# ODEJMOWANIE
# I DODAWANIE

## Algorytm dodawania i odejmowania pozycyjnego

*Problem*: Obliczyć reprezentację $\mathbf{S}=\{...,s_{i+1},s_i,s_{i-1},...\}_\beta$ sumy **X+Y** i różnicy **X–Y** liczb

$\mathbf{X}=\{...,x_{i+1},x_i,x_{i-1},...\}_\beta$, $\mathbf{Y}=\{...,y_{i+1},y_i,y_{i-1},...\}_\beta$ danych w notacji pozycyjnej.

Jeśli zbiór cyfr jest **standardowy**, $\mathbf{D}=\{0,1,\ldots,\beta-1\}$ a podstawa naturalna, to

Algorytm odejmowania:

1. Oblicz na pozycji $i$: $u_i = x_i - y_i - c_i$
2. Jeśli $u_i \geq 0$, to $s_i=u_i$ oraz $c_{i+1}=0$, w przeciwnym razie: ($u_i<0$) $s_i=u_i+\beta$ oraz $c_{i+1}=1$,

   albo: $x_i - y_i - c_i = -\beta c_{i+1} + s_i$, gdzie $x_i, y_i, s_i \in \{0,1,...,\beta-1\}, c_i \in \{0,1\}$

Algorytm dodawania:

1. Oblicz na pozycji $i$: $u_i = x_i + y_i + c_i$
2. Jeśli $u_i<\beta$, to $s_i=u_i$ oraz $c_{i+1}=0$, w przeciwnym razie: ($u_i \geq \beta$) $s_i=u_i-\beta$ oraz $c_{i+1}=1$.

   albo: $x_i + y_i + c_i = \beta c_{i+1} + s_i$, gdzie $x_i, y_i, s_i \in \{0,1,...,\beta-1\}, c_i \in \{0,1\}$

- wynik działania na pozycji nie zależy od wartości cyfr z wyższych pozycji
- przeniesienie $c_{i+1}$ jest jedynym powiązaniem pozycji bieżącej z poprzednią, dla reprezentacji (prawostronnie) skończonych na pozycji „$-m$” jest $c_{-m}=0$

# Odejmowanie i dodawanie

## *Odejmowanie*

- tworzy system pełny:
  - umożliwia proceduralne wytworzenie **0** (**0**=**X**–**X**),
  - umożliwia proceduralne wytworzenie liczby przeciwnej ($\underline{\mathbf{X}}$=**0**–**X**),
- *dodawanie* przez odejmowanie liczby przeciwnej: $\mathbf{X}+\mathbf{Y}=^{\mathrm{df}}\mathbf{X}-((\mathbf{X}-\mathbf{X})-\mathbf{Y})$.

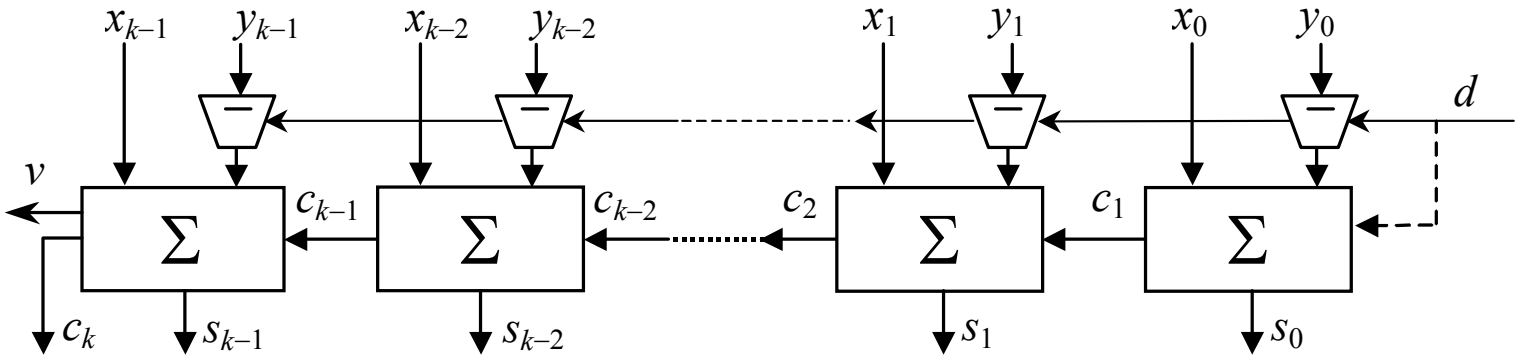
***Dodawanie*** – działanie arytmetyczne *powszechnie uznawane za podstawowe*, chociaż:

- jest **łączne i przemienne**
- *odejmowanie* przez dodawanie liczby przeciwnej: $\mathbf{X}-\mathbf{Y}=^{\mathrm{df}}\mathbf{X}+(\underline{\mathbf{Y}})$ wymaga **utworzenia** reprezentacji $\underline{\mathbf{Y}}$ liczby przeciwnej do **Y**, ale:
  - proceduralne wytworzenie „–Y” przez dodawanie nie jest możliwe
  - proceduralne wytworzenie „0” przez dodawanie nie jest możliwe
  - w systemach *uzupełnieniowych* (ang. *radix-complement*)

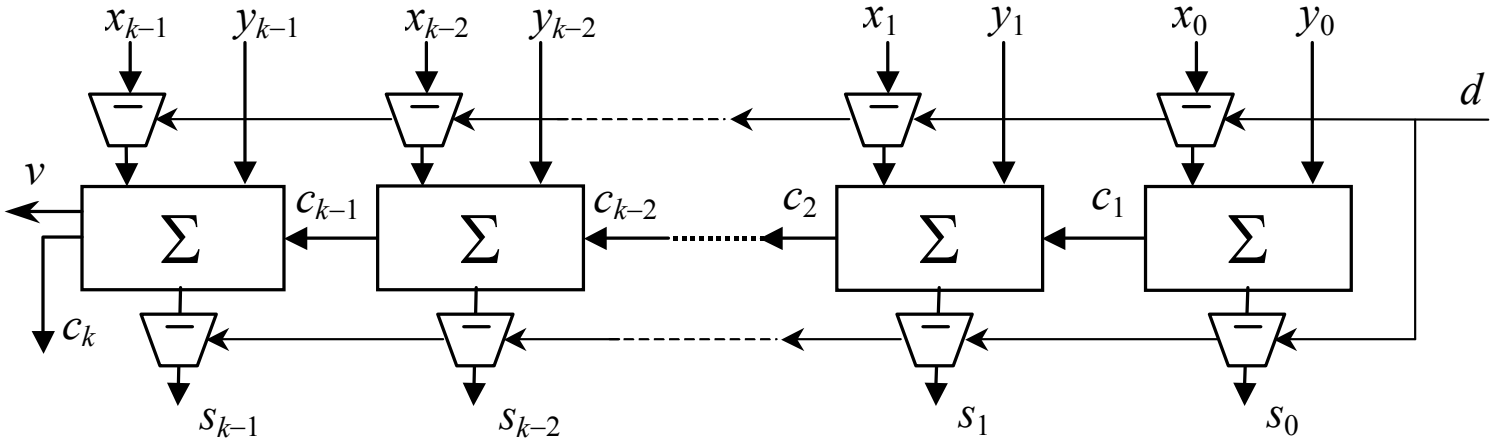
$$\underline{\mathbf{X}} = \mathbf{0} - \mathbf{X} = \underline{\mathbf{Q}} + \mathbf{Q} - \mathbf{X} = \underline{\mathbf{Q}} + \overline{\mathbf{X}}$$

gdzie $\mathbf{Q} = \{\ldots,\beta-1,\beta-1,\beta-1,\ldots\}$, $\overline{\mathbf{X}} = \{\ldots,\overline{x}_{i+1},\overline{x}_i,\overline{x}_{i-1},\ldots\}_\beta = \mathbf{Q}-\mathbf{X}$ to *dopełnienie* liczby **X**, a $\underline{\mathbf{Q}} = \mathbf{ulp} = [0,\ldots,0,1\}$ dla reprezentacji (prawostronnie) skończonych albo $\underline{\mathbf{Q}} = \mathbf{0}$ dla reprezentacji nieskończonych (np. *ułamki okresowe*).

# Uniwersalny schemat dodawania i odejmowania

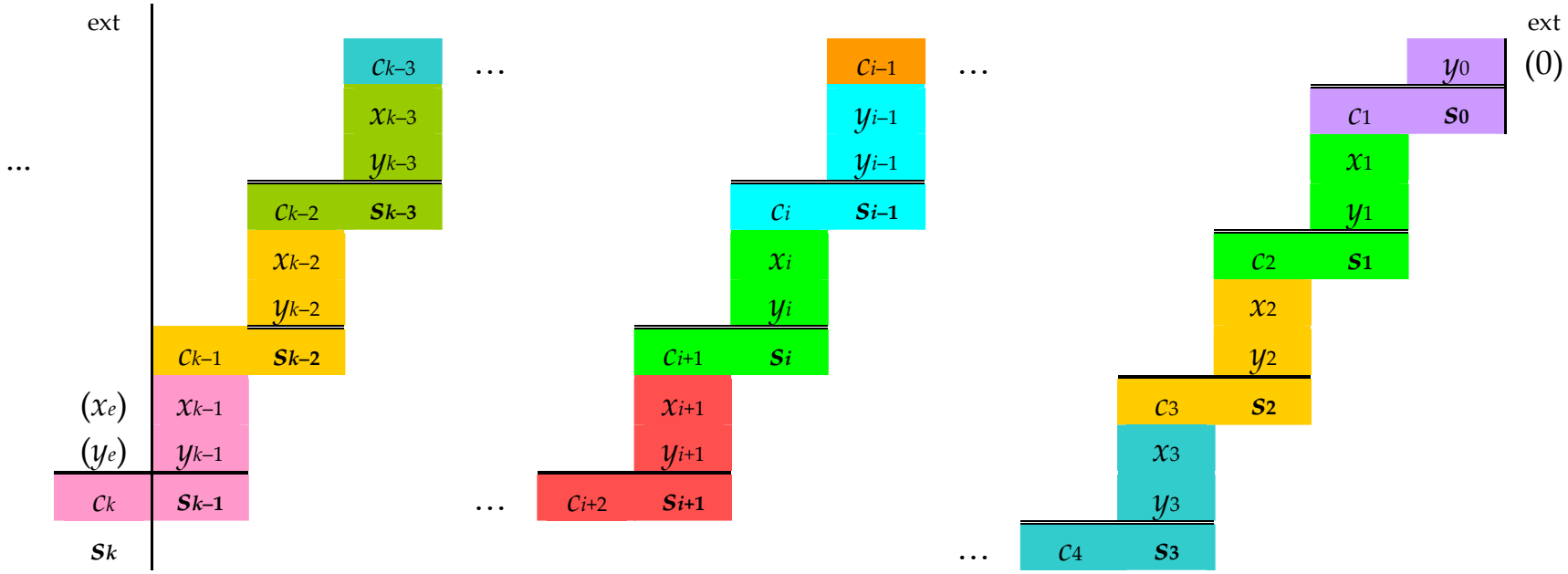



Zamienność dodawania i odejmowania w systemach uzupełnieniowych ($X-Y=X+\underline{Y}$)

