	Instytut Informatyki Politechniki Śląskiej Zespół Mikroinformatyki i Teorii Automatów Cyfrowych			
Rok akademicki:	**Rodzaj studiów*: SSI/NSI/NSM**	**Języki Asemblerowe**	**LAB**	**I**

TEMAT:

Tworzenie projektu asemblerowego dla środowiska Visual Studio 2010.

CEL:

Celem ćwiczenia jest poznanie możliwości VS w zakresie tworzenia i uruchamiania aplikacji z kodem mieszanym w języku C ++ oraz asemblerze. W założeniu aplikacja składa się z dwóch elementów – aplikacji napisanej w j. C++ oraz biblioteki DLL napisanej w asemblerze dla środowiska Windows. Konstrukcja projektu zakłada możliwość wywoływania funkcji bibliotecznych napisanych w asemblerze z poziomu aplikacji oraz pokazuje prawidłową konfigurację środowiska umożliwiającą debugowanie kodu do poziomu asemblera, obserwację stanu rejestrów i flag procesora czy obszarów pamięci danych.

ZAŁOŻENIA:

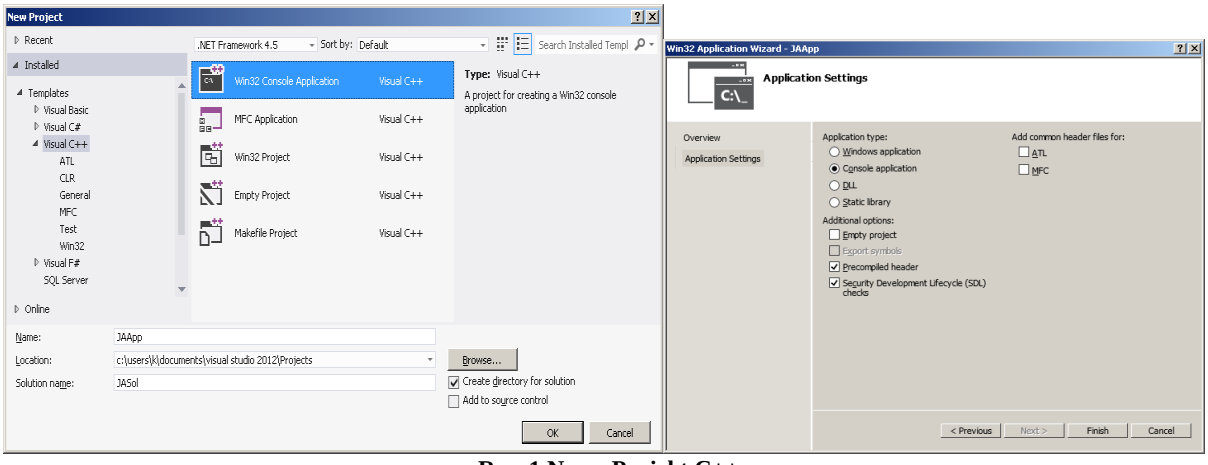
W środowisku VS 2010 zakładamy solucję składającą się z dwóch projektów:

- Projekt aplikacja WIN32 w j. C++,
- Projekt biblioteka DLL w asemblerze, zawierająca m.in. pliki:
  - mojadll.asm
  - mojadll.def

W bibliotece DLL utworzona zostanie funkcja, która będzie wywoływana przez aplikację.

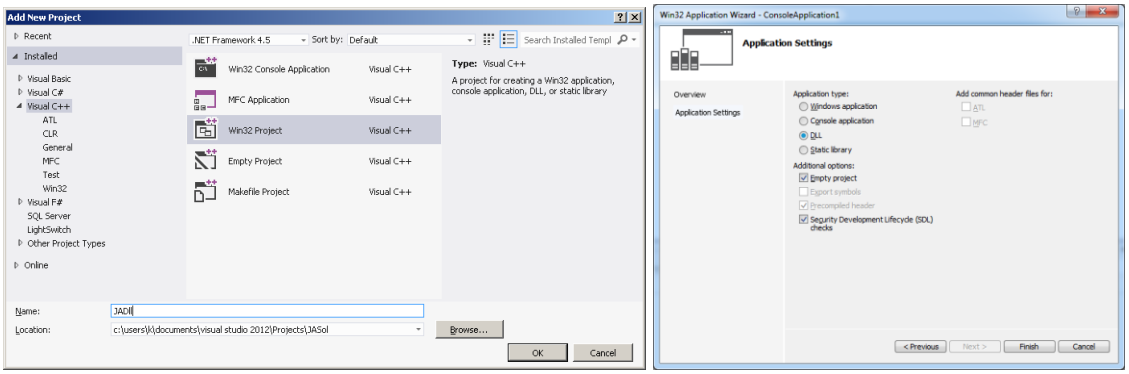
WYKONANIE:

