



Instytut Informatyki Politechniki Śląskiej
Zespół Mikroinformatyki i Teorii Automatów
Cyfrowych

Laboratorium JA



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 assemblerowego dla środowiska Visual Studio				
Skład sekcji:	1.Bartłomiej Krasoń			

Cel

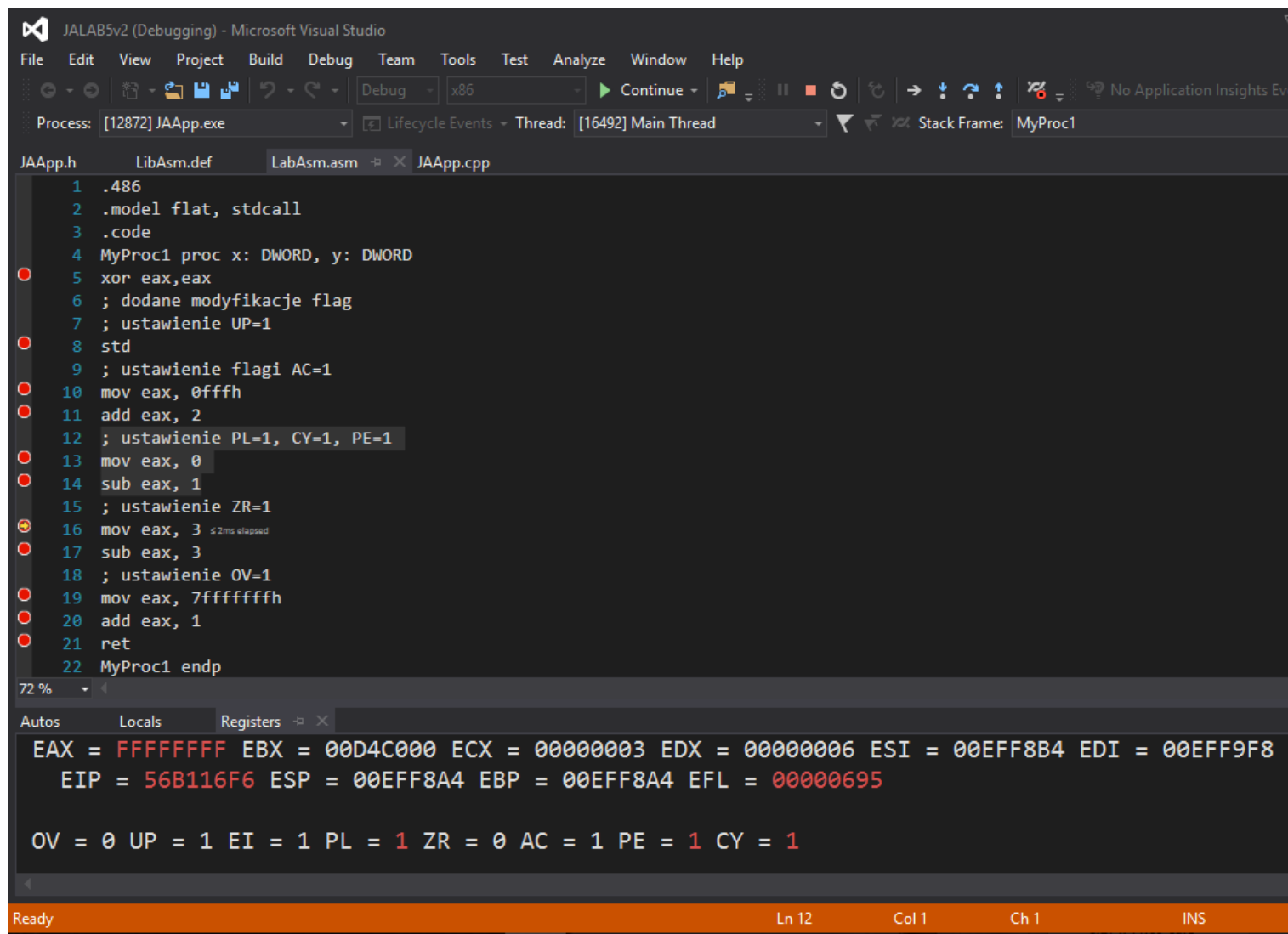
Celem ćwiczenia jest poznanie możliwości środowiska Visual Studio w zakresie tworzenia i uruchamiania aplikacji z kodem mieszanym w języku C++ oraz assemblerze. W założeniu aplikacja składa się z dwóch elementów aplikacji napisanej w j. C++ oraz biblioteki DLL napisanej w assemblerze dla środowiska Windows. Konstrukcja projektu zakłada możliwość wywoływania funkcji bibliotecznych napisanych w assemblerze z poziomu aplikacji oraz pokazuje prawidłową konfigurację środowiska umożliwiającą debugowanie kodu do poziomu assemblera, 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 assemblera) 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:



The screenshot shows the Visual Studio 2015 IDE in debug mode. The assembly code for `MyProc1` is displayed in the main editor window. The code includes instructions for setting various flags (UP, AC, PL, CY, PE, ZR, OV) and performing arithmetic operations on the `eax` register. The `Registers` window at the bottom shows the current state of the CPU registers and flags.

```
1 .486
2 .model flat, stdcall
3 .code
4 MyProc1 proc x: DWORD, y: DWORD
5 xor eax, eax
6 ; dodane modyfikacje flag
7 ; ustawienie UP=1
8 std
9 ; ustawienie flagi AC=1
10 mov eax, 0ffffh
11 add eax, 2
12 ; ustawienie PL=1, CY=1, PE=1
13 mov eax, 0
14 sub eax, 1
15 ; ustawienie ZR=1
16 mov eax, 3
17 sub eax, 3
18 ; ustawienie OV=1
19 mov eax, 7fffffffh
20 add eax, 1
21 ret
22 MyProc1 endp
```

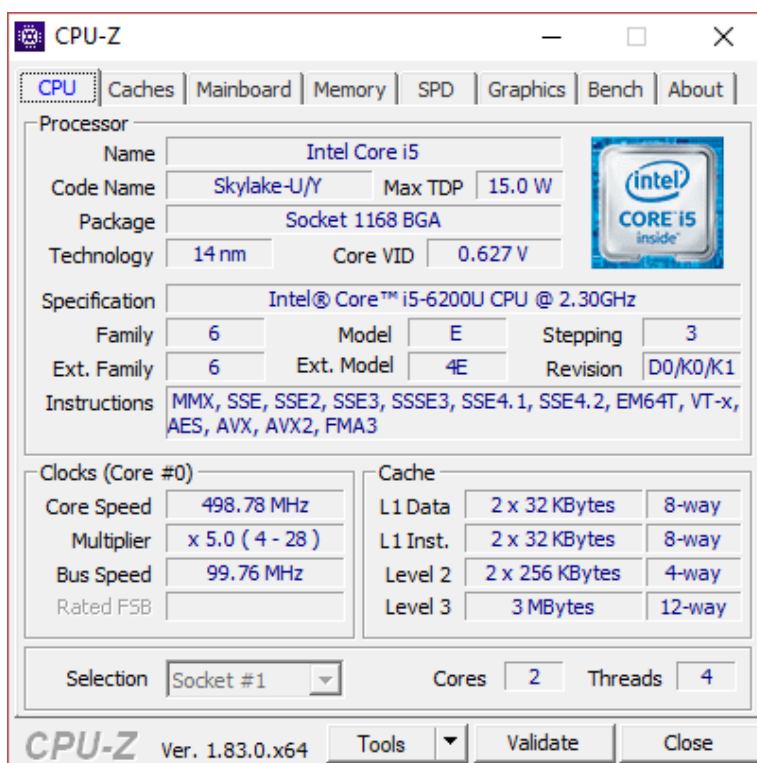
Registers window state:

EAX	=	FFFFFFFF	EBX	=	00D4C000	ECX	=	00000003	EDX	=	00000006	ESI	=	00EFF8B4	EDI	=	00EFF9F8
EIP	=	56B116F6	ESP	=	00EFF8A4	EBP	=	00EFF8A4	EFL	=	00000695						
OV = 0 UP = 1 EI = 1 PL = 1 ZR = 0 AC = 1 PE = 1 CY = 1																	

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 wartości, 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 assemblera, 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 assemblera. 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.