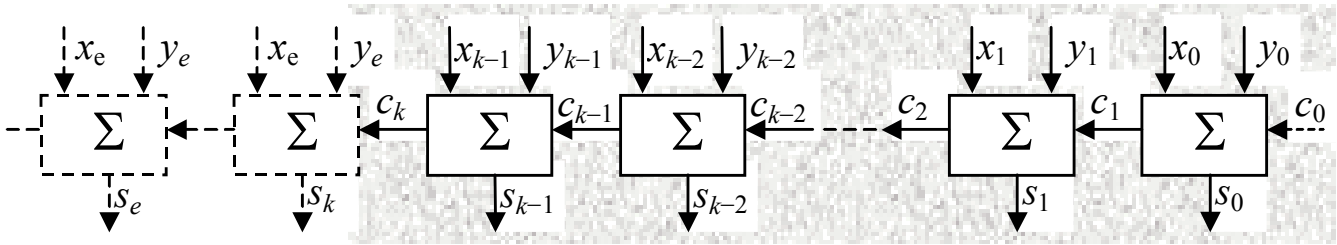



Zamienność dodawania i odejmowania w systemach uzupełnieniowych ($X-Y=\overline{\overline{X}+Y}$)

# Schemat dodawania i odejmowania w systemach pozycyjnych




$(x_e)$, $(y_e)$ – ciąg cyfr rozszerzenia lewostronnego (0 w systemach naturalnych)




Jeśli $s_k$ nie jest cyfrą rozszerzenia dla $\mathbf{S} = \{s_{k-1},...,s_1,s_0\}_\beta$, to zakres jest przekroczony

## Poprawność wytworzonej sumy lub różnicy

$$\left|\{(x_e), x_{k-1},\ldots,x_1,x_0\}\right|_\beta = -\sigma(x_e)\beta^k + \sum_{i=0}^{k-1} x_i\beta^i ,$$

gdzie $\sigma(z) = \frac{1}{2}(1+\mathrm{sgn}(2z-\beta+1))$ ($\sigma(0)$=0, $\sigma(\beta-1)$=1).

Suma lub różnica liczb $X = \left|\mathbf{X} = \{(x_e), x_{k-1},\ldots,x_1,x_0\}\right|_\beta$ oraz $Y = \left|\mathbf{Y} = \{(y_e), y_{k-1},\ldots,y_1,y_0\}_\beta\right|$ może być zawsze zapisana poprawnie przy użyciu jednej dodatkowej cyfry:

$$X \pm Y = \left| \mathbf{X} \pm \mathbf{Y} = \mathbf{S} = \{(s_e), s_k, s_{k-1},\ldots,s_1,s_0\} \right|_\beta$$

Jeśli $s_k \neq s_e$, to wynik jest poza zakresem ustalonym dla argumentów, a wskaźnikiem *nadmiaru stałoprzecinkowego* lub *przepełnienia* jest $\sigma(s_k) \neq \sigma(x_k) \pm \sigma(\overline{y}_k) \pm c_{k+1}$.

### Odejmowanie liczb w systemie naturalnym przez dodanie uzupełnienia

Odejmowanie liczb $\left|\{x_{k-1},\ldots,x_1,x_0\}_\beta\right|$ i $\left|\{y_{k-1},\ldots,y_1,y_0\}_\beta\right|$ jest równoważne dodawaniu $\left|\{(0), x_{k-1},\ldots,x_1,x_0\}_{\mathrm{U}\beta}\right|$ oraz $\left|\{(\beta-1), \overline{y}_{k-1},\ldots,\overline{y}_1,\overline{y}_0\}_{\mathrm{U}\beta}\right| + 1$ w systemie uzupełnieniowym.

Ponieważ $\overline{y}_k = \beta - 1$, $x_k$=0, więc różnica jest poprawna ($s_k$=0) jeśli $c_{k+1} = 1$.

## Praktyczne aspekty wykrywania nadmiaru

W systemie pozycyjnym o podstawie $\beta$ lewostronne rozszerzenie rozmiaru liczby o 1 pozycję daje $\beta$-krotne rozszerzenie zakresu, a ponieważ

$$-2\min(|X|,|Y|) \le X \pm Y \le 2\max(|X|,|Y|),$$ więc:

WNIOSEK 1: W rozszerzonej lewostronnie o 1 pozycję reprezentacji naturalnej lub uzupełnieniowej suma nie więcej niż $\beta$ liczb musi być poprawna.

WNIOSEK 2: Aby wykryć nadmiar w dodawaniu (odejmowaniu) należy dodać operandy rozszerzone lewostronnie o 1 pozycję $x_e = \sigma(x_{k-1})(\beta-1)$, $y_e = \sigma(y_{k-1})(\beta-1)$ i sprawdzić dodatkową pozycję sumy (różnicy) $s_e$.

| *test poprawności* | | *zakres dodawania/ odejmowania* | | | | | | | | *rozszerzenie* |
|---|---|---|---|---|---|---|---|---|---|---|
| | ? | | | | | | | | | |
| | $x_e$ | $x_{k-1}$ | $x_{k-2}$ | … | $x_{i+1}$ | $x_i$ | $x_{i-1}$ | … | $x_0$ | 0… 0 (0) |
| | $y_e$ | $y_{k-1}$ | $y_{k-2}$ | … | $y_{i+1}$ | $y_i$ | $y_{i-1}$ | … | $y_0$ | 0… 0 (0) |
| ± | $c_k$ | $c_{k-1}$ | $c_{k-2}$ | … | $c_{i+1}$ | $c_i$ | $c_{i-1}$ | … | 0 | 0… 0 (0) |
| | $s_e$ | $s_{k-1}$ | $s_{k-2}$ | … | $s_{i+1}$ | $s_i$ | $s_{i-1}$ | | $s_0$ | 0… 0 (0) |

Jeśli $s_e$ nie jest cyfrą rozszerzenia dla $s_{k-1}$, czyli $s_e \neq \sigma(s_{k-1})(\beta-1)$, to w dodawaniu (odejmowaniu) ustalonego zakresu ($k$-pozycyjnym) wystąpił nadmiar.

## Reprezentacja liczby przeciwnej w systemie uzupełnieniowym

W zapisie uzupełnieniowym nieskończony ciąg cyfr rozszerzenia lewostronnego $(\beta–1)$ reprezentuje wartość –1, więc $(\beta–1)+1=(0)$, a zatem ($x_e$ – cyfra rozszerzenia)

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | $(0)$ | $0$ | $0$ | … | $0$ | $0$ | $1$ | $[1]$ |
| + | $(\beta–1)$ | $\beta–1$ | $\beta–1$ | … | $\beta–1$ | $\beta–1$ | $\beta–1$ | $+[–1]$ |
| – | $(x_e)$ | $x_{k–1}$ | $x_{k–2}$ | … | $x_2$ | $x_1$ | $X_0$ | $-[X]$ |
| = | $(\overline{x}_e)$ | $\overline{x}_{k–1}$ | $\overline{x}_{k–2}$ | … | $\overline{x}_2$ | $\overline{x}_1$ | $\overline{x}_0$ | $=[\overline{X}]$ |
| + | $(0)$ | $0$ | $0$ | … | $0$ | $0$ | $1$ | $+[1]$ |

*Podejście formalne* (rozwinięcie prawostronnie skończone) *potwierdza powyższy wynik*:

Mamy $-1=-\beta^k+(\beta^{k-1}+\ldots+\beta^1+\beta^0)(\beta-1)$ i $\sigma(y_{k-1})=1-\sigma(\overline{y}_{k-1})$, więc

$$(1-1)-\left[-\sigma(y_{k-1})\beta^k+\sum_{i=0}^{k-1}y_i\beta^i\right]=\sigma(y_{k-1})\beta^k+\sum_{i=0}^{k-1}[(\beta-1)-y_i]\beta^i-\beta^k+1=$$

$$=\sigma(\overline{y}_{k-1})\beta^k+\sum_{i=0}^{k-1}\overline{y}_i\beta^i+1$$

czyli $\mathbf{0}-\mathbf{Y}=\overline{\mathbf{Y}}+\mathbf{ulp}$, gdzie $\mathbf{ulp}=\{\ldots001\}$ (jedynka na najniższej pozycji).

# Systemy dwójkowe

## Reprezentacja liczby przeciwnej w dwójkowym systemie uzupełnieniowym (rozwinięcie skończone)

W szczególnym przypadku $\beta$=2 otrzymujemy dla reprezentacji $k$-pozycyjnej:

$$\underline{X} = (-1 - X) + 1 = [-2^{k-1} + (2^{k-2} + \ldots + 2^1 + 2^0) - (-x_{k-1}2^{k-1} + \sum_{i=0}^{k-2} x_i 2^i)] + 1, \text{ więc}$$

$$\underline{X} = [-(1 - x_{k-1})2^{k-1} + \sum_{i=0}^{k-2}(1 - x_i)2^i] + 1 = \overline{X} + 1$$

*Uwaga*: Jeśli **X**={100…0} operacja jest niewykonalna (wystąpi nadmiar)

*algorytmy mnemotechniczne*: („1" – jednostka na najniższej pozycji, *ulp*)

- zaneguj bity oryginału i do uzyskanego kodu dodaj pozycyjnie „1"
- zaneguj bity oryginału, oprócz prawostronnego ciągu zer i poprzedzającej go „1" (*propagacja dodawanej „1" kończy się na pozycji najniższej „1" oryginału*)

## Wykrywanie nadmiaru w systemach dwójkowych uzupełnieniowych

W zapisie uzupełnieniowym rozszerzeniem lewostronnym jest kopia wiodącego bitu, w zapisie naturalnym 0. Na pozycji wiodącej obliczoną sumą lub różnicą jest $s_{k-1} = x_{k-1} \pm y_{k-1} \pm c_{k-1} \mp 2c_k$,na pozycji rozszerzenia $s_k = x_{k-1} \pm y_{k-1} \pm c_k \mp 2c_{k+1}$, co oznacza, że $s_k = s_{k-1}$ tylko wtedy, gdy $c_k = c_{k-1}$, bo ostatni składnik jest parzysty.

