reprezentacji stałoprzecinkowej. Reprezentacja stałoprzecinkowa liczb ze znakiem ma postać

$$X \leftrightarrow (x_{n-1} \mid x_{n-2} \ldots\ldots x_1 x_0). \tag{33}$$

Zwykle x_{n-1} rezerwuje się na znak liczby; znak liczby przyjmuje wartość

$$x_{n-1} = \begin{cases} 0 & \text{dla} \quad X \geq 0 \\ r-1 & \text{dla} \quad X < 0 \end{cases}, \tag{34}$$

cyfry $x_{n-2} \ldots\ldots x_1 x_0$ reprezentują moduł liczby.

Reprezentacja stałoprzecinkowa może zawierać przecinek (kropkę) , aby oddzielić część całkowitą liczby od części ułamkowej. Przecinek (kropka) może być umieszczony między dwoma dowolnymi cyframi reprezentacji.

8.1 Reprezentacja znak-moduł

Liczby nieujemne przy użyciu reprezentacji tej są przedstawiane następująco:

$$X \leftrightarrow (0\, x_{n-2} \ldots\ldots x_1 x_0)_r, \tag{35a}$$

a liczby ujemne

$$\overline{X} \leftrightarrow ((r-1)\, x_{n-2} \ldots\ldots x_1 x_0)_r, \tag{35b}$$

gdzie x_i dla $0 \le i \le n-2$ są cyframi danej liczby *X*. Dla reprezentacji tej, liczby o tych samych wartościach bezwzględnych mają te same cyfry, począwszy od drugiej. W systemie binarnym dla liczb nieujemnych reprezentacja ta ma postać

$$X \leftrightarrow (0\, x_{n-2}.....x_1 x_0)_2\,, \tag{36a}$$

a dla liczb ujemnych

$$\overline{X} \leftrightarrow (1\, x_{n-2}.....x_1 x_0)_2\,. \tag{36b}$$

Wartość liczby reprezentowanej przy użyciu (36a) lub (36b) można obliczyć jako

$$X = \begin{cases} \sum_{i=0}^{n-2} x_i \cdot 2^i & \text{gdy } x_{n-1} = 0 \\ -\sum_{i=0}^{n-2} x_i \cdot 2^i & \text{gdy } x_{n-1} = 1 \end{cases} \tag{37}$$

Przykład 8.1 Wyznaczyć 8-bitowe reprezentacje liczb 24 i -24 w kodzie znak-moduł.

+24	00011000
-24	10011000

8.2 Reprezentacja z uzupełnieniem do 1 (kod U1)

Liczby nieujemne dla *r*=2 przy użyciu reprezentacji tej są przedstawiane tak samo jak w kodzie znak-moduł:

$$X \leftrightarrow (0\, x_{n-2}.....x_1 x_0)\,, \tag{38a}$$

a liczby ujemne

$$\overline{X} \leftrightarrow (1\, \overline{x}_{n-2}.....\overline{x}_1 \overline{x}_0)\,, \tag{38b}$$

gdzie $\overline{x}_i = 1 - x_i$ dla $0 \le i \le n-2$, x_i są cyframi liczby dodatniej o tej samej wartości bezwzględnej co dana liczba ujemna.

Reprezentację *n*-bitową liczby ujemnej w tym kodzie otrzymuje się więc przyjmując jako znak liczbę 1 oraz dokonując negacji pozostałych cyfr (tj. zamiana zer na jedynki, a jedynek na zera) liczby dodatniej o tej samej wartości bezwzględnej co dana liczba ujemna, której reprezentacji poszukujemy. Postępowanie takie jest równoważne odejmowaniu takiej liczby od liczby $2^n - 1$ (złożonej z *n* jedynek).

Wartość liczby reprezentowanej przy użyciu (38a) lub (38b) można obliczyć jako

$$X = \begin{cases} \sum_{i=0}^{n-2} \bar{x}_i \cdot 2^i & \text{gdy } \bar{x}_{n-1} = 0 \\ -(2^n - 1 - \sum_{i=0}^{n-1} \bar{x}_i \cdot 2^i) & \text{gdy } \bar{x}_{n-1} = 1 \end{cases} . \tag{39}$$

Przykład 8.2 Wyznaczyć 8-bitowe reprezentacje liczb 24 i -24 w kodzie U1.
Reprezentacja liczby 24 jest taka sama jak w kodzie znak-moduł, czyli

+24 00011000,

a reprezentację -24 otrzymujemy dokonując negacji poszczególnych cyfr,

-24 11100111.

