Wykład 3

Implementacja działań arytmetycznych i logicznych

- **Flagi przy działaniach arytmetycznych**
- **Mnożenie i dzielenie w zapisach stałoprzecinkowych**
- **Złożone działania arytmetyczne**
- **Implementacja tablicy funkcyjnej**
- **Działania w podwyższonej precyzji**
- **Operacje na argumentach różnej długości**
- **Operacje na danych w zapisie zmiennoprzecinkowym**

Proste działania arytmetyczne

Działania na liczbach (przykład Intel 8086)

```
X := Y + Z; {unsigned}
            mov     ax, y
            add     ax, z
            mov     x, ax
            jc      uOverflow
X := Y - Z; {unsigned}
            mov     ax, y
            sub     ax, z
            mov     x, ax
            jc      uOverflow
```

```
X := Y * Z; {unsigned}
            mov     ax, y
            mul     z
            mov     x, ax
            jc      uOverflow
X := Y * Z; {signed}
            mov     ax, y
            imul    z
            mov     x, ax
            jo      sOverflow
```

W wyniku wykonania działania może nastąpić przekroczenie zakresu → warunkowe rozgałęzienie programu z testem flagi:

- przeniesienia CF (**JC/JNC**)
- przepełnienia stałoprzecinkowego OF (**JO/JNO**)

Proste działania arytmetyczne - AVR

Działania na liczbach (przykład AVR)

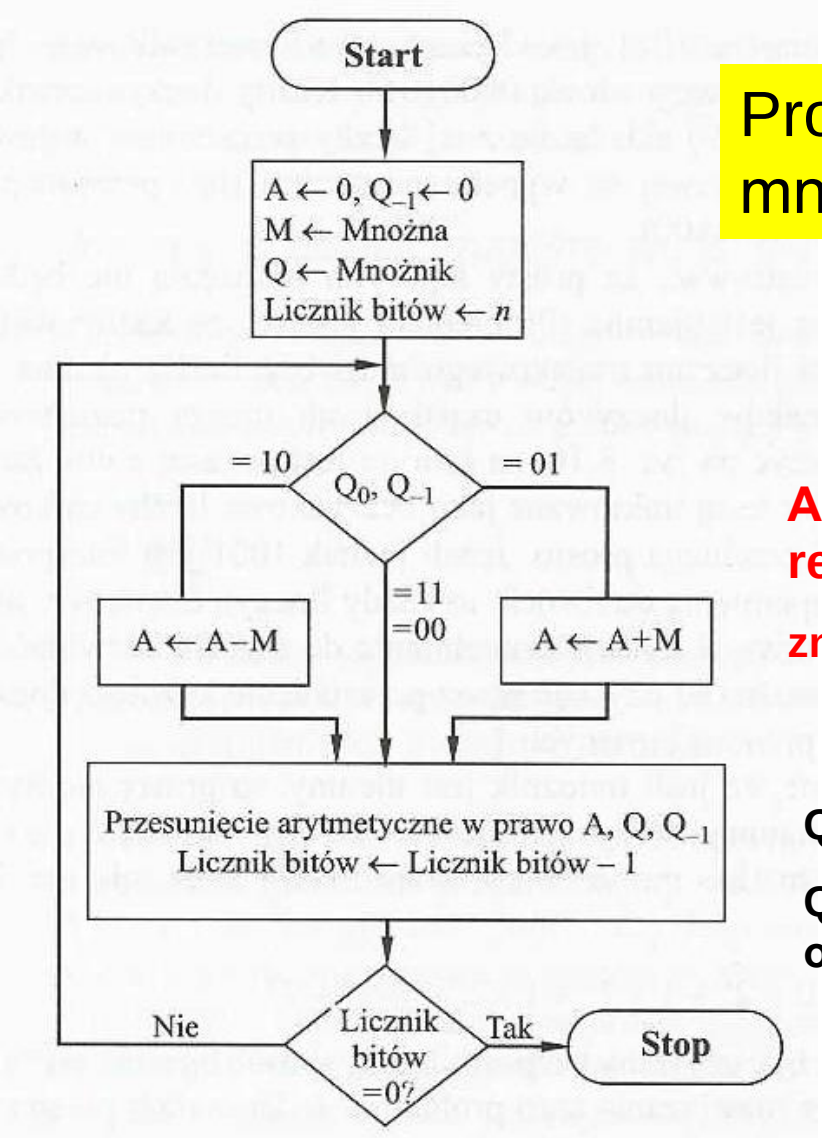
```
X:=X+Y              {unsigned}
        lds r1,x
        lds r2,y
        add r1,r2
        brcs uOverflow
X:=X+Y              {signed}
        lds r1,x
        lds r2,y
        add r1,r2
        brvs uOverflow
```

```
X:=X*Y              {unsigned}
        lds r1,x
        lds r2,y
        mul r1,r2
        brcs uOverflow
X:=X*Y              {signed}
        lds r1,x
        lds r2,y
        muls r1,r2
        brcs uOverflow
```

W wyniku wykonania działania może nastąpić przekroczenie zakresu → warunkowe rozgałęzienie programu z testem flagi:

- przeniesienia C (**BRCS/BRCC**)
- przepełnienia stałoprzecinkowego V (**BRVS/BRVC**)

Proste działania arytmetyczne



Procesor bez rozkazu mnożenia sprzętowego

Algorytm mnożenia liczb w reprezentacji:

znak-uzupełnienie do 2

Q_{n-1}, Q_{n-2}, ..., Q_0 – ciąg wynikowy

Q_{-1} – dodatkowy bit dla wykonywania obliczeń

Mnożenie bez użycia rozkazu mnożenia

Złożenie przesunięć arytmetycznych i sumowania/odejmowania (liczby bez znaku; przykład Intel 8086):

10*ax = 8*ax + 2*ax

```
shl     ax, 1       ;Multiply AX by two
mov     bx, ax      ;Save 2*AX for later
shl     ax, 1       ;Multiply AX by four
shl     ax, 1       ;Multiply AX by eight
add     ax, bx      ;Add in 2*AX to get 10*AX
```

Rozkaz mnożenia może być niedostępny, lub (Intel 8086) powolny w działaniu

Przesunięcie w lewo o 1 = pomnożenie x2 (wykonywane szybko)

Mnożenie – AVR, ARM

AVR, ARM - złożenie przesunięć arytmetycznych i sumowania/ odejmowania możliwe, lecz niecelowe

- AVR - wykonanie mnożenia zajmuje 2 takty
- CORTEX-A8 (ARM) - wykonanie mnożenia zajmuje 1 takt

Złożone działania arytmetyczne

Działania na liczbach (przykład Intel 8086)

```
W := X + Y * Z;

        mov     bx, x
        mov     ax, y
        mul     z
        add     bx, ax
        mov     w, bx
```

```
W := X + Y * Z;

        mov     ax, y
        mul     z
        add     ax, x
        mov     w, ax
```

W przypadku działań przemiennych możliwa jest pewna optymalizacja kodu

Złożone działania arytmetyczne

Działania na liczbach (przykład Intel 8086)

```
W := X/Y * Z
```

Jeżeli X*Z nie powoduje przepełnienia, dokładniejszy wynik uzyskamy implementując: **W:=X*Z/Y**

```
W := X/Y * Z

        mov     ax, x
        imul    z
        idiv    y
        mov     w, ax
```

Złożone działania arytmetyczne

Działania na liczbach (przykład Intel 8086)

```
W := X - Y * Z;

        mov     bx, x
        mov     ax, y
        imul    z
        sub     bx, ax
        mov     w, bx
```

zmienna tymczasowa

W złożonych operacjach arytmetycznych zwykle niezbędne jest użycie tymczasowych zmiennych pomocniczych

