X. REPREZENTACJA DANYCH W SYSTEMACH CYFROWYCH

X.1. POZYCYJNE SYSTEMY LICZBOWE

Zapis pozycyjny liczb oznacza, że w ciągu cyfr reprezentującym liczbę znak każdej cyfry zajmuje ściśle określoną pozycję, przy czym tej pozycji przyporządkowana jest odpowiednia waga będąca potęgą podstawy. Ilość znaków stosowanych w każdym pozycyjnym systemie liczbowym jest taka sama jak jego podstawa.

W słowie A o długości n (czyli ilości cyfr równej n) reprezentującym liczbę $L(A)=a_{n-1}...a_1a_0$ każdy znak a_i zajmuje określoną pozycję i tej pozycji przyporządkowana jest waga p^i. Stała p jest podstawą przyjętego zapisu liczbowego. Można wyrazić to w postaci matematycznej:

$$L(A) = \sum_{i=0}^{n-1} a_i \, p^i \qquad \text{(X.1)}$$

Od wielu wieków zapis informacji liczbowych przez człowieka realizowany jest w **systemie dziesiętnym**[1]. Zapis ten poparty jest tradycją, popularnością i zrozumiałością i jak nie trudno się domyślić podstawa kodu równa się liczbie palców obu rąk człowieka[2]. Podstawą systemu jest liczba dziesięć (p=10) dlatego używane jest dziesięć symboli cyfr, a mianowicie 0, 1, 2, 3, 4, 5, 6, 7, 8 i 9. Każdą liczbę w kodzie dziesiętnym można przedstawić jako sumę składników:

$$L(a_{n-1}...a_1a_0) = \sum_{i=0}^{n-1} a_i \, 10^i = a_{n-1}\times10^{n-1} + ...+ a_1\times10^1 + a_0\times10^0 \qquad \text{(X.2)}$$

na przykład: $L(2578) = \sum_{i=0}^{3} a_i \, 10^i = 2\times10^3 + 5\times10^2 + 7\times10^1 + 8\times10^0 = 2578_{10}$

[1] Terminy *system liczbowy* i dalej stosowany *kod liczbowy* są tu utożsamiane. Takie podejście spotykane jest w literaturze przedmiotu ponieważ kod jest umownym systemem znaków umożliwiającym przekazywanie informacji.

[2] Zapis dziesiętny jest w powszechnym użyciu dopiero od dziesięciu wieków. Został on opracowany w Indiach około piątego wieku n.e. i udostępniony cywilizacji europejskiej przez Arabów.
W starożytnym Rzymie też był stosowany kod dziesiętny, lecz nie był to kod pozycyjny. Stosowane były wtedy (i do dziś są stosowane w wybranych sytuacjach) następujące symbole $I=1_{10}$, $V=5_{10}$, $X=10_{10}$, $L=50_{10}$, $C=100_{10}$, $D=500_{10}$, $M=1000_{10}$. Tylko dwa spośród tych symboli można skojarzyć z pierwszymi literami łacińskich nazw liczebników: C-*centum* i M-*mille*. Pozostałe oznaczenia wynikają z przyjętej konwencji. W kodzie tym brak było symbolu zera, a odczyt poprzedzony być musiał analizą względnego rozmieszczenia poszczególnych symboli reprezentujących określone cyfry (do ciekawostek zaliczyć należy modyfikację zapisu rzymskiego cyfr dokonanego przez króla Francji Ludwika XIV; otóż kazał on się pisać Ludwik XIIII i żądał by zegary w Wersalu o godzinie czwartej wskazywały na cyfrę IIII, co zostało do dzisiaj na tarczach zegarowych wykorzystujących cyfry rzymskie [3]).
Koncepcja zera była jednak znana w starożytnej Mezopotamii, gdzie był stosowany pozycyjny kod liczbowy o 60-ciu symbolach. Kod ten został zarzucony ok. 1700 r p.n.e., ale wpływ tego rozwiązania trwa do dzisiaj albowiem godzina ma 60 minut, minuta ma 60 sekund, koło ma 6x60=360 stopni [4].

gdzie rodzaj zapisu zaznaczany jest przez dolny indeks przy liczbie[3].

System dwójkowy (binarny) posiada tylko dwie cyfry 0 i 1 i podstawą zapisu jest liczba 2 (p=2). Już od dawna budził on zainteresowanie filozofów i matematyków[4]. W tym systemie najpopularniejszym zapisem jest **naturalny kod dwójkowy**.

W naturalnym kodzie dwójkowym w słowie B o długości n reprezentującym liczbę $L(B)=b_{n-1}...b_1b_0$ każdy znak b_i zajmuje określoną pozycję i tej pozycji przyporządkowana jest odpowiednia waga p^i, gdzie podstawą przyjętego tego zapisu jest liczba 2 (p=2).

$$L(b_{n-1}...b_1b_0) = \sum_{i=0}^{n-1} b_i\, 2^i = b_{n-1}\times 2^{n-1} + ... + b_1\times 2^1 + b_0\times 2^0 \qquad (X.3)$$

na przykład: $L(1101) = \sum_{i=0}^{3} b_i\, 2^i = 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0 = 13_{10}$

W słowie dwójkowym bit związany z największą wagą (skrajny z lewej strony w ww. zapisie) jest bitem najbardziej znaczącym (najstarszym) zwanym bitem MSB (*Most Significant Bit*). Bit związany z najmniejszą wagą (skrajny z prawej strony) jest bitem najmniej znaczącym (najmłodszym) LSB (*Least Significant Bit*).