## Dodawanie i odejmowanie w dwójkowych systemach spolaryzowanych

Gdy $N = 2^{k-1}$ jest $X_{+N} = \sum_{i=0}^{k-1} x_i 2^i - 2^{k-1} = -(1-x_{k-1})2^{k-1} + \sum_{i=0}^{k-2} x_i 2^i$,

$$(ii) \qquad \Rightarrow \left|\{x_{k-1}, x_{k-2}, \ldots, x_0\}_{\mathrm{U2}}\right| = \left|\{\overline{x}_{k-1}, x_{k-2}, \ldots, x_0\}_{+2\uparrow(k-1)}\right|$$

Gdy $N = 2^{k-1}-1$, to ponieważ $(2^{k-1}-1) = \sum_{i=0}^{k-2} 2^i$, więc otrzymamy

$$X_{+N} = \sum_{i=0}^{k-1} x_i 2^i - (2^{k-1}-1) = x_{k-1}2^{k-1} + \sum_{i=0}^{k-2}(x_i - 1)2^i = -\left(-x_{k-1}2^{k-1} + \sum_{i=0}^{k-2}(1-x_i)2^i\right)$$

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

Łatwa konwersja (bitowa) na i z kodu U2 uzasadnia celowość algorytmu:

ALGORYTM

1. Wykonaj konwersję argumentów na U2 zgodnie z formułą (*i*) (lub (*ii*))
2. Wykonaj działanie w U2 i sprawdź poprawność (nadmiar)
3. Wykonaj konwersję sumy/różnicy zgodnie z formułą (*i*) (lub (*ii*))

## Logika dodawania i odejmowania w systemie dwójkowym*

Arytmetyczny algorytm dodawania/odejmowania: $x_i \pm y_i \pm c_i = \pm 2c_{i+1} + s_i$ definujący funkcje sumy/różnicy i przeniesienia, prowadzi do następujących wyrażeń logicznych:

- sumy arytmetycznej $s_i$ operandów dwójkowych $x_i, y_i, c_i$ na pozycji $i$-tej

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

- przeniesienia $c_{i+1}$ na wyższą pozycję

$$c_{i+1} = x_i y_i + (x_i \oplus y_i)c_i = x_i y_i + (x_i + y_i)c_i$$

definiujące *sumator pełny* (*full adder*, FA), albo

- różnicy arytmetycznej $s_i$ operandów dwójkowych $x_i, y_i, c_i$ na pozycji $i$-tej

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

- pożyczki $c_{i+1}$ z wyższej pozycji

$$c_{i+1} = \bar{x}_i y_i + (\bar{x}_i \oplus y_i)\, c_i = \bar{x}_i y_i + (\bar{x}_i + y_i)\, c_i$$

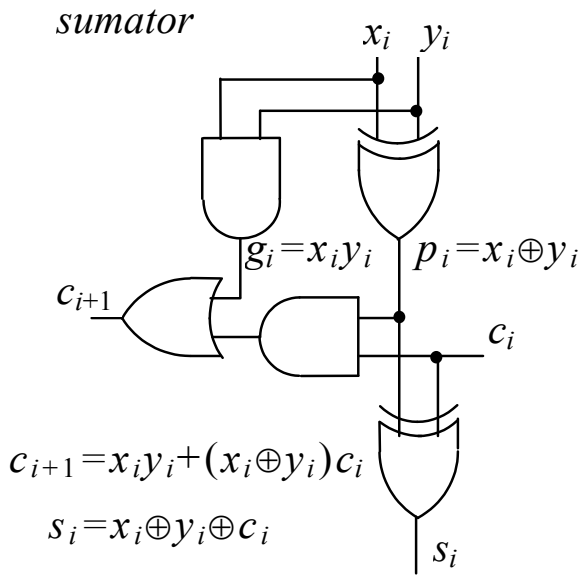
definiujące *subtraktor pełny* (*full subtracter*, FS)

*półsumator* (*half adder*, HA) – realizuje funkcje $s_i = x_i \oplus c_i, \quad c_{i+1} = x_i c_i$

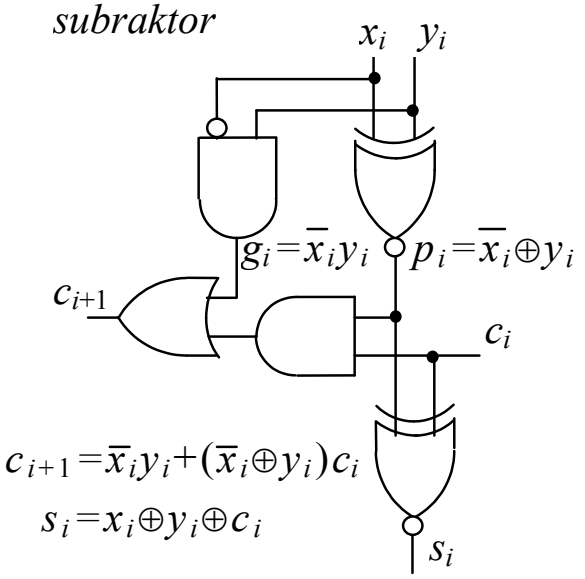
*półsubtraktor* (*half subtracter*, HS) – realizuje funkcje $s_i = x_i \oplus c_i, \quad c_{i+1} = \bar{x}_i c_i$

# Sumator, subtraktor, półsumator

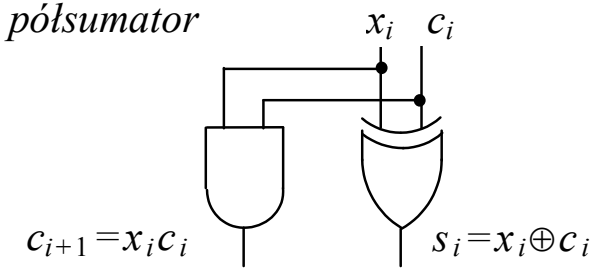
*sumator*



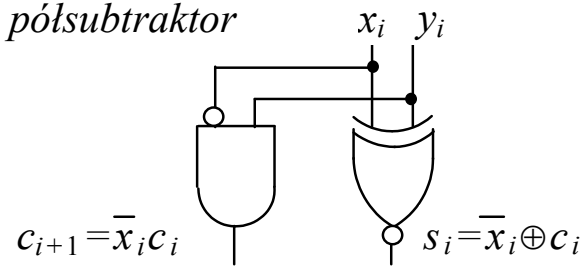

*subraktor*




*półsumator*




*półsubtraktor*



## Dodawanie wieloargumentowe w systemach naturalnych (1)

- dodawanie jest przemienne i łączne, więc:

$$X+Y+Z+\ldots=\sum_{i=0}^{n-1}x_i\beta^i+\sum_{i=0}^{n-1}y_i\beta^i+\sum_{i=0}^{n-1}z_i\beta^i+\ldots=\sum_{i=0}^{n-1}(x_i+y_i+z_i\ldots)\beta^i$$

- każda suma wartości cyfr na każdej pozycji $i$ może być zapisana jako liczba wielocyfrowa o wadze takiej jak waga pozycji ($\beta^i$):

$$x_i+y_i+\ldots+z_i=\ldots+\beta^2 r_{i+2}+\beta v_{i+1}+u_i$$

przy tym $x_i, y_i, \ldots, z_i, u_i, v_{i+1}, r_{i+2} \in \{0,1,\ldots,\beta-1\}$

- przekształcenie redukuje $m$ składników do $1+\log_\beta m$ składników

$$X+Y+\ldots=\sum_{i=0}^{n-1}(u_i+v_{i+1}+r_{i+2}\ldots)\beta^i=\sum_{i=0}^{n-1}u_i\beta^i+\sum_{i=1}^{n}v_i\beta^i+\sum_{i=2}^{n+1}r_i\beta^i+\ldots$$

- redukcja może być wykonana równolegle na poszczególnych pozycjach, co pozwala szybko zredukować sumowanie $m$ liczb $n$-pozycyjnych do sumowania dwóch liczb o rozmiarze $m+\log_\beta m$ pozycji każda
- procedurę **można powtarzać rekurencyjnie**.

## Dodawanie wieloargumentowe w systemach naturalnych (2)

Jeśli jest $\leq \beta+1$ składników jednocyfrowych, to ich suma jest dwucyfrowa:

$$\{v_{i+1}, u_i\} = \{k,\ x_i + y_i + \ldots + z_i - k\beta\} \text{ gdy } 0 \leq x_i + y_i + \ldots + z_i - k\beta < \beta,$$

Jeśli składników jest więcej procedurę **można powtarzać rekurencyjnie**, sumując cyfry o tej samej wadze w grupach małej liczności ($\leq \beta+1$)

Dodawanie liczb można wykonać etapami:

- niezależnie obliczyć sumę cyfr o tej samej wadze (na każdej pozycji),
- dodać otrzymane liczby dwucyfrowe, a jeśli tych liczb jest więcej niż $\beta+1$, powtórzyć procedurę .

