
		Instytut Informatyki Politechniki Śląskiej Zespół Mikroinformatyki i Teorii Automatów Cyfrowych			
Rok akademicki:	Rodzaj studiów*: SSI/NSI/NSM	Przedmiot (Języki Asemblerowe/SMiW):	Grupa	Sekcja	
<b>2019/2020</b>	<b>SSI</b>	Języki Asemblerowe	<b>1</b>	<b>2</b>	
Imię:	Michał	Prowadzący: OA/JP/KT/GD/BSz/GB	<b>JP</b>		
Nazwisko:	Jankowski				
<h2><i>Raport końcowy</i></h2>					
Temat projektu: <div style="text-align: center; margin-top: 100px;"> <h1>Efekt Sepii</h1> </div>					
Data oddania: dd/mm/rrrr			07/02/2020		

## **Główne założenia projektu:**

1. Przekształcenie zadanego obrazu na odcienie szarości;
2. Dodanie współczynnika  $W(I)$  dla zadanych składowych piksela do wcześniej przekształconego obrazu w odcienie szarości;

## **Założenia części głównej projektu w języku wysokiego poziomu:**

1. Wykonanie graficznego interfejsu użytkownika z wykorzystaniem Windows Forms w języku C#;

## **Założenia projektu dla funkcji biblioteki:**

1. Zostanie napisana w języku C#;
2. Będzie odpowiedzialna za przekształcanie obrazu w odcienie szarości;

Celem projektu jest stworzenie programu okienkowego w technologii Windows Forms, który mógłby przetwarzać obrazy na efekt Sepii, ale jednocześnie można uzyskać dzięki niemu odcienie szarości. Program będzie wspierał wielowątkowość (1 - 64). Program zostanie napisany w architekturze 64 bitowej. Będzie również mierzył czas wykonania biblioteki. GUI programu oraz powstaną dwie wersje algorytmu napisane w C# oraz w assemblerze.

## **Analiza zadania oraz uzasadnienie wyboru rozwiązania**

Program wymaga napisania dwóch bibliotek w Assemblerze oraz w C#. Obie biblioteki zostaną wykonane w architekturze 64 bitowej. Zgodnie z ogólnymi wymaganiami biblioteka assemblerowa będzie wykorzystywała instrukcje SIMD. W trakcie dogłębnej analizy doszedłem do wniosku, iż aby otrzymać efekt Sepii należy rozdzielić podany algorytm na dwie pętle. Pierwsza pętla będzie tworzyła odcienie szarości na danym obrazku, a następnie druga za pomocą kilku instrukcji warunkowych pozwoli otrzymać efekt wypełnienia sepia. W trakcie tworzenia programu rozszerzyłem go również o możliwość dodania głębi efektu oraz jego intensywności co wpływa na większą możliwość wyboru żadanego efektu. Ze względu na niekomplikowanie zadania wybrałem, iż będę przetwarzał tylko pliki typu \*.bmp, ponieważ okazały się one dla mnie najprostsze do przetworzenia. Podczas analizy zdecydowano również o tym iż program przetwarza 8 pikseli w jednej iteracji pętli, dlatego minimalnym rozmiarem pliku do przetworzenia jest 64 x 64 piksele.

W projekcie graficzny interfejs użytkownika został zaimplementowany w język C# z wykorzystaniem technologii Windows Forms, ponieważ posiadam doświadczenie w tworzeniu aplikacji na tę właśnie technologię oraz jest ona bardzo intuicyjna w tworzeniu interfejsu użytkownika ze względu na możliwość przeciągania i umieszczania elementów w tzw. „designerze” za pomocą elementów z „toolboxa”.

Wykorzystuję również język C# dla stworzenia biblioteki zgodnie z założeniami projektu. Zdecydowałem się na ten język z powodu łatwości implementacji algorytmu w C# oraz prostoty w łączenia pliku typu dll z GUI, gdyż są napisane w tych samych językach. Powoduje to, iż przekazywane obiekty do dllki będą tak samo rozpoznawane.

Wykorzystanie asemblera w projekcie było konieczne, natomiast ja wykorzystuje konkretnie MASM 64-bitowego w realizacji mojego projektu ze względu na doświadczenie nabyte w trakcie zajęć laboratoryjnych z przedmiotu Języki Asemblerowe.

## Wprowadzenie

Sepia jest techniką wywodzącą się z barwienia odbitek fotograficznych. W przeszłości stosowana głównie w celu zwiększenia trwałości odbitek. Obecnie symulacja podanego efektu jest otrzymywana przy wykorzystaniu programów komputerowych. Obrazy po zastosowaniu sepia otrzymują charakterystyczne brązowe zabarwienie. Sam efekt komputerowo jest uzyskiwany w niezbyt skomplikowany sposób w dwóch krokach. Pierwszym jest przekształcenie obrazu w odcienie szarości, a następnie koloryzuje się go określoną barwą. Podany proces wiąże się z odczytaniem barwy piksela dla otrzymanego obrazu w odcieniach szarości.

## Algorytm Sepii

Zgodnie z opisem przedstawionym w podpunkcie „Wprowadzenie” należy sposób na otrzymanie sepia podzielić na dwie części. W związku z traktowaniem obrazu jako jednowymiarowej tablicy bajtów algorytm wykonania podanego procesu będzie zrozumiały. Dlatego początkiem procesu jest otrzymanie podanej tablicy, gdzie każde cztery wartości reprezentują jeden piksel o składowych:

- Alpha -> składowa przezroczystości piksela;
- Red -> składowa czerwona piksela;
- Green -> składowa zielona piksela;
- Blue -> składowa niebieska piksela;

Blue	Green	Red	Alpha
------	-------	-----	-------

Rys.1 Przedstawiający reprezentacje wartości składowych piksela w programie

Należy pamiętać, iż dane w moim programie mają reprezentacje ARGB dla każdego piksela w tablicy.

### I część algorytmu: Skala szarości

Podany efekt otrzymujemy wykorzystując następujący wzór:

$$\begin{aligned}R_{szarosc} &= \frac{R_{kolor} + G_{kolor} + B_{kolor}}{3} \\G_{szarosc} &= \frac{R_{kolor} + G_{kolor} + B_{kolor}}{3} \\B_{szarosc} &= \frac{R_{kolor} + G_{kolor} + B_{kolor}}{3}\end{aligned}$$

Składowe R, G i B są w stosunku 1:1:1, ponieważ udział każdej składowej piksela jest taki sam. Uzyskujemy wzór na średnią arytmetyczną każdej składowej piksela. Następuje zsumowanie wartości każdej z barw, a potem dzielenia wartości bez reszty przez 3.

## II część algorytmu: Koloryzacja składowych obrazu

Podczas podanego procesu musimy wyodrębnić każdą składową piksela, a potem następnie dodaje się odpowiednie współczynniki wypełnienia dla odpowiednich składowych. Szczególnie wartość podanych współczynniki będzie wpływała na dwie wartości efektu sepii: intensywność oraz głębia efektu.

$$R = R + 2 * I$$

$$G = G + I$$

$$B = B - D$$

Gdzie:

- I -> współczynnik intensywności;
- D -> współczynnik głębi;

Dla współczynnika I ustalono przedział  $< 0,50 >$  ze skokiem co jedną wartość. Natomiast dla D przedział  $< 0, 120 >$  ze skokiem co dziesięć.

Należy również pamiętać o sprawdzaniu za pomocą instrukcji warunkowych czy nie przekroczono maksymalnej lub minimalnej wartości zmiennej typu byte. Musi się ona mieścić w przedziale  $< 0,255 >$ .

### Przykład

Dla jednego piksela o współczynnikach  $I = 20$  i  $D = 40$ .

B = 51	G = 204	R = 51	A = 101
--------	---------	--------	---------

Rys.2 składowe piksela dla przykładu z uwzględnieniem kanału przezroczystości



➔ Podany piksel reprezentuje wartości koloru zielonego

Średnia ze składowych :

$$R_{szarosc} = \frac{51 + 204 + 51}{3} = 102$$

$$G_{szarosc} = \frac{51 + 204 + 51}{3} = 102$$

$$B_{szarosc} = \frac{51 + 204 + 51}{3} = 102$$

Po obliczeniach otrzymujemy poniższy kolor:



→ Podany piksel reprezentuje wartości koloru szarości

Następnie uzupełniamy go o podane wcześniej współczynniki:

$$R = 102 + 2 * 20 = 142$$

$$G = 102 + 20 = 122$$

$$B = 102 - 40 = 62$$

Ostatecznie otrzymujemy poniższe wartości:

62	122	142	101
----	-----	-----	-----

Rys. 3 Końcowy wynik operacji, wartość kanału Alpha przepisana



→ Podany piksel reprezentuje wartości koloru o odcieniu brązowym

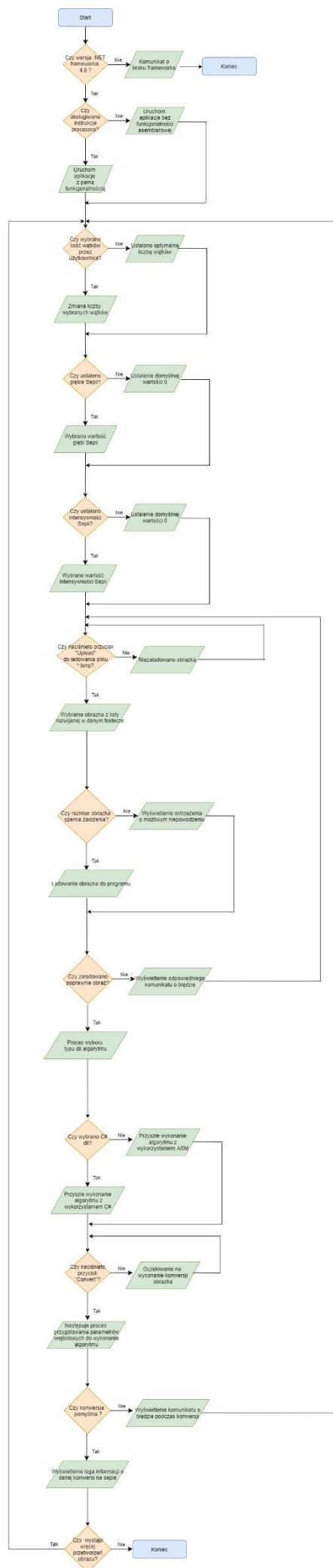
## Schemat blokowy programu

Poniższy rysunek reprezentuje sposób przepływu danych przez programu oraz jego interakcje z użytkownikiem. Przy starcie programu sprawdzane jest czy na systemie Windows zainstalowany jest aktualny .NET framework ( wersja minimum 4.8) oraz czy dany procesor obsługuje wykorzystywane instrukcje wektorowe SSE oraz AVX. Następnie założono, że użytkownik wybierze sobie parametru koloryzacji obrazu, ale również jeśli ich nie ustali to będą one domyślnie wybrane jako 0. Natomiast typ dllki ustawiono domyślnie na C#, a ilość optymalna wątków = ilość logicznych procesorów -1.

Formatem obrazu jaki może wybrać użytkownik jest .bmp. Jeżeli zostaną wybrane parametry obrazu Intensywność: 0 oraz Głębina: 0 to otrzymane zdjęcie będzie w odcieniach szarości. Po określeniu parametrów zdjęcia oraz jego samego wyboru użytkownik może wcisnąć przycisk „Convert”, który dokonuje konwersji obrazu na podany efekt.