Zamiana odwrotna tj. liczby dziesiętnej na dwójkową następuje poprzez kolejne dzielenie liczby dziesiętnej przez 2 i stwierdzenie czy jest reszta (1) czy jej nie ma (0).

Przykład

Rozważmy zamianę liczby dziesiętnej 13_{10} na postać w naturalnym kodzie dwójkowym. W tym celu należy dokonać następujących dzieleń i obserwacji czy w wyniku pozostaje reszta:

13/2=6 jest reszta	1	LSB
6/2 = 3 nie ma reszty	0	
3/2 = 1 jest reszta	1	
1/2 = 0 jest reszta	1	MSB

co daje $13_{10} = 1101_2$

[3] Indeksy dolne oznaczają: 10 – kod dziesiętny, 2 – naturalny kod dwójkowy, 16 – kod heksadecymalny, BCD – kod dwójkowo-dziesiętny, 1z10 – kod jeden z dziesięciu, ZM – kod znak-moduł, U1 – kod uzupełnienia do jednego, U2 – kod uzupełnienia do dwóch.

[4] Gottfreid Leibnitz (1646-1716) - niemiecki filozof, matematyk i fizyk. Jako pierwszy zaproponował rozszerzenie założeń obowiązujących w systemie dziesiętnym na systemy liczbowe o innych podstawach. Był on człowiekiem bardzo religijnym i uduchowionym dlatego przypisywał systemowi dwójkowemu niezwykłe znaczenie. Fakt, że każdą liczbę można przedstawić za pomocą zer i jedynek wiązał z tym, że Bóg (1) stworzył świat z niczego (0). Warto pamiętać, że Leibnitz jest twórcą rachunku różniczkowego i całkowego, choć do końca życia spierał się o pierwszeństwo w tej dziedzinie z Izaakiem Newton'em. Jednak Leibnitz nie szedł drogą Newton'a streszczającą się w słowach: zrobić co trzeba, a potem unikać dyskusji na temat zastosowanych metod - i podał uzasadnienie metod obliczania pochodnej i całkowania oraz wprowadził formalizm zapisu stosowany do dzisiaj (symbole pochodnej df/dx i $\partial f/\partial x$ i symbol całki pochodzący z manierycznie napisanego wyrazu ſumma). Pochodził prawdopodobnie z rodziny Lubienieckich - polskich Arian, którzy wyemigrowali do Niemiec [1].

Zauważmy, że wynik otrzymujemy w kolejności od bitu najmniej znaczącego (LSB) do najbardziej znaczącego (MSB).

Koniec przykładu.

Dla liczb ułamkowych wyrażanych w naturalnym kodzie dwójkowym stosuje się przy poszczególnych cyfrach z prawej strony symbolu części ułamkowej (kropki) wagi o wykładnikach ujemnych zależnych od umiejscowienia cyfry w części ułamkowej. Na przykład:

$$1011.0101_2=1\times2^3+0\times2^2+1\times2^1+1\times2^0+0\times2^{-1}+1\times2^{-2}+0\times2^{-3}+1\times2^{-4}=11.3125_{10}$$

Zamiana części ułamkowej liczby dziesiętnej na dwójkową polega na powtarzaniu mnożenia przez dwa i jeśli wynik jest większy lub równy jedności to bit o odpowiedniej wadze ma wartość 1. Wtedy to dalszemu mnożeniu podlega część pomniejszona o 1. Gdy natomiast w wyniku mnożenia przez dwa otrzymuje się liczbę mniejszą od jedności to bit o odpowiadającej jej wadze ma wartość 0 i ponownie wykonuje się mnożenie otrzymanego wyniku. Działanie prowadzi się do wyczerpania możliwości mnożenia lub gdy dojdzie się do dopuszczalnej ilości bitów określających część ułamkową.

Przykład

Zamienić liczbę ułamkową dziesiętną 0.68_{10} na postać ułamkową w naturalnym kodzie dwójkowym. W tym celu dokonujemy następujących działań:

liczba ułamkowa dziesiętnie	liczba pomnożona przez 2	czy wynik mnożenia większy lub równy 1	bit części ułamkowej w naturalnym kodzie dwójkowym
0.68	1.36	tak	1
0.36	0.72	nie	0
0.72	1.44	tak	1
0.44	0.88	nie	0
0.88	1.76	tak	1
0.76	1.52	tak	1
itd. w zależności od tego ile bitów mamy do dyspozycji w zapisie binarnym części ułamkowej			

W wyniku otrzymujemy $0.68_{10}=0.101011_2$. Wynik przekształcenia jest przybliżony co łatwo sprawdzić, gdyż $0.101011_2=0.671875_{10}$. Dysponując bowiem sześcioma bitami nie można uzyskać dla tej liczby większej dokładności.

