Janusz Biernat
UNIWERSYTET ZIELONOGÓRSKI, INSTYTUT INFORMATYKI I ELEKTRONIKI

Tajniki arytmetyki komputerów

dr hab. inż. Janusz BIERNAT

Autor monografii: Architektura układów arytmetyki resztowej, Arytmetyka komputerów, i podręczników akademickich: Metody i układy arytmetyki komputerowej, Architektura komputerów. Zainteresowania naukowe: projektowanie niezawodnych układów cyfrowych, algorytmy arytmetyki resztowej, kody korekcyjne, kryptoanaliza.

e-mail: janusz.biernat@pwr.wroc.pl

Streszczenie

Z zasad arytmetyki rozszerzeń nieskończonych wywiedziono definicję uzupełnieniowej reprezentacji liczb i reguły działań arytmetycznych w systemie uzupełnieniowym. Pokazano związek arytmetyki komputerowej z arytmetyką systemów uzupełnieniowych. Wskazano cechy dwójkowego systemu uzupełnieniowego użyteczne w konstrukcji algorytmów arytmetycznych. Na gruncie standardu zmiennoprzecinkowego IEEE 754 przeprowadzono dyskusję problemów arytmetyki zmiennoprzecinkowej. Opisano architekturę szybkich układów arytmetycznych. Omówiono zasady arytmetyki resztowej i koncepcję systemu resztowego RNS. Przywołując podstawowe twierdzenia wykorzystywane w arytmetyce resztowej, podano wnioski wynikające z własności reprezentacji resztowych, użyteczne w systemie RNS.

Abstract

From the concept of arithmetic of infinite extensions the definition of radix-complement representation and the rules of arithmetic operations in radix-complement system are derived. The relation between computer arithmetic and the arithmetic of radix-complement system is proved. The features of 2's complement system useful in design of arithmetic algorithms are described. On the base of the standard IEEE 754 basic problems of floating-point arithmetic are discussed. The architecture of fast arithmetic circuits is presented. The concept of residue arithmetic and residue number system, RNS is presented. The basic theorems applicable in residue arithmetic are recalled and some practical conclusions concerning operations in RNS are given.

Słowa kluczowe: arytmetyka uzupełnieniowa, szybki sumator, arytmetyka zmiennoprzecinkowa, system resztowy, arytmetyka resztowa

Keywords: radix-complement arithmetic, fast adder, floating-point arithmetic, residue number system, residue arithmetic

1. System uzupełnieniowy

Wynalazek zera umożliwił systematyczną notację dowolnych liczb naturalnych w postaci uporządkowanego ciągu znaków zwanych cyframi. Każdej pozycji z_i w ciągu jest jednoznacznie przypisany mnożnik, zwany wagą, który jest naturalną potęgą stałej naturalnej, zwanej podstawą (*radix*). Wartość liczby naturalnej o reprezentacji $\{\dots\ z_4\ z_3\ z_2\ z_1\ z_0\}$ może być obliczona jako:

$$Z = \dots + z_4\beta^4 + z_3\beta^3 + z_2\beta^2 + z_1\beta^1 + z_0 = \sum_{i=0}^{\infty} z_i\beta^i \qquad (1)$$

W systemach standardowych wartość cyfry musi liczbą naturalną mniejszą od podstawy: $z_i \in \{0,1,\dots\beta-1\}$. Ponieważ wartością pozycji, na której występuje cyfra 0 jest zero, więc dla oszczędności zapisu pomija się zwykle lewostronne wiodące zera i stosuje zapis skrócony: zamiast $(0)\,z_4\ z_3\ z_2\ z_1\ z_0$, gdzie $(0)=\dots0\dots00$ oznacza ciąg zer o dowolnej długości, piszemy $z_4\ z_3\ z_2\ z_1\ z_0$.

Systematyczność zapisu umożliwia jego rozszerzenie na dowolne liczby dodatnie przez dopuszczenie ujemnych potęg podstawy i użycie przecinka pozycyjnego (ang. *radix point*), umieszczanego bezpośrednio za pozycją o wadze $1=\beta^0$. Stosownie do tej konwencji wartością liczby o reprezentacji $\{\dots\ z_3\ z_2\ z_1\ z_0\ z_{-1}\ z_{-2}\ \dots\}$ jest

$$Z = \dots + z_2\beta^2 + z_1\beta^1 + z_0 + z_{-1}\beta^{-1} + z_{-2}\beta^{-2} \dots = \sum_{i=-\infty}^{\infty} z_i\beta^i \ . \qquad (2)$$

W arytmetyce rozszerzeń nieskończonych pełną reprezentacją pozycyjną liczby jest (pozycja przecinka musi być oznaczona):
$\{(0)\ \dots\ z_3\ z_2\ z_1\ z_0\ z_{-1}\ z_{-2}\ z_{-3}\ \dots\}$ – dla liczb niewymiernych,
$\{(0)\ \dots\ z_3\ z_2\ z_1\ z_0\ \dots(z_{-i}\ z_{-i-1}\ z_{-i-p})\}$ – dla liczb okresowych,
$\{(0)\ \dots\ z_3\ z_2\ z_1\ z_0\ \ z_{-1}\ z_{-2}\dots\ z_{-m}\ (0)\}$ – dla liczb skończonych.