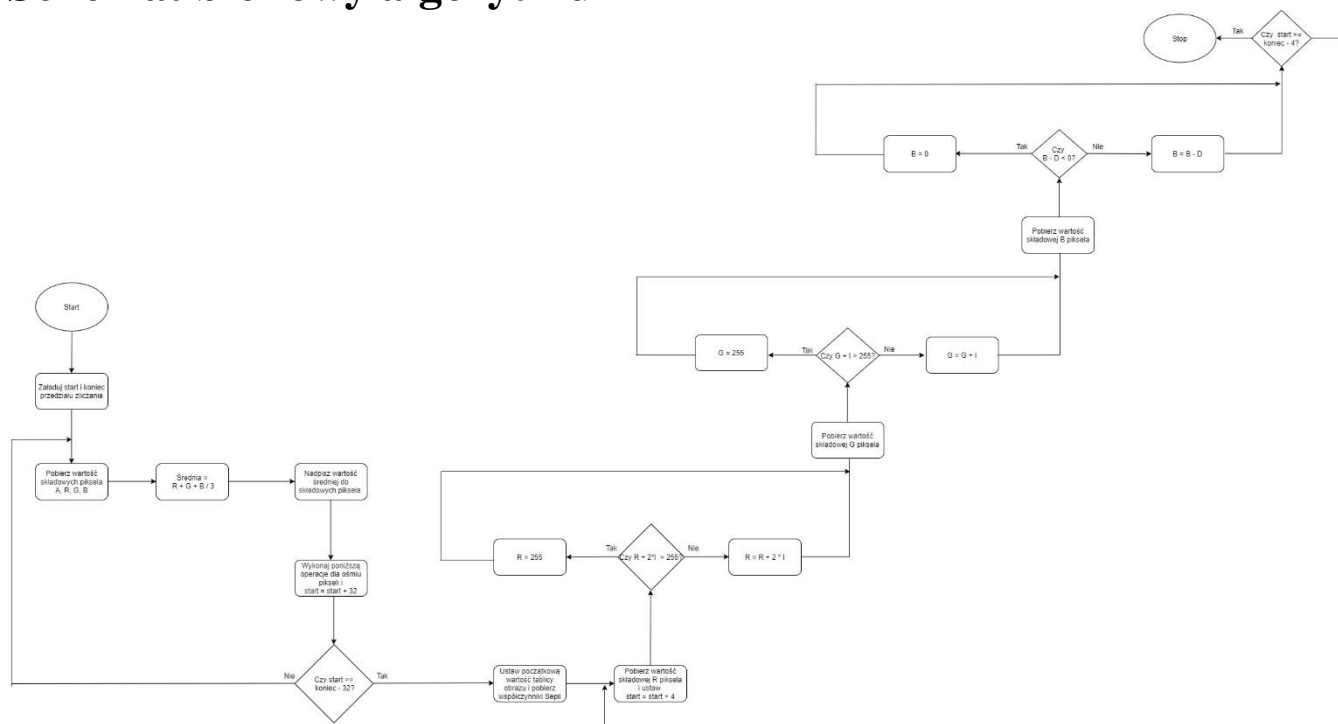
Podczas działania programu możemy napotkać na poniższe komunikaty zwrotne programu:

- Processor do not suport MMX or AVX instructions -> Error, blokuje możliwość obsługi dllki assemblerowej,
- Outdated version of .NET framework -> Error, wychodzi z program z powodu braku odpowiedniej wersji frameworka,
- File dimension is huge! Proceed at your own risk that it may fail to load/convert image -> Warning, ostrzega użytkownika o możliwy niepowodzeniu konwersji dla dużych plików wartościowo,
- Error while loading the image! File was corrupted or of incorrect file extension -> Error, zabezpieczenie przed załadowaniem niepoprawnego format pliku,
- Error while trying to convert image -> Error, zabezpieczenie przed próbom niepowodzenia w trakcie przygotowania do konwersji obrazka,
- Image is too small to be converted Please provide bigger image at least 64 x 64 -> Error, rozmiar pliku za mały do konwersji.



Rys.4 Reprezentuje przepływ działania w programie oraz reakcji na błędne dane wprowadzone przez użytkownika

# Schemat blokowy algorytmu



Rys. 5 Reprezentujący schemat blokowy algorytmu Sepii

Powyższy schemat przedstawia sposób rozwiązania zadania przekształcenia obrazu w sepie w bibliotece ASM i C#. Należy jednak mieć na uwadze fakt, iż dane ładowane w assemblerze będą znajdowały się w odpowiednich rejestrach co wiąże się z umiejętnym sposobem manipulowania wartościami, zaś w bibliotece C# dane są przekształcane z wykorzystaniem odpowiedniej tablicy bajtów. Schemat jest wizualną reprezentacją wcześniej opisanego podejścia rozwiązania problemu w podpunkcie „Algorytm Sepii”.

## Opis programu w języku wysokiego poziomu

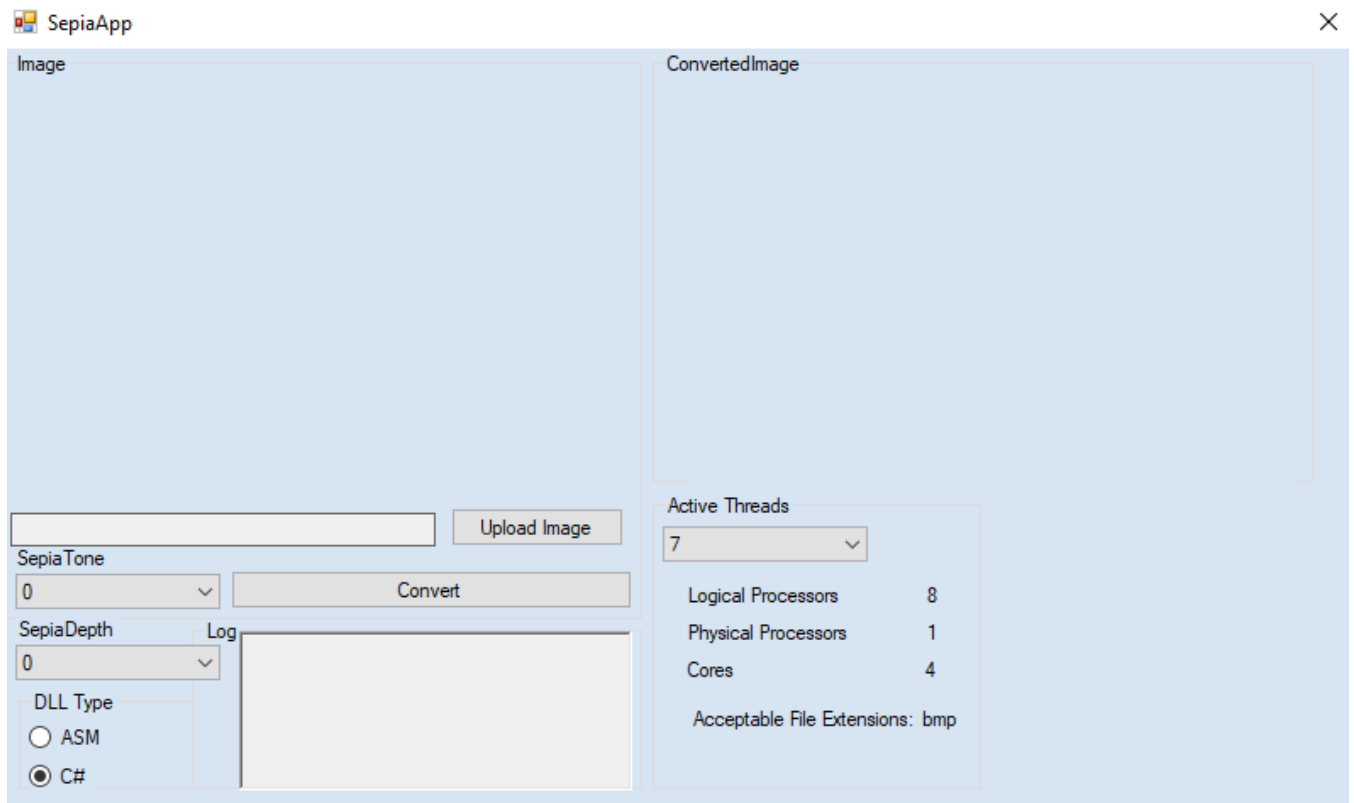
Program został napisany w języku C# w technologii Windows Forms daje użytkownikowi możliwość wprowadzenia następujących wartości:

- Ilości aktywnych wątków;
- Typu biblioteki (Asm lub C#);
- Intensywności sepia;
- Głębi sepia;
- Rodzaju obrazka w formacie bmp;

Intensywność efektu sepia znajduje się pod napisem „SepiaTone” można wybrać wartości z przedziału <0;50> ze skokiem co jedną wartość. Następnie pod napisem „SepiaDepth” znajduje się wartość głębokości sepia. Podany współczynnik można ustawić na wartości z przedziału <0;120> ze skokiem co dziesięć wartości. Zgodnie z założeniami ilość wątków do wybrania wynosi od 1 do 64. Jednocześnie w postaci dobrze przygotowanego opisu użytkownik otrzymuje informacje o ilości logicznych, fizycznych procesorów z uwzględnieniem funkcji HyperThreading. Również podana jest liczba samych rdzeni procesora. Informacje te są pobierane z odpowiednich rejestrów komputera. Jeżeli użytkownik nie ustali wartości to program sam ustawia domyślną wartość parametrów opisanych powyżej. Wartości te to: głębokość sepia -> 0, intensywność sepia -> 0 oraz liczba wątków jest ustawiona na optymalną ze wzoru: liczba procesorów logicznych - 1.

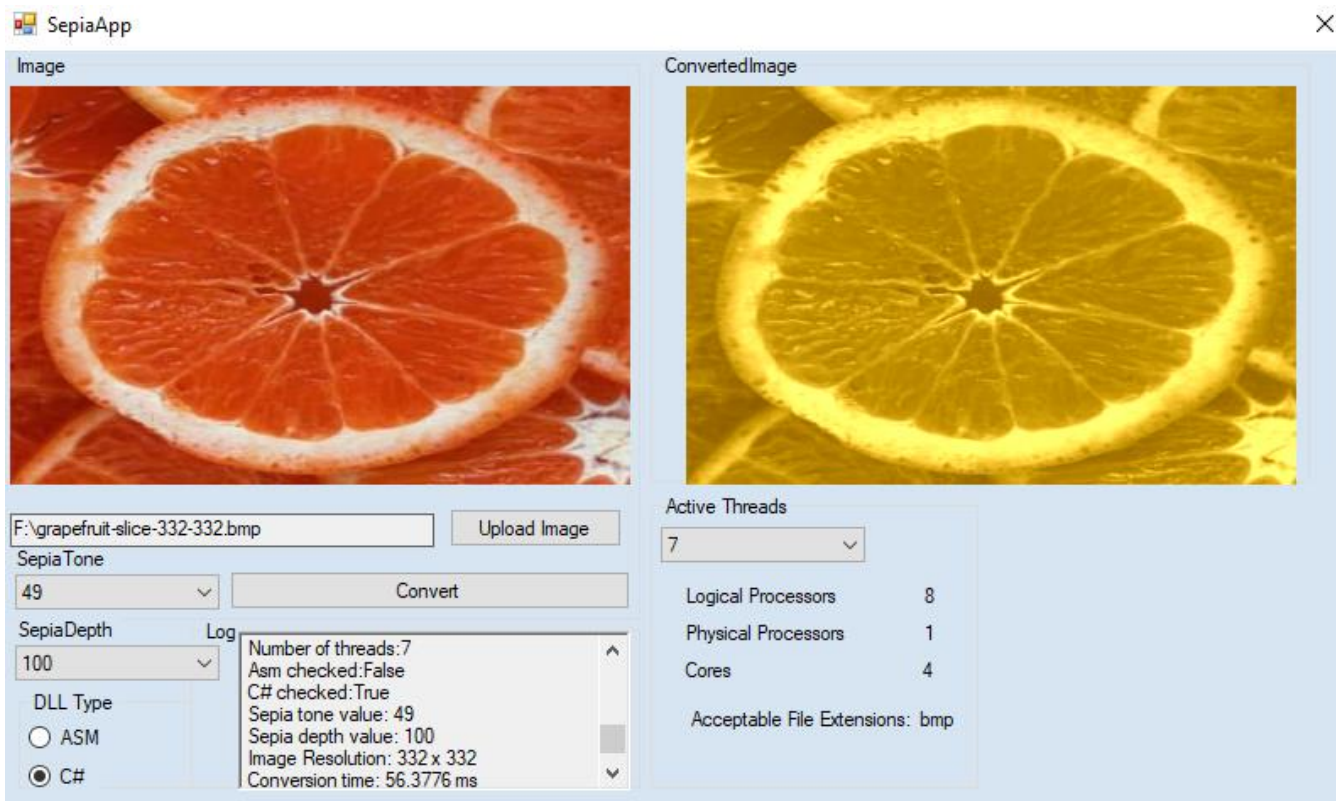
Kolejnym krokiem jest wybranie poszukiwanego obrazka poprzez wciśnięcie przycisku „Upload image”. Pojawi się wtedy dla użytkownika lista rozwijana folderów oraz ich zawartości spośród, których wybierze poszukiwany plik \*.bmp, a następnie zatwierdzi jego wybór. Użytkownik poprzez wybrany filtr ma możliwość wybrania tylko plików .bmp, ale również w trakcie zatwierdzenia wybrania obrazu jest

sprawdzana jego poprawność, a w przypadku dużego rozmiaru wyświetlane ostrzeżenie o niebezpieczeństwie związanym z poprawnością działania programu. Ostatnim etapem jest wciśnięcie przycisku Convert. Następnie zachodzi proces przydziału części elementów tablicy bajtów dla każdego wątku. Podział wątków jest dokonywany za pomocą klasy Thread z wykorzystaniem tablicy wątków Thread oraz metod isAlive() oraz Start() i GetLength(). Podział na części dla wątków jest tworzony w prosty stosunkowo sposób poprzez dzielenie rozmiaru tablic obrazu bajtów przez liczbę wątków. Następnie konkretne przedziały wartości tworzone są w dwóch pętlach while zachowując zawsze warunek, iż jeden wątek wykona ilość elementów, której reszta z dzielenia mod32 jest równa 0. Przedziały są reprezentowane przez dwie liczby start i stop, gdzie każdy wątek otrzymuje przedział <start,stop>, wartość stop nie jest wykonywana (z wyjątkiem ostatniego wątku), ponieważ jest ona początkiem przedziału kolejnego wątku. Po dokonaniu podziałów za pomocą obiektu ParametrizedThreadStart dokonujemy rozpoczęcia zliczania czasu oraz wejścia do odpowiedniej dllki wykonującej algorytm. Użytkownik po każdej operacji wciśnięcia przycisku „Convert” musi odczekać minimum 2 sekundy przed kolejną możliwą konwersją. Zabezpiecza się wtedy przed możliwym spowolnieniem pracy działania programu.