8.3 Reprezentacja z uzupełnieniem do 2 (kod U2)

Liczby nieujemne dla r=2 przy użyciu reprezentacji tej są przedstawiane tak samo jak w kodzie znak-moduł:

$$X \leftrightarrow (0\, x_{n-2} \ldots\ldots x_1 x_0), \tag{40a}$$

a liczby ujemne

$$\widetilde{X} \leftrightarrow ((1\, \bar{x}_{n-2} \ldots\ldots \bar{x}_1 \bar{x}_0) + 1) = (1\, \widetilde{x}_{n-2} \ldots\ldots \widetilde{x}_1 \widetilde{x}_0), \tag{40b}$$

gdzie $\bar{x}_i = 1 - x_i$ dla $0 \le i \le n-2$.

Reprezentację liczby ujemnej w tym kodzie otrzymuje się tworząc reprezentację w kodzie U1 danej liczby ujemnej i dodając następnie 1. Postępowanie takie jest równoważne odejmowaniu od liczby 2^n (złożonej z n jedynek) liczby dodatniej o tym samym module, co liczba ujemna, której reprezentacji poszukujemy.

Wartość liczby reprezentowanej przy użyciu (40a) lub (40b) można obliczyć jako

$$X = \begin{cases} \sum_{i=0}^{n-2} \widetilde{x}_i \cdot 2^i & \text{gdy } \widetilde{x}_{n-1} = 0 \\ -(2^n - \sum_{i=0}^{n-1} \widetilde{x}_i \cdot 2^i) & \text{gdy } \widetilde{x}_{n-1} = 1 \end{cases} . \tag{41}$$

Warto zauważyć, iż zamiast stosować bezpośrednio drugą z formuł w (41) można jej użyć w postaci ułatwiającej obliczenia czyli

$$-(2^n - 1 - \sum_{i=0}^{n-1} \widetilde{x}_i \cdot 2^i + 1) , \tag{42}$$

wystarczy więc zanegować cyfry liczby i następnie dodać 1.

Przykład 8.3 Wyznaczyć 8-bitowe reprezentacje liczb 24 i -24 w kodzie U2.

Reprezentacja liczby 24 jest taka sama jak w kodzie znak-moduł, czyli

+24 00011000,

a reprezentację -24 otrzymujemy dokonując najpierw negacji poszczególnych cyfr,

-24 11100111 (-24 w kodzie U1)

oraz następnie do tej reprezentacji dodajemy 1

$$\begin{array}{r} 11100111 \\ +\ \underline{00000001} \\ 11101000 \end{array}$$

czyli -24 w kodzie U2 ma tutaj postać

-24 11101000 (-24 w kodzie U2)

8.4 Podstawowe pojęcia systemów uzupełnieniowych

Poniżej podamy podstawowe pojęcia systemów uzupełnieniowych jak również wyjaśnimy sposób realizacji odejmowania w kodzie U2.

Oznaczenia:

$$R = r^n,\ Q = r^n - 1 = (r-1, r-1, ..., r-1)$$

Def. (dopełnienie liczby)

Dopełnieniem liczby X nazywamy liczbę $\overline{X} = Q - X$.

Własności dopełnienia:

1. $\overline{X} + X = Q \rightarrow \overline{X} = Q - X$ (43a)
2. $\overline{\overline{X}} = Q - \overline{X} \rightarrow \overline{X} = Q - (Q - X) = X$ (43b)

Def. (uzupełnienie liczby)

Uzupełnieniem liczby X nazywamy liczbę $X^* = R - X$.

Własności uzupełnienia:

1. $X^* = r^n - X = Q - X + 1 = \overline{X} + 1$ (44a)
2. $X^* + X = r^n$ (44b)

Należy zauważyć, iż dla systemu binarnego (r=2) reprezentacja liczby r^n ma same zera na n bitach.