Tak jak w przypadku liczb naturalnych, wiodące lewostronne zera są pomijane w zapisie liczby. Każdą liczbę skończoną można też przedstawić jako iloczyn liczby naturalnej i skali β^{-m}.

W praktyce posługujemy się zwykle reprezentacjami skończonymi – dokładność takiej reprezentacji jest równa wadze pozycji najmniej znaczącej $ulp=\beta^{-m}$. Przy założeniu jednakowej dokładności wszystkich liczb systemu, analizę działań arytmetycznych można wtedy ograniczyć do liczb całkowitych, pamiętając jednak o odpowiednim przeskalowaniu wyniku.

Oprócz systematyczności, istotną zaletą reprezentacji pozycyjnych jest prostota działań arytmetycznych, zwłaszcza dodawania i odejmowania, które na każdej pozycji *i* opisuje równość

$$x_i \pm y_i \pm c_i = \mp\beta c_{i+1} + s_i \,, \qquad (3)$$

gdzie $x_i, y_i, s_i \in \{0,1,\dots,\beta-1\}$ oraz $c_i, c_{i+1} \in \{0,1\}$.

W systemie zapisu pozycyjnego reprezentację liczby naturalnej większej lub mniejszej o 1 od liczby danej można wytworzyć wykonując dodawanie lub odejmowanie jedności od liczby danej. Rozszerzając tę zasadę na liczby mniejsze od zera stwierdzimy, że odjęcie 1 od zera zgodnie z (3) da jako wynik nieskończony ciąg cyfr $\{\beta-1\ \dots\ \beta-1\ \dots\ \beta-1\ \beta-1\}$, reprezentujący liczbę ujemną –1. Konsekwentnie, ciąg $\{\beta-1\ \dots\ \beta-1\ \dots\ \beta-1\ \beta-2\}$ jest reprezentacją liczby –2 itd. Zgodnie z taką regułą, reprezentacją liczby przeciwnej do danej jest wynik jej pozycyjnego odejmowania od zera.

Reprezentacją liczby przeciwnej do $\{(0)\ z_{k-1}\ \dots\ z_2\ z_1\ z_0\}$ jest więc liczba $\{(\beta-1)\ v_{k-1}\ \dots\ v_2\ v_1\ v_0\}$ taka, że

$$\{(0)\ z_{k-1}\ \dots\ z_2\ z_1\ z_0\}+\{(\beta-1)\ v_{k-1}\ \dots\ v_2\ v_1\ v_0\}=\{(0)\} \qquad (4)$$

Taki system reprezentacji liczb nazywa się systemem uzupełnieniowym (ang. *radix-complement*). Obowiązują w nim reguły dodawania i odejmowania takie, jak w systemach naturalnych (3).

Powstaje pytanie, czy wartość liczby $\{(\beta-1)\ v_{k-1}\ \dots\ v_2\ v_1\ v_0\}$ można obliczyć według zależności takiej jak (1). Zauważmy, że liczbę tę można przedstawić jako sumę liczb $\{(0)\ v_{k-1}\ \dots\ v_2\ v_1\ v_0\}$ i $\{(\beta-1)\ 0_{k-1}\ \dots\ 0_0\}$ o wartości $-\beta^k$. Gdyby dopuścić użycie cyfry o wartości –1, to liczbę $\{(\beta-1)\ v_{k-1}\ \dots\ v_2\ v_1\ v_0\}$ można by zapisać alternatywnie jako $\{(0)\ [-1]\ v_{k-1}\ \dots\ v_2\ v_1\ v_0\}$ o wartości:

$$V = -\beta^k + v_{k-1}\beta^{k-1} \dots + v_2\beta^2 + v_1\beta^1 + v_0 \,. \qquad (5)$$

Arytmetyka komputerów, z powodu ograniczeń sprzętowych musi być arytmetyką ograniczonego zakresu, gdzie wszystkie liczby całkowite są reprezentowane przez ciągi cyfr o skończonej długości *k*. Konieczne jest też ustalenie, które ciągi reprezentują liczby dodatnie, a które liczby ujemne. Konsekwencją naturalnego założenia o symetrii zakresu liczb dodatnich i ujemnych jest przyjęcie konwencji powiązania znaku liczby z wartością cyfry wiodącej: reprezentacje liczb dodatnich rozpoczynają się od cyfr o małej wartości (0,1,…), a reprezentacje liczb ujemnych od cyfr o dużej wartości (β–1,β–2,…). W systemach o podstawie nieparzystej przyjęcie tej konwencji prowadzi do znacznej asymetrii zakresu, z drugiej strony wymuszenie symetrii reprezentacji pociąga za sobą znaczny wzrost złożoności działań.

