

Temat 5. Kodowanie liczb

Spis treści do tematu 5

5.1. Kodowanie liczb stałopozycyjnych

5.1.1. Naturalny kod binarny NKB

5.1.2. Kod dwójkowo-dziesiętny BCD

5.1.3. Kod Graya

5.1.4. Kod znak-moduł

5.1.5. Kod uzupełnień do dwóch U2

5.2. Kodowanie liczb zmiennopozycyjnych

5.2.1. Ogólne zasady kodowania liczb zmiennoprzecinkowych

5.2.2. Kodowanie liczb zmiennoprzecinkowych wg. normy
IEEE 754

5.2.3. Niestandardowe typy zmiennoprzecinkowe

5.3. Kolejność zapisu bajtów liczb wielobajtowych

- *little-endian*

- *big-endian*

5.4. Literatura

5.1. Kodowanie liczb stałopozycyjnych

5.1.1. Naturalny kod binarny NKB (ang. *natural binary code*)

n -bitowe słowo $X = \langle x_{n-1}, x_{n-2}, \dots, x_1, x_0 \rangle$ reprezentuje w NKB liczbę całkowitą nieujemną o wartości

$$x = \sum_{i=0}^{n-1} 2^i x_i \quad (5.1)$$

Konwersja kodu NKB na kod dziesiętny

Konwersja wykonywana przez ludzi

Przykład: liczba 00101001 w NKB odpowiada w kodzie dziesiętnym

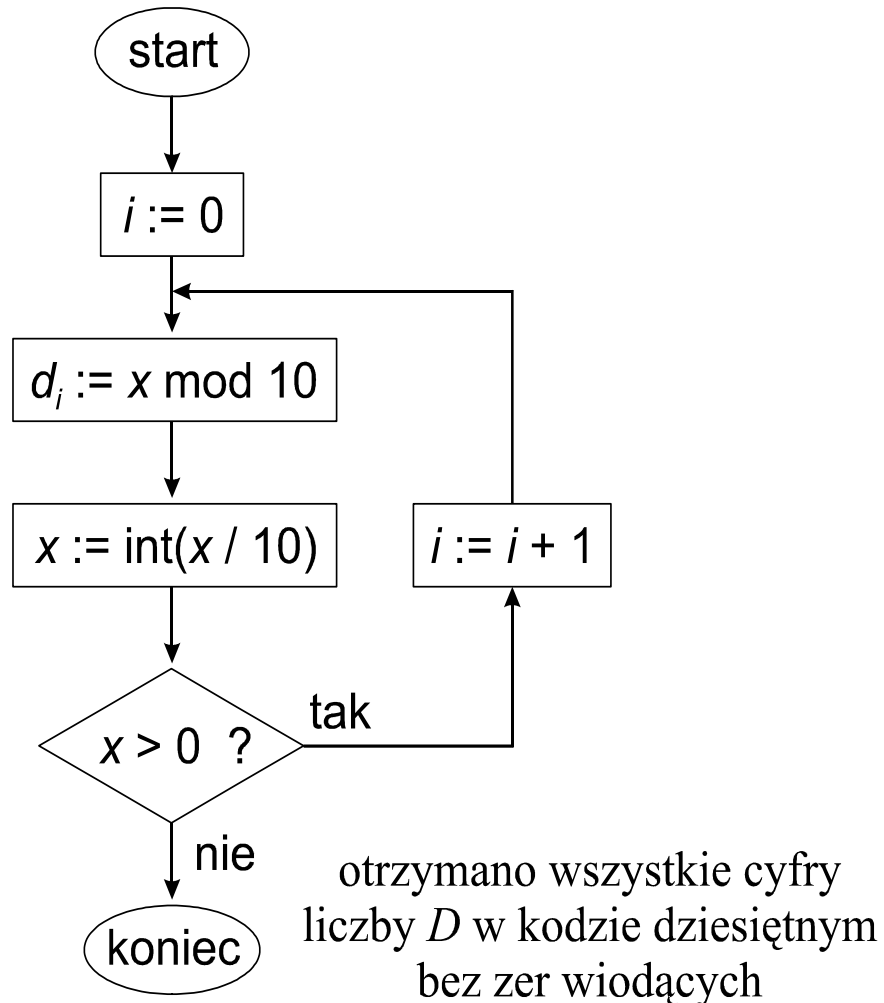
$$x = 2^7 \cdot 0 + 2^6 \cdot 0 + 2^5 \cdot 1 + 2^4 \cdot 0 + 2^3 \cdot 1 + 2^2 \cdot 0 + 2^1 \cdot 0 + 2^0 \cdot 1 = 32 + 8 + 1 = 41$$

Konwersja kodu NKB na kod dziesiętny

Konwersja w programie komputerowym

Kolejne cyfry w kodzie dziesiętnym $D = \langle d_{n-1}, d_{n-2}, \dots, d_1, d_0 \rangle$ znajdujemy wg. algorytmu

dana liczba x w NKB



Przykład obliczeń:

- 1) dana $x := 00101001_2 = 41_{10}$
- 2) $d_0 := 41_{10} \bmod 10_{10}$ (wynik 1_{10})
- 3) $x := \text{int}(41_{10}/10_{10})$ (wynik 4_{10})
- 4) $d_1 := 4_{10} \bmod 10_{10}$ (wynik 4_{10})
- 5) $x := \text{int}(4_{10}/10_{10})$ (wynik 0_{10})
- 6) koniec, bo $x = 0$

Otrzymano słowo cyfr dziesiętnych $\langle d_1, d_0 \rangle = \langle 4, 1 \rangle$

Rys. 5.1. Algorytm konwersji liczby całkowitej w NKB na liczbę w systemie dziesiętnym.

Konwersja kodu dziesiętnego na NKB

Konwersja wykonywana przez ludzi

Konwersja liczby przedstawionej w kodzie dziesiętnym na NKD przez ludzi przywykłych do używania tylko kodu dziesiętnego wymaga iteracyjnego powtarzania dzielenia przez dwa oraz znajdowania reszty z tego dzielenia.

