1.2 Kody numeryczne i arytmetyka cyfrowa

1. Kody numeryczne

Kod numeryczny (lub inaczej ***kod liczbowy***, kod arytmetyczny, ***zapis liczbowy***, notacja liczbowa, lub ***system zapisu liczb***). Jeśli zbiorem obiektów kodowanych V_1 są liczby to kod nazywamy kodem numerycznym Typowe alfabety V_2 dla kodu numerycznego to $V_2 = \{0,1\}$, $V_2 = \{0,1,2,3,4,5,6,7\}$, $V_2 = \{0,1,2,3,...,9\}$, $V_2 = \{0,1,...,9,A,B,C,D,E,F\}$ a w kryptografii $V_2 = \{a,b,c,...,z\}$.

W tym momencie mogłoby się zrodzić w umyśle Uważnego Czytelnika straszliwe podejrzenie, że na świecie mamy tylko 2 kategorie kodów kody alfanumeryczne i numeryczne. Na szczęście nie grozi nam nuda i mamy jeszcze wiele, wiele różnych rodzajów kodów np. kody paskowe (inaczej kreskowe). W kodach paskowych (są różne rodzaje tych kodów ale wszystkie są kodami o stałej długości słowa kodowego) obiektami kodowanymi są np. towary znajdujące się w handlu a literami alfabetu, w którym zapisujemy słowo kodowe są paski o różnej grubości (grubość paska jest ściśle związana z jedną z cyfr 0,1,2,3,4,5,6,7,8,9 . Przykładem takiego kodu jest np. kod EAN 13 .

O wartości i praktycznej przydatności kodu numerycznego czyli zapisu liczbowego decyduje łatwość wykonywania pięciu elementarnych działań na reprezentowanych danym zapisem liczbach. Konkretnie chodzi o działanie wyznaczania liczby przeciwnej do danej liczby (działanie jednoargumentowe), dodawania, odejmowania, mnożenia i dzielenia liczb (działania dwuargumentowe). Bardzo istotna jest również łatwość porównywania liczb w danym zapisie. Pewnego komentarza wymaga tu sformułowanie "łatwość wykonywania", które oznacza łatwość realizacji algorytmu przez człowieka lub przez hardware albo software komputera.

2. Naturalne kody wagowe dla liczb naturalnych i zera

Podstawowy fakt z arytmetyki umożliwiający skonstruowanie dużej klasy kodów liczbowych tzw. naturalnych kodów wagowych jest następujący:

Twierdzenie: Dla dowolnej liczby $m \in N$ i ustalonego ciągu $(m_i)_{i=0}^{\infty}$, gdzie $m_i \in \{2,3,...\}$ istnieje dokładnie jeden ciąg skończony $a_k, a_{k-1}, ..., a_1, a_0$ taki, że

1) dla każdego $i = 0,1,...,k$, $a_i \in \{0,1...m_i - 1\}$, $a_k \neq 0$

2) $$m = a_k m_{k-1} m_{k-2} \cdot ... \cdot m_0 + a_{k-1} m_{k-2} m_{k-3} \cdot ... \cdot m_0 + ... a_2 m_1 m_0 + a_1 m_0 + a_0$$

Dowód: Prosty dowód pozostawiamy Czytelnikowi (por. też D.E.Knuth [5]) ■

Wniosek: Dla dowolnej liczby $m \in N$ i ustalonej liczby $W \in \{2,3,...\}$ istnieje dokładnie jeden ciąg skończony $a_k, a_{k-1}, ..., a_1, a_0$ taki, że

1) $a_k, a_{k-1}, ..., a_1, a_0 \in \{0,1...W-1\}$, $a_k \neq 0$

2) $m = a_k W^k + a_{k-1} W^{k-1} + ... + a_1 W + a_0$

Uwaga: Powyższy wniosek pozostaje również z pewnymi modyfikacjami prawdziwy dla W = -2 (tzw. zapis minusdwójkowy) i innych wag ujemnych (por. [10]). Wagi ujemne nie mają jednak obecnie większego praktycznego znaczenia.

Liczba W nosi nazwę wagi lub podstawy zapisu wagowego (ang. radix). W praktyce na oznaczenie liczb $\{0,1,...W-1\}$ używamy ustalonych symboli z pewnego równolicznego z $\{0,1,...W-1\}$ alfabetu V, nazywając je cyframi. Wygodnie jest niekiedy te symbole utożsamiać z liczbą ze zbioru $\{0,1,...W-1\}$ w tym sensie, że raz traktujemy je jak symbole, raz jak liczby. Możemy też wręcz przyjąć, że $V = \{0,1,...W-1\}$.

Odwzorowanie $K : N \cup \{0\} \ni m \to K(m) = a_k a_{k-1} ... a_1 a_0 \in V^*$ określone na zbiorze $N \cup \{0\}$ i zdefiniowane w powyższym wniosku nazywamy kodem wagowym z wagą $W \geq 2$ przy czym przyjmujemy $K(0) = 0$.

Typowe najczęściej używane wagi to *2, 8, 10, 16, 26* (26 to liczba małych liter w alfabecie angielskim). Wag *W=26* i *W=256* używamy w kryptografii do utożsamienia tekstu z liczbą ze zbioru $N \cup \{0\}$.

Zapisy wagowe dla wag *2, 8, 10, 16,* noszą następujące nazwy:

W=2, ***kod NKB*** - ***naturalny kod binarny*** lub ***zapis dwójkowy naturalny*** (lub zwykły zapis dwójkowy), mamy dwie cyfry: 0, 1

W=8, ***zapis oktalny***, mamy 8 cyfr 0, 1, 2, 3, 4, 5, 6, 7

W=10, ***zapis dziesiętny,*** mamy 10 cyfr 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

W=16, ***zapis hexadecymalny*** (lub szesnastkowy) , mamy16 cyfr 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F (stosujemy dodatkowe cyfry A, B, C, D, E, F)

Jeśli mamy wątpliwości, jakiego zapisu używamy, to po zapisie liczby stosujemy znak spoza alfabetu w jakim zapisujemy liczbę, np.: *B* – dla zapisu binarnego, *O* - dla zapisu oktalnego, *D* - dla zapisu dziesiętnego i *H* - dla zapisu hexadecymalnego. Czasami z kontekstu wynika jakim zapisem operujemy więc dodatkowe wyjaśnienia nie są potrzebne. Zapis dziesiętny będziemy w dalszym ciągu traktować zwyczajowo jako wyróżniony i z reguły literę D będziemy pomijać.

Przykład: 101(*B*) lub 101*B* dla zapisu binarnego oraz 1A2(*H*) lub 1A2*H* dla zapisu hexadecymalnego. ■

Wygodnie jest też ciąg $K(m) = a_k a_{k-1} ... a_1 a_0 \in V^*$ utożsamiać jeśli trzeba z liczbą m tak jak to zwyczajowo czynimy dla zapisu z wagą *W=10* czyli dla dobrze znanego zapisu dziesiętnego.

Przykład: Zapis binarny : 5= 1001 B, 8= 1000 B, 7=111 B

Przykład: Zapis hexadecymalny: 8= 8 H, 7=7 H, 15=F H

Przykład: Kod *NKB* (naturalny kod binarny) dla zbioru liczb {0,1,...,7}.

$0 \to 0$
$1 \to 1$
$2 \to 10$
$3 \to 11$
$4 \to 100$
$5 \to 101$
$6 \to 110$
$7 \to 111$ ■

Warto podkreślić, że zbiorem obiektów kodowanych jest dla rozważanych w tym punkcie kodów zbiór liczb naturalnych z zerem tzn. $N \cup \{0\}$ a więc rozważane kody są kodami numerycznymi. Jednocześnie słowa kodowe mogą być dowolnie długie. Nie są to więc kody ze stałą długością słowa kodowego. Dla ustalonej liczby $a \in N \cup \{0\}$ i ustalonej wagi $W \in \{2,3,...\}$ łatwo można obliczyć długość $l(a)$ słowa kodowego reprezentującego liczbę *a.* Jeśli $a = 0$ to $l(a) = 1$, jeśli $a \neq 0$ to

$$l(a) = \lfloor \log_W a \rfloor + 1 \qquad (*)$$

gdzie $\lfloor \cdot \rfloor : R \to Z$ jest tzw. funkcją podłogi (lub jak mówimy krótko podłogą). Jeśli $x \in R$ to $\lfloor x \rfloor$ jest największą liczbą całkowitą nie większą od *x.*

Przykład: Obliczmy korzystając ze wzoru (*) za pomocą ilu bitów zapisujemy w kodzie NKB liczby 2^n, $2^n + 1$, $2^n - 1$. Ponieważ $\log_2 2^n = n$ i funkcja logarytmiczna jest ściśle monotoniczna łatwo obliczamy kolejno:

$l(a) = \lfloor \log_2 2^n \rfloor + 1 = n + 1$

$l(a) = \lfloor \log_2 (2^n + 1) \rfloor + 1 = n + 1$

$l(a) = \lfloor \log_2 (2^n - 1) \rfloor + 1 = n$

gdzie *a* oznacza słowo binarne reprezentujące liczbę.

Przykład: Ile cyfr hexadecymalnych potrzeba do zapisu liczby 2^n? Stosując wzór (*) dostajemy $l(a) = \lfloor \log_{16} 2^n \rfloor + 1 = \lfloor n/4 \rfloor + 1$.

Przykład: Obliczmy korzystając ze wzoru (*) za pomocą ilu cyfr dziesiętnych zapisujemy w naturalnym kodzie wagowym dziesiętnym (czyli jak mówimy krótko w zapisie dziesiętnym) liczby 2^n, dla *n*=10. Ponieważ $\log_{10} 2^n = n \log_{10} 2$ łatwo obliczamy kolejno: $l(a) = \lfloor \log_{10} 2^{10} \rfloor + 1 = \lfloor 10 \log_{10} 2 \rfloor + 1 = 4$. Podobnie można wykazać, że taka sama liczba cyfr dziesiętnych jest potrzebna do zapisu liczby $2^n + 1$ i $2^n - 1$ przy *n*=10. ■

Algorytm dodawania w naturalnym zapisie wagowym z wagą W . Algorytm dodawania w naturalnym zapisie wagowym z wagą *W* jest w zasadzie identyczny jak w przypadku naturalnego zapisu dziesiętnego. Jeśli dodajemy w naturalnym zapisie wagowym z wagą *W* 2 liczby *a* i *b* reprezentowane słowami

$$a = a_n a_{n-1} ... a_1 a_0 \quad \text{i} \quad b = b_k b_{k-1}, ... b_1 b_0$$

to żeby uzyskać sumę $s = s_n s_{n-1} ... s_1 s_0$ postępujemy tak:

1) Obliczamy $s_0 = a_0 + b_0 \pmod{W}$. Jeśli $s_0 = a_0 + b_0 \geq$ W to podstawiamy $c_1 := 1$ a jeśli $s_0 = a_0 + b_0 <$W to podstawiamy $c_1 := 0$

2) Obliczamy $s_1 = a_1 + b_1 + c_1 \pmod{W}$. Jeśli $s_1 = a_1 + b_1 + c_1 \geq W$ to podstawiamy $c_2 := 1$ a jeśli $s_1 = a_1 + b_1 + c_1 < W$ to podstawiamy $c_2 := 0$

.....

i) Obliczamy $s_1 = a_i + b_i + c_i \pmod{W}$. Jeśli $s_i = a_i + b_i + c_i \geq W$ to podstawiamy $c_{i+1} := 1$ a jeśli $s_i = a_i + b_i + c_i < W$ to podstawiamy $c_{i+1} := 0$.