W środowisku VS 2010 tworzymy nowy projekt (a jednocześnie nową solucję o nazwie **JASol**).WybieramyNew->Project-> Win32Console Applicationo nazwie **JAApp** (rys. 1)



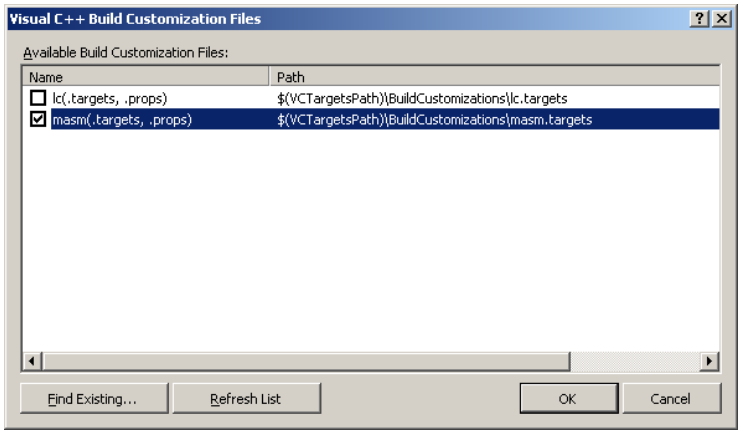
Rys. 1 Nowy Projekt C++

Następnie do solucji JASol dodajemy nowy projekt **JADll**. Klikamy prawym na solucji Add -> New Project -> Win32 Project, zaznaczamy Application type: DLL, **Empty Project (!!!)**(rys. 2).



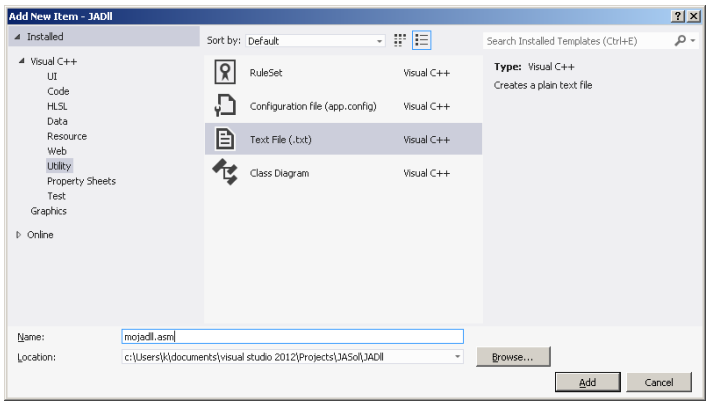
Rys. 2 Nowy projekt DLL

Dla środowiska VC 2010 dodanie pliku asm musi zostać poprzedzone włączeniem opcji asemblacji za pomocą asemblera MASM. W tym celu należy zaznaczyć projekt JADll i prawym przyciskiem myszy wybrać opcję BuildCustomizations...,a następnie zaznaczyć opcję masm(targets,props) (rys.3).



Rys. 3.BuildCustomizations

Po wykonaniu tej czynności do projektu JADll można dodać plik **mojadll.asm** poprzez wybranie opcji Add New Item - wybierając typ pliku Text file .txt (lub C++ File .cpp) a w nazwie pliku zmieniając domyślne rozszerzenie txt (cpp) na asm (rys. 4)



Rys. 4. Add new Item (asm)