Przykład: przedstawić liczbę 37_{10} w NKB

- 1) $37 / 2 = 18$ i reszta z dzielenia 1 $\rightarrow x_0 = 1$
- 2) $18 / 2 = 9$ bez reszty z dzielenia $\rightarrow x_1 = 0$
- 3) $9 / 2 = 4$ i reszta z dzielenia 1 $\rightarrow x_2 = 1$
- 4) $4 / 2 = 2$ bez reszty z dzielenia $\rightarrow x_3 = 0$
- 5) $2 / 2 = 1$ bez reszty z dzielenia $\rightarrow x_4 = 0$
- 6) $1 / 2 = 0$ i reszta z dzielenia 1 $\rightarrow x_5 = 1$
- 7) koniec gdy zerowy wynik dzielenia.

Wynik konwersji: $100101_2 = 37_{10}$

Konwersja w programie komputerowym

W programach komputerowych kod NKB jest bezpośrednio stosowany do kodowania zmiennych całkowitych.

Jeżeli dane są kolejne cyfry d_0, d_1, \dots, d_n liczby w kodzie dziesiętnym, to wartość w NKD obliczamy wprost ze wzoru:

$$x = \sum_{i=0}^{n-1} 10^i d_i \quad (5.2)$$

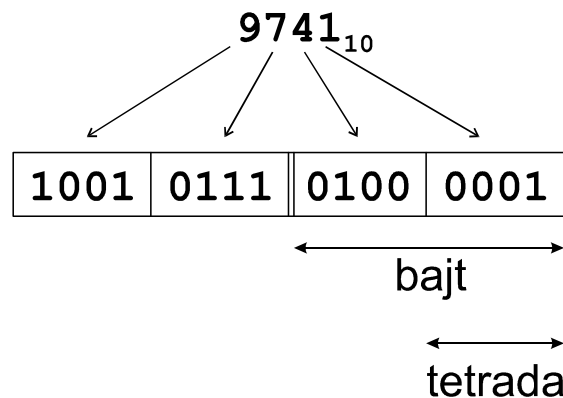
Znaczenie znajomości reguł konwersji pomiędzy kodem dziesiętnym i NKB

Procedury konwersji dostosowane do specyfiki konkretnego zadania są zazwyczaj krótsze i szybsze od procedur z bibliotek standardowych.

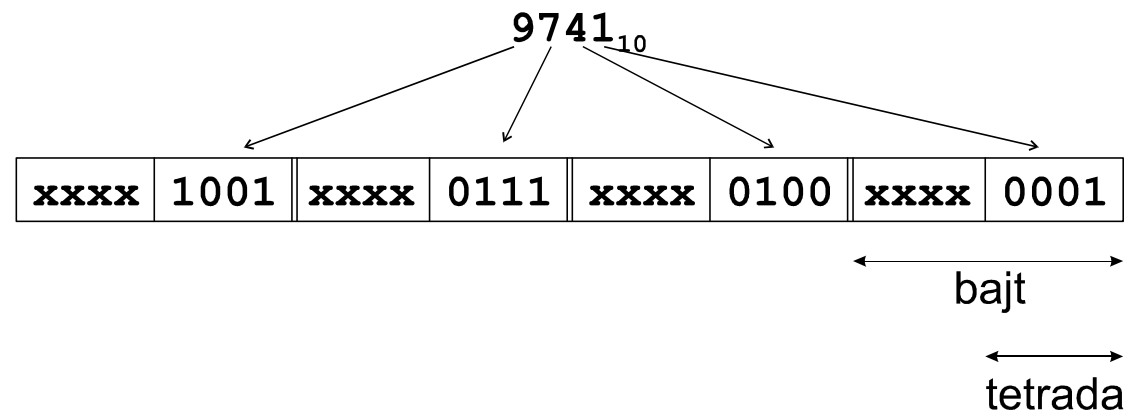
5.1.2. Kod dwójkowo-dziesiętny BCD (ang. *binary coded decimal*)

Jedna cyfra dziesiętna jest kodowana przy użyciu grupy czterech bitów zwanych **tetrada** (ang. *nibble*).

upakowany kod BCD 8421
(ang. *packed BCD*)



nieupakowany kod BCD
(ang. *unpacked BCD*)



Rys. 5.2. Format liczb w upakowanym i nieupakowanym kodzie BCD.

Szczególnym przypadkiem nieupakowanego kodu BCD jest liczba całkowita przedstawiona jako tekst ASCII, gdzie w miejscu „xxxx” musi wystąpić 0011_2 . Kod ASCII znaku drukarskiego 0 wynosi $48_{10} = 00110000_2$; kolejnym cyfrom przypisano kolejne kody.

5.1.3. Kod Graya

Każde dwie kolejne liczby zapisane w kodzie Graya różnią się dokładnie jednym bitem.

Ważniejsze zastosowania kodu Graya:

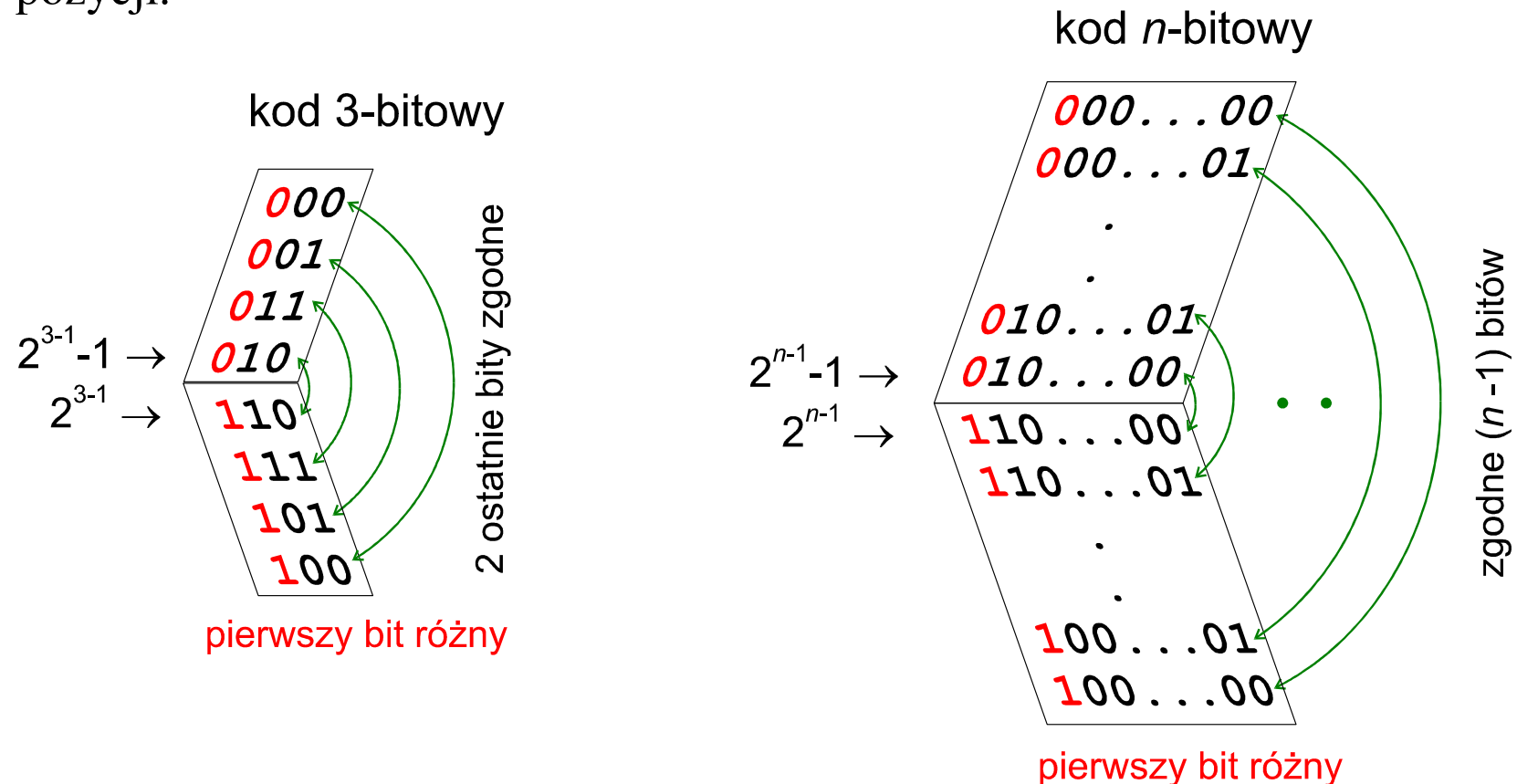
- Kod Graya został opracowany w celu zwiększenia niezawodności układów kontroli i sterowania przez eliminację jednoczesnych przełączeń na liniach wielobitowych, na których występują przejścia przez kolejne wartości.
- Obecnie kod Graya (wersja BRGC) jest często wykorzystywany do syntezy i minimalizacji wyrażeń logicznych, np. w graficznej metodzie tablic Karnaugh.

Dla danej długości słowa (liczby bitów) istnieje wiele możliwych kodów Graya.

Rys. 5.3. Przykład różnych możliwości zapisu kolejnej liczby w kodzie Graya gdy określony jest zapis liczb poprzednich.

dziesiętnie	kod Graya
0	0000
1	0001
2	0011
3	0010
4	0110
5	0100 ? 0111

Zazwyczaj przez kod Graya rozumie się kod BRGC (ang. *binary-reflected Gray code*). Istnieje tylko jeden kod BRGC. Pierwsze dwie liczby w kodzie BRGC są kodowane jako 0 i 1, a kolejne liczby są tworzone przez odbicie lustrzane tablicy wcześniej wygenerowanych liczb i rozszerzenie o 1 bit na pierwszej pozycji.



Rys. 5.4. Wykorzystanie zgodności bitów w dwóch kolejnych grupach 2^{n-1} liczb do prostego generowania liczb w n -bitowym kodzie BRGC.

5.1.4. Kod znak-moduł (ang. *sign-magnitude*)

Liczba całkowita w kodzie znak-moduł jest zdefiniowana jako zespół



$$x = (-1)^S \cdot M, \quad (5.3)$$

S - bit znaku, M - moduł w NKB

gdzie: S - znak (ang. *sign*),
M - moduł (ang. *magnitude*).

Rys. 5.5. Format liczby 8 bitowej w kodzie znak-moduł.

znak-moduł	dziesiętnie
01111111	127 ← maksimum
...	
00000001	1
00000000	+0 dwa sposoby
10000000	-0 kodowania zera
10000001	-1
...	
11111111	-127 ← minimum

Zastosowanie:

W standardzie kodowania liczb zmiennoprzecinkowych IEEE 754 znak liczby oraz moduł mantysy są kodowane niezależnie i razem można je rozważać jako mantysę ze znakiem w kodzie znak-moduł.

Tabela 5.1. Charakterystyczne wartości w kodzie znak-moduł dla liczby 8-bitowej. T5-8

5.1.5. Kod uzupełnień do dwóch U2 (ang. *two's complement*)

Liczby w kodzie U2 są zapisywane następująco:

➤ liczby dodatnie i zero:

kodowane jak w NKB (bez użycia najbardziej znaczącego bitu = 0)

➤ liczby ujemne wg zależności:

$$-X = \overline{X} + 1 \quad (5.4)$$

gdzie: $-X$ jest negacją arytmetyczną X , \overline{X} jest negacją bitową X .

kod U2		dziesiętnie
11111001 ₂	=	-7 ₁₀
↓ negacja bitów		
00000110 ₂	=	+6 ₁₀
↓ +1		
00000111 ₂	=	+7 ₁₀

Procedura obliczania liczby $-X$ w kodzie U2:

- 1). Zanegować wszystkie bity liczby X .
- 2). Dodać 1.