.....

n) Obliczamy $s_n = a_n + b_n + c_n \pmod{W}$. Jeśli $s_n = a_n + b_n + c_n \geq W$ to podstawiamy $c_{n+1} := 1$ a jeśli $s_n = a_n + b_n + c_n < W$ to podstawiamy $c_{n+1} := 0$.

Algorytm mnożenia w naturalnym zapisie wagowym z wagą W jest w zasadzie identyczny jak w przypadku naturalnego zapisu dziesiętnego ale używamy „tabelki mnożenia" właściwej dla W tzn. używamy „tabelki mnożenia" dla par liczb $< 0, W-1 > \times < 0, W-1 >$.

Istota rzeczy: algorytmy dodawania i mnożenia w naturalnych zapisach wagowych pozostają właściwie takie same jak w przypadku dodawania liczb w zapisie dziesiętnym.

Przykład: Mamy 2 liczby w zapisie NKB $a = 10101010$ $b = 11111111$. Stosując opisany algorytm otrzymujemy sumę:

Suma:
$$\begin{array}{r} 10101010 \\ 11111111 \\ \hline 110101001 \end{array}$$

Przykład: Mamy 2 liczby w zapisie NKB $a = 1010$ $b = 1111$. Stosując opisany algorytm mnożenia otrzymujemy iloczyn:

Iloczyn:
$$\begin{array}{r} 1010 \\ 1111 \\ \hline 1010 \\ 1010 \\ 1010 \\ 1010 \\ \hline 10010110 \end{array}$$
■

3. Naturalne kody wagowe o stałej długości słowa kodowego

Jeśli operujemy słowami kodowymi o stałej długości n i $V_2 = \{0,1,...,W-1\}$, gdzie $W \in \{2,3,...\}$ to różnych słów kodowych jest W^n . Tyle jest *n* elementowych wariacji

z powtórzeniami ze zbioru W elementowego. Zatem zbiór obiektów kodowanych może zawierać co najwyżej W^n elementów.

Naturalny kod wagowy o stałej długości słowa kodowego to kod, które dla ustalonego W koduje liczby ze zbioru $\{0,1,2,...,W^n-1\}$ w taki sposób, że słowa kodowe należą do iloczynu kartezjańskiego W^n. Algorytm tworzenia słów kodowych dla takiego kodu numerycznego polega na uzupełnianiu słowa utworzonego dla naturalnego zapisu wagowego zerami z lewej strony aż do uzyskania wymaganej długości słowa kodowego n.

Z kodami numerycznymi o stałej długości (w szczególności z kodami wagowymi o stałej długości) wagowymi wiążemy pojęcie ***zakresu zapisu***. Zakres zapisu to zbiór liczb reprezentowanych w tym zapisie. Jeśli rozważany kod jest binarny i słowo kodowe n bitowe ma postać $a_{n-1}...a_1a_0 \in \{0,1\}^n$ to bit a_{n-1} nazywamy bitem najbardziej znaczący lub bitem MSB (Most Significant Bit) a bit a_0 bitem najmniej znaczącym lub bitem LSB (Least Significant Bit).

Przykład: Kod *NKB* (naturalny kod binarny) o długości 3 dla zbioru liczb {0,1,...,7}. Kod *NKB* czyli naturalny kod binarny nazywa się też naturalnym kodem dwójkowym, lub naturalnym (lub zwykłym) zapisem dwójkowym.

$0 \rightarrow 000$
$1 \rightarrow 001$
$2 \rightarrow 010$
$3 \rightarrow 011$
$4 \rightarrow 100$
$5 \rightarrow 101$
$6 \rightarrow 110$
$7 \rightarrow 111$

Przykład: Naturalny kod binarny o długości 4 podany niżej ma zakres <0,15>.

$0 \rightarrow 0000$
$1 \rightarrow 0001$
$2 \rightarrow 0010$
$3 \rightarrow 0011$
$4 \rightarrow 0100$
$5 \rightarrow 0101$
$6 \rightarrow 0110$
$7 \rightarrow 0111$
$8 \rightarrow 1000$
$9 \rightarrow 1001$
$10 \rightarrow 1010$
$11 \rightarrow 1011$
$12 \rightarrow 1100$
$13 \rightarrow 1101$
$14 \rightarrow 1110$
$15 \rightarrow 1111$ ■

4. Konwersja naturalnych zapisów wagowych

Chcemy przejść z zapisu naturalnego wagowego z wagą W_1 na zapis naturalny wagowy z wagą W_2. Postępujemy według schematu

Zapis $a_k a_{k-1}...a_0$ z wagą $W_1 \xrightarrow{(1)}$ wartość m $\xrightarrow{(2)}$ Zapis $a_r' a_{r-1}'...a_0'$ z wagą W_2

(1) Obliczamy po prostu wartość *m* zgodnie ze wzorem

$$m = a_k W_1^k + a_{k-1} W^{W-1} + ... + a_1 W + a_0$$

Wygodnie w tym celu posłużyć się ***schematem Hornera***, czyli algorytmem obliczenia wartości wielomianu $a_k x^k + a_{k-1} x^{k-1} + ... + a_0$ w punkcie *x*.

Schemat Hornera: $(...((a_k x + a_{k-1})x + a_{k-2})x + ... + a_1)x + a_0 = a_k x^k + ... + a_0$

(2) Dzielimy z resztą liczbę m przez W_2 otrzymując ciąg reszt $a_r' a_{r-1}'....a_0'$, dokładniej:

$$\begin{aligned} m &= m_1 \cdot W_2 + a_0' \\ m_1 &= m_2 \cdot W_2 + a_1' \\ m_2 &= m_3 \cdot W_2 + a_2' \\ &\vdots \\ m_{r-1} &= m_r \cdot W_2 + a_{r-1}' \end{aligned} \quad .$$

itd. aż kolejne $m_r = 0$ co stanowi regułę stopu algorytmu konwersji.

Ostatecznie uzyskujemy zapis $a_r' a_{r-1}'...a_0'$ z wagą W_2

5. Uzupełnienie do W-1

Dokładniej *uzupełnienie do W-1* (gdzie $W \in N, W \geq 2$) słowa $a_n a_{n-1} a_{n-2}...a_0$ nad alfabetem $\{0,1,...,W-1\}$. Jest to słowo $b_n b_{n-1} b_{n-2}...b_0$ takie, że $a_i + b_i = W - 1$ dla i=0,1,2,...,*n*, gdzie dodawanie jest zwykłym dodawaniem ze zbioru liczb całkowitych .

Dla ustalonego $n \in N \cup \{0\}$ uzupełnienie do *W*-1 jest więc odwzorowaniem:

$$U_1 : \{0,1,...,W-1\}^{n+1} \to \{0,1,...,W-1\}^{n+1}$$

Łatwo sprawdzić, że odwzorowanie to jest idempotentne tzn. $U_1 \circ U_1 = id$, czyli uzupełnienie do *W*-1 zastosowane dwa razy do danego słowa daje to samo słowo.

Przykład: Weźmy *W*=2 i słowo ośmiobitowe 01010101, uzupełnieniem do W-1 czyli uzupełnieniem do jedności tego słowa jest słowo 10101010.

Przykład: Niech W=10 i niech będzie dane słowo 01010101, uzupełnieniem do W-1 czyli uzupełnieniem do dziewięciu tego słowa jest słowo 98989898.

Przykład: Niech W=16 i niech będzie dane słowo 01010101, uzupełnieniem do W-1 czyli uzupełnieniem do piętnastu tego słowa jest słowo FEFEFEFE. ■

Najważniejsze w praktyce jest uzupełnienie do jedności (ang. 1's complement)

6. Uzupełnienie do W

Dokładniej *uzupełnienie do W* (gdzie $W \in N, W \geq 2$) słowa $a_n a_{n-1} a_{n-2} ... a_0$ nad alfabetem $\{0,1,...,W-1\}$. Niech słowo $b_n b_{n-1} b_{n-2} ... b_0$ będzie uzupełnieniem do W-1 słowa $a_n a_{n-1} a_{n-2} ... a_0$. Potraktujmy słowo $b_n b_{n-1} b_{n-2} ... b_0$ jak liczbę w zapisie naturalnym wagowym z wagą W i dodajmy do tej liczby 1 arytmetycznie zaniedbując ewentualne przeniesienie na pozycję n+1. Uzyskana w ten sposób liczba $c_n c_{n-1} c_{n-2} ... c_0$ (w zapisie naturalnym wagowym z wagą W) nosi nazwę uzupełnienia do W.

Dla ustalonego $n \in N \cup \{0\}$ uzupełnienie do W jest więc odwzorowaniem:

$$U_2 : \{0,1,...,W-1\}^{n+1} \rightarrow \{0,1,...,W-1\}^{n+1}$$

Łatwo sprawdzić, że odwzorowanie to jest idempotentne tzn. $U_w \circ U_w = id$, czyli uzupełnienie do W zastosowane dwa razy do danego słowa daje to samo słowo.

Przykład: Weźmy W=2 i słowo ośmiobitowe 01010101, uzupełnieniem do W-1 czyli uzupełnieniem do dwóch tego słowa jest słowo (10101010 B) +1=10101011.

Przykład: Niech W=10 i niech będzie dane słowo 01010101, uzupełnieniem do W-1 czyli uzupełnieniem do dziewięciu tego słowa jest słowo (98989898 D)+1=98989899. ■

Najważniejsze w praktyce jest uzupełnienie do dwóch (ang. 2's complement)

7. Zapis moduł znak

W powyżej przedstawiony w punktach 1-4 sposób reprezentujemy łatwo słowami nad ustalonym alfabetem liczby ze zbioru $N \cup \{0\}$ czyli liczby naturalne i zero. Zajmiemy się teraz sposobami reprezentowania liczb ujemnych. Rozszerzymy więc zbiór V_1 liczb reprezentowanych na wszystkie liczby całkowite Z.

