Kamil Skarżyński

Programowanie niskopoziomowe Tworzenie i uruchamianie programów assemblerowych w Visual Studio Laboratorium 02

1. Wprowadzenie

Podczas laboratoriów zapoznamy się z:

1. Procesem konfiguracji VS 2015 w celu uruchomienia programu assemblerowego,
2. W jaki sposób zarządzać ustawineiami linkera w środowisku VS,
3. Procesem debuggowania programów w VS,
4. Narzędziem umożliwiającym podgląd wykonywanego programu,
5. Narzędziami umożliwiającymi podgląd zawratości rejestrów i pamięci w VS,
6. Podstawowymi instrukcjami arytmetycznymi dla dla procesora x86 w architekturze 32 bit.

2. Zadania

1. Utwórz nowy projekt w środowisku VS 2015 i skonfiguruj go na potrzeby masm32 (5) (2 punkty),
2. Utwórz program zapisujący wynik równania (z tabeli wariantów 4) do rejestru EAX (5.2) (2 punkty),
3. Wynik przepisz do zmiennej a następnie pobierz jej adres i pokaż w podglądzie pamięci (2 punkty),
4. Zaprezentuj przebieg wynonania programu za pomocą debuggera VS (5.4) (1 punkt)
5. Skonwertuj wynik do znaków ASCII (1 punkt),
6. Wypisz wynik za pomocą procedury WriteConsoleA (2 punkty).

3. Przydatne linki

1. Kody instrukcji

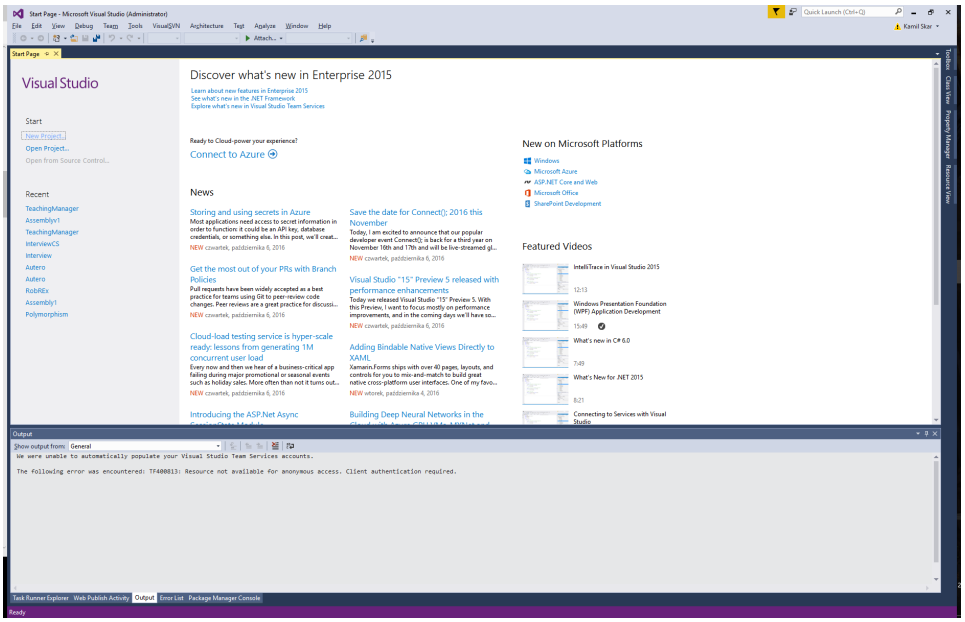
4. Tabela wariantów

Tabela 1. Tabela wariantów

Numer Wariantu	Równanie
1	5*A + 4*B - C
2	25*A - 4*A*B - C
3	7*(A - B) + C
4	11*A + B*C
5	2*A*B*C
6	2*A*C + 4*B*C
7	6*A + 4*B - 2*B*C
8	7*A + 2*B*C
9	2*A - B - C
10	(2*A + B + C)*C
11	5*A + 2*B - 10*C
12	17*A - 2*B - 2*C
13	10*A - 10*B - 10*C
14	2*A + 2*B - 2*C
15	5*A - 2*A*B - 4*B*C + 2*C