Rys. 15 Wygląd gotowego interfejsu użytkownika przed wprowadzeniem danych





Rys.16 Wygląd interfejsu po przykładowej konwersji, okno Log przechowuje informacje o istotnych operacjach w trakcie konwersji obrazu

## Opis interfejsu użytkownika

Informacje widoczne na rysunku Rys.15 są następujące:

- Upload Image -> Przycisk ładowania obrazka typu bmp,
- SepiaTone -> Wartość intensywności sepia z  $\langle 0,60 \rangle$  co 1,
- SepiaDepth -> Wartość głębi sepia z  $\langle 0,140 \rangle$  co 10,
- DLL Type -> Rodzaj użytej biblioteki: C# lub ASM,
- Log -> Zbiór informacji na temat ostatniej konwersji obrazu
- Active Threads -> Liczba aktywnych wątków użytych do konwersji,
- Logical Processors -> Liczba logicznych rdzeni procesora z uwzględnieniem HyperThreading,
- Physical Processors -> Liczba fizycznych rdzeni procesora,
- Cores -> Liczba rdzeni procesora,
- Acceptable File Extensions -> Rodzaj akceptowanych rozszerzeń przez program. Wybrano tylko dla rozszerzeń typu bmp,
- Image -> Graficzny obszar rysowania załadowanego obrazu przez użytkownika,
- ConvertedImage -> Obraz po konwersji, ewentualnie jego brak w razie błędnych danych,

# Opis funkcji bibliotek języka C#

```
namespace SepiaDll
{
    // klasa obsługująca efekt Sepii dla języka wysokiego poziomu.

    1 reference
    public class Sepia
    {
        /// <summary>
        /// metoda odpowiedzialna za koloryzowanie danego obrazu na sepie
        /// wykonuje najpierw w jednej pętli algorytm odcieni szarości dla obrazu
        /// w drugiej pętli wykonuje algorytm wypełnienia sepią dla przetworzonego
        /// wcześniej obrazu w pierwszej pętli
        /// </summary>
        /// <param name="args">
        /// object args -> Jest to obiekt 5 elementów typu Array, zadeklarowanych w GUI
        /// </param>
        /// parametry, które przechowuje ta kolekcja są następujące:
        /// RGBstart -> wartość typu int początku przedziału tablicy bajtów obrazu
        /// RGBstop -> wartość typu int koniec przedziału tablicy bajtów obrazu
        /// sepiavalue -> wartość typu int wypełnienia sepii
        /// depthvalue -> wartość typu int głębili sepii
        /// </param>
        /// returns void
        1 reference
        public static void CSharpDllFunc(object args)
        {
            // WEJŚCIE DO DLL C# SEPII
            // POBRANIE WARTOŚCI Z OBIEKTU 5 ELEMENTOWEGO DO ODPowiednich ZMIENNYCH
            // tworzenie tablicy 5 obiektów typu Array, ponieważ takie elementy znajdują się w obiekcie przekazywanym przez język wysokiego poziomu

            Array arguments = new object[5];
            arguments = (Array)args; // rzutowanie danego obiektu na Array

            int RGBstart = (int)arguments.GetValue(0); // zapisywanie do zmiennej początek przedziału tablicy bajtów dla danego wątku
            int RGBstop = (int)arguments.GetValue(1); // zapisywanie do zmiennej koniec przedziału tablicy bajtów dla danego wątku
            byte[] rgbvalues = (byte[])arguments.GetValue(2); // zapisywanie wskaźnika na tablice bajtów dla danego wątku
            int sepiavalue = (int)arguments.GetValue(3); // zapisywanie do zmiennej wartości wypełnienia sepią
            int depthvalue = (int)arguments.GetValue(4); // zapisywanie do zmiennej wartości głębili sepii
        }
    }
}
```

Rys.5 Reprezentacja wartości przyjmowanych przez bibliotekę

Na powyższym rysunku biblioteka z wykorzystaniem funkcji `CSharpDllFunc(object args)` ustawia do odpowiednich zmiennych wartości z pięcioelementowego obiektu `args`. Dane ze względu na przenoszenie ich jako tablice obiektów typu `Array` muszą być jawnie rzutowane w bibliotece na odpowiadające jej zmienne. Dalsze komentarze znajdują się na zdjęciu powyżej dotyczące kodu.

```

// TWORZENIE PETLI TYPU FOR
// SKOK W PETLI CO 32 WARTOŚCI, czyli 8 pikseli
for (int i = RGBstart; i <= RGBstop - 32; i += 32)
{
    // tworzenie zmiennej average (ŚREDNIA) i zapisanie danych tablicy i-tej do tej zmiennej
    int average = rgbValues[i];
    // dodanie wartości i + 1 tablicy do average
    average += rgbValues[i + 1];
    // dodanie wartości i + 2 tablicy do average
    average += rgbValues[i + 2];
    // dzielenie wartości average przez 3, aby otrzymać średnią i przypisywanie do zmiennej average (ŚREDNIA)
    average /= 3;
    // wpisywanie jako bajt wartości średniej do i-tej wartości tablicy
    rgbValues[i] = (byte)average;
    // wpisywanie jako bajt wartości średniej do i + 1 wartości tablicy
    rgbValues[i + 1] = (byte)average;
    // wpisywanie jako bajt wartości średniej do i + 2 wartości tablicy
    rgbValues[i + 2] = (byte)average;

    // przypisanie do zmiennej average danych tablicy i + 4
    average = rgbValues[i + 4];
    // dodanie wartości i + 5 tablicy do average
    average += rgbValues[i + 5];
    // dodanie wartości i + 6 tablicy do average
    average += rgbValues[i + 6];
    // dzielenie wartości average przez 3, aby otrzymać średnią i przypisywanie do zmiennej average (ŚREDNIA)
    average /= 3;
    // wpisywanie jako bajt wartości średniej do i + 4 wartości tablicy
    rgbValues[i + 4] = (byte)average;
    // wpisywanie jako bajt wartości średniej do i + 5 wartości tablicy
    rgbValues[i + 5] = (byte)average;
    // wpisywanie jako bajt wartości średniej do i + 6 wartości tablicy
    rgbValues[i + 6] = (byte)average;

    // przypisanie do zmiennej average danych tablicy i + 8
    average = rgbValues[i + 8];
    // dodanie wartości i + 9 tablicy do average
    average += rgbValues[i + 9];
    // dodanie wartości i + 10 tablicy do average
    average += rgbValues[i + 10];
    // dzielenie wartości average przez 3, aby otrzymać średnią i przypisywanie do zmiennej average (ŚREDNIA)
    average /= 3;
    // wpisywanie jako bajt wartości średniej do i + 8 wartości tablicy
    rgbValues[i + 8] = (byte)average;
    // wpisywanie jako bajt wartości średniej do i + 9 wartości tablicy
    rgbValues[i + 9] = (byte)average;
    // wpisywanie jako bajt wartości średniej do i + 10 wartości tablicy
    rgbValues[i + 10] = (byte)average;

    // przypisanie do zmiennej average danych tablicy i + 12
    average = rgbValues[i + 12];
    // dodanie wartości i + 13 tablicy do average
    average += rgbValues[i + 13];
    // dodanie wartości i + 14 tablicy do average
    average += rgbValues[i + 14];
    // dzielenie wartości average przez 3, aby otrzymać średnią i przypisywanie do zmiennej average (ŚREDNIA)
    average /= 3;
    // wpisywanie jako bajt wartości średniej do i + 12 wartości tablicy
    rgbValues[i + 12] = (byte)average;
    // wpisywanie jako bajt wartości średniej do i + 13 wartości tablicy
    rgbValues[i + 13] = (byte)average;
}

```

Rys. 6 Początek implementacji algorytmu odcieni szarości

```

    rgbValues[i + 14] = (byte)average;           // wpisywanie jako bajt wartości średniej do i + 14 wartości tablicy
    average = rgbValues[i + 16];                // przypisanie do zmiennej average danych tablicy i + 16
    average += rgbValues[i + 17];                // dodanie wartości i + 17 tablicy do average
    average += rgbValues[i + 18];                // dodanie wartości i + 18 tablicy do average
    average /= 3;                                // dzielenie wartości average przez 3, aby otrzymać średnią i przypisywanie do zmiennej average ($REDNIA)
    rgbValues[i + 16] = (byte)average;           // wpisywanie jako bajt wartości średniej do i + 16 wartości tablicy
    rgbValues[i + 17] = (byte)average;           // wpisywanie jako bajt wartości średniej do i + 17 wartości tablicy
    rgbValues[i + 18] = (byte)average;           // wpisywanie jako bajt wartości średniej do i + 18 wartości tablicy

    average = rgbValues[i + 20];                // przypisanie do zmiennej average danych tablicy i + 20
    average += rgbValues[i + 21];                // dodanie wartości i + 20 tablicy do average
    average += rgbValues[i + 22];                // dodanie wartości i + 21 tablicy do average
    average /= 3;                                // dzielenie wartości average przez 3, aby otrzymać średnią i przypisywanie do zmiennej average ($REDNIA)
    rgbValues[i + 20] = (byte)average;           // wpisywanie jako bajt wartości średniej do i + 20 wartości tablicy
    rgbValues[i + 21] = (byte)average;           // wpisywanie jako bajt wartości średniej do i + 21 wartości tablicy
    rgbValues[i + 22] = (byte)average;           // wpisywanie jako bajt wartości średniej do i + 22 wartości tablicy

    average = rgbValues[i + 24];                // przypisanie do zmiennej average danych tablicy i + 24
    average += rgbValues[i + 25];                // dodanie wartości i + 25 tablicy do average
    average += rgbValues[i + 26];                // dodanie wartości i + 26 tablicy do average
    average /= 3;                                // dzielenie wartości average przez 3, aby otrzymać średnią i przypisywanie do zmiennej average ($REDNIA)
    rgbValues[i + 24] = (byte)average;           // wpisywanie jako bajt wartości średniej do i + 24 wartości tablicy
    rgbValues[i + 25] = (byte)average;           // wpisywanie jako bajt wartości średniej do i + 25 wartości tablicy
    rgbValues[i + 26] = (byte)average;           // wpisywanie jako bajt wartości średniej do i + 26 wartości tablicy

    average = rgbValues[i + 28];                // przypisanie do zmiennej average danych tablicy i + 28
    average += rgbValues[i + 29];                // dodanie wartości i + 29 tablicy do average
    average += rgbValues[i + 30];                // dodanie wartości i + 30 tablicy do average
    average /= 3;                                // dzielenie wartości average przez 3, aby otrzymać średnią i przypisywanie do zmiennej average ($REDNIA)
    rgbValues[i + 28] = (byte)average;           // wpisywanie jako bajt wartości średniej do i + 28 wartości tablicy
    rgbValues[i + 29] = (byte)average;           // wpisywanie jako bajt wartości średniej do i + 29 wartości tablicy
    rgbValues[i + 30] = (byte)average;           // wpisywanie jako bajt wartości średniej do i + 30 wartości tablicy
}
// NASTĘPNA ITERACJA PĘTLI, PROCES POWTARZA SIĘ DOPÓKI WĄTEK W PĘTLI NIE SKOŃCZY OBLICZEŃ
// UZYSKANO EFEKT SZAROŚCI PO ZAKOŃCZENIU 1. PĘTLI

