Silesian University of Technology

Department of Graphics, Computer Vision and Digital Systems

Year	Type*: SSI/NSI/NSM	Subject: Assembler Programming Languages	Group	Section
2024/2025	SSI	APL – LAB	1	1
Tutor:	dr inż. Adam Opara		Class date: (week day, hour)	
Section:	1. Piotr Copek 2. Zuzanna Micorek		27.06.2025	
Contact Email:	pc21339@student.polsl.pl		8.30 – 10.00	

Report

Task 1

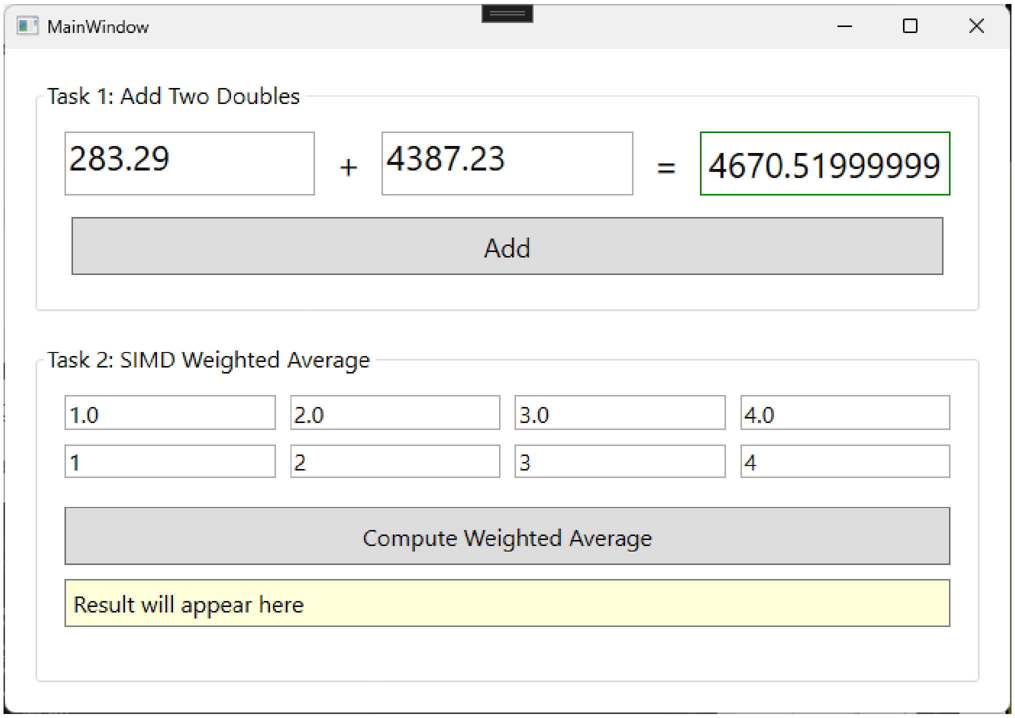
Create a solution with WPF main window and assembler DLL. The minimal functionality is adding at least 2 double point values given by values within text boxes.

```
.DATA
.CODE

PUBLIC asmAddTwoDoubles

asmAddTwoDoubles PROC
    ; add scalar double in xmm0 and xmm1
    addsd xmm0, xmm1
    ret
asmAddTwoDoubles ENDP

END
```