Z tego powodu praktyczne znaczenie mają tylko systemy o podstawie parzystej, gdzie cyframi rozszerzenia nieskończonego są

$$z_e = \begin{cases} 0 & gdy \quad 2z_{k-1} < \beta, \\ \beta - 1 & gdy \quad 2z_{k-1} \geq \beta, \end{cases} \qquad (6)$$

a skończonej reprezentacji liczby całkowitej $\{z_{k-1} \ldots z_2\ z_1\ z_0\}$ odpowiada reprezentacja nieskończona $\{(z_e)\ z_{k-1} \ldots z_2\ z_1\ z_0\}$. Wartość tej liczby można obliczyć jako:

$$Z = |\, z_e \,|\, \beta^k + z_{k-1}\beta^{k-1} \ldots + z_2\beta^2 + z_1\beta^1 + z_0 \,. \qquad (7)$$

gdzie $|z_e|=-1$ gdy $z_e=\beta-1$ albo 0 w przeciwnym razie ($z_e=0$).

W arytmetyce ograniczonego zakresu suma lub różnica jest tworzona w takim rozmiarze jak rozmiary argumentów i niezbędne jest sprawdzenie, czy wynik nie wykracza poza dozwolony zakres. Ponieważ wynik w rozszerzonym zakresie jest zawsze poprawny, więc najprostszy sposób weryfikacji to sprawdzenie czy wartość wytworzona zgodnie z (3) na pozycji rozszerzenia s_k jest wartością cyfry rozszerzenia dla pozycji najwyższej s_{k-1}.

Równość wartości liczb o reprezentacji $\{(\beta-1)\ v_{k-1} \ldots v_2\ v_1\ v_0\}$ oraz $\{(0)\ [-1]\ v_{k-1} \ldots v_2\ v_1\ v_0\}$ pozwala łatwo adaptować algorytmy mnożenia i dzielenia liczb dodatnich do działań na reprezentacjach uzupełnieniowych.

W mnożeniu na wszystkich pozycjach, na których następuje akumulacja skalowanych iloczynów częściowych muszą być użyte cyfry rozszerzenia. Jeśli mnożnik jest ujemny, to dodatkowym iloczynem jest uzupełnienie mnożnej (ciąg $(\beta-1)$ ma wartość -1).

Warunkiem poprawności procedury dzielenia jest takie skalowanie dzielnej względem dzielnika, aby pierwsza reszta była bezwzględnie mniejsza od dzielnika. Zapewnia to istnienie rozwiązania parametrycznego równania dzielenia:

$$r_{i+1} = \beta\, r_i - q_i\, D,\ |\, r_{i+1} \,| < |\, D \,|\,, \qquad (8)$$

gdzie D jest wartością dzielnika, r_i wartością reszty, a q_i wartością cyfry ilorazu odpowiadającej reszcie r_i.

Ponieważ kolejne cyfry ilorazu, oprócz pierwszej, mają zawsze wartości dodatnie, więc znak każdej reszty musi taki jak znak dzielnika. Jeśli więc znak dzielnej jest różny od znaku dzielnika, w pierwszym kroku należy wykonać dodawanie dzielnika do tak przeskalowanej dzielnej, aby uzyskać resztę o znaku przeciwnym do dzielnika. Wartością pierwszej cyfry ilorazu jest w takim przypadku -1 reprezentowane przez ciąg cyfr rozszerzenia $(\beta-1)$.

2. Dwójkowy system uzupełnieniowy U2

W dwójkowym systemie uzupełnieniowym (2'*s complement*) najwyższa cyfra argumentu jest jednocześnie cyfrą rozszerzenia. Dzięki temu możliwe jest znaczne uproszczenie algorytmów podstawowych działań arytmetycznych.

Wykrycie przekroczenia zakresu można wykonać przez sprawdzenie zgodności przeniesień: z pozycji najwyższej c_k i na tę pozycję c_{k-1}. Jeśli te przeniesienia są jednakowe, to wytworzone zgodnie z (3) wartości bitów sumy na pozycji najwyższej s_{k-1} i na pozycji rozszerzenia s_k są identyczne.

Mnożenie w systemie U2 można uprościć zauważając, że ponieważ $z_e=z_{k-1}$ oraz $|z_e|=-z_{k-1}$, więc wartość każdej liczby w tym systemie o reprezentacji $\{z_{k-1} \ldots z_2\ z_1\ z_0\}$ można przedstawić jako liczby dodatniej Z^* o postaci $\{1-z_{k-1} \ldots z_2\ z_1\ z_0\}$ i stałej 2^{k-1}:

$$Z = -2^{k-1} + (1 - z_{k-1})2^{k-1} + z_{k-2}2^{k-2} + \ldots + z_1 2^1 + z_0 \qquad (9)$$