Rys. 5.6. Przykład obliczania liczby przeciwnej do -7 zapisanej w kodzie U2.

znak-moduł	dziesiętnie
01111111	127 ← maksimum
...	
00000001	1
00000000	0
11111111	-1
...	
10000000	-128 ← minimum

Tabela 5.2. Charakterystyczne wartości w kodzie U2 dla liczby 8-bitowej.

Jeżeli X jest kodowane przy użyciu n bitów, to zachodzi związek

$$2^n - \text{NKB}(X) = \text{NKB}(-X), \quad (5.5)$$

gdzie $\text{NKB}(\dots)$ – rzutowanie (konwersja) bez ingerencji w bity.

binarnie	dziesiętnie
1 00000000	$= 2^8 = 256_{10}$
<u>– 11111101</u>	$\leftarrow \text{NKB}(-3_{10} \text{ w U2}) = 253_{10}$
00000011	$= 3_{10}$

Rys. 5.7. Przykład obliczania liczby przeciwnej do -3 przy wykorzystaniu właściwości uzupełniania $-X$ oraz $\text{NKB}(X)$ do 2^n w n -bitowym kodzie U2.

Zastosowanie:

Kod U2 jest powszechnie wykorzystywany do reprezentowania liczb całkowitych ze znakiem w systemach komputerowych i mikrokontrolerach. Ten sam układ logiczny umożliwia poprawne sumowanie, odejmowanie i mnożenie w kodach NKB i U2, także w przypadku mieszanych argumentów w NKB oraz U2.

	binarnie	dziesiętnie
NKB →	10000110	= +134 ₁₀
U2 →	10000000	= -128 ₁₀
+	<hr/>	
	X 00000110	= +6 ₁₀

Rys. 5.8. Przykład poprawnej operacji arytmetycznej na dwóch argumentach wyrażonych w kodach NKB oraz U2. Uzyskany wynik jest prawidłowy zarówno w kodzie NKB jak i U2.

bez przeniesienia arytm.

Przykład wykorzystania właściwości kodu U2 do optymalizacji programu generowanego przez kompilatory języków C/C++.

```
signed int suma(signed int a1, unsigned int a2) {  
    /* poprawne wykonanie tego sumowania nie wymaga  
    konwersji argumentów do typu danych o szerszym zakresie */  
    return a1 + a2;  
}
```

```
signed int a = -7;  
unsigned int b = 3;  
signed int w = suma(a, b);
```

5.2. Kodowanie liczb zmiennopozycyjnych

5.2.1. Ogólne zasady kodowania liczb zmiennopozycyjnych

Liczba zmiennopozycyjna (zmiennoprzecinkowa) jest zdefiniowana jako zespół

$$x = (-1)^S \cdot M \cdot 2^E, \quad (5.6)$$

gdzie: S – znak (ang. *sign*),
 M – mantysa (ang. *mantissa*),
 E – wykładnik (ang. *exponent*).

Wykładnik zazwyczaj zapisuje się jako przesuniętą liczbę bez znaku (ang. *biased exponent*)

$$E = E_b - B, \quad (5.7)$$

gdzie: E – wartość wykładnika,
 E_b – zapis w pamięci jako liczba NKB bez znaku z przesunięciem,
 B – przesunięcie (ang. *bias*) zależne od zakresu wartości E_b .

Liczby znormalizowane (ang. *normalized numbers*)

Z wielu możliwych reprezentacji danej liczby zmiennoprzecinkowej wybiera się taką która zawiera najwięcej cyfr znaczących w mantysie (po pominięciu zer nieznaczących z lewej strony mantysy). W liczbie znormalizowanej najbardziej znacząca cyfra (bit przy kodowaniu binarnym) mantysy zawsze $\neq 0$.

Liczby zdenormalizowane (ang. *denormal numbers*)

Liczby w których nastąpiła utrata precyzji mantysy. Najbardziej znacząca cyfra (bit przy kodowaniu binarnym) mantysy = 0.

Przykład:

Założmy, że w formularzu możemy zapisać tylko 5 cyfr mantysy, 2 cyfry wykładnika oraz znaki +/- dla mantysy i wykładnika. Normalizacja wyniku działania na liczbach znormalizowanych może być niewykonalna przy powyższych ograniczeniach, np.

$$3.2003 \cdot 10^{-99} / 1.0 \cdot 10^{+1} = ? \quad (\text{brak miejsca na zapis } 3.2003 \cdot 10^{-100})$$

Dwie możliwości zapisu wyników bliskich zero, których nie da się znormalizować:

1) zaokrąglenie do zera, czyli raptowny niedomiar (ang. *abrupt underflow*), np.

$$3.2003 \cdot 10^{-99} / 1.0 \cdot 10^{+1} = 0,$$

2) użycie liczby zdenormalizowanej, stopniowy niedomiar (ang. *gradual underflow*)

$$3.2003 \cdot 10^{-99} / 1.0 \cdot 10^{+1} = 0.3200 \cdot 10^{-99}.$$

5.2.2. Kodowanie liczb zmiennoprzecinkowych wg. normy IEEE 754

Formaty liczb zmiennoprzecinkowych określone w normie IEEE 754 są obecnie powszechnie wykorzystywane zarówno w systemach komputerowych jak i w mikrokontrolerach.