Koniec przykładu.

Zapis liczb w naturalnym kodzie dwójkowym powoduje konieczność operowania stosunkowo długimi ciągami znaków, co sprzyja popełnianiu błędów. Zapis ten jest niezbyt

poręczny z punktu widzenia programistów wykorzystujących język maszynowy komputerów[5]. Dla uproszczenia notacji i jej skrócenia wprowadzono kod **heksadecymalny** inaczej zwany **szesnastkowym**. Cyfry binarne są grupowane w tym kodzie w zespoły po 4 (grupowanie w tetrady). Każda możliwa kombinacja czterech cyfr binarnych otrzymuje symbol jak w tabeli X.1. Dla oznaczenia pierwszych dziesięciu grup liczb binarnych wykorzystano cyfry systemu dziesiętnego, a pozostałym przypisano symbole literowe odpowiednio A, B, C, D, E i F.

Tabela X.1. Kod heksadecymalny

$0000_2=0_{16}$	$0100_2=4_{16}$	$0100_2=8_{16}$	$1100_2=C_{16}$
$0001_2=1_{16}$	$0101_2=5_{16}$	$0101_2=9_{16}$	$1101_2=D_{16}$
$0010_2=2_{16}$	$0110_2=6_{16}$	$1010_2=A_{16}$	$1110_2=E_{16}$
$0011_2=3_{16}$	$0111_2=7_{16}$	$1011_2=B_{16}$	$1111_2=F_{16}$

W kodzie heksadecymalnym w słowie H o długości n reprezentującym liczbę $L(H)=h_{n-1}...h_1h_0$ każdy znak h_i zajmuje określoną pozycję i tej pozycji przyporządkowana jest odpowiednia waga p^i, gdzie podstawą przyjętego zapisu liczbowego jest 16 (p=16).

$$L(h_{n-1}...h_1h_0) = \sum_{i=0}^{n-1} h_i\,16^i = h_{n-1}\times16^{n-1} +...+ h_1\times16^1 + h_0\times16^0 \qquad (X.4)$$

na przykład:

$$L(2C3) = \sum_{i=0}^{2} h_i\,16^i = 2_{16}\times16^2 + C_{16}\times16^1 + 3_{16}\times16^0 = 2_{10}\times16^2 + 12_{10}\times16^1 + 3_{10}\times16^0 = 707_{10}$$

Notacja heksadecymalna jest używana szczególnie do zapisu dowolnego ciągu cyfr binarnych (ciągów binarnych) niezależnie od tego czy reprezentują one liczby, tekst czy inny rodzaj danych. Wynika to z tego, że jest ona bardziej zwarta niż notacja binarna. Ponadto w większości komputerów dane binarne reprezentowane są jako wielokrotność czterech bitów (tetrady) co oznacza, iż w zapisie można je zastąpić pojedynczymi znakami kodu heksadecymalnego. Zwiększa to zwięzłość zapisu i zmniejsza prawdopodobieństwo popełnienia błędów przez programistę. Istotne też jest, że konwersja między notacją binarną a heksadecymalną jest bardzo łatwa. Na przykład ciąg binarny 110111100001_2 jest równoważny zapisowi $DE1_{16}$, a doświadczony programista może w myśli przetwarzać zapis binarny na heksadecymalny i heksadecymalny na binarny w postaci ciągu tetrad oznaczanych pojedynczymi symbolami kodu szesnastkowego.

[5] Symbolicznym zapisem instrukcji i danych, które odpowiadają zapisowi binarnemu języka maszynowego komputera jest język niskiego poziomu zwany asemblerem.

X.2. KODY DWÓJKOWO DZIESIĘTNE

W kodach **dwójkowo dziesiętnych BCD** (*Binary Coded Decimal*) każda cyfra liczby dziesiętnej jest osobno kodowana dwójkowo w postaci odpowiedniego słowa.

W kodzie **dwójkowo dziesiętnym BCD 8421** każda cyfra liczby dziesiętnej jest oddzielnie kodowana dwójkowo w postaci czterobitowego słowa w naturalnym kodzie dwójkowym.

	1	3	7	
$137_{10}=$	0001_2	0011_2	0111_2	$= 000100110111_{BCD}$

naturalny kod dwójkowy
dla poszczególnych cyfr dziesiętnych

Cała liczba dziesiętna jest przedstawiana jako złożenie słów dwójkowych reprezentujących wszystkie kolejne cyfry tej liczby. Dla zapisu dziesięciu cyfr kodu dziesiętnego potrzebne jest czterobitowe słowo w kodzie dwójkowym. Z kolei słowo takie umożliwia zapis szesnastu liczb, czyli sześć z nich jest po prostu zbędne. Oznacza to, że kod BCD jest kodem nadmiarowym.

Kod BCD stosowany jest w bardzo wydajnych jednostkach komputerowych typu *mainframe*. Komputery tego typu posiadają duże zasoby obliczeniowe z zapis ten mimo nadmiarowości umożliwia osiągnięcie wysokiej precyzji obliczeń.

