```
;ecx - colorsBeforeFilter

;edx - colorsAfterFilter

;r8d - width

;r9d - startHeight

;r10d - endHeight



public MyProc1                                                  ;Nazwa procedury assemblerowej

.data                                                                   ;Deklaracja zmiennych w programie

;TE DANE MOGA BYC GLOBALNE:

maskArray byte 1, 2, 1,                                 ;}

                        2, 4, 2,                                        ;} ==> Tablica reprezentująca wartości maski dla rozmucia Gaussa

                        1, 2, 1                                 ;}

arrayRowSize qword 3                                    ;Przypisanie do zmiennej arrayRowSize liczby wierszy maski

minusOne dword -1                                               ;Wartość pomocnicza do rejestrów xmm

zero dword 0                                                    ;Wartość pomocnicza do rejestrów xmm

one dword 1                                                             ;Wartość pomocnicza do rejestrów xmm

two dword 2                                                             ;Wartość pomocnicza do rejestrów xmm

three dword 3                                                   ;Wartość pomocnicza do rejestrów xmm

colorsAfterFilterHolder qword 0                 ;Bufor trzymający wartość rejestru RDX aby pod koniec programu móc sciągnąć ze stosu pierwotną wartość


.code                                                                   ;Rozpoczęcie kodu assemblera

MyProc1 proc                                                    ;Nazwa procedury assemblerowej
```

```
mov r10d,DWORD PTR[rsp+40]                    ;Przypisanie do rejestru r10d wartości piątego argumentu przekazanego do funkcji assemblerowej czyli endHeight

push rbx                                      ;Odłożenie na stos wartości spod rejestru rbx

push rsi                                      ;Odłożenie na stos wartości spod rejestru rsi

push rdi                                      ;Odłożenie na stos wartości spod rejestru rdi

push rbp                                      ;Odłożenie na stos wartości spod rejestru rbp

push r12                                      ;Odłożenie na stos wartości spod rejestru r12

push r13                                      ;Odłożenie na stos wartości spod rejestru r13

push r14                                      ;Odłożenie na stos wartości spod rejestru r14

push r15                                      ;Odłożenie na stos wartości spod rejestru r15

mov colorsAfterFilterHolder, rdx              ;Przypisanie wartości rejestru rdx do zmiennej colorsAfterFilterHolder aby potem wrócić do stanu rejestrów spprzed wywołania funcji


loopOverRows:                                 ;for (int row = startHeight; row < endHeight; row++)
                                              poczatek petli po wierszach

cmp r9d, r10d                                 ;Sprawdz czy row < endHeight

je filteringDone                              ;Jeżeli warunek pętli loopOverRows nie jest spełniony wyjdź z niej


        movd xmm0, zero                       ;Początkowo inicjalizuje rejestr xmm0 wartością zero, następnie xmm0 = col

        movd xmm1, r8d                        ;Inicjalizuje rejestr xmm1 wartością r8d, więc xmm1 = width

        loopOverColumns:                      ;for (int col = 0; col < width; col++)
        poczatek wierszy po kolumnach

        comisd xmm0, xmm1                     ;Sprawdź czy col < width
```

```
        je loopOverColumnsDone                          ;Jeżeli warunek pętli loopOverColumns nie jest spełniony wyjdź z niej

            mov r11, 0                                  ;
            pinsrd xmm5, r11d, 0                        ;Wpisuję do rejestru xmm5[0](redSum) wartość zero
            mov r11, 0                                  ;
            pinsrd xmm5, r11d, 1                        ;Wpisuję do rejestru xmm5[1](greenSum) wartość zero
            mov r11, 0                                  ;
            pinsrd xmm5, r11d, 2                        ;Wpisuję do rejestru xmm5[2](blueSum) wartość zero
            pinsrd xmm6, zero, 0                        ;Wpisuję do rejestru xmm6(sumMask) wartość zero

            movd xmm7, minusOne                         ;Wpisuję do rejestru xmm7(maskRowIndex) wartość -1
            movd xmm10, minusOne                        ;Wpisuję do rejestru xmm10(maskRowIndex) wartość -1
            movd xmm9, minusOne                         ;Wartość pomocnicza dla ujemnych wartości rejestrów

                movd xmm8, two                          ;Inicjalizuje rejestr xmm8 wartością dwa
                loopOverMaskRows:                       ;for (int j = -1; j <= 1; j++)
                comisd xmm7, xmm8                       ;Sprawdź czy j < 2
                je loopOverMaskRowsDone                 ;Jeżeli warunek pętli loopOverMaskRows nie jest spełniony wyjdź z niej

                    movd xmm10, minusOne                ;Wyzerowanie rejestru zawierającego indeks petli wewnetrznej
