Silesian University
of Technology
Faculty of Automatic Control, Electronics and Computer Science
Department of Graphics, Computer Vision and Digital Systems

Rok akademicki	Rodzaj studiów*:	Przedmiot:	Grupa	Sekcja
2024/2025	**SSI**	**JA proj.**	**-**	**7**
Termin: (dzień, godzina)	**wtorek, 8:00**	**Prowadzący:**	**AO**	

Imię: Karol
Nazwisko: Ziaja
Email: kz306786@student.polsl.pl

Raport końcowy

Temat projektu:

Filtr dolnoprzepustowy dla sygnałów dźwiękowych

1. Opis

Celem projektu było stworzenie aplikacji, w której użytkownik będzie mógł nałożyć filtr na wskazany przez siebie plik dźwiękowy w formacie wav.
Filtr dolnoprzepustowy ma na celu wytłumić wysokie częstotliwości z sygnału akustycznego, przepuszczając jedynie częstotliwości niższe od zadanego progu.
Filtr został zaimplementowany za pomocą algorytmu FIR. Dla każdej próbki wyjściowej algorytm ten oblicza konwolucję z N ostatnich próbek wejściowych, mnożąc je przez odpowiednie współczynniki filtra.
Algorytm jest wykonywany w dwóch niezależnych bibliotekach DLL:

- C++
- ASM

Biblioteki mają identyczne zastosowanie, a zaimplementowane w nich procedury filtra zapewniają takie same dane wyjściowe. Po nałożeniu filtra w aplikacji wyświetlony zostaje wynik czasowy obrazujący czas wykonania algorytmu w wybranej przez użytkownika bibliotece DLL. Celem projektu jest pokazanie różnic w czasie wykonania algorytmów w każdej z bibliotek.

2. Założenia projektu

2.1. Projekt dotyczy architektury procesorów Intel X86/64

2.2. Projekt wykonywany jest w środowisku Visual Studio 2022

2.3. Założenie projektu w środowisku VS polega na utworzeniu powiązanych ze sobą projektów:

- Aplikacja interfejsu użytkownika napisana w języku wysokiego poziomu (C++ / C#). Musi zawierać graficzny interfejs pozwalający użytkownikowi na parametryzację aplikacji oraz wprowadzenie danych wejściowych.
- Biblioteki DLL wywoływanej dynamicznie z poziomu głównego programu i zawierającej funkcje realizujące algorytm w języku C++.
- Biblioteki DLL w języku asembler X64, o identycznym zastosowaniu.

2.4. Aplikacja interfejsu zawiera elementy:

- Opcję wyboru w której bibliotece należy wykonać algorytm.
- Wskaźnik czasu wykonania funkcji bibliotecznej.

2.5. Biblioteka w C++ jest uruchamiana w trybie Release, z włączoną optymalizacją.

2.6. Aplikacja musi wykorzystywać wielowątkowość, a użytkownik będzie mógł wybrać ilość wątków – od 1 do 64.

2.7. Procedura w języku asemblera musi wykorzystywać instrukcje wektorowe (SSE / AVX).

3. Opis kodu

3.1. Wytłumaczenie zasady działania algorytmu FIR.

- Algorytm operuje na liczbach zmiennoprzecinkowych. Aby uzyskać takie dane, należy przeprowadzić konwersje z pliku wav na tablicę floatów przed przekazaniem do procedury algorytmu. Jest to realizowane za pomocą metod klasy konwertującej w języku wysokiego poziomu (C#). Po wykonaniu algorytmu plik jest odtwarzany, tak aby ponownie uzyskać plik dźwiękowy.
- Funkcja realizująca algorytm przyjmuje 5 argumentów:
  - Wskaźnik do tablicy wejściowej.
  - Wskaźnik do tablicy wyjściowej.
  - Wskaźnik do tablicy współczynników.
  - Długość tablicy wejściowej (taka sama jak wyjściowej).
  - Długość tablicy współczynników.
- Dla każdej próbki wyjściowej algorytm wykonuje konwolucję z N próbek wejściowych. Wagami (b_i) są współczynniki, generowane w języku wysokiego poziomu.
  Są one charakterystyczne dla implementowanego filtra, a ich ilość jest podawana przez użytkownika w interfejsie graficznym.