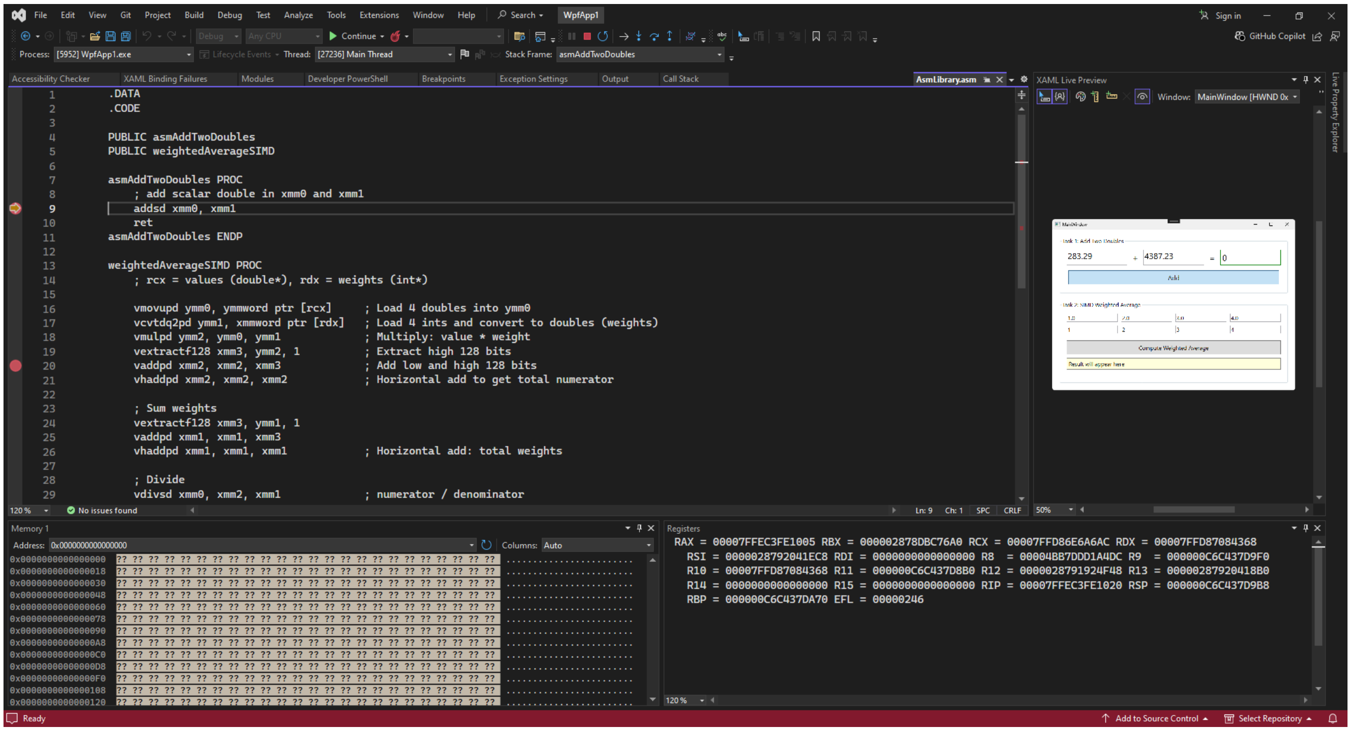
283.29 + 4387.23 = 4670.51999999

1.0 2.0 3.0 4.0

1 2 3 4

Result will appear here

[Figure 1] - Working application adding two double numbers.



Process: [5952] WpfApp1.exe Thread: [27236] Main Thread Stack Frame: asmAddTwoDoubles

Window: MainWindow [HWND 0x

```
.DATA
.CODE

PUBLIC asmAddTwoDoubles
PUBLIC weightedAverageSIMD

asmAddTwoDoubles PROC
    ; add scalar double in xmm0 and xmm1
    addsd xmm0, xmm1
    ret
asmAddTwoDoubles ENDP

weightedAverageSIMD PROC
    ; rcx = values (double*), rdx = weights (int*)

    vmovupd ymm0, ymmword ptr [rcx]     ; Load 4 doubles into ymm0
    vcvtdq2pd ymm1, xmmword ptr [rdx]   ; Load 4 ints and convert to doubles (weights)
    vmulpd ymm2, ymm0, ymm1             ; Multiply: value * weight
    vextractf128 xmm3, ymm2, 1          ; Extract high 128 bits
    vaddpd xmm2, xmm2, xmm3             ; Add low and high 128 bits
    vhaddpd xmm2, xmm2, xmm2            ; Horizontal add to get total numerator

    ; Sum weights
    vextractf128 xmm3, ymm1, 1
    vaddpd xmm1, xmm1, xmm3
    vhaddpd xmm1, xmm1, xmm1            ; Horizontal add: total weights

    ; Divide
    vdivsd xmm0, xmm2, xmm1             ; numerator / denominator
```