Omówimy 3 zasadnicze kody numeryczne służące do tego celu: tzw. zapis moduł znak (ang. sign magnitude notation), zapis uzupełnień do W, gdzie W jest wagą lub jak mówimy podstawą zapisu (w szczególności dla W=2 uzyskujemy zapis uzupełnień do 2), zapis uzupełnień do W-1 i jego szczególny przypadek zapis uzupełnień do 1.

Załóżmy, że mamy ustaloną wagę $W \geq 2$ i chcemy zapisać liczbę całkowitą a za pomocą zapisu moduł znak. Postępujemy wówczas tak: zapisujemy za pomocą naturalnego zapisu wagowego z wagą W liczbę $|a|$ uzyskując słowo $a_{n-1} a_{k-1} ... a_0$, gdzie $a_i \in <0, W-1>$.

Słowem reprezentującym liczbę a będzie słowo $a_n a_{n-1} a_{k-1} ... a_0$ gdzie $a_n \in \{0,1\}$ przy czym jeśli $a \geq 0$ to $a_n = 0$ a jeśli $a < 0$ to $a_n = 1$.

Zauważmy, że zapis moduł znak jest na dobrą sprawę tym samym pomysłem co zapis liczb dziesiętnych ze znakiem jakim posługujemy się na co dzień.

Zapis moduł znak oczywiście może oczywiście tak zmodyfikować by był kodem o stałej długości słowa kodowego. Istotny staje się wówczas zakres zapisu moduł znak. Jeśli używamy n+1 bitów do zapisu słowa kodowego to zakres ten jest równy $< -2^n + 1, 2^n - 1 >$.

Przykład: Zapiszmy liczbę -5 w zapisie binarnym moduł znak (ze słowem kodowym o długości 8). Liczbę 5 zapisujemy wstępnie jako słowo 7 bitowe 0000101 i ponieważ $-5<0$ jako pierwszy z lewej strony bit wpisujemy 1 uzyskując: 10000101. ■

8. Zapis uzupełnień do W i zapis uzupełnień do 2

Dokładniej, omówimy zapis uzupełnień do W (kod numeryczny uzupełnień do W) dla liczb całkowitych $n \in Z$ (a więc również ujemnych). Wykorzystamy następujący fakt.

Fakt: Wybierzmy dowolne $m \in Z$, $m \neq 0$ i ustalmy wagę W $(W \in N, W \geq 2)$. Wówczas istnieje dokładnie jeden ciąg: $a_n a_{n-1} ... a_0$, gdzie $n > 0$, taki że dla każdego $i = 1,2,...,n-1$ $a_i \in \{0,1,...,W-1\}$, $a_n \in \{0, W-1\}$, oraz $a_n = 0$ i $a_{n-1} \neq 0$ lub $a_n = 1$ taki, że:

$$n = -(\operatorname{sgn} a_n) W^n + \sum_{i=0}^{n-1} a_i W^i \qquad (*)$$

Dowód: Prosty dowód pozostawiamy Czytelnikowi jako ćwiczenie.■

Dla przypomnienia: funkcja $\operatorname{sgn} : R \to \{-1,0,1\}$ (tzw. funkcja signum, czyli znak),

zdefiniowana jest dla $x \in R$ wzorem: $\operatorname{sgn}(x) = \begin{cases} 1 \text{ gdy} & x > 0 \\ 0 \text{ gdy} & x = 0 \\ -1 \text{ gdy} & x < 0 \end{cases}$

Jak widać dla $m \in N, m \neq 0$ dzięki powyższemu faktowi możemy zdefiniować odwzorowanie $f : Z \setminus \{0\} \to < 0, W-1 >^*$ wzorem $f(m) = a_n a_k ... a_0 \in < 0, W-1 >^*$. Jeśli przyjmiemy dodatkowo, że $f(0) = 0 \in < 0, W-1 >^*$, to określimy odwzorowanie

$$f : Z \to < 0, W-1 >^*$$

nazywane kodem uzupełnień do W.

Uwaga: Widać od razu, że współrzędna n (a dokładniej a_n) określa znak reprezentowanej liczby.

1. Jeśli $a_n = 0$ (tzn. słowa kodowe są postaci $0 a_{n-1} a_{n-2} ... a_0$) to liczba jest nieujemna

2.Jeśli $a_n = W-1$ (tzn. słowa kodowe są postaci $(W-1)a_{n-1}a_{n-2}...a_0$) to liczba jest ujemna.

Algorytm obliczania liczby przeciwnej do danej jest w zapisach uzupełnień do W jest następujący. Jeśli $m \in Z$ i słowo $a_n a_{n-1}...a_0$ reprezentuje tę liczbę to słowo reprezentujące $-m$ uzyskujemy jako uzupełnienie do W słowa $a_n a_{n-1}...a_0$.

Powyżej zdefiniowany kod uzupełnień do W jest kodem o zmiennej długości słowa kodowego. Zdefiniujemy teraz kod uzupełnień do W o stałej długości słowa kodowego.

Wykorzystamy następujący fakt:

Fakt: Wybierzmy dowolne $m \in <-W^n, W^n - 1>$, gdzie $n > 0$ i ustalmy wagę W $(W \in N, W \geq 2)$. Wówczas istnieje dokładnie jeden ciąg: $a_n a_{n-1}...a_0$, taki że dla każdego $i = 1,2,...,n$, $a_i \in \{0,1,...,W-1\}$, taki, że:

$$n = -(\operatorname{sgn} a_n)W^n + \sum_{i=0}^{n-1} a_i W^i \qquad (*)$$

Dowód: Prosty dowód pozostawiamy Czytelnikowi jako ćwiczenie.

Odwzorowanie $f :<-W^n, W^n - 1> \rightarrow <0, W-1>^n$ zdefiniowane w powyższym twierdzeniu nosi nazwę zapisu uzupełnień do W (o stałej długości słowa kodowego).

Zakres tak zdefiniowanego kodu jest równy oczywiście $<-W^n, W^n - 1>$.

Algorytm obliczania liczby przeciwnej do danej jest w zapisach uzupełnień do W (o stałej długości) jest następujący. Jeśli $m \in <-W^n + 1, W^n - 1>$ i słowo $a_n a_{n-1}...a_0$ reprezentuje tę liczbę to słowo reprezentujące $-m$ uzyskujemy jako uzupełnienie do W słowa $a_n a_{n-1}...a_0$.

W systemach cyfrowych stosujemy głównie zapis uzupełnień do 2 (ang. two`s complement notation), który nazywany jest również zapisem U2. Jest to obok zapisu NKB najczęściej stosowany w technice mikroprocesorowej kod numeryczny. Popularność zapisu U2 wynika m.in. z faktu, że algorytmy dodawania i odejmowania w U2 dają się zrealizować a sposób naturalny w zwykłym, prostym sumatorze sumującym liczby w kodzie NKB.

W kodzie U2 (o stałej długości słowa n+1) słowo binarne $a_n a_{n-1}...a_0$ związane jest z reprezentowaną tym słowem liczbą za pomocą wzoru

$$n = -(\operatorname{sgn} a_n)2^n + \sum_{i=0}^{n-1} a_i 2^i = -a_n 2^n + \sum_{i=0}^{n-1} a_i 2^i$$

a zakres reprezentowanych liczb jest równy $<-2^n, 2^n - 1>$

Bit a_n nazywamy bitem najbardziej znaczącym lub bitem MSB od ang. Most Significant Bit) a bit a_0 bitem najmniej znaczącym lub bitem LSB od ang. Least Significant Bit.

W zapisie *U*2, żeby obliczyć *–m* mając kod U2 liczby *m* (czyli rozwinięcie, jak czasem mówimy) negujemy wszystkie bity i na najmniej znaczącej pozycji dodajemy arytmetycznie 1 (tzn. z uwzględnieniem przeniesień). Uzyskany wynik, tzn. słowo binarne reprezentuje

tzn. jest kodem *U*2 liczby $-m$. Jest to bardzo wygodny algorytm ale uwaga $-m$ musi należeć do zakresu zapisu.

Przykład: Zapiszmy liczbę -5 w 8 bitowym zapisie uzupełnień do 2 (tzn. ze słowem kodowym o długości 8). Liczbę 5 zapisujemy wstępnie jako słowo 8 bitowe 00000101 (w 8 bitowym kodzie NKB). Ponieważ $-5<0$ obliczamy uzupełnienie do 2 słowa 00000101 i uzyskujemy ostatecznie 11111011.

Przykład: Zapiszmy liczbę -1 w 16 bitowym zapisie uzupełnień do 2 (tzn. ze słowem kodowym o długości 8). Liczbę 1 zapisujemy wstępnie jako słowo 16 bitowe $\underbrace{00...1}_{n+1}$ (w 16 bitowym kodzie NKB). Ponieważ $-1<0$ obliczamy uzupełnienie do 2 słowa $\underbrace{00...001}_{15}$ (15 zer i jedynka na końcu) i uzyskujemy ostatecznie $\underbrace{11...1}_{16}$.
Czyli -1 to 16 jedynek. ■

9. Zapis uzupełnień do W–1 i zapis uzupełnień do 1

Zapis uzupełnień do *W*-1 służy do zapisywania liczb całkowitych. Jeśli liczba $m \in Z$ jest nieujemna, tzn. $m \geq 0$, to zapisujemy ją tak:

$$m = 0a_{n-1}a_{n-2}...a_0$$

gdzie $a_{n-1}a_{n-2}...a_0$ jest naturalnym zapisem wagowym z wagą W liczby m, a więc:

$$m = \sum_{i=0}^{n-1} a_i W^i .$$

Jeśli liczba $m \in Z$ jest ujemna (lub równa 0), tzn. $L \leq Z$, to wstępnie obliczamy jak wyżej słowo kodowe odpowiadające liczbie $|m|$ zapisujemy je tak: $|m| = b_n b_{n-1} b_{n-2}...b_0$ a następnie jako słowo kodowe odpowiadające m przyjmujemy słowo $a_n a_{n-1} a_{n-2}...a_0$ będące uzupełnieniem do *W*-1 słowa $b_n b_{n-1} b_{n-2}...b_0$.

Jeśli w opisanej wyżej procedurze stosujemy zamiast naturalnego zapisu wagowego z wagą W zapis naturalny wagowy z wagą W o stałej długości n słowa kodowego to uzyskujemy zapis uzupełnień do *W*-1 ze stałą długością słowa kodowego. Zakres takiego zapisu uzupełnień do jedności jest równy $< -W^n + 1, W^n - 1 >$.

W systemach cyfrowych wykorzystuje się najczęściej zapis uzupełnień do 1 ze słowem kodowym o stałej długości. Dla takiego zapisu przy słowie o długości n+1 zakres zapisu jest równy $< -2^n + 1, 2^n - 1 >$. Zapis uzupełnień do 1 nazywa się też zapisem U1.