                    loopOverMaskColumns:                ;for (int i = -1; i <= 1; i++)
                    comisd xmm10, xmm8                  ;Sprawdź czy j < 2
```

```
                    je loopOverMaskColumnsDone          ;Jeżeli warunek pętli
loopOverMaskColumns nie jest spełniony wyjdź z niej


                    ;Sprawdzenie poprawności ifa   ;if ((row + j) >= 0 && (row + j) < height
&& (col + i) >= 0 && (col + i) < width)

                    mov r11d, r9d                       ;przypisanie do
rejestru r11d aktualnego indeksu wiersza, gdyż row = r9d

                    movd xmm4, r11
          ;Wykorzystanie rejestru xmm4 jako bufor na wartość r11

                    addsd xmm4, xmm7                    ;operacja dodania do
rejestru xmm4 wartości xmm7 => row + j, gdyż xmm7 = j

                    movd r11, xmm4                          ;przypisanie
do rejestru r11 wartości xmm4, (row + j) -> r12

                    movd r12, xmm0                          ;przypisanie
do rejestru r12 aktualnego indeksu kolumny, gdyż col = r8d

                    movd xmm4, r12
          ;Wykorzystanie rejestru xmm4 jako bufor na wartość r12

                    addsd xmm4, xmm10                   ;operacja dodania do
rejestru xmm4 wartości xmm10 => col + i, gdyż xmm10 = i

                    movd r12, xmm4                          ;przypisanie
do rejestru r12 wartości xmm4, (col + i) -> r12

                    cmp r11d, 0                             ;Porównanie
wartości rejestru r11d z zerem, porównanie (row + j) z 0

                    jl isFalse                              ;jeżeli (row +
j) < 0 przejdź do etykiety isFalse

                    cmp r11d, r10d                      ;Porównanie wartości
rejestru r11d z r10d, porównanie (row + j) z endHight

                    jge isFalse                             ;jeżeli (row +
j) >= endHight przejdź do etykiety isFalse

                    cmp r12d, 0                             ;Porównanie
wartości rejestru r12d z zerem, porównanie (col + i) z 0

                    jl isFalse                              ;jeżeli (col + i)
< 0 przejdź do etykiety isFalse

                    cmp r12d, r8d                       ;Porównanie wartości
rejestru r12d z r8d, porównanie (col + i) z width

                    jge isFalse                             ;jeżeli (col + i)
>= width przejdź do etykiety isFalse
```

```
                ;Jeżeli if jest spełniony
                ;Lewa strona wyrazenia w tablicy colorsBeforeFilter
                movd xmm11, rax;                ;Robie bufor z xmm11
na trzymanie rax, żeby wrócić potem do poprzedniej wartości
                mov eax, r11d                   ;(row + j) -> eax
                mov r11d, 3                     ;Wpisuje 3 do
rejestru r11d
                mul r11d                        ;(row + j) * 3 -
> eax
                mul r8d                         ;(row + j) * 3 *
width -> eax
                mov r11d, eax                   ;Wpisuje wynik lewe
strona wyrazenia w tablicy colorsBeforeFilter do rejestru r11d
                movd rax, xmm11                 ;wracam rax
do poprzedniej wartości

                ;Prawa strona wyrazenia w tablicy colorsBeforeFilter
                movd xmm11, rax;                ;Robie bufor z xmm11
na trzymanie rax, żeby wrócić potem do poprzedniej wartości
                mov eax, r12d                   ;(column + i) -> eax
                mov r12d, 3                     ;Wpisuje 3 do
rejestru r12d
                mul r12d                        ;(column + i) *
3 -> eax
                mov r12d, eax                   ;Wpisuje wynik prawa
strona wyrazenia w tablicy colorsBeforeFilter do rejestru r12d
                movd rax, xmm11                 ;wracam eax
do poprzedniej wartości

                ;Składowa RED
                add r11d, r12d                  ;(row + j) * 3 * width
+ (column + i) * 3 -> r11d
                movd xmm4, r11
        ;Wykorzystanie rejestru xmm4 jako bufor na wartość r11
```

```
addss xmm4, zero ;(row + j) * 3 * width + (column + i) * 3 + 0 -> xmm4
movd r11, xmm4 ;przypisanie do rejestru r11 wartości xmm4, (row + j) * 3 * width + (column + i) * 3 + 0 -> r11
mov r13b, byte ptr[rcx+r11] ;Zmienna r13b to redColour, przypisuje do niej wynik z tablicy colorsBeforeFilter
pextrd r11d, xmm5, 0 ;r11d = redSum
mov r12, OFFSET maskArray ;przypisanie do rejestu r12 adresu początku tablicy maskArray, jest to tablica zawierająca wartości maski rozmywającej
movd r15, xmm7 ;Przypisanie wartości rejestru xmm7 do rejestru r15
add r15d, 1 ;Dodanie do rejestru r15d wartości 1
mov eax, r15d ;}
mul arrayRowSize ;}
add r12, rax ;} ===> Odwołanie do wartości tablicy dwuwymarowej, czyli (mask[i + 1][j + 1])
movd esi, xmm10 ;}
add esi, 1 ;}
mov al, [r12 + rsi] ;mask[i + 1][j + 1] -> al
mov r14, rax ;mask[i + 1][j + 1] -> r14 (Wykorzystuje r14 jako bufor)
mul r13b ;mask[i + 1][j + 1] * redColour -> al
add r11d, eax ;redSum += colour * mask[i + 1][j + 1], bo r11 = SUM !!!!!
