Silesian University of Technology

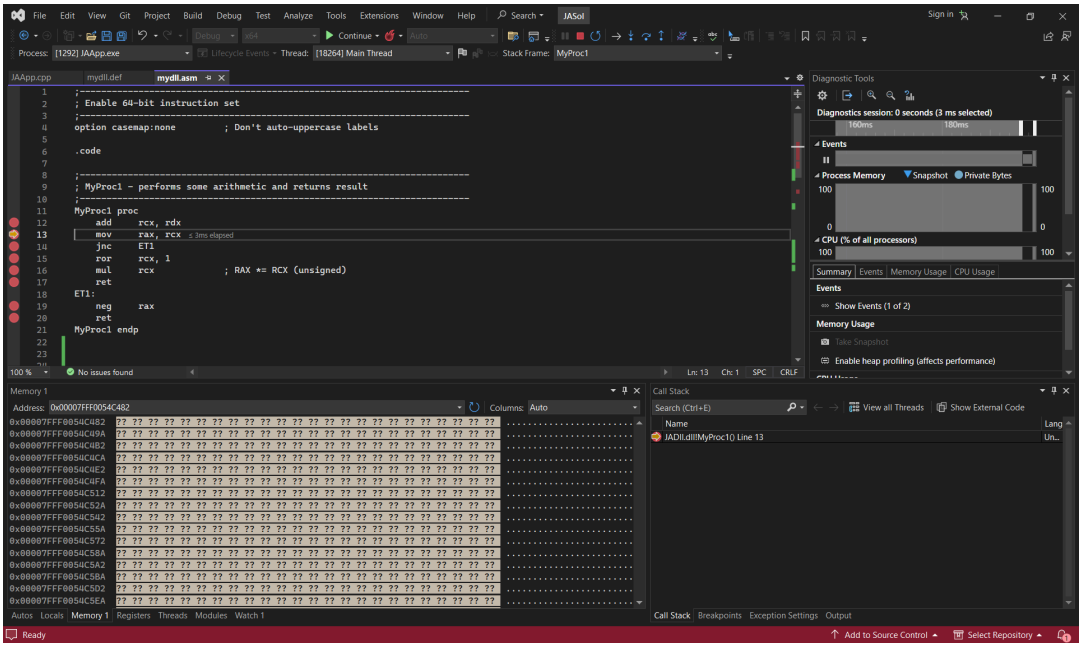
Department of Graphics, Computer Vision and Digital Systems

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**		**11/04/2025**	
Contact Email:	**pc312199@student.polsl.pl**		**8.30-10.00**	

