Reprezentacja liczb
Algorytmy podstawowych działań
(W1,W2)

Bóg stworzył liczby naturalne,

wszystkie inne
są dziełem człowieka

Leopold Kronecker (1893)

Liczenie i liczby

pojawienie się
i rozwój handlu

↓

potrzeba liczenia →

↓

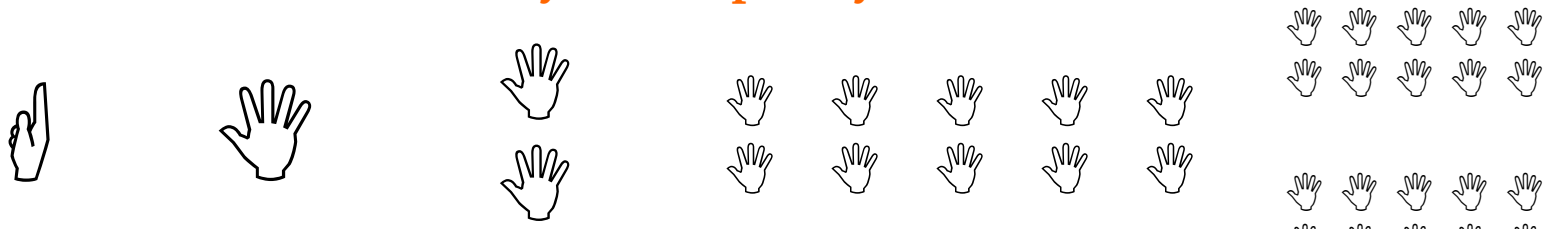
system miar i wag

trzy	*ziarna*
pięć	*garści ziaren*
cztery	*kubki ziaren*
trzy	*koszyki ziaren*
siedem	*worków ziaren*
liczba	**waga**
↓	↓
abstrakcja	*relacje wag/miar*

System liczbowy = zapis (reprezentacja) liczb + algorytmy działań

Zapis wartości liczby naturalnej

system/zapis rzymski



1	5	5x2	(5x2)x5	[(5x2)x5]x2
I	**V**	**X**	**L**	**C**

zapis pozycyjny – systematyczny i oszczędny:

$$235_{10}$$

$$2350_{10}=235*10$$

$$2350000_{10}=235*10*10*10*10=235*10^4$$

$$235{,}16_{10}=2*10^2+3*10^1+5*10^0+1*10^{-1}+6*10^{-2}$$

Zapis pozycyjny liczby (reprezentacja pozycyjna)

Standardowa reprezentacja/zapis pozycyjna:

$$\mathbf{X} = \{\ldots\ x_{i+1}\ x_i\ x_{i-1}\ \ldots\}_\beta ,$$

– ustalona podstawa β (liczba naturalna ≥ 2)

– numer pozycji, $\boldsymbol{i}$, określa jej wagę jako potęgę podstawy $\boldsymbol{\beta^i}$

– standardowy zbiór wartości cyfr $x_i \in \{0,1,\ldots,\beta-1\}$

Wtedy $\mathbf{X} = \{\ldots\ x_{i+1}\ x_i\ x_{i-1}\ \ldots\}_\beta$ reprezentuje wartość:

$$X = \ldots + x_{i+1}\beta^{i+1} + x_i\beta^i + x_{i-1}\beta^{i-1} + \ldots = \sum_i x_i\beta^i$$

Jeśli reprezentacja liczby jest prawostronnie skończona ($\forall i < -m : x_i = 0$) to

$$X = \ldots + x_{i+1}\beta^{i+1} + x_i\beta^i + \ldots + x_{-m}\beta^{-m} + \ldots = \sum_{i=-m}^{\infty} x_i\beta^i = \beta^{-m}\sum_{i=0}^{\infty} x_{i-m}\beta^i$$

więc liczbę o *skończonym zapisie* można wyrazić jako **skalowaną l. całkowitą**

Odejmowanie w systemach pozycyjnych

$$X - Y = \sum_i x_i \beta^i - \sum_i y_i \beta^i = \sum_i (x_i - y_i)\beta^i = ? = \sum_i s_i \beta^i$$

Różnica na pozycji $-\beta + 1 \le x_i - y_i \le \beta - 1$ może nie mieć reprezentacji w zbiorze dozwolonych cyfr, ale daje się zawsze przedstawić jako

$$x_i - y_i = -c_{i+1}\beta + u_i, \text{ gdzie } u_i \in \{0,1,...,\beta-1\} \text{ a } c_{i+1} \in \{0,1\}.$$

Wartość c_{i+1} można traktować jako niezbędną korekcję kolejnej pozycji, więc potrzebnym przekształceniem jest

$$x_i - y_i - c_i = -c_{i+1}\beta + s_i,$$

a ponieważ $-\beta \le x_i - y_i - c_i \le \beta - 1$, więc $s_i \in \{0,1,...,\beta-1\}$ a $c_{i+1} \in \{0,1\}$.

Zatem różnicę można przedstawić w zapisie pozycyjnym:

$$X - Y = \sum_i x_i \beta^i - \sum_i y_i \beta^i = \sum_i (x_i - y_i)\beta^i = \sum_i s_i \beta^i$$

Algorytm odejmowania w systemach pozycyjnych

0. $i=-m$, ; numer najniższej pozycji
   $c_{-m}=0$; korekta najniższej pozycji

Powtarzaj na kolejnych pozycjach:

1. Oblicz $u_i = x_i - y_i - c_i$; różnica tymczasowa

2. Gdy $u_i \geq 0$, to $c_{i+1} = 0$ oraz $s_i = u_i$,,
   gdy $u_i < 0$, to $c_{i+1} = 1$ oraz $s_i = u_i + \beta$, ; różnica i korekta

Wnioski:

1. Algorytm nie zależy od wartości argumentów X, Y, więc $0-Y$ jest wykonalne
2. $\underline{Y}=0-Y$ można traktować jak ***reprezentację liczby przeciwnej*** do Y

System uzupełnieniowy – rozszerzenie systemu naturalnego, w którym liczbę **Y** przeciwną do danej **Y** reprezentuje wynik pozycyjnego odejmowania **0–Y=Y**.

Dodawanie w systemach pozycyjnych

$$X+Y=\sum_i x_i\beta^i+\sum_i y_i\beta^i=\sum_i(x_i+y_i)\beta^i=?=\sum_i s_i\beta^i$$

Suma na pozycji $0\le x_i+y_i\le 2(\beta-1)$ może nie mieć reprezentacji w zbiorze dozwolonych cyfr, ale daje się zawsze przedstawić jako

$$x_i+y_i=c_{i+1}\beta+u_i,\ \text{gdzie}\ u_i\in\{0,1,...,\beta-1\}\ \text{a}\ c_{i+1}\in\{0,1\}.$$

Wartość c_{i+1} można traktować jako niezbędną korekcję kolejnej pozycji, więc potrzebnym przekształceniem jest

$$x_i+y_i+c_i=c_{i+1}\beta+s_i,$$

a ponieważ $0\le x_i+y_i+c_i\le 2\beta-1$, więc $s_i\in\{0,1,...,\beta-1\}$ a $c_{i+1}\in\{0,1\}$.

Zatem sumę można przedstawić w zapisie pozycyjnym:

$$X+Y=\sum_i x_i\beta^i+\sum_i y_i\beta^i=\sum_i(x_i+y_i)\beta^i=\sum_i s_i\beta^i$$

Algorytm dodawania w systemie pozycyjnym

1. $i=-m$, ; numer najniższej pozycji
   $c_{-m}=0$; korekta najniższej pozycji

Powtarzaj na kolejnych pozycjach:

1. Oblicz $u_i = x_i + y_i + c_i$; różnica tymczasowa
2. Gdy $u_i < \beta$, to $c_{i+1} = 0$ oraz $s_i = u_i$,,
   gdy $u_i \geq \beta$, to $c_{i+1} = 1$ oraz $s_i = u_i - \beta$, ; różnica i korekta

Wniosek:

1. Algorytm nie zależy od wartości argumentów X, Y.
2. Wartość sumy nie więcej niż β+1 liczb jednocyfrowych w podstawie β wynosi:

$$r_i = \sum_{j=1}^{\beta+1} x_{i,j} = u_{i+1}\beta + v_i \leq \beta^2 - 1$$

gdzie $v_i = r_i \bmod \beta$, $u_{i+1} = r_i \operatorname{int} \beta$

Uzupełnieniowa reprezentacja liczb

System uzupełnieniowy – rozszerzenie systemu naturalnego, w którym liczbę $\underline{\mathbf{X}}$ przeciwną do danej $\mathbf{X}$ reprezentuje wynik pozycyjnego odejmowania $\mathbf{0}-\mathbf{X}=\underline{\mathbf{X}}$.

Jeśli w ciągu cyfr $\mathbf{X}=\{\ldots x_{i+1}\ x_i\ x_{i-1}\ \ldots\}_\beta$ począwszy od pewnej pozycji n

- wszystkie kolejne cyfry są zerami, to liczba ma wartość dodatnią
  (tak jak w zapisie naturalnym)
- wszystkie kolejne cyfry są "$\beta-1$", to liczba ma wartość ujemną
  (cyfry "$\beta-1$" są wynikiem algorytmu w odejmowaniu od 0 liczby dodatniej)

Nieskończony lewostronnie ciąg cyfr 0 lub "$\beta-1$" oznaczany (0) lub $(\beta-1)$ nazywa się *rozszerzeniem nieskończonym*.

Z algorytmu odejmowania wynika, że $0-(\beta-1)=1$, więc $(\beta-1)=-1$.

Wartością liczby $\{(x_{\geq n})\ x_{n-1}\ \ldots\ x_i\ x_{i-1}\ \ldots\}_\beta$, gdzie $(x_{\geq n})=(0)$ lub $(\beta-1)$, jest

$$e(x_{\geq n})\beta^k+\sum_{i<n}x_i\beta^i$$

gdzie $e(x_{\geq n})$ – wartość lewostronnego rozszerzenia: 0 $(x_{\geq n}=0)$ lub $\underline{1}$ $(x_{\geq n}=\beta-1)$

Skrócona reprezentacja uzupełnieniowa

Ograniczenia *fizyczne* urządzeń arytmetyki (np. rozmiar słowa maszynowego) oraz ograniczenia *konceptualne* w językach programowania, etc.: implikują potrzebę:

→ *skrócona reprezentacja uzupełnieniowa* (umowa) za pomocą k cyfr:

- można zapisać β^k różnych liczb (całkowitych)
- *postulat symetrii*: jest reprezentowana liczba (Z) i jej przeciwieństwo ($\underline{Z}$),
  * podstawa β *parzysta* − nie istnieje przeciwieństwo liczby $½\beta^k$ lub $-½\beta^k$,
  → *dodatnie* są liczby, których wiodącą cyfrą jest 0,1,..., $½\beta-1$,
  → *ujemne* są liczby, których wiodącą cyfrą jest $½\beta,...,\beta-2,\beta-1$,
  → łatwe wykrycie znaku i w konsekwencji rozszerzenie reprezentacji
  * podstawa β *nieparzysta* − postulat symetrii wykonalny, ale trudne wykrycie znaku liczba z wiodącą cyfrą $½(\beta+1)$ może być dodatnia lub ujemna

Wartością liczby $\{x_{k-1} \ldots x_1\ x_0\}_\beta$ jest

$$e(x_{k-1})\beta^k + \sum_{i=0}^{k-1} x_i \beta^i$$

gdzie $e(x_{k-1})$ − wartość lewostronnego rozszerzenia: 0 $(x_{k-1} < \frac{1}{2}\beta)$ lub $\underline{1}$ $(x_{k-1} \geq \frac{1}{2}\beta)$

Zamienność dodawania i odejmowania

Dopełnienie pozycyjne liczby $\mathbf{X} = \{\ldots x_{k-1} \ldots x_1\ x_0 \ldots\}_\beta$ (ang. *digit-complement*)

$$\overline{\mathbf{X}} = \{\ldots \overline{x}_{k-1} \ldots \overline{x}_1\ \overline{x}_0 \ldots : \overline{x}_i = (\beta - 1) - x_i\}_\beta$$

$$\mathbf{X} + \overline{\mathbf{X}} = \{\ldots \beta-1\ \beta-1\ \beta-1 \ldots\}_\beta = \mathbf{Q}$$

$$\overline{\overline{\mathbf{X}}} = \mathbf{X} \quad \text{oraz} \quad \overline{\mathbf{Q}} = \mathbf{0}$$

Jeśli jest wykonalne działanie $\overline{\mathbf{X}} + \mathbf{Y}$ lub $\overline{\mathbf{X}} - \mathbf{Y}$ to mamy

$$\mathbf{X} - \mathbf{Y} = (\mathbf{Q} - \overline{\mathbf{X}}) - \mathbf{Y} = \mathbf{Q} - (\overline{\mathbf{X}} + \mathbf{Y}) = \overline{\overline{\mathbf{X}} + \mathbf{Y}}$$

$$\mathbf{X} + \mathbf{Y} = (\mathbf{Q} - \overline{\mathbf{X}}) + \mathbf{Y} = \mathbf{Q} - (\overline{\mathbf{X}} - \mathbf{Y}) = \overline{\overline{\mathbf{X}} - \mathbf{Y}} = \overline{\overline{\mathbf{Y}} - \mathbf{X}}$$

Ponieważ $\underline{\mathbf{X}}=\mathbf{0}-\mathbf{X}=-\mathbf{X}$, więc $\underline{\mathbf{Q}} = \mathbf{0} - \mathbf{Q}$, oraz $\underline{\mathbf{X}} = \mathbf{0} - \mathbf{X} = \underline{\mathbf{Q}} + \mathbf{Q} - \mathbf{X} = \underline{\mathbf{Q}} + \overline{\mathbf{X}}$ oraz

$$\mathbf{X} - \mathbf{Y} = \mathbf{X} + (\overline{\mathbf{Y}} + \underline{\mathbf{Q}})$$

W reprezentacjach uzupełnieniowych *skończonych* $\underline{\mathbf{Q}} = \mathbf{ulp} = [\ldots,0,0,0,1]$,
w *nieskończonych* $\underline{\mathbf{Q}} = \overline{\mathbf{Q}}$ (! są *dwie reprezentacje zera*: (0),(0) oraz (β−1),(β−1))

Arytmetyka komputerów

arytmetyka liczb całkowitych (**stałoprzecinkowa**)
– ograniczony rozmiar (zakres) argumentów (k pozycji, liczby całkowite)
– ograniczony rozmiar (zakres) sumy/różnicy (k pozycji, liczba całkowita)

Suma lub różnica **liczb danych w ograniczonym zakresie** w każdej podstawie $\beta \geq 2$ mieści się w zakresie *podwojonym*, więc też w zakresie rozszerzonym o 1 pozycję:

$$-\tfrac{1}{2}\beta^k \leq X, Y < \tfrac{1}{2}\beta^k \;\Rightarrow\; -\tfrac{1}{2}\beta^{k+1} < -\beta^k \leq X \pm Y < \beta^k < \tfrac{1}{2}\beta^{k+1}$$

Wniosek:

Suma (różnica) obliczona w ograniczonym zakresie jest poprawna, jeśli jest równa sumie (różnicy) obliczonej w zakresie rozszerzonym o 1 pozycję, czyli wtedy, gdy dodatkowa pozycja jest pozycją lewostronnego rozszerzenia.

W podstawie $\beta=2$ pozycja wiodąca jest pozycją rozszerzenia $e(x_{k-1}) = x_{k-1}$, więc

$$s_{k-1} = x_{k-1} \pm y_{k-1} \pm c_{k-1} \mp 2c_k$$
$$s_k = x_{k-1} \pm y_{k-1} \pm c_k \mp 2c_{k+1}$$

zatem $s_k = s_{k-1}$ wtedy i tylko wtedy, gdy $c_k = c_{k-1}$.

Mnożenie w naturalnym systemie pozycyjnym

Iloczyn dodatniej mnożnej (ang. *multiplicand*)) $A = \{a_{k-1} \ldots a_1 \; a_0\}_\beta$ i dodatniego mnożnika (ang. *multiplier*) $X = \{x_{m-1} \ldots x_1 \; x_0\}_\beta$ jest

sumą m iloczynów częściowych $x_i A\beta^i = \beta^i \sum_{j=0}^{k-1} x_i a_j \beta^j$:

$$A \cdot X = A \cdot (\sum_{i=0}^{\infty} x_i \beta^i) = (\sum_{i=m}^{\infty} x_e A\beta^i) + (\sum_{i=0}^{m-1} x_i A\beta^i) = (x_e) A\beta^m + (\sum_{i=0}^{m-1} x_i A\beta^i)$$

Poszczególne pozycje elementarnego iloczynu $A_i = x_i A = \{\ldots u_{k-1} \ldots u_1 \; u_0\}$

$$(v_{i,j+1}, u_{i,j}) = ((x_i y_j + v_{i,j}) \text{ int } \beta, (x_i y_j + v_{i,j}) \bmod \beta)$$

(jesli $k \le x_i y_j + v_{i,j} < k+1$, to $v_{\mathrm{i,j+1}} = k$ oraz $u_{i,j} = x_i y_j + v_{i,j} - k\beta$)

Tabliczki mnożenia

W podstawach $\beta \le 10$ wszystkie pozycje określają cyfry dziesiętne, więc obliczenie $x_i y_j$ int β oraz $x_i y_j \bmod \beta$ jest bardzo łatwe (mnożenie dziesiętne) i tworzenie tabliczki jest zbędne.

Mnożenie w systemie uzupełnieniowym

Iloczyn mnożnej (ang. *multiplicand*) $A = \{\ldots a_{k-1} \ldots a_1\ a_0\}_\beta$ i mnożnika (ang. *multiplier*) $X = \{\ldots x_{m-1} \ldots x_1\ x_0\}_\beta$ wynosi: $(\forall i > m-1 : x_i = x_e)$

$$A \cdot X = A \cdot (\sum_{i=0}^{\infty} x_i \beta^i) = (\sum_{i=m}^{\infty} x_e A\beta^i) + (\sum_{i=0}^{m-1} x_i A\beta^i) = e(x_{m-1}) A\beta^m + (\sum_{i=0}^{m-1} x_i A\beta^i)$$

jest sumą iloczynów częściowych $x_i A\beta^i = \beta^i \sum_{j=0}^{\infty} x_i a_j \beta^j$ oraz $e(x_{m-1})A\beta^m$, $e(x_{m-1}) = 0/\underline{1}$.

Jeśli więc $x_e = 0$, to $e(x_{m-1})A\beta^m = 0$, a jeśli $x_e = \beta - 1$, to $e(x_{m-1})A\beta^m = \underline{A} = 0 - A$.

Poszczególne pozycje elementarnego iloczynu $A_i = x_i A = \{\ldots u_{k-1} \ldots u_1\ u_0\}$

$$(v_{i,j+1}\ u_{i,j}) = ((x_i y_j + v_{i,j}) \operatorname{int} \beta, (x_i y_j + v_{i,j}) \bmod \beta)$$

(jesli $k \le x_i y_j + v_{i,j} < k+1$, to $v_{\mathrm{i,j+1}} = k$ oraz $u_{i,j} = x_i y_j + v_{i,j} - k\beta$)

Każdy *iloczyn częściowy* musi być określony *na wszystkich pozycjach akumulacji* nie wyłączając *rozszerzeń lewostronnych* (prawostronne są zerami)

Przekształcenia iloczynów częściowych

W mnożeniu $AX = \{...,a_{k-1},...,a_1,a_0\}_\beta \times \{...,x_{m-1},...,x_1,x_0\}_\beta$ każdy iloczyn częściowy $x_i A\beta^i = \beta^i\left(x_i e(a_{m-1})\beta^m + \sum_{j=0}^{m-1} x_i a_j \beta^j\right) = \beta^i\left(e(u_m)\beta^{m+1} + \sum_{j=0}^{m} u_j \beta^j\right)$ można zastąpić sumą liczby dodatniej $\beta^i\left([1+e(u_m)]\beta^{m+1} + \sum_{j=0}^{m} u_j \beta^j\right)$ (bo $1+e(u_m) = 1 \text{ lub } 0$) oraz $-\beta^i\beta^{m+1}$.