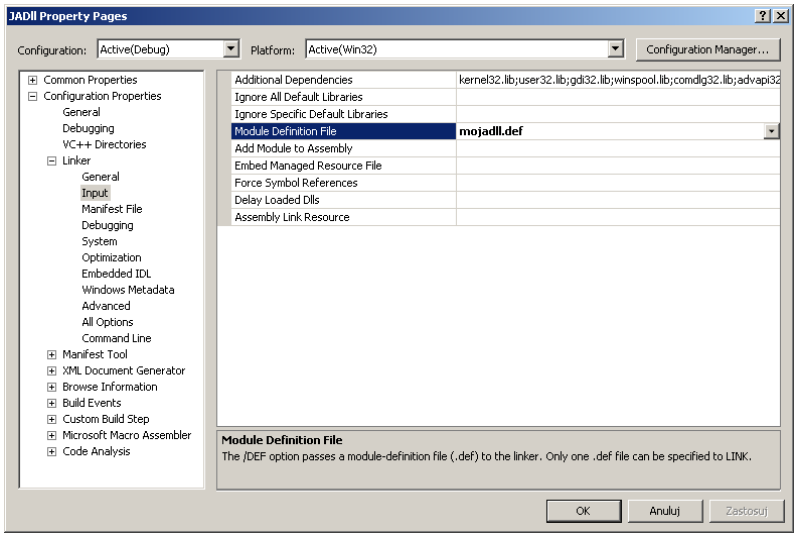
Zawartość pliku mojadll.asm (http://www.i-lo.tarnow.pl/edu/inf/prg/win32asm/pages/17.htm)
(W razie potrzeby doinstalować pakiet masm32):

```
;-------------------------------------------------------------------------
.386
.MODEL FLAT, STDCALL

OPTION CASEMAP:NONE

INCLUDE C:\masm32\include\windows.inc

.CODE

DllEntry PROC hInstDLL:HINSTANCE, reason:DWORD, reserved1:DWORD

mov    eax, TRUE
ret

DllEntry ENDP

;-------------------------------------------------------------------------
; To jest przykładowa funkcja. Jest tutaj, aby pokazać,
; gdzie należy umieszczać własne funkcje w bibliotece DLL
;-------------------------------------------------------------------------

MyProc1 proc x: DWORD, y: DWORD

xor    eax,eax
mov    eax,x
mov    ecx,y
ror    ecx,1
shld   eax,ecx,2
jnc ET1
mul    y
ret
ET1:
Mul    x
Neg    y
ret

MyProc1 endp

END DllEntry
;-------------------------------------------------------------------------
```

Następnie definiujemy, które funkcje będą eksportowane tzn. udostępnione na zewnątrz biblioteki. Do Source Files projektu JADll dodajemy plik mojadll.def (Add New Item - Module Definition File). Zawartość pliku mojadll.def:

```
;-------------------------------------------------------------------------
LIBRARY JADll
EXPORTS MyProc1
;-------------------------------------------------------------------------
```

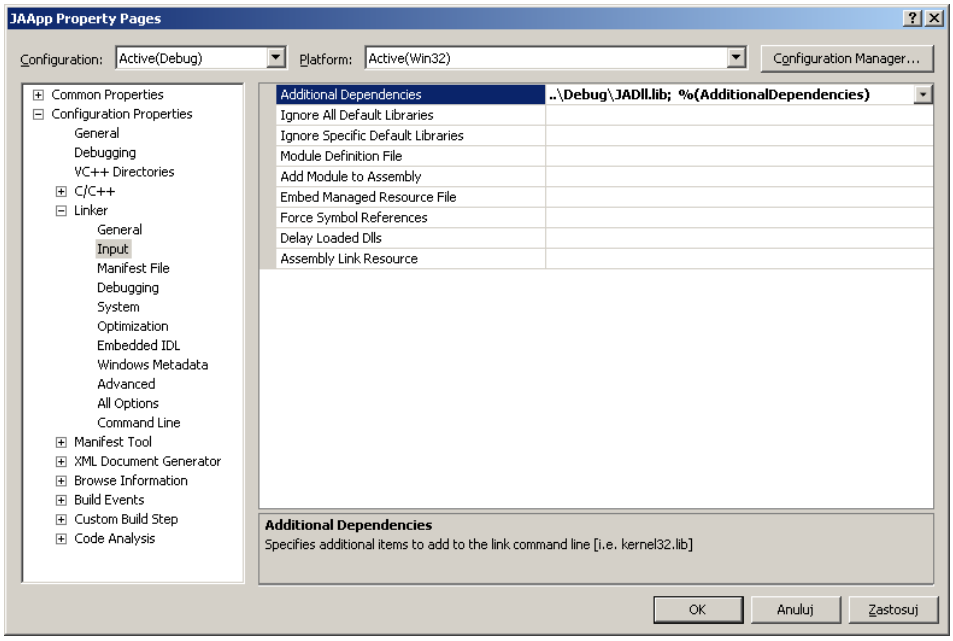
Należy się upewnić, czy w opcjach linkera jest wpisana właściwa nazwa pliku definicji (JADll Properties->Linker->Input, Module Definition File mojadll.def (rys. 5)



Rys. 5 Opcje linkera

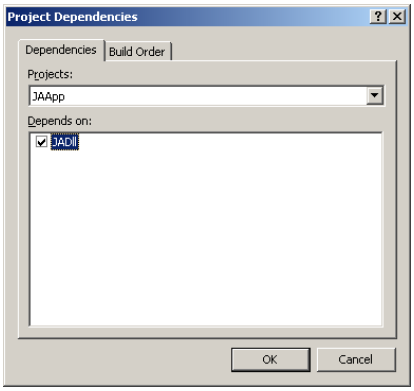
W aplikacji JAApp funkcje biblioteczne można wywoływać na dwa sposoby:

- dynamicznie ładując bibliotekę JADll.dll za pomocą funkcji API HLIBRARY LoadLibrary (LPCWSTR lplibFileName) / FreeLibrary oraz GetProcAddress,
- statycznie poprzez linkowanie pliku JADll.lib w opcjach linkera aplikacji **JAApp** jak na rys. 6. (Może być wymagane podanie pełnej ścieżki do pliku JADll.lib).