Złożone działania arytmetyczne

Działania na liczbach

```
W := (A + B) * (Y + Z);
```

Niezbędne jest wykonanie z użyciem dwóch zmiennych tymczasowych:

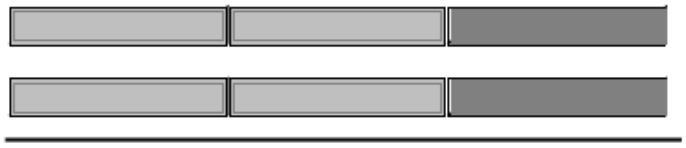
```
Temp1 := A + B;
Temp2 := Y + Z;
W := Temp1 * Temp2;
```

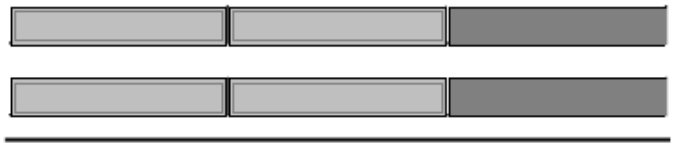
Liczba zmiennych rośnie ze stopniem złożoności wyrażenia (liczby uwzględnień wymaganej kolejności działań)

Działania w podwyższonej precyzji

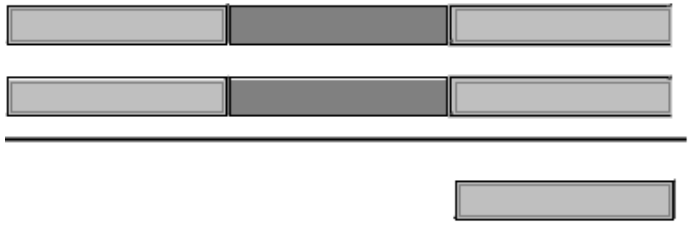
Przykład: dodawanie liczb zapisanych w N słowach

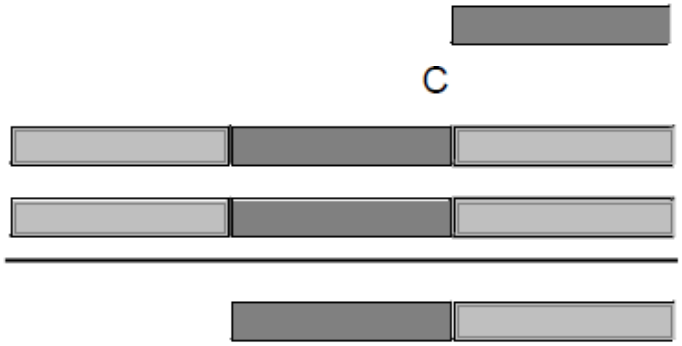
1. dodaj najmniej znaczące słowa



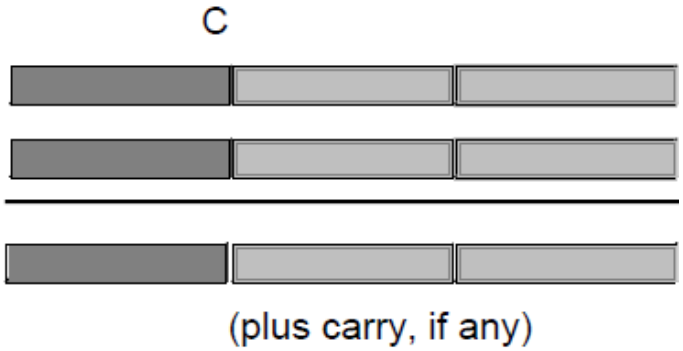


2. dodaj „środkowe” słowa





2. dodaj najbardziej znaczące słowa



Działania w podwyższonej precyzji

Działania na liczbach (przykład Intel 8086/16 bit)

```
X       dword   ?
Y       dword   ?
```

32 bity

Kod wykonujący sumowanie liczb, każda w formacie 2 słów; sumowanie instrukcją dodawania 16 bitowego

```
mov     ax, word ptr X
add     ax, word ptr Y
mov     word ptr Z, ax
mov     ax, word ptr X+2
adc     ax, word ptr Y+2
mov     word ptr Z+2, ax
```

Z=X+Y

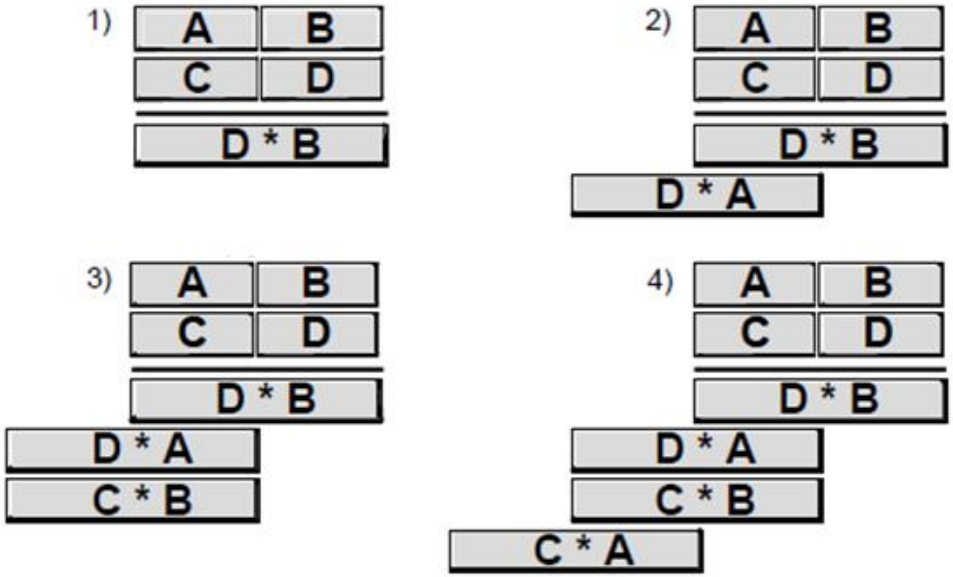
Działania w podwyższonej precyzji - ARM

Kod wykonujący sumowanie liczb, każda w formacie 2 słów 32bitowych;
sumowanie instrukcją dodawania 32 bitowego

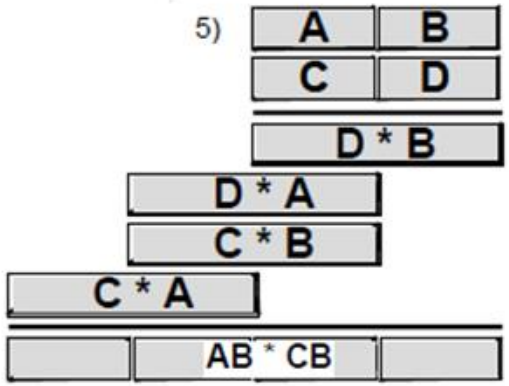
```
ldr     r0,x
ldr     r1,y
ldr     r2,z
ldm     r0,{r3-r4}
ldm     r1,{r5-r6}
adds    r3,r3,r5
addc    r4,r4,r6
stm     r2,{r3-r4}
```

Z=X+Y

Mnożenie w podwyższonej precyzji



Mnożenie w podwyższonej precyzji



Pomnóż składowe liczby, zastosuj odpowiednie skalowanie, dodaj sumy częściowe

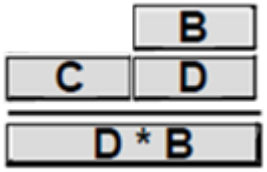
Mnożenie w podwyższonej precyzji

Działania na liczbach (przykład Intel 8086/16 bit)