Tak przekształcone iloczyny częściowe mają rozszerzenia zerowe, lecz niezbędna jest końcowa korekcja przez odjęcie $\{1,...,1,1\}*\beta^{m+1}$ lub dodanie liczby przeciwnej.

Mnożenie w systemie dwójkowym

W systemie dwójkowym $e(x_{k-1}) = x_{k-1}$. Kolejnymi iloczynami częściowymi są więc $2^i x_i A = 2^i(-x_i a_{m-1}2^{m-1} + \sum_{j=0}^{m-2} x_i a_j) = 2^i((1-x_i a_{m-1})2^{m-1} + \sum_{j=0}^{m-2} x_i a_j) - 2^i 2^{m-1}$ oraz

$$-2^{k-1}x_{k-1}A = 2^{k-1}(x_{k-1}a_{m-1}2^{m-1} - \sum_{j=0}^{m-1} x_{k-1}a_j 2^j) = 2^{k-1}(x_{k-1}a_{m-1}2^{m-1} + \sum_{j=0}^{m-1} \overline{x_{k-1}a_j} 2^j - 2^{m-1} + 1))$$

a suma stałych korekcyjnych wynosi $-2^{k-1}(2^{m-1} + ... + 2 + 1) = -2^{m+k-1} + 2^{k-1}$.

ALGORYTM DZIELENIA
(W2)

Algorytm dzielenia

TWIERDZENIE

Dla danych liczb całkowitych X oraz $D\neq 0$ **istnieją** liczby całkowite Q i R, takie że

$$X=Q\cdot D+R, \qquad |R|<|D|$$

Nazywa się je odpowiednio *dzielną* X (ang. *dividend*), *dzielnikiem* $D\neq 0$ (ang. *divisor*) *ilorazem* Q (ang. *quotient*) i *resztą* R (ang. *remainder*)

- jeśli $R\neq 0$ równanie dzielenia ma 2 rozwiązania ($|R-R'|=|D|$)

$$X=Q\cdot D+R=Q'\cdot D+R', \quad |R|<|D|, \quad (|R-R'|=|D|, |Q-Q'|=1)$$

Wyróżnia się

- **dzielenie całkowite** (ang. *integer division*), gdzie znak reszty jest taki jak znak dzielnej ($XR>0$) – np. „idiv” w architekturze IA-32/32e
- **dzielenie znakowane** (ang. *signed division*) – znak reszty zgodny ze znakiem dzielnika ($RD\geq 0$) – procedura dzielenia w reprezentacji uzupełnieniowej
- **dzielenie modularne** (ang. *modulus division*), gdzie znak reszty jest dodatni
  - jeśli $D\in\mathbb{N}$, R nazywa się resztą modulo D, $R=X \bmod D$, zaś $Q=X \operatorname{int} D$.

Uniwersalny schemat dzielenia całkowitego

W zbiorze liczb całkowitych iloraz jest rozwiązaniem równania:

$$R = X - QD, \quad |R| < |D|$$

- **jeśli znaki dzielnika i dzielnej są zgodne** iloraz można obliczyć jako:

$$R = (\ldots(((X - D) - D) - D)\ldots - D) \text{ dopóki } |R| \geq |D|$$

albo: $R_0 = X - D$, $R_1 = R_0 - D$, ..., $R_i = R_{i-1} - D$ dopóki $|R_i| \geq |D|$ (np. do chwili zmiany znaku różnicy), i wtedy **iloraz** Q = liczba wykonanych odejmowań (i)

- **jeśli znaki dzielnika i dzielnej są niezgodne** iloraz można obliczyć jako:

$$R = (\ldots(((X + D) + D) + D)\ldots + D) \text{ dopóki } |R| \geq |D|$$

albo: $R_0 = X + D$, $R_1 = R_0 + D$, ..., $R_i = R_{i-1} + D$ dopóki $|R_i| \geq |D|$ (np. do chwili zmiany znaku sumy) i wtedy **iloraz** $Q = 0 -$ liczba wykonanych dodawań (i)

Dowolna dzielna i dzielnik

Jeśli dzielnik jest liczbą całkowitą lecz dzielna nie, to istnieje rozwiązanie równania

$$X = Q \cdot D + R, \qquad |R| < |D|$$

bo uniwersalna procedura będzie skuteczna, choć reszta nie będzie całkowita

Jeśli dzielnik nie jest liczbą całkowitą, to rozróżnić trzeba 2 przypadki:

- **dzielnik** jest **liczbą wymierną**
- **dzielnik** jest **liczbą niewymierną**

W pierwszym przypadku, jeśli dzielnik jest równy $D = P/N$, to rozwiązaniem problemu jest użycie dzielnika skalowanego $P = K * D$ i wtedy równanie

$$KX = Q \cdot (KD) + R', \qquad |R'| < |KD|$$

ma rozwiązanie Q.

W drugim przypadku możliwe jest zastosowanie:

- arytmetyki przedziałowej
- metod numerycznych

Poprawa dokładności ilorazu

Dla danych liczb całkowitych X oraz $D \neq 0$ istnieją liczby całkowite Q i R,

$$X = Q \cdot D + R, \qquad |R| < |D|$$

albo

$$|R = X - Q \cdot D| < |D|$$

Jeśli N jest naturalne, to NR jest liczbą całkowitą, więc istnieją całkowite q i r takie że

$$NR = q \cdot D + r, \qquad |r| < |D|$$

albo

$$|r = NR - q \cdot D| < |D|$$

zatem

$$NX = N(Q + q/N) \cdot D + r, \qquad |r| < |D|$$

więc *przeskalowanie reszty* umożliwia obliczenie *ilorazu z dowolną dokładnością,* którą można rekurencyjnie poprawiać (r jest liczbą całkowitą ...)

$$X = (Q + q/N) \cdot D + r/N, \qquad |r| < |D|$$

Praktyczny (standaryzowany) algorytm dzielenia

Jeśli $\boxed{|X| < |\beta^k D|}$, to iloraz $X/(\beta^k D)$ jest ułamkiem właściwym, a równanie:

$$X = Q \cdot (\beta^k D) + R, \qquad |R| < |\beta^k D|$$

ma dwa rozwiązania:

- jeśli znaki dzielnej i dzielnika są zgodne, to $Q=0$ i wtedy $R=X$, więc znak reszty jest także taki jak znak dzielnika
- jeśli znaki dzielnej i dzielnika są niezgodne, to $Q=\underline{1}$ a reszta $R=X+\beta^k D$ spełnia nierówność $|R| < |\beta^k D|$ i ma znak taki jak dzielnik

Jeśli $RD>0$, to równanie $\beta R = q \cdot (\beta^k D) + r$, $|r| < |\beta^k D|$ ma rozwiązanie w zbiorze dozwolonych cyfr $\{0,1,\ldots, \beta-1\}$ oraz $rD>0$. Liczba q jest wtedy pierwszą pozycją ułamka ilorazu w podstawie β, a kolejne można uzyskać wg tej samej procedury.

Dzielnik można traktować jak liczbę całkowitą, bo jeśli jest skończony, to zawsze można znaleźć takie skalowanie (przesunięcie przecinka), że $\beta^s(\beta^k D)$ jest liczbą całkowitą, zaś $X/(\beta^k D) = X\beta^s/(\beta^{s+k} D)$.

W efekcie dzielnik można traktować jak liczbę całkowitą.

Dzielenie w systemie pozycyjnym i uzupełnieniowym

Iloraz rzeczywisty w systemie uzupełnieniowym może być przedstawiony jako

$$Q = XD^{-1} = \beta^k(q_0 + q_{-1}\beta^{-1} + q_{-2}\beta^{-2} + q_{-3}\beta^{-3} + \ldots)$$

Wartość wiodącej pozycji przeskalowanego ilorazu $\beta^{-k}Q$ musi spełniać zależność:

$$X(D\beta^k)^{-1} - q_0 = q_{-1}\beta^{-1} + q_{-2}\beta^{-2} + q_{-3}\beta^{-3} + q_{-4}\beta^{-4} + \ldots < 1$$

więc gdy iloraz jest dodatni ($XD \geq 0$), to $q_0=0$, gdy jest ujemny, ($XD<0$), to $q_0=\underline{1}=(\beta-1)$ ponieważ *prawa strona równości jest dodatnim ułamkiem właściwym*.

Pozycjonowanie danych tak, aby waga wiodącej pozycji przeskalowanej dzielnej była niższa od wagi wiodącej pozycji dzielnika gwarantuje spełnienie warunku $|X| < |\beta^k D|$ czyli $|\beta^{-k}XD^{-1}| < 1$.

Kolejną cyfrę ilorazu można obliczyć podobnie, na podstawie nierówności

$$0 \leq \beta(\beta^{-k}X - q_0 D)D^{-1} - q_{-1} = (R_1 = \beta R_0 - q_{-1}D)D^{-1} = q_{-2}\beta^{-1} + q_{-3}\beta^{-2} + q_{-4}\beta^{-3} + \ldots < 1$$

Podobnie obliczymy kolejne cyfry ilorazu, bo reszty są powiązane rekurencyjnie:

$$R_i = \beta R_{i-1} - q_{-i}D, \text{ przy tym } 0 \leq R_i D^{-1} = q_{-i-1}\beta^{-1} + q_{-i-2}\beta^{-2} + \ldots < 1, \quad (R_0 = \beta^{-k}X - q_0 D)$$

bo tylko wtedy istnieje rozwiązanie q_{-i} w zbiorze $\{0,1,2,\ldots\beta-1\}$. A zatem **znak każdej reszty musi** być taki jak **znak dzielnika**, a jej *amplituda* mniejsza od dzielnika.

Dzielenie – algorytm obliczeniowy

Cyfra wiodąca (najbardziej znacząca) – pozycja najbardziej lewostronna, której pominięcie zmieni wartość liczby (ostatnia przed rozszerzeniem lewostronnym)

Algorytm dzielenia w systemie uzupełnieniowym $(X/D = \{x_k, x_{k-1}, ...\}:\{d_m, d_{m-1}, ..., d_{s<m}\})$

0. Określ wagę najbardziej znaczącej pozycji dzielnej (β^k) i dzielnika (β^m). Podstaw $R_0 = X\beta^{-k+m-1}$, co odpowiada ustawieniu wiodącej cyfry dzielnika na pozycji wyższej niż wiodąca cyfra dzielnej.
1. i=0; jeśli $R_0 D \geq 0$, to $q_0 = 0$, $R_1 = R_0$, jeśli $R_0 D < 0$, to $q_0 = \underline{1} = (\beta - 1)$, $R_1 = R_0 + D$;
2. Oblicz największe q_i takie, że $R_{i+1} = \beta R_i - q_i D$ ma taki znak jak dzielnik
3. i++ ,
4. Jeśli ….i<k wróć do 2
5. Przeskaluj obliczony iloraz ułamkowy czynnikiem β^{k-m+1}.

Obliczanie pierwiastka kwadratowego

Podobnie jak w dzieleniu liczb naturalnych istnieje naturalne Q takie, że

$$Q^2 \le X < (Q+1)^2.$$

Algorytm: Ponieważ $1+3+\ldots+2n-1=n^2$, więc oszacowaniem pierwiastka jest liczba skutecznych odejmowań $(((X-1)-3)-5)-\ldots$ dopóki różnica jest dodatnia.

Dokładność przybliżenia można poprawić, zauważając że jeśli

$$(Q+\tfrac{q}{N})^2 \le X < (Q+\tfrac{q+1}{N})^2$$

to

$$(2NQ+q)q \le N^2(X-Q^2) < (2NQ+q+1)(q+1)$$

W zapisie pozycyjnym procedurę można standaryzować.
Przybliżeniem z dokładnością 1 cyfry jest $\sqrt{X} \cong Q = q\beta^m$, gdzie d=1,2,...β−1, i wtedy

$$(\beta^m d)^2 \le X < (\beta^m (d+1))^2,$$

więc $X\beta^{-2m}$ w części całkowitej jest liczbą najwyżej 2-cyfrową

$$1 \le d^2 \le X\beta^{-2m} < (d+1)^2 \le \beta^2.$$

Poprawianie dokładności pierwiastka kwadratowego w zapisie pozycyjnym

Jeśli $Q^2 \le X < (Q+1)^2$, to istnieje $0 \le q < \beta$ takie, że $(Q+q\beta^{-1})^2 \le X < (Q+(q+1)\beta^{-1})^2$, a $Q+\beta^{-1}q$ jest dokładniejszym przybliżeniem pierwiastka. Stąd wynika, że:

$$(\beta Q+q)^2 \le \beta^2 X < (\beta Q+(q+1))^2$$

czyli

$$(2\beta Q+q)\,q \le \beta^2 (X-Q^2) < (2\beta Q+q+1)\,(q+1)$$

Ponieważ

$$0 \le (X-Q^2) = R < 2Q+1$$

oraz

$$r = \beta^2 X - (\beta Q+q)^2 = \beta^2 (X-Q^2) - (2\beta Qq+q^2) = \beta^2 R - q\,(2\beta Q+q)$$

więc algorytm można powtarzać iteracyjnie biorąc

$$r = \beta^2 X - (\beta Q+q)^2 = \beta^2 R - q\,(2\beta Q+q)$$

gdzie Q jest aktualnie obliczonym przybliżeniem pierwiastka.

KONWERSJE PODSTAWY (W3)

Tworzenie pozycyjnej reprezentacji liczby

W systemie *o podstawie* β jednoznaczną *reprezentacją* $\{...,x_{i+1},x_i,x_{i-1},...\}_\beta$ liczby o danej wartości X (najczęściej w innej podstawie) jest rozwiązanie równania

$$\sum_i x_i\beta^i = ... + x_{k-1}\beta^{k-1} + x_{k-2}\beta^{k-2} + ... + x_{-m}\beta^{-m} + ... = X,$$

z warunkami: $x_i \in \{0,1,...,\beta-1\}$.

UWAGA: Rozwinięcie części ułamkowej może być nieskończone (okresowe).

W praktyce wartość danej liczby jest zwykle zapisana w systemie pozycyjnym, więc rozwiązanie problemu nazywa się ***konwersją podstawy***.

- działania wykonywane są w systemie źródłowym (o podstawie danej ω)
- podstawa systemu docelowego β musi być zapisana w systemie źródłowym (za pomocą kilku cyfr gdy $\beta>\omega$: $\beta=|\{b_p,\ldots,b_1,b_0\}_\omega|$)
- wartości kolejnych cyfr tworzone w systemie źródłowym $\{z_{s-1},z_{s-1},\ldots,z_{-r}\}_\omega$ należy zapisać w systemie o podstawie β ($s \le k\log_\beta\omega$): $\{d_{s-1},d_{s-1},\ldots,d_{-r}\}_\beta$

Konwersja do podstawy współdzielnej

podstawy współdzielne (*skojarzone*) – naturalne potęgi tej samej stałej: β^r oraz β^s

$$X = \ldots + x_5\beta^5 + x_4\beta^4 + x_3\beta^3 + x_2\beta^2 + x_1\beta^1 + x_0\beta^0 + x_{-1}\beta^{-1} + x_{-2}\beta^{-2} + \ldots =$$
$$= \ldots + (x_5\beta + x_4)\beta^4 + (x_3\beta + x_2)\beta^2 + (x_1\beta + x_0)\beta^0 + (x_{-1}\beta + x_{-2})\beta^{-2} + \ldots$$

a więc w ogólności (x_i – wartość cyfry w podstawie β):

$$\sum_i x_i\beta^i = \sum_j (x_{js+s-1}\beta^{s-1} + \ldots + x_{js+1}\beta + x_{js})(\beta^s)^j = \sum_j z_j(\beta^s)^j$$

gdzie $z_j = x_{js+s-1}\beta^{s-1} + \ldots + x_{js+1}\beta + x_{js} \in \{0,1,\ldots,\beta^s - 1\}$ – wartość cyfry w podstawie β^s

Złożenie konwersji - $\{\ldots\}_{\beta^s} \to \{\ldots\}_{\beta^k} \Leftrightarrow \{\ldots\}_{\beta^s} \to \{\ldots\}_{\beta} \to \{\ldots\}_{\beta^k}$

- zamiast konwersji $\{\ldots\}_{\omega} \to \{\ldots\}_{\beta}$ wygodniej realizować $\{\ldots\}_{\omega} \to \{\ldots\}_{\beta^s} \to \{\ldots\}_{\beta}$

Przykład

$$2347{,}35_8 = 2\cdot 8^3 + 3\cdot 8^2 + 4\cdot 8^1 + 7\cdot 8^0 + 3\cdot 8^{-1} + 5\cdot 8^{-2} =$$
$$= 010\ 011\ 100\ 111{,}011\ 101_2 = 0100\ 1110\ 0111{,}0111\ 0100_2 = 4\mathrm{E}7{,}74_{16}$$

$$(1)\,100\,111{,}011\,101_{\mathrm{U2}} = (7)47{,}35_{\mathrm{U8}} = (1111)\,1110\,0111{,}0111\,0100_{\mathrm{U2}} = (\mathrm{F})\mathrm{E}7{,}74_{\mathrm{U16}}$$

Tworzenie reprezentacji pozycyjnej w zadanej podstawie

Inne algorytmy dla części całkowitej i ułamka (*wagi=potęgi odwrotności podstawy*)

Dla części *całkowitej* X_I oraz *ułamkowej* X_F liczby X mamy odpowiednio

$$X_I = \sum_{i=0}^{\infty} x_i \beta^i = x_0 + \beta \sum_{i=0}^{\infty} x_{i+1} \beta^i = \ldots = x_0 + \beta(x_1 + \beta(x_2 + \ldots + \beta(x_{k-1} + \beta(x_e + \beta(x_e + \ldots)))))$$

$$X_F = \sum_{i=-1}^{-\infty} x_i \beta^i = \beta^{-1}\left(x_{-1} + \sum_{i=-1}^{-\infty} x_{i-1} \beta^i \right) = \ldots = \beta^{-1}\{x_{-1} + \beta^{-1}[x_{-2} + \beta^{-1}(x_{-3} + \beta^{-1}(x_{-4} + \ldots))]\}$$

Wniosek:
Algorytmy generowania reprezentacji są uniwersalne, ale muszą uwzględniać:

- zbiór dozwolonych cyfr reprezentacji niestandardowych
- ujemną podstawę w systemach negabazowych

Uwaga: W notacji uzupełnieniowej część ułamkowa liczby jest zawsze dodatnia
– ułamek o wartości ujemnej ma część całkowitą równą –1:

Jeśli $-1<f<0$, to $0<1+f<1$ zatem $f=\underline{1}+(1+f)$

Konwersja części całkowitej liczby

$A \bmod b$ – reszta z dzielenia A przez b, $A \operatorname{div} b$ – iloraz całkowity A przez b

$$X_I = Q_0 = x_0 + \beta\{x_1 + \beta[x_2 + \beta(x_3 + \ldots + \beta(x_{k-2} + \beta\, x_{k-1})\ldots)]\}$$

$$x_0 = Q_0 \bmod \beta$$

$$Q_0 \operatorname{div} \beta = Q_1 = x_1 + \beta[x_2 + \beta(x_3 + \beta[x_4 + \ldots + \beta(x_{k-2} + \beta\, x_{k-1})\ldots])]$$

$$x_1 = Q_1 \bmod \beta$$

$$Q_1 \operatorname{div} \beta = Q_2 = x_2 + \beta(x_3 + \beta[x_4 + \beta(x_5 + \ldots + \beta(x_{k-2} + \beta\, x_{k-1})\ldots)])$$

$$x_2 = Q_2 \bmod \beta$$

...

cyframi rozwinięcia części całkowitej X_I liczby X w *systemie o podstawie* β są:

$$x_j = Q_j \bmod \beta,\ Q_{j+1} = Q_j \operatorname{div} \beta = \operatorname{int}(Q_j / \beta),\ Q_0 = X_I$$

spostrzeżenie:

1. Kolejny iloraz jest mniejszy od poprzedniego $|Q_r| < |Q_{r-1}|$ dopóki nie jest on bezwzględnie najmniejszy (0 lub $\underline{1}$) i wtedy $Q_r = Q_{r-1}$
2. Jeśli $Q_r = Q_{r-1}$, to $x_{r+1} = x_r = x_e$, (jeśli dwa kolejne ilorazy są identyczne, to kolejne cyfry lewostronnego rozwinięcia są cyframi rozszerzenia nieskończonego)

Algorytm konwersji części całkowitej liczby

Procedura

Powtarzaj, aż uzyskasz iloraz całkowity taki jak poprzedni (0 lub –1):

1. Oblicz iloraz całkowity i resztę z dzielenia poprzedniego ilorazu całkowitego przez podstawę docelową β
2. Otrzymana reszta to kolejna cyfra rozwinięcia pozycyjnego w podstawie β

Algorytm wyznaczania reprezentacji części całkowitej

0. $X^{(0)}=A,\ \ i=0$	; podstaw wartości początkowe
1. $X^{(i+1)}=\mathbf{int}(X^{(i)}/\beta)$	; iloraz całkowity
2. $x_i=X^{(i)}-\beta X^{(i+1)}$	; reszta
3. i++	; zwiększ i
4. jeśli $X^{(i+1)}\neq X^{(i)}$ wróć do 1	; powtarzaj dopóki nie powtórzy się iloraz
5. $k=i;\ x_{k+j}=x_j,\ \ j=0,1,\ldots i$	; kolejne to pozycje rozszerzenia lewostronnego

Uwaga: Użycie podstawy wyższej (współdzielnej) β^k zapewnia przyśpieszenie obliczeń – w jednym kroku oblicza się k kolejnych pozycji.