Uwaga: Zauważmy, że z definicji zapisu uzupełnień do *W*-1 wynika różnowartościowość zdefiniowanego kodu na zbiorze $Z \setminus \{0\}$ lub $< -W^n + 1, W^n - 1 > \setminus \{0\}$ (w przypadku zastosowania słowa o długości n+1) ale liczba 0 może mieć 2 reprezentacje. Np. w przypadku zastosowania kodu uzupełnień do 1 o stałej długości słowa n+1 liczba 0 reprezentowana jest każdym ze słów: $\underbrace{00...0}_{n+1}$ oraz $\underbrace{11...1}_{n+1}$ czyli mamy 2 różne słowa oznaczające 0.

Algorytm obliczania liczby przeciwnej jest w zapisie uzupełnień do W-1 wyjątkowo łatwy. Obliczmy po prostu uzupełnienie do W-1 słowa wejściowego. W przypadku U1 sprowadza się to do zanegowania bitów.

Przykład: Zapiszmy liczbę –5 w 8 bitowym zapisie uzupełnień do 1 (tzn. ze słowem kodowym o długości 8). Liczbę 5 zapisujemy wstępnie jako słowo 8 bitowe 00000101 (w 8 bitowym kodzie NKB). Ponieważ –5<0 obliczamy uzupełnienie do 1 słowa 00000101 i uzyskujemy ostatecznie 11111010. ■

10. Zapis stałoprzecinkowy

Już wiemy z poprzednich punktów jak można reprezentować liczby naturalne i 0 w systemie cyfrowym oraz jak można reprezentować liczby całkowite czyli mówiąc niezbyt precyzyjnie liczby ze znakiem. Często jednak chcemy reprezentować w systemie cyfrowym ułamki (liczby wymierne) i do tego celu wykorzystujemy koncepcję zapisu stałoprzecinkowego (ang. fixed point notation).

Oczywiście naturalnym pomysłem jest reprezentowanie liczby wymiernej jako pary uporządkowanej liczb całkowitych (m,n), gdzie $m,n \in Z$. Tak czasem się robi ale algorytmy wykonywania działań arytmetycznych (i porównywanie liczb) okazuje się bardziej skomplikowane niż w zdefiniowanym poniżej zapisie stałoprzecinkowym.

Do uwzględnienia znaku można użyć zapisu uzupełnień do W, uzupełnień do W-1 lub zapisu moduł znak. W szczególności możemy użyć zapisu U2 lub U1. Dla ustalenia uwagi zakładamy, że używamy zapisu uzupełnień do W.

Zapis stałoprzecinkowy (uzupełnień do W) ze stałą długością słowa kodowego definiujemy następująco. Będziemy reprezentować liczby ze zbioru $A = \{-W^n + k \cdot W^{-r};\ k \in <0, W^{n+r+1}>\}$ (gdzie $n \geq 1,\ r \geq 0,\ n,r \in Z$). Liczbę $n \in A$ reprezentujemy słowem $a_n a_{n-1} ... a_0 a_{-1} ... a_{-r}$ (słowem nad alfabetem $<0, W-1>$) tak by był spełniony wzór

$$n = -(\operatorname{sgn} a_n) W^n + \sum_{i=-r}^{n-1} a_i W^i \qquad (*)$$

Jeśli założymy, że $a_n \in \{0, W-1\}$ i $a_i \in <0, W-1>$ to takie słowo $a_n a_{n-1} ... a_0 a_{-1} ... a_{-r}$ jest wyznaczone jednoznacznie.

Rys.1. Rozmieszczenie na osi liczbowej liczb reprezentowanych w zapisie stałoprzecinkowym. Warto zwrócić uwagę na równomierne rozmieszczenie reprezentowanych liczb. ”Skok” czyli odległość sąsiednich liczb wynosi W^{-r}.

Liczba dodatnia reprezentowana jest więc ciągiem $0a_{n-1}a_{n-2}...a_0 a_{-1}...a_{-r}$ i możemy zapisać

$$0a_{n-1}a_{n-2}...a_0a_{-1}...a_{-r} = \sum_{i=-r}^{n-1} a_i W^i$$

a liczba ujemna ciągiem

$$a_n a_{n-1}a_{n-2}...a_0a_{-1}...a_{-r} = -W^n + \sum_{i=-r}^{n-1} a_i W^i$$

gdzie $a_n = W - 1$.

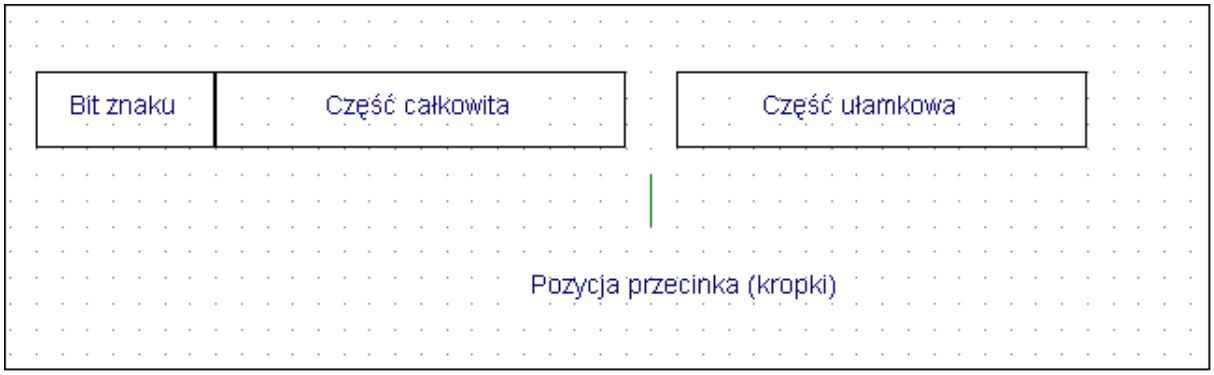
Uwaga: Zauważmy, że słowo o długości $n+r+1$ zostało podzielone na 3 części, 3 pola: „znak", „część całkowitą" (tutaj przecinek lub kropka) i „część ułamkową" (por. Rys. 2 dla przypadku $W=2$). Wprowadzamy przecinek lub kropkę pomiędzy część całkowitą i ułamkową nie zapisując jednak formalnie w specjalny sposób tego symbolu. Poprzestajemy na umowie, że kropka znajduje się w określonym miejscu słowa. Czasami w zapisie stałoprzecinkowym nie ma części całkowitej, czasami nie ma ułamkowej.

Dla $W=2$ czyli dla zapisu binarnego mamy 1 bit znaku, n bitów części całkowitej i r bitów części ułamkowej.

Algorytm obliczania liczby przeciwnej do danej jest identyczny jak w przypadku zapisu uzupełnień do W. W celu znalezienia liczby przeciwnej obliczamy uzupełnienie do W słowa wejściowego.

Analogicznie jak wyżej można zdefiniować zapis stałoprzecinkowy o zmiennej długości słowa kodowego.

Oczywiście zapisy U2, U1 i moduł znak tak jak zdefiniowano je w punktach poprzednich można traktować jako zapisy stałoprzecinkowe z odpowiednio umieszczonym przecinkiem.



Rys. 2. Koncepcja zapisu stałoprzecinkowego

11. Zapis zmiennoprzecinkowy

Zapis zmiennoprzecinkowy lub notacja zmiennoprzecinkowa (czasem mówimy zapis zmiennopozycyjny) to jeden z najczęściej stosowanych w mikroprocesorach kodów numerycznych. Angielskie nazwy to *floating point notation* lub *scientific notation*.

Zasadniczy pomysł jest bardzo prosty. Liczby bardzo duże i bardzo małe wygodnie jest zapisywać w następującej wykładniczej postaci np. $x = -1{,}5 \cdot 10^{15}$ lub ogólniej w postaci $x = (-1)^s \cdot f \cdot W^e$. Możemy zakładać, że $W \in N, \mathrm{W} \geq 2$ ale w praktyce wykorzystujemy $W = 10$ lub $W = 2$.

W dalszym ciągu będziemy zakładać, że $W = 2$ i będziemy rozważać zapis zmiennoprzecinkowy binarny. Liczba $s \in \{0,1\}$ reprezentuje znak liczby x. Liczbę $f \geq 0$ nazywamy mantysą (lub częścią ułamkową ang. mantissa lub fraction) a liczbę $e \in Z$ wykładnikiem (lub cechą ang. exponent). Tak więc liczbę $x = (-1)^s \cdot f \cdot W^e$ możemy reprezentować jako trójkę uporządkowaną (s, f, e) czyli trójkę $(znak, mantysa, wykladnik)$.

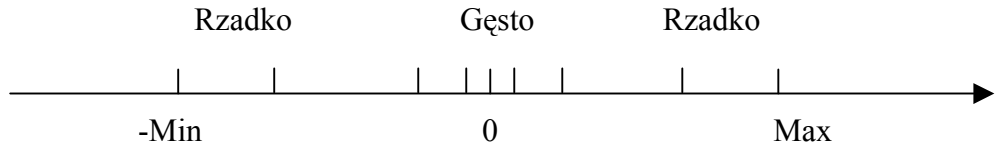
Oczywiście jeśli nie narzucimy dodatkowych warunków na f i e to takie przedstawienie nie jest jednoznaczne. Liczby f i e muszą być ponadto zapisane w ustalonych kodach numerycznych. Zakładamy w dalszym ciągu, że są to kody numeryczne binarne.

Przedstawienie liczby staje się jednoznaczne jeśli liczba ma tzw. *postać znormalizowaną*. Mówimy, że liczba reprezentowana w zapisie ma postać znormalizowaną jeśli mantysa f zapisana w zapisie stałoprzecinkowym bez znaku (z przecinkiem za pierwszym bitem) ma postać: 1.ciąg bitów czyli

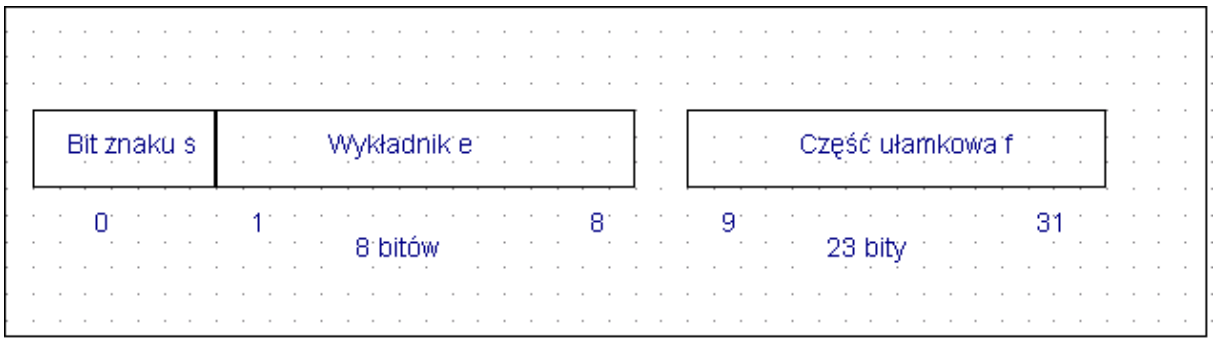
(jedynka przed przecinkiem jako pierwszy bit, przecinek (czyli kropka), ciąg bitów)

co odpowiada wartości f z przedziału [1,2). Ponieważ liczby pamiętamy w postaci znormalizowanej nie ma potrzeba pamiętania pierwszego bitu mantysy f.