120 % Ln: 9 Ch: 1 SPC CRLF 50%

MainWindow

Task 1: Add Two Doubles

283.29 + 4387.23 = 0

Add

Task 2: SIMD Weighted Average

1.0	2.0	3.0	4.0
1	2	3	4

Compute Weighted Average

Result will appear here

Address: 0x0000000000000000 Columns: Auto

```
0x0000000000000000  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x0000000000000018  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x0000000000000030  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x0000000000000048  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x0000000000000060  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x0000000000000078  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x0000000000000090  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x00000000000000A8  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x00000000000000C0  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x00000000000000D8  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x00000000000000F0  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x0000000000000108  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x0000000000000120  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
```

```
RAX = 00007FFEC3FE1005 RBX = 000002878DBC76A0 RCX = 00007FFD86E6A6AC RDX = 00007FFD87084368
  RSI = 0000028792041EC8 RDI = 0000000000000000 R8  = 00004BB7DDD1A4DC R9  = 000000C6C437D9F0
  R10 = 00007FFD87084368 R11 = 000000C6C437D8B0 R12 = 0000028791924F48 R13 = 00000287920418B0
  R14 = 0000000000000000 R15 = 0000000000000000 RIP = 00007FFEC3FE1020 RSP = 000000C6C437D9B8
  RBP = 000000C6C437DA70 EFL = 00000246
```

120 %

[Figure 2] - Screenshot from debug with breakpoint set in assembly code.

Task 2

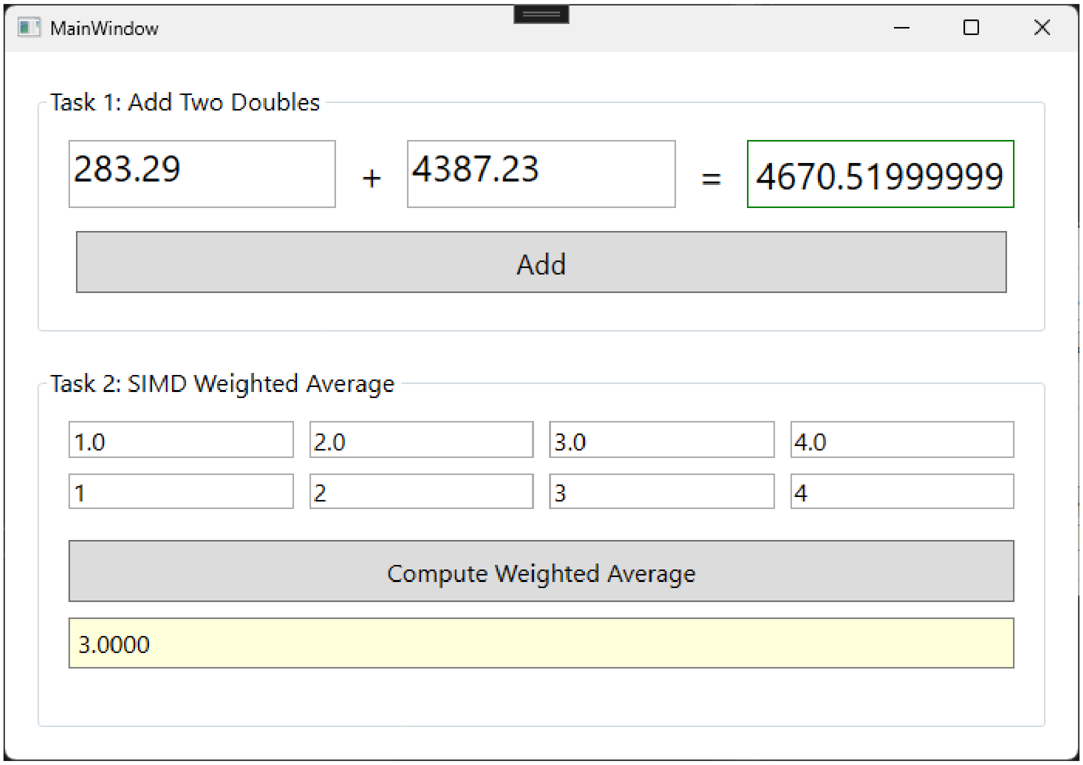
Create a new function with more advanced functionality using SIMD computing the weighted average of the four products given by the double and integer each.