Rys. 6. Dodawanie biblioteki statycznej do projektu

Aby projekty były kompilowane w prawidłowej kolejności (najpierw dll), klikamy prawym na JASol ->Project Dependencies… i zaznaczamy Projects: JAApp Depends on: JADll (rys. 7).



Rys. 7. Zależności projektu

Projekty powinny się kompilować i powinny powstawać w wyniku jego działania: plik exe i biblioteka dll z eksportowaną funkcją MyProc.
Uwagi:
- *.def jest potrzebny do utworzenia dll;
- *.lib nie jest potrzebny jeśli odwołujemy się do biblioteki dynamicznie czyli przez GetProcAddress.

W projekcie JAApp zawartość pliku JAApp.cpp może być następująca:

```
// JAApp.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
#include <windows.h>

extern "C" int _stdcall MyProc1 (DWORD x, DWORD y);


int _tmain(int argc, _TCHAR* argv[])
{
	int x = 3,y = 4, z = 0;

	z = MyProc1 (x, y);  // wywołanie procedury asemblerowej z biblioteki

	return 0;
}
```

Po skompilowaniu całości solucji możliwe będzie wywołanie funkcji bibliotecznej MyProc1 z aplikacji JAApp.exe a w trakcie debugowania w trybie krokowym możliwe jest wykonywanie pojedynczych rozkazów asemblerowych i obserwowanie zmian rejestrów procesora i flag.

PROJEKT W WINDOWS FORMS APPLICATIONS.

W przypadku użycia jako aplikacji wywołującej Windows Forms Application istnieje konieczność modyfikacji standardowych parametrów linkera aby możliwe było debugowanie krokowe.

1. Zaczynamy od utworzenia nowego projektu:
   File -> New -> Project
   W eksploratorze wybieramy **Visual C++ -> CLR -> Windows Forms Application**
   i podajemy nazwę JAApp**.**
2. Następnie dodajemy projekt JADll tak jak opisano powyżej

3. Teraz klikamy PPM na **Solutions** (eksplorator z lewej strony ekranu) i wybieramy z końca **Properties** po czym wybieramy **Project Dependencies** i w rozwijalnym menu wybieramy **JAApp** oraz zaznaczamy poniżej **biblioteka.** Zabieg ten służy ustaleniu zależności pomiędzy projektami.
4. Następnie klikamy PPM na **JAApp** (i znów eksplorator z lewej strony ekranu) i wybieramy z końca **Properties** i wybieramy **ConfigurationProperties -> General** , a następnie w oknie **CommonLanguage** zmieniamy na z **(/clr :pure)** na **(/clr)**.
5. Następnie w **ConfigurationProperties -> Linker -> Input** w oknie wybieramy trzy kropki (podanie ścieżki) w **AdditionalDependencies** i podajemy następującą ścieżkę: **..\Debug\LibAsm.lib** co potwierdzamy poprzez **OK.**

ZADANIA DO SPRAWOZDANIA

1. Utworzyć solucję JASol wraz z projektami JAApp oraz JADll (wg opisu powyżej). Ustawić pułapkę (breakpointa) na wybranym rozkazie procedury MyProc1, uruchomić program (run) i zaobserwować, czy debugger zatrzymuje się prawidłowo na rozkazie wyświetlając poprawnie kod źródłowy pliku asm.
   W sprawozdaniu zamieścić zrzut ekranu, który zawiera okna:
   - okno debuggera wstrzymanego na wybranym rozkazie,
   - okno stosu wskazujące aktualną funkcję,
   - okno rejestrów,
   - okno pamięci.
2. Zaproponować nową procedurę MyProc2 tak by wykonując ją krokowo zaobserwować zmiany znaczników rejestru flag: OV UP EI PL ZR AC PE CY.
   W sprawozdaniu umieścić w formie tabeli rozkazy i wartości modyfikowanych flag.
3. Osoby chętne mogą wywołać funkcję MyProc2 w sposób dynamiczny (LoadLibrary / FreeLibrary oraz GetProcAddress),
4. Wygenerować indywidualne sprawozdanie w formacie PDF zawierające wnioski.