Np. $2^6=64$: konwersja na pośredni zapis ósemkowy jest banalna $\rightarrow$ 1 krok=6 bitów

Konwersja części ułamkowej liczby

$\operatorname{int} A$ – część całkowita liczby A

$$X_F = f_1 = \beta^{-1}\{x_{-1} + \beta^{-1}[x_{-2} + \beta^{-1}(x_{-3} + \ldots + \beta^{-1}(x_{-m+1} + \beta^{-1}x_{-m})\ldots)]\}$$

$$x_{-1} = \operatorname{int} \beta f_1$$

$$\beta f_1 - x_{-1} = f_2 = \beta^{-1}[x_{-2} + \beta^{-1}(x_{-3} + \beta^{-1}[x_{-4} + \ldots + \beta^{-1}(x_{-m+1} + \beta^{-1}x_{-m})\ldots])]\}$$

$$x_{-2} = \operatorname{int} \beta f_2$$

$$\beta f_2 - x_{-2} = f_3 = \beta^{-1}(x_{-3} + \beta^{-1}[x_{-4} + \beta^{-1}(x_{-5} + \ldots + \beta^{-1}(x_{-m+1} + \beta^{-1}x_{-m})\ldots)])$$

$$x_{-3} = \operatorname{int} \beta f_3$$

cyframi rozwinięcia części ułamkowej X_F liczby X w *systemie o podstawie* β są

$$x_{-j} = \operatorname{int} \beta f_j, \quad f_{j+1} = \beta f_j - x_{-j} < 1, \quad f_1 = X_F < 1$$

Zakończenie algorytmu:

1. Jeśli $f_r = 0$, to $x_{-(r+1)} = 0$, $f_{r+1} = 0$ itd. (kolejne cyfry rozwinięcia są zerami)
2. Jeśli dla $r > r_0$ jest $f_r = f_{r-k}$, to rozwinięcie jest okresowe (okres ma k cyfr)
3. Obliczono wymaganą liczbę cyfr

Algorytm konwersji ułamka wymiernego

Procedura (na podstawie rozwinięcia rekurencyjnego)

Powtarzaj tak długo aż:

– uzyskasz wymaganą dokładność β^{-m} (odpowiednią liczbę cyfr) lub iloczyn=0,

– wykryjesz okresowość (pojawi się argument (ułamek) taki jak wcześniej).

1. Pomnóż ułamek przez podstawę systemu docelowego β
2. Część całkowita iloczynu jest kolejną cyfrą rozwinięcia pozycyjnego
3. Część ułamkową iloczynu zwróć do procedury

Reprezentacja części ułamkowej ($A<1$)

0. $f_{(0)}=A,\ x_0=0,\ i=0$; podstaw wartości początkowe

Powtarzaj dopóki $i \le m$ lub $X^{(i+1)} \neq 0$ lub wykryto okresowość:

1. $x_{-i}=\mathbf{int}(\beta f_{(i)})$; część całkowita iloczynu
2. $f_{(i+1)}=\beta f_{(i)}-x_{-i}$; część ułamkowa iloczynu
3. i++ ; zwiększ i

Uwaga: liczba kroków algorytmu wynosi $\log_\beta A$, więc użycie podstawy β^k zapewni k-krotne przyśpieszenie – w jednym kroku oblicza się k kolejnych pozycji.

Mnożenie ułamka okresowego przez liczbę naturalną

Ułamek okresowy w systemie o podstawie β i okresie k-pozycyjnym ma wartość:

$$0,(d_{-1}d_{-2}...d_{-k})_\beta = (\sum_{i=0}^{i=k-1} d_{i-k}\beta^i)\beta^{-k} + (\sum_{i=0}^{i=k-1} d_{i-k}\beta^i)\beta^{-2k} + ... = \sum_{s=1}^{\infty} D\beta^{-sk} = D/(\beta^k - 1)$$

gdzie $D = \sum_{i=0}^{i=k-1} d_{i-k}\beta^i$ jest k-pozycyjną liczbą naturalną daną przez cyfry okresu.

Ułamek ten jest wymierny, więc jego iloczyn przez mnożnik naturalny jest liczbą wymierną, której część ułamkowa jest ułamkiem okresowym lub jest równa 0:

$$X \cdot 0,(d_{-1}d_{-2}...d_{-k}) = \sum_{i=0}^{k-1} z_i\beta^i + (\sum_{i=0}^{k-1} z_{i-k}\beta^i + \sum_{i=0}^{k-1} z_i\beta^i)\beta^{-k} + (\sum_{i=0}^{k-1} z_{i-k}\beta^i + \sum_{i=0}^{k-1} z_i\beta^i)\beta^{-2k} + ...$$

gdzie $X(\sum_{i=0}^{i=k-1} d_{i-k}\beta^i) = \beta^k \sum_{i=0}^{k-1} z_i\beta^i + \sum_{i=0}^{k-1} z_{i-k}\beta^i$ (w okresie wystąpi *przeniesienie cykliczne*)

0,(37) x 13	0,(98) x 99	0,35(7) x 7	0,00(7) x 7	0,(7) x 16	→ 0,(77) x 16	→ 0,(777) x 16
6,(29) + (06)	97,(02) + (97)	2,45+	0,04(9) + (4)		12,(32) + (12)	12,(432) (12)
6,(35)	98 = 97,(99)	2,50(4)	0,05(3) + (1)		12,(44)	12,(444)

Działania w dwójkowym systemie uzupełnieniowym (W4)

Przekształcenie Booth'a

Ciąg *jedynek* na kolejnych pozycjach można zastąpić ciągiem *zer* poprzedzonych „1" i zakończonych „1", np. 011111110 =10000001. Można to opisać przekształceniem

$$X = 2X - X = \sum_i x_i 2^{i+1} - \sum_i x_i 2^i = \sum_i (x_{i-1} - x_i) 2^i$$

czego efektem może być zmniejszenie liczby niezerowych iloczynów częściowych. Jest tak, bo wartością $x_{i-1} - x_i$ jest 0 jeśli sąsiednie bity są równe 1.

Warto zauważyć, że aby wynik przekształcenia był poprawny, konieczne jest uwzględnienie bitów lewostronnego rozszerzenia. Wskutek tego przekształcenia wszystkie te pozycje, oprócz najniższej, zostają zamienione na zera.

$2X$		(1)	0	1	1	1	0	1	0	1	1	1	1	0	1	1 (0)
$-X$	−		(1)	0	1	1	1	0	1	0	1	1	1	1	0	1 1
X		(0)	1	1	0	0	1	1	1	1	0	0	0	0	1	0 1

Przekształcenie Booth'a-McSorley'a

$2X$		(1)	0	1	0	1	0	1	0	1	1	1	1	1	1	0	1 (0)
$-X$	−		(1)	0	1	0	1	0	1	0	1	1	1	1	1	1	0 1
X		(0)	1	1	1	1	1	1	1	1	0	0	0	0	0	1	1 1
X		(0)	0	1	0	1	0	1	0	1	0	0	0	0	0	1	0 1

Wartości współczynników na sąsiednich pozycjach mogą być jednakowe tylko wtedy gdy są zerami, bo jeśli $x_i - x_{i+1} = x_{i-1} - x_i$ to $2x_i = x_{i+1} + x_{i-1}$, więc albo wszystkie są równe 0 albo wszystkie równe 1 − wtedy obie różnice =0.

Różne wartości mogą tworzyć pary 00, 01, 10, 0$\underline{1}$, $\underline{1}$0, 1$\underline{1}$, $\underline{1}$1, ale 1$\underline{1}$=01, $\underline{1}$1=0$\underline{1}$, więc każda para takich współczynników na każdych 2 kolejnych pozycjach może być przekształcona na parę zawierającą zero, przy tym:

$$(x_i - x_{i+1})2^{i+1} + (x_{i-1} - x_i)2^i = (-2x_{i+1} + x_i + x_{i-1})2^i$$

a wartością współczynnika $-2x_{i+1} + x_i + x_{i-1}$ może być tylko $\underline{1}$0 ($\underline{2}$), 0$\underline{1}$, 00, 01, 10 (2).

Skutkiem tego przekształcenia co najmniej połowa iloczynów częściowych jest 0

Algorytm Baugh'a-Wooley'a (bez rozszerzeń)

Jeśli wagą lewostronnego rozszerzenia liczby jest 2^k, czyli $Z:\{(z_k)\, z_{k-1}z_{k-2}...z_0...\}$, to

$$2^k + Z = 2^k + \sum_{i=k}^{\infty} z_e 2^i + \sum_{i=k-1}^{-\infty} z_i 2^i = \overline{z_e} 2^k + \sum_{i=k-1}^{-\infty} z_i 2^i = \tilde{Z} \geq 0$$

gdzie $\tilde{Z}:\{\overline{z_k}z_{k-1}z_{k-2}...z_0...\}$ jest *liczbą dodatnią*. Wynikiem takiej konwersji każdego iloczynu częściowego $2^i M_i = 2^i x_i A$, także $\sum_{i=n}^{\infty} x_e A 2^i = 2^n (\sum_{i=0}^{\infty} x_e \beta^i) A = 2^n M_n$, jest

$$\sum_{i=0}^{n} 2^i M_i = \sum_{i=0}^{n} 2^i (\tilde{M}_i - 2^k) = \sum_{i=0}^{n} 2^i \tilde{M}_i + 2^k - 2^{n+k+1}$$

		(1)	0	1	1	1	1
	+ (0)	1	0	0	0	0	0
		(1)	0	1	1	1	1
	+ (0)	1	0	0	0	0	0
		(0)	1	0	0	0	0
+ (0)	1	0	0	0	0	0	
– (0)	**1**	**1**	**1**				

	(0)	0	1	1	1	1
(0)	0	1	1	1	1	
+ **(1)**	1	0	0	0	0	
+ (1)	**0**	**0**	**1**			

Dzielenie nieodtwarzające

Dzielenie w systemie dwójkowym można wykonać wg powtarzanego schematu:

- przeskaluj resztę i odejmij od niej dzielnik $(2r_i - D)$
- jeśli znak różnicy jest taki jak znak dzielnika, kolejną cyfrą ilorazu jest 1, w przeciwnym kolejną cyfrą ilorazu jest 0
- jeśli znak różnicy jest inny niż znak dzielnika skoryguj obliczoną różnicę dodając do niej dzielnik (odtworzenie poprawnej reszty) $((2r_i - D) + D)$

Jeśli znak różnicy jest inny niż znak dzielnika, przesłanką kolejnej decyzji jest:

$$r_{i+1} = 2[(2r_i - D) + D] - D = 2(2r_i - D) + D$$

Stąd wynika, że algorytm można uprościć

- jeśli znaki różnicy i dzielnika są zgodne, kolejną cyfrą ilorazu jest 1, a kolejną przesłanką jest różnica przeskalowanej poprzedniej różnicy i dzielnika
- jeśli znaki różnicy i dzielnika są różne, kolejną cyfrą ilorazu jest 0, a kolejną przesłanką jest suma przeskalowanej poprzedniej różnicy i dzielnika $2r_i + D$.

Algorytm dzielenia nieodtwarzającego

0. Skaluj dzielnik D tak aby był bezwzględnie większy od dzielnej X, $|2^k D| > |X|$
   („wysuń” wiodącą cyfrę dzielnika o 1 pozycję przed wiodącą cyfrą dzielnej)
1. $i=0$,
   - jeśli znaki dzielnej i dzielnika są zgodne ($XD>0$) oblicz $r_0 = 2^{-k} X - D$
   - jeśli znaki dzielnej i dzielnika są zgodne ($XD<0$) oblicz $r_0 = 2^{-k} X + D$
2. Jeśli różnica ma taki znak jak dzielnik $r_i D > 0$, kolejnym bitem ilorazu jest $q_i = 1$,
   w przeciwnym razie $q_i = 0$.
3. Przeskaluj różnicę i oblicz kolejną jako $r_{i+1} = 2r_i - (1-2q_i)D$
4. Zwiększ i, jeśli $i<k$ wróć do p. 2

Pierwszy bit ilorazu q_0 jest jednocześnie bitem lewostronnego rozszerzenia.

Dzielenie odtwarzające i nieodtwarzające

D = (1) 0 1 1 1

		q		
(1) 1 1 0 1				(1) 1 1 0 1
− (1) 0 1 1 1				− (1) 0 1 1 1
(0) 0 1 1 0	→	**0**	←	(0) 0 1 1 0 0
+ (1) 0 1 1 1				
(1) 1 1 0 1 0				
− (1) 0 1 1 1				+ (1) 0 1 1 1
(0) 0 0 1 1	→	**0**	←	(0) 0 0 1 1 1
+ (1) 0 1 1 1				
(1) 1 0 1 0 1				
− (1) 0 1 1 1				+ (1) 0 1 1 1
(1) 1 1 1 0 0	→	**1**	←	(1) 1 1 1 0 0
− (1) 0 1 1 1				− (1) 0 1 1 1
(0) 0 1 0 1	→	**0**	←	(0) 0 1 0 1

Arytmetyka przybliżeń (W5)

Standard zmiennoprzecinkowy IEEE754-2008

Notacja naukowa (*inżynierska*), *postać wykładnicza*, *format wykładniczy*:

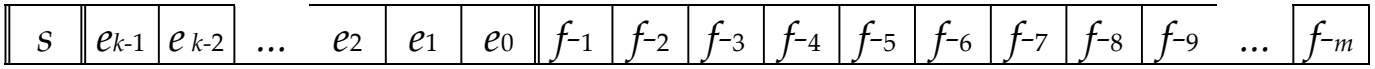
$$\pm M \cdot \beta^{E} = \pm d_0 , f_{-1} f_{-2} \cdots f_{-m} \cdot \beta^{\pm e_{k-1} e_{k-2} \ldots e_1 e_0}$$

- *przybliżenie liczby rzeczywistej* z dokładnością m pozycji ułamka
- reprezentacja *znormalizowana* (unikatowa): 1 cyfra części całkowitej

Standard IEEE754-2008 (dwójkowy):

jednostka kodu (rekord)

$d_0 = 1$ (0 gdy e="00...00")



znak E – wykładnik ***k* bitów** f –ułamek mnożnika (***m* bitów**), $|M| = d_0 + f$

nazwa/rozmiar	*dawna nazwa*		*extended* (*f. wewnętrzny*)
16b	(SEMI)	$k=5,\ m=10$	$k \geq 8,\ m \geq 16$
32b	SINGLE /REAL	$k=8,\ m=23$	$k \geq 11,\ m \geq 32$
64b	DOUBLE	$k=11,\ m=52$	$k \geq 15,\ m \geq 64$
128b	(QUADRUPLE)	$k=15,\ m=112$	$k \geq 17,\ m \geq 128$
n×32b	(VOID)	$k \geq 17$, $m = 32n - k - 1$	

Kodowanie IEEE754-2008:

formaty dwójkowe

znak s: 0 – liczba dodatnia, 1 – liczba ujemna

wykładnik E – k-bitowy kod spolaryzowany $+2^{k-1}-1$

kody specjalne:

nieskończoność – wykładnik „11...1”, ułamek „00...0”

nie-liczba – wykładnik „11...1”, ułamek dowolny ≠„00...0”

liczby najmniejsze – wykładnik „00...0” o wartości jak wykładnik „00...01”

kody znormalizowane: – wykładnik „00...01” .... „11...10”

mnożnik znormalizowany $M = 1 + f = 1, f_{-1} f_{-2} \ldots f_{-m}$,

ułamek f w zapisie naturalnym, ukryta (niekodowana) „1” części całkowitej

mnożnik denormalizowany – wykładnik „00...0”, $M = 0 + f = 0, f_{-1} f_{-2} \ldots f_{-m}$,

ułamek f w zapisie naturalnym, ukryte (niekodowane) „0” części całkowitej

formaty dziesiętne

ułamek w kodzie *DPD* (ang. *Densely Packed Decimal*), wykładnik w kodzie *BCD*

Działania w formacie zmiennoprzecinkowym

Sposób wykonania działań:

$$F_1 \pm F_2 = (M_1 \beta^{E_1}) \pm (M_2 \beta^{E_2}) = (M_1 \pm M_2 \beta^{-(E_1 - E_2)}) \beta^{E_1}$$

$$F_1 F_2 = (M_1 \beta^{E_1})(M_2 \beta^{E_2}) = (M_1 M_2) \beta^{E_1 + E_2}$$

$$F_1 F_2 = (M_1 \beta^{E_1}) / (M_2 \beta^{E_2}) = (M_1 / M_2) \beta^{E_1 - E_2}$$

Po obliczeniu następuje *normalizacja* wyniku. Jeśli po normalizacji:

- wykładnik jest zbyt duży: sygnalizuj *nadmiar wykładnika*
  a wyniku nie można zapisać → *obsługa wyjątku*
- wykładnik jest zbyt mały: sygnalizuj *niedmiar wykładnika*
  a wynik jest *liczbą zdenormalizowaną*

Uzyskany po normalizacji wynik trzeba zaokrąglić (standardy IEEE754-2008)

- *do zera* – obcinanie
- *do nieskończoności* – arytmetyka przedziałowa (*interval arithmetic*)
  dodatnie zawsze w górę, ujemne zawsze w dół (lub odwrotnie)
- *do najbliższej* (parzystej) – symetryczne (środek …xx0)

Ochrona przed utratą dokładności: bity *G,R,S*

Działania na kodach wykładników

Kodowanie wykładnika – (k – *liczba bitów*, e – *kod*, E – *wartość wykładnika)*

- kod spolaryzowany "$+2^{k-1}-1$" ($e_{\min}=00\ldots01_2$, $e_{\max}=11\ldots10_2$)
- *łatwa konwersja na kod U2 i odwrotnie*

$$\left|\{x_{k-1},x_{k-2},\ldots,x_1,x_0\}_{+(2^{k-1}-1)}\right| = -\left|\{x_{k-1},\overline{x}_{k-2},\ldots,\overline{x}_1,\overline{x}_0\}_{\mathrm{U2}}\right|$$

- *kody specjalne*:
  - $e=00\ldots00_2$ – *liczba zdenormalizowana* (zero i b.małe), $E=E_{\min}$
  - $e=11\ldots11_2$ – *nieskończoności* ($f=0$) i *nie-liczby, NaN* ($f\neq0$)

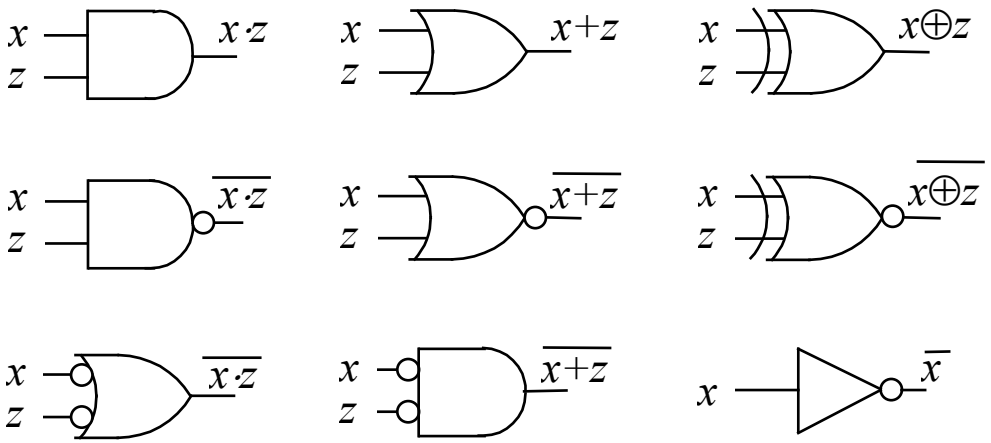
Uniwersalna procedura działania w kodzie $+2^{k-1}-1$
(*operacyjna* zmiana znaku argumentów jest odwracalna)

0. Jeśli argument zdenormalizowany, kod 00...00 zmień na 00...01
1. Przekoduj wykładniki na kod U2
2. Wykonaj działanie w kodzie U2
3. Przekoduj wynik na kod spolaryzowany $+(2^{k-1}-1)$
4. Jeśli potrzebna jest normalizacja skoryguj kod wynikowy

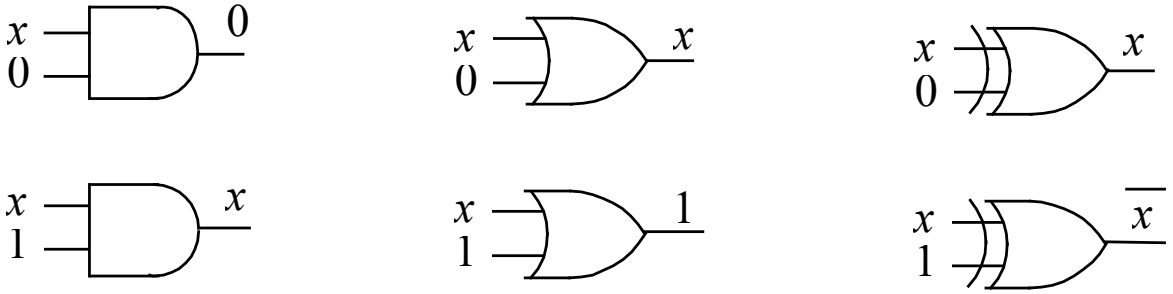
Struktury logiczne układów cyfrowych (W6)

Logika układów cyfrowych

Podstawowe bramki logiczne



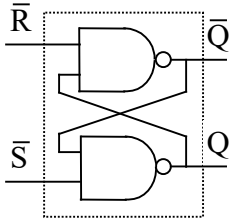
Bramki logiczne jako elementy sterowania

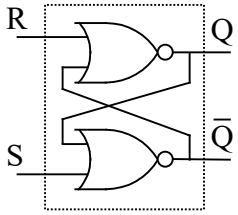


0 – *sygnał dominujący* 1 – *sygnał dominujący*

Elementy pamiętające

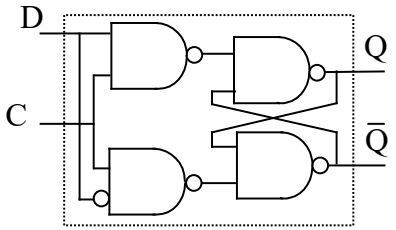
asynchroniczny przerzutnik RS

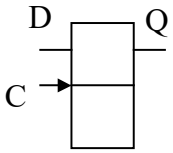




R	S	Q^{t+1}
0	0	Q^t
0	1	1
1	0	0
1	1	—

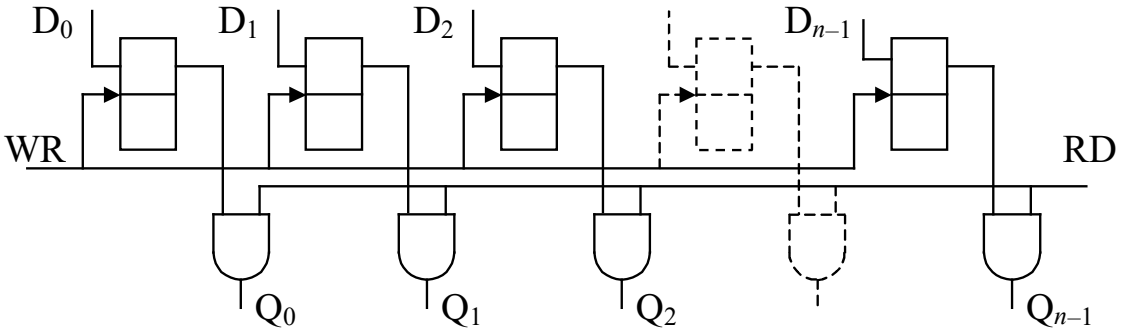
synchroniczny przerzutnik D





C	D	Q^{t+1}
0	—	Q^t
1	0	0
1	1	1

rejestr równoległy



Logika dodawania i odejmowania

Logika dodawania i odejmowania pozycyjnego:

Dla danych $x_i, y_i \in \{0,1,\ldots,\beta-1\}, c_i \in \{0,1\}$ istnieją $s_i \in \{0,1,\ldots,\beta-1\}, c_{i+1} \in \{0,1\}$ takie, że

$$\begin{aligned} x_i + y_i + c_i &= \beta c_{i+1} + s_i, \\ x_i - y_i - c_i &= -\beta c_{i+1} + s_i, \end{aligned} \quad \text{gdzie } x_i, y_i, s_i \in \{0,1,\ldots,\beta-1\}, c_i \in \{0,1\},$$

czemu odpowiada iteracyjne (kaskadowe) powiązanie pozycji.

W podstawie β=2 równaniom arytmetycznym odpowiadają funkcje logiczne:

dodawanie $\overline{X+Y} = \overline{X} - Y = \overline{Y} - X$

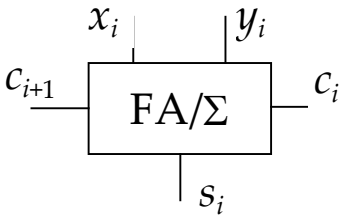
$$s_i = x_i \oplus y_i \oplus c_i = h_i \oplus c_i$$

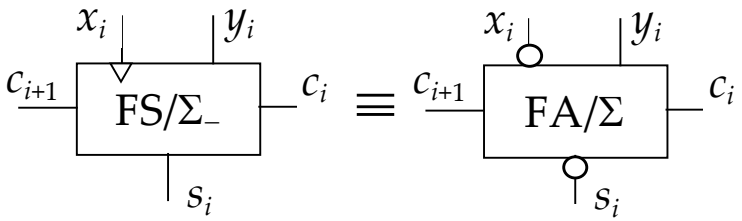
sumator:
$$\begin{aligned} c_{i+1} &= x_i y_i + (x_i \oplus y_i)\, c_i = \\ &= x_i y_i + (x_i + y_i)\, c_i = g_i + p_i c_i \end{aligned}$$

odejmowanie $\overline{X-Y} = \overline{X} + Y$

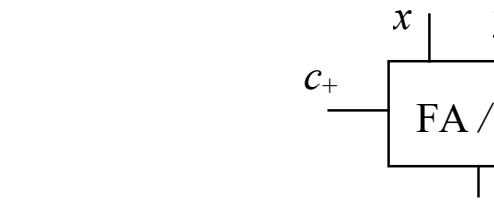
$$\bar{s}_i = \bar{x}_i \oplus y_i \oplus c_i = h_i \oplus c_i$$

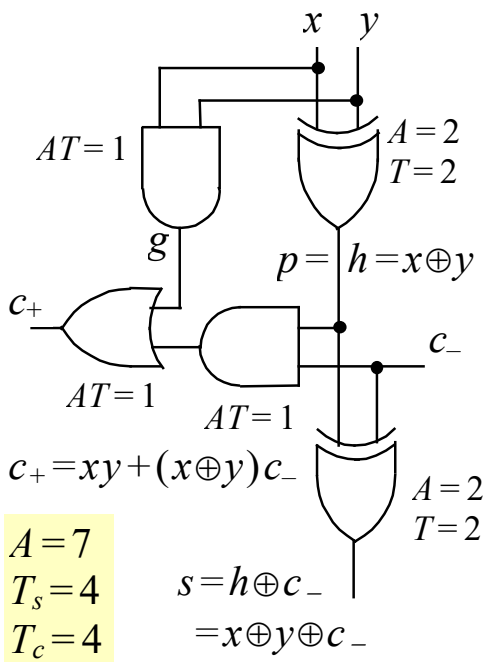
subtraktor:
$$\begin{aligned} c_{i+1} &= \bar{x}_i y_i + (\bar{x}_i \oplus y_i)\, c_i = \\ &= \bar{x}_i y_i + (\bar{x}_i + y_i)\, c_i = g_i + p_i c_i \end{aligned}$$

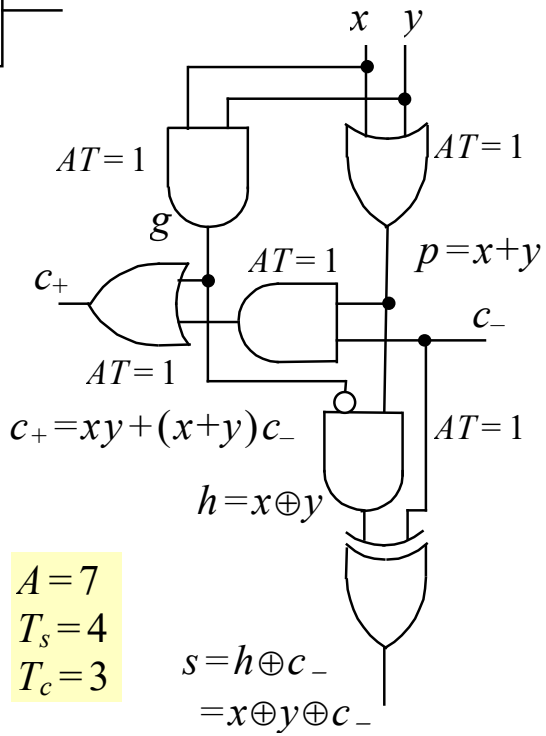


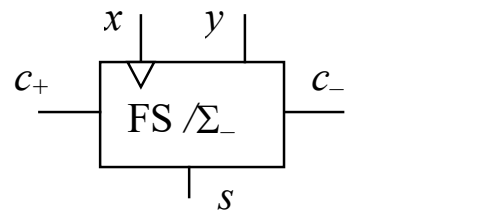


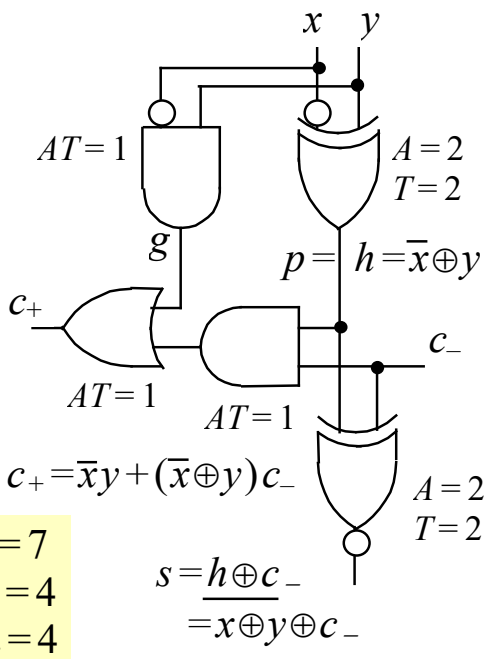
Sumatory


x
y
c_+
FA / Σ
c_-
s


x
y
AT = 1
A = 2
T = 2
g
p = h = x⊕y
c_+
c_-
AT = 1
AT = 1
$c_+ = xy + (x⊕y)c_-$
A = 2
T = 2
A = 7
T_s = 4
T_c = 4
$s = h⊕c_-$
$= x⊕y⊕c_-$


x
y
AT = 1
AT = 1
g
AT = 1
p = x+y
c_+
c_-
AT = 1
$c_+ = xy + (x+y)c_-$
AT = 1
$h = x⊕y$
A = 7
T_s = 4
T_c = 3
$s = h⊕c_-$
$= x⊕y⊕c_-$

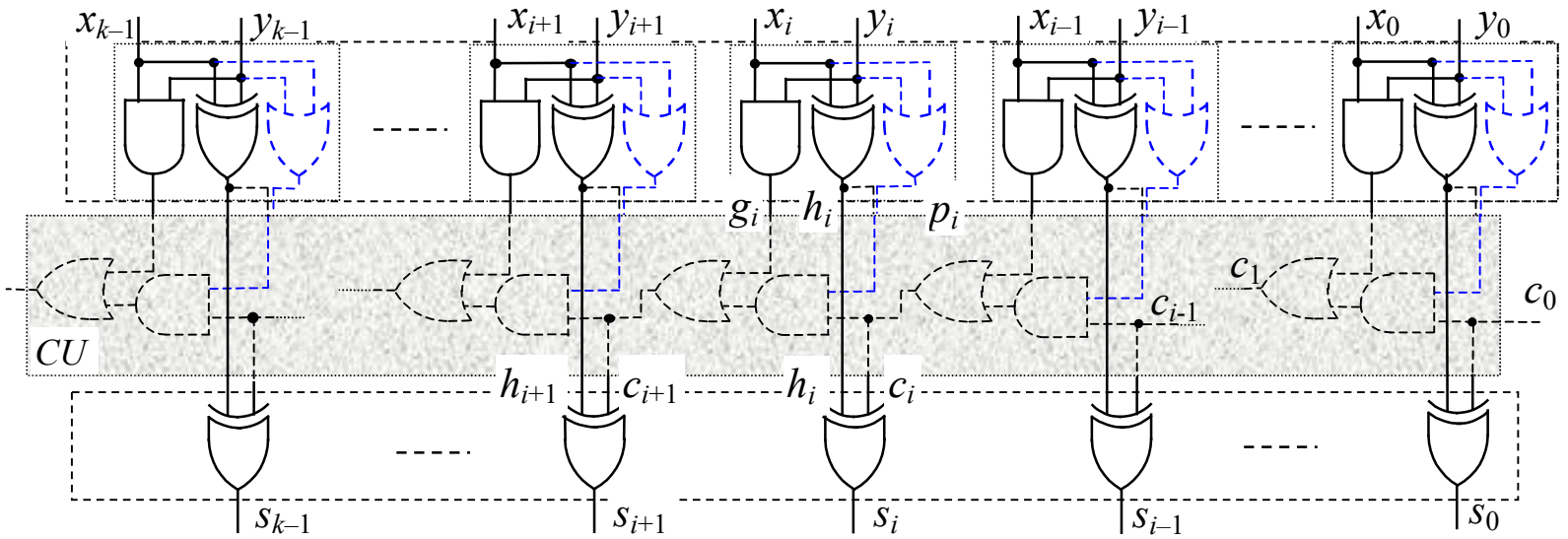

x
y
c_+
FS /Σ_
c_-
s


x
y
AT = 1
A = 2
T = 2
g
p = $h = \bar{x}⊕y$
c_+
c_-
AT = 1
AT = 1
$c_+ = \bar{x}y + (\bar{x}⊕y)c_-$
A = 2
T = 2
A = 7
T_s = 4
T_c = 4
$s = \overline{h⊕c_-}$
$= x⊕y⊕c_-$

Sumator kaskadowy

W podstawowym układzie dodawania/odejmowania można wyróżnić 3 bloki:

- blok wejściowy do równoległego tworzenia funkcji pomocniczych h_i, g_i, p_i(..=h_i)
- blok wytwarzania przeniesień c_i
- blok tworzenia sum s_i



Sumator kaskadowy RCA (ang. *Ripple-Carry Adder*) – sekwencyjne tworzenie przeniesień

Sumatory równoległe (W7)

Przyspieszanie dodawania dwuargumentowego

Przyśpieszanie wytwarzania przeniesień

- antycypacja przeniesień (*carry look-ahead adder*, CLA)
  – tworzenie przeniesień dla kilku (zwykle 4) sąsiednich pozycji
- równoległe wytwarzanie przeniesień (*parallel prefix adder*, PPA), albo
  - korekcja sum tymczasowych aktualizowanym przeniesieniem (ELM)
- skracanie ścieżki propagacji przeniesienia (*carry skip adder*, CSKA)

Składanie sum tymczasowych

- sumator z przełączaniem sum częściowych (*carry-select adder*, CSLA)
  – równoległe wytwarzanie alternatywnych blokowych sum częściowych
  - składanie sum warunkowych (*conditional sum adder*, COSA)
    – tworzenie wariantowych sum dla bloków 2^i kolejnych pozycji
- korekcja półsum (*carry-increment adder*, CIA)
  – korekcja sum blokowych przeniesieniami

Dodawanie bez przeniesień (carry-free)

- *wykorzystanie nadmiarowej reprezentacji argumentów (np. kod SD)* − tworzenie sum tymczasowych w zawężonym zakresie i opóźniona korekcja

Powiązania przeniesień/pożyczek

Rekurencyjną zależność kolejnych przeniesień / pożyczek:

$$c_{i+1} = g_i + p_i c_i = g_i + p_i(g_{i-1} + p_{i-1}c_{i-1}) = g_i + p_i(g_{i-1} + p_{i-1}(g_{i-2} + p_{i-2}c_{i-2})$$

można opisać za pomocą *podstawowego operatora przeniesień* (*FCO*):

$$(c_{i+1>j}, \ldots) = (g_i, p_i) \circ \ldots \circ (g_{j+1}, p_{j+1}) \circ (g_j, p_j) \circ (c_j, \ldots)$$

gdzie $(a,b) \circ (e,f) = (a + be, bf)$.