3.2. Opis matematyczny

$$y[n] = \sum_{i=0}^{N} b_i \cdot x[n-i]$$

$$y[n] = b_0 x[n] + b_1 x[n-1] + \ldots + b_N x[n-N]$$

3.3. Kod biblioteki asemblerowej wraz z komentarzami:

```
.code

; Definicja procedury ProcessFIRFilter
ProcessFIRFilter proc
    push rbp                    ; Zachowanie wartości bazowego wskaźnika stosu
    mov rbp, rsp                ; Ustawienie nowego bazowego wskaźnika stosu
    jmp _start                  ; Przejście do etykiety _start

_start:
    ; Opis argumentów przekazywanych do procedury:
    ; rcx = wskaźnik do tablicy wejściowej (input array)
    ; rdx = wskaźnik do tablicy wyjściowej (output array)
    ; r8  = wskaźnik do tablicy współczynników (coefficients array)
    ; r9  = długość tablicy wyjściowej (outputLength)
    ; [rsp + 48] = długość tablicy współczynników (coefficientsLength)
    ; r10 = długość tablicy współczynników (przechowywana w r10 dla wydajności)

    mov r10, [rsp + 48]         ; Pobranie długości tablicy współczynników z stosu

    mov rax, r10
    dec rax                     ; rax = coefficientsLength - 1

    ; Inicjalizacja indeksu dla tablicy wyjściowej
    mov r11, 0                  ; r11 = indeks aktualny dla output array

loop_n:
    cmp r11, r9                 ; Porównanie bieżącego indeksu z outputLength
    jge end_loop_n              ; Jeśli indeks >= outputLength, zakończ pętlę zewnętrzną

    ; Inicjalizacja indeksu dla tablicy współczynników
    mov r12, 0                  ; r12 = indeks aktualny dla coefficients array

    cmp r11, rax                ; Sprawdzenie, czy r11 < coefficientsLength - 1
    jl skip_coef                ; Jeśli tak, przejdź do pomijania obliczeń współczynników

    ; Czyszczenie rejestrów YMM, aby przygotować je do akumulacji wyników
```

```
37.     vxorps ymm0, ymm0, ymm0      ; Zerowanie ymm0
38.     vxorps ymm4, ymm4, ymm4      ; Zerowanie ymm4
39.     vxorps ymm5, ymm5, ymm5      ; Zerowanie ymm5
40.     vxorps ymm6, ymm6, ymm6      ; Zerowanie ymm6
41.
42. loop_k:
43.     cmp r12, r10                 ; Porównanie indeksu współczynników z coefficientsLength
44.     jge end_loop_k               ; Jeśli indeks >= coefficientsLength, zakończ pętlę wewnętrzną
45.
46.     mov r13, r11                 ; Kopiowanie bieżącego indeksu wyjściowego do r13
47.     sub r13, r12                 ; Obliczenie odpowiedniego indeksu wejściowego
48.     shl r13, 2                   ; Przemnożenie indeksu przez 4 (rozmiar float) dla obliczeń adresów
49.
50.     cmp r13, 0                   ; Sprawdzenie, czy obliczony indeks jest nieujemny
51.     jl end_loop_k                ; Jeśli indeks < 0, pomiń tę iterację
52.
53.     sub r13, 28                  ; Korekta indeksu wejściowego (offset)
54.
55.     ; Ładowanie danych wejściowych i współczynników do rejestrów YMM
56.     vmovups ymm5, YMMWORD PTR [rcx + r13]      ; Ładowanie 8 floatów z input array
57.     vmovups ymm6, YMMWORD PTR [r8 + r12 * 4]   ; Ładowanie 8 floatów z coefficients array
58.
59.     add r12, 8                   ; Zwiększenie indeksu współczynników o 8 (przetwarzanie wektorowe)
60.     jmp multiply_and_add         ; Przejście do etapu mnożenia i akumulacji
61.
62. multiply_and_add:
63.     ; Wykonanie operacji mnożenia i dodania dla wektorów
64.     vfmadd231ps ymm4, ymm5, ymm6  ; ymm4 += ymm5 * ymm6
65.     jmp loop_k                    ; Kontynuacja pętli wewnętrznej
66.
67. end_loop_k:
68.     ; Redukcja YMM4 do pojedynczej wartości float
69.     vxorps xmm0, xmm0, xmm0                 ; Zerowanie rejestru xmm0
70.     vextractf128 xmm0, ymm4, 1               ; Wyodrębnienie górnej części YMM4 do xmm0
71.     vaddps xmm4, xmm4, xmm0                  ; Dodanie górnej części do dolnej
72.     vhaddps xmm4, xmm4, xmm4                 ; Suma horyzontalna 8 floatów do 4
73.     vhaddps xmm4, xmm4, xmm4                 ; Suma horyzontalna 4 floatów do 2
74.     vmovss DWORD PTR [rdx + r11 * 4], xmm4    ; Przechowanie wyniku jako pojedynczego float w output
array
75.
76.     ; Inkrementacja indeksu wyjściowego i kontynuacja pętli zewnętrznej
77.     inc r11
78.     jmp loop_n
79.
80. skip_coef:
81.     ; Jeżeli bieżący indeks wyjściowy < coefficientsLength - 1, pomiń obliczenia współczynników
82.     inc r11                      ; Inkrementacja indeksu wyjściowego
83.     jmp loop_n                   ; Kontynuacja pętli zewnętrznej
84.
85. end_loop_n:
86.     ; Zakończenie procedury FIR filter
87.     mov rsp, rbp                 ; Przywrócenie wskaźnika stosu
88.     pop rbp                      ; Przywrócenie bazowego wskaźnika stosu
89.     ret                          ; Powrót z procedury
90.
91. ProcessFIRFilter endp
92.
93. END
94.
```