pinsrd xmm5, r11d, 0 ;(to co wyżej) -> xmm5[0](redSum)

;Składowa GREEN
addss xmm4, one ;(row + j) * 3 * width + (column + i) * 3 + 1 -> xmm4
movd r11, xmm4 ;przypisanie do rejestru r11 wartości xmm4, (row + j) * 3 * width + (column + i) * 3 + 1 -> r11
mov r13b, byte ptr[rcx+r11] ;Zmienna r13b to greenColour, przypisuje do niej wynik z tablicy colorsBeforeFilter
```

```
                pextrd r11d, xmm5, 1            ;r11d = greenSum
                mov rax, r14                    ;mask[i + 1][j + 1] ->
rax
                mul r13b                        ;mask[i + 1][j
+ 1] * greenColour -> al
                add r11d, eax                   ;greenSum +=
greenColour * mask[i + 1][j + 1], bo r11 = SUM !!!!!
                pinsrd xmm5, r11d, 1            ;(to co wyżej) ->
xmm5[1](greenSum)

                ;Składowa BLUE
                addss xmm4, one                 ;(row + j) * 3 *
width + (column + i) * 3 + 2 -> xmm4
                movd r11, xmm4                  ;przypisanie
do rejestru r11 wartości xmm4, (row + j) * 3 * width + (column + i) * 3 + 2 -> r11
                mov r13b, byte ptr[rcx+r11]     ;Zmienna r13b to
greenColour, przypisuje do niej wynik z tablicy colorsBeforeFilter
                pextrd r11d, xmm5, 2            ;r11d = blueSum
                mov rax, r14                    ;mask[i + 1][j + 1] ->
rax
                mul r13b                        ;mask[i + 1][j
+ 1] * blueColour -> al
                add r11d, eax                   ;blueSum +=
blueColour * mask[i + 1][j + 1], bo r11 = SUM !!!!!
                pinsrd xmm5, r11d, 2            ;(to co wyżej) ->
xmm5[2](blueSum)


                pextrd r11d, xmm6, 0            ;przypisanie wartości rejestru
xmm6 do rejestru r11d, więc r11d = sumMask
                mov r12d, r11d                  ;przypisanie wartości
rejestru r11d do rejestru r12d, więc r12d = sumMask
                mov eax, r11d                   ;sumMask += -> eax
                add rax, r14                    ;sumMask += mask[i +
1][j + 1] -> rax
```

```
                pinsrd xmm6, eax, 0             ;xmm6[0] -> sumMask
                pinsrd xmm6, eax, 1             ;xmm6[1] -> sumMask
                pinsrd xmm6, eax, 2             ;xmm6[2] -> sumMask

                comisd xmm10, xmm9              ;Porównanie wartości rejestrów xmm10 oraz xmm9
                je xmm10RegisterIsNegative      ;Jeżeli xmm10 == xmm9 przeskocz do etykiety xmm10RegisterIsNegative
                addss xmm10, one                ;Dodaj wartość 1 do rejestru xmm10
                jmp loopOverMaskColumns         ;Skocz do etykiety loopOverMaskColumns

                ;Jeżeli if nie jest spełniony
                isFalse:                        ;Początek etykiety isFalse
                comisd xmm10, xmm9              ;Porównanie wartości rejestrów xmm10 oraz xmm9
                je xmm10RegisterIsNegative      ;Jeżeli xmm10 == xmm9 przeskocz do etykiety xmm10RegisterIsNegative
                addss xmm10, one                ;Dodaj wartość 1 do rejestru xmm10
                jmp loopOverMaskColumns         ;Skocz do etykiety loopOverMaskColumns

                xmm10RegisterIsNegative:        ;Początek etykiety xmm10RegisterIsNegative
                movd xmm10, zero                ;Wpisz do rejestru xmm10 wartość 0
                jmp loopOverMaskColumns         ;Skocz do etykiety loopOverMaskColumns

                loopOverMaskColumnsDone:        ;Początek etykiety loopOverMaskColumnsDone
```

```
comisd xmm7, xmm9 ;Porównanie wartości rejestrów xmm7 oraz xmm9
je xmm7RegisterIsNegative ;Jeżeli xmm7 == xmm9 przeskocz do etykiety xmm7RegisterIsNegative
addss xmm7, one ;Dodaj wartość 1 do rejestru xmm7
jmp loopOverMaskRows ;Skocz do etykiety loopOverMaskRows

xmm7RegisterIsNegative: ;Początek etykiety xmm7RegisterIsNegative
movd xmm7, zero ;Wpisz do rejestru xmm7 wartość 0