Podstawowe założenia przyjęte w normie IEEE 754:

- 1). Mantysa składa się z jednobitowej części całkowitej i wielobitowej części ułamkowej.
 - Mantysa liczb znormalizowanych $1 \leq M < 2$,
 - mantysa liczb zdenormalizowanych $M < 1$.
- 2). Liczby są przedstawiane jako znormalizowane zawsze gdy jest to możliwe. Tylko bardzo małe liczby są przedstawiane jako liczby zdenormalizowane.
- 3). Odróżnia się następujące typy liczb i symboli:
 - liczba 0: $E_b = E_{\min}$; $M = 0$; $S = 0$ albo 1 (odróżniane zero ze znakiem + albo -),
 - liczby zdenormalizowane: $E_b = E_{\min}$; $0 < M < 1.0$,
 - liczby znormalizowane: $E_{\min} < E_b < E_{\max}$,
 - $\pm\infty$: $E_b = E_{\max}$; $M = 0$; $S = 0$ albo 1,
 - NaN - nieliczba (ang. not a number): $E_b = E_{\max}$; $M \neq 0$; S – nieistotne.

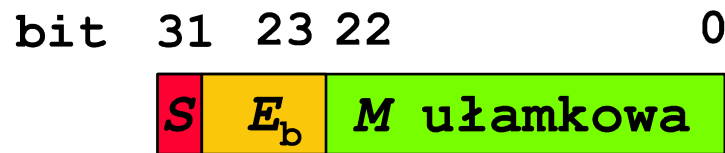
Norma IEEE 754 precyzuje dwa typy zmiennoprzecinkowe:

1). Pojedyncza precyzja (ang. *float, single precision*)

Wartość liczby znormalizowanej wyznacza się ze wzoru:

$$x = (-1)^S \cdot M \cdot 2^{(Eb - 127)} \tag{5.8}$$

Liczba cyfr znaczących w zapisie dziesiętnym: 6...7.



Rys. 5.9. Format liczb zmiennoprzecinkowych IEEE 754 pojedynczej precyzji, gdzie: *S* - znak, *M* - mantysa w NKB, *E_B* - wykładnik bez znaku w NKB z przesunięciem.

2). Podwójna precyzja (ang. *double precision*)

Wartość liczby znormalizowanej wyznacza się ze wzoru:

$$x = (-1)^S \cdot M \cdot 2^{(Eb - 1023)}. \tag{5.9}$$

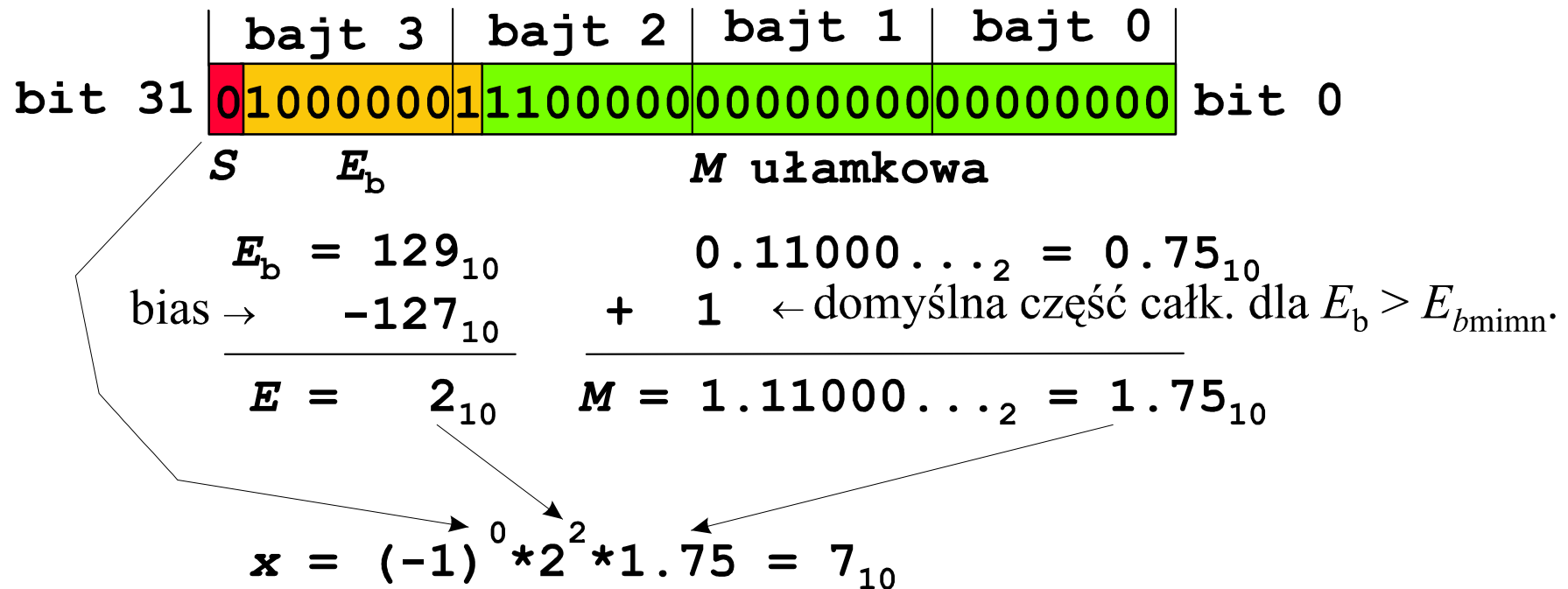
Liczba cyfr znaczących w zapisie dziesiętnym: 15...16.



Rys. 5.10. Format liczb zmiennoprzecinkowych IEEE 754 podwójnej precyzji.

Uwaga: w formatach IEEE 754 bit części całkowitej mantysy nie jest jawnie przechowywany – wartość tego bitu jest odtwarzana na podstawie wykładnika E_b

- $M = 0.$ (część ułamkowa M) dla $E_b = E_{\min} (0)$; $E = 1 - B$,
- $M = 1.$ (część ułamkowa M) dla $E_b > E_{\min} (0)$; $E = E_b - B$.