Własności (43a) i (43b) są bardzo istotne z punktu widzenia przydatności systemów uzupełnieniowych. Z własności (43a) wynika, że wyznaczenie uzupełnienia liczby X wymaga

znalezienia dopełnienia, co wymaga tylko zanegowania wszystkich bitów X, i następnie dodania 1.

Własność (43b) pozwala na zastąpienie odejmowania dodawaniem uzupełnienia. Przy odejmowaniu mamy

$$-X+X=0,$$

natomiast przy dodawaniu uzupełnienia

$$X^* + X = r^n - X + X = r^n \tag{45}$$

Ponieważ reprezentacja r^n ma same zera na n pozycjach, liczba ta na n pozycjach ma taką samą reprezentację jak liczba 0. Widać stąd, że zastosowanie liczby X^* zamiast $-X$ daje ten sam wynik, czyli zamiast odejmować liczbę X możemy dodawać liczbę X^*.

Przykład 8.4. Zrealizować odejmowanie X=24 i Y=15 stosując 8-bitowe reprezentacje binarne. Przedstawimy te działania w systemie dziesiętnym. Mamy więc

$$Y^* = 2^8 - 15 = 241,$$

$$D = X + Y^* = \left|24 + 241\right|_{2^8} = 9.$$

Zastosowanie operacji modulo 2^n odpowiada ograniczeniu reprezentacji liczby binarnej do n-bitów.

9. Wybrane działania arytmetyczne w systemach uzupełnieniowych

Poniżej zostanie przedstawione dodawanie i odejmowanie w kodzie U2.

9.1 Dodawanie w systemie U2

Niech liczby X i Y mają następujące reprezentacje w U2

$$X \leftrightarrow \mathsf{X} = (\tilde{x}_{n-1}, \tilde{x}_{n-2}, \ldots, \tilde{x}_1, \tilde{x}_0)$$

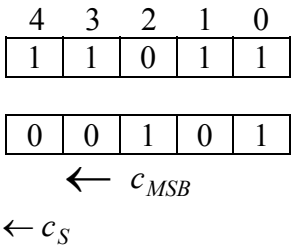
$$Y \leftrightarrow \mathsf{Y} = (\tilde{y}_{n-1}, \tilde{y}_{n-2}, \ldots, \tilde{y}_1, \tilde{y}_0)$$

Dodawanie odbywa się w dwóch krokach :

Krok 1. Dodaj kolejne cyfry poczynając od najmniej znaczącej pary aż do pary najbardziej znaczącej (cyfr znaku), zakładając zerowe przeniesienie z prawej strony.

Krok 2. Porównaj przeniesienie wchodzące do pozycji znaku z przeniesieniem wychodzącym z pozycji znaku. Jeżeli obydwa są równe 0 lub obydwa 1, to przeniesienie z pozycji znaku należy zignorować i rezultat jest w poprawnej formie U2. Jeżeli przeniesienia różnią się, oznacza to, że wystąpił nadmiar i rezultat jest niepoprawny.

Przykład 9.1. Przeniesienia do pozycji znaku (c_{MSB} **)** i z pozycji znaku (c_S **)**.



Rejestry przedstawione powyżej mają długość *n*=5, a poszczególne pozycje są numerowane od 0 do *n*-1 od prawej strony. c_{MSB} jest więc przeniesieniem wychodzącego z pozycji 3 i wchodzącym do pozycji 4, a c_S wychodzącym z pozycji 5**.**

Przykład 9.2. Dodawanie w kodzie U2.

a) dwie liczby dodatnie

+13	0.01101	
+11	+0.01011	
+24	0.11000	$c_{MSB}=0,\ c_S=0$

b) dwie liczby ujemne

-13	1.10011	(-13 w U2)
+(-11)	+1.10101	(-11 w U2)
-24	1.01000	
		$c_{MSB}=1,\ c_S=1$

c) liczby o różnych znakach

+13	0.01101	
+ (-11)	+1.10101	(-11 w U2)
+2	0.00010	
		$c_{MSB}=1,\ c_S=1$
(-13)	1.10011	(-13 w U2)
+ 11	+0.01011	
-2	1.11110	
		$c_{MSB}=0,\ c_S=0$

Rozpatrzone zostaną teraz poszczególne przypadki różnych kombinacji znaków liczb X i Y.