```

Rys. 7 Koniec implementacji algorytmu odcieni szarości

Na powyższych dwóch rysunkach uzyskano podany efekt dla 32 wartości, czyli 8 pikseli w jednej iteracji pętli. Wynika to z faktu wektorowego przetwarzania wielu pikseli w bibliotece asemblerowej. Zgodnie z zaleceniem prowadzącego dokonano takiej operacji, aby obie reprezentacje biblioteczne algorytmów pokrywały się. Zmienną average wykorzystano do przechowywania średniej arytmetycznej dla danego piksela. Podana zmienna musi być jawnie rzutowana na typ byte, ponieważ wartości składowych piksela w bibliotece są typu byte.

```

// DRUGA CZĘŚĆ OPERACJI
//Pętla tworząca efekt sepii dla przetworzonego obrazka odcienie czarości, skok w pętli co 4 wartości
for (int i = RGBstart; i <= RGBstop - 4; i += 4)
{
    // NASTĄPI W TEJ PĘTLI SPRAWDZENIE KILKU WARUNKÓW IF, aby odpowiednio uzyskać koloryzację obrazu odpowiednią barwą, aby otrzy efekt sepii
    if (rgbValues[i] < depthValue) // warunek sprawdzający czy podana wartość głębii sepii jest większa niż wartość składowej niebieskiej piksela
    {
        rgbValues[i] = 0; // jeżeli wartość składowej jest mniejsza to ustawiana jest ona na 0
    }
    else
    {
        rgbValues[i] -= (byte)depthValue; // jeżeli wartość głębii sepii jest mniejsza od składowej niebieskiej piksela to następuje przypisanie do składowej niebieskiej wartości pomniejszonej o wartość głębii
    }

    if (rgbValues[i + 1] > (255 - sepiValue)) // warunek sprawdzający czy przekroczono wartość 255 dla koloru zielonego
    {
        rgbValues[i + 1] = 255; // ustawienie maksymalnej dozwolonej wartości koloru, gdy przekroczono maksymalną wartość 255 dla składowej czerwonej piksela
    }
    else
    {
        rgbValues[i + 1] += (byte)sepiValue; // ustawienie dla wartości składowej czerwonej piksela wypełnienia współczynnikiem sepii wynikającej z algorytmu
    }

    if (rgbValues[i + 2] > (255 - 2 * sepiValue)) // warunek sprawdzający czy przekroczono wartość 255 dla koloru czerwonego
    {
        rgbValues[i + 2] = 255; // ustawienie maksymalnej dozwolonej wartości koloru czerwonego, gdy przekroczono maksymalną wartość 255
    }
    else
    {
        rgbValues[i + 2] += (byte)(2 * sepiValue); // dodanie podwójnej wartości składowej wypełnienia do wartości koloru czerwonego piksela, wynikające z algorytmu
    }
}

```

Rys.8 Proces koloryzacji obrazu po wykonaniu algorytmu odcieni szarości

Na powyższym rysunku znajduje się kontynuacja zadania z rysunku 7 oraz 8. Jest to kolejna pętla odpowiedzialna za koloryzację do sepii. Zachodzi tutaj wyodrębnienie pojedynczych składowych oraz sprawdzenie z wykorzystaniem instrukcji warunkowych czy kwalifikują się na zmianę wartości, aby otrzymać brązowy efekt. Zmienna depthValue odpowiada za głębie Sepii, natomiast sepiaValue za intensywność podanego efektu. Zgodnie z algorytmem opisanym w podpunkcie „Schemat blokowy algorytmu” zostało przedstawione działanie algorytmu. Należy pamiętać również o jawnym rzutowaniu wartości na typ byte, aby zachować odpowiednią strukturę tablicy.

## Opis funkcji bibliotek języka Asembler

```

; parametry wejściowe dla funkcji wykonanego poziomu w rc
; Sepia(ptr, start, stop, tonevalue)
; opis:
; ptr -> wskaźnik do tablicy bajtów obrazu typu byte
; start -> początek przedziału dla tablicy obrazu, dla której dany wątek wykonuje obliczenia typu int
; stop -> koniec przedziału dla tablicy obrazu, dla której dany wątek wykonuje obliczenia typu int
; tonevalue -> wartość wypełnienia sepii typu int

; Deklaracja stałych wykorzystywanych w celu maskowania odpowiednich wartości składowych piksela R,G,B
data
divider_value dq 0000000000000000h ; divider_value -> stała wartość 1 zapisana w postaci maski w celu wykonania dzielenia wektorowego typu Quad word
alpha_mask dq 00ff000000ff0000h ; alpha_mask -> maska kanału przezroczystości dla każdego piksela typu Quad word
color_r_mask dq 00ff000000ff0000h ; color_r_mask -> maska koloru czerwonego dla każdego piksela typu Quad word
code

; procedura odpowiedzialna za przetwarzanie kolorowego obrazu do efektu sepii
;-----
; 1.parametry wejściowe przekazywane przez rejestry rax rax 2.parametry w języku wysokiego poziomu
; 1.rcx 2. imageptr Adres obrazka
; 1.rcx 2. start Start przedziału tablicy dla wątku
; 1.rcx 2. stop koniec przedziału tablicy dla wątku
; 1.rcx 2. tonevalue współczynnik wypełnienia sepii
; 1. word ptr [rbp + 48] 2. depthvalue współczynnik głębii sepii
; parametry wyjściowe:
; void
;-----
; główna procedura sepii zajmująca się konwersją obrazka na odcienie szarości i wypełnienia składowych piksela, aby uzyskać efekt sepii
sepia proc
; zachowanie rejestru rbp na stos
mov rbp, rsp
; zapis rejestru rbp do rbp.
mov rdi, rbp
; zachowanie rejestru rdi na stos
mov rsi, rdi
; zachowanie rejestru rsi na stos
mov r10, rsi
; zachowanie rejestru r10 na stos
mov r11, r10
; zachowanie rejestru r11 na stos
mov r12, r11
; pobranie wartości 5. parametru do rejestru r14 zawierającego wartość głębii sepii. Jest to parametr pobrany ze stosu
mov r14, qword ptr [rbp + 48]
; ładowanie adresu obrazka do rcx, aby go zapamiętać
mov rcx, rcx
; ładowanie początku tablicy do r11 na późniejsze przetwarzanie tablicy
mov r11, r11
; załadowanie do rcx końca przedziału tablicy
mov rcx, rcx
; załadowanie wartości wypełnienia sepii do rejestru r12
mov r12, r12
; przesunięcie adresu tablicy pikseli obrazka do rejestru rdi (rejestr indeksowy), aby obliczyć indeks od którego będziemy liczyć piksele
mov rdi, r11
; dodanie przesunięcia zwanego z podziałem na wątki do rejestru rdi dodając to niego rejestr rcx, w rejestrze rdi aktualnie znajduje się adres tablicy pikseli obrazka
mov rdi, rcx
; załadowanie ilości bitów do przetworzenia, czyli koniec przedziału - początek przedziału do rejestru rcx poprzez odjęcie rejestru rcx od rdx
mov rcx, rdx

```

Rys.9 Początek pliku asm funkcji Sepia proc oraz ładowanie parametrów do odpowiednich rejestrów

```

mov rax, color_r_mask ; ładowanie stałej maski koloru czerwonego 0 do rejestru rax, w celu późniejszego zapisania danych składowych piksela
mov rax, alpha_mask ; ładowanie stałej maski koloru czerwonego 0 do dolnej części rejestru ymm i wykorzystanie rejestru ymm, aby wykonać operację wektorową na całym rejestrze ymm
mov rax, divider_value ; ładowanie stałej maski liczby "0" (divider_value) dla wartości składowych piksela do rejestru ymm, aby wykonać dzielenie przez sumę składowych danego piksela
mov rax, alpha_mask ; ładowanie stałej maski przezroczystości dla kanału alpha do rejestru ymm, aby zapamiętać kanał przezroczystości dla piksela
mov rax, alpha_mask ; ładowanie stałej maski przezroczystości do dolnej części rejestru ymm i wykorzystanie rejestru ymm, aby zapamiętać kanał przezroczystości dla 4 pikseli
mov rax, 00h ; ustawienie maski koloru czerwonego w całym obszarze rejestru ymm, począwszy od początku rejestru ymm
mov rax, 00h ; w celu wykonania maskowania dla całego rejestru i uwzględnienie odpowiedniego przesunięcia na początek wartości
mov rax, 00h ; ustawienie dzielnika w całym obszarze rejestru ymm, począwszy od początku rejestru ymm
mov rax, 00h ; w celu wykonania maskowania dla całego rejestru i uwzględnienie odpowiedniego przesunięcia na początek wartości
mov rax, 00h ; ustawienie maski kanału przezroczystości w całym obszarze rejestru ymm, począwszy od początku rejestru ymm
mov rax, 00h ; w celu wykonania maskowania dla całego rejestru i uwzględnienie odpowiedniego przesunięcia na początek wartości
mov rax, 00h ; zamiana wartości dzielnika 1 (divider_value) z typu int na float w celu wykonania dzielenia wektorowego przez liczbę zmiennoprzecinkową i zapamiętanie w rejestrze ymm

```

Rys. 10 Ładowanie do odpowiednich rejestrów maski oraz ustawienie jej na odpowiednią górną i dolną część rejestru

average_loop:	start pętli wykonującej odcienie szarości algorytmu. Złazł pierwsza wykonawcza sekcja algorytmu sepi.
movl rdi, [rdi]	połączenie 4 pikseli z tablicy bajtów obrazu z rejestru rdi do xmm0, aby wykonać na nich operacje niezależne do wykonania algorytmu sepi.
rci, 16	przesunięcie rejestru indeksowego o 16 pozycji do przodu, w celu pobrania nowych wartości z rejestru rdi.
rci, 16	przesunięcie rejestru zliczającego o 16 pozycji do tyłu z rejestru rcx, aby nie wyjść poza zakres tablicy.
xmm, [0]	połączenie kolejnych 4 pikseli z rejestru rdi do xmm, aby wykonać na nich kolejne operacje niezależne do wykonania algorytmu sepi.
psrlq xmm, xmm, xmm0, 8	przesunięcie 4 pikseli do górnej części rejestru xmm, aby później wykonać na nich operacje wektorowe SIMD na 4 pikselach.
psrlq xmm, xmm, xmm0, 1	przesunięcie 4 pikseli do dolnej części rejestru xmm, aby później wykonać operacje wektorowe SIMD na 4 pikselach.
	przetworzenie aktualnie 8 pikseli jednocześnie w rejestrze xmm.
xmm, xmm	zamianowanie składników alpha 8 kolejnych pikseli w rejestrze xmm, aby odpowiednio później przekazać wartości kanału przezroczystości do przetworzonych pikseli.
xmm, xmm, xmm	zamianowanie kanału przezroczystości 8 kolejnych pikseli w rejestrze xmm, aby otrzymać jedynie wartości składników pikseli R,G,B.
xmm, xmm	przepisanie 4 kolejnych pikseli bez kanału przezroczystości do rejestru xmm i w celu wyłączenia z podanego rejestru odpowiednich składników pikseli.
psrlq xmm, xmm, 1	logiczne przesunięcie rejestru xmm o 1 wartości w kierunku lewym w lewo, które jest zamaskowane wewnętrznie stałą maską koloru R, aby otrzymać maskę koloru R.
xmm, xmm	wykorzystanie do zamaskowania kolorów zielonych dla danego pikseli z jednocześnie przetwarzania tych wartości w podanej pozycji.
xmm, xmm	zamaskowanie rejestru xmm do xmm, aby przekształcić stałą maskę koloru zielonego.
xmm, xmm, 1	logiczne przesunięcie rejestru xmm o 2 wartości w kierunku lewym w lewo, które jest zamaskowane wewnętrznie stałą maską koloru B, aby otrzymać maskę koloru B.
	wykorzystanie do zamaskowania kolorów niebieskich dla danego pikseli z jednocześnie przetwarzania tych wartości w podanej pozycji.
xmm, xmm, xmm	zamaskowanie składników pikseli R, G, B (R - kanał alpha) rejestru xmm z wykorzystaniem maski koloru czerwonego z rejestru xmm i ten sposób uzyskujemy same kolory czerwone 8 pikseli.
xmm, xmm, xmm	zamaskowanie składników pikseli R, G, B rejestru xmm z wykorzystaniem maski koloru czerwonego z rejestru xmm. Używamy same kolory zielone 8 pikseli.
xmm, xmm, xmm	zamaskowanie składników pikseli R, G, B rejestru xmm z wykorzystaniem maski koloru czerwonego z rejestru xmm. Używamy same kolory niebieskie 8 pikseli.
xmm, xmm, xmm	zamaskowanie rejestru xmm i xmm do rejestru xmm. Używamy zielonego koloru czerwone R i zielone G ( Bw). W celu wykonania średniej arytmetycznej składników pikseli.
xmm, xmm, xmm	zamaskowanie rejestru xmm i xmm do rejestru xmm. Używamy tego koloru czerwonego i zielonego oraz koloru niebieskiego ( BwB). Można dzięki temu obliczyć średnią arytmetyczną pikseli.
xmm, xmm	zamiana wartości int rejestru xmm na float do rejestru xmm w celu wykonania dzielenia wektorowego. Dzielenie wartości z kolejnych składników pikseli przez wartość stałej maski dzielnika 1 (divisor_value).
xmm, xmm, xmm	wykonanie dzielenia wektorowego w celu otrzymania średniej arytmetycznej ( BwB) / 1, dzielenie wektorowe rejestru xmm przez rejestr stałego dzielnika 1 (divisor_value) (rejestr xmm) do rejestru xmm.
xmm, xmm	zamiana wartości float rejestru xmm na int do rejestru xmm, aby później móc zapisywać wartości do tablicy bajtów.
xmm, xmm	przekazanie rejestru xmm do rejestru xmm, aby zamknąć wartości średniej arytmetycznej pikseli do późniejszego przekazania.
psrlq xmm, xmm, 1	logiczne przesunięcie rejestru xmm ( aktualnie obliczone średnie arytmetyczne pikseli) o 2 wartości lewym w celu wyznaczenia wartości x na odpowiednią pozycję, gdzie x = ( R + G + B ) / 3 do rejestru xmm.
	reprezentacja wartości w rejestrze xmm: R,G,B,R.
xmm, xmm, xmm	logiczne operacja OR na rejestrze xmm i xmm do rejestru xmm, w celu ustalenia pozycji pikseli na odpowiednie bity.
	wykonawca w celu ustalenia 3 pierwszych pozycji dla pikseli.
psrlq xmm, xmm, 1	logiczne przesunięcie rejestru xmm o 2 wartości lewym w celu ustalenia pozycji pikseli na odpowiednie bity.