$$\text{Weighted Average} = \frac{\sum_{i=1}^{4} \text{value}_i \times \text{weight}_i}{\sum_{i=1}^{4} \text{weight}_i}$$

```
weightedAverageSIMD PROC
    vmovupd ymm0, ymmword ptr [rcx]
    vcvtdq2pd ymm1, xmmword ptr [rdx]
    vmulpd ymm2, ymm0, ymm1
    vextractf128 xmm3, ymm2, 1
    vaddpd xmm2, xmm2, xmm3
    vhaddpd xmm2, xmm2, xmm2
    vextractf128 xmm3, ymm1, 1
    vaddpd xmm1, xmm1, xmm3
    vhaddpd xmm1, xmm1, xmm1
    vdivsd xmm0, xmm2, xmm1

    ret
weightedAverageSIMD ENDP
```

1. `vmovupd ymm0, ymmword ptr [rcx]` – Loads 4 double-precision floats (weights) into `ymm0` from memory at `rcx`.
2. `vcvtdq2pd ymm1, xmmword ptr [rdx]` – Converts 4 integers from memory at `rdx` to 4 double-precision floats and stores them in `ymm1` (zero-extends to YMM).
3. `vmulpd ymm2, ymm0, ymm1` – Multiplies weights (`ymm0`) and converted values (`ymm1`) element-wise, storing the result in `ymm2`.
4. `vextractf128 xmm3, ymm2, 1` – Extracts the upper 128 bits of `ymm2` into `xmm3`.
5. `vaddpd xmm2, xmm2, xmm3` – Adds the lower and upper halves of the product vector to sum all products partially.
6. `vhaddpd xmm2, xmm2, xmm2` – Horizontally adds to get the final sum of products in `xmm2`.
7. Repeat steps 4–6 for `ymm1` – Sums all weights.
8. `vdivsd xmm0, xmm2, xmm1` – Divides the total weighted sum by the total weight to get the weighted average.