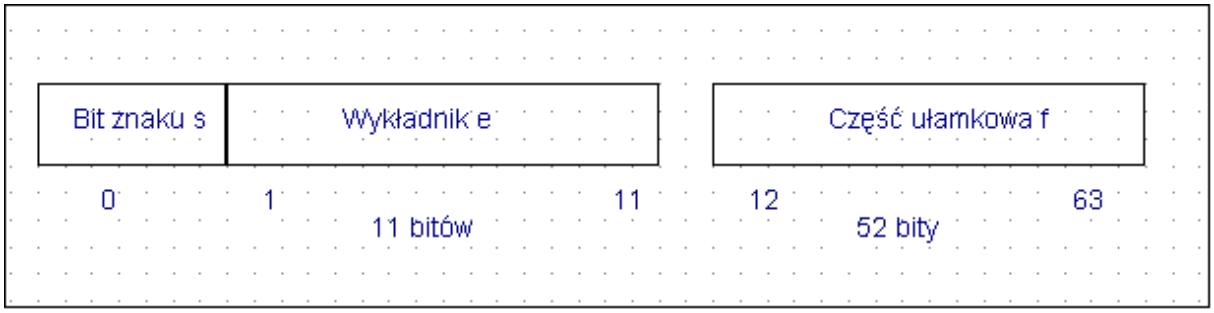
Najczęściej zapis zmiennoprzecinkowy używany jest zgodnie ze standardem IEEE 754. Poniżej opiszemy formaty tego standardu dla słów 32 i 64 bitowych.



Rys.3. Rozmieszczenie na osi liczbowej liczb reprezentowanych w zapisie zmiennoprzecinkowym. Warto zwrócić uwagę na charakterystyczne nierównomierne rozmieszczenie reprezentowanych liczb. Odległość sąsiednich liczb jest najmniejsza w pobliżu zera i rośnie wraz ze wzrostem odległości od 0.



Rys.4. Format standardu zapisu zmiennoprzecinkowego IEEE 754 dla pojedynczej precyzji (ang. single precision). Słowa są 32 bitowe.



Rys.5. Format standardu zapisu zmiennoprzecinkowego IEEE 754 dla podwójnej precyzji (ang. double precision). Słowa są 64 bitowe.

Część ułamkowa f zgodnie uwagą o postaci znormalizowanej pozbawiona jest w standardzie IEEE 754 pierwszej 1. Chcąc więc odtworzyć rzeczywistą wartość f należy poprzedzić ciąg bitów f jedynką i odczytać $(1,f)$ jako liczbę zapisaną w zapisie stałoprzecinkowym bez znaku.

Z kolei bit znaku s=1 oznacza liczb ujemną a równy 0 liczbę dodatnią.

W dosyć specjalnym kodzie zapisany jest wykładnik e jest to mianowicie tzw. obciążony kod NKB. W ogólnym przypadku dla słowa n bitowego jest to kod numeryczny przyporządkowujący liczbom całkowitym ze zbioru $< -2^{n-1}+1,\ 2^{n-1} >$ kolejne słowa kodowe kodu n bitowego NKB. W standardzie IEEE 754 dla słowa 32 bitowego na zapamiętanie e mamy 8 bitów więc stosujemy do kodowania e obciążony 8 bitowy kod NKB.

12. Kody BCD

Ogólnie rzecz biorąc, kod BCD (od ang. Binary Coded Decimal czyli cyfra dziesiętna zakodowana binarnie) to każde odwzorowanie $k : \{0,1,...,9\} \to \{01\}^n$. Zbiór obiektów kodowanych jest więc zbiorem $V_1 = \{0,1,...,9\}$ cyfr dziesiętnych. Kod BCD można traktować jako kod alfanumeryczny lub kod liczbowy. Kody BCD nazywa się również w literaturze polskiej kodami dwójkowo-dziesiętnymi.

Ponieważ zależy nam na tym, żeby kod był możliwie krótki, kody BCD są z reguły funkcjami $k : \{0,1,...,9\} \to \{0,1\}^4$. Oczywiście nie możemy przyjąć tu krótszych słów kodowych niż 4 bitowe (bo funkcja k nie będzie różnowartościowa) tak więc $n \geq 4$.

Wszystkich możliwych kodów 4 bitowych BCD jest bardzo dużo bo tyle ile 10 elementowych wariacji bez powtórzeń ze zbioru 16 elementowego czyli

$$\frac{16!}{(16-10)!} = \frac{16!}{6!}$$

a różnych kodów BCD n bitowych dla $n \geq 4$ mamy tyle ile 10 elementowych wariacji bez powtórzeń ze zbioru 2^n elementowego czyli

$$\frac{2^n!}{(2^n-10)!}$$

Kodów BCD używanych w praktyce jest jednak zaledwie kilka

- kod BCD 8421 – kod naturalny, najbardziej popularny kod BCD.
- kod BCD 2421 tzw. kod Aikena (kod 4 bitowy)
- kod z nadmiarem 3 tzw. kod excess 3 (kod 4 bitowy)
- kod 2 z 5 (kod 5 bitowy)
- kod 1 z 10 (kod 10 bitowy)
- kod Graya z nadmiarem 3 (kod 4 bitowy)
- kod wskaźników cyfrowych 7 –mio segmentowych (kod 7 bitowy)

Kod BCD 8421 jest następujący

$0 \rightarrow 0000$
$1 \rightarrow 0001$
$2 \rightarrow 0010$
$3 \rightarrow 0011$
$4 \rightarrow 0100$
$5 \rightarrow 0101$
$6 \rightarrow 0110$
$7 \rightarrow 0111$
$8 \rightarrow 1000$
$9 \rightarrow 1001$

Czterobitowy kod BCD w naturalny sposób umożliwia zapisanie 2 słów kodowych BCD na jednym bajcie. Taką sytuację nazywamy ***spakowanym kodem BCD***. Czasem jednak wygodnie jest w jednym bajcie mieć tylko jedno słowo kodowe BCD na mniej znaczących bitach. Taką sytuację nazywamy ***nie spakowanym kodem BCD***. Kod BCD 8421 w postaci nie spakowanej ma tę zaletę, że słowa kodowe dają się łatwo i w naturalny sposób zastąpić odpowiednimi słowami kodowymi kodu alfanumerycznego ASCII.

Przykład: Zapiszmy w postaci spakowanej w jednym bajcie w kodzie BCD 8421 liczbę 45. Odszukujemy w tabelce definiującej kod BCD 8421 słowa kodowe odpowiadające liczbie 4 i liczbie 5, zestawiamy te słowa razem i dostajemy: 01000101 ■

Korekcja wyniku dodawania 2 cyfr zapisanych w kodzie BCD 8421. Sumator mikroprocesora z reguły nie ma specjalnego sumatora sumującego w kodzie BCD i używa do sumowania zwykłego sumatora sumującego w kodzie NKB. Dlatego też wynik dodawania 2 cyfr (zwykłego dodawania 4 bitowych słów binarnych) nie musi być wynikiem prawidłowym. Jeśli uzyskany wynik jest mniejszy równy 9 (w kodzie NKB) wynik jest prawidłowy. Jeśli jest większy od 9 lub pojawiło się przeniesienie na pozycję 4-tą (przy numeracji bitów od 0). wynik należy skorygować. Korekcja polega na dodaniu do 4 bitowego wyniku liczby 6.

Kody BCD używane są z reguły połączeniu z innymi kodami numerycznymi np. BCD i naturalny kod dziesiętny. Algorytm dodawania tak zapisanych liczb dziesiętnych kodowanych kodem BCD 8421 polega na kolejnym dodawaniu tetrad (tetrada to słowo 4 bitowe) z ewentualną korekcją wyniku.

W systemach mikroprocesorowych korekcję wyniku przy operacjach w kodzie BCD 8421 realizuje specjalny rozkaz zwykle o mnemonice (czyli symbolicznym oznaczeniu rozkazu) DAA (Decimal Addition Adjust).

Uwaga: W kodzie BCD 8421 korekcja może być „łańcuchowa” w tym sensie, że korekcja na najmniej znaczącej tetradzie może powodować korekcję w następnej, i tak dalej. Ale o potrzebie korekcji na danej pozycji dowiemy się po skorygowaniu poprzedniej.

Czas ustalania poprawnej wartości wyniku, przy dodawaniu z użyciem tego kodu, może być więc długi.

Kod BCD excess 3. Kod excess 3 uzyskujemy dodając arytmetycznie 3 (czyli 0011) do słów kodu BCD 8421. Kod BCD excess 3 jest następujący

$0 \rightarrow 0011$
$1 \rightarrow 0100$
$2 \rightarrow 0101$
$3 \rightarrow 0110$
$4 \rightarrow 0111$
$5 \rightarrow 1000$
$6 \rightarrow 1001$
$7 \rightarrow 1010$
$8 \rightarrow 1011$
$9 \rightarrow 1100$

Algorytm korekcji wyniku dodawania dla kodu excess 3 jest następujący
- Jeśli nie pojawiło się przeniesienie z danej pozycji, to odejmij 3 od wyniku.
- Jeśli pojawiło się przeniesienie z danej pozycji, to dodaj 3 do wyniku.

Algorytm korekcji jest więc sterowany jest przeniesieniem, jest to proste i wygodne, (tzn. potrzeba korekcji sygnalizowana jest przez przeniesienie). Korekcja przeprowadzana jest „jakby na boku". Maksymalne opóźnienie tego algorytmu może być znacznie mniejsze niż w przypadku algorytmu dla kodu BCD 8421. Uzasadnienie opisanego algorytmu korekcji pozostawiamy Czytelnikowi.

Uwaga: Kod BCD excess 3 to tzw. kod samouzupełniający się. Jest to bardzo użyteczna własność polegająca na tym, że negacja bitów słowa kodowego daje uzupełnienie do 9.