Specjalnym rodzajem kodu dwójkowo dziesiętnego jest **kod 1 z 10** zwany też kodem pierścieniowym (tab.X.2). Kod ten posiada słowo dziesięciobitowe w którym zawsze dziewięć bitów zajmują zera, a jeden bit przeznaczony jest dla jedynki. Położenie jedynki decyduje o tym, jakiej liczbie dziesiętnej odpowiada zapis binarny w tym kodzie.

Tab.X.2. Kod 1z10

$0_{10}=0000000001_{1z10}$	$1_{10}=0000000010_{1z10}$	$2_{10}=0000000100_{1z10}$	$3_{10}=0000001000_{1z10}$	$4_{10}=0000010000_{1z10}$
$5_{10}=0000100000_{1z10}$	$6_{10}=0001000000_{1z10}$	$7_{10}=0010000000_{1z10}$	$8_{10}=0100000000_{1z10}$	$9_{10}=1000000000_{1z10}$

Kod ten jest najprostszym koncepcyjnie kodem do wprowadzania do układu cyfrowego cyfr kodu dziesiętnego z klawiatury urządzenia cyfrowego, której poszczególne klawisze reprezentują liczby od zera do dziewięciu (nie dotyczy to klawiatury komputera, o czym będzie mowa dalej).

X.3. ZAPIS LICZB DWÓJKOWYCH ZE ZNAKIEM

Zapis **znak-moduł** (*signed magnitude*) charakteryzuje się tym, że wartość bezwzględna liczby i jej znak reprezentowane są niezależnie. Najbardziej znaczący bit odzwierciedla znak liczby: 0 - gdy liczba jest dodatnia, 1- gdy liczba jest ujemna. Pozostałe bity charakteryzują wartość bezwzględną (zapisaną w naturalnym kodzie dwójkowym). Zmiana znaku sprowadza się zatem do zanegowania najstarszego bitu jej reprezentacji dwójkowej.

Ogólnie można napisać, że słowo $A = a_{n-1}\ a_{n-2}\ a_{n-3\ldots}a_0$ zawierające n bitów w zapisie znak-moduł ma dziesiętną wartość następującą:

$$L(A) = \begin{cases} \sum_{i=0}^{n-2} 2^i a_i & \text{jesli } a_{n-1} = 0 \\ -\sum_{i=0}^{n-2} 2^i a_i & \text{jesli } a_{n-1} = 1 \end{cases} \qquad \text{(X.5)}$$

na przykład: $+18_{10} = 00010010_{ZM}$ natomiast $-18_{10} = 10010010_{ZM}$

Wadami reprezentacji znak-moduł jest to, że dodawanie i odejmowania wymaga rozważenia znaków liczb jak i ich modułów. Inna wadą jest występowanie dwóch reprezentacji liczby zero. Występuje mianowicie tzw. „plus zero" $+0_{10} = 00000000_{zm}$ i „minus zero" $-0_{10} = 10000000_{ZM}$. Wymaga to bardziej złożonej operacji sprawdzania zera, która to operacja jest często wykonywana przez komputery.

Kod **uzupełnienia**[6] **do 1** (*one's complemented*) oznaczany symbolicznie **U1**. Znak liczby reprezentowany jest przez najstarszy bit w taki sam sposób jak w zapisie znak moduł. W pozostałych bitach reprezentowana jest wartość bezwzględna liczby z tym, że gdy liczba jest dodatnia to zapisana jest w kodzie znak-moduł, a gdy ujemna to wszystkie bity są zanegowane.

Liczba binarna zapisana w naturalnym kodzie dwójkowym na swój odpowiednik w kodzie uzupełnienia do jednego w postaci takiej, że zanegowane są wszystkie bity tej liczby (patrz przypis dolny). Przedstawiane w tej formie są jedynie liczby ujemne kodu U1. Wynika z tego, że notacja uzupełnieniowa pozwala uprościć odejmowanie przez zastąpienie go dodawaniem.

Przykład

[6] Każdą liczbę zapisaną w kodzie pozycyjnym można przedstawić w kodzie uzupełnieniowym. Istnieją dwa rodzaje takich uzupełnień: uzupełnienie do podstawy i uzupełnienie do zmniejszanej podstawy. W przypadku systemu dziesiętnego będzie to uzupełnienie do dziesięciu (na przykład liczba 347 ma uzupełnienie do dziesięciu 1000 – 347 = 653) i uzupełnienie do dziewięciu (liczba 347 ma uzupełnienie do dziewięciu 999 – 347 = 652). W przypadku systemu dwójkowego będzie to uzupełnienie do dwóch (na przykład liczba 0101_2 ma uzupełnienie do dwóch $10000_2 - 0101_2 = 1011_2$) i uzupełnienie do jednego (liczba 0101_2 ma uzupełnienie do jednego $1111_2 - 0101_2 = 1010_2$).

Dodaj liczby 23_{10} i -9_{10} w kodzie uzupełnienia do 1.