Rys. 11 Początek konwersji obrazu do odcieni szarości

xmm, xmm, xmm	logiczna operacja OR na rejestrze xmm i xmm do rejestru xmm, w celu ustalenia pozycji pikseli na odpowiednie bity.
	wykonawca w celu ustalenia 3 ostatnich pozycji dla pikseli.
xmm, xmm, xmm	przepisanie wartości kanału alpha.
xmm, xmm, xmm	logiczne OR na rejestrze xmm i xmm w celu przepisania do rejestru xmm wartości kanału alpha z rejestru xmm na odpowiednią pozycję dla pikseli.
rci, 16	przesunięcie rejestru indeksowego o 16 pozycji do tyłu, w celu podniesienia odpowiedniego umieszczenia pikseli w tablicy obrazu na starszej pozycji tablicy.
movl rdi, xmm0, 0	przepisanie dolnej części rejestru xmm do rejestru xmm0 w celu wpisania wynikowych wartości do tablicy ( 4 piksele).
movl [rdi], xmm0	przesunięcie wartości wynikowych pikseli z rejestru xmm0 do rejestru indeksowego rdi, gdzie są one zapisywane do tablicy.
rci, 16	przesunięcie rdi o 16 do przodu w celu umieszczenia pikseli w tablicy obrazu na starszej pozycji tablicy.
movl rdi, xmm0, 1	przepisanie górnej części rejestru xmm do rejestru xmm0 w celu wpisania wynikowych wartości do tablicy ( 4 piksele).
movl [rdi], xmm0	przesunięcie wartości wynikowych pikseli z rejestru xmm0 do rejestru indeksowego rdi, gdzie są one zapisywane do obrazu.
rci, 16	przesunięcie rdi o 16, aby znajdował się na właściwej pozycji obrazu w trakcie przetwarzania pikseli.
rci, 16	odjęcie od indeksu zliczającego rcx 16, aby nie wyjść poza zakres tablicy.
rci, 31	sprawdzenie czy przekroczono rozmiar tablicy i zakończono zliczanie.
jeq prepare	jeżeli został przekroczony rozmiar tablicy przygotowujemy do wypełnienia obrazu efektem sepi.
jmp average_loop	bezwzględny skok do pętli zliczającej średnią arytmetyczną, czyli powrót do wykonywania pierwszej części algorytmu.

Rys.12 Kontynuacja algorytmu odcieni szarości z informacją o jej ładowaniu po przetworzeniu powrotem do tablicy obrazu

prepare:	wykonanie przygotowania wartości dla obliczenia obrazu sepi.
	OPERACJE RĘCZNE SĄ TU WYKONYWANE NA USTAWIENIE POCAŁKOWYCH WARTOŚCI REJESTRÓW ORAZ SPRAWDZANIU W PĘTLI CZY DANA WARTOŚĆ SPENIA PODANE INSTRUKCJE WARUNKOWE.
	TAK TAK W WZGLĘDNYCH POZIOM RĘCZNYCH WYKONYWANA ITERACJA PĘTLI O 4. WARTOŚCI.
rci, rdi	pętla przygotowująca do wypełnienia obrazu nadający efekt sepi. Przypisywanie wartości początkowych dla danych rejestrów.
rci, rdi	przepisanie do rdi początku adresu obrazu.
rci, rdi	indeksowanie początku zliczania elementów tablicy związanej z podziałem na wątki poprzez dodanie do rejestru rdi wartości początkowej tablicy rejestru r11.
rci, 16	załadowanie do rcx listy bajtów do przetworzenia.
rci, rdi	odjęcie przesunięcia (offsetu) związanego z podziałem na wątki z rejestru rcx poprzez odjęcie końca przedziału zliczania z rejestru r11.
r11, 255	załadowanie do rejestru r12 wartości 255, aby sprawdzić czy nie przekroczono maksymalnej dozwolonej wartości pikseli.
r11, r11	zamaskowanie wartości rejestru r12 do rejestru r11, w celu późniejszego wykorzystania rejestru r12 ( wartości 255).
r11, rcx	odjęcie od rejestru 11 wartości spód rejestru rcx. 255 - x, gdzie x to wartość danej przez użytkownika efektu wypełnienia. wykorzystywane do sprawdzenia czy wartość wypełnienia sepi nie przekracza 255.
r11, rcx	kolejne odjęcie do rejestru r11 wartości spód rejestru rcx wynikającego z algorytmu sepi.
rcx, r11	porównanie rejestru rcx z r11 czy wartość wypełnienia nie przekracza maksymalnej wartości 255.
max_red_first	skok w przypadku przekroczenia maksymalnej wartości 255.
rcx, 0	jeżeli nie przekroczono maksymalnej wartości dodajemy wartość wypełnienia do akumulatora rcx dla czerwonej składowej pikseli.
rcx, 0	drugi rcx dodajemy do akumulatora wartości wypełnienia, ponieważ wynika to z algorytmu.
tone_loop	skok bezwarunkowy do etykiety kontynuującej wypełnienia pikseli, poprzez pobranie kolejnego pikseli.
max_red_first:	etykieta w przypadku przekroczenia maksymalnej wartości 255.
rci, r11	załadowanie do składowej czerwonej pikseli wartości maksymalnej 255, ponieważ wyliczaliśmy wypełnienia przekroczył maksymalną wartość 255.

Rys.13 Zakończenie procesu I części algorytmu i przygotowanie odpowiednich rejestrów i ustawienie początkowych wartości, aby dokonać koloryzacji obrazu



```

tone_loop:
    ; pobiera wynikowy wypełnienia pikseli dla obrazu przez SMI1 z jego kontenera i podaje go do
    ; przeniesienia wartości składowej niebieskiej pikselu do akumulatora al
    mov al, [0]
    ; porównania wartości składowej niebieskiej pikseli tablicy obrazu z wartością parametru głębi obrazu
    cmp rax, r1d
    ; jeżeli wartość składowej niebieskiej jest mniejsza niż parametru głębi to ustaw wartość składowej na 0 poprzez skok do etykiety min_blue
    jle min_blue
    ; wartość składową jest większa niż parametru głębi, więc wartość składowej niebieskiej pikselu to jej wartość pomniejszona o podany parameter głębi. Utrzymamy ją poprzez odejęcie jej od parametru głębi
    sub continue_blue, rax
    ; skok do etykiety, ustawiającej obliczoną wartość składowej niebieskiej w tablicy obrazu oraz pobierającą nową wartość do akumulatora do wykonania przez pozostałe instrukcje procedury
    jmp continue_blue
min_blue:
    ; wyłazła min_blue, w której miejscu ustawiona wartość w akumulatorze, która wynika z algorytmu głębi. Następną instrukcją przypiszemy nową wartość nowej składowej niebieskiej obrazu do akumulatora
    mov rax, rax
    ; wyłazła z min_blue, więc ustawiamy wartość składowych niebieskich pikseli dla obrazu, aby otrzymać efekt głębi SMI1
    continue_blue:
    ; przeniesienie wartości przetworzonej składowej niebieskiej pikseli z akumulatora do tablicy obrazu
    mov al, [0+1]
    ; przeniesienie do akumulatora al wartości obrazu
    ; przeniesienie do r15 wartości 255, w celu sprawdzenia czy nie przekroczone maksymalnej dopuszczalnej wartości
    mov r15, r1d
    ; odejęcie od rejestru r15 wartości skok rejestru rax. 255 - x, gdzie x to wartość składowej przez użytkownika efektu wypełnienia
    sub r15, rax
    ; porównania z akumulatorem rax czy przekroczone maksymalną wartość 255
    cmp max_green, rax
    ; skok w przypadku przekroczenia maksymalnej wartości 255
    jg max_green
    ; jeżeli nie przekroczone maksymalnej wartości dodajemy wartość wypełnienia do akumulatora rax dla składowej składowej pikseli
    ; skok do etykiety kontynuującej wypełnienia składowej zielonej pikseli
    jmp continue_green
max_green:
    ; etykieta następująca maksymalną wartość 255 dla koloru zielonego, ponieważ przekroczył maksymalną dopuszczalną wartość
    ; wstawiamy do akumulatora rax wartość 255 z rejestru r15 dla składowej zielonej pikseli, wynika to z algorytmu
    mov rax, r15
    ; etykieta do kontynuowania efektu wypełnienia i wstawienia wartości obliczonego wypełnienia do tablicy
    continue_green:
    ; przeniesienie wartości akumulatora al do adresu obrazu, gdzie akumulator zawiera obliczoną wartość wypełnienia
    mov al, [0+2]
    ; pobranie kolejnej wartości pikseli do akumulatora al, aby wykonać obliczenia algorytmu dla następnych pikseli
    ; przeniesienie do r15 wartości 255, aby sprawdzić czy nie przekroczone maksymalnej wartości
    mov r15, r1d
    ; odejęcie od rejestru r15 wartości skok rejestru rax. 255 - x, gdzie x to wartość składowej przez użytkownika efektu wypełnienia
    sub r15, rax
    ; kolejne odejęcie do rejestru r15 wartości skok rejestru rax wynikającego z algorytmu
    ; porównania z akumulatorem rax czy przekroczone maksymalną wartość 255
    cmp rax, r15
    ; skok w przypadku przekroczenia maksymalnej wartości 255
    jg max_red
    ; jeżeli nie przekroczone maksymalnej wartości dodajemy wartość wypełnienia do akumulatora rax wartości składowej czerwonej pikseli
    ; drugi raz dodajemy do akumulatora wartość wypełnienia
    ; skok bezwarunkowy do etykiety kontynuującej wypełnienia składowej czerwonej pikseli
    jmp continue_red
max_red:
    ; etykieta w przypadku przekroczenia maksymalnej wartości 255
    ; wstawiamy do akumulatora rax wartość 255 z rejestru r15 dla składowej czerwonej pikseli, ponieważ przekroczone maksymalną dopuszczalną wartość
    ; etykieta do kontynuowania efektu wypełnienia i wstawienia wartości obliczonego wypełnienia do tablicy
    continue_red:
    ; przeniesienie wartości akumulatora al do adresu obrazu, gdzie akumulator zawiera obliczoną wartość wypełnienia
    mov rdi, 4
    ; przesunięcie o 4 bity ( 1 piksel) do przodu wartości rejestru indeksowego, gdzie znajdują się adresy obrazu
    ; odejęcie do wartości rejestru zliczającego 4 ( 1 piksel), aby nie wyjść poza zakres tablicy
    dec rdi, 3
    ; sprawdzenia czy zakończono liczenie
    ; jeżeli wartość poniżej zera albo równa zero skoczy do etykiety koniec, zakończono zliczanie dla drugiej części algorytmu
    jle koniec
    ; skok bezwarunkowy do pętli zewnętrznej wykonującej główny proces wypełnienia obrazu, ponieważ nie zakończono przekształcania obrazu
    jmp tone_loop
koniec:
    ; etykieta, w której kończymy wykonywanie naszego przetwarzania obrazu
    ; przywrócenie początkowej wartości adresu tablicy obrazu do rejestru zliczeniowego, aby nie narazić się na nieodpowiedni odczyt wartości z rejestru zliczeniowego przez błąd wysokiego poziomu
    ; zakończenie algorytmu i powrotem wartości stanu stosu
    mov rbp, rbp
    ; odczytanie rejestru rbp ze stosu
    mov r14, rbp
    ; odczytanie rejestru r14 ze stosu
    mov r13, rbp
    ; odczytanie rejestru r13 ze stosu
    mov r12, rbp
    ; odczytanie rejestru r12 ze stosu
    mov r11, rbp
    ; odczytanie rejestru r11 ze stosu
    ; powrót z głównej procedury SMI1
    ; koniec procedury wykonywania SMI1
    sepi endb

```

Rys.14 Proces sprawdzania skrajnych wartości z przedziału w przypadku przekroczenia, którejś z wartości przedziału  $<0,255>$ . Skoki do odpowiednich etykiet reprezentują instrukcje warunkowe z języka wysokiego poziomu

W programie wykorzystano następujące etykiety :

- **average\_loop** -> Pełniąca funkcje pętli do przetworzenia obrazu do odcieni szarości, w tej pętli następuje wykorzystanie instrukcji SIMD, aby przyspieszyć działanie programu oraz jednocześnie przetworzyć 8 pikseli poprzez wykorzystanie całego rejestru ymm7 (256 bitów);
- **prepare** -> Ustawiająca początkowe wartości w rejestrach, aby móc wykorzystać je do II części algorytmu, a mianowicie koloryzacji do Sepii;
- **tone\_loop** -> Pełniąca funkcję pętli do przetworzenia obrazu do efektu Sepii z wykorzystaniem pośrednich etykiet takich jak m.in. max\_red, max\_green, max\_blue;
- **max\_red\_first** -> etykieta sprawdzająca pierwszą wartość piksela, aby można było wejść do pętli tone\_loop;
- **max\_red, max\_green, max\_blue** -> etykiety do ustawienia wartości 255 dla poszczególnych składowych piksela;
- **continue\_blue, continue\_green, continue\_red** -> etykiety ustawiające odpowiednie wartości zgodnie z algorytmem działania opisanym w podpunkcie „Algorytm Sepii”
- **min\_blue** -> ustawienie wartości minimalnej 0, w przypadku przekroczenia wartości minimalnej zgodnie z algorytmem,
- **koniec** -> etykieta do zakończenia działania programu poprzez odczytanie zapamiętanych początkowo wartości ze stosu.

