|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| logowydzialu | | Instytut Informatyki Politechniki Śląskiej  Zespół Mikroinformatyki i Teorii Automatów Cyfrowych  **Laboratorium JA** | | logoii | |
| **Rok akademicki** | | **Rodzaj studiów\*: SSI/NSI/NSM** | **Numer ćwiczenia:** | **Grupa** | **Sekcja** |
| **2017/2018** | | **SSI** | **1** | **6** | **1** |
| **Data i godzina planowana ćwiczenia:**  dd/mm/rrrr - gg:mm | | **27/02/2018-11:45** | **Prowadzący**:  OA/AO | **OA** | |
| **Data i godzina wykonania ćwiczenia:**  dd/mm/rrrr - gg:mm | | **27/02/2018-11:45** |
| ***Sprawozdanie*** | | | | | |
| **Temat ćwiczenia:**  Tworzenie projektu asemblerowego dla środowiska Visual Studio | | | | | |
| **Skład sekcji:** | 1.Bartłomiej Krasoń | | | | |

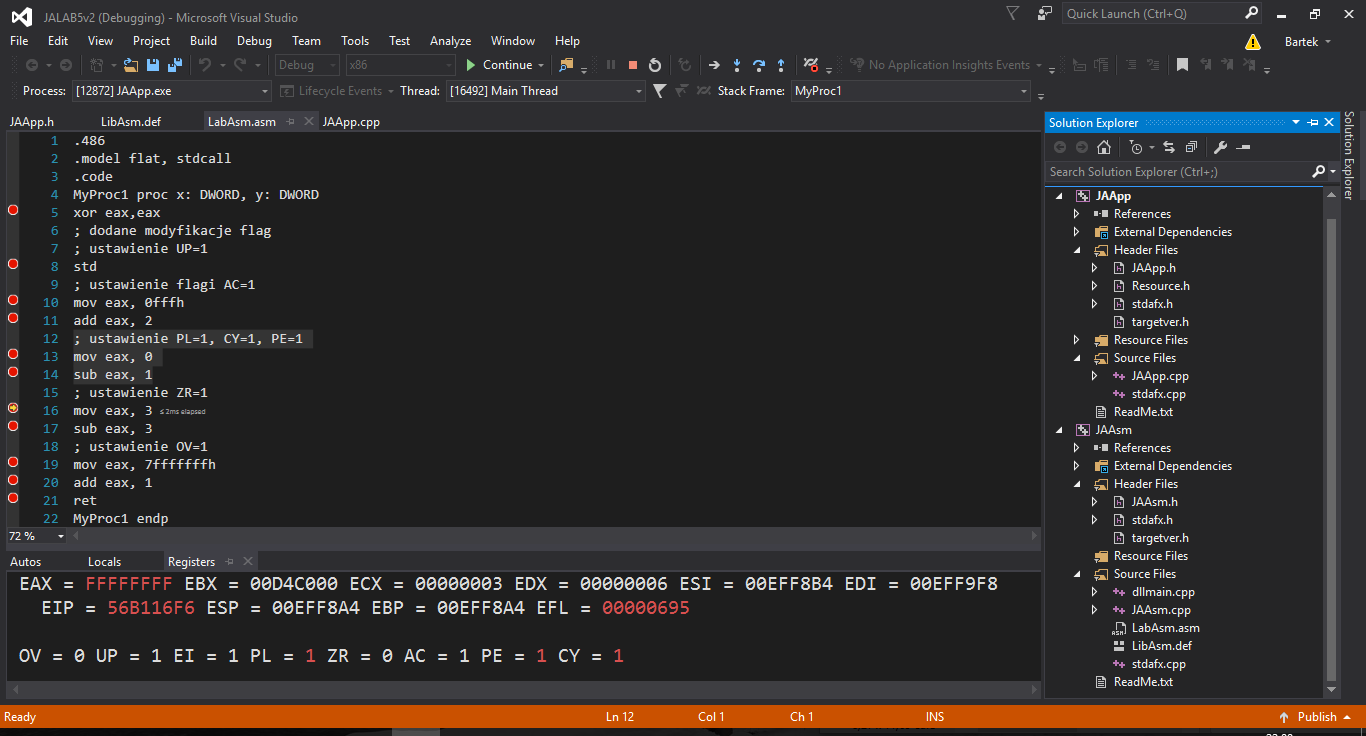
# Cel

Celem ćwiczenia jest poznanie możliwości środowiska Visual Studia 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.