5. Pomoc

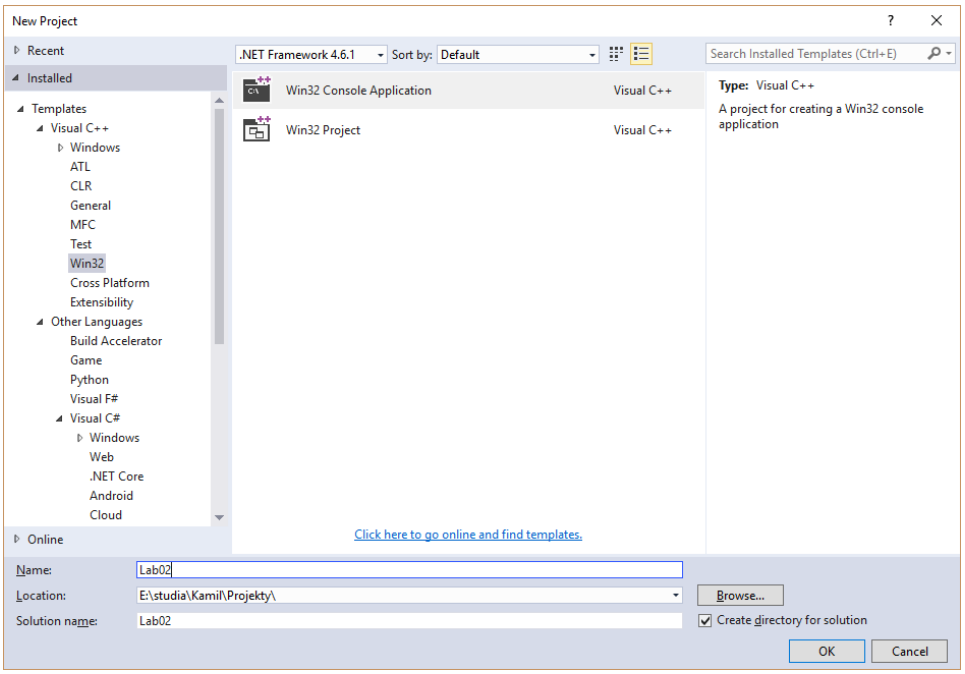
5.1. Konfiguracja środowiska Visual Studio 2015



Rysunek 1. Tworzenie nwoego projektu

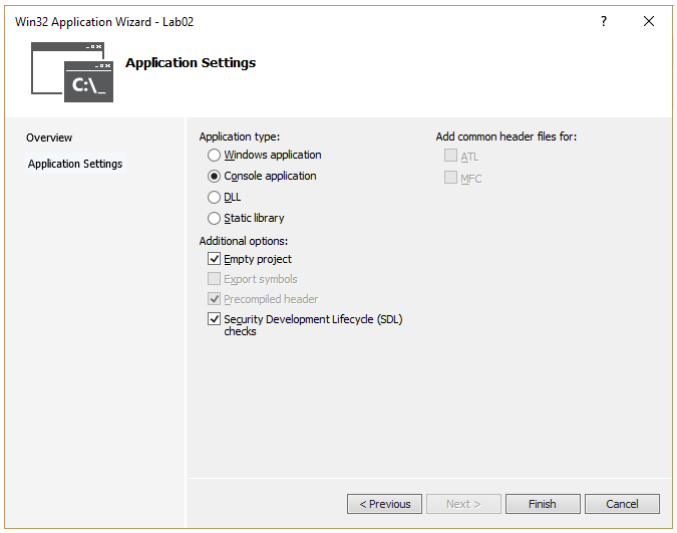
1. Przy pierwszym uruchomieniu środowiska niezbędne będzie utworzenie nowego projektu, oraz odpowiednia jego konfiguracja. Po uruchomieniu Visual Studio 2015 Enterprise Eiditon klikamy „New Project” lub wybieramy przez File->New->Project

2. Wybieramy z zakładek po lewej stronie Installed->Templates->Visual C++->Win32->Win32 Console Application. A następnie wpisujemy odpowiednią nazwę projektu (warto również zmienić lokalizacje projektu poprzez utworzenie własnego katalogu by projekty różnych grup ze sobą nie kolidowały). A następnie klikamy Ok.