Początkowo w programie są zapamiętywane na stos rejestry:

- rdi,
- r12,
- r13,
- r14,
- rbx,
- rbp.

Zgodnie z dokumentacją na stronie MSDN zostały muszą one zostać zapamiętane w przypadku wykorzystywania powyższych rejestrów.

Początkowo w głównej procedurze zapamiętywane są na stos wartości odpowiednich rejestrów z powodów konieczności poprawności działania programu. Kolejnym krokiem jest pobranie wartości z parametrów przekazywanych do procedury. Jest ich 5 co powoduję, że należało wykorzystać stos do zapamiętania jednego parametru:

- rcx -> wskaźnik na tablice bajtów,
- rdx -> start przedziału tablicy dla wątku,
- r8 -> koniec przedziału tablicy dla wątku,
- r9 -> wartość intensywności sepii,
- qword ptr [rbp + 48] -> wartość głębi sepii,

Również w trakcie działania programu tylko do przesuwania odpowiednich wartości korzystam z rejestru, w którym umieszczam stałą wartość w celu otrzymania „maski” wartości dla ładowania odpowiednich wartości do rejestrów.

Są to maski:

- color\_R\_mask -> Ustawia wartość FF na pozycji wszystkich składowych czerwonych piksela.
- divider\_value -> Ustawia wartości stałej 3 na pozycji pierwszej składowej czerwonej piksela,
- alpha\_mask -> Ustawia wartości FF na pozycji wszystkich składowych przezroczystości piksela,

Maska divider\_value jest ustawiona na pierwszej składowej czerwonej piksela, ponieważ w trakcie działania programu na tej pozycji zostanie dokonane dzielenie wektorowe sumowanych wszystkich składowych danego piksela. Maski znajdują się w rejestrze odpowiednio: ymm0, ymm1 oraz ymm2.

Sposób podejścia do ładowania wartości do rejestrów ymm (256 bitów) wymaga szerszego opisanie.

Oprócz wcześniej opisanego podejścia do ładowania stałych wartości, również będą tutaj przenoszone wartości z tablicy bajtów za pomocą wskaźnika w rejestrze rdi. Na początku wartości są pobierane z rejestru xmm10, a następnie z xmm9. Następnie z zapamiętaniem, iż rejestr xmm10 przechowuje starsze wartości przesunięte o 16 elementów do tyłu, oba rejestry xmm10 oraz xmm9 są ładowane do rejestru ymm4. Odpowiednio xmm10 do górnej części rejestru, a xmm9 do dolnej części rejestru. Następnie maskowane są kanały przezroczystości z wykorzystaniem maski alpha\_mask do rejestru ymm4. W taki sposób otrzymałem w rejestrze ymm6 tylko składowe R,G,B, które należy odpowiednio wyodrębnić, aby uzyskać średnią arytmetyczną.

Kolejnym krokiem jest maskowanie odpowiednich wartości składowych R,G,B z wykorzystaniem color\_R\_mask do trzech oddzielnych rejestrów:

- ymm4 -> przechowuje wartości składowych R piksela,
- ymm7 -> przechowuje wartości składowych G piksela,
- ymm8 -> przechowuje wartości składowych B piksela.

Natomiast samo otrzymywanie podanych wartości polegało na odpowiednim logicznym przesunięciu o dwie wartości zawartości rejestru color\_R\_mask, aby maskowała inne składowe, a następnie operacji logicznej AND i przepisaniu do odpowiedniego rejestru opisanego powyżej. Należało, również zawartości rejestru do siebie dodać za pomocą odpowiednich instrukcji wektorowych. Kolejnym etapem była zamiana otrzymanych wartości z dodawania na typ float, aby można było dokonać dzielenia wektorowego z wykorzystaniem instrukcji wektorowej vdivps oraz zawartości rejestru ymm3 (divider\_value). Potem z wykorzystaniem instrukcji vpor dodano na odpowiednią pozycję zapamiętane wcześniej kanały przezroczystości z rejestru ymm4 do otrzymanych wartości średniej arytmetycznej.

Pozostałe operacje są wykonywane zgodnie z algorytmem opisanym w podpunkcie „schemat blokowy algorytmu”. Jedynie po wykonaniu operacji odcieni szarości i przed wykonaniem koloryzacji obrazu należało w etykiecie „prepare” ustawić poprzednie wartości rejestru na pozycje początkowo, aby dokonać koloryzacji dla całej tablicy.



## Instrukcje wektorowe

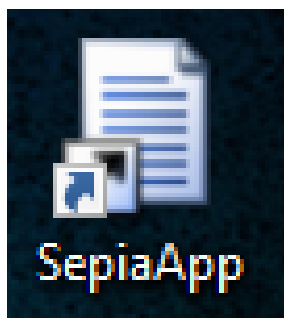
Zostaną przedstawione wykorzystywane instrukcje wektorowe z pominięciem ładowania wartości do i z rejestrów 128 bitowych (xmm) lub 256 bitowych (ymm) ze względu na trywialność ich działania.

Wykorzystano następujące istotne instrukcje wektorowe:

- `vdivps ymm7, ymm7, ymm2` -> instrukcja dzielenia równoległe (packed) wartości pojedynczej precyzji typu float rejestru ymm7 przez zawartość rejestru ymm2 o takim samym typie i zwrócenie zawartości do rejestru ymm7;
- `vpand ymm7, ymm7, ymm0` -> instrukcja bitowego iloczynu logicznego (AND) zawartości rejestru ymm7 i ymm0, a następnie wpisanie zawartości do rejestru ymm7;
- `vpor ymm7, ymm7, ymm5` -> instrukcja bitowej sumy logicznej (OR) zawartości rejestru ymm7 i ymm0, a następnie wpisanie zawartości do rejestru ymm7;
- `vpaddq ymm5, ymm4, ymm7` -> instrukcja dodawania równoległe wartości stałooprzecinkowych dword.

## Opis uruchamiania programu i jego testowania

Program uruchamiamy poprzez otwarcie pliku wykonywalnego „SepiaGUI.exe”. Jednocześnie w danym katalogu, w którym znajduje się plik SepiaGUI.exe muszą być umieszczone pliki: SepiaAsm.dll oraz SepiaDll.dll. Jednakże, jeśli użytkownik korzystał z instalatora oraz skryptu InstallerScript znajdującego się w katalogu Instalacja powinien być przeprowadzony przez proces instalacji oprogramowania w postaci okna instalacji programu SepiaApp. Wtedy w zależności od lokalizacji jaką użytkownik wybrał, przykładowo na pulpicie, powinien znaleźć się skrót do uruchomienia programu o nazwie SepiaApp.



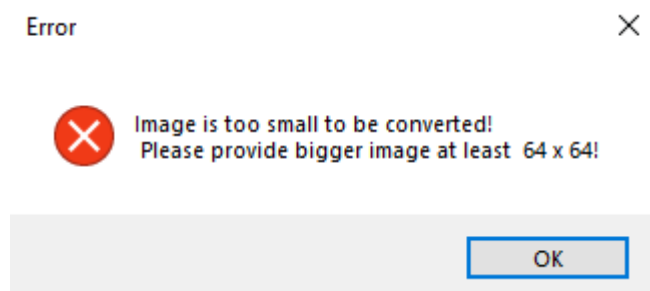
Rys.15 Wygląd ikony po instalacji programu SepiaApp

Należy pamiętać, iż dołączono również odpowiednie biblioteki, których na czystej maszynie wirtualnej Windows 10 v.1903 nie było i bez nich program nie mógł poprawnie działać. Jest to pakiet redystrybucyjny Microsoft Visual C++ procesora x64: vc\_redist.x64 oraz platformę programistyczną .NET Framework Microsoft w wersji 4.8 : NDP48-x86-x64-AIOS-ENU. Bez podanych pakietów program nie działał poprawnie. Użytkownik może je zainstalować ręcznie lub z wykorzystaniem skryptu InstallerScript.

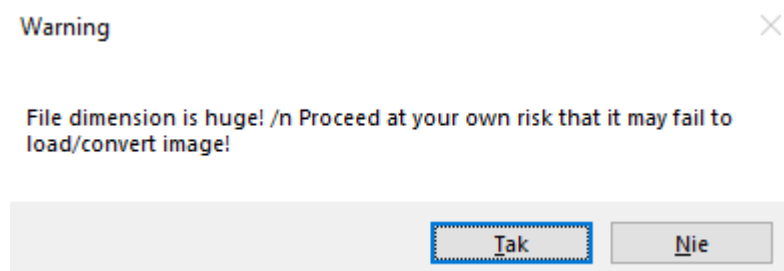
Program został przetestowany z wykorzystaniem poprawnych danych oraz również niepoprawnych danych.

Danymi poprawnymi są obrazy o rozszerzeniu .bmp, których ilość pikseli oraz głębia pikseli nie przekracza 150 milionów wartości. Czyli ilość pikseli na szerokość razy ilość pikseli na wysokość razy głębia w bitach przez osiem nie przekracza 150 milionów wartości. Przekroczenie tej wartości nie spowoduje nie wykonania konwersji, ale ostrzeże o możliwym niepowodzeniu konwersji, ponieważ ilość elementów do konwersji jest spora. Sytuacje wyjątkowe jakie mogą się pojawić zostały opisane w podpunkcie „Schemat blokowy programu”.

Jednocześnie przetestowano, co może się wydarzyć w przypadku gdy zmienimy format plików nie będących bitmapą na ten format i spróbujemy załadować plik do programu. Użytkownik otrzyma informacje o możliwym uszkodzonym pliku lub niepoprawnym formacie pliku. Jednocześnie pliki o formacie mniejszym niż 64x64 są traktowane za niepoprawne, dlatego w przypadku konwersji tak małego obrazu użytkownik otrzyma informacje o tym, iż obrazek powinien być większy. Testowałem również program dla dużych bitmap. Przykładowo dla rozmiaru 50000 x 4000 udało mi się skonwertować bitmapę w programie. Obciążenie programu wynosiło wykorzystanie prawie 3 GB pamięci VRAM. Poniżej zostaną przedstawione prawie wszystkie sytuacje wyjątkowe jakie zostały opisane wcześniej. Nie zostanie przedstawiony przypadek o braku wymaganych bibliotek lub instrukcji procesora, jednak poniższa funkcjonalność działa. Ponieważ była przetestowana na komputerach laboratoryjnych podczas zajęć, nie posiadających odpowiedniej wersji .NET frameworka oraz nie obsługiwała wymaganych instrukcji procesora.