Zapisujemy liczby 23_{10} i 9_{10} (a nie -9_{10}) w kodzie znak-moduł otrzymując wartość bezwzględną poprzez wcześniej omówione dzielenie liczby dziesiętnej przez 2 i stwierdzenie czy jest reszta, czy jej nie ma. W wyniku otrzymujemy: $23_{10}=00010111_{ZM}$ i $9_{10}=00001001_{ZM}$. Zapisujemy je następnie w kodzie U1 wtedy liczba dodatnia pozostaje bez zmian, a ujemna jest zanegowaniem wszystkich bitów jej formy dodatniej i tak: $23_{10}=00010111_{U1}$ i $-9_{10}=11110110_{U1}$.

Wykonujemy dodawanie:

1	←	1	1	1		1	1			←przeniesienia	
		0	0	0	1	0	1	1	1		23_{10}
	+	1	1	1	1	0	1	1	0		-9_{10}
		0	0	0	0	1	1	0	1	suma pośrednia	
	+								1		
		0	0	0	0	1	1	1	0	suma końcowa	14_{10}

Uwaga: ostatnie przeniesienie jeśli występuje jest dodawane do sumy pośredniej. Jeśli nie ma tego przeniesienia to wynik pierwszego sumowania jest ostateczny.

Koniec przykładu.

Wadą kodu U1 jest to, że w dalszym ciągu zero można przedstawić na dwa sposoby tj. 0000_{U1} i 1111_{U1} oraz to, że gdy wystąpi przeniesienie na ostatniej pozycji dodawania to należy je dodać (dodać jedynkę) do otrzymanej sumy pośredniej. Między innymi z tego powodu kod ten ma dzisiaj znaczenie wyłącznie historyczne i został zarzucony na rzecz znacznie efektywniejszej notacji uzupełnienie do dwóch omówionej poniżej.

Kod **uzupełnienia do 2** (*two's complemented*) oznaczany symbolicznie **U2**. Znak liczby reprezentowany jest przez najstarszy bit w taki sam sposób jak w zapisie znak-moduł i w kodzie U1. W pozostałych bitach reprezentowana jest wartość bezwzględna liczby z tym, że gdy liczba jest dodatnia to zapisana jest w kodzie znak moduł, a gdy jest ujemna to wszystkie bity są zanegowane (jak w kodzie U1) i do wyniku na ostatniej pozycji dodana jest arytmetyczne jedynka. W procesie dodawania uwzględnia się przeniesienia.

Przykład

Dodaj liczby 23_{10} i -9_{10} w kodzie uzupełnienie do 2.

Zapisujemy liczby 23_{10} i 9_{10} (a nie -9_{10}) w kodzie znak-moduł otrzymując wartość bezwzględną tego zapisu poprzez, wcześniej omówione, dzielenie liczby dziesiętnej przez 2 i stwierdzenie czy jest reszta, czy jej nie ma. W wyniku otrzymujemy: $23_{10}=00010111_{ZM}$ i $9_{10}=00001001_{ZM}$. Zapisujemy je następnie w kodzie U2, gdzie liczba dodatnia pozostaje bez

zmian, a ujemna jest zapisywana w kodzie U1 (zanegowanie wszystkich bitów jej formy dodatniej) $9_{10}=11110110_{U1}$ i dodawana jest jedynka do ostatniej cyfry. Wykonujemy dodawanie:

	1	1	1	1	1	0	1	1	0
+									1
	1	1	1	1	1	0	1	1	1

W wyniku dodania jedynki na ostatnim miejscu liczby ujemnej zapisanej w kodzie U1 otrzymujemy liczbę ujemną wyrażoną w kodzie U2 zatem: $-9_{10}=11110111_{U2}$. Gdy w dodawaniu jedynki nastąpi przeniesienie to trzeba je uwzględniać.

Mając zapis obu liczb w kodzie U2 wykonujemy dodawanie:

1	←	1	1	1		1	1	1		←przeniesienia	
		0	0	0	1	0	1	1	1		23_{10}
	+	1	1	1	1	0	1	1	1		-9_{10}
		0	0	0	0	1	1	1	0		14_{10}

Uwaga: w tym przypadku ostatnie przeniesienie trzeba odrzucić.

Koniec przykładu.

Z powyższego przykładu widać, że w pewnych przypadkach odrzucenie przeniesienia pochodzącego z najbardziej na lewo umiejscowionych bitów tj. bitów znaku nie zmienia wyniku dodawania. Jednak nie jest tak zawsze. Istnieje reguła wykrywania sytuacji nieprawidłowych zwanych błędami przepełnienia. *Błędy przepełnienia* występują wówczas, gdy przy dodawaniu w kodzie U2 na skutek przeniesienia na pozycjach znaku liczby wynik może być nieprawidłowy.

Reguła wykrywania błędów przepełnienia brzmi: Jeżeli wartość przeniesienia wchodzącego do bitu znaku jest taka sama jak wartość przeniesienia z niego wychodzącego, to nie nastąpiło przepełnienie. Jeżeli zaś przeniesienia te mają różne wartości, mamy odczynienia z przepełnieniem, a więc błędem.