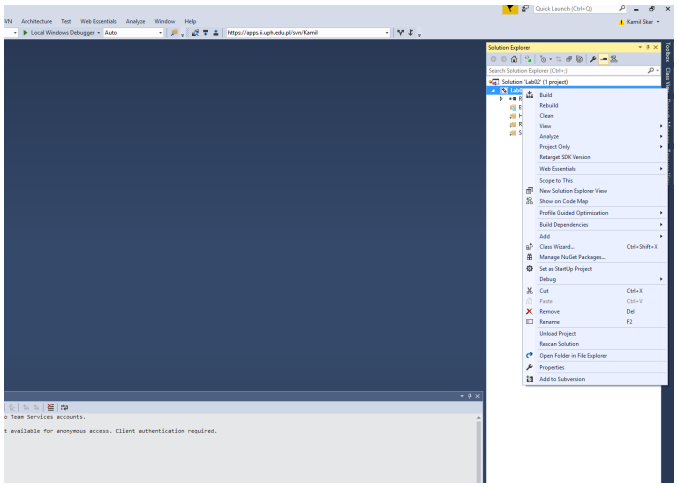
Rysunek 2. Tworzenie nwoego projektu

3. Klikamy Next, zaznaczamy opcję Empty project i klikamy Finish.

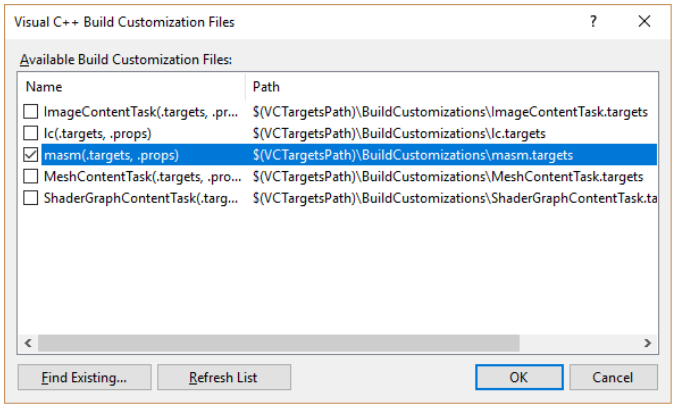


Rysunek 3.

4. Po utworzeniu nowego projektu klikamy na na zakładkę Project->Build Customization i zaznaczamy opcję masm w celu specyfikacji opcji kompilacji.

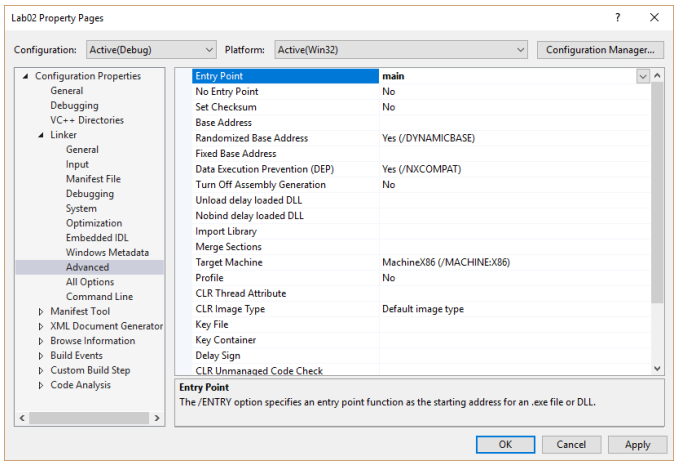


Rysunek 4.



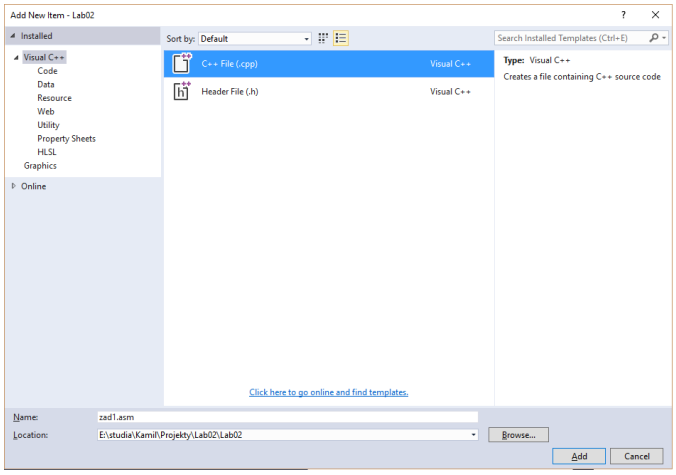
Rysunek 5.