jmp loopOverMaskRows ;Skocz do etykiety loopOverMaskRows

loopOverMaskRowsDone: ;Początek etykiety loopOverMaskRowsDone

divps xmm5, xmm6 ;Instrukcja wektorowa, dziele xmm5[0](redSum), xmm5[1](greenSum), xmm5[2](blueSum) przez sumMask
CVTPS2DQ xmm12, xmm5 ;Rzutuje wynik dzielenia z floatów na inty i wpisuje wynik do rejestru xmm12
mov eax, r9d ;startHeight -> eax
mul three ;3 * startHeight -> eax
mul r8d ;3 * startHeight * width -> eax
mov r11d, eax ;3 * startHeight * width -> r11d
movd eax, xmm0 ;col -> eax
mul three ;col * 3 -> eax
mov r12d, eax ;col * 3 -> r12d
add r11d, r12d ;3 * startHeight * width + col * 3 -> r11d
```

```
movd xmm4, r11                                  ;Przypisanie wartości rejestru r11 do xmm4
addss xmm4, zero                                ;3 * startHeight * width + col * 3 + 0-> r11d
movd r11, xmm4                                  ;Przypisanie wartości spod rejestru xmm4 do rejestru r11
pextrd r15d, xmm12, 0                           ;Przypisanie wartości spod rejestru xmm5 do rejestru r15
mov rdx, colorsAfterFilterHolder                ;Wracam wartosc rejestru rdx
mov byte ptr[rdx + r11], r15b;r15b              ;Wpisuj do tablicy colorsAfterFilter piksele po rozmyciu

addss xmm4, one                                 ;3 * startHeight * width + col * 3 + 1-> r11d
movd r11, xmm4                                  ;Przypisanie wartości spod rejestru xmm4 do rejestru r11
pextrd r15d, xmm12, 1                           ;Przypisanie wartości spod rejestru xmm12 do rejestru r15
mov rdx, colorsAfterFilterHolder                ;Wracam wartosc rejestru rdx
mov byte ptr[rdx + r11], r15b;r15b              ;Wpisuj do tablicy colorsAfterFilter piksele po rozmyciu

addss xmm4, one                                 ;3 * startHeight * width + col * 3 + 2-> r11d
movd r11, xmm4                                  ;Przypisanie wartości spod rejestru xmm4 do rejestru r11
pextrd r15d, xmm12, 2                           ;Przypisanie wartości spod rejestru xmm5 do rejestru r15
mov rdx, colorsAfterFilterHolder                ;Wracam wartosc rejestru rdx
mov byte ptr[rdx + r11], r15b;r15b              ;Wpisuj do tablicy colorsAfterFilter piksele po rozmyciu

addss xmm0, one                                 ;Dodaje wartość jeden do rejestru xmm0
jmp loopOverColumns                             ;Skocz do etykiety loopOverColumns, czyli wróć do pętli loopOverColumns
```

```
loopOverColumnsDone:                                    ;Początek etykiety
loopOverColumnsDone

inc r9d                                                 ;Zinkrementuj
wartość licznika pętli, czyli rejestru r9d(startHeight)

jmp loopOverRows                                        ;Skocz do etykiety,
czyli wróć do pętli loopOverRows


filteringDone:                                          ;Początek etykiety
filteringDone a zarazem koniec pętli loopOverRows

pop r15                                                 ;Wróć ze
stosu poprzednią wartość rejestru r15

pop r14                                                 ;Wróć ze
stosu poprzednią wartość rejestru r14

pop r13                                                 ;Wróć ze
stosu poprzednią wartość rejestru r13

pop r12                                                 ;Wróć ze
stosu poprzednią wartość rejestru r12

pop rbp                                                 ;Wróć ze
stosu poprzednią wartość rejestru rbp

pop rdi                                                 ;Wróć ze
stosu poprzednią wartość rejestru rdi

pop rsi                                                 ;Wróć ze
stosu poprzednią wartość rejestru rsi

pop rbx                                                 ;Wróć ze
stosu poprzednią wartość rejestru rbx

ret
        ;Powrót z procedury

MyProc1 endp                                            ;Koniec procedury
MyProc1

end
        ;Wyjście z DDL'ki assemblerowej, koniec assemblerowego maina
```