Kod 2 z 5 **.** Kod 2 z 5 (należy do grupy kodów BCD i kodów o stałej liczbie jedynek w słowie kodowym tzw. kodów *m* z *n*). Słowo kodowe dla kodu 2 z 5 ma długość 5. Zbiór obiektów kodowanych $V_1 = \{0,1,...,9\}$ czyli jest taki jak w każdym kodzie BCD. Alfabet kodu $V_2 = \{0,1\}$:

$0 \rightarrow 00011$
$1 \rightarrow 00101$
$2 \rightarrow 01001$
$3 \rightarrow 10001$ — 4 słowa z 1 na pozycji 5
$4 \rightarrow 00110$
$5 \rightarrow 01010$
$6 \rightarrow 10010$ — 3 słowa z 1 na pozycji 4 i 0 na pozycji 5
$7 \rightarrow 01100$
$8 \rightarrow 10100$ — 2 słowa z 1 na pozycji 3 i 0 na pozycji 5 i 4
$9 \rightarrow 11000$ — 1 słowo z 1 na pozycji 2 i 0 na pozycji 5,4 i 3

Cechy charakterystyczne kodu 2 z 5:
- długość podstawowego słowa kodowego jest zawsze równa 5
- mamy w słowie kodowym stałą równą 2, liczbie jedynek,
- kod 2 z 5 może być traktowany jako alfanumeryczny, jeśli 0,1,...,9 uważamy za symbole lub jako liczbowy, jeśli 0,1,...,9 uważamy za liczby;
- jako kod numeryczny, kod 2 z 5 jest kodem niewagowym;
- kod 2 z 5 jest kodem redundancyjnym.

Kod 1 z 10 . Dla takiego kodu mamy jak dla każdego kodu BCD $V_1 = \{0,1,2,3,...,8,9\}$

$0 \rightarrow 1000000000$
$1 \rightarrow 0100000000$
$2 \rightarrow 0010000000$
$3 \rightarrow 0001000000$
$4 \rightarrow 0000100000$
$5 \rightarrow 0000010000$
$6 \rightarrow 0000001000$
$7 \rightarrow 0000000100$
$8 \rightarrow 0000000010$
$9 \rightarrow 0000000001$

Zasada tworzenia kolejnych słów kodowych jest przesuwająca się jedynka od lewej strony do prawej. Kod 1 z 10 jest oczywiście przykładem redundancyjnego kodu BCD.

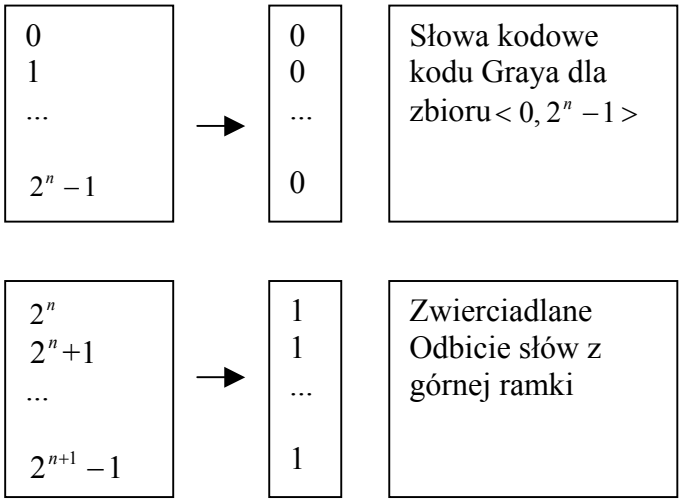
13. Kody Graya

Kody Graya (w najprostszym ujęciu) to kody numeryczne, które liczbom całkowitym ze zbioru $< 0, 2^n - 1 >$ (gdzie $n \geq 1$) przyporządkowują słowa kodowe n bitowe w taki sposób, że sąsiednie liczby (tzn. różniące się o 1 lub $2^n - 1$) mają słowa kodowe różniące się tylko na jednej pozycji. Oznacza to, że odległość Hamminga słów kodowych sąsiednich liczb jest równa 1. Można pokazać, że kod Graya jest kodem niewagowym.

Przykład: Poniżej podany jest przykład 4 bitowego kodu Graya kodującego liczby ze zbioru $< 0,15 >$

$0 \rightarrow 0000$
$1 \rightarrow 0001$
$2 \rightarrow 0011$
$3 \rightarrow 0010$
$4 \rightarrow 0110$
$5 \rightarrow 0111$
$6 \rightarrow 0101$
$7 \rightarrow 0100$
$8 \rightarrow 1100$
$9 \rightarrow 1101$
$10 \rightarrow 1111$
$11 \rightarrow 1110$
$12 \rightarrow 1010$
$13 \rightarrow 1011$
$14 \rightarrow 1001$
$15 \rightarrow 1000$ ■

Sposób tworzenia tego kodu dla zbioru $< 0, 2^{n+1} - 1 >$ jeśli już mamy kod Graya dla zbioru $< 0, 2^n - 1 >$ wyjaśnia Rys. 6. Opisany na Rys. 6. mechanizm umożliwia zdefiniowanie kodu Graya dla dowolnego zbioru liczb postaci $< 0, 2^{n+1} - 1 >$. Definicja ma charakter indukcyjny. Punktem wyjścia jest oczywiście stwierdzenie, że kod dla zbioru liczb $< 0,1 >= \{0,1\}$ zdefiniowany jako funkcja realizująca przyporządkowanie $0 \rightarrow 0$, $1 \rightarrow 1$ jest kodem Graya.



Rys. 6. Sposób tworzenia kodu Graya dla zbioru liczb $<0,2^{n+1}-1>$ jeśli mamy kod Graya dla zbioru liczb $<0,2^n-1>$.

Kody Graya stosowane są w urządzeniach do cyfrowego pomiaru położenia w urządzeniach mechanicznych np. w tarczach kodowych do pomiaru kąta określającego położenie wału obrotowego.

Uogólnieniem koncepcji kodów Graya są kody Graya na grafach. Chodzi w nich o takie przyporządkowanie słów kodowych wierzchołkom grafu by połączone gałęzią wierzchołki miały słowa kodowe różniące się tylko na jednej pozycji.

14. Kody m z n

Kody m z n są kodami numerycznymi binarnymi o stałej długości n charakteryzujące się stałą liczbą m jedynek na n pozycjach w słowie kodowym. Przy ustalonym m i n liczba takich słów kodowych jest równa $\binom{n}{m}=\frac{n!}{m!(n-m)!}$. Zaletą takich kodów jest stosunkowo łatwa kontrola poprawności słowa kodowego.

Do najprostszych kodów m z n należy kod 1 z n. W rozdziale poświęconym kodom BCD poznaliśmy również kod 2 z 5 oraz kod 1 z 10.

Kod 1 z n. Zbiór obiektów kodowanych to $V_1=\{0,1,...,n-1\}$. Alfabet kodu $V_2=\{0,1\}$. Kod 1 z n to odwzorowanie różnowartościowe $f:V_1\to V_2^n$ zadane wzorem:

$$V_1 \ni m \to \underbrace{00...0\ \underset{\vee}{1}\ 0...00}_{n} \in \{0,1\}^n \qquad m+1$$

Czyli m kodujemy za pomocą pojedynczej jedynki umieszczonej na pozycji $m+1$ licząc od lewej strony.

Przykład: Kod 1 z 8 . Dla takiego kodu mamy $V_1 = \{0,1,2,...,7\}$

$0 \rightarrow 10000000$
$1 \rightarrow 01000000$
$2 \rightarrow 00100000$
$3 \rightarrow 00010000$
$4 \rightarrow 00001000$
$5 \rightarrow 00000100$
$6 \rightarrow 00000010$
$7 \rightarrow 00000001$ ■

Zasada tworzenia kolejnych słów kodowych to przesuwająca się jedynka od lewej strony do prawej.

Uwaga: Kod 1 z n bywa definiowany bez 0 w zbiorze V_1, tzn. zbiór obiektów kodowanych ma postać $V_1 = \{1,2,...,n\}$

Uwaga: Kod 1 z n jest kodem numerycznym, redundancyjnym o stałej długości słowa równej n choć może być traktowany jako kod alfanumeryczny jeśli traktujemy elementy zbioru V_1 jako cyfry.

15. Zapisy resztowe

Zapis resztowy to inaczej zapis rezidualny lub zapis RNS (od ang. Residue Number System). Koncepcja zapisu rezidualnego (lub nieco ogólniej arytmetyki rezidualnej) pochodzi od A.Svobody i M.Valacha i została podana w pracy zamieszczonej w czeskim czasopiśmie ”Stroje na Zapracovani Informaci” w roku 1955. Niezależnie od A.Svobody i M.Valacha pomysł zapisu rezidualnego podał w 1959 r. H.L.Garner w pracy opublikowanej w IRE Transactions EC-8.

Zapis resztowy umożliwia realizację szybkiej arytmetyki (dokładniej: dodawania, odejmowania i mnożenia) bez propagacji przeniesień. Mnożenia które są dosyć złożonymi działaniami (a np. w cyfrowym przetwarzaniu sygnałów mnożeń wykonujemy bardzo dużo) można jak zobaczymy wykonywać bardzo szybko i prosto. Skomplikowane stają się natomiast w zapisie RNS algorytmy dzielenia i porównania liczb.

Symbolem $[x]_m$ lub $x \pmod m$ będziemy w dalszym ciągu oznaczać resztę z dzielenia liczby $x \in Z$ przez liczbę $m \in N,\ m \geq 2$. Potrzebne nam będzie ponadto pojęcie kongruencji. Niech $m \in N,\ m \geq 2$, mówimy, że dwie liczby $a,b \in Z$ przystają modulo m jeśli dają tę samą resztę z dzielenia przez m. Zapisujemy to pisząc $a \equiv b \pmod m$. Zapis taki nazywamy kongruencją modulo m.

Podstawą zapisu resztowego jest następujące chińskie twierdzenie o resztach.

Twierdzenie (*chińskie twierdzenie o resztach dla pierścienia liczb całkowitych Z)*
Niech liczby naturalne $m_1, m_2, ...m_n$ większe od jedności będą parami względnie pierwsze (tzn. dla każdego $i \neq j$ mamy $\mathrm{NWD}(m_i, m_j) = 1$) wówczas dla dowolnych n liczb całkowitych $x_1, x_2, ...x_n \in Z$ układ kongruencji liniowych

$$\begin{aligned} x &\equiv x_1 \ (\mathrm{mod}\, m_i) \\ x &\equiv x_2 \ (\mathrm{mod}\, m_2) \\ &.... \\ x &\equiv x_i \ (\mathrm{mod}\, m_n) \end{aligned} \tag{1}$$

ma dokładnie jedno rozwiązanie x w zbiorze $< a, a+M-1 >$, gdzie, $a \in Z$, $M = m_1 \cdot m_2 \cdot ... m_n$. Rozwiązanie to jest dane wzorem

$$x = a + \left[\sum_{i=1}^{n} k_i x_i \right]_M \tag{2}$$

gdzie k_i dla i=1,2,3,...n są liczbami całkowitymi spełniającymi warunek

$$\begin{aligned} k_i &\equiv 1 \,(\mathrm{mod}\, m_i) \text{ dla } i=1,2,3...,n \\ k_i &\equiv 0 \,(\mathrm{mod}\, m_j) \text{ dla } i \neq j \end{aligned} \tag{3}$$

Dowód: Możemy znaleźć n liczb $c_1, c_2, \ldots c_n$ takich, że $c_i \in Z_{m_i}$ oraz $c_i \otimes_{m_i} \left[\frac{M}{m_i} \right]_{m_i} = 1$ (lub równoważnie c_i jest elementem odwrotnym do elementu $\left[\frac{M}{m_i} \right]_{m_i}$ w pierścieniu Z_{m_i} czyli $c_i = \left[\frac{M}{m_i} \right]_{m_i}^{-1}$). Łatwo sprawdzić, że $k_i = c_i \frac{M}{m_i}$ i=1,2,3,...,n spełniają warunek (**). Wynika to z tego, że $NWD(m_i, \left[\frac{M}{m_i} \right]_{m_i}) = 1$ ■

Uwaga: Oczywiście rozwiązaniem układu kongruencji (*) będzie również każda taka liczba całkowita y dla której mamy $x \equiv y \,(\mathrm{mod}\ m_1 \cdot m_2 \cdot ... m_n)$ zbiór $\{x + kM; k \in Z\}$ jest jednocześnie zbiorem wszystkich rozwiązań układu kongruencji (*).