Operator ten jest obustronnie **łączny** (ang. *associative*) lecz nie jest przemienny:

$$[(x,y) \circ (q,r)] \circ (a,b) = (x + yq, yr) \circ (a,b) = (x + yq + yra, yrb),$$
$$(x,y) \circ [(q,r) \circ (a,b)] = (x,y) \circ (q + ra, rb) = (x + yq + yra, yrb).$$

Rekurencyjnie powiązane funkcje $(G_{i:j}, P_{i:j}) = (g_i, p_i) \circ (g_{i-1}, p_{i-1}) \circ \ldots \circ (g_j, p_j)$ opisują:

- warunki wymuszania przeniesienia/pożyczki $c_{i+1} = 1$ (funkcja $G_{i,j}$)
- warunki propagacji ($c_{i+1} = c_j$) przeniesienia/pożyczki (funkcja $P_{i,j}$)

Ale każde przeniesienie zależy od c_0, więc (*argument ... jest nieistotny*, może być 0)

$$(c_{i+1}, \ldots) = (g_i, p_i) \circ \ldots \circ (g_2, p_2) \circ (g_1, p_1) \circ (g_0, p_0) \circ (c_0, \ldots) = (G_{i:0} + P_{i:0}c_0, \ldots)$$

Tworzenie przeniesień

Z uwagi na łączność *operatora przeniesień* (operatora Brenta-Kunga) mamy:

$$(c_{i+1},0) = (g_i,p_i)\circ\ldots\circ(g_3,p_3)\circ(g_2,p_2)\circ(g_1,p_1)\circ(g_0,p_0)\circ(c_0,0) = (G_{i:0}+P_{i:0}c_0,0) =$$
$$= (g_i,p_i)\circ\ldots\circ(g_3,p_3)\circ(g_2,p_2)\circ(g_1,p_1)\circ(g_0+p_0c_0,0) = (G^*_{i:0},0)$$

a poszczególne przeniesienia można wytwarzać:

– **sekwencyjnie** (po kolei) – struktura RC (ang. *Ripple-Carry*):

$(c_1,0) = (g_0,p_0)\circ(c_0,0)$, $(c_2,0) = (g_1,p_1)\circ(c_1,0)$, $(c_3,0) = (g_2,p_2)\circ(c_2,0)$,...

– **jednocześnie w grupach** po k pozycji – struktura CL (ang. *Carry Lookahead*):

$(c_{i+1},0) = (g_i,p_i)\circ(c_i,0)$,

$(c_{i+2},0) = (g_{i+1},p_{i+1})\circ(g_i,p_i)\circ(c_i,0)$,

...

$(c_{i+k},0) = (g_{i+k-1},p_{i+k-1})\circ\ldots\circ(g_{i+1},p_{i+1})\circ(g_i,p_i)\circ(c_i,0)$,

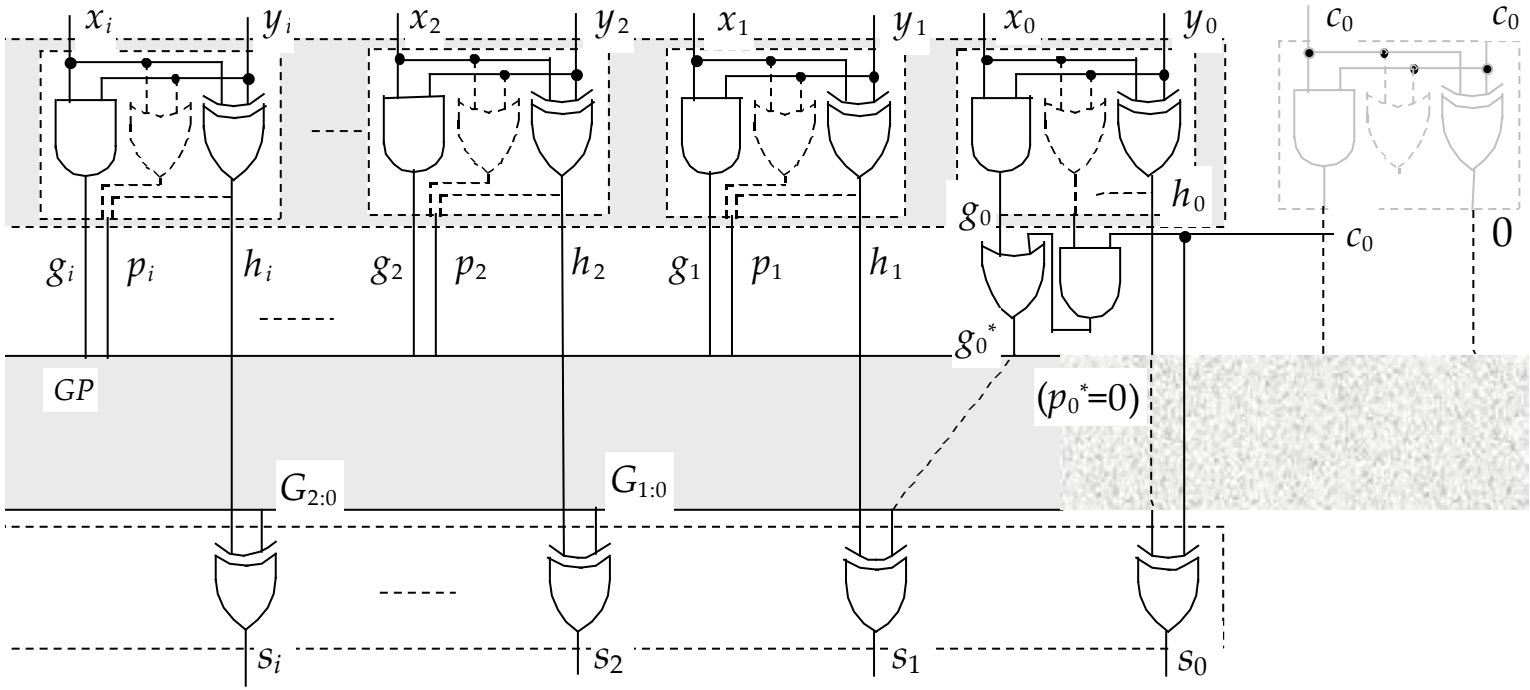
– **w pełni równolegle** – struktura prefiksowa PP (ang. *Parallel Prefix*), np. tak:

$$(c_{i+1},0) = (g_i,p_i)\circ\ldots\circ[(g_5,p_5)\circ(g_4,p_4)]\circ[(g_3,p_3)\circ(g_2,p_2)]\circ[(g_1,p_1)\circ(g_0,p_0)]\circ(c_0,0) =$$
$$= (g_i,p_i)\circ\ldots\circ[(g_5,p_5)\circ(g_4,p_4)]\}\circ\{[(g_3,p_3)\circ(g_2,p_2)]\circ[(g_1,p_1)\circ(g_0,p_0)]\}\circ(c_0,0)$$

albo:

$$(c_{i+1},0) = (g_i,p_i)\circ\ldots\circ(g_5,p_5)]\circ[(g_4,p_4)\circ(g_3,p_3)]\}\circ\{[(g_2,p_2)\circ(g_1,p_1)]\circ[(g_0,p_0)\circ(c_0,0)]\}$$

Sumator prefiksowy z redukcją rozgałęzienia c_0



Jeśli c_0 jest przetworzone w bloku wstępnym, to $c_i = G_{i-1:0}$. Są dwa schematy:

A) $(c_\#, 0) = \ldots \circ \{(g_3, p_3) \circ (g_2, p_2)\} \circ \{(g_1, p_1) \circ [(g_0, p_0) \circ (c_0, 0)]\}$

B) $(c_\#, 0) = \ldots \circ (g_3, p_3)] \circ [(g_2, p_2) \circ (g_1, p_1)] \circ [(g_0, p_0) \circ (c_0, 0)]\}$

Dodatkowa korzyść - uproszczone wykrywanie nadmiaru U2:

$$ov = G_{n-1:0} \oplus G_{n-2:0} = (g_{n-1} + p_{n-1} G_{n-2:0}) \oplus G_{n-2:0} = \overline{g}_{n-1} G_{n-2:0}$$

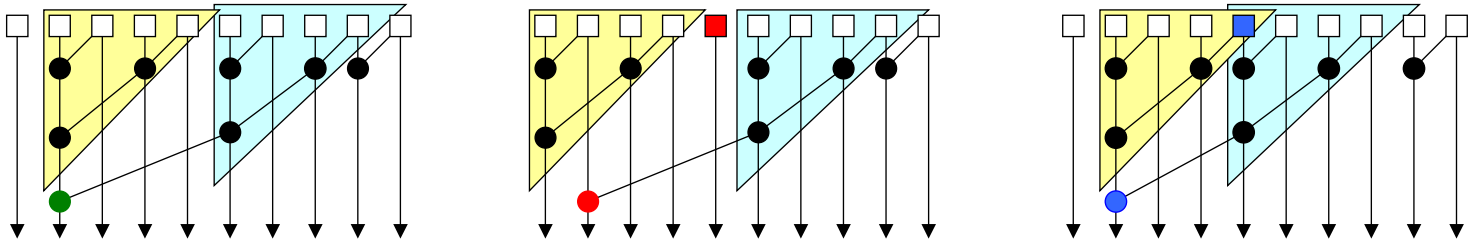
Zasady konstrukcji sieci prefiksowej GP

Węzeł sieci *GP* realizuje **dwie** funkcje (G_{HL}, P_{HL}) **czterech** zmiennych (G_H,P_H,G_L,P_L), przy tym H=*i*:*k*+1, P=*k*:*j*, HL=*i*:*j*, *i*>*k*≥*j*

$$(G_{HL}, P_{HL}) = (G_H, P_H) \circ (G_L, P_L) = (G_H + P_H G_L, P_H P_L)$$

Zasady tworzenia struktury *GP* integrującej funkcje G_H, P_H oraz G_L, P_L

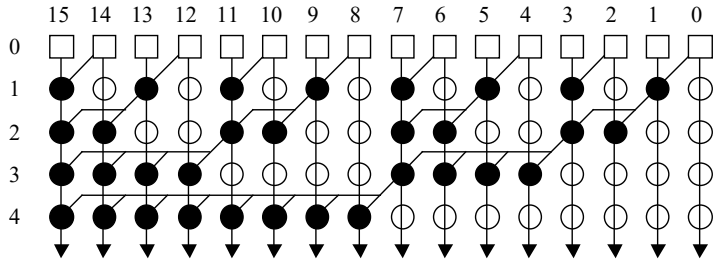
- bloki H i L **powinny być rozdzielnie sąsiadujące**
- bloki H i L **nie mogą być rozdzielone innym blokiem**
- bloki H i L *mogą mieć część wspólną* – funkcje G_{HL} i P_{HL} są **nadmiarowe**



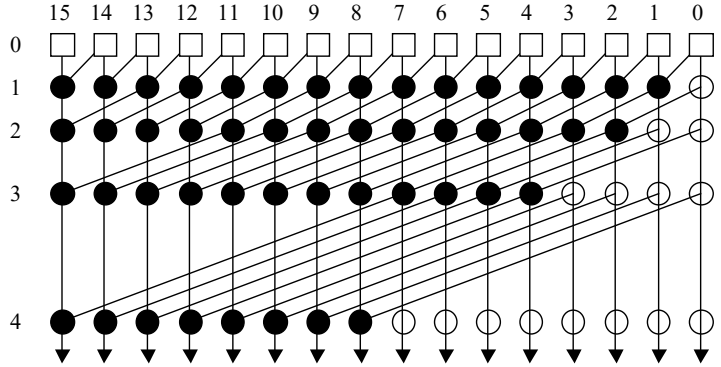
graf optymalny **graf błędny** graf poprawny

- □ – wytwarzanie $G_{i:i}=g_i$ i $P_{i:i}=p_i$, ● – operator: $(G_{HL}, P_{HL}) = (G_H, P_H) \circ (G_L, P_L)$
- regularne struktury dla $n=2^k$ wejść (pozycji),
- w innych przypadkach przyjąć $k=\text{int}(1+\log_2 n)$ i usunąć zbędne gałęzie (sieć integrującą 2^{k-1} pozycji połączyć siecią integrującą pozostałe wejścia)

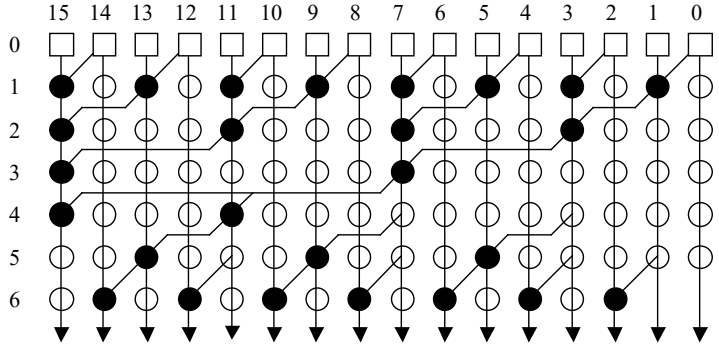
Grafy sieci równoległego generowania i propagacji przeniesienia (PPA)



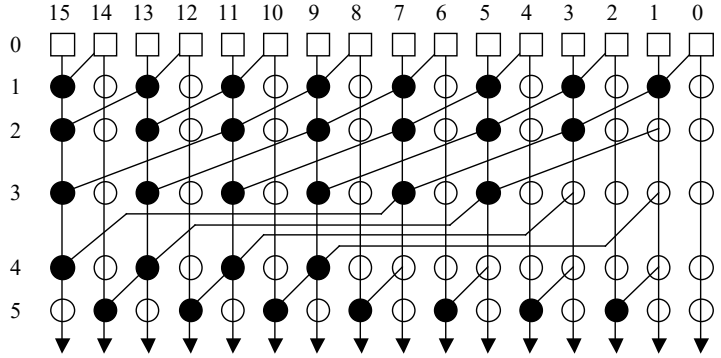
Graf prefixowy (Sklansky / Ladner-Fischer) (16 b)



Graf prefixowy (Kogge & Stone) (16 b)



Graf prefixowy (Brent–Kung) (16 b)



Graf prefixowy – (Han & Carlson) (16 b)

□ – wytwarzanie funkcji $G_{i:i}=g_i$ oraz $P_{i:i}=p_i$ ○ – przekazywanie G, P bez zmiany

● – operator prefiksowy $(G_{HL},P_{HL})=(G_H,P_H)\circ(G_L,P_L)$

Dodawanie wielu liczb (W8)

Dodawanie wielu liczb naturalnych

Ponieważ dodawanie jest łączne i przemienne, więc ($r = \lfloor \log_\beta n \rfloor - 1$

$$\sum_{s=1}^{n}\sum_{i=0}^{k-1} x_{s,i}\beta^i = \sum_{i=0}^{k-1}\beta^i \sum_{s=1}^{n} x_{s,i} = \sum_{i=0}^{k-1}\beta^i (v_{r,i}\beta^r + \ldots + v_{1,i}\beta + v_{0,i})$$

Sumę n liczb k-cyfrowych można więc obliczyć przez dodawanie tworzonych niezależnie k liczb r-cyfrowych $(v_{r,i}\beta^r + \ldots + v_{1,i}\beta + v_{0,i})\beta^i$, $r = \lfloor \log_\beta n \rfloor - 1$.

Procedurę można wykonać etapami, np. ograniczając rozmiar sum pośrednich.

Największą liczbą liczb 1-cyfrowych, których suma jest najwyżej 2-cyfrowa to β+1, bo $(\beta+1)(\beta-1)=\beta^2-1$. 2. Wartość sumy nie więcej niż β+1 liczb jednocyfrowych w podstawie β jest liczbą dwucyfrową $\{u_{i+1}\ v_i\}$ o wartości:

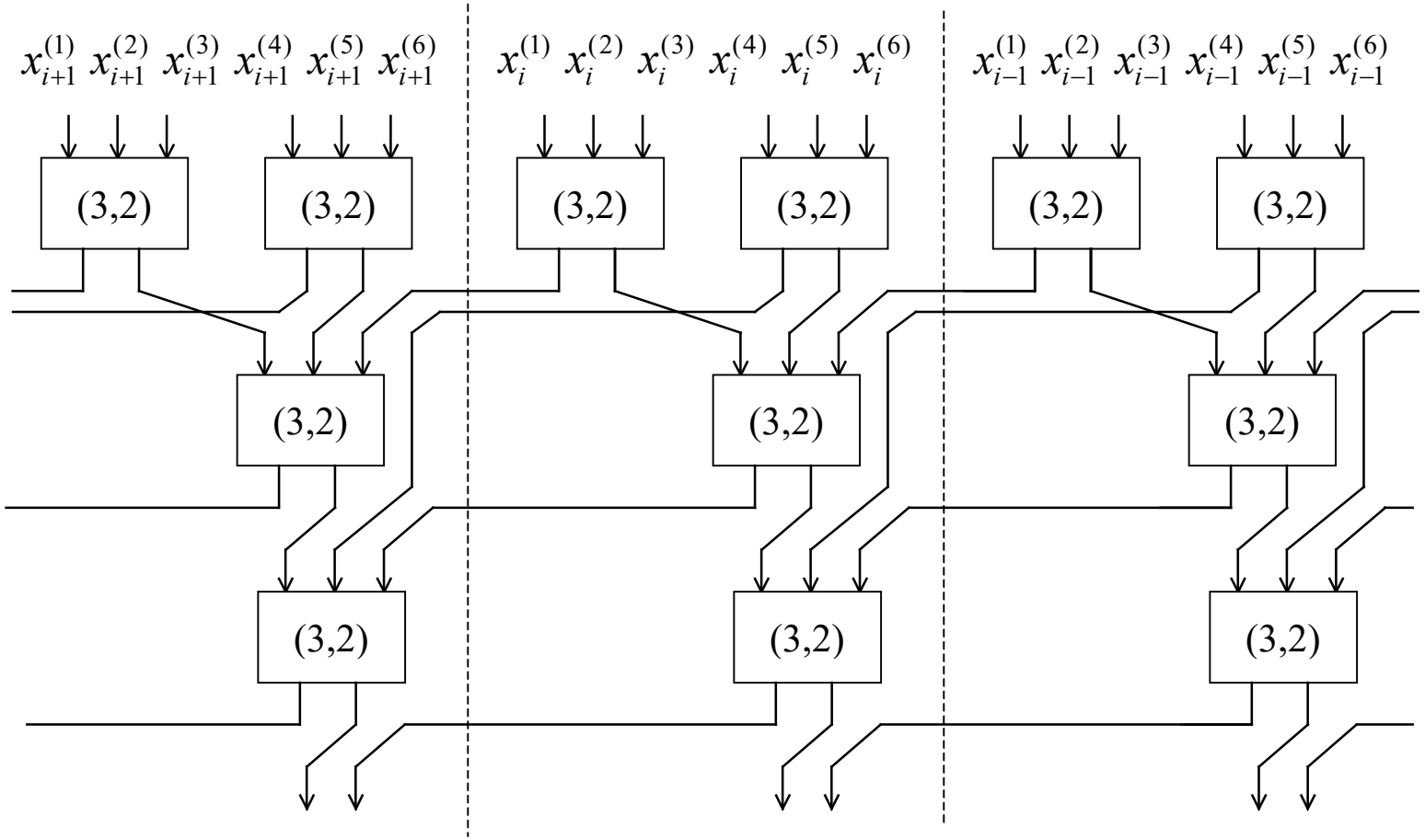
$$r_i = \sum_{j=1}^{\beta+1} x_{i,j} = u_{i+1}\beta + v_i \le \beta^2 - 1$$