3.4. Istotne instrukcje:

- Wczytywanie wektorowe (8 wartości na raz) do rejestrów. Wczytywane są zarówno współczynniki jak i próbki wejściowe.

  vmovups ymm5, YMMWORD PTR [rcx + r13] ; Ładowanie 8 floatów z input array
  *vmovups ymm6, YMMWORD PTR [r8 + r12 * 4] ; Ładowanie 8 floatów z coefficients array*

- Instrukcja mnożąca zawartość rejestru ze współczynnikami oraz rejestru z próbkami wejściowymi. Wynik operacji jest zapisywany do rejestru zawierającego skumulowane wyniki.

```
; Wykonanie operacji mnożenia i dodania dla wektorów
vfmadd231ps ymm4, ymm5, ymm6  ; ymm4 += ymm5 * ymm6
```

- Dodawanie horyzontalne rejestru ze skumulowanymi wynikami. Celem jest uzyskanie jednej wartości zmiennoprzecinkowej jako wyniku aktualnej próbki wyjściowej.

```
;Redukcja YMM4 do pojedynczej wartości float
vxorps xmm0, xmm0, xmm0          ; Zerowanie rejestru xmm0
vextractf128 xmm0, ymm4, 1       ; Wyodrębnienie górnej części YMM4 do xmm0
vaddps xmm4, xmm4, xmm0           ; Dodanie górnej części do dolnej
vhaddps xmm4, xmm4, xmm4          ; Suma horyzontalna 8 floatów do 4
vhaddps xmm4, xmm4, xmm4          ; Suma horyzontalna 4 floatów do 2
```