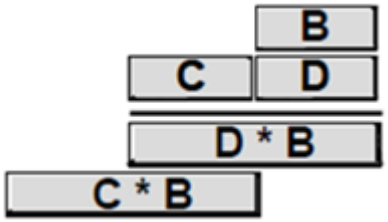
AX *Word $\longrightarrow$ DX :AX

Mnożenie w podwyższonej precyzji

DX :AX

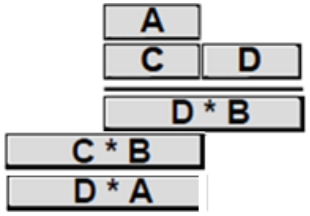



```
; Multiply the L.O. word of Multiplier times Multiplicand:
        mov     ax, word ptr Multiplier
        mov     bx, ax                      ;Save Multiplier val
        mul     word ptr Multiplicand       ;Multiply L.O. words
        mov     word ptr Product, ax        ;Save partial product
        mov     cx, dx                      ;Save H.O. word
```

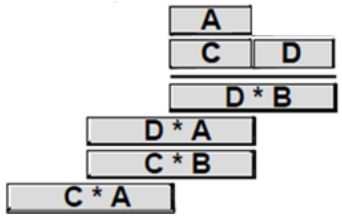



```
        mov     ax, bx                      ;Get Multiplier in BX
        mul     word ptr Multiplicand+2     ;Multiply L.O. * H.O.
        add     ax, cx                      ;Add partial product
        adc     dx, 0                       ;Don't forget carry!
        mov     bx, ax                      ;Save partial product
        mov     cx, dx                      ; for now.
```

```
; Multiply the H.O. word of Multiplier times Multiplicand:
```

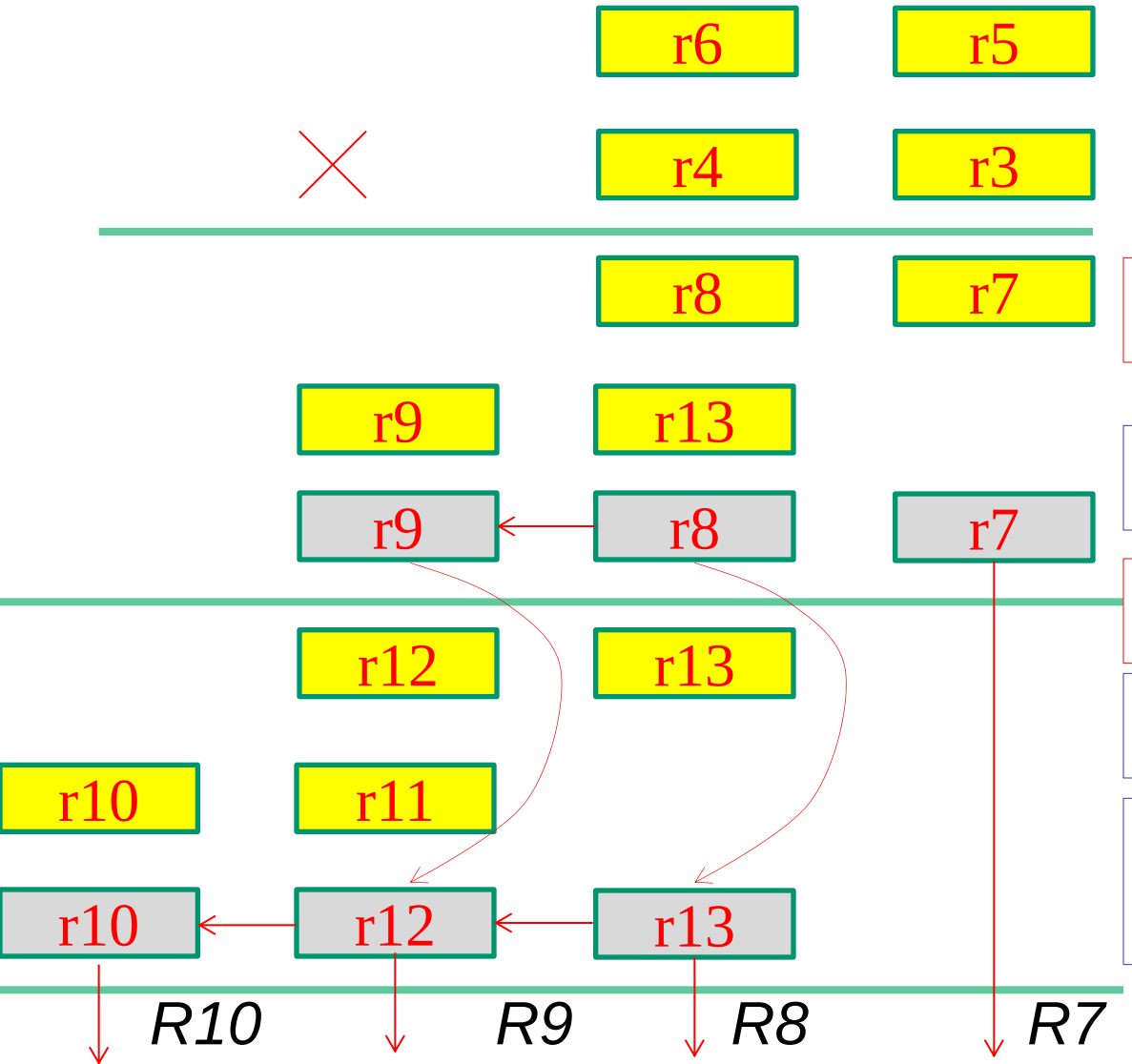



```
        mov     ax, word ptr Multiplier+2   ;Get H.O. Multiplier
        mul     word ptr Multiplicand       ;Times L.O. word
        add     ax, bx                      ;Add partial product
        mov     word ptr product+2, ax      ;Save partial product
        adc     cx, dx                      ;Add in carry/H.O.!
```




```
        mov     ax, word ptr Multiplier+2   ;Multiply the H.O.
        mul     word ptr Multiplicand+2     ; words together.
        add     ax, cx                      ;Add partial product
        adc     dx, 0                       ;Don't forget carry!
        mov     word ptr Product+4, ax      ;Save partial product
        mov     word ptr Product+6, dx
```

Mnożenie w podwyższonej precyzji – ARM7




```
ldr     r0,x
ldr     r1,y        Z=X*Y
ldr     r2,z
ldm     r0,{r3-r4}
ldm     r1,{r5-r6}
umull   r8,r7,r3,r5
umull   r9,r13,r3,r6
eor     r14,r14
adds    r8,r8,r13
addc    r9,r9,r14
umull   r12,r13,r4,r5
umull   r10,r11,r4,r6
adds    r12,r12,r11
addc    r10,r10,r14
adds    r8,r8,r13
addcs   r9,r9,r12
addc    r10,r10,r14
stm     r2,{r7-r10}
```

Porównania w podwyższonej precyzji

Przykład Intel 80386

```
QWordValue    qword     ?
QWordValue2   qword     ?

        mov     eax, dword ptr QWordValue+4
        cmp     eax, dword ptr QWordValue2+4
        jg      IsGreater
        jl      IsLess
        mov     eax, dword ptr QWordValue
        cmp     eax, dword ptr QWordValue2
        jg      IsGreater
        jl      IsLess
        ..........
IsLess:
        ..........
IsGreater:
        ..........
```