Przykład

Znajdź sumę liczb 126_{10} i 8_{10} w arytmetyce kodu U2.

Wykonujemy dodawanie:

0	←	1	1	1	1					←przeniesienia	
		0	1	1	1	1	1	1	0		126_{10}
	+	0	0	0	0	1	0	0	0	+	8_{10}
		1	0	0	0	0	1	1	0		-122_{10}
		bit znaku									***(BŁĄD!!!)***

Jak widać przeniesienie wchodzące do bitu znaku (jedynka) i zeń wychodzące (zero) nie są takie same. Nastąpiło zatem przepełnienie i otrzymany wynik jest błędny. Dodanie tych liczb zapisanych w kodzie U2 i przy zastosowaniu jednobajtowego słowa jest niemożliwe.

Koniec przykładu.

Błąd przepełnienia może być z łatwością wykrywany przez proste obwody kombinacyjne realizujące wyżej przytoczoną regułę wykrywania tego błędu.

Kod U2 jest najpopularniejszym sposobem zapisu liczb ze znakiem. Liczby wyrażone za pomocą kodu U2 łatwo się dodaje i odejmuje. Zero w tym kodzie ma jedną reprezentacje i to najlepszą z możliwych – wszystkie bity są zerowe. W tabeli X.3 przedstawiono porównawczo wybrane liczby zapisane w omawianych kodach znak-moduł, uzupełnienia do jednego i uzupełnienia do dwóch.

Tabela X.3. Kody liczbowe zapisu liczb dwójkowych ze znakiem

dziesiętnie	znak-moduł	U1	U2
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
0	0000 , 1000	0000 , 1111	0000
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1010	1011	1100
-5	1011	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001

X.4. ZMIENNOPRZECINKOWA REPREZENTACJA LICZB

Dotychczas przedstawione reprezentacje liczb binarnych były tzw. stałoprzecinkowe (*fixed-point notation*). Oznacza to że ustawienie kropki oddzielającej część całkowitą od części ułamkowej liczby było stałe i nie podlegało przesunięciu np. bbbb.bbbb, gdzie b={0,1}. Tego typu zapisy mają ograniczenia wynikające z możliwości zapisu liczb w dużym zakresie zmienności, stąd zaczęto stosować **zapis zmiennoprzecinkowy** (*floating point notation*) zwany

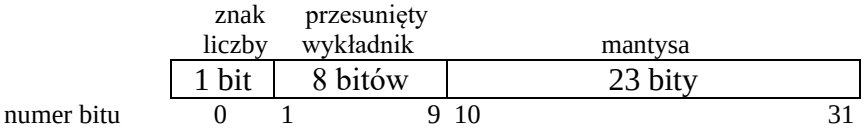
też zmiennopozycyjnym. Liczbę L w zapisie zmiennoprzecinkowym zapisuje się w postaci wykładniczej jak niżej:

$$L = \pm M\, p^{\pm E} \qquad \text{(X.6)}$$

gdzie: ± znak liczby, M - mantysa, p - podstawa, E - wykładnik.

Zapis składa się z trzech części: bitu znaku liczby, mantysy nazywanej też częścią ułamkową liczby i wykładnika potęgi będącym liczbą całkowitą. Podstawa jest ustalana jednorazowo i jej wartość jest trwale przechowywana. Poszczególne urządzenia cyfrowe, które używają zapisu zmiennoprzecinkowego przyjmują z góry pewien standard określając stosowaną podstawę np. p=2 czy p=10 lub p=16 i nie trzeba jej każdorazowo wykazywać przy zapisie liczb tą metodą. Zakres liczby zapisanej zmiennoprzecinkowo zależy do ilości cyfr jej wykładnika, a dokładność reprezentacji zależy od ilości cyfr jej mantysy.

Dla przybliżenia szczegółów tego zapisu przyjmijmy przykładowy zapis 32 bitowy formatu zmiennoprzecinkowego w standardzie IEEE-754[7] pojedynczej precyzji (rys.X.1).



Rys.X.1. Format liczby w zapisie zmiennoprzecinkowym w standardzie IEEE-754 pojedynczej precyzji.

Zerowy bit reprezentuje znak liczby. Ma on wartość 0 gdy liczna jest dodatnia a wartość 1gdy liczba jest ujemna. Wartość wykładnika jest przechowywana w bitach od 1 do 9. Aby nie tracić jednego bitu wykładnika na pokazanie czy wykładnik jest dodatni czy ujemny stosuje się tzw. *przesunięcie*, zatem reprezentacja wykładnika jest reprezentacją przesuniętą. Za pomocą pola ośmiobitowego wykładnika można reprezentować liczby od 0_{10} do 255_{10}. Po zastosowaniu przesunięcia równego 128_{10} prawdziwe wartości wykładnika znajdują się w zakresie od -128_{10} do $+127_{10}$. Każda liczba mająca przed przesunięciem wykładnik większy od 128_{10} będzie traktowana jako dodatnia a pozostałe będą liczbami ujemnymi. Przy odczycie wartości wykładnika liczba zwana przesunięciem jest odejmowana od zapisu będącego w polu wykładnika w celu otrzymania jego prawdziwej wartości.