Uwaga 2: Łatwo można sprawdzić, że $\sum_{i=1}^{n} k_i \equiv 1 \,(\mathrm{mod}\, M)$

Niech będzie dana liczba $x \in Z$ i niech liczby naturalne $m_1, m_2, ... m_n$ większe od jedności będą parami względnie pierwsze. Zapisem resztowym liczby x nazywamy ciąg skończony

$$([x]_{m_1}, [x]_{m_2}, ..., [x]_{m_n}) \tag{4}$$

W ten sposób definiujemy kod numeryczny $f :< 0, m_1 \cdot m_2 \cdot ... \cdot m_n - 1 > \rightarrow Z_{m_1} \times Z_{m_2} \times ... \times Z_{m_n}$. Zakres tak zdefiniowanego zapisu RNS jest oczywiście równy $< 0, m_1 \cdot m_2 \cdot ... \cdot m_n - 1 >$.

Przykład: Wybieramy moduły takie: $m_1 = 3$ $m_2 = 5$ $m_3 = 7$ $m_4 = 8$. Możemy wówczas reprezentować 840 liczb (ponieważ $m_1 m_2 m_3 m_4 = 840$). Wszystkie 4 współrzędne dają się zapisać za pomocą 3 bitowego słowa co daje w sumie 11 bitów. Widać pewną redundancję w stosunku do naturalnego zapisu binarnego NKB (10 bitów). ■

Dodawanie, odejmowanie i mnożenie liczb w zapisie resztowym jest wykonywane po współrzędnych. Dokładniej, jeśli mamy 2 liczby zapisane w zapisie resztowym $a = ([a]_{m_1},[a]_{m_2},...,[a]_{m_n}) = (a_1,a_2,...,a_n)$ i $b = ([b]_{m_1},[b]_{m_2},...,[b]_{m_n}) = (b_1,b_2,...,b_n)$ to wówczas (jeśli wynik działania należy do zakresu zapisu) algorytmy dodawania, odejmowania i mnożenia dane są wzorami

$$a+b = (b_1,b_2,...,b_n)+(b_1,b_2,...,b_n) = (a_1 \oplus_{m_1} b_1, a_2 \oplus_{m_2} b_2,..., a_n \oplus_{m_n} b_n) \tag{5}$$

$$a-b = (b_1,b_2,...,b_n)-(b_1,b_2,...,b_n) = (a_1 -_{m_1} b_1, a_2 -_{m_2} b_2,..., a_n -_{m_n} b_n) \tag{6}$$

$$a \cdot b = (b_1,b_2,...,b_n)\cdot(b_1,b_2,...,b_n) = (a_1 \otimes_{m_1} b_1, a_2 \otimes_{m_2} b_2,..., a_n \otimes_{m_n} b_n) \tag{7}$$

gdzie $a_i \oplus_{m_i} b_i$, $a_i -_{m_i} b_i$, $a_i \otimes_{m_i} b_i$ są odpowiednio dodawaniem, odejmowaniem i mnożeniem modulo m_i.

Przykład: Przyjmijmy $m_1 = 3$ $m_2 = 5$ $m_3 = 7$ i niech $x = 3$, $y = 6$. Stosując zapis resztowy mamy $x = (0,3,3)$, $y = (0,1,6)$.

Dodając liczby $x = (0,3,3)$, $y = (0,1,6)$ w zapisie resztowym według zaproponowanego wyżej algorytmu mamy

$$x + y = (0,3,3)+(0,1,6) = (0 \oplus_{m_1} 0, 3 \oplus_{m_2} 1, 3 \oplus_{m_n} 6) = (0,4,2)$$

Zauważmy, że $x + y = 9$ i zapisując 9 w zapisie resztowym mamy (0,4,2) a więc z zaproponowanego algorytmu otrzymaliśmy wynik prawidłowy.

Odejmując liczby $y = (0,1,6)$ i $x = (0,3,3)$, w zapisie resztowym według zaproponowanego wyżej algorytmu mamy

$$y - x = (0,1,6)-(0,3,3) = (0 -_{m_1} 0,\ 1 -_{m_2} 3,\ 6 -_{m_n} 3) = (0,3,3)$$

Zauważmy, że $y - x = 3$ i zapisując 3 w zapisie resztowym mamy (0,3,3) a więc z zaproponowanego algorytmu otrzymaliśmy wynik prawidłowy.

Mnożąc liczby $y = (0,1,6)$ i $x = (0,3,3)$, w zapisie resztowym według zaproponowanego wyżej algorytmu mamy

$$x \cdot y = (0,1,6)\cdot(0,3,3) = (0 \otimes_{m_1} 0,\ 1 \otimes_{m_2} 3,\ 6 \otimes_{m_n} 3) = (0,3,4)$$

Zauważmy, że $x \cdot y = 18$ i zapisując 18 w zapisie resztowym mamy (0,3,4) a więc z zaproponowanego algorytmu otrzymaliśmy wynik prawidłowy. ■

Konwersja zapis NKB liczby $x \rightarrow$ na zapis RNS liczby *x* sprowadza się do obliczenia *n* reszt z dzielenia liczby *x* przez moduły m_1, m_2, $m_3,...,m_n$.

Konwersja zapis RNS $\rightarrow$ zapis NKB jest nieco bardziej złożona. Podamy dwa algorytmy konwersji.

Podstawowy algorytm konwersji jest na ogół formułowany w samym chińskim twierdzeniu o resztach. Przypomnijmy algorytm:

Niech $(a_1, a_2, ..., a_n)$ będzie zapisem *RNS* liczby x oraz $M = m_1 \cdot m_2 \cdot ... m_n$.

I. Znajdujemy liczby $b_j \in Z$ takie, że:

$$b_j \cdot \frac{M}{m_j} \equiv 1 \pmod{m_j} \text{ (dla } j=1,2,...,n). \tag{8}$$

II. Jeśli $x \in [0, M]$, to

$$x = [a_1 \cdot b_1 \cdot \frac{M}{m_1} + a_2 \cdot b_2 \cdot \frac{M}{m_2} + ... + a_n \cdot b_n \cdot \frac{M}{m_n}]_M \ , \tag{9}$$

a ogólnie, jeśli $x \in [a, a+m)$, to

$$x = a + (a_1 \cdot b_1 \cdot \frac{M}{m_1} + a_2 \cdot b_2 \cdot \frac{M}{m_2} + ... + a_n \cdot b_n \cdot \frac{M}{m_n} - a) \pmod{M}. \tag{10}$$

Przykład: Niech $m_1 = 3, m_2 = 5, m_3 = 7$ oraz niech $x = (0,3,3)$. Wyznaczamy stałe b_1, b_2, b_3 spełniające warunek (8) otrzymując $b_1 = 2$, $b_2 = 1$, $b_3 = 1$. Korzystając ze wzoru (9) otrzymujemy $x = 3$. ■

Konwersja wykorzystująca „mixed radix representation" (algorytm Garnera).

I. Niech $(a_1, a_2, ..., a_n)$ będzie zapisem *RNS* liczby x. Definiujemy $\binom{n}{2}$ stałych c_{ij} (liczymy je tylko jeden raz – wstępnie).

$$c_{ij} \cdot m_i \equiv 1 \pmod{m_j} \text{ dla } 1 \le i < j \le n, \tag{11}$$

lub co na jedno wychodzi, poszukujemy odwrotności elementu $[m_i]_{m_j}$ w pierścieniu Z_{m_j}, czyli takiego c_{ij}, że $c_{ij} \otimes_{m_j} m_i = 1$ w pierścieniu Z_{mj}.

II. Obliczamy ciąg $v_1, v_2, ..., v_n$ gdzie $v_i \in [0, m_i - 1)$ będący zapisem mixed radix szukanej liczby x,

$$\begin{aligned}
v_1 &:= a_1 \pmod{m_1} \\
v_2 &:= (a_2 - v_1)c_{12} \pmod{m_2} \\
v_3 &:= ((a_3 - v_1)c_{13} - v_2)c_{23} \pmod{m_3} \\
&\ \ \vdots \\
v_r &:= (...(u_n - v_1)c_{1r} - v_2)c_{2n} - ... - v_{r-1})c_{r-1,n} \pmod{m_n}
\end{aligned} \tag{12}$$

III. Obliczamy

$$x = v_n \cdot m_{n-1} \cdot \ldots \cdot m_1 + v_{n-1} \cdot m_{n-2} \cdot \ldots \cdot m_1 + \ldots + v_3 \cdot m_2 \cdot m_1 + v_2 \cdot m_1 + v_1 \qquad (13)$$

Łatwo zauważyć, że w zapisie *RNS* złożoność konwersji z $RNS \rightarrow$ zapis binarny; zapis binarny $\rightarrow RNS$ w pewnym stopniu zależy od wyboru modułów m_1, m_2, m_3,..., m_n.

Dla modułów postaci $2^e - 1$ konwersja liczby *x* na jej zapis *RNS* jest bardzo prosta.

Ogólnie rzecz biorąc, jeśli mamy liczbę $x \in Z$ otrzymujemy jej reprezentację resztową przez dzielenie *x* przez kolejne moduły m_i, co daje ciąg $(a_1, a_2, ..., a_r)$. Jednak dla modułów postaci $2^{e_j} - 1$, można zrobić tak:

Liczba *x* zapisuje się na pewnej liczbie bitów. Grupując po e_j bitów od najmniej znaczącego bitu począwszy, możemy zapisać liczbę *x* w postaci:

$$x = a_t A^t + a_{t-1} A^{t-1} + \ldots + a_1 A + a_0 \qquad (14)$$

gdzie $A = 2^{e_j}$ i $0 \le a_k < 2^{e_j}$ dla każdego $k = 0,1,...,t$.