Ponieważ kolejne iloczyny częściowe są liczbami w systemie U2, więc zamiast ich zwykłej akumulacji, opisanej wyżej, można wykonać akumulację liczb dodatnich (rozszerzeniach zerowych), a uzyskany wynik skorygować przez dodanie stałej $-2^{k+m-1}+2^{m-1}$, gdzie k jest liczbą bitów mnożnika, a m – liczbą bitów mnożnej. Jeśli bowiem 2^iA_i jest i-tym przeskalowanym iloczynem częściowym (zależnie od wartości cyfry z_i mnożnika A_i jest równe mnożnej A, jej uzupełnieniu $-A$, lub zeru), to cały iloczyn jest równy

$$\sum_{i=0}^{k-1} A_i 2^i = \sum_{i=0}^{k-1} (A_i^* - 2^{m-1}) = \sum_{i=0}^{k-1} A_i^* - 2^{k+m-1} + 2^{m-1} \qquad (10)$$

Możliwe jest też uproszczenie dzielenia. Ponieważ cyfrą ilorazu może być tylko 0 lub 1, więc zamiast sprawdzać warunek (8) można, zależnie od zgodności znaku bieżącej reszty ze znakiem dzielnika, dodawać lub odejmować dzielnik od przeskalowanej reszty bieżącej. Jeśli w wyniku odejmowania $2r_i-D$ otrzymamy resztę o znaku przeciwnym do dzielnika, to możliwa jest korekcja tej reszty po skalowaniu przez dodanie dzielnika, bo

$$2 \cdot (2r_i - D) + D = 2 \cdot 2r_i - D \,. \qquad (11)$$

Jeśli znaki reszty i dzielnika są zgodne, wykonuje się odejmowanie dzielnika. Takie dzielenie nazywa się nieodtwarzającym.

3. Arytmetyka zmiennoprzecinkowa

Wygodną formą zapisu bardzo dużych i bardzo małych liczb jest tzw. notacja naukowa lub inżynierska, polegająca na zapisie wartości liczby w postaci $\pm M_*\beta^E$, przy tym wykładnik E jest liczbą całkowitą. Aby wykluczyć różne reprezentacje tej samej wartości, konieczne jest ograniczenie dozwolonych wartości mnożnika kolejnymi potęgami podstawy β: $\beta^p \le M < \beta^{p+1}$.

W arytmetyce komputerowej konieczne są dodatkowe ustalenia dotyczące sposobu kodowania mnożnika M i wykładnika E. Ponieważ liczba pozycji przeznaczonych na zapis tych wartości jest ograniczona, więc konsekwencją jest ograniczenie zakresu liczb (wartości wykładnika) i dokładności ich zapisu. W efekcie takie liczby są reprezentowane z pewną dokładnością względną, którą wyznacza waga najmniej znaczącej cyfry mnożnika. Format taki nazywa się zmiennoprzecinkowym (ang. *floating point*).

Aby umożliwić przenośność danych kodowanych w formacie zmiennoprzecinkowym, niezbędna jest standaryzacja reprezentacji liczb. Zgodnie ze standardem IEEE 754 wykładnik jest k-bitową liczbą całkowitą a mnożnik liczbą z przedziału [1,2). Znak liczby jest kodowany osobno. Zdefiniowanych jest kilka formatów zapisu wartości, które różnią się rozmiarem wykładnika i mnożnika.

Ponieważ każdą liczbę z przedziału [1,2) można przedstawić jako $1+f$, więc w kodzie liczby zapisane są tylko bity ułamka f. Kod wykładnika zawierający same jedynki wskazuje obiekty specjalne, takie jak nieskończoności czy też nie-liczby (NaN), czyli wyniki działań, które nie są liczbami rzeczywistymi. Kod wykładnika zawierający same zera oznacza liczbę zdenormalizowaną, czyli taką, w której mnożnik jest liczbą z przedziału [0,1). Aby umożliwić standardowy zapis odwrotności każdej liczby znormalizowanej zakres wartości wykładnika musi mieć asymetrię dodatnią. Skutkiem tego wykładnik jest zapisany w tzw. kodzie spolaryzowanym z przesunięciem $2^{k-1}-1$ (kod naturalny liczby uzyskanej przez powiększenie jej oryginalnej wartości o przesunięcie), a nie w kodzie U2. Kody te są jednak silnie powiązane, bowiem

$$\sum_{i=0}^{k-1} z_i 2^i - (2^{k-1} - 1) = -[-z_{k-1}2^{k-1} + \sum_{i=0}^{k-1} (1 - z_i)\, 2^i] \qquad (12)$$

Wykładnik o reprezentacji $\{z_{k-1}\ z_{k-2} \ldots z_2\ z_1\ z_0\}$ w kodzie spolaryzowanym $+(2^{k-1}-1)$ ma więc wartość przeciwną do liczby o reprezentacji $\{z_{k-1}\ (1-z_{k-2}) \ldots (1-z_2)\ (1-z_1)\ (1-z_0)\}$ w kodzie U2. Wynika stąd możliwość zastąpienia dodawania i odejmowania w kodzie spolaryzowanym działaniami w kodzie U2, zgodnie z oczywistą równością:

$$X \pm Y = -[\,(-X) \pm (-Y)\,]\,. \qquad (13)$$

A zatem, należy przekodować każdy argument na kod U2 przez zanegowanie wszystkich jego bitów oprócz najwyższego, wykonać działanie w kodzie U2 (wraz ze sprawdzeniem poprawności) i przekodować wynik zgodnie z tą samą regułą, co argumenty.