- Zapis zsumowanej wartości do próbki wyjściowej

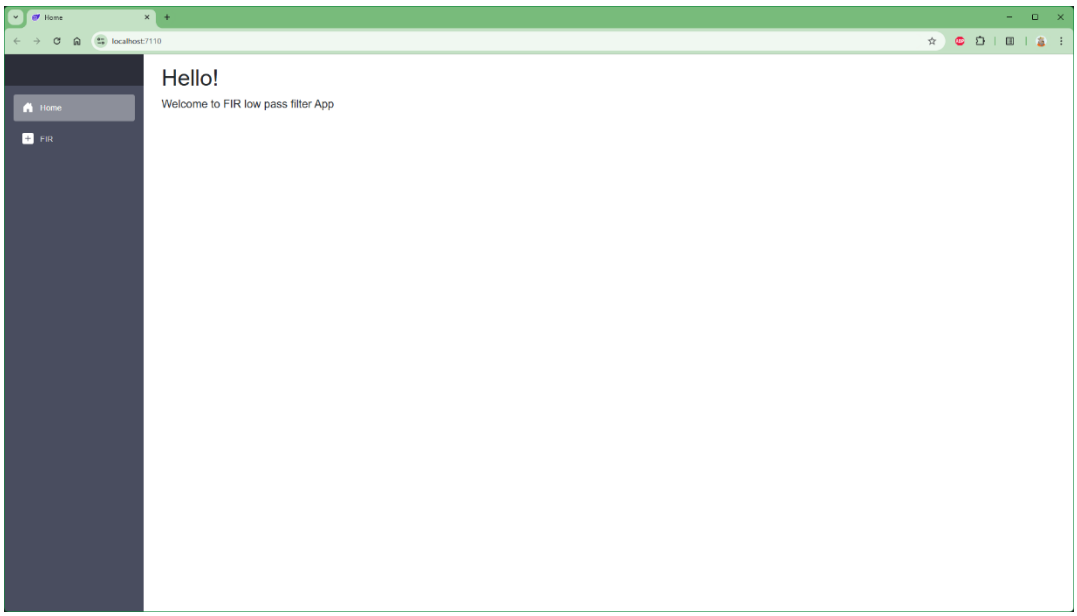
```
vmovss DWORD PTR [rdx + r11 * 4], xmm4    ; Przechowanie wyniku jako pojedynczego float w output array
```

3.5. Odpowiadający kod biblioteki w C++:

```
 1. extern "C" __declspec(dllexport) void ProcessFIRFilter(float* input, float* output, float*
coefficients, int inputLength, int coefficientsLength) {
 2.     for (int n = 0; n < inputLength; ++n) {
 3.         if (n < coefficientsLength - 1) {
 4.             output[n] = 0;
 5.             continue;
 6.         }
 7.
 8.         for (int k = 0; k < coefficientsLength; ++k) {
 9.             if (n - k >= 0) {
10.                 output[n] += coefficients[coefficientsLength - 1 - k] * input[n - k];
11.             }
12.             else {
13.                 break;
14.             }
15.         }
16.     }
17. }
18.
```

4. Interfejs użytkownika

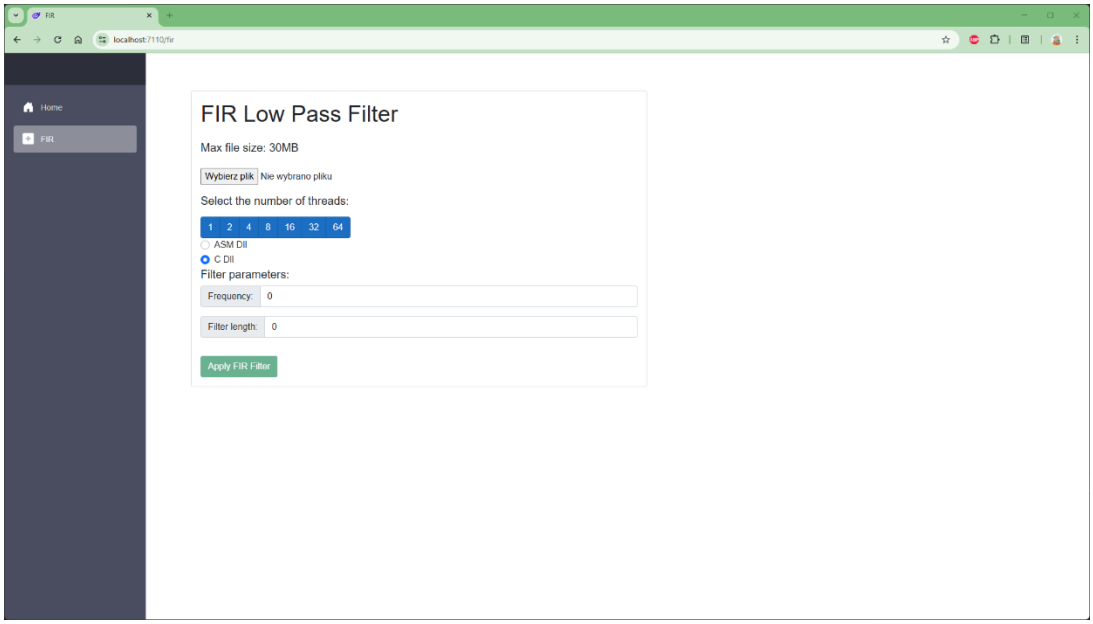
4.1. Aplikacja została wykonana w technologii ASP.NET Core, co oznacza, że jest to aplikacja webowa.