gdzie $v_i = r_i \bmod \beta$, $u_{i+1} = r_i \operatorname{int} \beta$

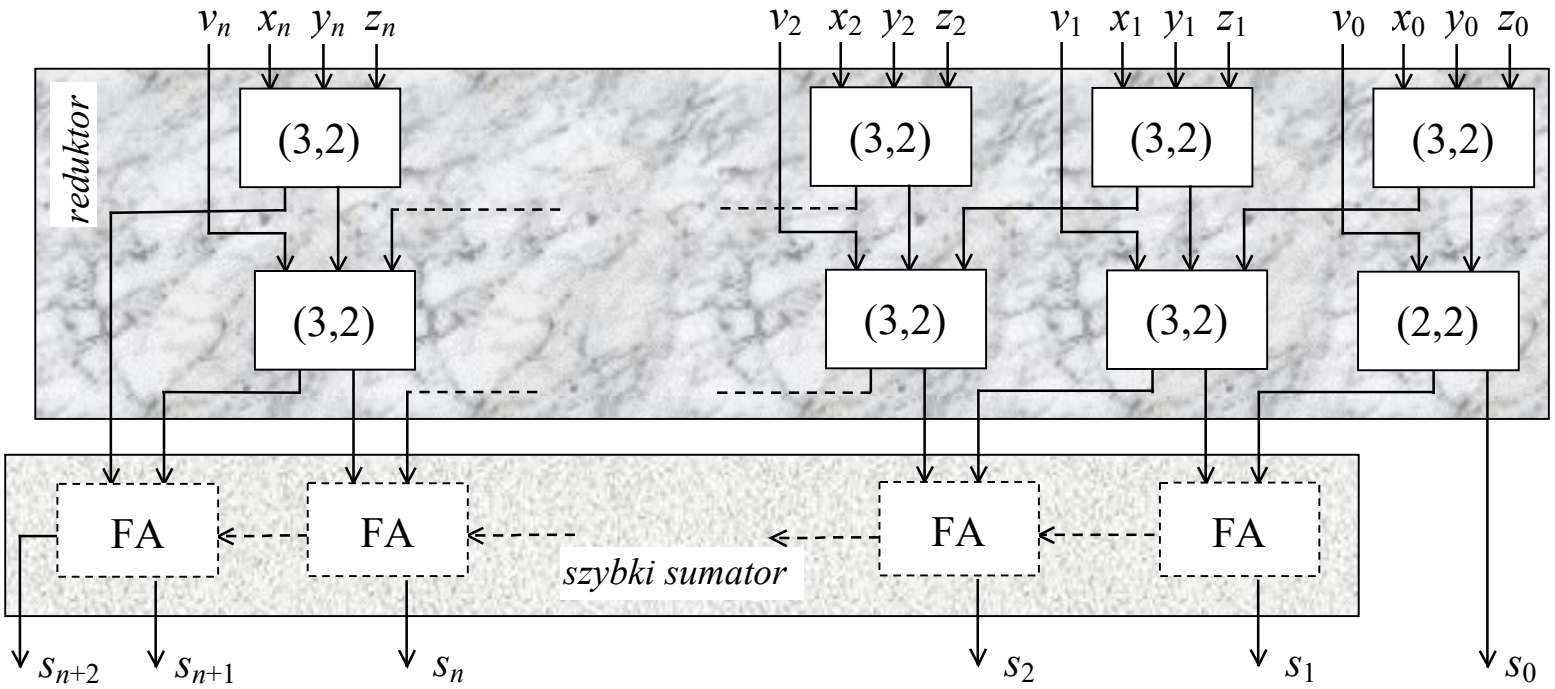
W systemie dwójkowym β+1=3, ale jest gotowy układ, który tworzy 2-cyfrową sumę 3 liczb 1-bitowych − jest to *elementarny sumator dwójkowy*.

Reduktor CSA

Reduktor przetwarza problem dodawania k liczb n-bitowych do dodawania 2 liczb.



Dwójkowe sumatory wieloargumentowe (CSA)



Sumator czterooperandowy CSA

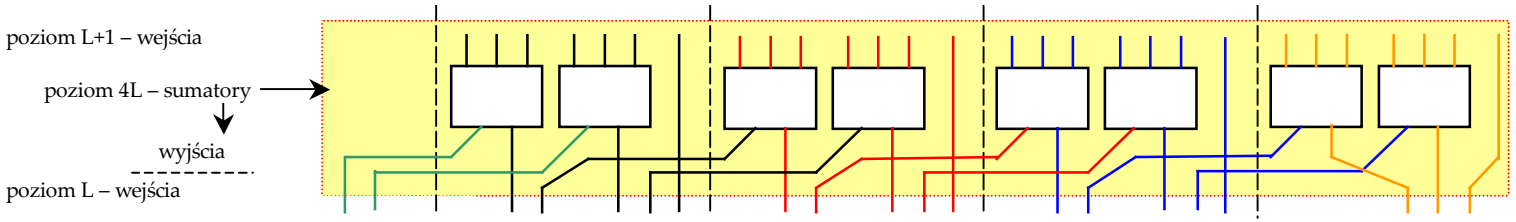
czas dodawania = czas redukcji + czas dodawania końcowego

Konstrukcja reduktora CSA

Dopóki w jakiejś kolumnie pozostało więcej niż 2 sygnały

- każde 3 sygnały wejściowe **o tej samej wadze** przyłącz do wejść modułu (3,2)
- sygnał nieprzyłączony przekaż na niższy poziom CSA lub opcjonalnie parę sygnałów **o tej samej wadze** przekształć przez półsumator HA (układ (2,2))
- wytwórz wyjścia wszystkich modułów (3,2) (lub (2,2))
  – pamiętaj, że wyjścia *s* (sumy) i *c* (przeniesienia) mają **różne wagi**!
- w poszczególnych kolumnach zbierz sygnały **o jednakowych wagach**

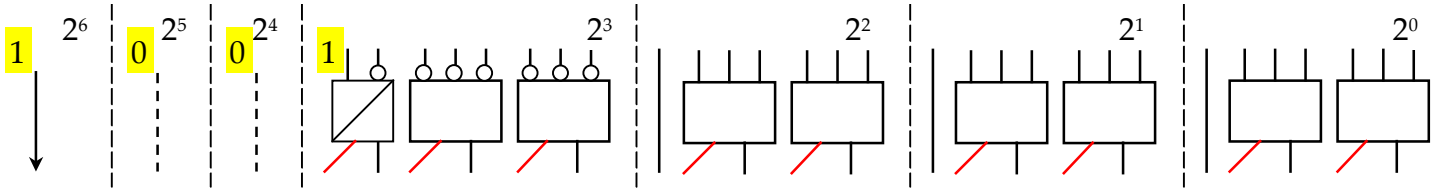
UWAGA: Zwrotne przekazanie sygnału na poprzedni poziom redukcji jest sprzeczne z ideą szybkiej redukcji argumentów − jest to **poważny błąd**



Schemat konstrukcji – sygnały tej samej wagi mają taki sam kolor

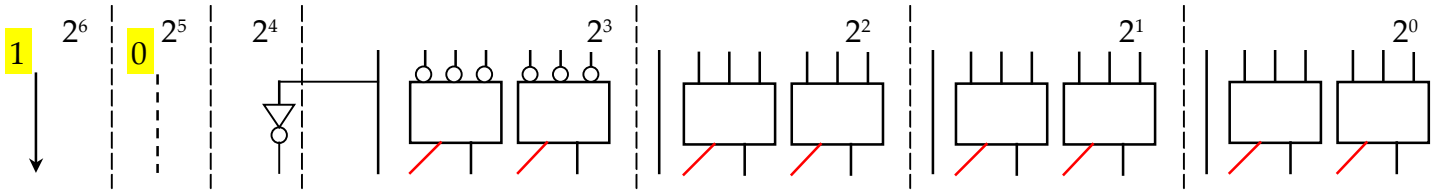
Dwójkowe sumatory wieloargumentowe kodu U2 (2)

Przekodowanie każdego argumentu $\{x_{n-1}, x_{n-2}, ..., x_1, x_0\}$ danego w kodzie U2 na sumę liczby naturalnej o reprezentacji $\{\overline{x}_{n-1}, x_{n-2}, ..., x_1, x_0\}$ i ujemnej stałej -2^n pozwala zastąpić użycie rozszerzeń lewostronnych przez dopełnianie (negację) wiodących bitów argumentów i dodanie stałej korekcyjnej $-k \cdot 2^{n-1}$:



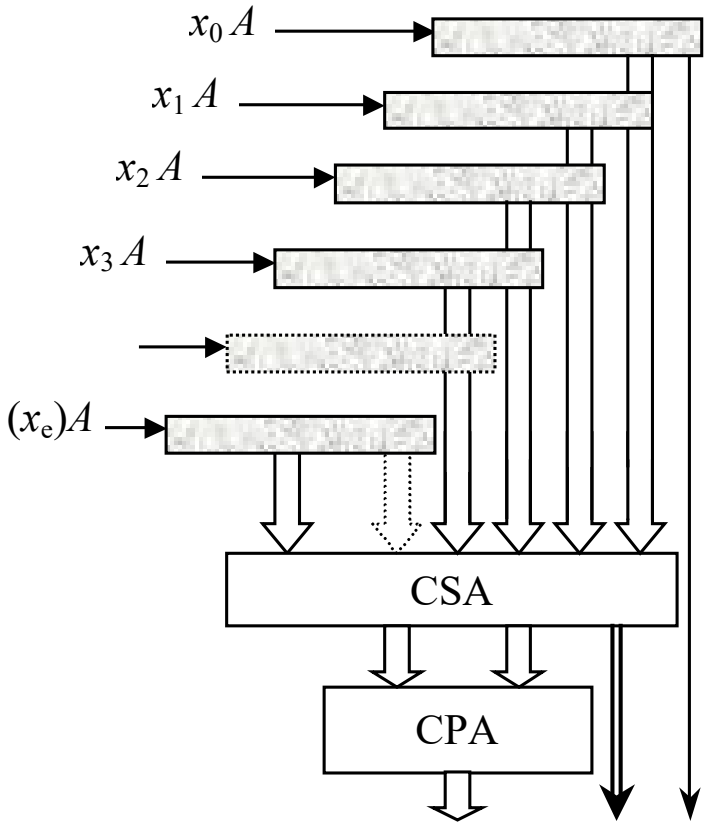
Modyfikacja drzewa CSA do dodawania k=7 liczb n=4-bitowych w kodzie U2

UWAGA: Ponieważ $...01...11 + ...00x = ... \, x \, \overline{x} \, ... \overline{x} \, \overline{x}$, więc schemat można uprościć:

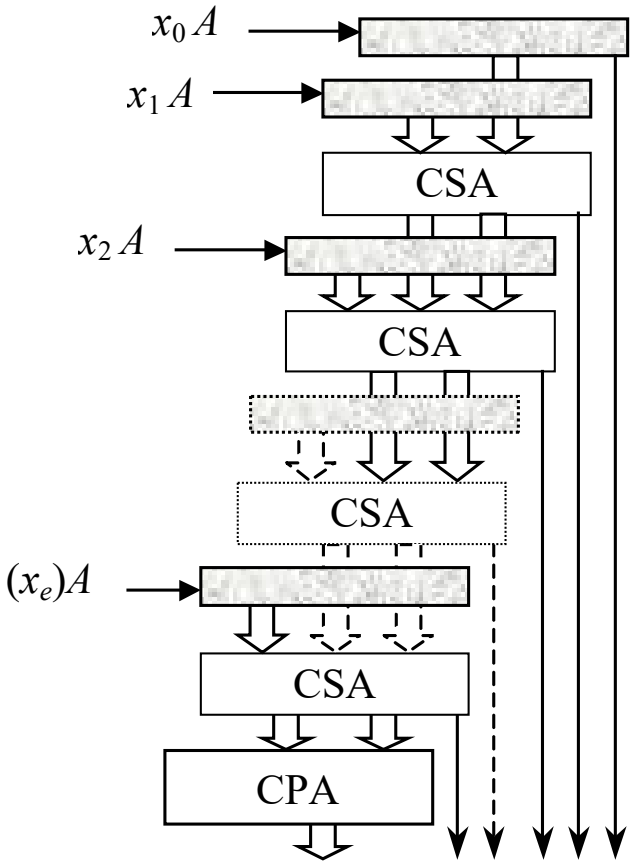


Uproszczone drzewo CSA do dodawania k=7 liczb n=4-bitowych w kodzie U2

Szybka akumulacja iloczynów częściowych

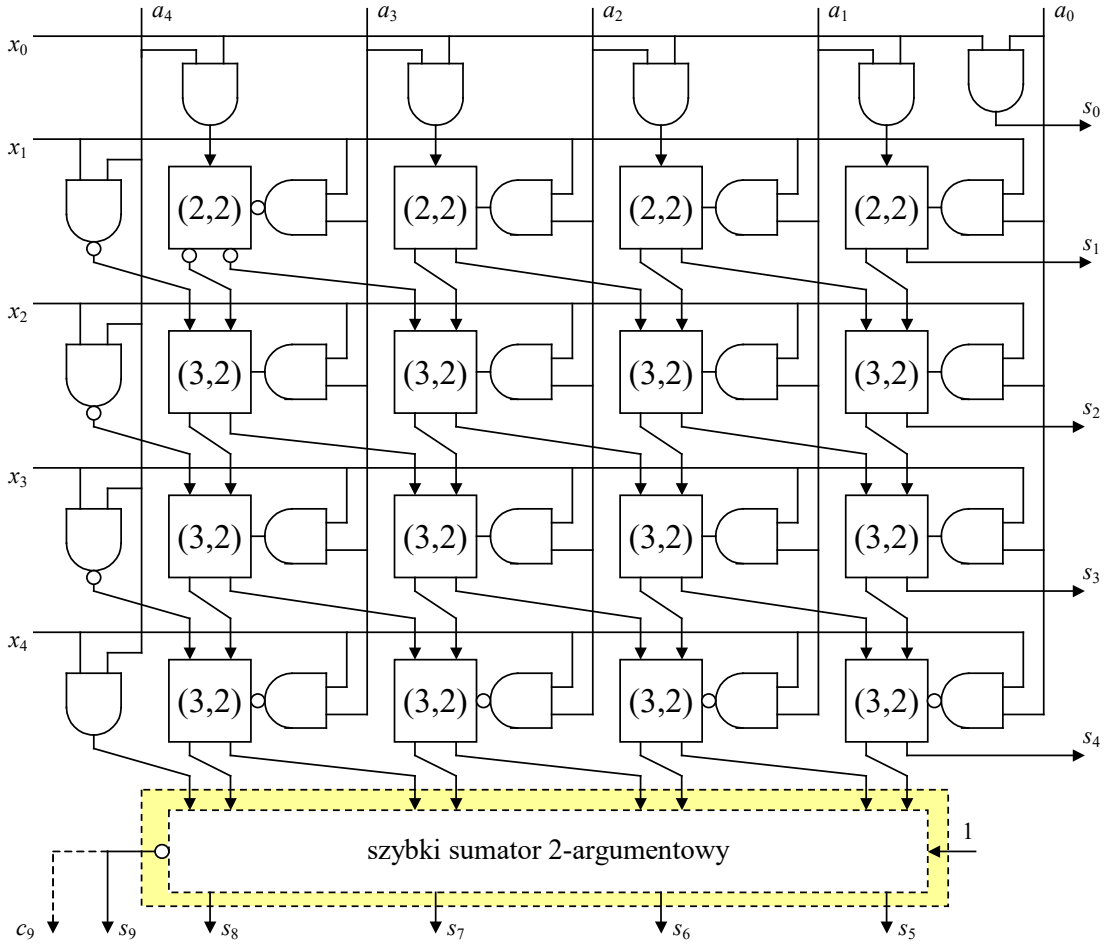


akumulacja równoległa
(drzewo Wallace'a)



akumulacja sekwencyjna
(matryca mnożąca)

Matryca mnożąca kodu uzupełnieniowego (Baugh'a-Wooley'a)



(ostatni iloczyn częściowy: negacja bitów mnożnej i korekcja)

Arytmetyka resztowa (W9)

Twierdzenie Eulera

Twierdzenie (Fermata)

Niech p będzie liczbą pierwszą. Jeśli p nie jest podzielnikiem liczby a, to wtedy $a^{p-1} \equiv 1 \pmod{p}$ zaś dla dowolnego a zachodzi $a^p \equiv a \pmod{p}$.

Funkcja Eulera $\varphi(N)$ (totient)

... co druga naturalna jest podzielna przez 2, co trzecia z pozostałych dzieli się przez 3, co piąta z niepodzielnych przez 2 lub 3 dzieli się przez 5, etc.

Lemat

Liczb naturalnych mniejszych od $N = p_1^{e_1} p_2^{e_2} \ldots p_m^{e_m}$, $p_i \in \mathbb{P}$ i względnie pierwszych z tą liczbą (ang. *totatives*) jest

$$\varphi(N) = \prod_{i=1}^{i=m} (p_i - 1) p_i^{e_i - 1}, \quad p_i \in \mathbb{P}$$

Twierdzenie (Eulera)

Jeśli $\varphi(N)$ jest liczbą liczb mniejszych od N i względnie pierwszych z N, to

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

Twierdzenie Carmichaela

Najmniejszą potęgę taką, że $a^r \bmod N=1$ nazywa się *rzędem* liczby a modulo N.

Twierdzenie (Carmichaela)

Maksymalny rząd modulo N elementu a takiego, że $NWD(a,N)=1$, wynosi:

$$\lambda(N)=\lambda(p_1^{e_1}p_2^{e_2}\ldots p_m^{e_m})=NWW(\lambda(p_1^{e_1}),\lambda(p_2^{e_2}),\ldots,\lambda(p_m^{e_m}))\le\varphi(N)$$

gdzie: $\lambda(p)=\varphi(p)=p-1$, $\lambda(p^m)=\varphi(p^m)$ dla $p\ge 3$, $\lambda(2^q)=\varphi(2^{q-1})$ dla $q\ge 3$, $\lambda(4)=2$.