Rys.16 Error w przypadku za małego obrazka



Rys.17 Ostrzeżenie w przypadku dużego obrazka

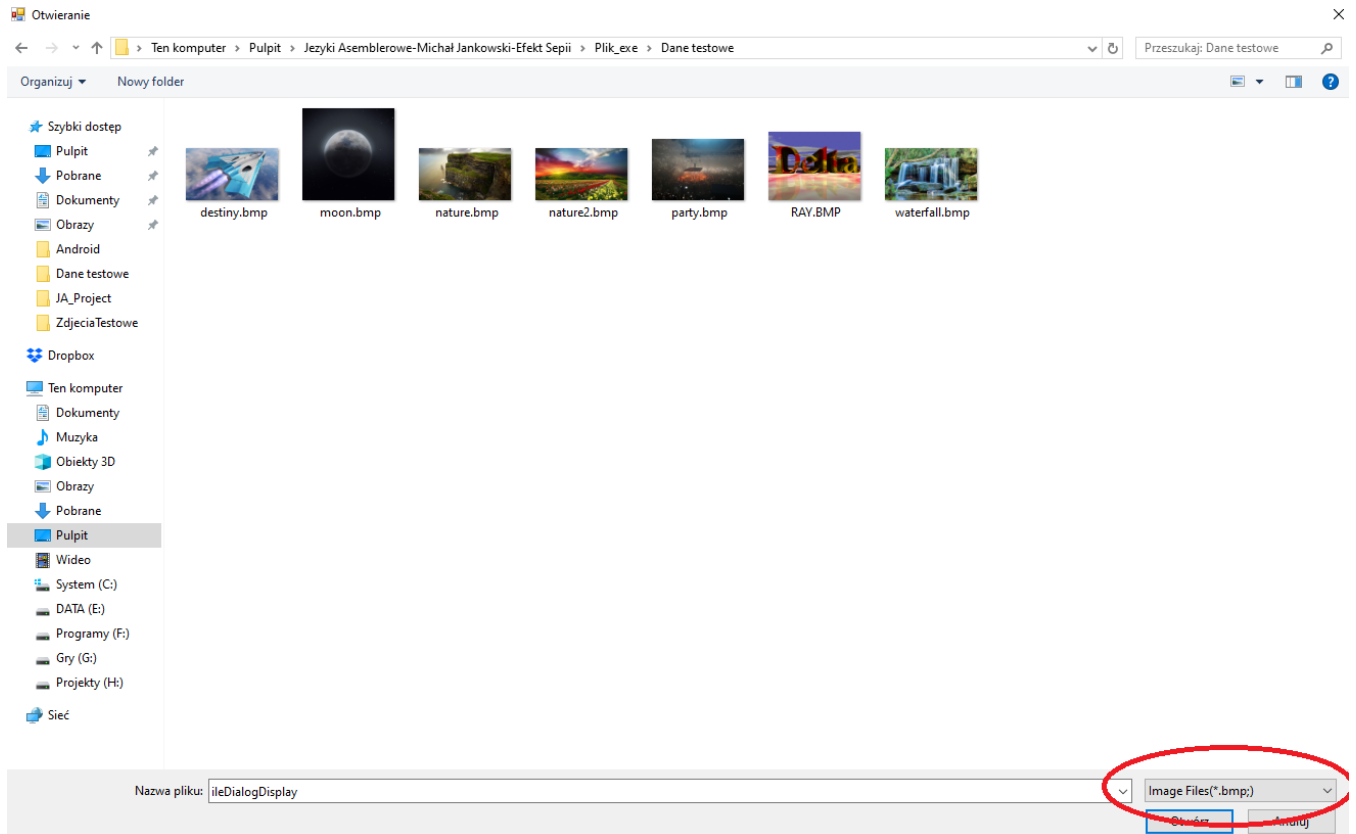
Error



Error while loading the image ! File was corrupted or of incorrect file extension!

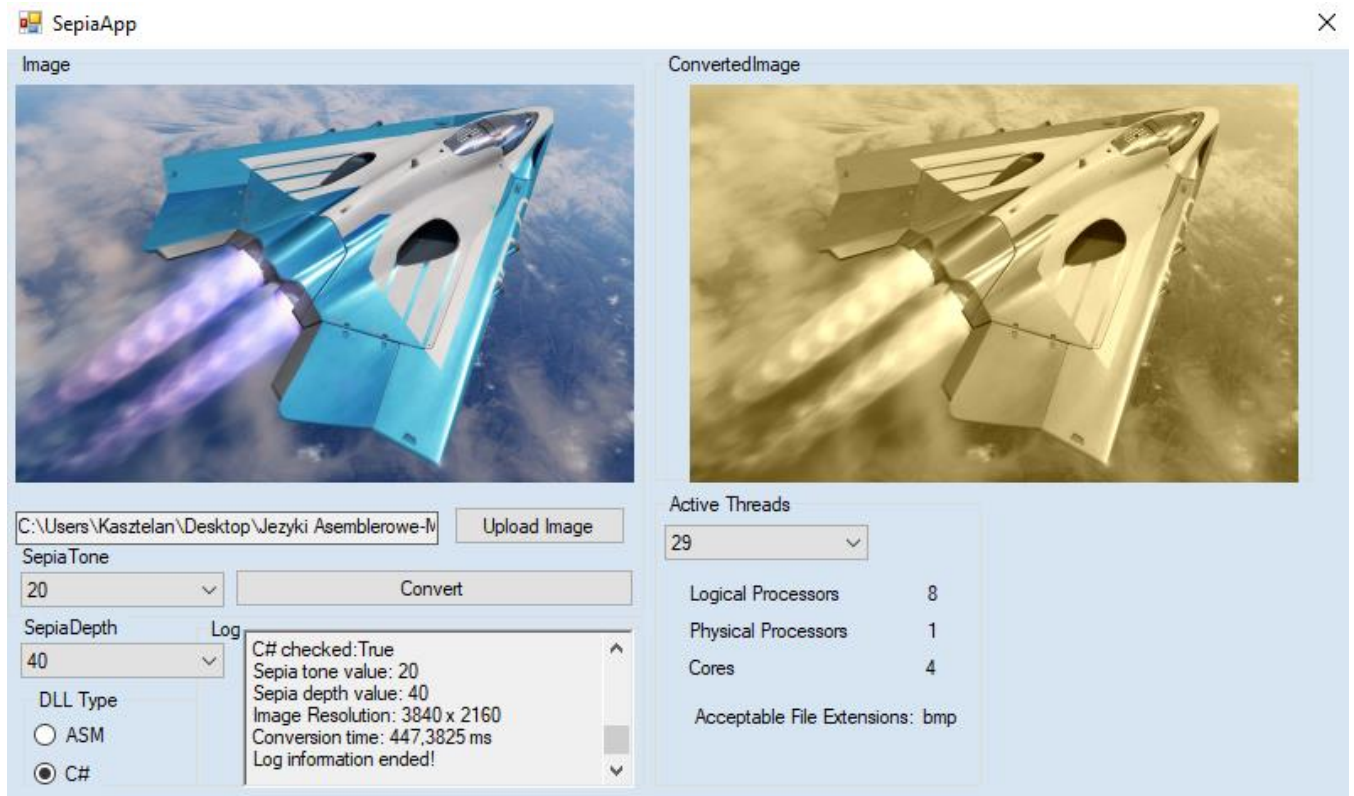
OK

Rys.17 Error w przypadku nieprawidłowego formatu



Rys.18 Informacja o możliwym typie danych przyjmowanych przez program

## Wyniki pomiarów czasy wykonania programu

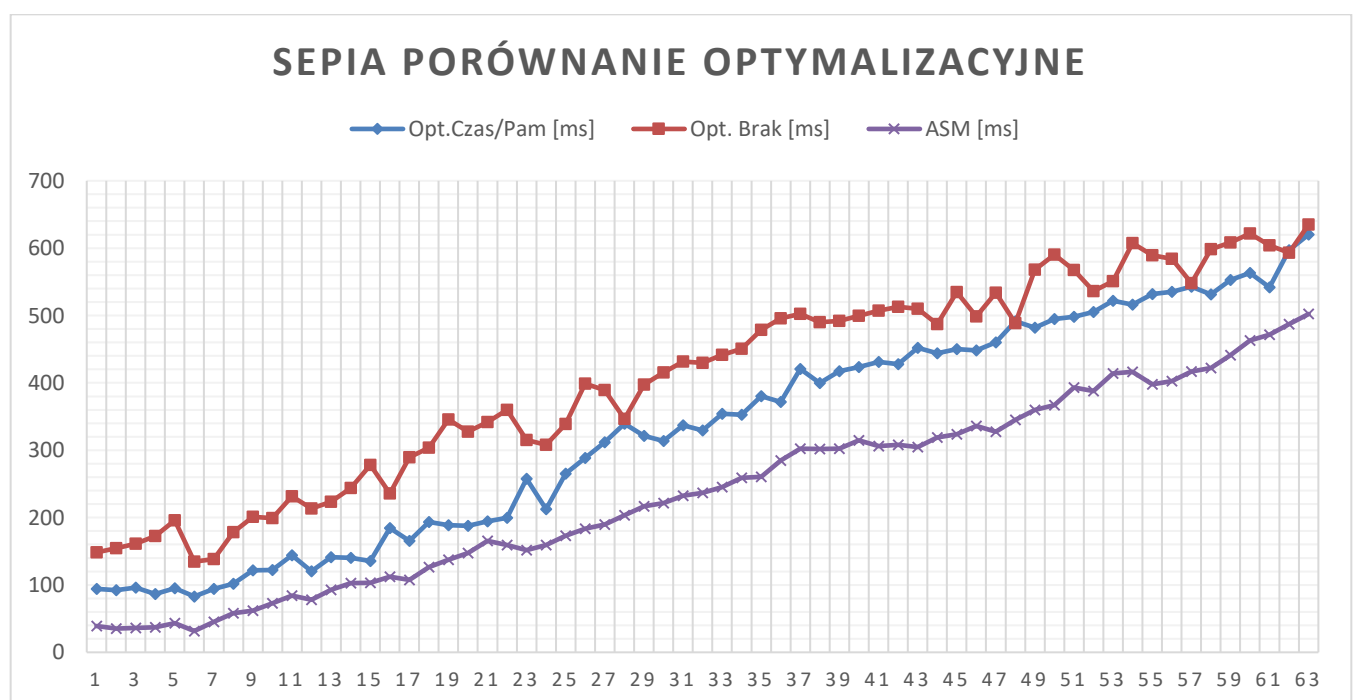


Rys.19 Przykładowy pomiar dla 29 wątków, C#

Program został przetestowany na obrazie 3840 x 2160 pikseli na komputerze o specyfikacji :

- Intel Core i5-8250u 1.6 GHz,
- NVIDIA GeForce MX 150,
- 8 GB DDR4 Ram,
- 1 TB HDD,
- Windows 10 v.1903.

Testowany obraz został dołączony do folderu dane testowe podczas oddawania implementacji projektu.



Rys.20 Wykres pomiarów optymalizacyjnych dla powyższego obrazka

Po przeanalizowaniu wyników można stwierdzić, iż Asembler wykonuje swoje działanie szybciej. Zgodnie z moimi przewidywaniami okazało się to prawdzie, ponieważ asembler wykorzystuje instrukcje SIMD, które znacząco przyspieszają działanie programu. Ponieważ w języku C# biblioteka posiada tylko jedną opcję „Optimize”, która odpowiada zarówno za optymalizację czasową oraz i pamięciową, ponieważ kompilator JIT ( Just-int-Time) stara się usuwać zbędne zmienne ze stosu wpływając na zużycie pamięci, ale zarówno na optymalizację czasową powodując konwersje funkcji na „inline functions” przyspieszając ich działanie.

Najgorzej wypadła wersja bez optymalizacji osiągając dla 64 wątków czas około 634 ms. Najlepszą wersją okazał się asembler dla 7 wątków osiągając czas około 32 ms.

Wątki	Opt.Czas/Pam [ms]	ASM [ms]	Opt. Brak [ms]
1	137,235	37,127	156,775
2	94,24	38,91	148,335
3	92,413	35,255	154,765
4	96,213	36,112	161,252
5	86,567	36,994	172,643
6	95,213	43,224	195,867
7	82,972	31,599	134,6346
8	94,125	45,013	138,43
9	101,9922	58,231	178,352
10	121,598	61,901	201,237
11	122,245	72,991	198,97
12	144,2456	84,241	231,6743
13	120,456	78,21412	213,572
14	141,2	92,633	223,454
15	140,244	102,633	243,765
16	135,564	103,0431	278,203
17	184,255	112,221	236,006
18	165,2455	107,4743	289,574
19	193,292	126,4363	303,784
20	188,765	137,4353	345,784
21	187,981	147,3252	327,358
22	194,2556	165,241	341,864
23	199,675	159,3234	359,975
24	257,4464	151,596	315,324
25	212,664	159,2422	308,235
26	265,446	172,943	339,013
27	288,563	183,435	398,742
28	312,035	189,5934	389,117
29	339,561	203,454	346,357
30	321,548	216,923	397,221
31	313,549	221,5558	415,668
32	337,256	232,543	431,389
33	329,539	236,832	429,557
34	354,2156	245,123	441,482
35	352,564	258,912	450,742

36	380,256	260,668	478,592
37	371,564	284,912	495,624
38	420,689	302,521	502,471
39	399,528	301,683	489,882
40	417,447	302,452	492,025
41	423,512	314,6442	499,662
42	431,0125	306,112	507,262
43	428,012	308,256	512,75
44	451,9201	304,548	510,225
45	444,1291	318,991	487,221
46	450,1248	323,691	534,742
47	448,2192	336,01	498,446
48	460,1923	327,562	533,557
49	491,2925	345,122	488,432
50	481,923	359,756	567,831
51	494,8271	367,012	590,221
52	497,9123	393,036	567,736
53	505,288	387,912	536,218
54	521,915	414,2131	550,996
55	516,0124	416,2813	607,221
56	531,928	397,975	589,331
57	535,235	402,6881	584,221
58	542,824	416,985	548,0043
59	531,285	421,912	598,182
60	552,7192	441,249	608,558
61	563,291	462,992	621,894
62	542,0192	471,7213	603,995
63	597,12	487,2412	592,963
64	620,2724	502,221	634,991

Tabela przedstawiająca pomiary dla obrazka 3840 x 2160, na którym dokonano testów optymalizacyjnych.