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | $x_{k-1}$ | $x_{k-2}$ | $x_{k-3}$ | … | $x_{-m+3}$ | $x_{-m+2}$ | $x_{-m+1}$ | $x_{-m}$ |
| | $y_{k-1}$ | $y_{k-2}$ | $y_{k-3}$ | … | $y_{-m+3}$ | $y_{-m+2}$ | $y_{-m+1}$ | $y_{-m}$ |
| | … | … | … | … | … | … | … | … |
| $\pm$ | $z_{k-1}$ | $z_{k-2}$ | $z_{k-3}$ | … | $z_{-m+3}$ | $z_{-m+2}$ | $z_{-m+1}$ | $z_{-m}$ |
| | $u_{k-1}$ | $u_{k-2}$ | $u_{k-3}$ | … | $u_{-m+3}$ | $u_{-m+2}$ | $u_{-m+1}$ | $u_{-m}$ |
| $v_k$ | $v_{k-1}$ | $v_{k-2}$ | … | $v_{-m+4}$ | $v_{-m+3}$ | $v_{-m+2}$ | $v_{-m+1}$ | |
| $s_k$ | $s_{k-1}$ | $s_{k-2}$ | … | … | $s_{-m+3}$ | $s_{-m+2}$ | $s_{-m+1}$ | $s_{-m}$ |

## Rekurencyjne dodawanie wieloargumentowe

→ jeśli liczba argumentów $k>\beta+1$, to dodawanie można wykonać etapami

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $>\beta+1$ argumentów | | 0 | 0 | $a_{k-1}$ | $a_{k-2}$ | … | $a_3$ | $a_2$ | $a_1$ | $a_0$ |
| | | 0 | 0 | $b_{k-1}$ | $b_{k-2}$ | … | $b_3$ | $b_2$ | $b_1$ | $b_0$ |
| | | 0 | 0 | $c_{k-1}$ | $c_{k-2}$ | … | $c_3$ | $c_2$ | $c_1$ | $c_0$ |
| | | 0 | 0 | $d_{k-1}$ | $d_{k-2}$ | … | $d_3$ | $d_2$ | $d_1$ | $d_0$ |
| | | … | … | … | … | … | … | … | … | … |
| | + | 0 | 0 | $p_{k-1}$ | $p_{k-2}$ | … | $p_3$ | $p_2$ | $p_1$ | $p_0$ |
| | | … | … | … | … | … | … | … | … | … |
| | | … | … | … | … | … | … | … | … | … |
| $\leq\beta+1$ arg. | | 0 | 0 | ${}^{(0)}x_{k-1}$ | ${}^{(0)}x_{k-2}$ | … | ${}^{(0)}x_3$ | ${}^{(0)}x_2$ | ${}^{(0)}x_1$ | ${}^{(0)}x_0$ |
| | | 0 | ${}^{(1)}x_{k-1}$ | ${}^{(1)}x_{k-2}$ | … | ${}^{(1)}x_3$ | ${}^{(1)}x_2$ | ${}^{(1)}x_1$ | ${}^{(1)}x_0$ | 0 |
| | | ${}^{(2)}x_{k-1}$ | ${}^{(2)}x_{k-2}$ | … | ${}^{(2)}x_3$ | ${}^{(2)}x_2$ | ${}^{(2)}x_1$ | ${}^{(2)}x_0$ | 0 | 0 |
| | + | … | … | … | … | … | … | … | … | … |
| 2 arg | … | ${}^{(0)}u_{k+1}$ | ${}^{(0)}u_k$ | ${}^{(0)}u_{k-1}$ | ${}^{(0)}u_{k-2}$ | … | ${}^{(0)}u_3$ | ${}^{(0)}u_2$ | ${}^{(0)}u_1$ | ${}^{(0)}x_0$ |
| | … | ${}^{(1)}u_k$ | ${}^{(1)}u_{k-1}$ | ${}^{(1)}u_{k-2}$ | … | ${}^{(1)}u_3$ | ${}^{(1)}u_2$ | ${}^{(1)}u_1$ | 0 | |
| | … | $s_{k+1}$ | $s_k$ | $s_{k-1}$ | $s_{k-2}$ | | $s_3$ | $s_2$ | ${}^{(0)}u_1$ | ${}^{(0)}x_0$ |

## Dodawanie wieloargumentowe w systemach uzupełnieniowych

W dodawaniu $m$ argumentów zakres wyniku jest o $\log_2 m$ bitów większy, więc

→ jeśli liczba argumentów $m > \beta+1$ dodawanie należy wykonać rekurencyjnie

→ należy użyć co najmniej $\lceil \log_\beta m \rceil$ cyfr lewostronnego rozszerzenia

Dodawanie można wykonać dwuetapowo (z użyciem cyfr rozszerzenia):

- na każdej pozycji obliczyć wielopozycyjne (wektorowe) sumy argumentów jednocyfrowych (w dowolnej kolejności – są niezależne)
- dodać wielocyfrowe sumy z uwzględnieniem ich wag

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| … | $(x_e)$ | $(x_e)$ | $x_{k-1}$ | $x_{k-2}$ | … | $x_3$ | $x_2$ | $x_1$ | $x_0$ |
| … | $(y_e)$ | $(y_e)$ | $y_{k-1}$ | $y_{k-2}$ | … | $y_3$ | $y_2$ | $y_1$ | $y_0$ |
| … | … | … | … | … | … | … | … | … | … |
| + | $(z_e)$ | $(z_e)$ | $z_{k-1}$ | $z_{k-2}$ | … | $z_3$ | $z_2$ | $z_1$ | $z_0$ |
| | … | … | … | … | | … | … | … | … |
| | $(\ldots)$ | $(\ldots)$ | $u_{k-1}$ | $u_{k-2}$ | | $u_3$ | $u_2$ | $u_1$ | $u_0$ |
| | $(\ldots)$ | $v_k$ | $v_{k-1}$ | $v_{k-2}$ | … | $v_3$ | $v_2$ | $v_1$ | 0 |
| $(s_e)$ | $(\ldots)$ | $s_k$ | $s_{k-1}$ | $s_{k-2}$ | | $s_3$ | $s_2$ | $s_1$ | $s_0$ |

## Dodawanie wieloargumentowe w systemach uzupełnieniowych

W zapisie **uzupełnieniowym** należy użyć $\lceil \log_\beta m \rceil$ cyfr lewostronnego rozszerzenia każdego argumentu (suma ma zakres $\lceil \log_\beta m \rceil$ razy większy niż argument).

Alternatywą jest przekodowanie argumentów i korekcja sumy:

- dodanie do $k$-cyfrowej liczby całkowitej w zapisie uzupełnieniowym wartości $½\beta^k$ zamienia każdy argument na liczbę dodatnią
- suma tak otrzymanych $m$ liczb dodatnich wymaga korekcyjnego odjęcia wartości $½\, m\beta^k$, czyli liczby o wartości $½\, m\beta$ i wadze $\beta^{k-1}$
- zapis liczby $½\, m\beta$ wymaga użycia tylko $1+\lceil \log_\beta m/2 \rceil$ cyfr

W dwójkowym systemie uzupełnieniowym (**U2)**
wyeliminowanie cyfr (bitów) lewostronnego rozszerzenia w dodawaniu $m$ liczb można wykonać jako **zanegowanie wiodącego bitu** każdego operandu, bo:

$$|\mathbf{X}| = -x_{k-1}2^{k-1} + \sum_{i=0}^{k-1} x_i 2^i = -2^{k-1} + [(1-x_{k-1})2^{k-1} + x_{k-2}2^{k-2} + x_{k-3}2^{k-3} + \ldots + x_0]$$

więc $\left|\mathbf{X} = \{x_{k-1}, x_{k-2}, \ldots, x_0\}\right|_{U2} = \left|\{\overline{x}_{k-1}, x_{k-2}, \ldots, x_0\}_2 = \mathbf{X}'\right| - 2^{k-1}$, oraz odjęcie liczby $m2^{k-1}$.

# MNOŻENIE

## Skalowanie w systemach naturalnych i uzupełnieniowych

*Skalowanie* („*przesuwanie przecinka*") – mnożenie przez całkowitą potęgę podstawy $\beta^k$

- skalowania można składać ponieważ $\beta^k X = \beta \ldots \beta\beta X$
- mnożenie przez *całkowitą potęgę* podstawy powoduje cykliczne *przemieszczenie cyfr w lewo* (potęga dodatnia), lub *w prawo* (potęga ujemna), ponieważ

$$\beta^r \sum x_i \beta^i = \sum x_i \beta^{i+r} \sum x_{i-r} \beta^i$$

W reprezentacji ograniczonej, jeśli *najwyższą pozycją* liczby *nie jest* cyfra rozszerzenia $x_k = \sigma(x_{k-1})(\beta - 1)$, to w wyniku skalowania przez $\beta$ wystąpi *nadmiar*:

$$\mathbf{X}_\mathrm{e} = \{\sigma(x_{k-1})(\beta-1), x_{k-1}, x_{k-2}, \ldots, x_0\} \Rightarrow \beta \mathbf{X}_\mathrm{e} = \{x_{k-1}, x_{k-2}, \ldots, x_0, 0\},$$

Wynik skalowania odwrotnego (mnożenia przez $\beta^{-1}$) jest zawsze poprawny:

$$\mathbf{X} = \{x_{k-1}, x_{k-2}, \ldots, x_1, x_0\} \Rightarrow \beta^{-1}\mathbf{X} = \{\sigma(x_{k-1})(\beta-1), x_{k-1}, x_{k-2}, \ldots, x_1\}$$

W systemie *dwójkowym* (U2) $\sigma(x_{k-1}) = x_{k-1}$, więc

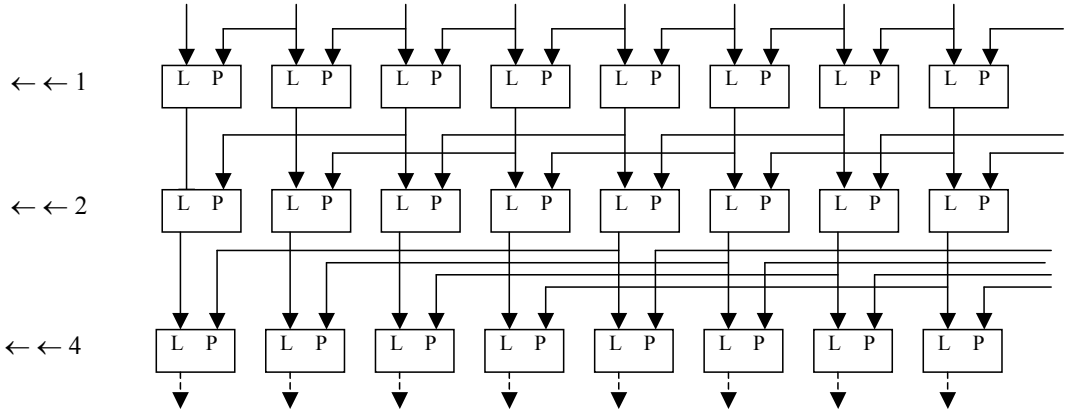
$$\mathbf{X} = \{x_{k-1}, x_{k-2}, \ldots, x_1, x_0\}_{\mathrm{U2}} \Rightarrow 2^{-1}\mathbf{X} = \{x_{k-1}, x_{k-1}, x_{k-2}, \ldots, x_1\}$$

W reprezentacji *prawostronnie skończonej* każdą liczbę można wyrazić jako *skalowaną liczbę całkowitą*: $X = \sum_{i=-s} x_i \beta^i = \beta^{-s} \sum_{i=0} x_{i+s} \beta^i = \beta^{-s} X_C$, $x_i \in \{0,1,\ldots,\beta-1\}$.