[7] W latach 80-tych ubiegłego wieku każdy producent komputerów stosował własną odmianę reprezentacji zmiennoprzecinkowej liczb. Aby tę niekorzystną tendencję zniwelować grupa robocza amerykańskiej organizacji IEEE (*Institute of Electrical and Electronic Engineers*) wprowadziła standaryzację zapisu oznaczając ją symbolem IEEE-754.

Wartość mantysy jest zawsze z przedziału [0.5_{10} , 1.0_{10}) co dwójkowo można zapisać: 0.1000...0≤M<0.111...1. Można zauważyć, że dwie pierwsze pozycje mantysy są zawsze takie same. Wynika z tego, że nie ma konieczności przedstawiania wszystkich bitów liczby 0.1 b_{-2} b_{-3} b_{-4}… ponieważ zapis b_{-2} b_{-3} b_{-4}… daje tę samą informacje (oczywiście przy zachowaniu zasady, że pominięta - ale zawsze występująca część zapisu - jest w odpowiednim momencie odtwarzana). Stosowanie takiego sposobu zapisu mantysy nazywa się jej *normalizacją* (notacją znormalizowaną).

Przykład

Zapiszmy liczbę $+625.625_{10}$ w zapisie zmiennoprzecinkowym w standardzie IEEE-754 pojedynczej precyzji.

Liczbę tę można przedstawić w formacie binarnym jako 1001110001.101_2. Normalizując przesuwamy o 10 pozycji w prawo i otrzymujemy $0.1001110001101\times2^{10}$ zatem zapis znormalizowany mantysy (po pominięciu pierwszych bitów 0.1) jest następujący 001110001101 co po uzupełnieniu do znormalizowanej postaci 23 bitowej mamy: 00111000110100000000000.

Wykładnik ma wartość 10_{10} a aby zastosować reprezentację przesuniętą jego zapisu z dodajemy przesunięcie 128_{10} zatem jego wartości jest 138_{10} co binarnie wyraża się 10001010_2.

W wyniku otrzymujemy: 01000101000111000110100000000000 co można przedstawić szczegółowo jak poniżej:

	znak liczby	przesunięty wykładnik		mantysa znormalizowana	
	0	10001010		00111000110100000000000	
numer bitu	0	1	9	10	31

Koniec przykładu.

W standardzie IEEE-754 określono jeszcze zapis zmiennoprzecinkowy o podwójnej precyzji. Słowo w tym rodzaju zapisu jest 64 bitowe i składa się z jednobitowego pola znaku, 11 bitowego pola wykładnika i 52 bitowego pola mantysy. W tym standardzie można zapisać liczby z znacznie większego przedziału wartości.

II.5. KODY ZNAKOWE

Kody znakowe umożliwiają zapis poszczególnych znaków (liter, cyfr, symboli graficznych, znaków sterujących i in.) jako ciągu bitów. Kody te umożliwiają przechowywanie, przetwarzanie i transmisję znaków w urządzeniach cyfrowych.

Przedstawicielem tej grupy kodów jest **kod ASCII** (*czytaj: aski*), którego pełna nazwa brzmi: *American Standard Code for Information Interchange*. Pochodzi on od sposobu kodowania stosowanego od lat sześćdziesiątych ubiegłego wieku w teleksach i dalekopisach. Każdy znak w tym kodzie jest reprezentowany przez unikatowy ciąg siedmiu bitów, zatem w tym kodzie może być reprezentowanych 128 różnych znaków. Około jedna czwarta z nich (znaki o liczbach 0_{10} do 31_{10} oraz 127_{10}) jest zarezerwowana dla znaków sterujących[8]. W tabeli X.4 przedstawiono kod ASCII. Dla uproszczenia liczby odpowiadające poszczególnym znakom przedstawiono w formie dziesiętnej. W rzeczywistości stosowany jest tam, jak wcześniej wspomniano, naturalny kod dwójkowy o siedmiobitowym słowie.

Gdyby wszyscy ludzie na świecie posługiwali się językiem angielskim, który w zapisie korzysta wprost z alfabetu łacińskiego bez znaków narodowych, to kod ASCII byłby zupełnie wystarczający do zapisu wszystkich znaków literowych. Jednak jak wiadomo tak nie jest[9] i względnie znaków narodowych wymagało rozszerzenia kodu ASCII. **Rozszerzony kod ASCII** posługiwał się jednobajtową reprezentacją binarną. Zwiększyło to dwukrotnie ilość możliwych do zapisania znaków o wartości kodu od 128_{10} do 255_{10} i powstały tabele kodowe ASCII po jednej dla określonego języka. Dotyczyło to jednak języków stosujących alfabet łaciński w którym dodatkowo uwzględniano znaki narodowe. Tabele te jako strony kodowe wprowadzane były do oprogramowania komputerów. Nie wyczerpywało to jednak wszystkich potrzeb w dziedzinie reprezentacji znaków.