Rys. 5.11. Przykład kodowania liczby o wartości 7_{10} jako liczby zmiennoprzecinkowej pojedynczej precyzji zgodnej z IEEE 754.

Tabela 5.3. Reprezentacja binarna liczb zmiennoprzecinkowych w pojedynczej i podwójnej precyzji.

liczba lub symbol	reprezentacja binarna			poza pamięcią
	S	E_b	M - część ułamkowa	M - część całkowita
0	0/1	000...000	000...000	0
liczby zdenorm.	0/1	000...000	111...111	0
		
		000...000	000...001	
liczby znorm.	0/1	111...110	111...111	1
		
		000...001	000...000	
$\pm\infty$	0/1	111...111	000...000	1
NaN	0/1	111...111	111...111	1
			...	
			000...001	

Ponadto w nowszych wydaniach normy oraz koprocessorów *x87* odróżnia się nieliczby pasywne (ang. *quiet NaN*) oraz nieliczby aktywne (ang. *signaling NaN*). Nieliczby pasywne mogą być bezpiecznie używane jako argumenty operacji, natomiast użycie nieliczby aktywnej powoduje zgłoszenie wyjątku niedozwolonej operacji.

W językach C/C++ od wersji C99 i C++11 zdefiniowano funkcje do odróżniania przypadków szczególnych od poprawnych liczb zmiennopozycyjnych:

```
int isfinite(x); // sprawdza czy argument jest liczbą skończoną i nie jest nie liczbą
```

```
int isinf(x); // sprawdza czy argument jest nieskończony
```

```
int isnan(x); // sprawdza czy argument jest nie liczbą
```

```
int isnormal(x); // sprawdza czy argument jest liczbą znormalizowaną,
```

przy czym funkcje te istnieją w trzech wariantach dla argumentów typu: float, double i long double.

Przykłady:

```
isnan(0.0 / 0.0); // zwraca wynik 1 (true)
```

```
isnormal(DBL_MIN / 2); // zwraca wynik 0 (false)
```

```
isinf(exp(800)); // zwraca wynik 1 (true)
```

5.2.3. Niestandardowe typy zmiennoprzecinkowe

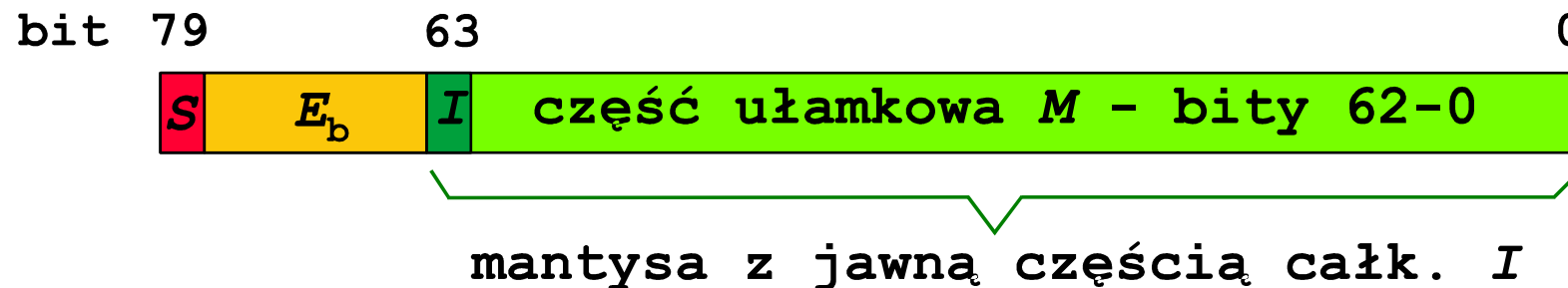
Przykład 1

Wewnętrzny format *extended precision* rejestrów koprocesora x87 - można używać także w pamięci operacyjnej jako liczbę 80-bitową (1 bit znaku, 15 bitów wykładnika, 64 bitów mantysy).

Wartość liczby wyznacza się ze wzoru:

$$x = (-1)^S \cdot (I.M) \cdot 2^{(Eb - 16383)}. \quad (5.9)$$

Liczba cyfr znaczących w zapisie dziesiętnym: 18...19.



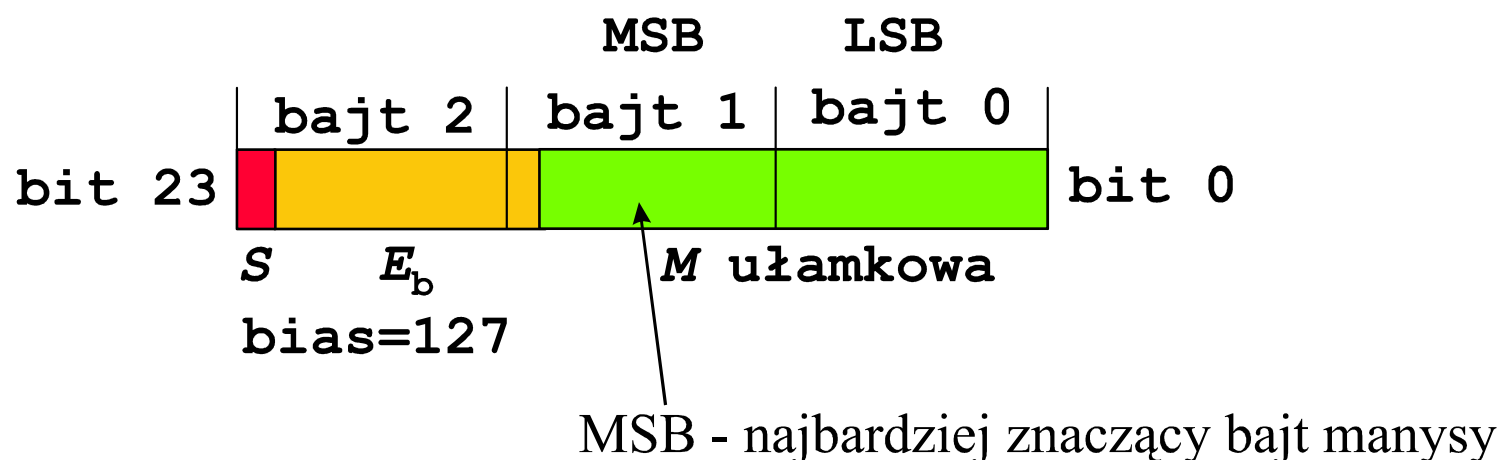
Rys. 5.12. Wewnętrzny format liczb zmiennoprzecinkowych w koprocesorach x87. Oznaczenia: *S* - znak, *EB* - wykładnik bez znaku w NKB z przesunięciem, **jawny bit *I* części całkowitej mantysy**, *M* – część ułamekowa mantysa w NKB.