Wniosek (ulepszenie twierdzenia Eulera)

Jeśli $\lambda(N)$ jest *rzędem* liczby a modulo N, to

$$a^{\lambda(N)} \bmod N = 1$$

Potęgowanie modulo

W obliczaniu reszt potęgę można redukować modulo $\varphi(p)$ lub $\lambda(p)$:

$$a^x \bmod N=(a^{\lambda(N)})^{[x \operatorname{int} \lambda(N)]}a^{x \bmod \lambda(N)} \bmod N = a^{x \bmod \lambda(N)} \bmod N.$$

Jeśli znana jest binarna reprezentacja $\{\ldots, x_i, x_{i-1},\ldots, x_1, x_0\}$ liczby naturalnej x, to w obliczeniu reszty potęgi możemy też wykorzystać równość:

$$a^{\sum x_i 2^i} \bmod N=[(a \bmod N)^{x_0}(a^2 \bmod N)^{x_1}(a^{2^2} \bmod N)^{x_2}\ldots]\bmod N$$

Chińskie twierdzenie o resztach

CHIŃSKIE TWIERDZENIE O RESZTACH (CRT) (*SUN-TZU, III W., QIN JIUSHAO, 1247*)

Niech $\mathbf{W}=\{w_1,w_2,...,w_n: \quad \forall i\neq j: \quad NWD(w_i,w_j)=1\}$, $W = w_1w_2...w_n$. *Reprezentacja* $\langle x_1, x_2, \ldots, x_n: x_i = X \bmod w_i,\ w_i \in \mathbf{W}\rangle$ *każdej liczby* $0 \le X < W$ *jest unikatowa oraz*

$$X = |\mathbf{X}| = \left(\sum_{s=1}^{n} \hat{w}_s (\hat{w}_s^{-1} \bmod w_s) x_s \right) \bmod W$$

gdzie $\hat{w}_s = W w_s^{-1}$, zaś $\hat{w}_s^{-1} \bmod w_s$ *– odwrotność* $\hat{w}_s$ *względem modułu* w_s.

DOWÓD (*nieformalny szkic dowodu konwersji odwrotnej*).
Ze względu na zachowawczość kongruencji wobec dodawania mamy

$$\langle x_1, x_2, \ldots, x_n \rangle = x_1 \cdot \langle 1,0,\ldots,0,0\rangle + x_2 \cdot \langle 0,1,\ldots,0,0\rangle + \ldots + x_n \cdot \langle 0,0,\ldots,0,1\rangle .$$

W systemie $RNS(w_1,w_2,\ldots,w_m)$ liczba p_s o reprezentacji $\langle 0,\ldots,0,1_s,0,\ldots,0\rangle$ jest podzielna przez każde w_i oprócz w_s, jest więc $p_s = k \cdot \hat{w}_s$ (liczby p_s istnieją, bo różnych reprezentacji jest dokładnie W). Ponieważ jej reszta względem w_s jest równa 1, więc $k = \hat{w}_s^{-1} \bmod w_s$ jest odwrotnością $\hat{w}_s$ oraz $p_s = \hat{w}_s (\hat{w}_s^{-1} \bmod w_s)$. $\langle x_1, x_2, \ldots, x_n \rangle$ jest więc reprezentacją liczby $(x_1p_1 + x_2p_2 + \ldots + x_np_n) \bmod W$. ❐

Algorytm Euklidesa

Twierdzenie:

Dla dowolnych liczb naturalnych n i m: $NWD(n,m) = NWD(m, n \bmod m)$, przy tym istnieją l. całkowite u, v takie, że $NWD(n,m) = un + vm$ (kombinacja liniowa m n).

Z twierdzenia Euklidesa wynika rekurencyjna liniowa zależność kolejnych reszt:

$$r_{i+1} = r_{i-1} \bmod r_i = r_{i-1} - q_{i+1} r_i$$

gdzie $q_{i+1} = r_{i-1} \operatorname{int} r_i$ – iloraz całkowity, $r_{-1} = n$, $r_0 = m$, więc w konsekwencji:

$$NWD(n,m) = B_i r_{i-1} + A_i r_i$$

Współczynniki skalujące są również powiązane rekurencyjnie:

$$B_i r_{i-1} + A_i r_i = B_i r_{i-1} + A_i (r_{i-2} - q_i r_{i-1}) = A_i r_{i-2} + (B_i - q_i A_i) r_{i-1}$$

skąd widać, że $A_i = B_{i-1}$ i $A_{i-1} = B_i - A_i q_i$, więc $B_{i-2} = B_i - B_{i-1} q_i$ albo $A_{i-1} = A_{i+1} - A_i q_i$

Jeśli więc $r_{k+1} = 0$, to $r_k = NWD(n,m) = r_{k-2} - q_k r_{k-1} = A_k r_{k-2} + A_{k-1} r_{k-1}$, skąd $A_k = 1$, $A_{k-1} = -q_k$, a *równanie diofantyczne* $NWD(n,m) = un + vm$ ma postać:

$$NWD(n,m) = A_1 r_{-1} + A_0 r_0 = A_1 n + A_0 m$$

Obliczanie odwrotności

W konwersji reprezentacji resztowej na skalarną potrzebne jest obliczenie odwrotności modularnych.

Wniosek z twierdzenia Euklidesa

Jeśli $NWD(n,m)$=1, to w równaniu diofantycznym

$$NWD(n,m) = A_1 r_{-1} + A_0 r_0 = A_1 n + A_0 m$$

$A_1 = n^{-1} \bmod m$ oraz $A_0 = m^{-1} \bmod n$

Współczynniki A_i można obliczyć na podstawie zależności rekurencyjnej $A_{i-1} = A_{i+1} - A_i q_i$, gdzie $A_k = 1$, $A_{k-1} = -q_k$, a współczynniki q_i są kolejnymi ilorazami w procedurze obliczania NWD.

Wniosek z twierdzenia Carmichaela

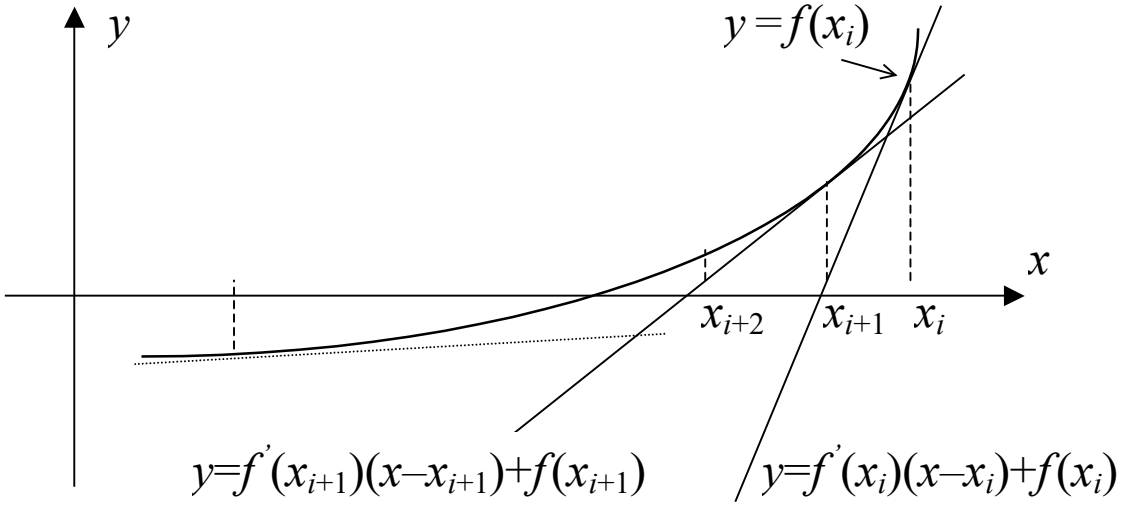
Jeśli odwrotność istnieje ($NWD(a,N)$=1, można ją obliczyć przez potęgowanie:

$a^{-1} \bmod N = a^{\lambda(N)-1} \bmod N$.

Elementarne metody numeryczne (W10)

Metoda Newtona-Raphsona

metoda iteracyjna Newtona-Raphsona



kolejne przybliżenia miejsca zerowego $f(x)$ określa równanie rekurencyjne

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)},$$

Biorąc funkcję $f(x) = x^{-1} - D$ otrzymujemy

$$x_{i+1} = x_i(2 - Dx_i)$$

jako kolejne przybliżenia *odwrotności dzielnika* D

Obliczanie pierwiastka i odwrotności pierwiastka kwadratowego

Liczba pierwiastkowana jest znormalizowana $\frac{1}{4} \le A < 1$.

metoda iteracyjna Newtona – równanie rekurencyjne: $x_{i+1} = x_i - \dfrac{f(x_i)}{f'(x_i)}$

Obliczanie pierwiastka kwadratowego:

Jeśli $f(x)=x^2-A$, to $x=sqrt(A)$ $(\sqrt{A})$ i wtedy $f'(x)=2x$, więc

$$x_{i+1} = x_i - \frac{x_i^2 - A}{2x_i} = \tfrac{1}{2} x_i + \frac{\frac{1}{2} A}{x_i}$$

wada: konieczność dzielenia

Obliczanie odwrotności pierwiastka kwadratowego:

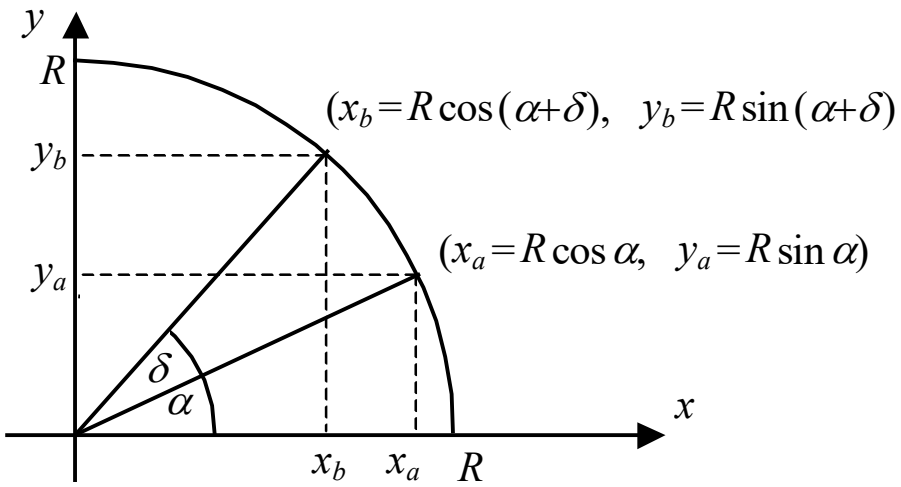
$f(x) = x^{-2} - A$ i wtedy $f'(x) = -2x^{-3}$ oraz

$$x_{i+1} = x_i - \frac{(x_i^{-2} - A)}{-2x_i^{-3}} = \tfrac{1}{2} x_i (3 - x_i^2 A)$$

CORDIC – algorytm Voldera

J.Volder (1959, sterowanie samolotu B-58)

Obrót wektora zaczepionego w punkcie (0,0) przestrzeni kartezjańskiej



Z tożsamości trygonometrycznych

$$\cos(\alpha+\delta) = \cos\alpha\cos\delta - \sin\alpha\sin\delta$$

$$\sin(\alpha+\delta) = \sin\alpha\cos\delta + \cos\alpha\sin\delta$$

wynika, że w wyniku obrotu wektora o kąt δ punkt (x_a, y_a) przemieści się do:

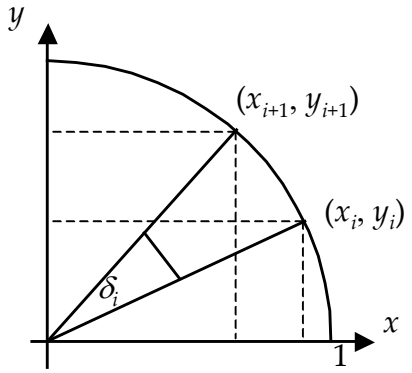
$$x_b = R\cos(\alpha+\delta) = R\cos\alpha\cos\delta - R\sin\alpha\sin\delta = (x_a - y_a\tan\delta)\cos\delta$$

$$y_b = R\sin(\alpha+\delta) = R\cos\alpha\sin\delta + R\sin\alpha\cos\delta = (x_a\tan\delta + y_a)\cos\delta$$

CORDIC – obrót wektora

Obrót o kąt $\delta = \operatorname{arctg} t$ jest sumą obrotów o kąty $\delta_i = \operatorname{arctg} t_i$. Najlepszą zbieżność można uzyskać biorąc $t_i = \pm 2^{-i} = d_i 2^{-i}$ (ang. *non-restoring decomposition*).

Wynikiem przesunięcia punktu (x_i, y_i) na okręgu o kąt $\operatorname{arctg} t_i$ jest:



$$\begin{aligned} \sqrt{1+t_i^2}\; x_{i+1} &= x_i - t_i y_i \\ \sqrt{1+t_i^2}\; y_{i+1} &= t_i x_i + y_i \end{aligned}, \quad i = 0, 1, \ldots$$

$$(\cos^{-1}\delta = \sqrt{1+\tan^2\delta})$$

$$(\sqrt{1+t_i^2} = \cos\arctan t_i)$$

Obie współrzędne są jednakowo skalowane, więc obliczenia można uprościć:

$$x_{i+1}^* = x_i^* - t_i y_i^* \quad \text{oraz} \quad y_{i+1}^* = y_i^* + t_i x_i^*$$

gdzie $x_0^* = x_0$, $y_0^* = y_0$, dokonując korekty w ostatnim kroku obliczeń, bo

$$x_n^* = x_n \prod_{i=0}^{n-1} \sqrt{1+t_i^2}, \quad y_n^* = y_n \prod_{i=0}^{n-1} \sqrt{1+t_i^2}\,.$$

Liczbę kroków iteracji może być określona przez odległość $\delta - \sum_{i=0}^{n} d_i \arctan 2^{-i}$

CORDIC – tryb obrotu

W trybie obrotu (ang. *rotation mode*) oblicza się wartości funkcji trygonometrycznych. Na podstawie wartości zmiennej pomocniczej z (odległości od kąta docelowego):

$$z_{n+1} = z_n - \arctan t_i = z_n - d_i \arctan 2^{-i}$$

określa się wartość $d_i = z_i / |z_i|$ (1 lub −1), a krokiem iteracji jest

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -d_i 2^{-i} \\ d_i 2^{-i} & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

Jeśli $|z_0| \le \sum_{i=0}^{\infty} 2^{-i} = 1{,}74328662047234000 35...$, to

$$\lim_{n\to\infty} \begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix} = K \cdot \begin{pmatrix} x_0 \cos z_0 - y_0 \sin z_0 \\ x_0 \sin z_0 + y_0 \cos z_0 \\ 0 \end{pmatrix}$$

gdzie $K = \prod_{i=0}^{\infty} \sqrt{1 + 2^{-2i}} = 1{,}64676025812106564836 6051...$

Biorąc $x_0 = K^{-1} = 0{,}60725293500888125616 94...$, $y_0 = 0$, $z_0 = \alpha$ obliczymy $\sin\alpha$ i $\cos\alpha$:

$$x_n \cong \cos\alpha$$

$$y_n \cong \sin\alpha$$

CORDIC – tryb normowania

W trybie normowania (ang. *vectoring mode*) oblicza się kąt nachylenia wektora [(0,0),(x_0,y_0)] do osi [0,x), czyli arctan y_0/x_0. Można to wykonać opisując ruch punktu (x_0,y_0) do punktu ($\sqrt{x_0^2+y_0^2}$,0), czyli obrót o kąt −arctan y_0/x_0 jako sumę obrotów o kąty $d_i \arctan 2^{-i}$. Wynikiem jednostkowego przesunięcia o zbyt duży kąt (w trybie *non-restoring decomposition*) jest przekroczenie osi [0,x), czyli $y_i \le 0$ i wtedy w kolejnym kroku konieczna jest korekta w kierunku przeciwnym. Stąd wynika, że d_i=1 gdy $y_i \le 0$ albo −1 gdy $y_i > 0$.

Krokiem iteracji jest zatem $z_{n+1} = z_n + \arctan t_i = z_n + d_i \arctan 2^{-i}$ oraz

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -d_i 2^{-i} \\ d_i 2^{-i} & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix},$$

przy tym

$$\lim_{n\to\infty} \begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix} = \begin{pmatrix} K\sqrt{x_0^2+y_0^2} \\ 0 \\ z_0 + \arctan y_0/x_0 \end{pmatrix}.$$

CORDIC – uogólnienie

Walther (1971), uogólnienie na exp, log

Podobne tożsamości dotyczą funkcji hiperbolicznych sinh i cosh (wzór Eulera)

$$\cosh(\alpha+\delta)=\cosh\alpha\,\cosh\delta-\sinh\alpha\,\sinh\delta$$

$$\sinh(\alpha+\delta)=\sinh\alpha\,\cosh\delta+\cosh\alpha\,\sinh\delta$$

gdzie (wzór Eulera: $\exp ix=\cos x+i\sin x$)

$$2\sinh x=-2i\sin x=\exp x-\exp(-x)$$

$$2\cosh x=2\cos x=\exp x+\exp(-x)$$

$$\exp x=\cosh x+\sinh x$$

Wartości $t=\operatorname{tg}\delta=\pm 2^{-n}$, można łatwo tablicować i wtedy wszystkie obliczenia można wykonać za pomocą dodawania, odejmowania i przesunięcia.

Wyróżnia się dwa schematy obliczeń:

- w *trybie obrotu* (ang. *rotation mode*) − obrót o znany kąt (dodatni),
- w *trybie normowania* (ang. *vectoring mode*) − przesunięcie wektora z I ćwiartki na oś $[0,x)$ (o nieznany kąt ujemny)

CORDIC – podsumowanie

trzecia zmienna – z_i odległość kątowa wektora od osi $[0, x)$:

$$x_{i+1} = x_i + m\,\sigma_i\,t_i\,y_i$$

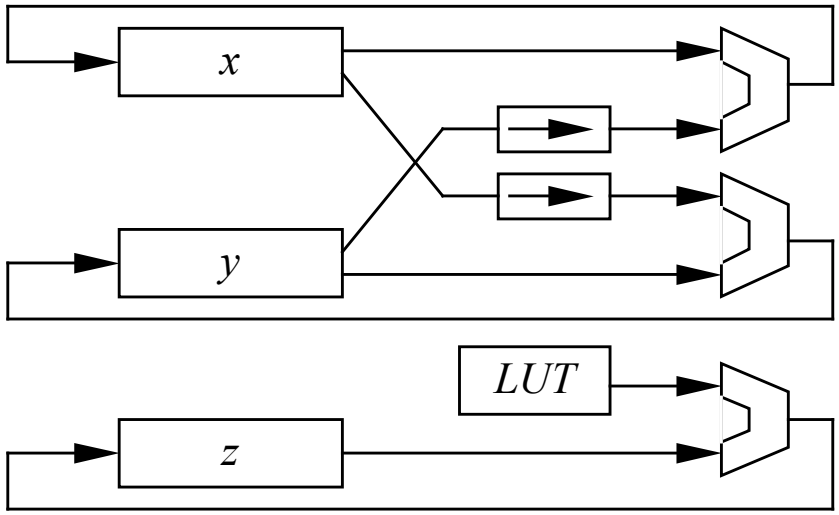
$$y_{i+1} = y_i - \sigma_i\,t_i\,x_i$$

$$z_{i+1} = z_i + \sigma_i\,(1/\sqrt{m})\arctan\sqrt{m}\,t_i$$

gdzie $t_i = 2^{-S(m,i)}$ – przyjęta sekwencja iteracji przyrostów

$K = \prod_{i=0}^{n-1}\sqrt{1+t_i^2}$			*tryb obrotu* ($z_i \to 0$)	*tryb normowania* ($y_i \to 0$)
			$\sigma_i = -\operatorname{sign} z_i$	$\sigma_i = \operatorname{sign}(x_i y_i)$
trygonometr.	$m=1$	$\arctan 2^{-k}$	$x_n \to K(x_0\cos z_0 - y_0 \sin z_0)$ $y_n \to K(x_0 \sin z_0 + y_0 \cos z_0)$ $z_n \to 0$	$x_n \to K\sqrt{x_0^2 + y_0^2}$ $y_n \to 0$ $z_n \to z_0 - \arctan(y_0/x_0)$
hiperboliczny	$m=-1$	$\tanh^{-1} 2^{-k}$	$x_n \to K(x_0\cosh z_0 + y_0 \sinh z_0)$ $y_n \to K(y_0\cosh z_0 + x_0 \sinh z_0)$ $z_n \to 0$	$x_n \to K\sqrt{x_0^2 - y_0^2}$ $y_n \to 0$ $z_n \to z_0 - \tanh^{-1}(y_0/x_0)$

CORDIC – realizacja układowa



Zalety algorytmu CORDIC
obliczanie funkcji elementarnych za pomocą prostych działań arytmetycznych
prosta implementacja układowa algorytmu (Cyrix, procesory DSP)

Wada – wolna zbieżność, konieczność wykonania dużej liczby obliczeń,
$\rightarrow \rightarrow$ wersja ulepszona CORDIC–2.