64 bity

Porównaj starsze słowa;
młodsze, jeżeli starsze równe

Porównania w podwyższonej precyzji - ARM

```
        ldr     r0,x
        ldr     r1,y
        ldm     r0,{r2-r3}
        ldm     r1,{r4-r5}
        cmp     r5,r3
        cmpeq   r4,r3
        bgt     Gt
        ble     Le
Gt:     ……
        ……
Le:     ……
```

Dzielenie w podwyższonej precyzji

Implementacja dzielenia n-bit/m-bit (n,m dowolne) mało efektywna

Przykład: dzielenie liczby 64-bit przez 16-bit

DX:AX / Memory

⟶ AX : DX

Wynik: 64-bit
Reszta: 16-bit

```
dividend        qword
divisor         word
Quotient        qword
Modulo          word

mov     ax, word ptr dividend+6
sub     dx, dx

div     divisor
mov     word ptr Quotient+6, ax
mov     ax, word ptr dividend+4
div     divisor
mov     word ptr Quotient+4, ax
mov     ax, word ptr dividend+2
div     divisor
mov     word ptr Quotient+2, ax
mov     ax, word ptr dividend
div     divisor
mov     word ptr Quotient, ax
mov     Modulo, dx
```

Konwersja na U2 w podwyższonej precyzji

Działania na liczbach (przykład Intel 8086)

```
neg     dx
neg     ax
sbb     dx,0
```

konwersja na U2 liczby 64-bitowej (dx:ax)

NEG wyznacza U2 pojedynczego słowa;

SBB dekrementuje dx, jeżeli pojawiła się „pożyczka” przy konwersji ax

Konwersja na U2 w podwyższonej precyzji

```
Value dword     0,0,0,0,0
        .       ;160 bit integer
        .
        .
```

```
mov     eax, 0
sub     eax, Value
mov     Value, eax
```

```
mov     eax, 0
sbb     eax, Value+4
mov     Value+8, ax
```

```
mov     eax, 0
sbb     eax, Value+8
mov     Value+8, ax
```

```
mov     eax, 0
sbb     eax, Value+12
mov     Value+12, ax
```

```
mov     eax, 0
sbb     eax, Value+16
mov     Value+16, ax
```

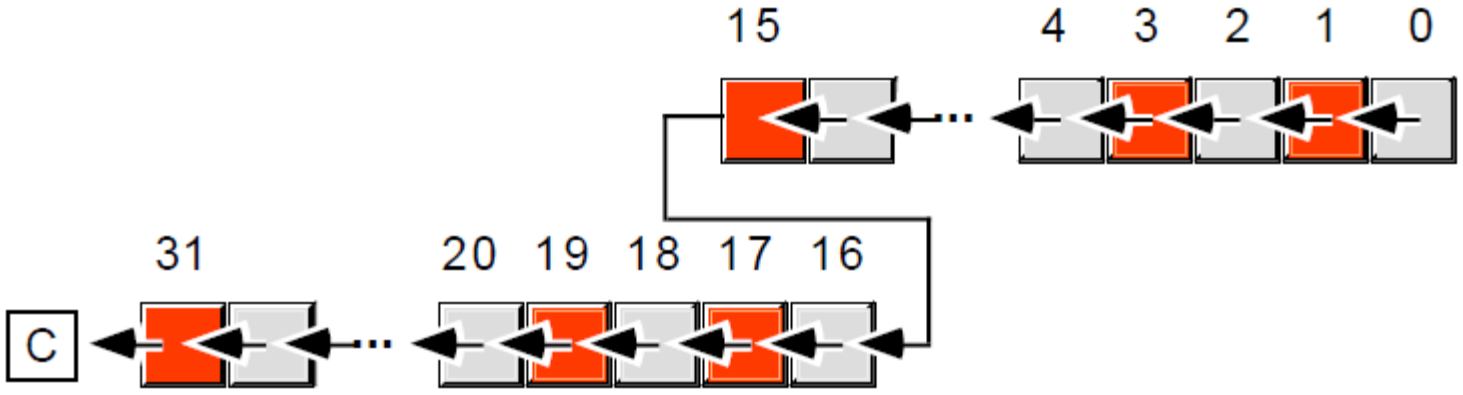
Działania na liczbach (przykład Intel 8086)

konwersja na U2 liczby 160-bitowej

Przy znacznej długości liczby korzystniej odjąć wartość od 0

Przesunięcia logiczne w podwyższonej precyzji

Przesuwanie w lewo liczby 32-bitowej w procesorze o 16-bitowej architekturze



Bit 15 wprowadź na bit 16 = wprowadź na bit 0 słowa zawierającego starszą część ciągu logicznego

Przesunięcia logiczne w podwyższonej precyzji

Działania logiczne (przykłady Intel 8086)

```
shl     ax, 1
rcl     dx, 1
```

Przesuwanie w lewo liczby 32-bitowej

```
shl     word ptr Operand, 1
rcl     word ptr Operand+2, 1
rcl     word ptr Operand+4, 1
```

Przesuwanie w lewo liczby 48-bitowej

Ograniczenie: przesuwanie jednorazowo tylko o 1 pozycję

Przesunięcia logiczne w podwyższonej precyzji

Działania logiczne (przykłady Intel 8086)

Przesuwanie w lewo liczby 48-bitowej o liczbę pozycji zapisaną w CX

```
ShiftLoop:  shl     word ptr Operand, 1
            rcl     word ptr Operand+2, 1
            rcl     word ptr Operand+4, 1
            loop    ShiftLoop
```

Sekwencja powtarzana CX razy

Operacje na danych o różnej długości

Działania na liczbach (przykład Intel 8086)

Dodawanie liczby 8-bitowej do liczby 16-bitowej

```
var1            byte    ?
var2            word    ?
```

```
Unsigned addition:

mov     al, var1
mov     ah, 0
add     ax, var2
```

```
Signed addition:

mov         al, var1
cbw
add         ax, var2
```

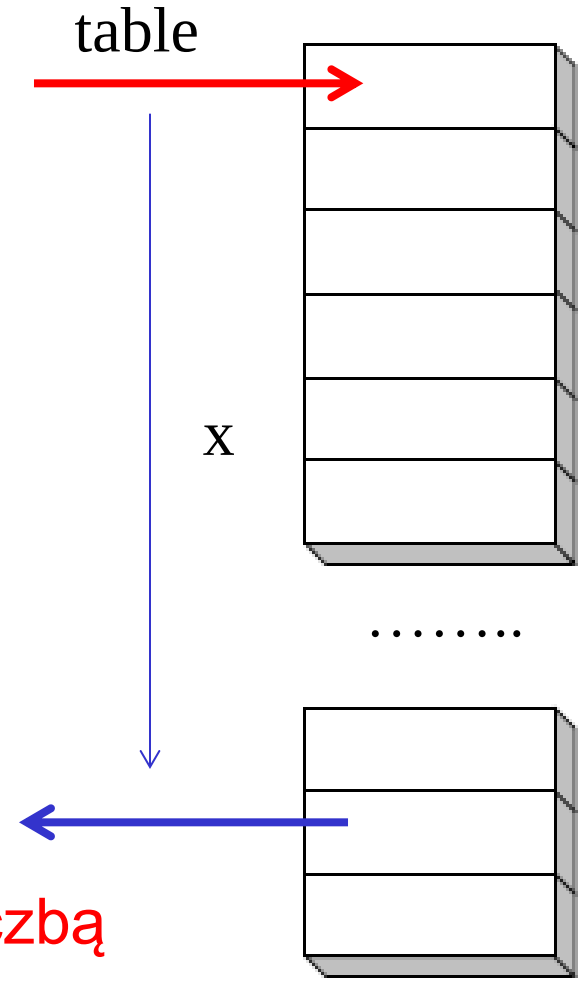
Rozszerz liczbę o krótszej reprezentacji o 0 (liczba bez znaku), lub wartość znaku (liczba ze znakiem)