Zatem:

$$u = a_t + a_{t-1} + \ldots + a_1 + a_0 \pmod{2^{e_j} - 1} \qquad (15)$$

ponieważ $A \equiv 1 \pmod{2^{e_j} - 1}$ i ostatecznie:

$$u_i = a_t \oplus a_{t-1} \oplus \ldots \oplus a_0 \qquad (16)$$

$$u_i = a_t + a_{t-1} + \ldots + a_0 \pmod{(2^{e_j} - 1)}. \qquad (17)$$

Zajmijmy się modułami postaci $m_j = 2^{e_j} - 1$, gdzie $e_j \in N, e_j \ge 2$, dokładniej. Pokażemy jakie korzyści wynikają z wyboru modułów $m_1, m_2, \ldots m_r$ tak, by było $m_j = 2^{e_j} - 1$, $e_j \in N, e_j \ge 2$. Zależy nam na tym, żeby były to moduły parami względnie pierwsze. Następujące twierdzenie stanowi kryterium na to, by dwie liczby postaci $2^f - 1$ i $2^e - 1$ były względnie pierwsze.

Twierdzenie: Dla $e, f \in N$ mamy $NWD(2^e - 1), (2^f - 1) = 2^{NWD(e,f)} - 1$.

Dowód: 1. Zauważmy, że zachodzi równość:

$$((2^e - 1) \pmod{2^f - 1}) = 2^{e \pmod f} - 1 \qquad (18)$$

Jeśli $e \le f$, to jest to oczywiste. Niech *e>f*. Równość (18) jest równoważna do kongruencji

$$(2^e - 1) \equiv 2^{e \bmod f} - 1 \pmod{(2^f - 1)} \qquad (19)$$

i dalej

$$2^e \equiv 2^{e \bmod f} \qquad (\mathrm{mod}(2^f - 1)) \qquad (20)$$

$$2^{e(\mathrm{mod}\, f)} 2^f \ldots q^f \equiv 2^{e \bmod f} \qquad (\mathrm{mod}(2^f - 1)) \qquad (21)$$

ale: $2^f \equiv 1 \quad (\mathrm{mod}(2^f - 1))$ zatem prawdziwa jest równość (18).

2. Wracamy do dowodzonej równości. Niech $e>f$ (dla $e=f$ równość jest oczywista) wówczas na mocy (18) i algorytmu Euklidesa $NWD(2^e - 1, 2^f - 1) = NWD(2^f - 1, 2^{e \bmod f} - 1)$.

Kolejno stosujemy wzór (18), czyli obliczamy resztę z dzielenia przez liczbę mniejszą, zauważmy przy tym, że $NWD(e, f) = NWD(f, e(\mathrm{mod}\, f))$.

Postępując zgodnie z algorytmem Euklidesa otrzymamy wreszcie, w kolejnym kroku, iloraz reszt =0.

Będziemy więc mieli: $NWD(2^e - 1, 2^f - 1) = NWD(2^a - 1, 2^0 - 1) = 2^a - 1$

oraz $NWD(e, f) = NWD(a, 0) = a$ ■

Wniosek: Liczby $2^e - 1$ i $2^f - 1$ są względnie pierwsze wtedy i tylko wtedy, gdy *e* i *f* są względnie pierwsze.

Przykład: Możemy wybrać jako moduły np.: $m_1 = 2^{35} - 1$, $m_2 = 2^{34} - 1$, $m_3 = 2^{33} - 1$, $m_4 = 2^{31} - 1$, $m_5 = 2^{29} - 1$. Jest to bardzo wygodne jeśli mamy słowo maszynowe 35 bitowe. Możemy w tej sytuacji reprezentować liczby z zakresu od 0 do $m_1 \cdot m_2 \cdot \ldots \cdot m_5 > 2^{161}$. ■

Słowo maszynowe *k*-bitowe, ogólnie rzecz biorąc może być jednak takie, że $2^k > m_i$ lub $2^k < m_i$. Jeśli moduły dają się zapisać w kodzie NKB na słowie maszynowym to jest to oczywiście wygodne, ale nie jest to niezbędne. ■

Zasadnicze znaczenie dla szybkości wykonywania działań arytmetycznych w zapisie resztowym mają sposoby wykonywania działań modulo *m*. Działania modulo *m* powinny być realizowane za pomocą odpowiednio zaprojektowanych układów.

Uogólnieniem zapisu RNS na liczby zespolone jest zapis resztowy QRNS (Quadratic Residue Numer System)

Uwaga: Istotę zapisu resztowego w języku algebry można wyrazić tak. Pierścień ilorazowy $Z/(m_1 m_2 \ldots m_n Z)$ i suma prosta pierścieni ilorazowych $(Z/m_1 Z) \oplus (Z/m_2 Z) \oplus \ldots \oplus (Z/m_n Z)$ są izomorficzne tzn.

$$Z/(m_1 m_2 \ldots m_n Z) \cong (Z/m_1 Z) \oplus (Z/m_2 Z) \oplus \ldots \oplus (Z/m_n Z)$$

i izomorfizm zadany jest wzorem $\gamma(x) = ([x]_{m_1}, [x]_{m_2}, \ldots, [x]_{m_n})$. Jest to nieco inne sformułowanie chińskiego twierdzenia o resztach.

16. Kody wagowe ze zmienną wagą

Kody wagowe ze zmienną wagą lub zapisy ze zmienną podstawą (ang Mixed Radix Notation lub Mixed Radix Representation) to kody liczbowe, zasadniczo służące do reprezentowania (czyli zapisu) liczb ze zbioru $N \cup \{0\}$.

Przypomnijmy podstawowe twierdzenie umożliwiające konstruowanie kodów wagowych. Jest to jednocześnie zasadnicze twierdzenie, na którym opiera się definicja kodów wagowych ze zmienną wagą.

Twierdzenie: Dla dowolnej liczby $m \in N$ i ustalonego ciągu $(m_i)_{i=0}^{\infty}$, gdzie $m_i \in \{2,3,...\}$ istnieje dokładnie jeden ciąg skończony $a_k, a_{k-1},..., a_1, a_0$ taki, że

1) dla każdego $i = 0,1,...,k$, $a_i \in \{0,1...m_i - 1\}$, $a_k \neq 0$

2) $$m = a_k m_{k-1} m_{k-2} \cdot ... \cdot m_0 + a_{k-1} m_{k-2} m_{k-3} \cdot ... \cdot m_0 + ... a_2 m_1 m_0 + a_1 m_0 + a_0$$

Oznaczmy $A_i = \{0,1...m_i - 1\}$ dla $i = 0,1,2,...$ i niech $A = \bigcup_{i=0}^{\infty} A_i$ oraz

$$B = A_0 \cup A_0 \times A_1 \cup A_0 \times A_1 \times A_2 \cup ... \cup A_0 \times A_1 \times A_2 \times ... \times A_k \cup ...$$

Odwzorowanie $K : N \cup \{0\} \ni m \to K(m) = a_k a_{k-1} ... a_1 a_0 \in B$, (gdzie $a_i \in A_i$) określone na zbiorze $N \cup \{0\}$ i zdefiniowane w powyższym twierdzeniu nazywamy *kodem wagowym ze zmienną wagą* lub ***zapisem wagowym ze zmienną wagą*** (ang. mixed radix notation) przy czym przyjmujemy $K(0) = 0$. Ta przejrzysta i prosta definicja wymaga jednak kilku uwag.

Elementy zbioru $A = \bigcup_{i=0}^{\infty} A_i$ nazywamy cyframi i często utożsamiamy je, czy wiążemy z pewnymi symbolami np. pisanymi na papierze. Raz więc możemy na element ze zbioru $A = \bigcup_{i=0}^{\infty} A_i$ patrzeć jak na liczbę raz jak na symbol. Jest to w praktyce dosyć wygodne. Oczywiście byłoby lepiej gdyby $A = \bigcup_{i=0}^{\infty} A_i$ było zbiorem skończonym. Posługiwanie się nieskończonym zbiorem symboli jest bowiem wysoce niepraktyczne.

Zauważmy, że jeśli ciąg $(m_i)_{i=0}^{\infty}$ jest ograniczony to $A = \bigcup_{i=0}^{\infty} A_i$ jest zbiorem skończonym a więc alfabetem. Ponadto istnieje takie k, że $m_i \leq m_k$ dla każdego $i = 0,1,2,...$ oraz $A_i \subset A_k = A$ dla każdego $i = 0,1,2,...$. Istnieje więc taki alfabet A, że odwzorowanie K przyjmuje wartości z A^* a więc *kod wagowy ze zmienną wagą* jest kodem w sensie przyjętej definicji kodu.

Jeśli ciąg $(m_i)_{i=0}^{\infty}$ nie jest ograniczony to $A = \bigcup_{i=0}^{\infty} A_i = N \cup \{0\}$ a więc A nie jest zbiorem skończonym i formalnie rzecz biorąc *zapis wagowy K ze zmienną wagą* nie jest kodem

w sensie przyjętej definicji. Jeśli jednak użyć jakiegokolwiek pomocniczego kodu numerycznego $K' : N \cup \{0\} \to V^*$ do zapisywania liczb z ze zbioru $A = \bigcup_{i=0}^{\infty} A_i = N \cup \{0\}$, gdzie V jest jakimkolwiek alfabetem (np. $K' : N \cup \{0\} \to V^*$ może być zwykłym zapisem dziesiętnym znanym ze szkoły) to łatwo odwzorowanie $K : N \cup \{0\} \ni m \to K(m) = a_k a_{k-1} ... a_1 a_0 \in B$ tak zmodyfikować by uzyskać odwzorowanie będące kodem. Wystarczy przyjąć:

$$\widetilde{K} : N \cup \{0\} \ni m \to \widetilde{K}(m) = K'(a_k), K'(a_{k-1}), ..., K'(a_1), K'(a_0) \in (V \cup \{,\})^*$$

Przykład: Niech $m_i = p_i \pmod{10}$, gdzie $p_i, p_2, ...$ są kolejnymi liczbami pierwszymi, wówczas 21 w zapisie ze zmienną wagą oznacza liczbę 5 (w zapisie dziesiętnym). Oczywiście alfabet $A = \{0,1,2,3,4,5,6,7,8,9\}$ ■

Uwaga: W oczywisty sposób zapisy ze zmienną wagą jest uogólnieniem naturalnych zapisów wagowych. Wystarczy przyjąć $m_i = W$ dla i=1,2,...

Podobnie jak ma to miejsce w zapisie wagowym z ustaloną wagą W, możemy w zapisie ze zmienną wagą reprezentować również liczby ułamkowe „wprowadzając kropkę”.

Zapis ze zmienną wagą jest wykorzystywany w jednym z algorytmów konwersji zapisu RNS na zapis wagowy.