## Skalowanie jako złożenie przesunięć

Ponieważ $\beta^{k=\sum x_i 2^i} = \beta^{x_0}\beta^{2x_1}\beta^{4x_2}\ldots$ więc *mnożenie* przez $\beta^k$ jest *złożeniem przesunięć*

w lewo gdy $k>0$, albo w prawo gdy $k<0$




W systemie dwójkowym mnożenie przez podstawę jest sumą liczby i jej samej:

$$x_i + x_i + c_i = 2c_{i+1} + s_i \;\Rightarrow\; c_{i+1} = x_i,\; s_i = c_i \;\Rightarrow\; s_{i+1} = x_i$$

W systemie uzupełnieniowym, *przesunięcie logiczne* (*logical shift*):

$$\mathbf{X}_e = \{x_{k-1}, x_{k-1}, x_{k-2}, \ldots, x_1, x_0\} \Rightarrow \mathbf{X}_e + \mathbf{X}_e = \{x_{k-1}, x_{k-2}, \ldots, x_1, x_0, 0\},$$

a *przesunięcie arytmetyczne* (*arithmetic shift*) (mnożenie przez $2^{-1}$) daje w wyniku:

$$\mathbf{X}_e = \{x_{k-1}, x_{k-2}, \ldots, x_1, x_0\} \Rightarrow \tfrac{1}{2}\mathbf{X}_e = \{x_{k-1}, x_{k-1}, x_{k-2}, \ldots, x_1\}$$

## Skalowanie iloczynu

W reprezentacji *prawostronnie skończonej* każdą liczbę można wyrazić jako skalowaną liczbę całkowitą: $Z = \sum_{i=-s} z_i \beta^i = \beta^{-s} \sum_{i=0} z_{i+s} \beta^i = \beta^{-s} Z_C$, $z_i \in \{0,1,...,\beta-1\}$, gdzie $s$ określa położenie przecinka pozycyjnego ($s$ pozycji w lewo od najniższej).
Iloczynem $\mathbf{A} = \{...,a_{i+1},a_i,...,a_{-s}\}_\beta$ i $\mathbf{X} = \{...,x_{j+1},x_j,...,x_{-r}\}_\beta$ jest
$AX = (\beta^{-s} A_C)(\beta^{-r} X_C) = \beta^{-(s+r)} A_C X_C$.

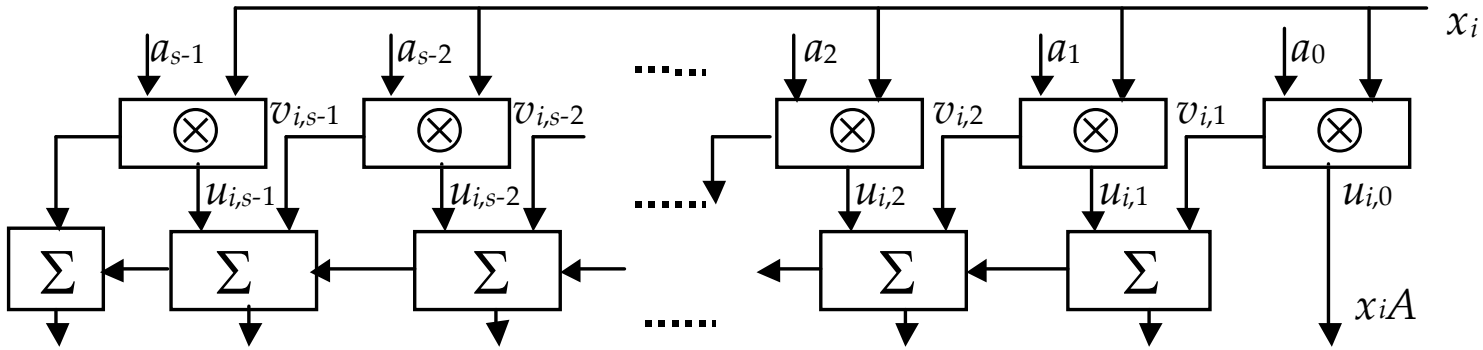
Iloczynem liczb całkowitych $A = |\{a_{s-1},...,a_1,a_0\}_\beta|$ oraz $X = |\{x_{k-1},...,x_1,x_0\}_\beta|$ jest

$$A \cdot X = A \cdot \left( \sum_{i=0}^{k-1} x_i \beta^i \right) = \sum_{i=0}^{k-1} \beta^i (x_i A)$$

Iloczyn częściowy $x_i A = x_i a_0 \beta^0 + x_i a_1 \beta^1 + x_i a_2 \beta^2 + ... + x_i a_{s-1} \beta^{s-1}$ jest wielokrotnością mnożnej (ang. *multiplicand*) przez wartość cyfry mnożnika (ang. *multiplier*) i jaki taki jest sumą skalowanych iloczynów elementarnych $x_i a_j$, z których każdy może być liczbą dwucyfrową $x_i a_j = u_{i,j} + v_{i,j+1}\beta$, gdzie $u_{i,j} = x_i a_j \bmod \beta$, $v_{i,j} = x_i a_j \operatorname{int} \beta$
Algorytm pisemny – akumulacja skalowanych iloczynów częściowych ($S_0 = 0$)

$$S_{i+1} = S_i + \beta^i (x_i A), \quad i=0,1,...,k-1, \quad S_k = A \cdot X$$

## Sekwencyjny algorytm mnożenia w systemie naturalnym




*Iloczyn elementarny* $x_i a_j = u_{i,j} + v_{i,j+1}\beta$ można interpretować jako dwa wektory cyfr reprezentujących liczby w zapisie pozycyjnym $\{u_{i,s-1},...,u_{i,1},u_{i,0}\}$, $\{v_{i,s},...,v_{i,2},v_{i,1}\}$.

Algorytm *dodaj-przesuń* (*add-and-shift*) – skalowanie sum częściowych

$$P_i = \beta^{-i} S_i$$

i wtedy

$$P_{i+1} = \beta^{-1}(P_i + x_i A)$$

$$\beta^k P_k = P_0 + A\{\sum_{i=0}^{k-1} x_i \beta^i\} = A \cdot X$$

Dodawanie iloczynów częściowych jest dodawaniem wieloargumentowym.

## Mnożenie sekwencyjne w systemach uzupełnieniowych

Wartością mnożnika w systemie uzupełnieniowym jest

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

$\sigma(x_{k-1}) = \frac{1}{2}(1+\mathrm{sgn}(2x_{k-1}+1-\beta))$ – funkcja znaku (1 gdy $X<0$ albo 0 gdy $X\geq 0$). Zatem:

$$A \cdot X = A \cdot \left( -\sigma(x_{k-1})\beta^k + \sum_{i=0}^{k-1} x_i \beta^i \right) = \sigma(x_{k-1})(-A)\beta^{k-1} + \sum_{i=0}^{k-1} \beta^i (x_i A)$$

WNIOSKI:

W mnożeniu w systemach uzupełnieniowych rozszerzenie mnożnika zapewnia, że wartością najwyższej cyfry jest zawsze 0 albo –1 („$\beta-1$”).

W mnożeniu przez $k$-cyfrowy mnożnik należy użyć $k$ cyfr lewostronnego rozszerzenia mnożnej (akumulacja iloczynów jest na $k+m$ pozycjach)

Inny sposób:

Funkcja $z_{k-1} = x_{k-1} - \beta\sigma(x_{k-1})$ opisuje przekształcenie zbioru $\mathbf{D}=\{0,1,\ldots,\beta-1\}$ **na** nieredundantny zbiór cyfr znakowanych $\mathbf{D}_S = \{-\frac{1}{2}\beta,\ldots,-1,0,1,\ldots,\frac{1}{2}\beta-1\}$

# Mnożenie pisemne w dziesiętnym systemie uzupełnieniowym – przykład

*bez rozszerzenia mnożnika*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $A=-124$ | | 9 | 9 | 9 | 9 | 9 | 8 | 7 | 6 |
| $X=-231$ | | | | | | 7 | 6 | 9 | |
| $x_0=9$ | | 9 | 9 | 9 | 9 | 8 | 8 | 8 | 4 |
| $x_1=6$ | | 9 | 9 | 9 | 9 | 2 | 5 | 6 | |
| $x_2=-3$ | $-3A$ | 0 | 0 | 0 | 3 | 7 | 2 | | |
| $X\cdot A$ | | 0 | 0 | 0 | 2 | 8 | 6 | 4 | 4 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $A=124$ | | | | | 0 | 1 | 2 | 4 |
| $X=-231$ | | | | | | 7 | 6 | 9 |
| $x_0=9$ | | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 6 |
| $x_1=6$ | | 0 | 0 | 0 | 0 | 7 | 4 | 4 | |
| $x_2=-3$ | $-3A$ | 9 | 9 | 9 | 6 | 2 | 8 | | |
| $X\cdot A$ | | 9 | 9 | 9 | 7 | 1 | 3 | 5 | 6 |