Implementacja z użyciem tablicy

Alternatywą dla operacji jest odczytywanie wyniku operacji z tablicy

```
mov     al, x
lea     bx, table
xlat
mov     y, al
```

y=f(x)

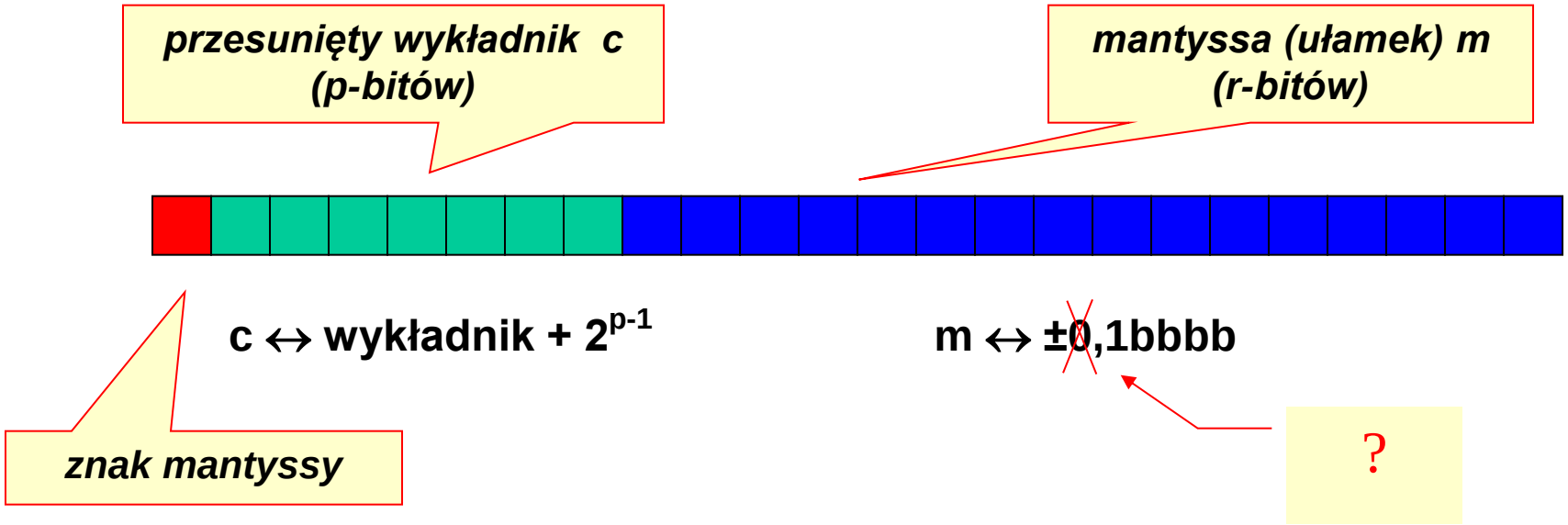


Ograniczenie: argument „musi” być liczbą całkowitą

Operacje na danych zmiennoprzecinkowych

REPREZENTACJE ZMIENNOPRZECINKOWE

Zapis liczby w postaci zmiennoprzecinkowej umożliwia przedstawienie większego zakresu liczb aniżeli wynika to z liczby bitów użytych do zapisania liczby.



Liczba = m · 2 wykładnik

Operacje na danych zmiennoprzecinkowych

$$a \cdot 2^b + c * 2^d = a \cdot 2^b + c * 2^{d-b} * 2^b =$$

$$= (a + c * 2^{d-b}) * 2^b$$

Operacje na danych zmiennoprzecinkowych

DODAWANIE LICZB W ZAPISIE ZMIENNOPRZECINKOWYM

- **Aby dodać dwie liczby „zmiennoprzecinkowe” przeprowadzić normalizację.**
- **Normalizacja polega na takiej zmianie zapisu mantyssy jednej z liczb, aby jej wykładnik równy był wykładnikowi drugiej liczby.**
- **Liczbę o *mniejszym wykładniku* dopasowuje się do liczby o *większym wykł.***
- **Dodawanie liczb wykonuje się poprzez dodanie *znormalizowanych* mantyss obu liczb.**

przesunięcie o 127

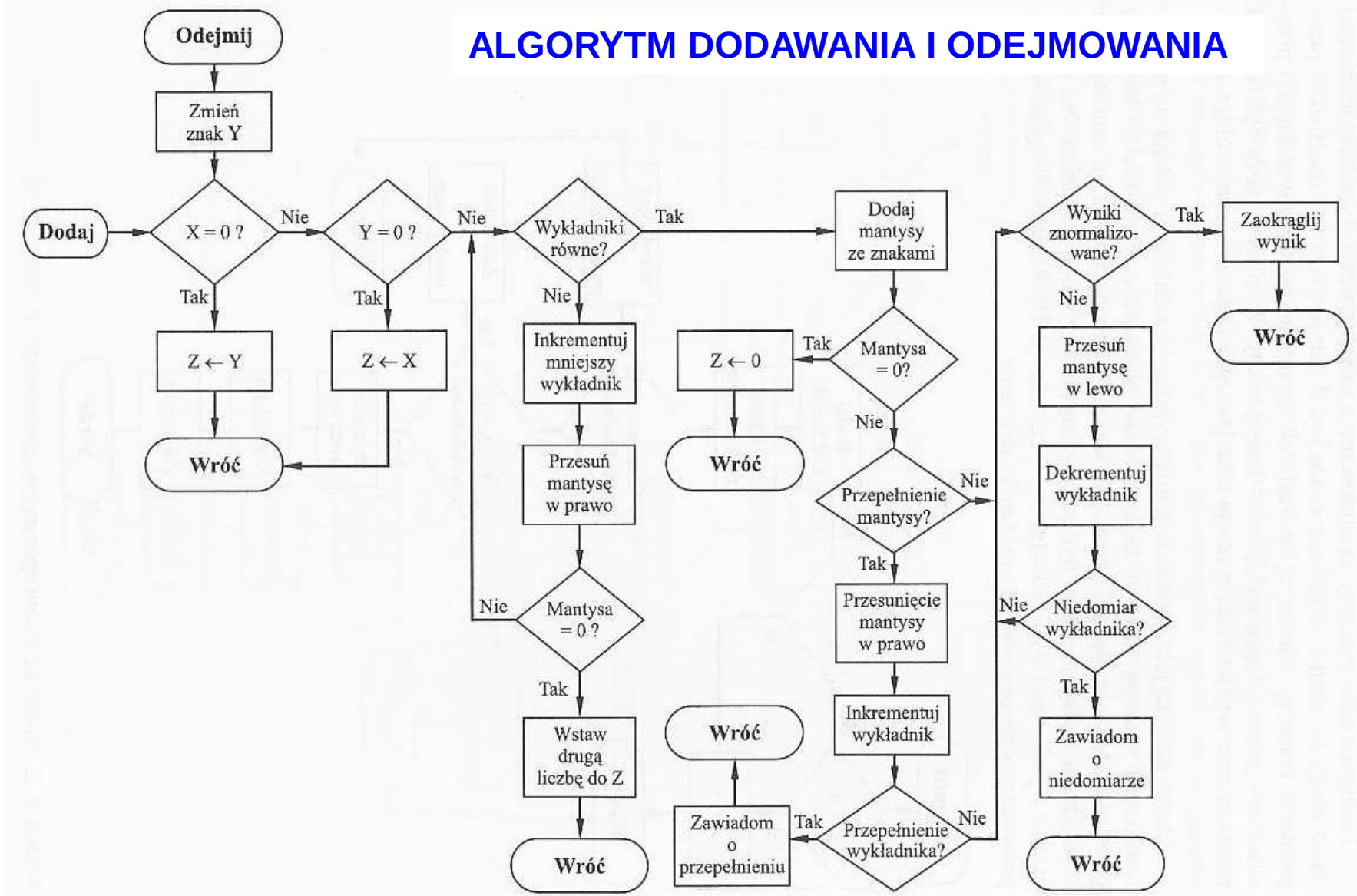
Przykład

A	**=**	**$0{,}785156 \cdot 2^6$**	**=**	**50,25**
B	**=**	**$0{,}855469 \cdot 2^2$**	**=**	**3.421875**
A+B	**=**	**$0{,}838623 \cdot 2^6$**	**=**	**53,67188**

