Министерство образования Республики Беларусь

Учреждение Образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра электронных вычислительных средств

Лабораторная работа № 4

«ПРОГРАММИРОВАНИЕ НА АССЕМБЛЕРЕ ПРОЦЕССОРА С

АРХИТЕКТУРОЙ INTEL x86»

|  |  |
| --- | --- |
| Проверил: | Выполнили: |
| Порхун М. И. | ст. гр. 850702 |
|  | Маковский Р. А. |
|  | Турко В. Д. |
|  |  |
|  |  |

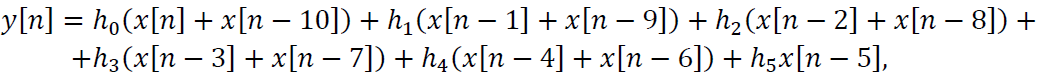
Минск 2020

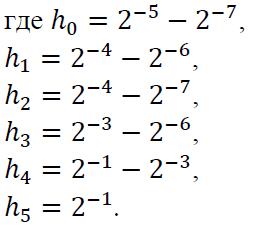
# ЦЕЛЬ РАБОТЫ

Научиться интегрировать ассемблерные вставки в высокоуровневый код; реализовать алгоритм цифровой обработки сигналов средствами ассемблера процессора с архитектурой Intel x86. Научиться оценивать быстродействие программ при помощи внутреннего счетчика тактов процессора.

# ЗАДАНИЕ. ВАРИАНТ 1

Необходимо описать цифровой фильтр двумя способами: на языке C, и с помощью ассемблерной вставки; оценить быстродействие двух разработанных вариантов фильтров. Для демонстрации результата работы Вашего фильтра необходимо построить спектрограммы выходных wav-файлов.





# ХОД РАБОТЫ

#include <iostream>

#include <conio.h>

#include <cmath>

#include "wav\_lib.h"

int main()

{

    WAVInfo info;

    float h0 = pow(2, -5) - pow(2, -7);

    float h1 = pow(2, -4) - pow(2, -6);

    float h2 = pow(2, -4) - pow(2, -7);

    float h3 = pow(2, -3) - pow(2, -6);

    float h4 = pow(2, -1) - pow(2, -3);

    float h5 = pow(2, -1);

    int\* x, \* y, \* y\_asm;

    x = wavread("src\_var\_01.wav", info);

    y = new int[info.NumberOfSamples];

    memset(y, 0, info.NumberOfSamples);

    y\_asm = new int[info.NumberOfSamples];

    memset(y\_asm, 0, info.NumberOfSamples);

    size\_t last = (size\_t)x + info.NumberOfSamples \* 4;

    int beg, end;

    \_\_asm

    {

        rdtsc

        mov beg, eax            // calculate number of cycles

        mov ecx, x              // ecx = x (ptr)

        add ecx, 4\*10           // ecx += 10

        mov ebx, y\_asm          //....

        add ebx, 4\*10           //....

    loop1:

        mov eax, [ecx]          // eax = x[n]

        add eax, [ecx - 4\*10]   // eax += x[n - 10]

        mov edi, eax

        sar eax, 5              // eax >>= 5

        sar edi, 7              // edi >>= 7

        sub eax, edi

        mov [ebx], eax          // y[n] = eax

        mov eax, [ecx - 4]      // eax = x[n - 1]

        add eax, [ecx - 4\*9]    // eax += x[n - 9]

        mov edi, eax

        sar eax, 4              // eax >>= 4

        sar edi, 6              // edi >>= 6

        sub eax, edi

        add [ebx], eax          // y[n] += eax

        mov eax, [ecx - 4\*2]    // eax = x[n - 2]

        add eax, [ecx - 4\*8]    // eax += x[n - 8]

        mov edi, eax

        sar eax, 4              // eax >>= 5

        sar edi, 7              // edi >>= 7

        sub eax, edi

        add[ebx], eax           // y[n] += eax

        mov eax, [ecx - 4\*3]    // eax = x[n - 3]

        add eax, [ecx - 4\*7]    // eax += x[n - 7]

        mov edi, eax

        sar eax, 3              // eax >>= 3

        sar edi, 6              // edi >>= 6

        sub eax, edi

        add[ebx], eax           // y[n] += eax

        mov eax, [ecx - 4\*4]    // eax = x[n - 4]

        add eax, [ecx - 4\*6]    // eax += x[n - 6]

        mov edi, eax

        sar eax, 1              // eax >>= 1

        sar edi, 3              // edi >>= 3

        sub eax, edi

        add[ebx], eax           // y[n] += eax

        mov eax, [ecx - 4\*5]    // eax = x[n - 5]

        sar eax, 1              // eax >>= 1

        add[ebx], eax           // y[n] += eax

        add ecx, 4              // x++

        add ebx, 4              // y\_asm++

        cmp ecx, last           // if ecx < last then

        jl loop1                // goto cycle;

        rdtsc                   // calculate number of cycles

        mov end, eax

    }

    std::cout << "Number of processor cycles for asm: " << end - beg << std::endl;