*z rozszerzeniem mnożnika*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $A=-124$ | | 9 | 9 | 9 | 9 | 9 | 8 | 7 | 6 |
| $X=-231$ | | | | | 9 | 7 | 6 | 9 | |
| $x_0=9$ | | 9 | 9 | 9 | 9 | 8 | 8 | 8 | 4 |
| $x_1=6$ | | 9 | 9 | 9 | 9 | 2 | 5 | 6 | |
| $x_2=7$ | | 9 | 9 | 9 | 1 | 3 | 2 | | |
| $x_3=-1$ | $-A$ | 0 | 0 | 1 | 2 | 4 | | | |
| $X\cdot A$ | | 0 | 0 | 0 | 2 | 8 | 6 | 4 | 4 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $A=124$ | | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 4 |
| $X=-231$ | | | | | 9 | 7 | 6 | 9 | |
| $x_0=9$ | | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 6 |
| $x_1=6$ | | 0 | 0 | 0 | 0 | 7 | 4 | 4 | |
| $x_2=7$ | | 0 | 0 | 0 | 8 | 6 | 8 | | |
| $x_3=-1$ | $-A$ | 9 | 9 | 8 | 7 | 6 | | | |
| $X\cdot A$ | | 9 | 9 | 9 | 7 | 1 | 3 | 5 | 6 |

## Algorytm mnożenia sekwencyjnego w systemach uzupełnieniowych

Algorytm mnożenia *dodaj-przesuń* (*add-and-shift*) – $[x_{k-1} = (\beta - 1)\sigma(x_{k-1})]$

0. $x_k = (\beta - 1)\sigma(x_{k-1})$ ; dopisz pozycję rozszerzenia
1. $P_0 = 0$, $i = 0$
2. $M_i = x_i A$ ; oblicz iloczyn częściowy
3. $P_{i+1} = \beta^{-1}(P_i + M_i)$ ; przeskaluj (przesuń) sumę częściową
4. $i$++ ; zwiększ $i$
5. **if** $i<k$ **goto** 3 ; powtarzaj do przedostatniego
6. $P_{k+1} = P_k + \sigma(x_k)(-A)$ ; dodaj dopełnienie mnożnej lub 0
7. $A \cdot X = \beta^k P_{k+1}$ ; iloczyn

!! *UWAGA*: Wszystkie sumy częściowe są przesuwane w prawo
*więc* muszą być obliczane z lewostronnym *rozszerzeniem*

# Mnożenie „dodaj-przesuń" w dziesiętnym systemie uzupełnieniowym

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $A=-124$ | | 9 | 9 | 8 | 7 | 6 | | |
| $X=-231$ | | | | 7 | 6 | 9 | | |
| $x_0=9$ | $+9A$ | 9 | 8 | 8 | 8 | 4 | | |
| | $\rightarrow$ | 9 | 9 | 8 | 8 | 8 | 4 | |
| $x_1=6$ | $+6A$ | 9 | 9 | 2 | 5 | 6 | | |
| | | 9 | 9 | 1 | 4 | 4 | 4 | |
| | $\rightarrow$ | 9 | 9 | 9 | 1 | 4 | 4 | 4 |
| $x_2=-3$ | $-3A$ | 0 | 0 | 3 | 7 | 2 | | |
| | | 0 | 0 | 2 | 8 | 6 | 4 | 4 |
| $X\cdot A$ | $\rightarrow$ | 0 | 0 | 0 | 2 | 8 | 6 | 4 | 4 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $A=124$ | | 0 | 1 | 2 | 4 | | | |
| $X=-231$ | | | | 7 | 6 | 9 | | |
| $x_0=9$ | $+9A$ | 0 | 1 | 1 | 1 | 6 | | |
| | $\rightarrow$ | 0 | 0 | 1 | 1 | 1 | 6 | |
| $x_1=6$ | $+6A$ | 0 | 0 | 7 | 4 | 4 | | |
| | | 0 | 0 | 8 | 5 | 5 | 6 | |
| | $\rightarrow$ | 0 | 0 | 0 | 8 | 5 | 5 | 6 |
| $x_2=-3$ | $-3A$ | 9 | 9 | 6 | 2 | 8 | | |
| | | 9 | 9 | 7 | 1 | 3 | 5 | 6 |
| $X\cdot A$ | | 9 | 9 | 9 | 7 | 1 | 3 | 5 | 6 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | … | | … | | |
| | $\rightarrow$ | 9 | 9 | 9 | 1 | 4 | 4 | |
| $x_2=7$ | $+7A$ | 9 | 9 | 1 | 3 | 2 | | |
| | | 9 | 9 | 0 | 4 | 6 | 4 | 4 |
| | $\rightarrow$ | 9 | 9 | 9 | 0 | 4 | 6 | 4 | 4 |
| $x_3=-1$ | $-A$ | 0 | 0 | 1 | 2 | 4 | | |
| $X\cdot A$ | | 0 | 0 | 0 | 2 | 8 | 6 | 4 | 4 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | … | | … | | |
| | $\rightarrow$ | 0 | 0 | 0 | 8 | 5 | 5 | 6 |
| $x_2=7$ | $+7A$ | 0 | 0 | 8 | 6 | 8 | | |
| | | 0 | 0 | 9 | 5 | 3 | 5 | 6 |
| | $\rightarrow$ | 0 | 0 | 0 | 9 | 5 | 3 | 5 | 6 |
| $x_3=-1$ | $-A$ | 9 | 9 | 8 | 7 | 6 | | |
| $X\cdot A$ | | 9 | 9 | 9 | 7 | 1 | 3 | 5 | 6 |

## Mnożenie maszynowe w dwójkowych systemach uzupełnieniowych

W dwójkowych systemach uzupełnieniowych $\sigma(x_{k-1}) = x_{k-1}$, więc

$$X = -x_{k-1}2^{k-1} + \sum_{i=0}^{k-2} x_i 2^i$$

1. $P_0$=0, $i$=0
2. $M_i = x_i A$ ; oblicz iloczyn częściowy
3. $S_{i+1} = P_i + M_i$ ; oblicz sumę częściową
4. $P_{i+1} = 2^{-1} S_{i+1}$ ; przeskaluj sumę (przesuń w prawo)
5. $i$++ ; zwiększ $i$
6. **if** $i$<$k$–1 **goto** 3 ; powtarzaj do przedostatniego
7. $P_k = P_{k-1} + x_{k-1}(-A)$ ; dodaj dopełnienie mnożnej lub 0
8. $A \cdot X = 2^{k-1} P_k$ ; iloczyn

- działania na wszystkich pozycjach sum częściowych

# Mnożenie w dwójkowym systemie uzupełnieniowym (U2)

*Mnożenie pisemne*

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $A=-7$ | | | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| $X=-5$ | | | | | | | 1 | 0 | 1 | 1 |
| $x_0=1$ | | | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| $x_1=1$ | | | 1 | 1 | 1 | 1 | 0 | 0 | 1 | |
| $x_2=0$ | | | 0 | 0 | 0 | 0 | 0 | 0 | | |
| $x_3=1$ | $(-A)$ | | 0 | 0 | 1 | 1 | 1 | | | |
| $X\cdot A=35$ | | | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $A=+5$ | | | | | | | 0 | 1 | 0 | 1 |
| $X=-3$ | | | | | | | 1 | 1 | 0 | 1 |
| $x_0=1$ | | | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| $x_1=0$ | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $x_2=1$ | | | 0 | 0 | 0 | 1 | 0 | 1 | | |
| $x_3=1$ | $(-A)$ | | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| $X\cdot A=-15$ | | | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

*Mnożenie maszynowe*

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $A=-7$ | | | | 1 | 0 | 0 | 1 | | |
| $X=-5$ | | | | 1 | 0 | 1 | 1 | | |
| $P_0=0$ | | | 0 | 0 | 0 | 0 | 0 | | |
| $x_0=1$ | $+A$ | + | 1 | 1 | 0 | 0 | 1 | | |
| | | | 1 | 1 | 0 | 0 | 1 | | |
| (Shr) | $\rightarrow$ | | 1 | 1 | 1 | 0 | 0 | 1 | |
| $x_1=1$ | $+A$ | + | 1 | 1 | 0 | 0 | 1 | 0 | |
| | | | 1 | 0 | 1 | 0 | 1 | 1 | |
| | $\rightarrow$ | | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| $x_2=0$ | $\rightarrow$ | | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| $x_3=1$ | $-A$ | + | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| $X\cdot A=35$ | | | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $A=+5$ | | | | 0 | 1 | 0 | 1 | | |
| $X=-3$ | | | | 1 | 1 | 0 | 1 | | |
| $P_0=0$ | | | 0 | 0 | 0 | 0 | 0 | | |
| $x_0=1$ | $+A$ | | 0 | 0 | 1 | 0 | 1 | | |
| | | | 0 | 0 | 1 | 0 | 1 | | |
| | $\rightarrow$ | | 0 | 0 | 0 | 1 | 0 | 1 | |
| $x_1=0$ | $\rightarrow$ | | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| $x_2=1$ | $+A$ | + | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| | | | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| | $\rightarrow$ | | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| $x_3=1$ | $-A$ | + | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| $X\cdot A=-15$ | | | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

## Uproszczenie mnożenia w dwójkowym systemie uzupełnieniowym

W dwójkowym systemie uzupełnieniowym

$$\left[-x_{k-1}2^{k-1}+\sum_{i=0}^{k-2}x_i 2^i\right]+2^{k-1}=(1-x_{k-1})2^{k-1}+\sum_{i=0}^{k-2}x_i 2^i \geq 0$$

Iloczyn można więc przedstawić jako sumę przekształconych iloczynów częściowych ($x_i A+2^{k-1}$) $2^i$ pomniejszoną o sumę ($2^{k-1}+2^{k-1}+\ldots+2^{k+m-1}$):

$$AX=-2^{m-1}x_{m-1}A+\sum_{i=0}^{m-1}2^i x_i A=2^{m-1}(x_{m-1}(-A)^+-2^{k-1})+\sum_{i=0}^{m-1}2^i(x_i A^+-2^{k-1}).$$

a zatem:

$$AX=[2^{m-1}(x_{m-1}(-A)^+)+\sum_{i=0}^{m-1}2^i(x_i A^+)]+[-2^{m+k-1}+2^{k-1}]$$

→ *algorytm* (*Baugh'a-Wooley'a*)

- zastąpić bit wiodący iloczynu częściowego (mnożnej, jej uzupełnienia lub zera) jego dopełnieniem (zero→10...00)
- dodać stałą korekcyjną $2^{k-1}-2^{m+k-1}$ („1" na pozycji najwyższego bitu mnożnej i *pozycjach* wyższych od najwyższego bitu ostatniego iloczynu)