$=0{,}053467 \cdot 2^6$

0 . 1 1 0 1 1 0 1 1 @ 1 0 0 0 0 0 0 1	**B**	**3.421875**
0 . 1 1 0 0 1 0 0 1 @ 1 0 0 0 0 1 0 1	**A**	**50,25**
0 . 0 0 0 0 1 1 0 1 @ 1 0 0 0 0 1 0 1	**B n.**	**3.25**
0 . 1 1 0 0 1 0 0 1 @ 1 0 0 0 0 1 0 1	**A**	**50,25**
0 . 1 1 0 1 0 1 1 0 @ 1 0 0 0 0 1 0 1	**A+B**	**53.5**

Operacje na danych zmiennoprzecinkowych



Operacje na danych zmiennoprzecinkowych

MNOŻENIE LICZB W ZAPISIE ZMIENNOPRZECINKOWYM

- **Mnożenie liczb zmiennoprzecinkowych wykonuje się przez mnożenie *mantyss* i dodawanie *wykładników* obydwu liczb.**

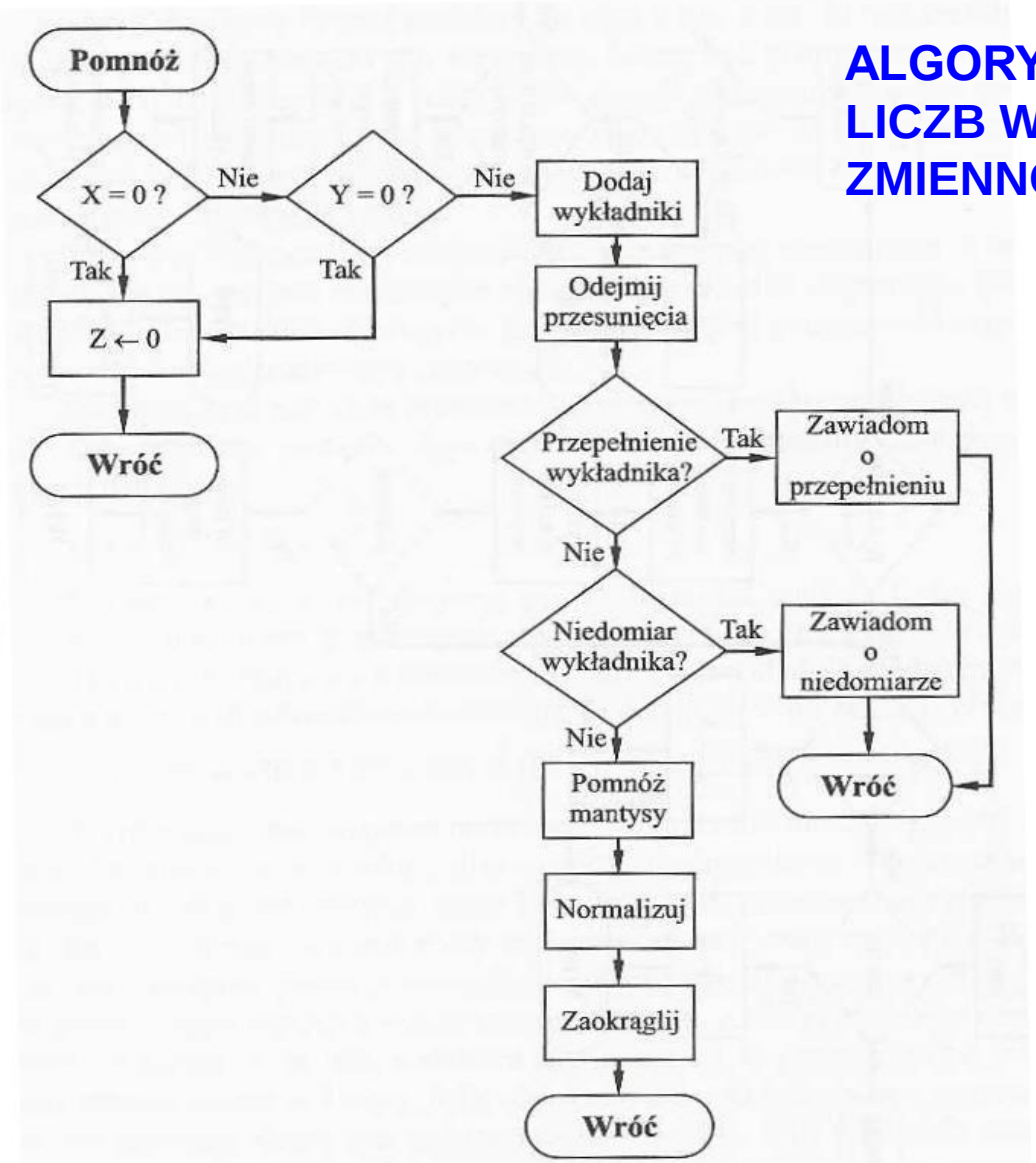
Przykład

A	**=**	**$0{,}7578125 \cdot 2^{6}$**	**=**	**48,5**
B	**=**	**$0{,}5 \cdot 2^{-2}$**	**=**	**0,125**
A·B	**=**	**$0{,}37890625 \cdot 2^{4}$**	**=**	**6,025**

```
                  . 1 1 0 0 0 0 1 0 @ 1 0 0 0 0 1 0 1 | A
                  . 1 0 0 0 0 0 0 0 @ 0 1 1 1 1 1 0 1 | B
-------------------------------------------------------+-----
. 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 @ 0 0 0 0 0 0 0 0 1 | A·B
        o 1 poz
. 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 @ 0 0 0 0 0 0 1 0     -1
                  . 1 1 0 0 0 0 1 0 @ 1 0 0 0 0 0 1 1 | A·B norm
                                                          -127
```