Opisana własność kodu spolaryzowanego $+(2^{k-1}-1)$ umożliwia łatwą konwersję kodu wykładnika przy zmianie jego rozmiaru k. Ponieważ lewostronnym rozszerzeniem kodu U2 jest powielenie najwyższego bitu, więc rozszerzeniem kodu spolaryzowanego $+(2^{k-1}-1)$ na $k+$ bitów jest $\{z_{k-1}\ (1-z_{k-1}) \ldots (1-z_{k-1})\ z_{k-2} \ldots z_2\ z_1\ z_0\}$.

Wynik działania na liczbach w formacie zmiennoprzecinkowym musi mieć dokładność taką jak argumenty. Problem utrzymania dokładności powstaje w dzieleniu, bo iloraz mnożników

$$2^{-1} < M_1 / M_2 = (1 + f_1)/(1 + f_1) < 2 \qquad (14)$$

może być mniejszy od 1 i wtedy konieczna jest normalizacja, która polega na przesunięciu bitów ilorazu w lewo.

Aby podczas normalizacji nie utracić dokładności konieczny jest dodatkowy bit, zwany bitem chroniącym G (ang. *guard*). Ponieważ wynik dzielenia jest niemal zawsze przybliżony, do poprawnego zaokrąglenia potrzebny jest kolejny bit R (ang. *round*). Aby uniknąć kumulacji błędów, gdy R=1, a na pozycjach wyniku o wagach niższych niż bit R wytworzone są bity 0…0, stosowane jest zaokrąglanie symetryczne (do najbliższej parzystej) – w górę lub w dół, zależnie od wartości bitu wyniku na pozycji G. Informację o tym czy na pozycjach następujących po bicie R są same zera zawiera bit wskaźnikowy S (ang. *sticky*).

Dodatkowe bity G, R i S, wytwarzane w procedurze dzielenia, zapewniają poprawne zaokrąglenie każdego ilorazu. Aby zapewnić jednakowy rozmiar wszystkich argumentów działań, argumenty zewnętrzne, traktowane są jako wartości dokładne, więc dołączane do nich bity G, R i S są zerowane.

Problem utrzymania dokładności nie występuje w mnożeniu, gdzie iloczyn mnożników ma dwa razy tyle bitów co argumenty. Można wtedy łatwo wytworzyć dodatkowe bity *G, R, S* i w razie potrzeby wykonać zaokrąglenie iloczynu.

Problem utrzymania dokładności nie występuje w dodawaniu liczb tego samego znaku lub odejmowaniu liczb o znakach przeciwnych. W takich przypadkach znak wyniku jest znany, zaś dodawanie wartości bezwzględnych ($E_1 \geq E_2$)

$$M_1 2^{E_1} + M_2 2^{E_2} = (M_1 + M_2 2^{-(E_1 - E_2)}) 2^{E_1} \tag{15}$$

zachowuje dokładność, bo wszystkie bity zdenormalizowanego składnika są dokładne.

Bity *G, R, S* nie wystarczają do wytworzenia poprawnego wyniku, jeśli wykonywane jest odejmowanie liczb zgodnego znaku lub dodawanie liczb przeciwnych znaków. Mamy wtedy, przy założeniu porządkowym, że $E_1 \geq E_2$:

$$M_1 2^{E_1} - M_2 2^{E_2} = (M_1 - M_2 2^{-(E_1 - E_2)}) 2^{E_1} . \tag{16}$$

Jeśli $M_1 - M_2\, 2^{-a} < ½$ ($a = E_1 - E_2$), to nastąpi tzw. katastroficzna utrata dokładności – nie można określić wartości najniższego i być może kilku kolejnych wyższych bitów ułamka. Jedynym wyjątkiem jest sytuacja, gdy bity S obu argumentów są zerami. Jest to bardzo mało prawdopodobne, gdy jeden z argumentów jest wynikiem wcześniejszego działania. Skutecznym rozwiązaniem problemu jest tylko zmiana algorytmu obliczeniowego [3].

Przekroczenie zakresu liczb zmiennoprzecinkowych daje różne skutki, zależnie od tego, czy wytworzony wynik jest zbyt mały, czy zbyt duży. Wynik zbyt mały jest zapisywany jako liczba zdenormalizowana i może być sygnalizowany jako niedomiar zmiennoprzecinkowy. Następuje wtedy także utrata dokładności. Wynik zbyt duży, sygnalizowany jako nadmiar zmiennoprzecinkowy, może być w niektórych sytuacjach zapisany jako nieskończoność. Takie rozwiązanie nie jest właściwe, jeśli jest to pośredni wynik obliczeń. Prostą techniką obsługi takiej sytuacji jest zanegowanie najwyższego bitu obliczonego wykładnika, czemu odpowiada skalowanie przez $2^{-(k-2)}$, i stosowna korekta wyniku w kolejnych krokach algorytmu obliczeniowego [3].