Zrzut ekranu 1: Ekran startowy aplikacji

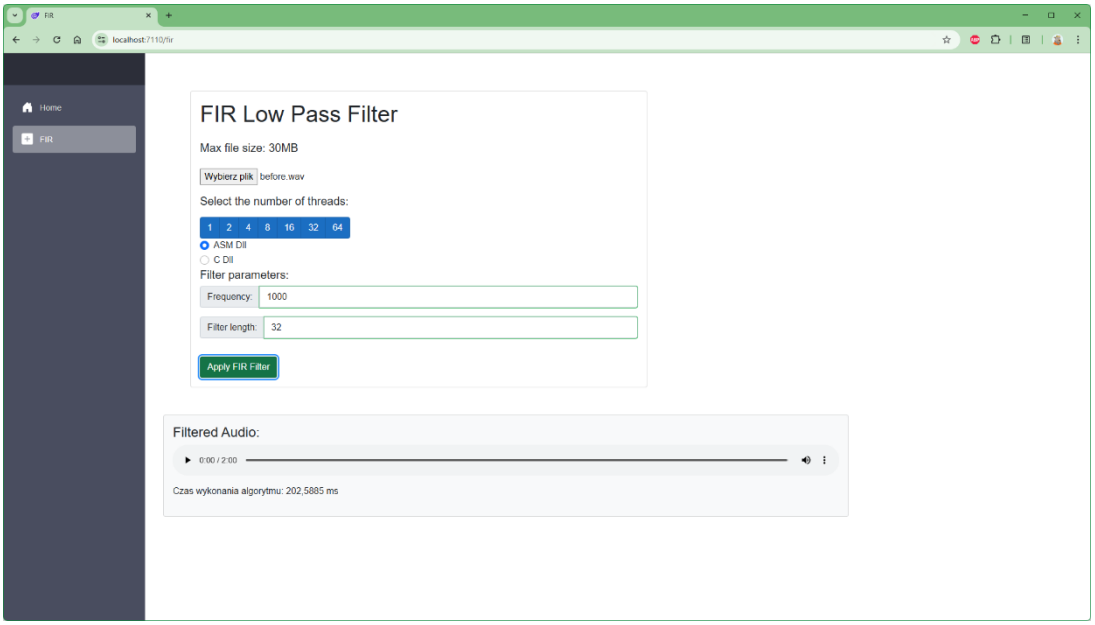
4.2. Interfejs posiada stronę umożliwiającą użytkownikowi:

- Wybranie pliku dźwiękowego.
- Podanie częstotliwości odcięcia oraz ilości współczynników filtra.
- Wybranie ilości wątków oraz docelowej biblioteki DLL.



Zrzut ekranu 2: Interfejs użytkownika aplikacji

4.3. Po wykonaniu algorytmu na stronie umieszczony zostanie plik dźwiękowy z możliwością odtworzenia, oraz czas wykonania funkcji bibliotecznej.