# Instrukcja obsługi programu

Aby użytkownik mógł zainstalować program należy mieć wszystkie wymagane biblioteki opisane w podpunkcie „Opis uruchamiania programu i jego testowanie”. Należy ręcznie lub z wykorzystaniem skryptu uruchomić proces instalacji.

W katalogu Plik\_exe znajduje się informacja w pliku README.txt o sposobie instalacji:  
„Folder Instalacja zawiera skrypt, który po kliknięciu instaluje niezbędny pakiet redystrybucyjny i .NET Frameworka oraz samą aplikację. Należy go zainstalować, aby poprawnie uruchomić program.”

Zawartość skryptu InstallerScript.bat:

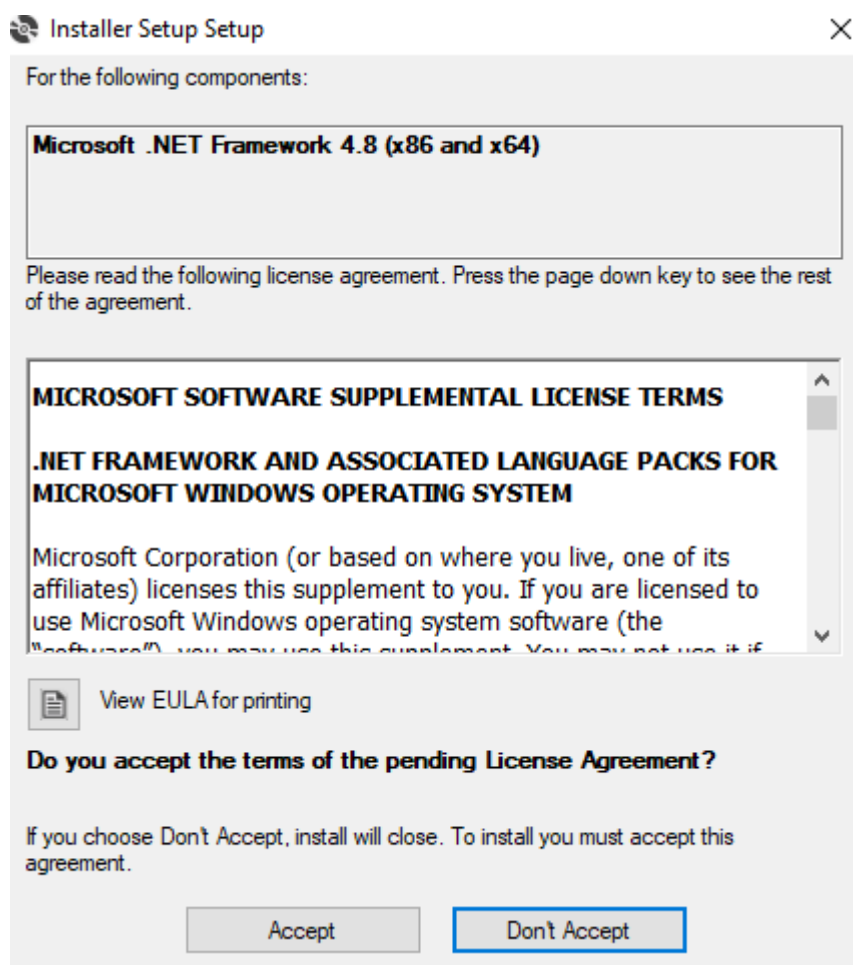
```
@ECHO OFF
```

```
Assets\Setup.exe
```

```
Assets\vc_redist.x64
```

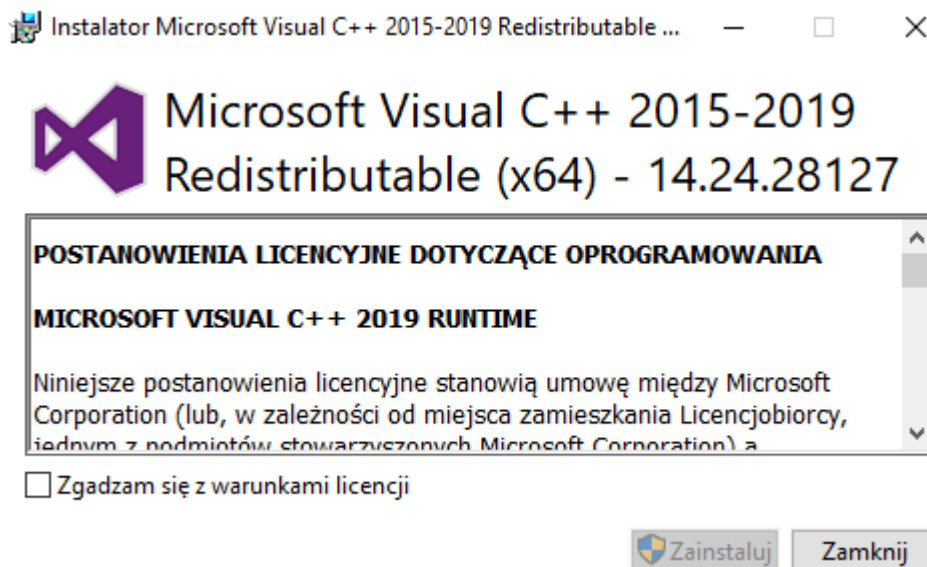
```
Assets\InstallerSetup.msi
```

Jak można zauważyć, musi on zawierać w tym samym katalogu, w którym znajduje się skrypt folder Assets. Posiada on plik Setup.exe oraz InstallerSetup.msi. Setup.exe instaluje pakiet redystrybucyjny oraz .NET Framework. Natomiast InstallerSetup.msi instaluje aplikację SepiaApp, która zawiera wszystkie dllki programu oraz jego plik .exe.

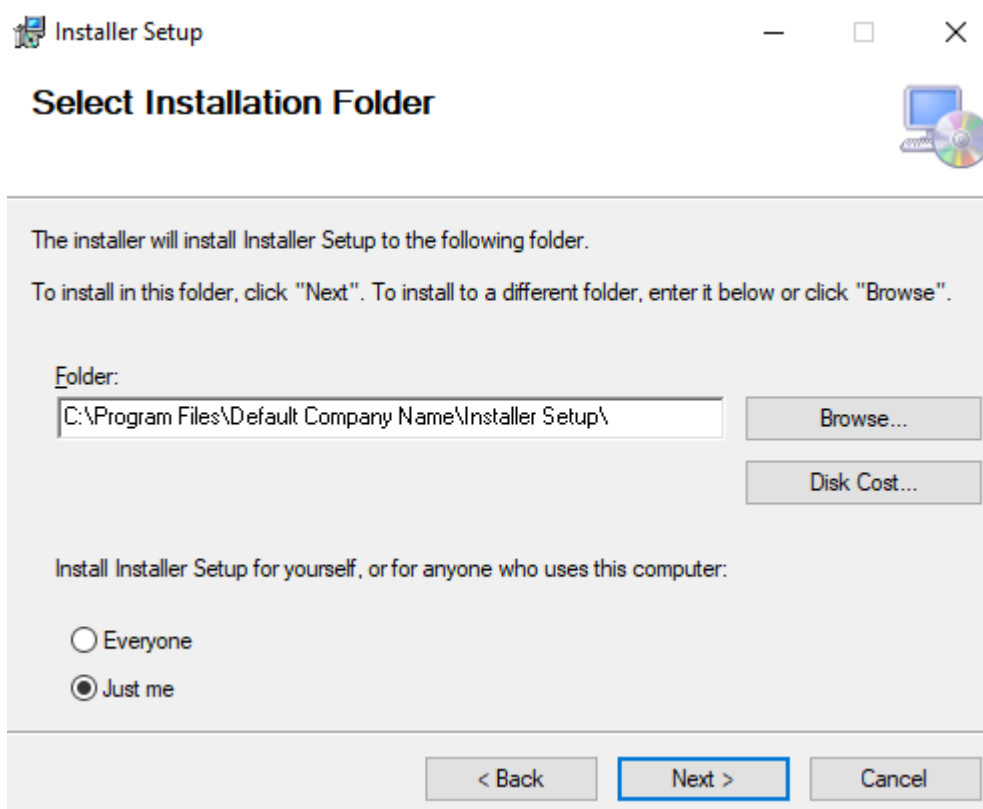


Rys. 21 Okno instalacji pakiet .NET Framework 4.8





Rys. 22 Okno instalacji pakietu redystrybucyjnego



Rys. 23 Okno instalacji aplikacji SepiaApp

Po zainstalowaniu wszystkich niezbędnych pakietów program powinien pomyślnie się uruchomić. Okno działającego programu znajduje się na Rys.19.

Użytkownik może następnie swobodnie korzystać z aplikacji pod warunkiem posiadania odpowiednich instrukcji AVX oraz SSE procesora. Obsługa programu jest prostolinijna, ponieważ użytkownik nie posiada zbyt wielu opcji wyboru właściwości oraz współczynników. W większości sposób obsługi gotowego programu został opisany w podpunkcie „Opis uruchamiania programu i jego testowanie” oraz „Opis programu w języku wysokiego poziomu”. Użytkownik musi pamiętać o ograniczeniu obrazów do pliku bitmap, a konwersja obrazu bazuje na manipulacji dwoma rozwijanymi paskami z wartościami. Mianowicie: SepiaTone oraz SepiaDepth. Po ich wyborze oraz poprawnym



wyborze zdjęcia użytkownik może nacisnąć przycisk „Convert”, a następnie oczekiwać na konwersję obrazu i log informacji ile dany proces zajął poprzedzone dodatkową informacją o typie dllki oraz rozmiaru pliku. Pasek manipulacji wyborem Asm oraz C# jak i ilością wątków z punktu widzenia użytkownika będzie głównie wyznacznikiem szybkości wykonania konwersji, która będzie miała znaczenie przy dużym rozmiarowo pliku do konwersji.

## Wnioski

Projekt pozwolił mi na dogłębne zapoznanie się z działaniem MASM w wersji 64 bitowej poprzez wykorzystywanie instrukcji 64 bitowych oraz sposobu przyjmowania parametrów poprzez rejestry rcx, rdx, r8, r9. Nauczyłem się tworzyć biblioteki programu oraz łączyć wykorzystanie ich w innym języku programowania niż zostały one same stworzone. Nabyłem istotną wiedzę w programowaniu w języku C# oraz Asm. Sam projekt przypominał mi jak zachodzą procesy fotograficzne przetwarzania obrazów, ale również przypominał algorytmiczne zasady rozwiązywania zadań. Największą trudnością podczas tworzenia projektu było podłączeniu obu bibliotek programu i ich poprawna komunikacja z językiem wysokiego poziomu. Pod koniec mogę stwierdzić, iż program sprostał moim początkowym założeniom i działa zgodnie z moim zamyśleniem.

## Literatura

- <http://www.algorytm.org/przetwarzanie-obrazow/sepia.html>;
- <http://przemyslawczatrowski.com/2010/04/17/skala-szarosci-i-sepia/>;
- <https://www.dyclassroom.com/image-processing-project/how-to-convert-a-color-image-into-sepia-image>;
- <https://docs.microsoft.com/pl-pl/dotnet/api/system.threading?view=netframework-4.8>;
- <https://docs.microsoft.com/en-us/cpp/assembler/masm/masm-for-x64-ml64-exe?view=vs-2019>;
- <https://docs.microsoft.com/pl-pl/cpp/build/x64-calling-convention?view=vs-2019>;
- <https://support.microsoft.com/pl-pl/help/2977003/the-latest-supported-visual-c-downloads>;
- <https://docs.microsoft.com/pl-pl/dotnet/framework/winforms/>;