5. W celu poprawnego działania linkera potrzebne jest wskazanie głównej procedury dla naszego projektu. Wybieramy zakładkę Project->Properties->Configuration Properties->Linker->Advanced->Entry Point wpisujemy wartość main.



Rysunek 6.

6. Klikamy na folder Source Files znajdujący się w hierarchi projektu prawym przyciskiem i wybieramy Add->New Item... następnie Visual C++->C++ File i zmieniamy jego nazwę (razem z rozszerzeniem) na lab01.asm i klikamy Add



Rysunek 7.

5.2. Szablon programu dla VS

```
.386
.MODEL flat, STDCALL

stala equ 10

ExitProcess PROTO :DWORD ; brak importu biblioteki,
;jest ona dolaczona bezposrednio przez srodowisko

.data
zmienna1 db 10h

.code

main proc

;w celu poprawnego zakonczenia programu na platformie windows
        push 0
        call ExitProcess
main endp

END
```

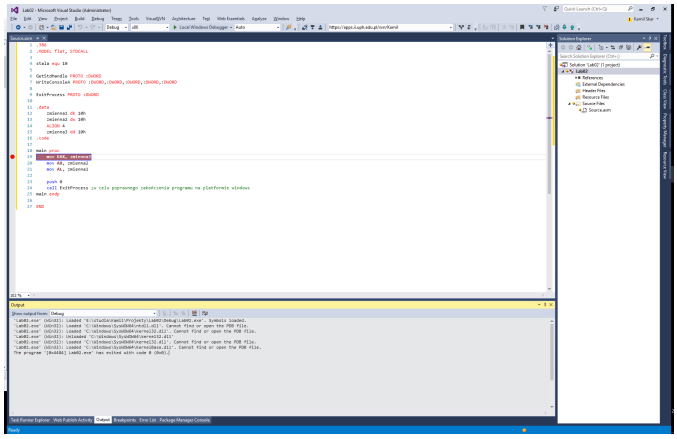
5.3. Operacje arytmetyczne

```
        ;5*10
        mov ebx, 10
        mov eax, 5
```

```
;mnozy rejestr EAX*EBX wynik zapisywany do EDX:EAX
;(32 starsze bity w EDX, 32 mlodsze w EAX)
mul ebx ;
```

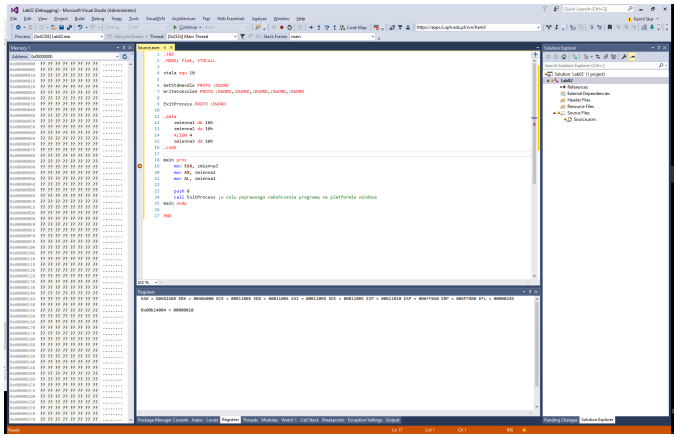
5.4. Debugowanie w VS

1. Debugowanie to proces pozwalający na prześledzenie wykonania kodu, linijka po linijce. W celu odnalezienia błędu możemy analizować stany rejestrów/pamięci po każdej wykonanej instrukcji. W tym celu musimy oznaczyć która instrukcja nas interesuje za pomocą punktu przerwania (ang. breakpoint). By go ustalić klikamy obok linni kodu (na szarym pasku), po czym pojawia się czerwona kropka.