Przypadek 1.

$$X>0 \quad \text{i} \quad Y>0$$

Reprezentacje X i Y mają w tym przypadku następującą formę

$$X \leftrightarrow \mathsf{X} = (0, \tilde{x}_{n-2}, \ldots, \tilde{x}_1, \tilde{x}_0)$$
$$Y \leftrightarrow \mathsf{Y} = (0, \tilde{y}_{n-2}, \ldots, \tilde{y}_1, \tilde{y}_0)$$

Nie będzie w tym przypadku przeniesienia z pozycji znaku ($c_S = 0$), gdyż obydwa znaki są równe 0. Powstająca suma będzie dodatnia i w poprawnej formie $X+Y$, jeżeli

$$X + Y < 2^{n-1},$$

czyli jeżeli nie będzie nadmiaru to $c_{MSB} = 0$.

Przypadek 2.

$$X<0 \quad \text{i} \quad Y<0$$

$$X \leftrightarrow \mathsf{X} = (1, \tilde{x}_{n-2}, \ldots, \tilde{x}_1, \tilde{x}_0)$$
$$Y \leftrightarrow \mathsf{Y} = (1, \tilde{y}_{n-2}, \ldots, \tilde{y}_1, \tilde{y}_0)$$

Niech

$$X^* = 2^n - |X| \text{ oraz } Y^* = 2^n - |Y|.$$

wtedy sumę uzupełnień można przedstawić w sposób następujący:

$$X^* + Y^* = 2^n - |X| + 2^n - |Y| = 2 \cdot 2^n - (|X| + |Y|) =$$
$$2^n + (2^n - (|X| + |Y|)).$$

Uzyskujemy tym samym uzupełnienie do 2 liczby $|X|+|Y|$.

A więc w rozważanym przypadku

a) występuje zawsze przeniesienie z pozycji znaku, czyli $c_S = 1$, wynika to z występowania 2^n.

b) jeżeli $|X| + |Y| < 2^{n-1}$, to rezultat będzie poprawny i przedstawiony w kodzie U2 oraz pojawi się przeniesienie z bitu najbardziej znaczącego czyli $c_{MSB} = 1$. Wynika to z faktu, że suma dwóch liczb ujemnych przedstawionych w kodzie U2 zawsze przekroczy $2^{n-1} - 1$. Ponieważ $X^* + X = 2^n$, czyli jeśli $X \le 2^{n-1} - 1$ to $X^* > 2^{n-1}$.

Przypadek 3.

W przypadku tym mogą wystąpić dwie możliwe sytuacje:

$$X>0 \text{ i } Y<0.$$

Reprezentacje mają tutaj postać

$$X \leftrightarrow \mathsf{X} = (0, \tilde{x}_{n-2}, \ldots, \tilde{x}_1, \tilde{x}_0),$$
$$Y \leftrightarrow \mathsf{Y} = (1, \tilde{y}_{n-2}, \ldots, \tilde{y}_1, \tilde{y}_0).$$

lub dla

$$X>0 \text{ i } Y<0$$
$$X \leftrightarrow \mathsf{X} = (0, \tilde{x}_{n-2}, \ldots, \tilde{x}_1, \tilde{x}_0)$$
$$Y \leftrightarrow \mathsf{Y} = (1, \tilde{y}_{n-2}, \ldots, \tilde{y}_1, \tilde{y}_0)$$

W przypadku tym nie wystąpi nadmiar.

Niech $Y^* = 2^n - |Y|$

$$S = X + Y^* = X + 2^n - |Y| = 2^n - (|Y| - |X|)$$

Jeżeli

a) $|Y| - |X| > 0$ i równe c,

mamy więc

$$|S|_{2^n} = \left|2^n - (|Y| - |X|)\right|_{2^n} = \left|2^n - c\right|_{2^n} = 2^n - c.$$

czyli rezultat będzie poprawny oraz nie wystąpi przeniesienie z najstarszego bitu (z bitu MSB) do pozycji znaku. Przeniesienie z pozycji znaku nie pojawia się, gdyż tylko znak Y jest równy 1, a więc $c_{MSB} = 0$ i $c_S = 0$ .Otrzymana wartość reprezentuje -c w kodzie U2.