    \_\_asm

    {

        rdtsc                   // calculate number of cycles

        mov beg, eax

    }

    for(size\_t i = 10; i < info.NumberOfSamples; ++i) {

        y[i] = h0 \* (x[i - 0] + x[i - 10])

             + h1 \* (x[i - 1] + x[i - 9])

             + h2 \* (x[i - 2] + x[i - 8])

             + h3 \* (x[i - 3] + x[i - 7])

             + h4 \* (x[i - 4] + x[i - 6])

             + h5 \*  x[i - 5];

    }

    \_\_asm

    {

        rdtsc                   // calculate number of cycles

        mov end, eax

    }

    std::cout << "Number of processor cycles for c: " << end - beg << std::endl;

    // End of student's code

    wavwrite("proc\_var\_01asm.wav", info, y\_asm);

    wavwrite("proc\_var\_01c.wav", info, y);

    delete[] x;

    delete[] y;

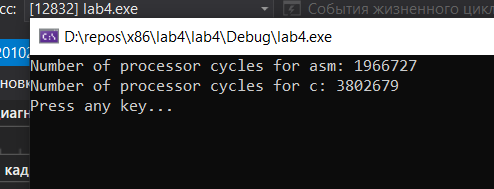
    delete[] y\_asm;

    std::cout << "Press any key..." << std::endl;

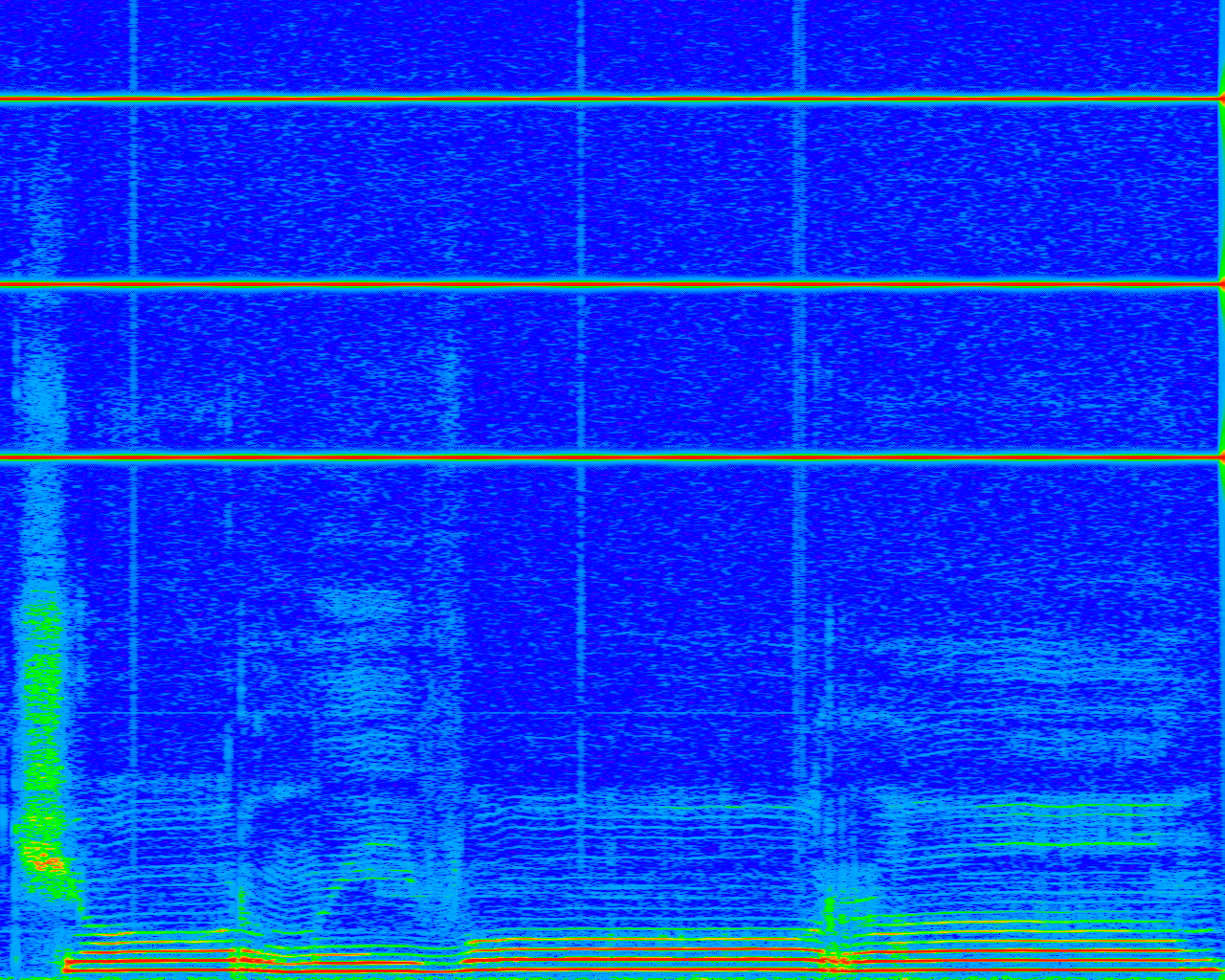
    \_getch();