*~Źródło – Instrukcja ćwiczenia LAB1*

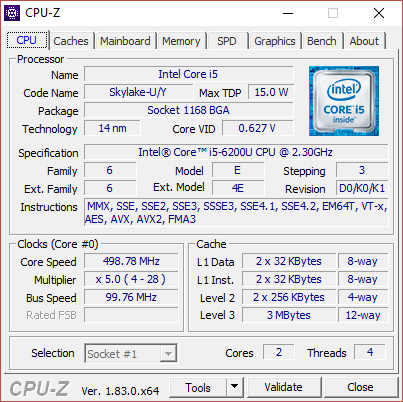
Dodatkowo: Sprawdzić programem CPU-Z jakie instrukcje obsługuje mój procesor. Wstawić screena.

# Rozwiązanie

Solucję JALAB5 zawierającą dwa projekty JAApp (aplikacja okienkowa w języku C++) oraz JAAsm (biblioteka dll w języku asemblera) utworzyłem według instrukcji ćwiczenia, jednakże musiałem nieco zmodyfikować wykonywane operację gdyż pracowałem w środowisku Visual Studio 2015. Ostatecznie udało się skompilować całą solucję i następnie przystąpić do debugowania, co obrazuje poniższy zrzut ekranu:

Ponadto na powyższym zrzucie widać zmodyfikowany kod procedury MyProc1, która pozwala krok po kroku zaobserwować zmiany na poszczególnych znacznikach rejestru flagi (które wyświetlane są w oknie Registers), tak oto flaga:

* UP zmienia wartość na 1 po zastosowaniu rozkazu STD oraz oznacza zmianę kierunku wykonywania kroku przez rejestry ESI i EDI,
* AC zmienia wartość na 1 gdy wystąpi przeniesienie/pożyczka między 3. i 4. bitem w ostatniej operacji,
* PL zmienia wartość na 1 gdy liczba stanie się ujemna
* CY zmienia wartość na 1 gdy w wyniku dodawania/odejmowania zostanie przekroczony możliwy zakres wartość, czyli wystąpi przeniesienie/pożyczka,
* PE zmienia wartość na 1 gdy liczba jedynek w najmłodszym bajcie stanie się parzysta,
* ZR zmienia wartość na 1 gdy wynik ostatniego działania wyniesie 0,
* OV zmienia wartość na 1 gdy nastąpi przeniesienie do bitu znaku,
* EI jest flagą systemową, co oznacza, że na jej wartość nie wpływa się z poziomu aplikacji, a systemu operacyjnego, dokładniej ustawiona na 1 pozwala na maskowane przerwania sprzętowe.

Screen programu CPU-Z, sprawdzający jakie instrukcje obsługuje mój procesor:

# Wnioski

Pierwszą cenną lekcją nabytą podczas wykonywania tego ćwiczenia było nauczenie się, jak tworzyć aplikacje z kodem mieszanym. Jest to o tyle wartościowe, że bez problemu możemy łączyć program napisany w języku C++ np. z poszczególnymi procedurami napisanymi w języku asemblera, co daje nam niekiedy istotną poprawę wydajności tworzonych przez nas aplikacji. Równie cenny jest dla mnie fakt, że przedstawione zostało mi jak w łatwy sposób można za pomocą narzędzia jakim jest Visual Studio obsługiwać kod napisany w języku asemblera. Debugowanie plików \*.asm jest równie przejrzyste i proste jak debugowanie projektów języka C++ czy C#, ponadto pozwala ono na bieżąco podglądać wartości wpisane w poszczególnych rejestrach, co jest niesamowitym ułatwieniem pracy programisty oraz przybliżeniem go do serca sprzętu, z którym pracuje.