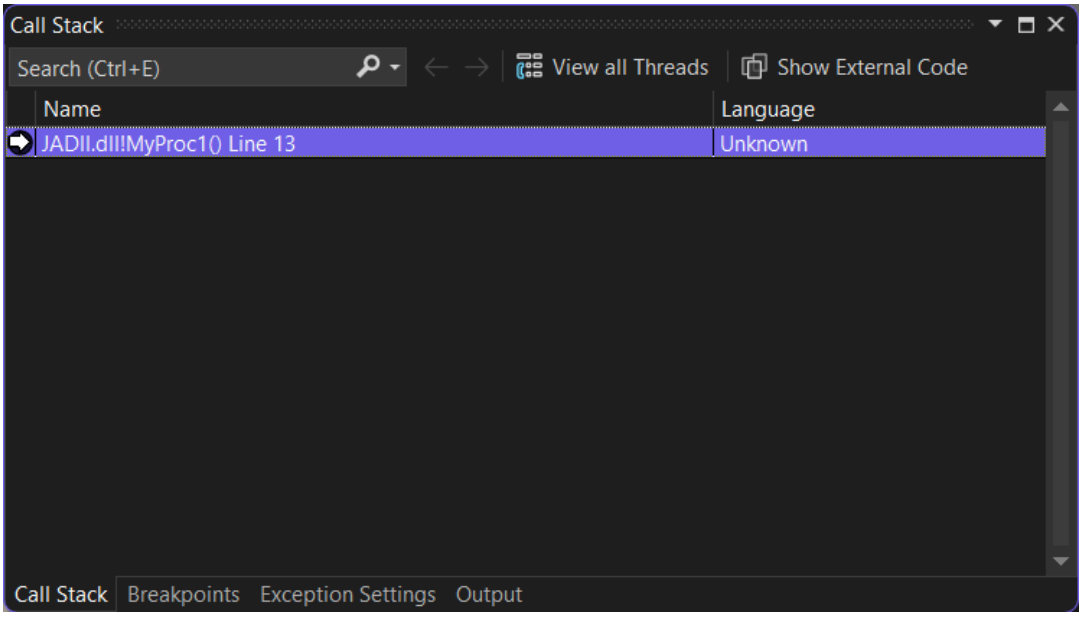
Report

Task 1

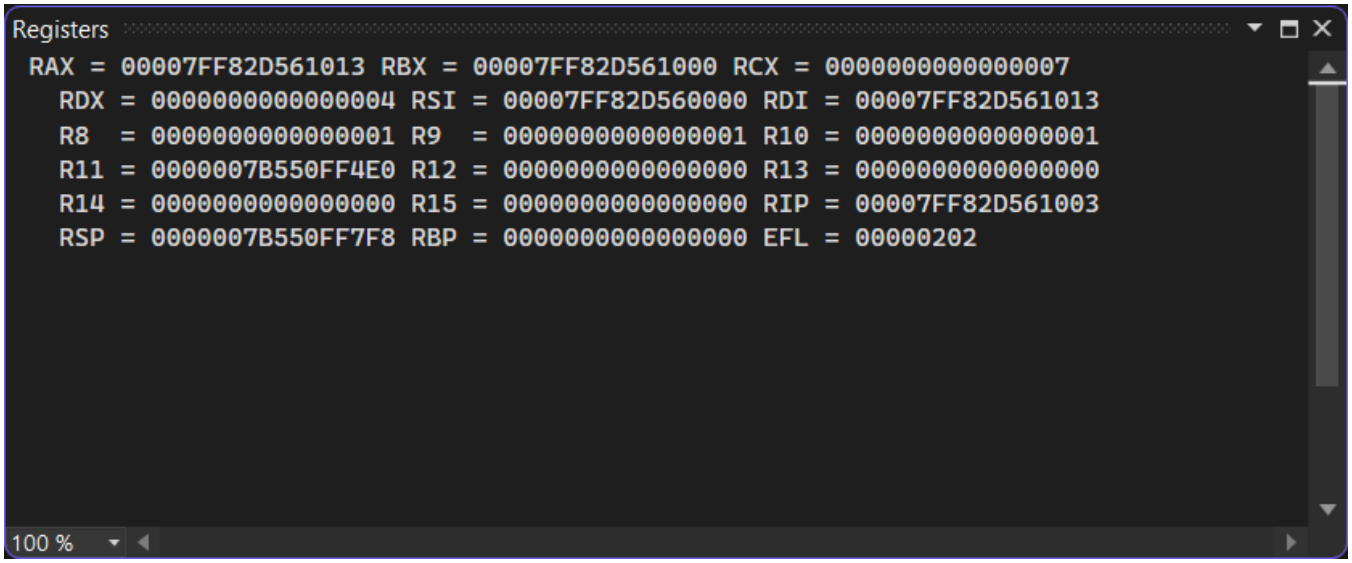
After setting up Visual Studio I created few breakpoints in `mydll.asm` to check if `JAApp.cpp` will call it properly. Dll process was called successfully after adding full path of the `JADll.lib` to the `JAApp - Linker - Input - Additional Dependencies`.



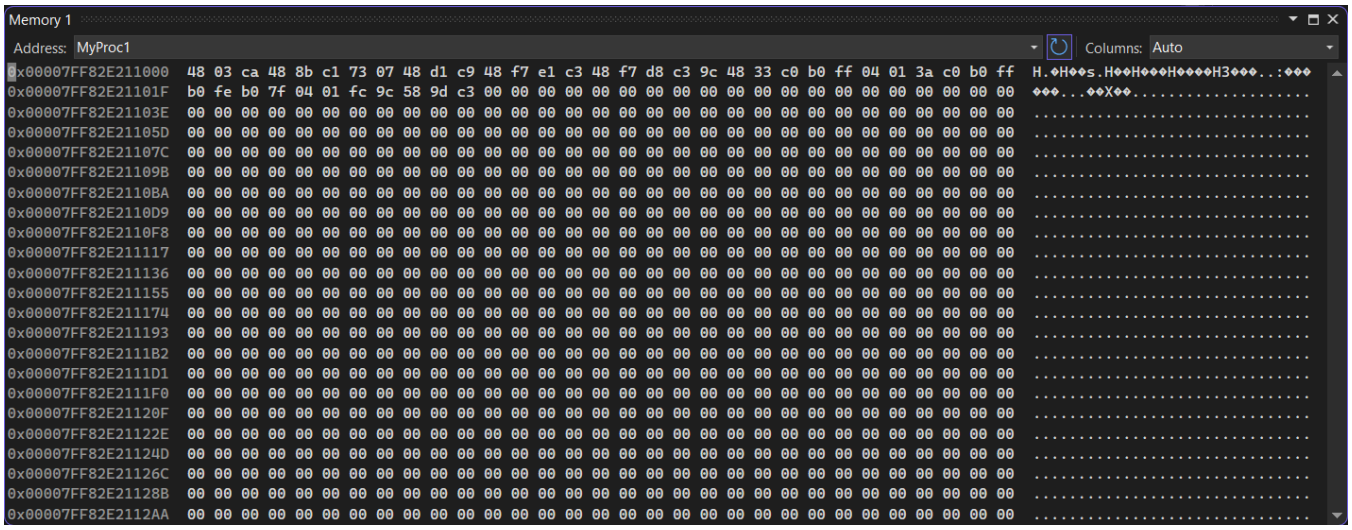
[Figure 1] - Debugger window paused on a selected command.