MainWindow

Task 1: Add Two Doubles

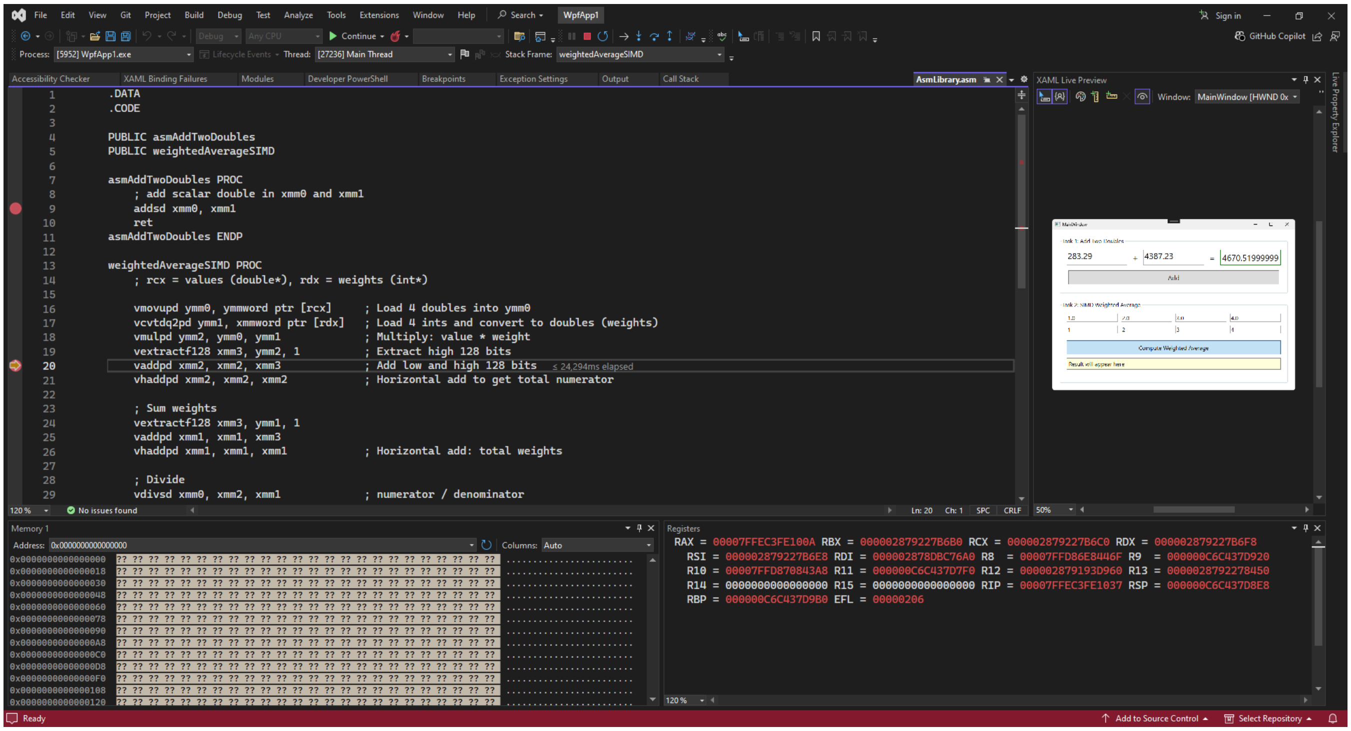
283.29 + 4387.23 = 4670.51999999

Task 2: SIMD Weighted Average

1.0	2.0	3.0	4.0
1	2	3	4

3.0000

[Figure 3] - Working application computing the weighted average of the four products.



Process: [5952] WpfApp1.exe Thread: [27236] Main Thread Stack Frame: weightedAverageSIMD

```
.DATA
.CODE

PUBLIC asmAddTwoDoubles
PUBLIC weightedAverageSIMD

asmAddTwoDoubles PROC
    ; add scalar double in xmm0 and xmm1
    addsd xmm0, xmm1
    ret
asmAddTwoDoubles ENDP

weightedAverageSIMD PROC
    ; rcx = values (double*), rdx = weights (int*)

    vmovupd ymm0, ymmword ptr [rcx]     ; Load 4 doubles into ymm0
    vcvtdq2pd ymm1, xmmword ptr [rdx]   ; Load 4 ints and convert to doubles (weights)
    vmulpd ymm2, ymm0, ymm1             ; Multiply: value * weight
    vextractf128 xmm3, ymm2, 1          ; Extract high 128 bits
    vaddpd xmm2, xmm2, xmm3             ; Add low and high 128 bits  ≤ 24,294ms elapsed
    vhaddpd xmm2, xmm2, xmm2            ; Horizontal add to get total numerator

    ; Sum weights
    vextractf128 xmm3, ymm1, 1
    vaddpd xmm1, xmm1, xmm3
    vhaddpd xmm1, xmm1, xmm1            ; Horizontal add: total weights

    ; Divide
    vdivsd xmm0, xmm2, xmm1             ; numerator / denominator
```

120 % No issues found Ln: 20 Ch: 1 SPC CRLF

Window: MainWindow [HWND 0x

283.29 + 4387.23 = 4670.51999999

Add

Task 2: SIMD Weighted Average

1.0 2.0 3.0 4.0

1 2 3 4

Compute Weighted Average

Result will appear here

Memory 1

Address: 0x0000000000000000 Columns: Auto

```
0x0000000000000000 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x0000000000000018 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x0000000000000030 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x0000000000000048 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x0000000000000060 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x0000000000000078 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x0000000000000090 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x00000000000000A8 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x00000000000000C0 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x00000000000000D8 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x00000000000000F0 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x0000000000000108 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
0x0000000000000120 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ........................
```

Registers

```
RAX = 00007FFEC3FE100A RBX = 000002879227B6B0 RCX = 000002879227B6C0 RDX = 000002879227B6F8
  RSI = 000002879227B6E8 RDI = 000002878DBC76A0 R8  = 00007FFD86E8446F R9  = 000000C6C437D920
  R10 = 00007FFD870843A8 R11 = 000000C6C437D7F0 R12 = 000002879193D960 R13 = 0000028792278450
  R14 = 0000000000000000 R15 = 0000000000000000 RIP = 00007FFEC3FE1037 RSP = 000000C6C437D8E8
  RBP = 000000C6C437D9B0 EFL = 00000206
```

[Figure 4] - Screenshot from debug with breakpoint set in assembly code.

Conclusions

The project demonstrates effective integration of assembly code with a WPF C# application. Task one implemented double addition, while task two extended this to compute a weighted average of four products combining doubles and integers. The solution shows performance benefits of SIMD.