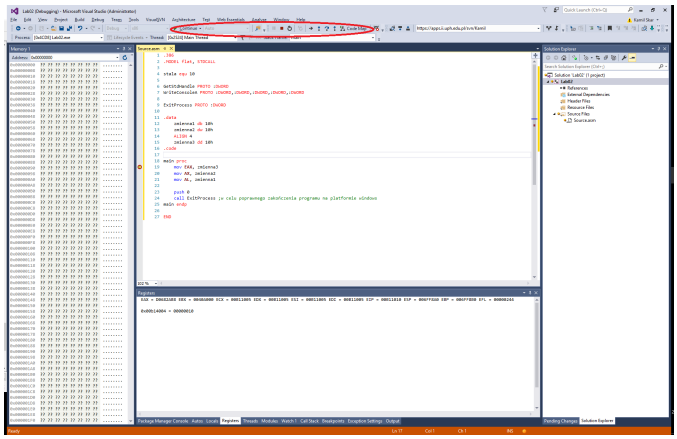
Rysunek 8. Ustanowienie punktu przerwania

2. Następnie uruchamiamy program w trybie debugowania (zielona strzała, Local Windows Debugger), spowoduje to zatrzymanie się wykonania programu w punkcie przerwania (przed wykonaniem instrukcji). Będąc w trybie debugowania, mamy dodatkowo możliwość konfiguracji wyglądu środowiska w celu podglądu aktualnego stanu rejestrów i pamięci. w celu pokazania okien z możliwymi oknami przechodzimy do pozycji w menu Debug->Windows i wybieramy interesujące nas pozycje.



Rysunek 9. Tryb debugowania

3. By wykonać rozkaz przed którym się zatrzymaliśmy, możemy posłużyć się przyciskiem F10, spowoduje to wykonanie jednej linni kodu i zatrzymanie się przed następną. By wznowić wykonanie programu możemy wcisnąć klawisz F5. Po wznowieniu program zatrzyma się w kolejnym ustanowionym punkcie przerwania.



Rysunek 10. Opcje dostępne w trybie debugowania

5.5. GetStdHandle - pobranie uchwytu do konsoli

W celu uzyskania uchwytu do aktualnie uruchomionego okna konsoli możemy wykorzystać procedure GetStdHandle:

```
HANDLE WINAPI GetStdHandle(
_In_ DWORD nStdHandle
);
```

nStdHandle może przyjmować następujące wartości:
1. -10 - uchwyt wejściowy, do odczytu z konsoli,
2. -11 - uchwyt wyjsciowy, do wypisywania znaków na konsolę,
3. -12 - uchwyt umożliwiający odczyt błędów.

Uchwyt zwracany jest do rejestru EAX po wykonaniu procedury.

Przykład wykorzystania:

```
push STD_OUTPUT_HANDLE ; stala -11
call GetStdHandle
mov outputHandle, EAX
```

5.6. WriteConsoleA - wypisywanie znaków na konsolę

Biblioteki Win32 udostępniają procedury umożliwiające odczyt znaków z interesującej nas konsoli. Można wykorzystać do tego procedurę WriteConsoleA o następującej sygnaturze:

```
BOOL WINAPI WriteConsoleA(
_In_             HANDLE  hConsoleOutput,
_In_       const VOID    *lpBuffer,
_In_             DWORD   nNumberOfCharsToWrite,
```

```
_Out_       LPDWORD lpNumberOfCharsWritten,
_Reserved_  LPVOID  lpReserved
);
```

Parametry:

1. hConsoleOutput - **uchwyt wyjściowy** do konsoli, pobierany za pomocą procedury GetStdHandle (5.5),
2. *lpBuffer - adress tablicy przechowującej znaki do wypisania,
3. nNumerOfCharsToWrite - liczba znaków które zostaną wypisane z poprzednio podanego adresu,
4. lpNumberofCharsWriten - adres zmiennej typu DWORD do której procedura zapisze ilość faktycznie wypisanych znaków,
5. lpReserved - wstawiamy null czyli 0

Przykład wykorzystania:

```
push 0
push OFFSET nOfCharsWritten
push nOfCharsToWrite
push OFFSET charsToWrite
push outputHandle
call WriteConsoleA
```

Segment danych:

```
nOfCharsWritten DWORD 0
charsToWrite    BYTE "Wprowadz argument A",0
nOfCharsToWrite DWORD $ - charsToWrite
```