# Schemat dodawania iloczynów częściowych w kodzie U2

a)

| | A | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | * | * | * | * | * | * | * | o | o | o | o | o |
| | * | * | * | * | * | * | o | o | o | o | o | |
| | * | * | * | * | * | o | o | o | o | o | | |
| | * | * | * | * | o | o | o | o | o | | | |
| | * | * | * | o | o | o | o | o | | | | |
| | ~~*~~ | ~~*~~ | ~~o~~ | ~~o~~ | ~~o~~ | ~~o~~ | ~~o~~ | | | | | |
| | | | | | | | | | | | | |

b)

| | A | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | • | o | o | o | o | o |
| | | | | | • | o | o | o | o | o | |
| | | | | • | o | o | o | o | o | | |
| | | | • | o | o | o | o | o | | | |
| | | • | o | o | o | o | o | | | | |
| | ~~•~~ | ~~o~~ | ~~o~~ | ~~o~~ | ~~o~~ | ~~o~~ | | | | | |
| + | (1) 0 | 0 | 0 | 0 | 0 | 1 | | | | | |

Matryca iloczynów częściowych: a) z rozszerzeniem (∗ – bit najwyższy i jego kopia), b) bez rozszerzeń (• – dopełnienie najbardziej znaczącego bitu)

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 1 | | 1 | 1 | 0 | 1 | 1 | 0 | 1 | | |
| 1 | | 0 | 1 | 0 | 0 | 1 | 1 | | | |
| | | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | **0** | 0 | 1 | 1 | 0 | 1 |
| | | **1** | 0 | 0 | 0 | 0 | 0 | |
| | **0** | 0 | 1 | 1 | 0 | 1 | | |
| **1** | 1 | 0 | 0 | 1 | 1 | | | |
| (1) 0 | 0 | 0 | 1 | | | | | |
| (0) 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |

## Schemat mnożenia binarnego metodą „dodaj-przesuń"

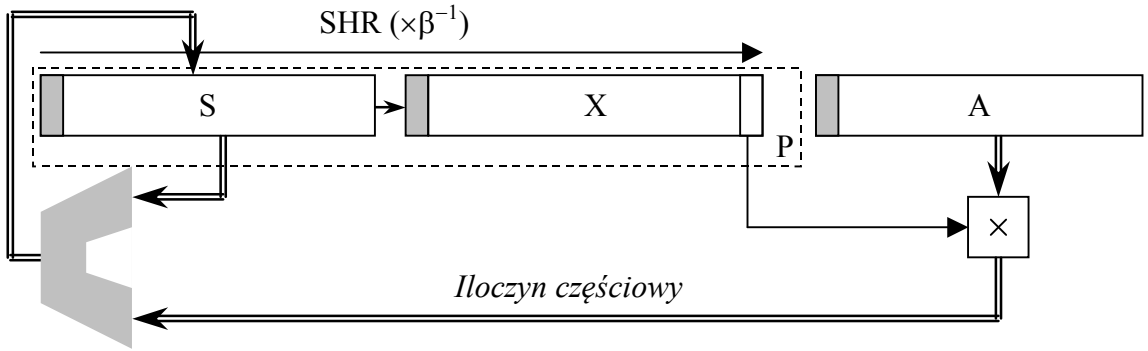
A ← **A**, X/P ← **X**, S/P ← **0**, $i$=1.

*Krok* 1. S/P ← (S/P) + $x_i$(A)

*Krok* 2. S/P||X/P ← $\beta^{-1}$(S/P||X/P) = S-R(S/P||X/P)

*Krok* 3. $i$=$i$+1. Jeśli $i < k+m$, wróć do kroku 1.

*Wynik*: $A \cdot X$=(S/P||X/P)




Schemat blokowy układu mnożącego metodą „*dodaj-przesuń*" (ang. *add-and-shift*) :
S – rejestr sum częściowych, X – rejestr mnożnika, A – rejestr mnożnej, P – rejestr iloczynu

## Zakres iloczynu w systemie naturalnym i uzupełnieniowym

Wynik mnożenia $k$-pozycyjnej mnożnej przez $p$-pozycyjny mnożnik
można zawsze zapisać na $k+p$ pozycjach iloczynu.

$$-2^{k-1} \le A < 2^{k-1},\ -2^{p-1} \le X < 2^{p-1} \Rightarrow -2^{k+p-2} < AX \le 2^{k+p-2}$$

($A$ oraz $X$ są argumentami przeskalowanymi do wartości całkowitych)

W układach cyfrowych wymaga się często,
aby operandy (argumenty i wynik działania) były takiego samego rozmiaru.
Przekroczenie zakresu odpowiadającego rozmiarowi operandu jest zwykle sygnalizowane jako nadmiar.

Wyróżnia się ponadto:

- mnożenie *dolne* – wynikiem jest niższa część (połowa bitów) iloczynu
  sygnalizacja nadmiaru jeśli wyższe bity nie są rozszerzeniem
- mnożenie *górne* – wynikiem jest wyższa część (połowa bitów) iloczynu,
  odpowiada mnożeniu ułamków z obcięciem niższych bitów,
  nadmiar nie może wystąpić
- mnożenie *z normalizacją* – ustalona liczba bitów części całkowitej

## Redukcja liczby iloczynów częściowych – przekodowanie Booth'a

(*Andrew D.Booth*, 1949)

- zastąpienie serii dodawań jednym odejmowaniem i jednym dodawaniem

$$2^{s-1} + 2^{s-2} + \ldots + 2^{l+1} + 2^{l} = 2^{s} - 2^{l}$$

$$|\{...0[11...11]0...\}_{U2}| = |\{...1[00...00]0...\}_{U2}| - |\{...0[00...01]0...\}_{U2},$$

$$|\{...0[11...11]0...\}_{SD2}| = |\{...1[00...0\underline{1}]0...\}_{SD2}|$$

$\rightarrow$ reguła Booth'a $\equiv$ *przekodowanie mnożnika* na kod SD

- reprezentacja w systemie NB lub U2 jest reprezentacją w systemie SD, ale

*przekodowanie* według reguły Booth'a:

- U2 $\rightarrow$ SD – *wykonalne*, bo $[x1...11]0... = [(1-x)0...00]0... - [00...01]0...$
- NB $\rightarrow$ SD – *niewykonalne bez rozszerzenia* gdy $\{1,1,\ldots,1,0,x,\ldots\}$, bo

$$x \geq 0 \wedge z \neq 1 \Rightarrow |\{1,x,...,x,x\}_{SD}| > |\{z,y,y,...,y\}_{SD}|$$

$\Rightarrow$ konieczne rozszerzenie $\{1,\ldots,1,0,x,\ldots\}_{NB} = \{\mathbf{0},1,\ \ldots,1,0,x,\ldots\}_{U2}$

## Algorytm Booth'a

Uzasadnienie teoretyczne – równoważność $X=2X-X$ ($x_{i>n-1}=x_e=x_{n-1}$, $x_{i<0}=0$)

$$X=\left[\sum_{i=n+1}^{\infty}x_{n-1}2^i+\sum_{i=0}^{n-1}x_i2^{i+1}\right]-\left[\sum_{i=n}^{\infty}x_{n-1}2^i+\sum_{i=0}^{n-1}x_i2^i\right]=\sum_{i=0}^{n-1}(x_{i-1}-x_i)2^i=2\sum_{i=0}^{n-2}(x_i-x_{i+1})2^i-x_0$$

| | | | $(2^{n-1})$ | | $(2^{i+1})$ | $(2^i)$ | $(2^{i-1})$ | | $(2^1)$ | $(2^0)$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | $2X_{U2}$ | $x_{n-1}$ | $x_{n-2}$ | … | $x_i$ | $x_{i-1}$ | $x_{i-2}$ | … | $x_0$ | 0 | $(x_{-1}=0)$ |
| | $-X_{U2}$ | $-x_{n-1}$ | $-x_{n-1}$ | … | $-x_{i+1}$ | $-x_i$ | $-x_{i-1}$ | … | $-x_1$ | $-x_0$ | |
| proste | $Y_{SD2}$ | (0) | $y_{n-1}$ | … | $y_{i+1}$ | $y_i$ | $y_{i-1}$ | … | $y_1$ | $y_0$ | $y_i=x_{i-1}-x_i\in\{\underline{1},0,1\}$ |
| przesunięte | $2Y_{SD2}+x_0$ | (0) | $y_{n-2}$ | … | $y_i$ | $y_{i-1}$ | $y_{i-2}$ | … | $y_0$ | $-x_0$ | $y_i=x_i-x_{i+1}\in\{\underline{1},0,1\}$ |

*Wady*:

*– zmienna liczba działań arytmetycznych, zależna od kodu liczby,*

– nieefektywne kodowanie izolowanych jedynek –...010101(0) → …1$\underline{1}$1$\underline{1}$1$\underline{1}$.

*Sąsiednie cyfry* $y_{i+1}$, $y_i$ *są różne* lub równe 00 (00,01,10,0$\underline{1}$,$\underline{1}$0,1$\underline{1}$,$\underline{1}$1), ale 1$\underline{1}$=01 i $\underline{1}$1=0$\underline{1}$

WNIOSEK (*przekodowanie Booth'a Mc Sorley'a*):

Możliwe jest przekodowanie mnożnika, które zawiera co najmniej połowę zer.

Maksymalizacja liczby zer: *przekodowanie kanoniczne* (czasochłonne)

## Usprawnienie – algorytm Booth'a-McSorley'a

Wynik algorytmu Booth'a można przekodować tak,
aby w *każdej parze* sąsiednich cyfr wystąpiło *najmniej jedno zero* (00,01,10,0$\underline{1}$,$\underline{1}$0).