    return 0;

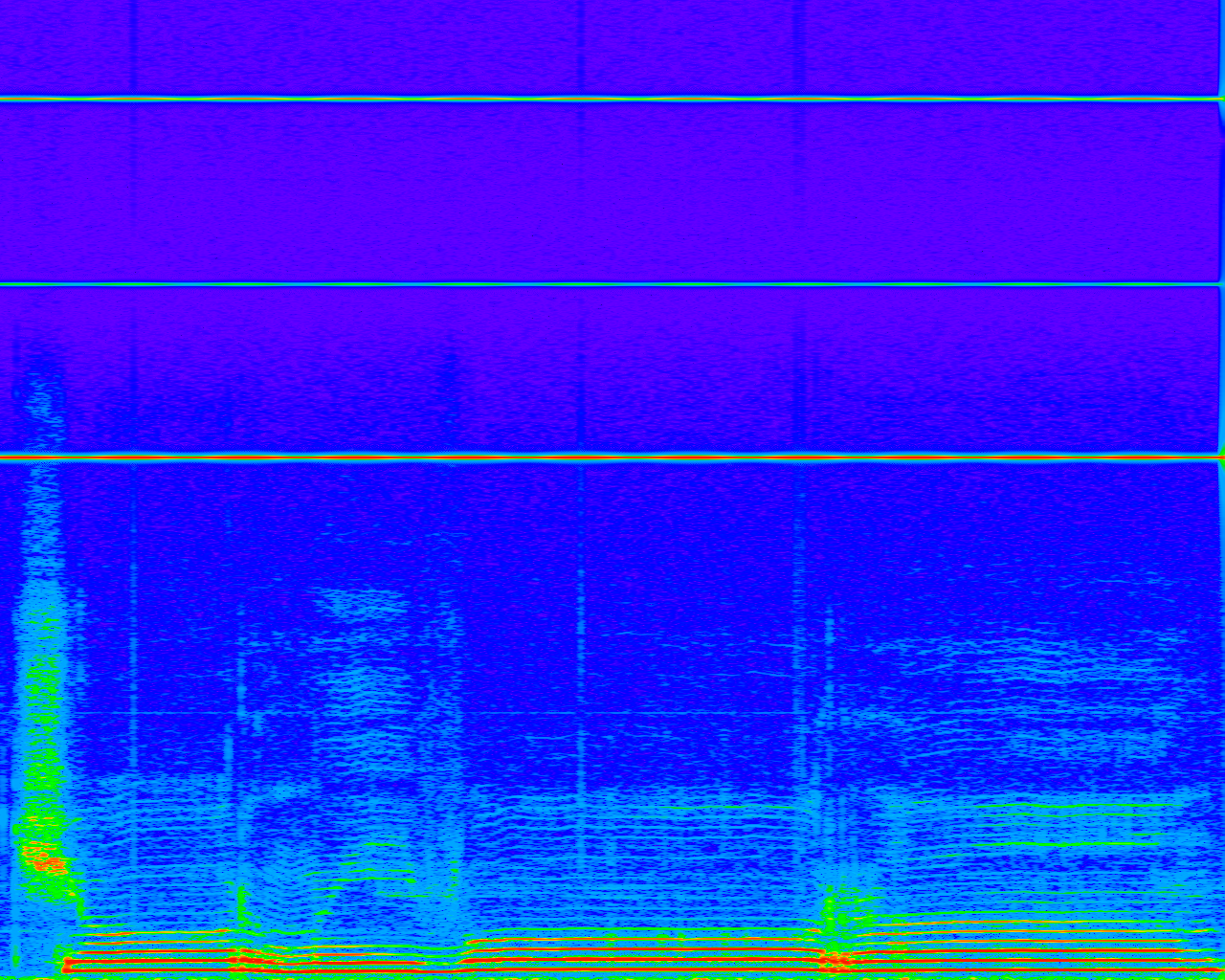
}

****

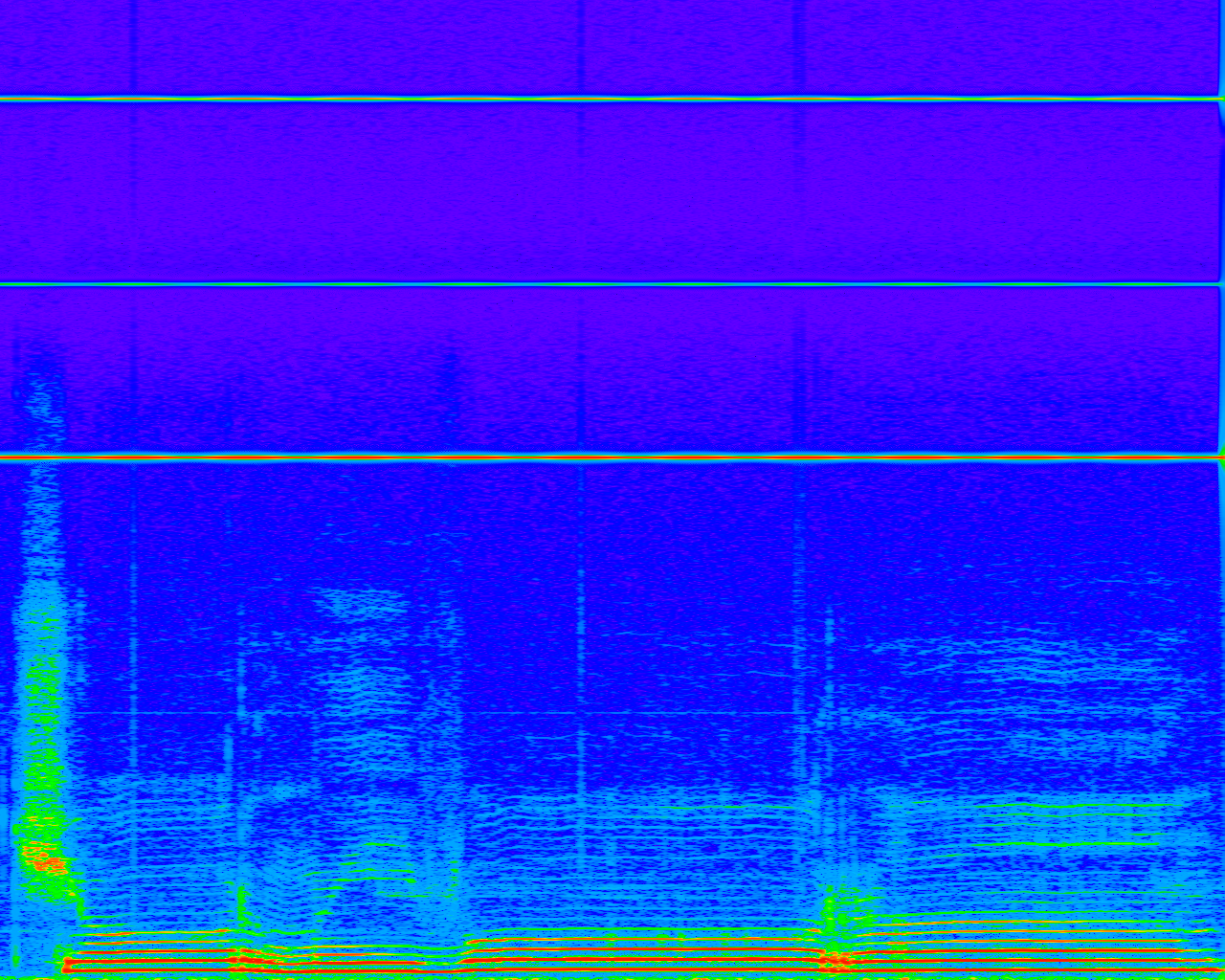
Сравнение работы программы с ассемблерной вставкой и без

****

Спектрограмма исходного файла



Спектрограмма после обработки файла фильтром с низкоуровневой вставкой



Спектрограмма после обработки файла только на высокоуровневом языке C++

# ВЫВОД

В ходе лабораторной работы научились использовать ассемблерные вставки в компиляторе MSVC для Windows на архитектуре Intel x86. Также измерили их эффективность при выполнении на одной машине при помощи счетчика тактов. В результате мы получили разницу почти в 2 раза по количеству тактов затраченных на выполнение одной и той же задачи.