Wykażemy teraz, że $c_{MSB} = 0$. Mamy

$$S = X + Y^* = X + 2^n - |Y| = 2^{n-1} + 2^{n-1} - (|Y| - |X|) = 2^{n-1} + 2^{n-1} - c.$$

Liczba 2^{n-1} oznacza 1 na pozycji znaku, liczba $2^{n-1} - c$ jest mniejsza lub równa niż $2^{n-1} - 1$, nie będzie przeniesienia do pozycji znaku.

b) $|Y| - |X| < 0$ i równe -c ,

wtedy

$$S = 2^n - (-(|Y| - |X|)) = 2^n - (-c) = 2^n + c,$$

a więc występuje w tym przypadku przeniesienie z pozycji znaku, czyli $c_S = 1$, a wynik działania jest równy *c*, gdyż

$$|S|_{2^n} = \left|2^n - (|Y| - |X|)\right|_{2^n} = \left|2^n + c\right|_{2^n} = c.$$

Wystąpienie przeniesienia z najwyższej pozycji liczby (bitu MSB) można wykazać następująco:

$$S = X + Y^* = X + 2^n - |Y| = 2^{n-1} + 2^{n-1} - 1 + |X| - |Y| + 1$$

Liczba 2^{n-1} oznacza 1 na pozycji znaku, liczba $2^{n-1} - 1$ ma same jedynki na *n*-1 pozycjach, więc skoro $|X| - |Y|$ tutaj jest liczba dodatnią, to musi wystąpić przeniesienie do pozycji znaku, gdyż suma przekroczy $2^{n-1} - 1$. W rezultacie otrzymujemy $c_{MSB} = 0$ oraz $c_S = 1$.

Wykazano więc, że jeżeli rezultat dodawania U2 jest poprawny to obydwa przeniesienia tzn. przeniesienie do pozycji znaku i przeniesienie z pozycji znaku muszą być równe.

9.2 Odejmowanie w systemie U2

Niech liczby *X* i *Y* mają następujące reprezentacje w U2

$$X \leftrightarrow X = (x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$$

$$Y \leftrightarrow Y = (y_{n-1}, y_{n-2}, \ldots, y_1, y_0)$$

Odejmowanie *X*-*Y* odbywa się w trzech krokach :

Krok 1. Utwórz uzupełnienie do dwóch odjemnika.

Krok 2. Dodaj kolejne cyfry odjemnej i uzupełnienia odjemnika do 2 poczynając od najmniej znaczącej pary aż do pary najbardziej znaczącej (cyfr znaku), zakładając zerowe przeniesienie z prawej strony.

Krok 3. Porównaj przeniesienie wchodzące do pozycji znaku z przeniesieniem wychodzącym z pozycji znaku. Jeżeli obydwa są równe 0 lub obydwa 1, to przeniesienie z pozycji znaku należy zignorować i rezultat jest w poprawnej formie U2. Jeżeli przeniesienia różnią się, oznacza to, że wystąpił nadmiar i rezultat jest niepoprawny.

Przykład 9.3 Odejmowanie w kodzie U2.

a) odejmowanie liczby dodatniej, sprowadza się ono do dodawania liczby ujemnej w U2, czyli utworzenia uzupełnienia odjemnika

-odjemna +13 0.01101
-odjemnik +11 0.01011
-uzupełnienie odjemnika do 2 1.10101

+13	0.01101	
- (+11)	+ 1.10101	(-11 w U2)
+2	0.00010	

$c_{MSB} = 1,\ c_S = 1$

b) odejmowanie liczby ujemnej
-odjemna +13 0.01101
-odjemnik -11 1.10101
-uzupełnienie odjemnika do 2 0.01011

(-13)	1.10011	(-13 w U2)
-(-11)	+0.01011	
-2	1.11110	

$c_{MSB} = 1,\ c_S = 1,$