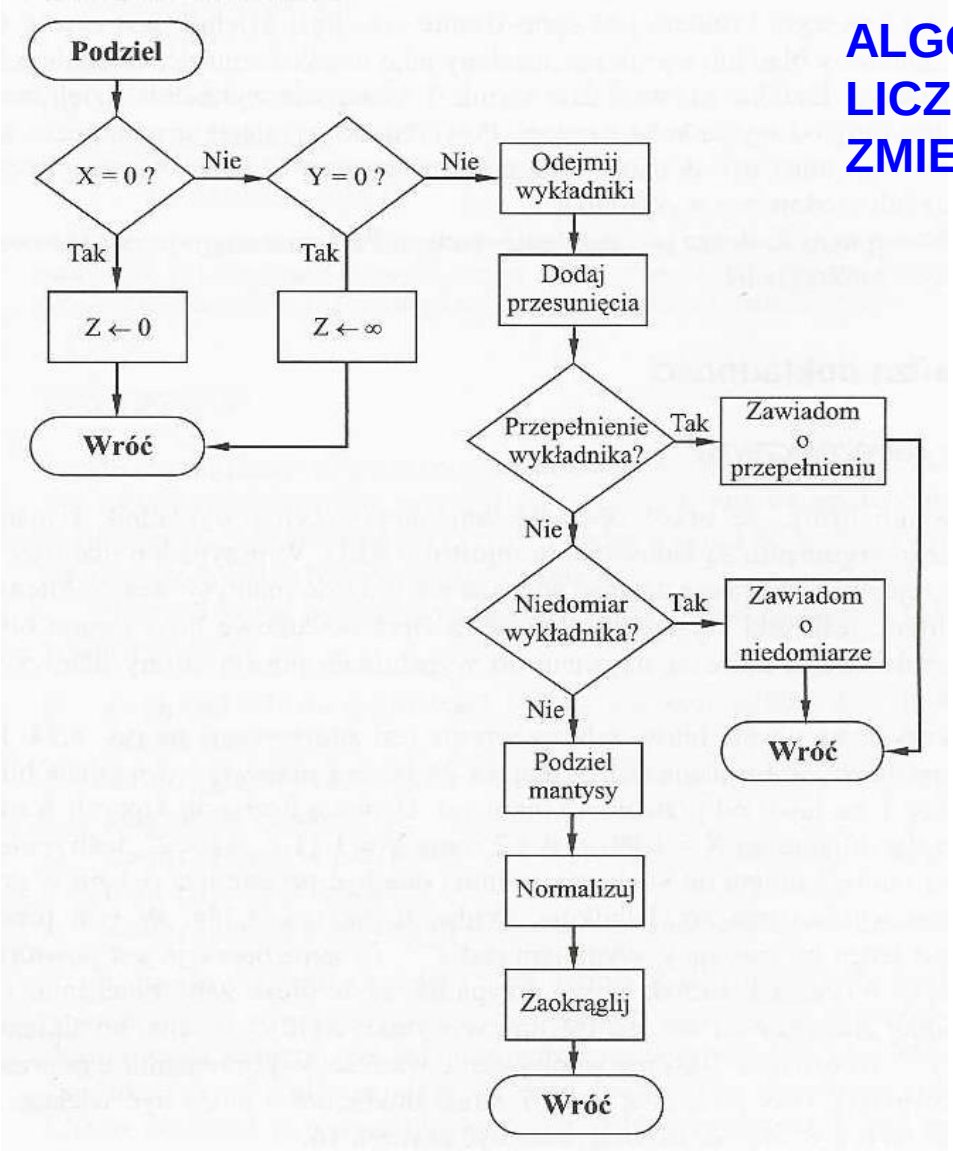
- **Po wykonaniu mnożenia – normalizacja (przesuwanie mantyssy w lewo tak, aby usunąć nieznaczące zera; obcięcie najmłodszych bitów wyniku). W rezultacie normalizacji może nastąpić przepełnienie (ujemne) wykładnika.**

Operacje na danych zmiennoprzecinkowych



ALGORYTM MNOŻENIA LICZB W REPREZENTACJI ZMIENNOPRZECINKOWEJ

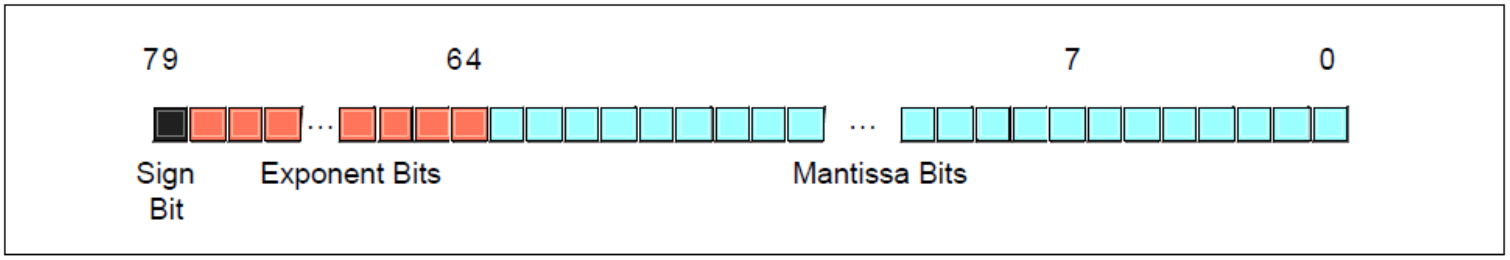
Operacje na danych zmiennoprzecinkowych

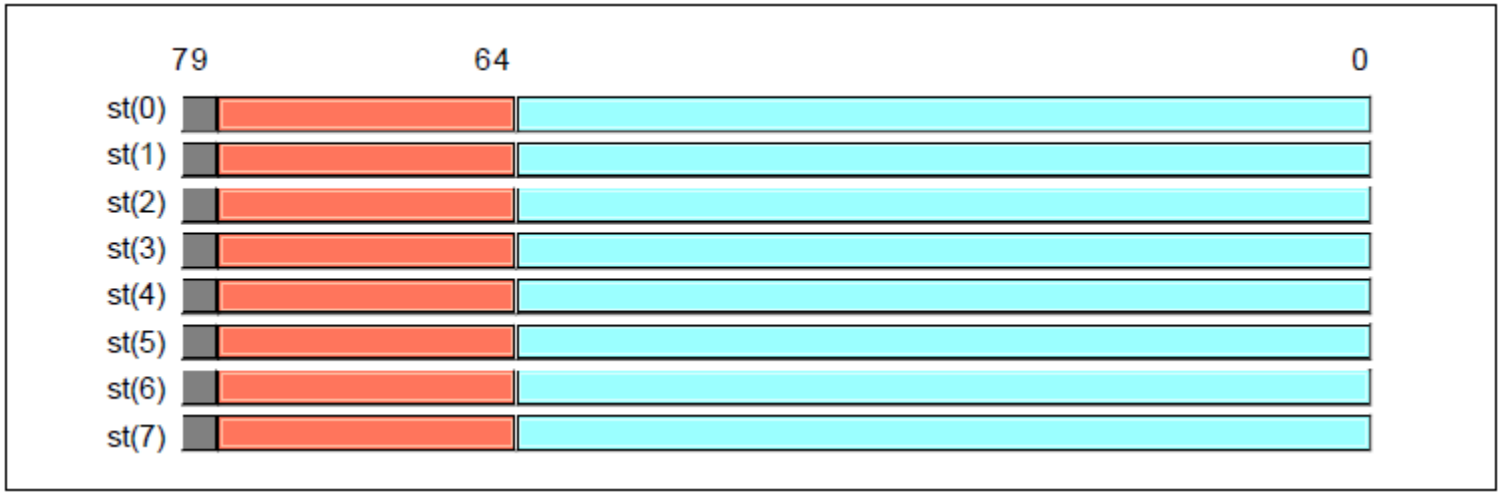


ALGORYTM DZIELENIA LICZB W REPREZENTACJI ZMIENNOPRZECINKOWEJ

Operacje na danych zmiennoprzecinkowych

Działania na liczbach (przykład Intel 8087)