4. Równoległość w sprzęcie

Jak wynika z wcześniejszej dyskusji, w kontekście szybkości wykonywania działań kluczową sprawą jest konstrukcja szybkiego sumatora kodu U2. Zgodnie z analizą przeprowadzoną w rozdziale 1, działanie takiego sumatora opisuje równanie arytmetyczne (3), któremu odpowiada para równań logicznych:

$$\begin{aligned} s_i &= x_i \oplus y_i \oplus c_i, \\ c_{i+1} &= x_i \wedge y_i \vee c_i \wedge x_i \oplus y_i = x_i \wedge y_i \vee c_i \wedge (x_i \vee y_i) \end{aligned} \tag{17}$$

Rekurencyjna zależność przeniesień staje się ewidentna po zdefiniowaniu pomocniczych funkcji: półsumy $h_i = x_i \oplus y_i$, wytwarzania przeniesienia $g_i = x_i \wedge y_i$ i propagacji przeniesienia $p_i = x_i \vee y_i$. Mamy wtedy:

$$\begin{aligned} s_i &= h_i \oplus c_i, \\ c_{i+1} &= g_i \vee c_i \wedge h_i = g_i \vee c_i \wedge p_i \end{aligned} \tag{18}$$

Ponieważ elementarne funkcje g_i, p_i, h_i są wytwarzane jednocześnie na wszystkich pozycjach dodawania, więc szybkość ich wyznaczenia ma znikomy wpływ na czas dodawania. Kluczową sprawą jest szybkość wytworzenia wszystkich przeniesień.

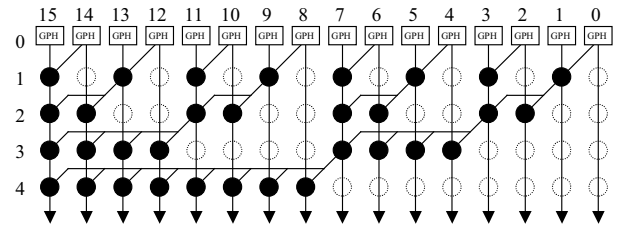
Z postaci równania przeniesienia (17) wynika, że przeniesienie wyjściowe c_{i+1} może być równe 1, jeśli spełniony jest warunek wymuszenia g_i=1 lub przeniesienie wejściowe c_i=1 i spełniony jest warunek propagacji p_i=1. Spostrzeżenie to można rozciągnąć na spójny blok kolejnych pozycji dodawania

$$c_{i+r+1} = G_{i:i+r} \vee P_{i:i+r} \wedge c_i \tag{19}$$

przy czym $G_{i:i} = g_i$, $P_{i:i} = p_i$ a blokowe funkcje G i P są powiązane zależnościami rekurencyjnymi

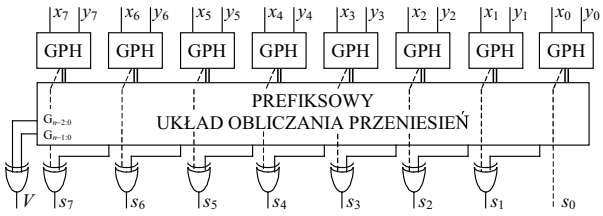
$$\begin{aligned} G_{i:j} &= G_{k:j} + P_{k:j} G_{i:k-1}, \\ P_{i:j} &= P_{k:j} P_{i:k-1}. \end{aligned} \tag{20}$$

Równoległe wytworzenie wszystkich funkcji $G_{0:i}$ oraz $P_{0:i}$ jest możliwe w strukturze prefiksowej zawierającej $2\log_2 k$ poziomów logicznych [1][3]. Przykładowy schemat takiej struktury dla sumatora 16-bitowego pokazano na rysunku poniżej. Symbol GPH oznacza układ wytwarzający elementarne jednobitowe funkcje półsumy $h_i = x_i \oplus y_i$, generowania przeniesienia $g_i = x_i \wedge y_i$ i propagacji przeniesienia $p_i = x_i \vee y_i$. Węzeł zaczerniony oznacza układ logiczny realizujący odpowiednie blokowe funkcje G i P (20).



Rys. 1. Prefiksowa struktura obliczania przeniesień dla sumatora 16-bitowego
Fig. 1. The structure of parallel-prefix carry-computation of 16-bit adder

Architekturę sumatora prefiksowego dla kodu U2 pokazano na rys. 2. Ponieważ w zwykłym sumatorze przeniesienie wejściowe c_0 jest równe 0, więc wtedy zgodnie z (19) $c_i = G_{0:i-1}$. Z analizy przedstawionej w rozdziale 2 wynika więc, że wskaźnikiem przekroczenia zakresu jest $v = G_{0:n-1} \oplus G_{0:n-2}$.