Zrzut ekranu 3: Wynik działania aplikacji

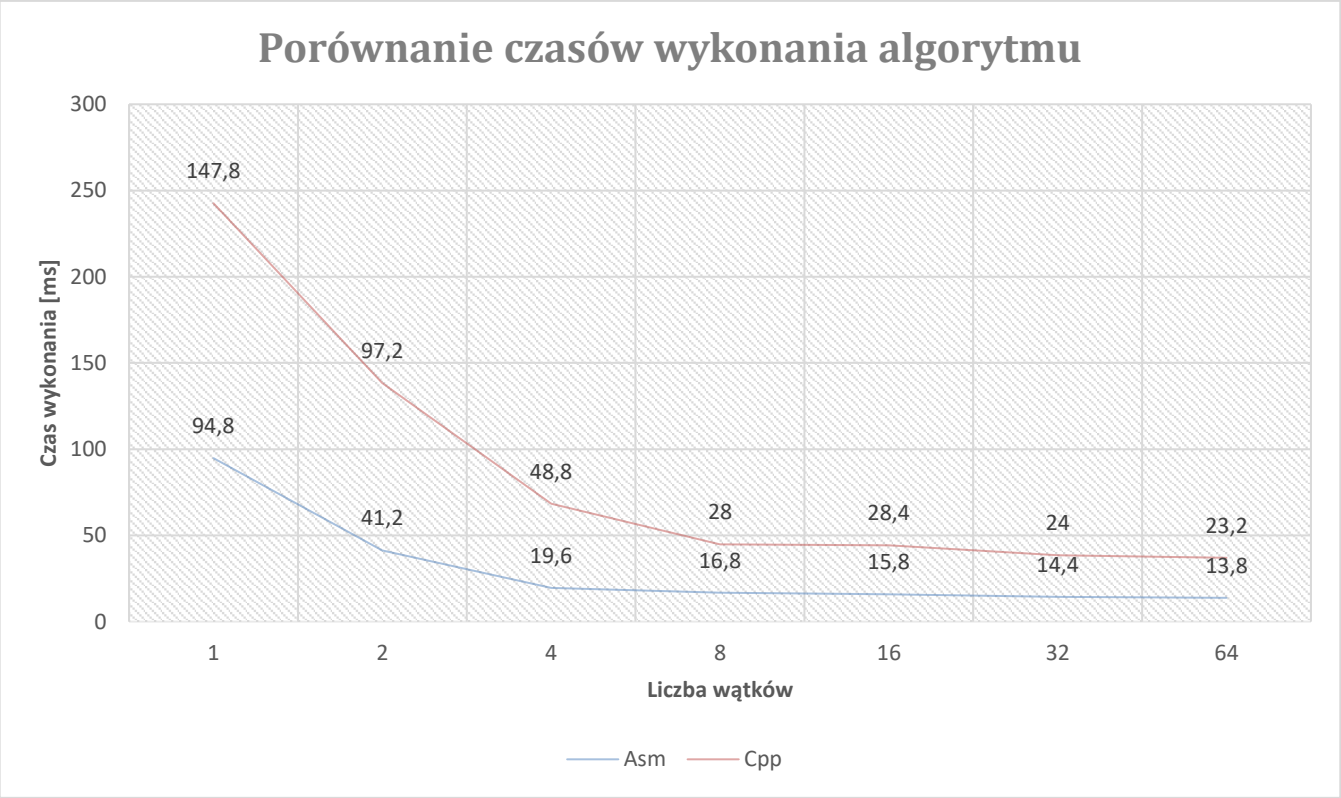
5. Wyniki czasowe

5.1. Dla każdej liczby wątków wykonano 5 pomiarów (nie wliczając pierwszego). Powtórzono to zarówno dla wykonania w C++ jak i w asemblerze. Każdy wynik czasowy to średnia wyciągnięta z 5 pomiarów.

5.2. Warunki

- C++ 20
- Włączony tryb Release z optymalizacją
- Wejściowy plik wav o rozmiarze 20MB
- Ilość współczynników – 16
- Częstotliwość odcięcia – 1000Hz (nie wpływa na wynik czasowy)