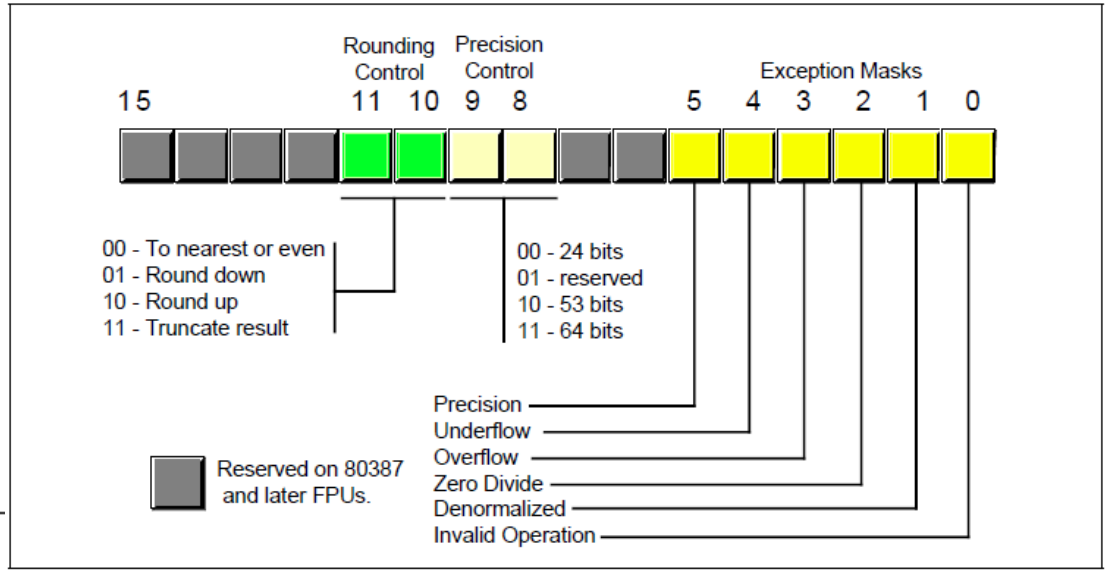


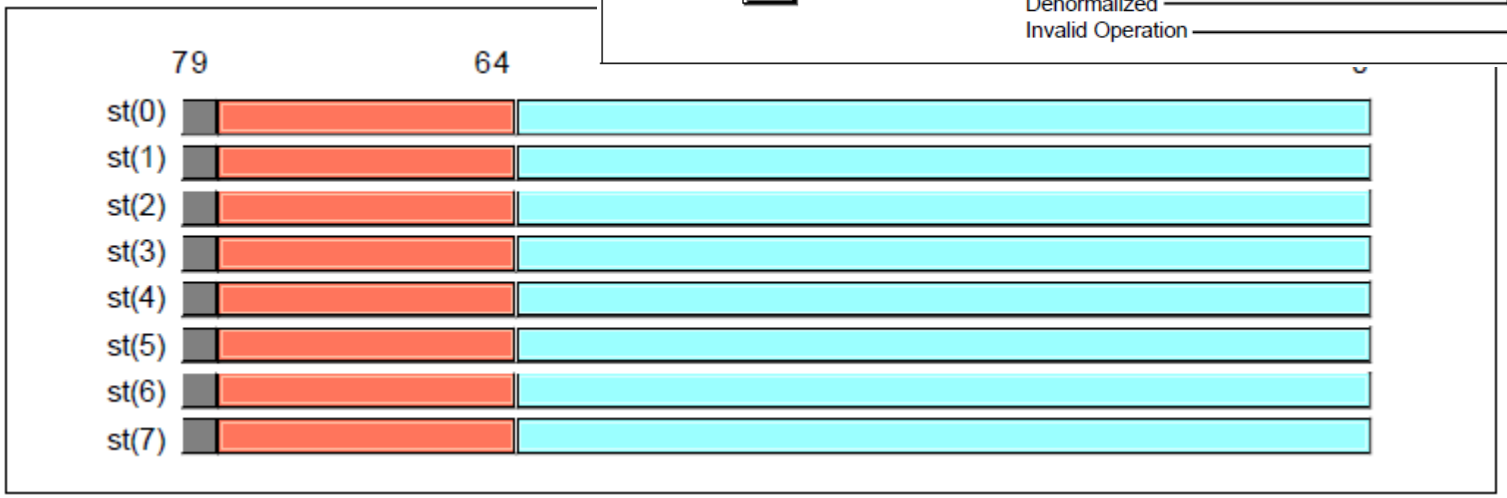


Architektura rejestrów 80x87 FPU

Operacje na danych zmiennoprzecinkowych

Działania na liczbach
(przykład Intel 8087)





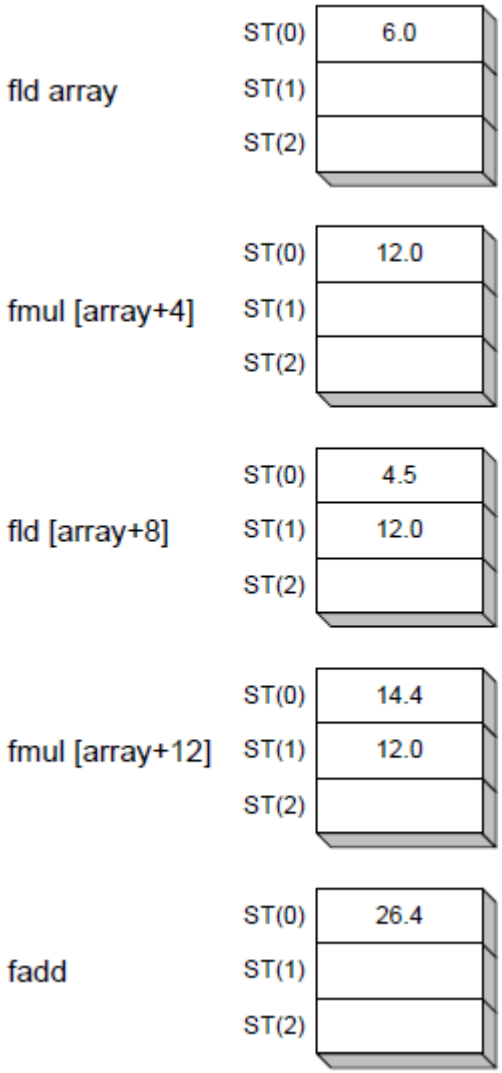
Architektura rejestrów 80x87 FPU

Operacje na danych zmiennoprzecinkowych

Działania na liczbach (przykład Intel 8087)

Przykład: Y=(A[4]*A[3])+(A[2]*A[1])

```
array REAL4 6.0, 2.0, 4.5, 3.2
dotProduct REAL4 ?
fld  array        ; push 6.0 onto the stack
fmul [array+4]    ; ST(0) = 6.0 * 2.0
fld  [array+8]    ; push 4.5 onto the stack
fmul [array+12]   ; ST(0) = 4.5 * 3.2
fadd              ; ST(0) = ST(0) + ST(1)
fstp dotProduct   ; pop stack into memory operand
```



Wykorzystanie stosowej architektury FPU
i rozkazów zmiennoprzecinkowych

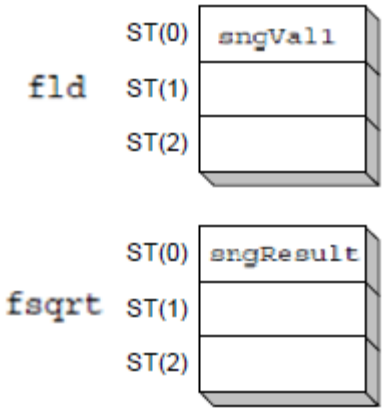
Operacje na danych zmiennoprzecinkowych

Działania na liczbach (przykład Intel 8087)

Przykład: Y=sqrt(sgnVal1)

```
sngVal1    REAL4 25.0
sngResult  REAL4 ?

fld sngVal1         ; load into FP stack
fsqrt               ; ST(0) = square root
fstp sngResult      ; store the result
```



Bezpośrednie użycie rozkazu zmiennoprzecinkowego fsqrt