Rys. 2. Architektura sumatora prefiksowego kodu U2
Fig. 2. Architecture of parallel-prefix adder for 2's complement code

Drugim problemem jest szybkie dodawanie wielu argumentów niezbędne na przykład w mnożeniu. Dla liczb w reprezentacjach pozycyjnych można w tym celu zastosować zasady przemienności i łączności dodawania. Jeśli reprezentacją pozycyjną j-tej liczby k-cyfrowej jest $X^j = \{x^j_{k-1}\, x^j_{k-2} \ldots x^j_1\, x^j_0\}$, to zachodzi równość

$$\sum_{j=1}^{n} X^j = \sum_{j=1}^{n} \sum_{i=0}^{k-1} x_i^j\, \beta^i = \sum_{i=0}^{k-1} \beta^i \sum_{j=1}^{n} x_i^j = \sum_{i=0}^{k-1} \beta^i \sum_{s=0}^{\log_\beta n} u_s^i \beta^s, \tag{21}$$

gdzie $U^i = \{\ldots u_1^i\, u_0^i\}$ reprezentuje sumę cyfr na pozycji i.

Dodawanie n liczb k-pozycyjnych można więc zastąpić dodawaniem k liczb $(\log_\beta n+1)$-pozycyjnych. Przekształcenie można wykonać w strukturze sumatora równoległego. Jego realizacja może być etapowa – każdą grupę $\beta+1$ cyfr o tej samej wadze zamieniamy na liczbę dwucyfrową, uzyskane tak argumenty przekształcamy dalej zgodnie z tą samą regułą.

W systemie dwójkowym przekształcenia etapowe są realizowane w strukturze sumatora zachowującego przeniesienia CSA, którego modułem konstrukcyjnym jest sumator jednopozycyjny opisany funkcjami logicznymi (17). Warto zauważyć, że czas dodawania w strukturze drzewa szybkich sumatorów (PPA) jest dłuższy niż w strukturze CSA. Jest tak dlatego, że w strukturze drzewa rozmiar argumentów na kolejnych poziomach dodawania wzrasta, a w strukturze CSA końcowe dodawanie jest zawsze k-pozycyjne.

5. Arytmetyka resztowa (modularna)

Reprezentacje resztowe umożliwiają zastąpienie działań na argumentach dużego zakresu równoległymi działaniami mniejszej skali [1]. Arytmetyka resztowa jest też stosowana w algorytmach kryptograficznych [6].

W systemie resztowym RNS($m_1, m_2,..., m_n$) liczba całkowita X jest reprezentowana przez wektor reszt z jej dzielenia przez liczby naturalne m_i: $X=\{X \bmod m_1, X \bmod m_2,\ldots, X \bmod m_n\}$. Jeśli liczby m_i, zwane modułami systemu, są wzajemnie względnie pierwsze, to taka reprezentacja liczby jest unikatowa (niepowtarzalna) dla każdej liczby z przedziału $A \le X < A+M$, gdzie $M=m_1 \cdot m_2 \cdot \ldots \cdot m_n$. Zwykle przyjmuje się $A=0$ (liczby naturalne) lub $A=-\frac{1}{2}M$ (symetryczny przedział liczb całkowitych). Stanowi o tym tzw. chińskie twierdzenie o resztach (CRT) [1][4]. Rozszerzona wersja tego twierdzenia podaje algorytm konwersji odwrotnej, czyli obliczenia wartości liczby całkowitej X, jeśli znana jest jej reprezentacja resztowa $\{r_1, r_2,\ldots, r_n\}$ w systemie RNS($m_1, m_2,..., m_n$).

Algorytm konwersji odwrotnej oparty jest na spostrzeżeniu, że skoro reprezentacje kolejnych M liczb całkowitych z przedziału $A \le X < A+M$ są unikatowe, to istnieją w tym przedziale liczby, których reprezentacjami są izolowane jedynki, czyli wektory $\{\ldots,0_{i1}, 1_i, 0_{i+1},,\ldots\}$. Liczby te zwane są jedynkami resztowymi. Z postaci wektorów reszt wynika, że jedynka resztowa $\mathbf{1}_i$ musi być wielokrotnością iloczynu wszystkich modułów oprócz modułu m, $\mathbf{1}_i = s \cdot M \cdot (m_i)^{-1}$, i taką liczbą, że reszta z jej dzielenia przez m_i jest równa 1. Czynnik s jest zatem rozwiązaniem kongruencji

$$s \cdot (M \cdot m_i^{-1}) \bmod m_i = 1. \tag{22}$$

Zgodnie z terminologią algebraiczną, rozwiązanie to nazywa się odwrotnością multyplikatywną niepełnego iloczynu modułów $\hat{m}_i = M \cdot (m_i)^{-1}$ i oznacza $\hat{m}_i^{-1} \bmod m_i$.

Obliczenie odwrotności multyplikatywnej można wykonać za pomocą odwróconego algorytmu Eulera, który w wersji podstawowej służy do obliczenia największego wspólnego podzielnika dwóch liczb naturalnych [1][4][6].