Reguły rozróżniania liczb znormalizowanych, zdenormalizowanych, 0, $\pm\infty$, oraz NaN, są w znacznym stopniu zgodne z normą IEEE754. Jawny bit *I* powoduje odróżnianie liczb nieznormalizowanych, które nie są tożsame z liczbami zdenormalizowanymi.

Przykład 2

Zmodyfikowany typ float IEEE 754 w kompilatorze HI-Tech C for the PIC 10/12/16 MCU Family.

Ten typ danych skrócono do 24-bitów przez obcięcie najmniej znaczących bitów mantysy.

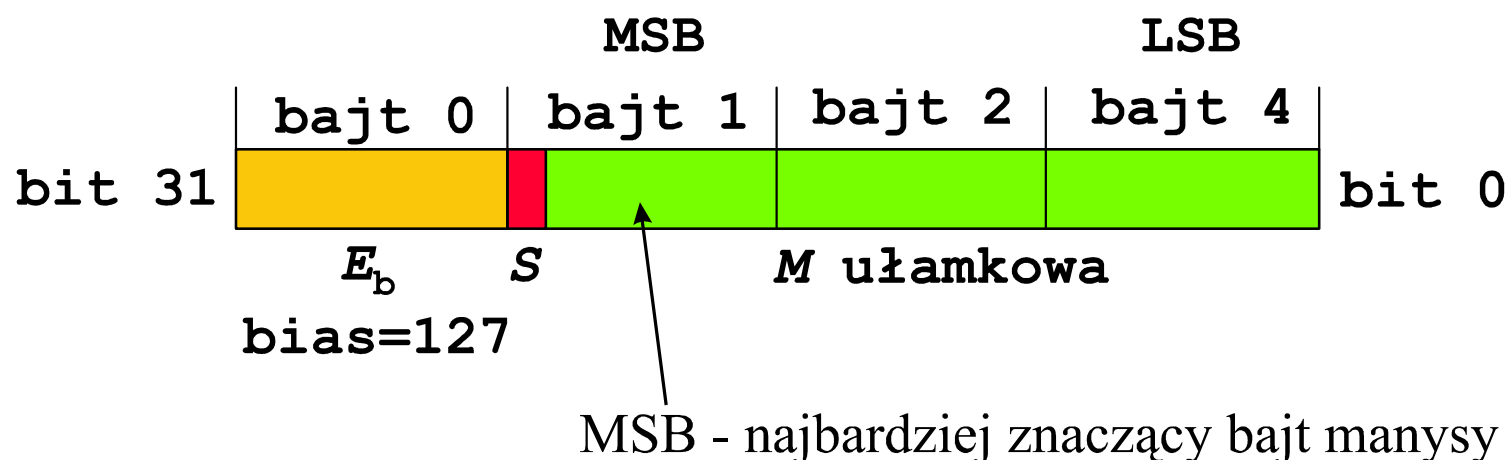


Rys. 5.13. Format 24-bitowych (1 bit znaku, 8 bitów wykładnika, 15 bitów mantysy) liczb zmiennoprzecinkowych opracowany przez skrócenie formatu IEEE 754 pojedynczej precyzji.

W kompilatorze HI-Tech C for the PIC 10/12/16 MCU Family dostępny jest także 32-bitowy typ danych zgodny z IEEE 754.

Przykład 2

Typ float w CCS C Compiler for PIC 12/14/16/18. Zachowano daleko idącą zgodność z IEEE 754 z wyjątkiem przesunięć składników.



Rys. 5.14. Niestandardowy format 32-bitowych liczb zmiennoprzecinkowych pojedynczej precyzji. Przesunięcie wykładnika E oraz bitu znaku S , wprowadzono w celu przyspieszenia obliczeń w 8-bitowej jednostce arytmetyczno-logicznej.

5.3. Kolejność zapisu bajtów liczb wielobajtowych

Kolejność poszczególnych bajtów liczb wielobitowych przechowywanych w pamięci operacyjnej systemu mikroprocesorowego zależy od budowy jednostki centralnej CPU/MPU. W praktyce występują najczęściej trzy główne przypadki:

- *little-endian* (Intel x86, x86-64, VAX)
najmniej znaczący bajt pod najniższym adresem,
- *big-endian* (Motorolla 680xx)
najmniej znaczący bajt pod najwyższym adresem,
- *bi-endian* (Alpha, ARM, Intel Itanium, MIPS, PowerPC, ARM od wersji 3)
sprzętowe wsparcie dla obu kolejności *little-endian* i *bi-endian*.

W prostych systemach 8-bitowych kolejność jest określona przez oprogramowanie, np.

- ❖ w kompilatorze HI-Tech C for the PIC 10/12/16 MCU Family
little-endian – wszystkie typy całkowite i zmiennoprzecinkowe,
- ❖ w CCS C Compiler for PIC 12/14/16/18
little-endian – wszystkie typy całkowitoliczbowe,
big-endian – typ zmiennoprzecinkowy float.