[Figure 2] - Stack window showing current function.



[Figure 3] - The register window.



[Figure 4] - Memory window.

Task 2

I created simple flags triggering code, and called it dynamically in main function to obtain flags results

```
MyProc2 proc
    pushfq                  ; Save caller's FLAGS
    xor     rax, rax        ; Clear RAX
    ; Trigger CF and AF
    mov     al, 0FFh
    add     al, 1           ; CF and AF

    ; Trigger ZF
    sub     al, al          ; ZF
    ; Trigger SF
    mov     al, -1          ; SF

    ; Trigger PF
    mov     al, 0FEh        ; Even parity - PF

    ; Trigger OF
    mov     al, 7Fh
    add     al, 1           ; OF

    ; Direction and interrupt flags
    cld                     ; DF
    ; sti                   ; Needs more privileges

    ; Capture flags
    pushfq

    ; Store final FLAGS in RAX
    pop     rax
    ; Restore caller's original FLAGS
    popfq
    ret
MyProc2 endp
```

Instruction	Affected Flags	Value
`add al, 1`	`CF`	0
`mov al, 0FEh`	`PF`	0
`add al, 1`	`AF`	1
`sub al, al`	`ZF`	0
`mov al, -1`	`SF`	1
`add al, 1`	`OF`	1
`cld`	`DF`	0
`sti`	`IF`	1

[Table 1] - Values of modified flags.

Instruction `sti` was omitted because it requires higher privileges to be executed.

Task 3

```
#include <windows.h>
#include <iostream>

typedef int(__fastcall* MYPROC1)(long long, long long);
typedef unsigned long long(__fastcall* MYPROC2)();

extern "C" int _fastcall MyProc1(long long x, long long y);
extern "C" unsigned long long _fastcall MyProc2();

int main()
{
    HMODULE hDll = LoadLibrary(L"JADll.dll");
    if (!hDll) {
        std::cerr << "Failed to load DLL" << std::endl;
        return 1;
    }

    MYPROC1 MyProc1 = (MYPROC1)GetProcAddress(hDll, "MyProc1");
    if (!MyProc1) {
        std::cerr << "Failed to find MyProc1" << std::endl;
        FreeLibrary(hDll);
        return 1;
    }

    MYPROC2 MyProc2 = (MYPROC2)GetProcAddress(hDll, "MyProc2");
    if (!MyProc2) {
        std::cerr << "Failed to find MyProc2" << std::endl;
        return 1;
    }
    int x = 3, y = 4;
    int z = MyProc1(x, y);

    std::cout << "Result: " << z << std::endl;
```

```
    unsigned long long flags = MyProc2();

    std::cout << "Flags: 0x" << std::hex << flags << std::endl;

    std::cout << "CF: " << ((flags >> 0) & 1) << std::endl;
    std::cout << "PF: " << ((flags >> 2) & 1) << std::endl;
    std::cout << "AF: " << ((flags >> 4) & 1) << std::endl;
    std::cout << "ZF: " << ((flags >> 6) & 1) << std::endl;
    std::cout << "SF: " << ((flags >> 7) & 1) << std::endl;
    std::cout << "OF: " << ((flags >> 11) & 1) << std::endl;
    std::cout << "DF: " << ((flags >> 10) & 1) << std::endl;
    std::cout << "IF: " << ((flags >> 9) & 1) << std::endl;

    FreeLibrary(hDll);

    return 0;
}
```

Conclusions

Tasks demonstrated key aspects of assembly programming and debugging. The debugger paused at the specified breakpoints in MyProc1, allowing observation of the call stack, registers, and memory. MyProc2 procedure was designed to manipulate specific flags, of which results were shown in the provided table (Table 1). Additionally MyProc2 was dynamically loaded and called using LoadLibrary and GetProcAddress which highlighted practical DLL usage in C++.