Ponieważ kongruencje są niezmiennicze wobec dodawania [4], więc $\{r_1, r_2,\ldots, r_n\} = \{r_1, 0,\ldots, 0\} + \{0, r_2,\ldots, 0\} + \ldots + \{0, 0,\ldots, r_n\}$. Wartością liczby reprezentowanej w systemie RNS($m_1, m_2,..., m_n$) przez $\{r_1, r_2,\ldots, r_n\}$ jest więc

$$X = A + \sum_{i=1}^{n} r_i \cdot \mathbf{1}_i = A + \sum_{i=1}^{n} r_i \cdot (\hat{m}_i \cdot \hat{m}_i^{-1} \bmod m_i). \tag{23}$$

Z niezmienniczości kongruencji wobec dodawania wynika też możliwość następującej dekompozycji reprezentacji resztowej:

$$\begin{aligned}\{r_1, r_2,\ldots, r_{i1},\ldots, r_n\} = \{&(r_1-r_i) \bmod m_1,\ldots, 0,\ldots, (r_n-r_i) \bmod m_n\} \\ &+ \{r_i \bmod m_1,\ldots, r_i,\ldots, r_i \bmod m_n\}\end{aligned} \tag{24}$$

Pierwsza z tych liczb o wartości X_i ,obliczonej zgodnie z (23) jest wielokrotnością modułu m_i, druga ma wartość r_i, a zatem

$$X = A + m_i \cdot X_i + r_i. \tag{25}$$

Definiując system resztowy wygodnie jest przyjąć, że jeden z modułów jest potęgą podstawy systemu liczenia β^d. Jeśli jest to moduł m_i, to reprezentacją liczby X w systemie o podstawie β będzie wtedy złożenie obliczonych zgodnie z (23) cyfr liczby X_i oraz d cyfr reprezentacji liczby r_i (wraz z wiodącymi zerami).

W obliczaniu reprezentacji resztowych liczb danych w systemie dwójkowym można wykorzystać pewien wniosek wynikający z twierdzenia Eulera [1][4]. Twierdzenie to stanowi, że jeśli N jest liczbą naturalną, a liczba naturalna $a<N$ nie ma wspólnych podzielników z N, to:

$$a^{\varphi(N)} \bmod N = 1, \tag{26}$$

gdzie $\varphi(N)$ jest funkcją Eulera. Podaje ona liczbę mniejszych od N liczb naturalnych względnie pierwszych z N. Jeśli rozkładem liczby N na czynniki pierwsze jest $N=p^i q^j r^k \ldots$, to

$$\varphi(p^i q^j r^k \ldots) = (p-1)p^{i-1}(q-1)q^{j-1}(r-1)^{k-1}\ldots \tag{27}$$

Z twierdzenia tego wynika, że jeśli N jest liczbą nieparzystą, to istnieje taka najmniejsza potęga naturalna liczby 2, $d=2^{-q}\varphi(N)$, dla której $2^d \bmod N=1$ lub $2^d \bmod N=-1$. Liczba d nazywa się odpowiednio okresem lub półokresem potęg dwójki względem N.

Istnienie okresu lub półokresu pozwala obliczać reszty mod N przez dodawanie, bo $2^{d+s} \bmod N = \pm 2^s$, a więc:

$$\left(\sum_{i=0}^{n} x_i 2^i\right) \bmod N = \left(\sum_{i=0}^{n/d} (-1)^{ci} \sum_{j=0}^{d-1} x_{j+id} 2^j\right) \bmod N \tag{28}$$

gdzie $c=0$ jeśli d jest okresem, $c=1$ jeśli d jest półokresem.

Dodawanie wieloargumentowe występujące w powyższym wzorze można wykonać w strukturze CSA. Jeśli $d > \log_2 N$, to po wstępnym zredukowaniu argumentu zgodnie z (28) konieczne jest jeszcze obliczenie reszty, co w niektórych przypadkach może wymagać dzielenia. Jeśli $d > \log_2 N$, to ewentualną korekcję wyniku (28) można wykonać przez odjęcie wartości N.

6. Podsumowanie

W artykule zaprezentowano niektóre wybrane właściwości arytmetyki realizowanej w układach cyfrowych. Opisano niektóre przydatne właściwości działań arytmetycznych umożliwiające uproszczenie algorytmów obliczeniowych, zwłaszcza tworzonych w językach niskiego poziomu, takich jak asembler. Wskazano też niektóre problemy, jakie mogą powstać w procedurach arytmetycznych, zwłaszcza w obliczeniach zmiennoprzecinkowych.

W części dotyczącej arytmetyki resztowej podano sposoby sprawnego wykonywania działań typowych dla tej arytmetyki oraz wskazówki pozwalające na uproszczenie obliczeń.

7. Literatura

[1] Biernat J: Architektura układów arytmetyki resztowej, AOW EXIT, 2007.
[2] Biernat J: Architektura komputerów, PWN, Warszawa, 1996.
[3] Koren I.: Computer Arithmetic Algorithms, A K Peters, Natick, MA, 2002
[4] Narkiewicz W.: Teoria liczb, Wydawnictwo Naukowe PWN, Warszawa, 2003.
[5] Parhami B.: Computer Arithmetic. Algorithms and Hardware Designs, Oxford University Press, New York – Oxford, 2000.
[6] Yan S.Y.: Teoria liczb w informatyce, Wydawnictwo Naukowe PWN, Warszawa, 2006.

Title: The ins and outs of Computer Arithmetic.

Artykuł recenzowany