Arytmetyka wielkich liczb (W11)

Wielkie liczby

Wielkie liczby (*big numbers*) to takie, których rozmiar znacząco przekracza rozmiar słowa maszynowego (kilkaset cyfr dziesiętnych, kilka tysięcy bitów). Zakłada się zwykle że wielkie liczby są dodatnimi liczbami całkowitymi.

W pamięci komputera są składowane jako jednowymiarowe tablice słów, z których każde reprezentuje jedną cyfrę w podstawie wyznaczonej rozmiarem słowa maszynowego (2^{32} or 2^{64}) lub mieszczącej się w nim dziesiętnej ($10^9<2^{32}$, $10^{19}<2^{64}$). Kolejność cyfr odpowiada przyjętej konwencji LE (*little endian*) lub BE (*big endian*)

Arytmetyka wielkich liczb obejmuje 4 obszary:

- dodawanie, odejmowanie i porównanie
- mnożenie, dzielenie i obliczanie pierwiastka kwadratowego
- obliczanie funkcji elementarnych
- arytmetyka resztowa w zastosowaniach kryptograficznych

Algorytm dodawania/odejmowania pozycyjnego nie zależy od długości liczb, więc może być realizowany jako sekwencja działań w wysokiej podstawie (*high radix*) z przeniesieniem jako łącznikiem między pozycjami. W językach symbolicznych przeniesienia te są zwykle niedostępne, i muszą być wytwarzane w algorytmie.

Dodawanie i odejmowanie rozszerzonej precyzji

- architektura IA-32, cyfra=słowo 32-bitowe, konwencja LE (***little endian***)
- wskaźniki i rozmiar argumentów i wyniku przekazywane przez stos

```
.type exadd @function                  # definicja funkcji
exsub:  push %ebp                      # .....
        movl %esp, %ebp                # wskaźnik parametrów wywołania
        movl 8(,%ebp,4), %ecx          # rozmiar argumentów ze stosu
        movl 12(,%ebp,4), %edx         # adres odjemnej ze stosu
        movl 16(,%ebp,4), %ebx         # adres odjemnika ze stosu
        movl 20(,%ebp,4), %edi         # adres różnicy/sumy ze stosu
        movl $-1, %esi                 # wartość początkowa wskaźnika
        clc                            # ustawienie CF=0
next:   inc %esi
        movl (%ebx,%esi,4), %eax
        sbb %eax, (%edx,%esi,4)        # w dodawaniu adc zamiast sbb
        movl (%ebx,%esi,4), %eax
        loop next                      # licznik pętli w %ecx
        movl %ebp, %esp                # przywrócenie wskaźników
        pop %ebp
        ret                            # wskaźnik nadmiaru w OF lub CF
```

Mnożenie rozszerzonej precyzji

Mnożenie polega na akumulacji iloczynów częściowych.

- system naturalny, architektura IA-32, cyfra=słowo maszynowe 32-bitowe

```
prod:   movl $0, (%ebx)          # zerowanie najniższych cyfr iloczynu
        movl $0, 4(%ebx)
accum:  push %eax                # licznik cyfr mnożnika (iloczynów częściowych)
        push %ebx                # bieżący indeks najniższej cyfry iloczynu
        movl 8(,%ebp,4), %ecx    # odtworzenie licznika cyfr mnożnej
partp:  movl (,%esi,4), %eax     # kolejna cyfra mnożnej
        movl (,%edi,4), %edx     # kolejna cyfra mnożnika
        mull %edx                # iloczyn częściowy w %edx:eax
        addl %eax, (,%ebx,4)     # aktualizacja niższej cyfry iloczynu częściowego
        adcl %eax, 4(,%ebx,4)    # aktualizacja wyższej cyfry iloczynu częściowego
        incl %esi                # wskaźnik kolejnej cyfry mnożnej
        loop partp               # zliczanie cyfr mnożnej
        pop %ebx                 # przygotowanie obliczenia następnego
        inc %ebx                 # iloczynu częściowego
        inc %edi                 # wskaźnik kolejnej cyfry mnożnika
        pop %eax
        dec %eax                 # zliczanie cyfr mnożnika
        jnz accum
        ret
```

Mnożenie wielkich liczb

Jak w dodawaniu, liczby są reprezentowane jako ciągi cyfr w wysokiej podstawie. Działaniem elementarnym jest instrukcja mnożenia maszynowego.

Złożoność takiej szkolnej metody (ang. *schoolbook algorithm*) jest rzędu O(mn), gdzie m,n są rozmiarami czynników.

Algorytm może być zrealizowany z użyciem dekompozycji argumentów. Najprostsza dekompozycja polega na zastąpieniu mnożenia liczb długości $2n$ przez działania na ich połówkach długości n:

$$XY = (x_h\beta^n + x_l)(y_h\beta^n + y_l) = x_h y_h \beta^{2n} + (x_h y_l + x_l y_h)\beta^n + x_l y_l = C_2\beta^{2n} + C_1\beta^n + C_0$$

Procedura może być realizowana iteracyjnie w trybie "dziel i zwyciężaj", lecz nie zmniejsza to złożoności działania − liczba elementarnych mnożeń i dodawań jest taka jak w algorytmie szkolnym.

Działania elementarne na najniższym poziomie mogą być zrównoleglone, lecz dodawania muszą być wykonane sekwencyjnie.

Algorytm Karatsuby-Ofmana

Karatsuba zauważył, że dekompozycję można przyśpieszyć obliczając środkowy współczynnik jako $C_1 = (x_h + x_l)(y_h + y_l) - C_2 - C_0$ lub $C_1 = C_2 - (x_h - x_l)(y_h - y_l) + C_0$.
Wymaga to jednego mnożenia i 4 dodawań zamiast 2 mnożeń i jednego dodawania.
Algorytm jest rekurencyjny
Wejście: $A = \sum_{i=0}^{n-1} a_i \beta^i$ and $B = \sum_{i=0}^{n-1} b_i \beta^i$, wyjście: $C = AB = \sum_{i=0}^{2n-1} c_i \beta^i$ w podstawie β.

KaratsubaMul(A,B)
$k := n$,
if $k=1$ **then** return AB
$k := \lceil n/2 \rceil$
$(A_0, B_0) := (A, B) \bmod \beta^k$, $(A_1, B_1) := (A, B) \operatorname{int} \beta^k$
$C_0 := KaratsubaMul(A_0, B_0)$
$C_2 := KaratsubaMul(A_1, B_1)$
$C_1 := KaratsubaMul(A_0 - A_1, B_0 - B_1)$
return $C = C_2 \beta^{2k} + (C_0 + C_2 - C_1)\beta^k + C_0$

Szacowana złożoność to $O(n^{\log 3})$.

Zastosowanie interpolacji Lagrange'a

Twierdzenie Lagrange'a o interpolacji stanowi, że aby określić wielomian $C(x)$ stopnia k wystarczy podać jego wartość w k+1 punktach.

Reprezentację pozycyjną liczby można traktować jako współczynniki wielomianu, przy tym wartość liczby jest równa wartości tego wielomianu dla $x=\beta$:

$$\sum_{i=0}^{n} d_i \beta^i = \sum_{i=0}^{n} d_i x^i \Big|_{x=\beta} = \sum_{i=0}^{n/k} D_i x^i \Big|_{x=\beta^k}$$

Iloczyn dużych liczb można więc obliczyć jako wartość iloczynu wielomianów.

Iloczyn wielomianów $p(x)q(x)$ stopnia m i n jednoznacznie wyznaczają jego wartości w m+n+1 punktach. Problem można sprowadzić do rozwiązania układu równań liniowych. $\mathbf{c} = \mathbf{B} \times [c_i]$., gdzie $\mathbf{c} = [p(i)q(i)]$ jest wektorem iloczynów cząstkowych

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & e_1 & e_1^2 & \dots & e_1^t \\ \dots & \dots & \dots & \dots & \dots \\ 1 & e_{d-1} & e_{d-1}^2 & \dots & e_{d-1}^t \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

Algorytm mnożenia Toom-Cook

Algorytm Toom–Cook korzysta z interpolacji Lagrange'a. Aby uprościć obliczenia punktami waluacji są nieduże liczby całkowite: 0,1,−1,2,−2,... oraz zawsze ∞.

Procedurę można opisać "Vandermonde-podobną" macierzą A (2d+1)x(d+1)

$$\mathbf{p} = [p(i)] = \mathbf{A} \times [p_i],$$
$$\mathbf{q} = [q(i)] = \mathbf{A} \times [q_i],$$

a współczynnikami iloczynu są rozwiązania układu równań liniowych:

$$\mathbf{c} = \mathbf{B} \times [c_i].$$

gdzie **B** jest "Vandermonde-podobną" macierzą kwadratową (2d+1)x(2d+1) zaś $\mathbf{c} = [p(i)q(i)]$ jest wektorem iloczynów. Dla punktów ewaluacji $0, e_1, ..., e_{d-1}, \infty$ macierze mają postać (**A**: t=d lub **B**: t=2d):

$$\begin{bmatrix} 1 & 0 & 0 & ... & 0 \\ 1 & e_1 & e_1^2 & ... & e_1^t \\ ... & ... & ... & ... & ... \\ 1 & e_{d-1} & e_{d-1}^2 & ... & e_{d-1}^t \\ 0 & 0 & 0 & ... & 1 \end{bmatrix}$$

Algorytm mnożenia Toom-Cook 3

Najpopularniejszą wersją jest algorytm ToomCook3, gdzie duże liczby są reprezentowane za pomocą wielomianów kwadratowych

$$p(x) = p_2 x^2 + p_1 x + p_0,$$
$$q(x) = q_2 x^2 + q_1 x + q_0,$$

których iloczyn jest wielomianem stopnia 4,więc potrzeba 5 punktów ewaluacji Dla $0,1,-1,-2,\infty$ mamy:

$$p(0) = p_0, \quad p(1) = p_2 + p_1 + p_0, \quad p(-1) = p_2 - p_1 + p_0,$$
$$p(-2) = 4p_2 - 2p_1 + p_0, \quad p(\infty) = p_2,$$

Liczbę dodawań można zredukować:

$$p^* = p_2 + p_0, \quad p(0) = p_0, \quad p(1) = p^* + p_1,$$
$$p(-1) = p^* - p_1, \quad p(2) = 2[p(-1) + p_2] - p_0 \quad p(\infty) = p_2,$$

Analogiczne obliczenia należy wykonać dla drugiego wielomianu oraz iloczynu $c(x)=p(x)q(x)$. Całkowita liczba mnożeń , równa liczbie punktów waluacji jest mniejsza niż w zwykłym algorytmie.

Algorytm Karatsuby jest równoważny ToomCook2 z punktami waluacji $0,1,\infty$ lub $0,-1,\infty$.

Dzielenie wielkich liczb – algorytm szkolny

„Szkolny” algorytm ma podstawy w teorii liczb. Wielkie liczby są reprezentowane jako łańcuchy cyfr w wysokiej podstawie, a podstawowe działania wykonywane są na krótkich łańcuchach tych cyfr.

Aby zapewnić dobrą dokładność wyników częściowych wymaga się normalizacji dzielnika tak, by jego wiodąca cyfra była większa od połowy podstawy. Zapewni to mnożenie oryginalnego dzielnika przez potęgę 2. Dzielna i dzielnik są dane w zapisie pozycyjnym: $A = \sum_{i=0}^{m+n-1} a_i \beta^i$, $D = \sum_{i=0}^{n-1} d_i \beta^i$, gdzie $d_{n-1} \geq \frac{1}{2}\beta$ (normalizacja). Wtedy $a_{m+n-1} < 2\beta^m d_{n-1}$, więc wiodącą cyfrą ilorazu jest 0 ($a_{m+n-1} < \beta^m d_{n-1}$) lub 1.

Kolejne cyfry ilorazu określa iloraz liczby utworzonej przez 2 wiodące cyfry reszty przez wiodącą cyfrę znormalizowanego dzielnika. Jeśli wiodąca cyfra reszty częściowej $R = \sum_{i=0}^{n+j} r_i \beta^i$ jest taka jak wiodąca cyfra dzielnika, potrzebna jest uprzedzająca korekcja w kroku 1. Ponieważ każda reszta musi być mniejsza od dzielnika, więc obliczona reszta i cyfra ilorazu muszą być korygowane.

Obliczona w ten sposób cyfra ilorazu może być najwyżej o 2 większa od faktycznej wartości, skąd wynika schemat korekcji.

Algorytm szkolny – poprawność procedury

$R < \beta^j D$ jest niezmiennikiem procedury. Początkowo wynika to z założenia. Jeśli $q_j = \beta - 1$, to $R < \beta^{j+1} D - q_i \beta^j D = \beta^j D(\beta - (\beta - 1)) = \beta^j D$. W innych przypadkach

$$\frac{r_{j+n}\beta + r_{j+n-1}}{d_{n-1}} - \frac{d_{n-1} - 1}{d_{n-1}} \le q_j = \left\lfloor \frac{r_{j+n}\beta + r_{j+n-1}}{d_{n-1}} \right\rfloor \le \frac{r_{j+n}\beta + r_{j+n-1}}{d_{n-1}},$$

skąd wynika, że

$$\begin{aligned} R - q_j \beta^j D &\le (r_{n+j}\beta + r_{n+j-1} + \ldots)\beta^{n+j-1} - q_j \beta^j d_{n-1} = \\ &= (r_{n+j}\beta + r_{n+j-1})\beta^{n+j-1} + R \bmod \beta^{n+j-1} - (r_{n+j}\beta + r_{n+j-1} - d_{n-1} + 1)\beta^{n+j-1} \\ &\le (d_{n-1} - 1)\beta^{n+j-1} + R \bmod \beta^{n+j-1} < \beta^j D \end{aligned}$$

Z drugiej strony, ponieważ $q_j d_{n-1} \le r_{n+j}\beta + r_{n+j-1}$, więc

$$R - q_j \beta^j D > (r_{n+j}\beta + r_{n+j-1})\beta^{n+j-1} - q_j (d_{n-1} + 1)\beta^{n+j-1} > -q_j \beta^{n+j-1}$$

i tak q może być najwyżej o 2 większe od faktycznej wartości.

Modyfikacje procedury:
– obliczenie więcej niż 1 cyfry w jednym kroku
– "divide and conquer"

Algorytm szkolny - procedura

If $A \geq \beta^m D$ **then** q=1, $R \leftarrow A - \beta^m D$, ; obliczenie pierwszej reszty

otherwise q=0, $R \leftarrow A$

1. **for** j=m−1 **downto** 0 **step** 1 **do**

$$q_j \leftarrow \left\lfloor \frac{r_{j+n}\beta + r_{j+n-1}}{d_{n-1}} \right\rfloor,$$; wstępne obliczenie q

$q_j \leftarrow \min(q_j, \beta - 1)$; korekcja gdy $q \geq \beta$

$R \leftarrow R - q_j \beta^j D$

2. **while** R<0 **do** $q_j \leftarrow q_j - 1$, $R \leftarrow R + \beta^j D$; korekcja q oraz R

return $Q = \sum_{i=0}^{m-1} q_i \beta^i$, R.

Algorytm Svobody

W wyniku skalowania dzielnika przez $k = \left\lfloor \beta^{m+1} / d_{n-1} \right\rfloor$ jego wiodąca cyfra jest równa 1, a wartość jest nieco większa od β^{m+1}. Bieżąca cyfra ilorazu jest wtedy równa lub o 1 mniejsza od wiodącej cyfry reszty częściowej (więc jest mniejsza od β).

Ponieważ procedura zwraca iloraz Q' z dzielenia przez kD, konieczna jest korekcja:

$$A = QD + r = Q'kD + R = (kQ')D + D \cdot R\operatorname{div} D + R \bmod D = (kQ' + q_0)D + r .$$

Czynnik k można obliczyć dokładniej, lecz nie wpływa to na złożoność procedury.

Dla danych: $A = \sum_{i=0}^{m+n-1} a_i \beta^i$, $D = \sum_{i=0}^{n-1} d_i \beta^i$, oblicz $k = \left\lfloor \beta^{m+1} / d_{n-1} \right\rfloor$ i $D' = kD$, $R \leftarrow A$

1. **for** j=m−1 **downto** 1 **step** 1 **do**

   $q_j \leftarrow r_{j+n}$,

   $R \leftarrow R - q_j \beta^j D'$; $R = \sum_{i=0}^{n+j} r_i \beta^i$

2. **if** R<0 **then** $q_j \leftarrow q_j - 1$, $R \leftarrow R + \beta^{j-1} D'$; poprawka

   $Q' = \sum_{j=1}^{m-1} q_j \beta^j$,

   $(q_0, r) = (R\operatorname{div} D, R \bmod D)$; korekcja końcowa

   $Q = kQ' + q_0$

Obliczanie NWD

Lehmer (1938) zauważył, kilka pierwszych ilorazów w algorytmie Euklidesa można ocenić na podstawie najbardziej znaczących cyfr wielkich liczb.

Załóżmy, że obliczono początkowe ilorazy: $q_1 = a_H \operatorname{int} b_H$, $q_2 = b_H \operatorname{int}(a_H - q_1 b_H)$.

Wtedy $GCD(a,b) = GCD(b, a - q_1 b) = GCD(a - q_1 b, b - q_2(a - q_1 b))$ a te współczynniki są zwykle mniejsze niż pierwiastek podstawy (mają rozmiar połowy słowa).

Aby uzyskać głębszą redukcję można użyć 2 cyfr wiodących.

W czterech krokach uzyskamy $GCD(a,b) = GCD(ta + ub, va + wb)$, gdzie

$$
\begin{aligned}
t &= 1 + q_2 q_3, \\
u &= -(q_1 + q_3 + q_1 q_2 q_3), \\
v &= -(q_2 + q_4 + q_2 q_3 q_4), \\
w &= 1 + q_1 q_2 + q_4(q_1 + q_3 + q_1 q_2 q_3).
\end{aligned}
$$

W pięciu krokach będzie

$$
\begin{aligned}
t &= -(q_2 + q_4 + q_2 q_3 q_4), \\
u &= 1 + q_1 q_2 + q_4(q_1 + q_3 + q_1 q_2 q_3), \\
v &= 1 + q_2 q_3 + q_5(q_2 + q_4 + q_2 q_3 q_4), \\
w &= (q_1 + q_3 + q_1 q_2 q_3) + q_5[1 + q_1 q_2 + q_4(q_1 + q_3 + q_1 q_2 q_3)].
\end{aligned}
$$

REPETYTORIUM
(W12)

KOLOKWIUM
(W13)