Wynikiem przekodowania na kolejnych parach pozycji $y_{i+1}$, $y_i$ jest:

| | | | | $(2^{k-1})$ | $(2^{k-2})$ | | $(2^{i+1})$ | $(2^{i})$ | $(2^{i-1})$ | $(2^{i-2})$ | | $(2^{2})$ | $(2^{1})$ | $(2^{0})$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $2X_{U2}$ | $x_{k-1}$ | $x_{k-1}$ | $x_{k-2}$ | $x_{k-3}$ | … | $x_i$ | $x_{i-1}$ | $x_{i-1}$ | … | … | $x_1$ | $x_0$ | 0 |
| | $-X_{U2}$ | $-x_{k-1}$ | $-x_{k-1}$ | $-x_{k-1}$ | $-x_{k-2}$ | … | $-x_{i+1}$ | $-x_i$ | $-x_i$ | … | … | $-x_2$ | $-x_1$ | $-x_0$ |
| proste | $Y_{SD2}$ | | (0) | $y_{k-1}$ | $y_{k-2}$ | … | $y_{i+1}$ | $y_i$ | $y_{i-1}$ | $y_{i-2}$ | … | | $y_1$ | $y_0$ |
| przesunięte | $2Y_{SD2}+x_0$ | | (0) | $y_{k-2}$ | $y_{k-3}$ | … | $y_i$ | $y_{i-1}$ | $y_{i-2}$ | $y_{i-3}$ | … | $y_1$ | $y_0$ | $x_0$ |

$$2^{i+1}(x_i-x_{i+1})+2^i(x_{i-1}-x_i)=2^i[-2x_{i+1}+x_i+x_{i-1})]=2^i(2y_{i+1}+y_i)\in\{\underline{2},\underline{1},0,1,2\}$$

Jeśli $i$ jest parzyste ($2s$), to $z_s=2y_{2s+1}+y_{2i}$=wartość cyfry liczby Z w bazie $2^2=4$:

$$\sum_{i=0}^{n-1}(x_{i-1}-x_i)2^i=\sum_{s=0}^{n/2}(2y_{2s+1}+y_{2s})2^{2s}=\sum_{s=0}^{n/2}z_s4^s$$

Jeśli $i$ jest nieparzyste ($2s+1$), to $z_S=2y_{2s+2}+y_{2s+1}$=wartość cyfry liczby 2Z w bazie $2^2=4$:

$$\sum_{i=0}^{n-1}(x_{i-1}-x_i)2^i=\sum_{s=0}^{n/2}(x_{2s}+x_{s2+1}-2x_{2s+2})2^{2s+1}-x_0=\sum_{s=0}^{n/2}(2y_{2s+2}+y_{2s+1})2^{2s+1}-x_0=2\sum_{s=0}^{n/2}z_s4^s-x_0$$

## Algorytm Booth'a i Booth'a-McSorley'a – przykłady

**X**={(1),1,0,1,1,0,0,1}$_{U2}$ – w bazie 2 – **Y**={$\underline{1}$,1,0,$\underline{1}$,0,1,$\underline{1}$}$_{SD2}$

– alternatywne w bazie 4 – **Y**={0$\underline{1}$,10,$\underline{1}$0,01}$_{SD4}$

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $A=-19$ | | | | | | | ... | ... | 1 | 0 | 1 | 1 | 0 | 1 |
| $X=-39$ | | | | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| $-A$ | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* | 1 | 0 | 0 | 1 | 1 | |
| $+A$ | *1* | *1* | *1* | *1* | *1* | *1* | 1 | 0 | 1 | 1 | 0 | 1 | | |
| 0 | *0* | *0* | *0* | *0* | *0* | 0 | 0 | 0 | 0 | 0 | 0 | | | |
| $-A$ | *0* | *0* | *0* | *0* | *0* | 1 | 0 | 0 | 1 | 1 | | | | |
| 0 | *0* | *0* | *0* | 0 | 0 | 0 | 0 | 0 | 0 | | | | | |
| $+A$ | *1* | *1* | 1 | 0 | 1 | 1 | 0 | 1 | | | | | | |
| $-A$ | *0* | 0 | 1 | 0 | 0 | 1 | 1 | | | | | | | |
| $XA=741$ | *0* | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | ... | ... | 1 | 0 | 1 | 1 | 0 | 1 |
| | | | | | | | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| $A$ | *1* | *1* | *1* | *1* | *1* | *1* | *1* | 1 | 0 | 1 | 1 | 0 | 1 | |
| $-2A$ | *0* | *0* | *0* | *0* | *0* | 1 | 0 | 0 | 1 | 1 | | | | |
| $2A$ | *1* | *1* | 1 | 0 | 1 | 1 | 0 | 1 | | | | | | |
| $-A$ | *0* | *0* | 1 | 0 | 0 | 1 | 1 | | | | | | | |
| | *0* | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | |

*Uwaga*: W polach zacienionych wpisano cyfry rozszerzenia znakowego.

## Alternatywny algorytm Booth'a i Booth'a-McSorley'a – przykłady

Efektem przekodowania przesuniętego (alternatywnego) jest ustalenie początkowej wartości sumy iloczynów częściowych jako $P_0=-x_0A$ zamiast $P_0=0$.

**X**={1,1,0,1,0,1}$_{U2}$ – w bazie 2 – **Y**={0,$\underline{1}$,1,$\underline{1}$,1}$_{SD2}$,

– alternatywnie w bazie 4 – **Y**={00,0$\underline{1}$,$\underline{01}$}$_{SD4}$, $P_0=-x_0A$

| | **Y**$^{(2R)}$ | | | | | | | | | | | **Y**$^{(4R)}$ | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $A=-3$ | | | | | | .. | 1 | 1 | 1 | 1 | 0 | 1 | | | | | | .. | 1 | 1 | 1 | 1 | 0 | 1 |
| $X=-11$ | | | | | | | | 0 | 1 | 1 | 1 | 1 | | | | | | | 0 | 0 | 0 | 1 | 0 | 1 |
| $P_0=$ | $-x_0A$ | *0* | *0* | *0* | *0* | *0* | 0 | 0 | 0 | 0 | 1 | 1 | $-x_0A$ | *0* | *0* | *0* | 0 | 0 | 0 | 0 | 1 | 1 |
| | $+2A$ | *1* | *1* | *1* | *1* | 1 | 1 | 1 | 1 | 0 | 1 | | $-2A$ | *0* | *0* | 0 | 0 | 0 | 0 | 1 | 1 | |
| | $-2A$ | *0* | *0* | *0* | *0* | 0 | 0 | 0 | 1 | 1 | | | $-2A$ | 0 | 0 | 0 | 0 | 1 | 1 | | | |
| | $+2A$ | *1* | *1* | 1 | 1 | 1 | 1 | 0 | 1 | | | | | *0* | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| | $-2A$ | *0* | 0 | 0 | *0* | 0 | 1 | 1 | | | | | | | | | | | | | | |
| $XA=33$ | | *0* | *0* | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | | | | | | | | | | |

*Uwaga*: W polach zacienionych wpisano cyfry rozszerzenia znakowego.

# REALIZACJA PROGRAMOWA

## Programowa realizacja działań rozszerzonej precyzji

Słowo procesora o rozmiarze $n$ bitów reprezentuje
**jedną cyfrę** systemu liczbowego o podstawie $\mathbf{2}^{n}$

**Wniosek:**
**W systemie naturalnym:**
**działania rozszerzonej precyzji można opisać jako sekwencję powiązanych ze sobą działań jednopozycyjnych**
**W systemie uzupełnieniowym:**
**konieczne uwzględnienie cyfr rozszerzenia nieskończonego**

**W dodawaniu i odejmowaniu – przeniesienie można interpretować jako cyfrę wyższej wagi o wartości 0 lub 1**

**W mnożeniu – iloczyn liczb 1-cyfrowych jest liczbą 2-cyfrową**
**(wyjątek: system dwójkowy)**
**iloczyn dolny – niższa cyfra iloczynu 2-cyfrowego**
**iloczyn górny – wyższa cyfra iloczynu 2-cyfrowego**

## 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
```

## Zmiana znaku i wartość bezwzględna liczby rozszerzonej precyzji

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

```
.type exabs @function                  # definicja funkcji
exabs:  push %ebp                      # .....
        movl %esp, %ebp                # wskaźnik parametrów wywołania
        movl 8(,%ebp,4), %ecx          # rozmiar argumentu ze stosu
        movl 12(,%ebp,4), %ebx         # adres argumentu ze stosu
        add $0, -4(%ebx,%ecx,4)        # sprawdzenie znaku liczby
        jge end                        # koniec, jeśli dodatnia
                                       # zmiana znaku
        movl $-1, %esi                 # wartość początkowa wskaźnika
        stc                            # ustawienie CF=1 (ulp)
next:   inc %esi
        notl (%ebx,%esi,4)             # dopełnienie (negacja bitów) cyfry
        adc $0, (%edx,%esi,4)          # dodawanie ulp
        loop next                      # licznik pętli w %ecx
end:    movl %ebp, %esp                # przywrócenie wskaźników
        pop %ebp
        ret                            # wskaźnik nadmiaru w OF
```

## Dekrementacja/inkrementacja liczby rozszerzonej precyzji

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

```
.type exdec @function                  # definicja funkcji
exdec:  push %ebp                      # .....
        movl %esp, %ebp                # wskaźnik parametrów wywołania
        movl 8(,%ebp,4), %ecx          # rozmiar argumentu ze stosu
        movl 12(,%ebp,4), %ebx         # adres argumentu ze stosu
        movl $–1, %esi                 # wartość początkowa wskaźnika
        stc                            # ustawienie CF=1 (ulp)
next:   inc %esi
        sbb $0, (%edx,%esi,4)          # inkrementacja: adc zamiast sbb
        loop next                      # licznik pętli w %ecx
end:    movl %ebp, %esp                # przywrócenie wskaźników
        pop %ebp
        ret                            # wskaźnik nadmiaru w OF
```

albo:

```
        clc                            # dekrementacja przez dodanie -1
next:   inc %esi
        adc $-1, (%edx,%esi,4)         # –1=(1) ciąg „1” takiej długości
        loop next                      #
```

## Mnożenie rozszerzonej precyzji – kontekst funkcji

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

```
.type exadd @function
exmul:  push %ebp                   # definicja funkcji
        movl %esp, %ebp             # .....
        movl 8(,%ebp, 4), %ecx      # wskaźnik parametrów wywołania
        movl 12(,%ebp, 4), %esi     # rozmiar mnożnej ze stosu
        movl 16(,%ebp, 4), %eax     # adres mnożnej ze stosu
        movl 20(,%ebp, 4), %edi     # rozmiar mnożnika ze stosu
        movl 24(,%ebp, 4), %ebx     # adres mnożnika ze stosu
                                    # adres iloczynu ze stosu

        call prod                   # obliczenie iloczynu

        movl %ebp, %esp             # przywrócenie wskaźników
        push %ebp
        ret
```

- mnożenie uzupełnieniowe przez przekodowanie na zapis znak-moduł

## Mnożenie rozszerzonej precyzji – algorytm

- 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
```