5.3. Dla wykonanych pomiarów uzyskano wyniki czasowe zaprezentowane na wykresie:



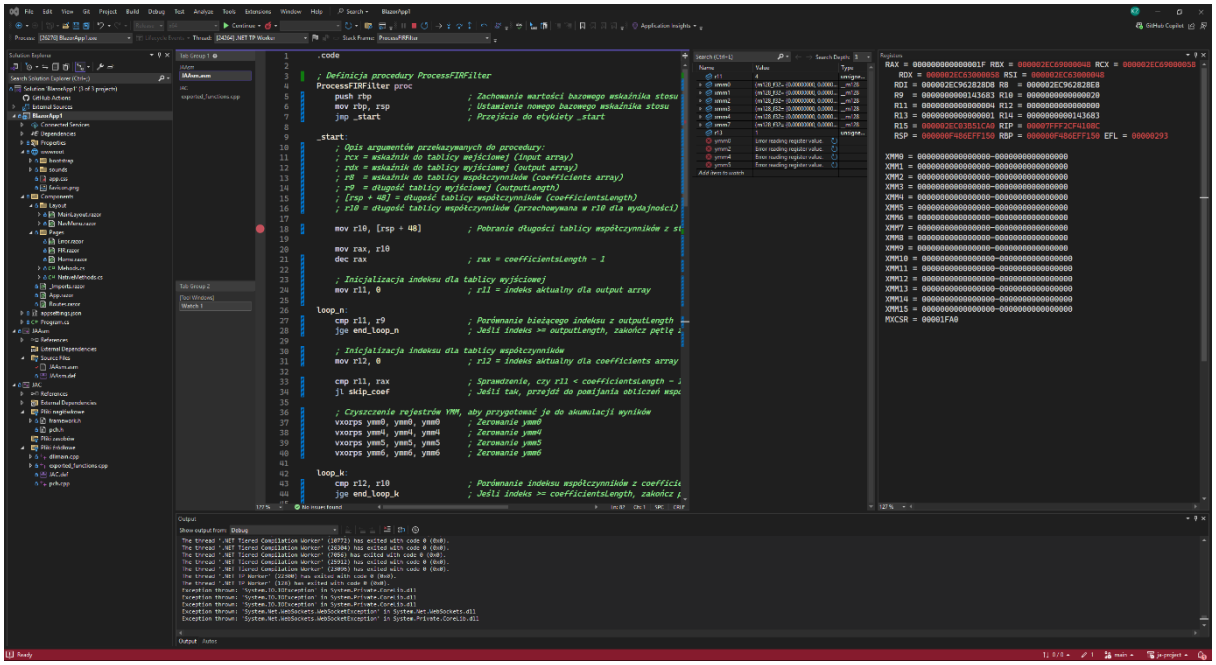
Wykres 1: Porównanie czasów wykonania algorytmu

5.4. Na powyższym wykresie zaobserwowano średnio dwukrotne przyspieszenie. Jest to jednak wynik uzyskany dla tych konkretnych warunków testowania. W przypadku większego pliku wejściowego, lub większej liczby współczynników dane mogą się różnić, a różnica między C++, a asemblerem może być bardziej zauważalna.

6. Opis testowania

6.1. Testowanie zostało przeprowadzone dla aplikacji skompilowanej w trybie Release dla architektury X64.

6.2. W trakcie tworzenia aplikacji możliwe było debugowanie kodu asemblerowego oraz wgląd do aktualnych rejestrów procesora. Możliwość debugowania była kluczowa dla dokładnego zrozumienia instrukcji działających na rejestrach AVX.



Zrzut ekranu 4: Debugowanie kodu asemblerowego

7. Wnioski

Analizując wyniki czasowe na wykresie, można dostrzec wyraźną różnicę czasową między kodem napisanym w C++, a tym w asemblerze. Wykonanie projektu pozwoliło zrozumieć, że mając większą kontrolę nad kodem jesteśmy w stanie wykonywać obliczenia na wielu próbkach na raz. Równoległość poziomu danych była tutaj kluczowa i niemożliwa do uzyskania w języku wysokiego poziomu. Projekt nauczył również wykorzystywania biblioteki dołączanej dynamicznie w celu zaimplementowania małego kawałku kodu, tudzież prostego algorytmu.
Dzięki połączniu tych dwóch rozwiązań, oraz wielowątkowości jesteśmy w stanie uzyskać zoptymalizować kod i uzyskać znacznie lepsze wyniki czasowe.