Aby zaspokoić wszystkie możliwości zapisu znaków i to we wszystkich językach (łącznie z takimi, które mają znaki w postaci ideogramów jak języki azjatyckie) wprowadzono nowy system kodowania noszący nazwę Unicode i posiadający dwie reprezentacje USC-2 i USC-4. W **Unicode USC-2** każdy znak ma reprezentację dwubajtową umożliwiająca zapis 65536 różnych znaków pisarskich i symboli. W **Unicode USC-4** do reprezentacji każdego znaku wykorzystywane są aż cztery bajty.

[8] Klawiatura dalekopisu podobna była do elektrycznej maszyny do pisania i wymagała np. sterowania ruchem przesuwanego papieru, powrotem karetki, zastąpienia błędnego znaku i wielu innych czynności.
[9] Na świecie jest ponoć około 6800 języków [2]

Tabela X.4. Kod ASCII

Dziesiętnie	Znak	Skrót	Dziesiętnie	Znak	Dziesiętnie	Znak	Dziesiętnie	Znak
0	zero	NUL	32	Spacja	64	@	96	`
1	początek nagłówka	SOH	33	!	65	A	97	a
2	początek tekstu	STX	34	"	66	B	98	b
3	koniec tekstu	ETX	35	#	67	C	99	c
4	koniec transmisji	EOT	36	$	68	D	100	d
5	nawiązanie łączności	ENQ	37	%	69	E	101	e
6	potwierdzenie	ACK	38	&	70	F	102	f
7	sygnał dźwiękowy	BEL	39	'	71	G	103	g
8	usunięcie znaku	BS	40	(	72	H	104	h
9	tabulator poziomy	HT	41	)	73	I	105	i
10	nowy wiersz	LF	42	*	74	J	106	j
11	tabulator pionowy	VT	43	+	75	K	107	k
12	nowy arkusz/nowa strona	FF	44	,	76	L	108	l
13	powrót karetki	CR	45	-	77	M	109	m
14	włączenie specjalnego zestawu znaków	SO	46	.	78	N	110	n
15	wyłączenie specjalnego zestawu znaków	SI	47	/	79	O	111	o
16	zmiana znaczenia kolejnych znaków	DLE	48	0	80	P	112	p
17	sterowanie urządzeniem 1	DC1	49	1	81	Q	113	q
18	sterowanie urządzeniem 2	DC2	50	2	82	R	114	r
19	sterowanie urządzeniem 3	DC3	51	3	83	S	115	s
20	sterowanie urządzeniem 4	DC4	52	4	84	T	116	t
21	potwierdzenie negatywne	NAK	53	5	85	U	117	u
22	synchronizacja w stanie bezczynności	SYN	54	6	86	V	118	v
23	koniec bloku transmisji	ETB	55	7	87	W	119	w
24	anulowanie	CAN	56	8	88	X	120	x
25	koniec nośnika	EM	57	9	89	Y	121	y
26	znak do zastąpienia	SUB	58	:	90	Z	122	z
27	przełączenie znaczenia kodu	ESC	59	;	91	[	123	{
28	separator pliku	FS	60	<	92	\	124	|
29	separator grupy	GS	61	=	93	]	125	}
30	separator rekordu	RS	62	>	94	^	126	~
31	separator jednostki	US	63	?	95	_	127	Delete

Kody znakowe są powszechnie wykorzystywane przy współpracy klawiatury komputera PC z jednostką centralną. Naciśnięcie każdego klawisza generuje słowo bitowe reprezentujące określony znak przez co staje się on zrozumiały dla komputera.

Literatura

[1] Kordos M.: Wykłady z historii matematyki. SCRIPT, Warszawa 2006

[2] Clark Scott H.A.: W sercu PC. Helion. Gliwice 2003

[3] Crilly T.: 50 teorii matematyki. PWN, Warszawa 2009

[4] Kalisz J.: Postawy elektroniki cyfrowej. WKiŁ, Warszawa 2007

Literatura przedmiotu

Horowitz P., Hill W.: Sztuka elektroniki, tom I. WKiŁ, Warszawa 1997

Horowitz P., Hill W.: Sztuka elektroniki, tom II. WKiŁ, Warszawa 1997

Kaźmierkowski M.P., Matysik J.T.: Wprowadzenie do elektroniki i energoelektroniki. Oficyna Wydawnicza PW, Warszawa 2005

Łuba T., Zbierzchowski B.: Układy logiczne. Wydawnictwo WIT, Warszawa 2005

Mano M.M., Kime Ch.R.: Podstawy projektowania układów logicznych i komputerów. WNT, Warszawa 2007

Null L., Labur J.: Struktura organizacyjna i architektura systemów komputerowych. Helion, Gliwice 2004

Stallings W. Organizacja i architektura systemu komputerowego. WNT, Warszawa 2004

Skorupski A. Podstawy techniki cyfrowej. WKiŁ, Warszawa 2001

Tanenbaum A.S.: Struktura organizacyjna systemów komputerowych. Helion, Gliwice 2006

Titze U., Schenk Ch.: Układy półprzewodnikowe. WNT, Warszawa 2009

Watson J. Elektronika. WKiŁ, Warszawa 1999