Różnice w kolejności zapisu bajtów utrudniają przenoszenie między systemami zarówno plików źródłowych oprogramowania jak i danych zapisanych w postaci binarnej.

Przykład nieprzenośnego źródła programu w języku C

```
/* wersja dla little-endian */
union {
    unsigned short val;
    unsigned char bajty[2];
} uval;

/* pobierz mniej znaczący bajt */
uval.bajty[0] = pomiar_lo();

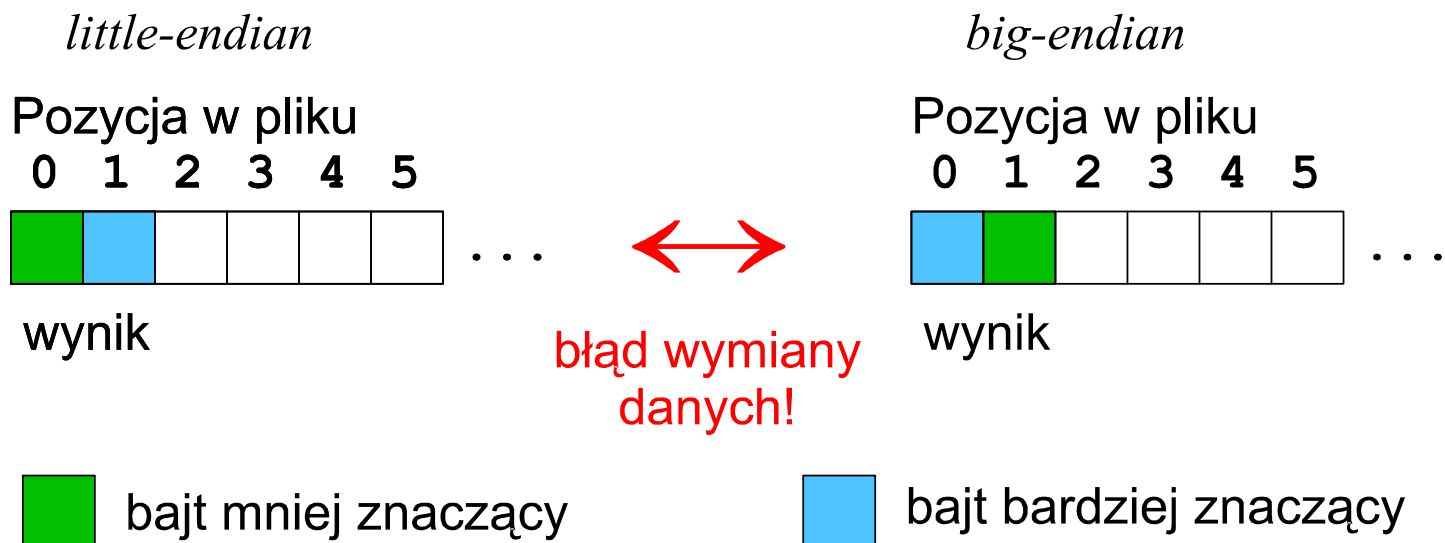
/* pobierz bardziej znaczący bajt */
uval.bajty[1] = pomiar_hi();
obliczenia(uval.val);
```

```
/* wersja dla big-endian */
union {
    unsigned short val;
    unsigned char bajty[2];
} uval;

/* pobierz mniej znaczący bajt */
uval.bajty[1] = pomiar_lo();

/* pobierz bardziej znaczący bajt */
uval.bajty[0] = pomiar_hi();
obliczenia(uval.val);
```

Przykład niezgodności na poziomie wymiany danych w plikach binarnych.



Rys. 5.15. Ryzyko błędnej wymiany danych w plikach między systemami *little-endian* oraz *big-endian*. Oba pliki zostały zapisane przy użyciu programów identycznych na poziomie kodu źródłowego.

Tekst źródłowy programu do przykładu na rys. 2.15.

```
unsigned short wynik = pobierz_pomiar();

/* Otwórz plik binarny do zapisu */
FILE * f = fopen("C:\\pomiar\\wyniki1.bin", "wb");

fwrite(&wynik, sizeof(wynik), 1, f);
...
```


Przykładowe środki zaradcze na problemy z kolejnością zapisu bajtów:

1. Wymiana danych poprzez pliki tekstowe ASCII.
2. Zdefiniowanie jednego formatu pliku i przygotowanie osobnych wersji programu dla systemów *little-endian* i *big-endian*.
3. Przygotowanie jednej wersji programu dla maszyny wirtualnej. W Javie dane są zapisywane do plików binarnych w kolejności *big-endian* niezależnie od sprzętu. Otrzymanie innej kolejności wymaga operacji jawnie zapisanych w programie.

5.4. Literatura

- [1] H. Kamionka-Mikuła, H. Małysiak, B. Pochopień, *Synteza i analiza układów cyfrowych*, Wydawnictwo Pracowni Komputerowej Jacka Skamierskiego, Gliwice 2006.
- [2] M. Tuszyński, R. Goczyński, *Koprocesory 80287, 80387 oraz i486*, Komputerowa Oficyna Wydawnicza „Help”, Warszawa 1992.
- [3] P. Prinz, T. Crawford, *Język C w pigułce*, APN Promise, Warszawa 2016.
- [4] P. Misiurewicz, *Układy automatyki cyfrowej*, Wydawnictwa Szkolne i Pedagogiczne, Warszawa, 1984.
- [5] A. Barczak, J. Florek, T. Sydoruk, *Elektroniczne techniki cyfrowe*, VIZJA PRESS&IT Sp. z o.o., Warszawa 2006.
- [6] A. Skorupski, *Podstawy techniki cyfrowej*, WKiŁ, Warszawa 2004.
- [7] Dokumentacja kompilatorów *HI-Tech C for the PIC 10/12/16 MCU* oraz *CCS C Compiler for PIC 12/14/16/18* (w zainstalowanych kompilatorach).